Network Architectures
and Services
NET 2011-07-1

SN
SS2011

Proceedings of the Seminar Sensor Nodes – Operation, Network and Application (SN)

Summer Semester 2011

Munich, Germany, 12./13.07.2011

Editors

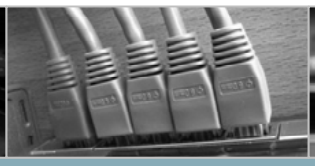
Georg Carle, Corinna Schmitt, Alexander Klein,
Uwe Baumgarten, Christoph Söllner

Organisation

Chair for Network Architectures and Services
Chair for Operating Systems and System Architectures
Department of Computer Science, Technische Universität München

Technische Universität München




Network Architectures
and Services
NET 2011-07-1

SN

SS 2011

Proceeding zum Seminar
Sensorknoten – Betrieb, Netze und Anwendungen (SN)

Sommersemester 2011

München, Germany, 12./13.07.2011

Editors: Georg Carle, Corinna Schmitt, Alexander Klein,
Uwe Baumgarten, Christoph Söllner

Organisiert durch den Lehrstuhl Netzarchitekturen und Netzdienste (I8)
und den Lehrstuhl für Betriebssysteme und Systemarchitektur (I13),
Fakultät für Informatik, Technische Universität München

Technische Universität München

SN SS 2011
Proceedings of the Seminar Sensor Nodes – Operation, Network and Application (SN)

Editors:

Georg Carle
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
Technische Universität München
D-85748 Garching b. München, Germany
E-mail: carle@net.in.tum.de
Internet: http://www.net.in.tum.de/~carle/

Corinna Schmitt
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
E-mail: schmitt@net.in.tum.de
Internet: http://www.net.in.tum.de/~schmitt/

Alexander Klein
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
E-mail: klein@net.in.tum.de
Internet: http://www.net.in.tum.de/~klein/

Uwe Baumgarten
Lehrstuhl Betriebssysteme und Systemarchitektur (I13)
Technische Universität München
D-85748 Garching b. München, Germany
E-mail: baumgaru@in.tum.de
Internet: http://www13.in.tum.de/

Christoph Söllner
Lehrstuhl Betriebssysteme und Systemarchitektur (I13)
E-mail: cs@tum.de
Internet: http://www13.in.tum.de/

Cataloging-in-Publication Data

SN SS 2011
Proceedings of the Seminar Sensor Nodes – Operation, Network and Application (SN)
Munich, Germany, 12./13.07.2011
Georg Carle, Corinna Schmitt, Alexander Klein, Uwe Baumgarten, Christoph Söllner
ISBN: 3-937201-23-8
DOI: 10.2313/NET-2011-07-1
ISSN: 1862-7803 (print)
ISSN: 1862-7811 (electronic)
Lehrstuhl Netzarchitekturen und Netzdienste (I8) NET 2011-07-1
Series Editor: Georg Carle, Technische Universität München, Germany

Vorwort

Wir präsentieren Ihnen hiermit das Proceeding zum Seminar „Sensorknoten – Betrieb, Netze und Anwendungen“ (SN), das im Sommersemester 2011 an der Fakultät Informatik der Technischen Universität München stattfand.
Im Seminar SN wurden Vorträge zu verschiedenen Themen im Forschungsbereich Sensorknoten vorgestellt. Die folgenden Themenbereiche wurden abgedeckt:

- Einführung in Sensornetze I – Unterschiedliche Hardware-Plattformen für Sensorknoten
- Energieverbrauch von Sensorkomponenten – Optimierung der Sensorkommunikation
- Einführung in Sensornetze – Abhängigkeiten zwischen Protokolldesign und verwendeter Hardware
- Betriebssysteme für Wireless Sensor Network Motes
- Event Detection in WSNs - Fahrzeugverfolgung
- MAC – Vergleich von Präambel-basierten Kanalzugriffsprotokollen
- Akustische Unterwasserkommunikation
- Collection Tree Protocol
- RPL: IPv6 Routing Protocol for Low Power and Lossy Networks
- IP-basierende Kommunikation in drahtlosen Sensornetzwerken
- TCP/IP Kommunikation in drahtlosen Sensornetzwerken
- “Random Key Distribution” Verfahren in WSNs – Vergleich verschiedener probabilistischer Ansätze
- Sicherheit in WSNs – Node Replication Angriffe

Wir hoffen, dass Sie den Beiträgen dieser Seminare wertvolle Anregungen entnehmen können. Falls Sie weiteres Interesse an unseren Arbeiten habe, so finden Sie weitere Informationen auf unserer Homepages http://www.net.in.tum.de und http://www13.in.tum.de.

München, Juli 2011


Georg Carle


Corinna Schmitt


Alexander Klein


Uwe Baumgarten


Christoph Söllner

Preface

We are very pleased to present you the interesting program of our main seminar on “Sensor nodes – Operating, Network and Application” (SN) which took place in the summer semester 2011.

In the seminar SN talks to different topics in current research tasks in the field of sensor nodes were presented. The seminar language was German, and also the seminar papers. The following topics are covered by this seminar:

- Introduction to WSNs I – Different Hardware Platforms for WSNs
- Energy Consumption by Sensor Components – Optimization of Sensor Communication
- Introduction to WSNs – Dependencies between Protocol Design and used Hardware
- Operating Systems for Sensor Nodes
- Event Detection in WSNs - Vehicle Tracking
- MAC – Comparison of preamble-based MAC protocols
- Acoustic Underwater Communication
- Collection Tree Protocol
- RPL: IPv6 Routing Protocol for Low Power and Lossy Networks
- IP-based Communication in Wireless Sensor Network
- TCP/IP communication in a WSN
- “Random Key Distribution” Protocols in WSNs – Comparison of different probabilistic Approaches
- Security in WSNs – Node Replication Attacks

We hope that you appreciate the contributions of these seminars. If you are interested in further information about our work, please visit our homepages http://www.net.in.tum.de and http://www13.in.tum.de.

Munich, July 2011

Seminarveranstalter

Lehrstuhlinhaber

Georg Carle, Uwe Baumgarten, Technische Universität München, Germany

Seminarleitung

Corinna Schmitt, Technische Universität München, Germany

Betreuer

Alexander Klein, *Technische Universität München, Wiss. Mitarbeiter I8*
Corinna Schmitt, *Technische Universität München, Wiss. Mitarbeiterin I8*
Christoph Söllner, *Technische Universität München, Wiss. Mitarbeiterin I13*

Kontakt:

{carle,schmitt,klein}@net.in.tum.de
baumgaru@in.tum.de, cs@tum.de

Seminarhomepage

http://www.net.in.tum.de/de/lehre/ss11/seminare/

Inhaltsverzeichnis

Session 1: Grundlagen und Anwendungen

Session 2: Kommunikation

Session 3: Sicherheit

Einführung in Sensornetze I
Unterschiedliche Hardware-Plattformen für Sensorknoten

Dominik Wetzel
Betreuer: Christoph Söllner
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: wetzeld@in.tum.de

KURZFASSUNG
Fortschritte in der Elektronik und drahtlosen Kommunikation haben die Entwicklung von kleinen, billigen Sensorknoten ermöglicht. Zum Aufbau sogenannter drahtlosen Sensornetzwerke aus diesen Knoten steht inzwischen eine Vielzahl unterschiedlicher Hardware-Plattformen zur Verfügung. Anhand der Waldbranderkennung als Anwendungsbeispiel werden die Plattformen MICA2/z, TelosB, ScatterWeb und BTnode miteinander verglichen und deren Eignung herausgearbeitet. Wichtige Kriterien stellten dabei die Verfügbarkeit der benötigten Sensoren, sowie die eingesetzten Prozessoren und Funkchips dar. Mit allen Plattformen lässt sich das geforderte Netzwerk aufbauen, jedoch erfüllt von den vorgestellten Plattformen TelosB die aufgestellten Kriterien am besten.

Schlüsselworte
Drahtloses Sensornetz, Vergleich, MICA2, MICAz, TelosB, ScatterWeb, BTnode

1. EINLEITUNG
Fortschritte in der Elektronik und drahtlosen Kommunikation haben die Entwicklung von kleinen, billigen Sensorknoten ermöglicht. Die Charakteristika sind ihr geringer Energieverbrauch, ihre vielfältige Einsetzbarkeit, und die Fähigkeit, drahtlos über kurze Distanzen lose miteinander kommunizieren zu können [3]. Ein Sensorknoten (im englischen auch „mote“ genannt) besteht im Allgemeinen aus Komponenten zur Datenerfassung, Datenverarbeitung und zur Kommunikation mit anderen Sensorknoten.

Als Sensornetz (engl. „Wireless Sensor Network,“ WSN) bezeichnet man den Zusammenschluss aus häufig vielen Sensorknoten, die dicht im oder in der Nähe des mit den Sensoren beobachteten Phänomens platziert sind. Die Position der einzelnen Sensorknoten muss nicht unbedingt im Voraus bekannt sein, sie können auch zufällig verteilt werden, z.B. durch Abwurf aus einem Flugzeug bei schwer zugänglichem Gelände [3]. Dies bedeutet, dass die verwendeten Netzwerkprotokolle und Algorithmen sich selbst organisieren können müssen. Typischerweise bilden die Sensorknoten ein engmaschiges multi-hop Netzwerk, durch den Aufbau von Verbindungen zu ihren Nachbarn [6]. Ein weiteres besonderes Merkmal eines WSNs ist die Zusammenarbeit der einzelnen Sensorknoten. Durch ihren Prozessor können sie eine lokale Vorverarbeitung durchführen und müssen nicht die Rohdaten übertragen [3]. Die Anwendungsgebiete umfassen unter anderem die Bereiche Militär, Umweltüberwachung, Gesundheitswesen, Lagerverwaltung oder auch Heimautomation [23, S. 50ff].

Mittlerweile existieren verschiedene Hardware-Plattformen, die den Aufbau eines Sensornetzes erleichtern. Ziel dieses Artikels ist es, einen Überblick über die derzeit verfügbaren Produkte zu geben. Dazu wird im Folgenden zunächst der generelle Hardwareaufbau eines Sensorknotens erörtert. Anschließend werden verschiedene Plattformen vorgestellt und miteinander verglichen.

2. KOMPONENTEN EINES SENSORKNOTENS
Obwohl sich die Hardware-Plattformen in vielen Details unterscheiden, ist der grundlegende Aufbau aller Sensorknoten gleich. Dieser ist in Abb. 1 dargestellt.

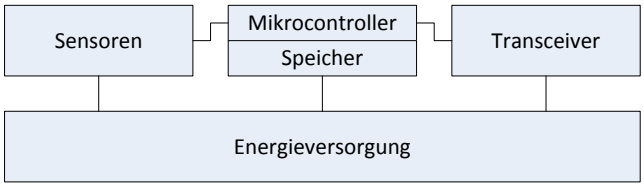


Abbildung 1: Schematischer Aufbau eines Sensorknotens. Quelle: [11, Fig. 2.1]

Mikrocontroller Der Mikrocontroller steuert den Sensorknoten. Er erfasst die Sensordaten, führt die Vorverarbeitung durch und koordiniert die Kommunikation mit den benachbarten Knoten. Der Mikrocontroller ist in der Regel frei programmierbar [11, S. 18ff]. Immer öfter kommen allerdings speziell auf Sensorknoten zugeschnittene Betriebssysteme wie zum Beispiel tinyOS [13] zum Einsatz. Durch diese kann die Software unabhängig von der verwendeten Hardware werden [23, S. 80]. Ein weiterer Vorteil des Einsatzes spezieller Betriebssysteme ist, dass durch deren Verwendung mitunter auch ganze Sensornetze simuliert werden können, was deren Aufbau und Planung erheblich vereinfacht. Für tinyOS steht die Simulationssoftware TOSSIM zur Verfügung [12].

Speicher Im Speicher werden Programme und Zwischenergebnisse abgelegt. Neben dem im Mikrocontroller vorhandenen RAM kommt zusätzlich meist nichtflüchtiger

doi: 10.2313/NET-2011-07-1_01

Speicher in Form von EEPROM oder Flash zum Einsatz. Dieser Speicher kann wiederum im Mikrocontroller integriert, oder aber extern angebunden sein [11, S. 21].

Sensoren Die Sensoren bilden die Schnittstelle zwischen der Außenwelt und dem WSN. Sensoren bestehen meist aus einem analogen Bauteil, welches die Messgröße erfasst (z.B. Thermistor für die Messung von Temperatur), und einem Analog-Digital-Wandler [3]. Letzterer kann ebenfalls im Microcontroller enthalten sein (s.u.).

Transceiver Der Transceiver ist für die eigentliche Datenübertragung zwischen einzelnen Sensorknoten verantwortlich. Er kann sowohl Daten empfangen, als auch senden. Obwohl die meisten WSNs Funk als Übertragungsmedium einsetzen, wird in [10] auch die Möglichkeit der optischen Kommunikation in Betracht gezogen[11, S. 28]. Überwiegend funken die WSNs in den Frequenzen zwischen 433 MHz und 2,4 GHz [11, S. 21].

Energieversorgung Für die unabhängig von einer festen Stromquelle agierenden Sensorknoten ist die Energieversorgung eine entscheidende Komponente [11, S. 32]. In der Regel erfolgt sie über Batterien [6]. Bei relativ geringem Energiebedarf eines Sensorknotens werden oft nicht wiederaufladbare Batterien gewählt, da diese gegenüber den Wiederaufladbaren eine höhere Energiedichte, geringere Selbstentladung und geringe Anschaffungskosten aufweisen. Eine Batterielebensdauer von mindestens einem Jahr ist erstrebenswert [26]. Aus diesem Grund ist es notwendig den Energieverbrauch des Sensorknotens streng zu kontrollieren.

Die größten Energieverbraucher sind vor allem der Mikrocontroller und der Transceiver [11, S. 36]. Im Rahmen dieses Artikels wird daher lediglich der Energieverbrauch dieser beiden Komponenten herangezogen, um den Stromverbrauch verschiedener Plattformen zu vergleichen.

Eine gezielte Regelung des Energieverbauchs eines Sensorknotens kann dadurch erfolgen, dass die einzelnen Komponenten in verschiedene Schlafzustände versetzt, oder mit geringerer Frequenz oder Betriebsspannung betrieben werden [11, S. 48]. Zusätzlich kann die Lebensdauer eines Sensorknoten dadurch verlängert werden, dass die Batterie z.B. über Photovoltaic wiederaufgeladen wird. Solche Techniken werden als Energy Scavenging oder Energy Harvesting bezeichnet [11, S. 34].

3. HARDWARE-PLATTFORMEN

Um ein Sensornetz aufzubauen stehen unterschiedliche Hardware-Plattformen zur Verfügung. Wie im Verlauf dieses Abschnitts herausgearbeitet wird, variieren diese unter Anderem in Faktoren wie:

- Mikrocontroller
- Funktechnik
- Anbindung der Sensoren
- Verfügbarkeit von Sensoren seitens des Plattform-Entwicklers
- Energieverbrauch
- Verfügbare Betriebssysteme

Unterschiede in diesen Eigenschaften bewirken, dass manche Hardware-Plattformen für einen bestimmten Anwendungsbereich eher geeignet sind als andere. Im Folgenden werden einige beispielhafte Plattformen vorgestellt und deren Eignung für das Einsatzgebiet der Waldbranderkennung untersucht. In [5] wurde ein solches System mittels eines Sensornetzes entwickelt. Die von den Sensoren erfassten Parameter waren:

- Helligkeit
- Luftfeuchtigkeit
- Temperatur

Diese Parameter werden deshalb hier zum Vergleich herangezogen. Zusätzlich wird angenommen, dass die Sensorknoten an bestimmten, vorab ausgesuchten Orten angebracht werden. Dadurch ist ihre Position bekannt, und es werden keine diesbezüglichen Sensoren wie zum Beispiel GPS benötigt.

Ähnlich wie in [5] sollen die Sensorknoten unterschiedliche Gefahrenstufen kennen, und bei Erreichen derselben Warnmitteilungen versenden. Zum Beispiel könnte zu einem Zeitpunkt eine hohe Temperatur in Kombination mit einer sehr niedrigen Luftfeuchtigkeit gemessen werden. Sollte daraufhin über den Helligkeitssensor eine Verdunklung entdeckt werden, die möglicherweise vom Rauch eines sich entwickelnden Brandes stammt, wird eine höhere Gefährdungsstufe für das vom Sensorknoten überwachte Gebiet erreicht und kommuniziert.

3.1 MICA2 / MICAz

Die MICA-Plattformen sind aus Forschungsprojekten der University of California Berkeley („UC Berkeley“) hervorgegangen und wurden urspünglich von Crossbow Technology vermarktet [11, S. 54]. Anfang 2010 wurden die Produktlinien von Memsic übernommen und werden seitdem von dieser Firma vertrieben [14]. Bis auf den eingesetzten Funkchip sind die Plattformen MICA2 und MICAz identisch, weshalb sie hier zusammen betrachtet werden. Ein MICA2-Sensorknoten ist in Abb. 2 dargestellt.

Eigenschaften der Plattformen sind:

- Prozessor: ATmega128L
- Funkchip MICA2: ChipCon CC1000
- Funkchip MICAz: ChipCon CC2420
- 51-Pin Anschlussbuchse für Erweiterungskarten (überträgt analoge Eingänge, digitale I/O-Pins, SPI, I^2C und UART)
- Batteriehalter für zwei AA-Batterien
- Abmessungen ohne Batteriehalter: $58 \times 32 \times 7$mm [16, 15, 6]

doi: 10.2313/NET-2011-07-1_01

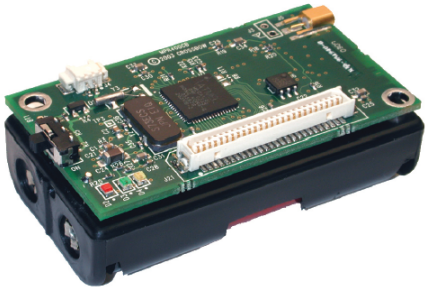


Abbildung 2: Ein MICA2-Sensorknoten. Quelle: [15]

Der verwendete Prozessor ATmega128L [4] hat die folgenden Merkmale:

- 8-bit RISC-Architektur
- bis zu 8MHz Prozessortakt
- 128 KB Flash-Speicher
- 4 KB RAM
- 4 KB EEPROM
- 10 bit Analog-Digital-Wandler (8 Kanäle)
- 53 digitale I/O-Pins
- UART, SPI und I^2C-Schnittstellen
- Energieverbrauch: 8mA im Betrieb und $< 15\,\mu A$ im Schlafmodus

Ein Vergleich der beiden Funkchips ist in Tabelle 1 dargestellt.

Tabelle 1: Vergleich der Funkchips der MICA2 und MICAz-Plattform. Quelle: [15, 16]

	CC 1000	**CC 2420**
Allgemeine Eigenschaften		
Frequenzbereiche	868 / 916 MHz	2,4 GHz
Datenrate	38,4 kbps	250 kbps
Reichweite (außen)	ca. 150 m	75-100 m
Energieverbrauch		
senden max.	27 mA	17,4 mA
empfangen	10 mA	19,7 mA
schlafen	$< 1\,\mu A$	$1\,\mu A$

Auf den MICA-Boards sind an sich keine Sensoren vorhanden. Memsic bietet jedoch einige Sensorboards an, die über den 51-Pin Anschluss mit dem Basisboard verbunden werden können. Das Sensorboard MTS400CC [17] enthält je einen Temperatur-, Feuchtigkeits-, Luftdruck-, Helligkeits- und Beschleunigungssensor und deckt damit die geforderten Parameter ab.

Auf den MICA-Boards kommt üblicherweise das Betriebssystem TinyOS zum Einsatz [11, S. 54].

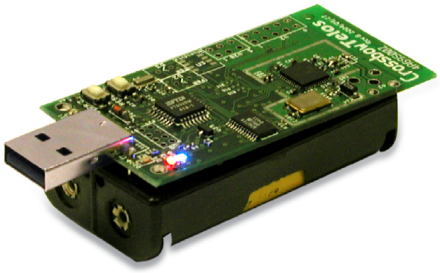


Abbildung 3: Ein TelosB-Sensorknoten. Quelle: [18]

3.2 TelosB

TelosB (Abb. 3) ist eine Hardware-Plattform, die als Nachfolger der MICA-Plattform angesehen werden kann. Sie wurde ebenfalls an der UC Berkeley entwickelt und wird derzeit von Memsic (ehemals Crossbow) vertrieben.

TelosB hat die folgenden Merkmale:

- Prozessor: Texas Instruments (TI) MSP430F1611,
- Funkchip: ChipCon CC2420
- Antenne auf der Platine integriert
- 1 MB externer Flash-Speicher für Messwerte / Logging
- Batteriehalter für zwei AA-Batterien
- Abmessungen ohne Batteriehalter: $65 \times 31 \times 6$mm

Der verwendete Prozessor TI MSP430F1611 [25] hat diese Eigenschaften:

- 16-bit RISC-Architektur
- 10 KB RAM
- 48 KB Flash-Speicher für Programmcode
- 12 Bit Analog-Digital-Wandler (8 Kanäle)
- 48 digitale I/O-Pins
- UART, SPI und I^2C-Schnittstellen
- Weniger als $6\,\mu s$ zum Aufwachen aus Sleep-Modus benötigt
- Energieverbrauch: 1,8 mA im Betrieb und $5{,}1\,\mu A$ im Schlafmodus

Es gibt zwei Ausführungen der TelosB motes: eine mit und eine ohne Sensoren. Die Variante TPR2420CA enthält einen Helligkeits-, einen Feuchtigkeits- und einen Temperatursensor. Zusätzliche Sensoren könnten prinzipiell über einen 6- und 10-pin Anschluss angebunden werden, allerdings bietet Memsic keine diesbezüglichen Produkte an. Durch die verfügbaren Sensoren ist TelosB für das Anwendungsbeispiel geeignet.

doi: 10.2313/NET-2011-07-1_01

Die TelosB-Plattform ist quelloffen [18]. Dadurch gibt es weitere Hersteller, die darauf basierende Produkte anbieten. Advantic [1] kann als Beispiel dienen. Das Produkt MTM-CM5000-MSP entspricht weitestgehend dem TelosB-Referenzdesign. Variationen des Originaldesigns verzichten aus Platzgründen auf den USB-A-Stecker und besitzen eine spezielle Schnittstelle zum Anschluss von Sensorboards. Davon hat Advantic eine Vielzahl im Angebot (vgl. [2]).

Auch auf den TelosB-Sensorknoten wird das Betriebssystem TinyOS eingesetzt [18].

3.3 ScatterWeb

ScatterWeb ist eine Plattform, die an der Freien Universität Berlin entwickelt wurde [20]. Eine aktives Produkt dieser Plattform ist das „Modular Sensor Board MSB-430 H“ [19]. Es ist in Abb. 4 dargestellt.

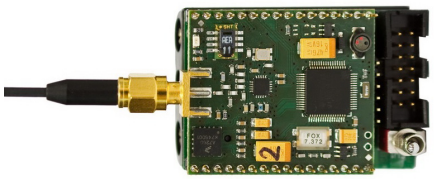


Abbildung 4: Ein ScatterWeb MSB430-H-Sensorknoten. Quelle: [19]

Die Charakteristika des MSB-430 H sind:

- Prozessor: TI MSP430F1612
- Funkchip: ChipCon CC1100
- Externer Flash-Speicher per SD-Karte möglich
- Batteriehalter für drei AA-Batterien [19]

Der Prozessor MSP430F1612 unterscheidet sich von der Variante F1611 der TelosB-Plattform nur geringfügig. Die Größe des Arbeitsspeichers beträgt 5 KB anstelle von 10 KB, während 48 KB anstelle von 55 KB Flash vorhanden sind.

Sensoren für Feuchtigkeit und Temperatur, sowie ein Accelerometer sind auf dem Basisboard vorhanden. Für das oben beschriebene Anwendungsbeispiel der Waldbranderkennung fehlt allerdings der Helligkeitssensor. Zum Anschluss weiterer Sensoren verfolgt die ScatterWeb-Plattform allerdings ein interessantes Konzept: An den Randbereichen des Boards sind Stiftleisten angebracht, auf die die Sensorboards gesteckt werden können. Diese haben an ihrer Oberseite ebenfalls Stiftleisten mit denselben Kontakten. Auf diese Weise können mehrere (unterschiedliche) Sensorboards übereinander gestapelt werden (vgl. [19]).

Eine weitere Besonderheit der Plattform ist der SD-Karten-Slot. Während auf der TelosB-Plattform beispielsweise nur 1 MB an externem Flash Speicher zur Verfügung stehen, sind hier ohne Weiteres mehrere Gigabyte möglich. Dadurch kann bei Bedarf eine viel größere Menge an Daten vor der Übertragung zwischengespeichert werden.

Die ScatterWeb-Knoten benutzen die auf sie spezialisierte Software ScatterWeb^2 [21]. Für eine ältere, nicht mehr verfügbare Produktlinie, in der zum Beispiel das „Embedded Sensor Board“ enthalten ist, wird offiziell zusätzlich das Betriebssystem tinyOS unterstützt. Die neueren Modelle der MSB-Serie sind davon allerdings explizit ausgenommen [22].

3.4 BTnodes

Die Plattform BTnodes wird an der ETH Zürich entwickelt. Wie der Name bereits andeutet, diente als Kommunikationsmedium ursprünglich nur Bluetooth. In der aktuellen Revision 3 ist zusätzlich auch der Funkchip der MICA2 motes zu finden [9]. Eine BTnode ist in Abb. 5 dargestellt.

Abbildung 5: Ein BTnode. Quelle: [9]

Die BTnodes haben die folgenden Merkmale.

- Prozessor: Atmel ATmega 128L
- Funkchip: Chipcon CC1000 und Zeevo ZV4002 (Bluetooth)
- Zwei Anschlussbuchsen für Erweiterungskarten (übertragen analoge Eingänge, digitale I/O-Pins, SPI, I^{2}C, UART, Timer- und LED-Signale)
- Batteriehalter für zwei AA-Batterien
- Abmessungen: $58,15 \times 33$mm [8].

Der Bluetooth-Chip, auf dessen technische Merkmale hier nicht genauer eingegangen werden soll, wurde integriert, um die BTnodes als Plattform zur Forschung an Mobile Ad Hoc Networks (MANET) nutzen zu können [8]. MANETs sind eng mit drahtlosen Sensornetzwerken verwandt. Sie unterstützen allerdings weniger Knoten pro Netzwerk und verwenden eine höhere Übertragungsleistung, wodurch allerdings auch höhrere Reichweiten erzielt werden können [3]. Der Energieverbrauch einer BTnode ist in Tabelle 2 aufgelistet.

Tabelle 2: Energieverbrauch der BTnode. Quelle: [7]

CPU im Sleep-Modus, Funk aus	3 mA
CPU an, Funk aus	12 mA
CPU an, Bluetooth an	28 mA
CPU an, CC1000 an	25 mA
Max. Energieverbrauch mit Bluethooth	60 mA
Max. Energieverbrauch mit CC1000	31 mA
CPU an, CC1000 an, Bluetooth an	41 mA

doi: 10.2313/NET-2011-07-1_01

Für die BTnodes gibt es ein eigenes Betriebssystem BTnut, für das über eine C-API eigene Anwendungen geschrieben werden können. Zusätzlich ist die BTnode kompatibel mit tinyOS 2.x [8].

Die BTnode besitzt auf dem Basisboard keine Sensoren. Zudem bietet das BTnode Projekt keine fertigen Sensorboards an. Die Sensoren für den hier betrachteten Anwendungsfall müssten daher auf einer selbst zu entwickelnden Erweiterungskarte sitzen. Neben der Hardwareentwicklung wären auch entsprechende Software-Treiber zu programmieren.

4. VERGLEICH UND BEWERTUNG

Die bisher vorgestellten Plattformen erfüllen die Kriterien des Szenarios der Waldbranderkennung unterschiedlich gut. Um eine Sensorplattform umfassend bewerten zu können, müssen mehrere Gesichtspunkte in Betracht gezogen werden. Nicht alle davon lassen sich in dieser theoretischen Abhandlung sinnvoll beurteilen. So steht es beispielsweise außer Frage, dass die 10 KB Arbeitsspeicher der vorgestellten TI-Prozessoren „besser“ sind als die 4 KB der Atmel-Pendants. Allerdings kann dieser Unterschied für die praktische Anwendung vollkommen unerheblich sein. Genau so verhält es sich mit der Bitbreite des Mikrocontrollers. Ein 8-Bit Mikrocontroller benötigt beispielsweise mehr Rechenoperationen und somit mehr Energie um Berechnungen mit 16-Bit breiten Daten durchzuführen, als ein 16-Bit Mikrocontroller. Ob dies von Belang ist kann nur mit genauer Kenntnis der zur Verarbeitung der Messdaten eingesetzten Algorithmen festgestellt werden.

Bei der Verfügbarkeit der geforderten Sensoren ist die Situation eindeutiger. Wichtig ist, dass keine der hier vorgestellten Plattformen gänzlich für die Waldbranderkennung ungeeignet wäre. Die Plattformen unterscheiden sich lediglich in dem selbst zu leistenden Entwicklungsaufwand. Die TelosB-Plattform bietet eine Variante, die alle Sensoren aufweist. Das Sensornetz aus dem vorgestellten Szenario könnte damit folglich am leichtesten umgesetzt werden. Im Gegensatz dazu befinden sich auf der BTnode keine Sensoren, und es sind des Weiteren keine Sensor-Erweiterungskarten verfügbar. Um dennoch die BTnode einsetzen zu können, ist, wie bereits erwähnt, Hard- und Softwareentwicklung nötig, was einen zusätzlichen Zeit- und Kostenaufwand nach sich zieht. Auch bei der ScatterWeb-Plattform ist aufgrund des nicht verfügbaren Helligkeitssensors Hardware-Entwicklungsarbeit zu leisten. Dagegen sind für die MICA2- und MICAz-Plattformen Sensorboards erhältlich, durch die die Plattformen alle Anforderungskriterien erfüllen.

Ein interessanter Gesichtspunkt ist auch die Art und Weise, wie zusätzliche Sensorik an die einzelnen Plattformen angeschlossen werden kann. Besonders praktisch scheint hier das Konzept von ScatterWeb. Der modulare Ansatz ermöglicht es, dass mehrere Sensorboards gleichzeitig an den Knoten gekoppelt werden können. Bei den anderen hier vorgestellten Plattformen sehen die Hersteller, sofern sie überhaupt zusätzliche Sensorboards anbieten, nur jeweils eines vor. Allerdings muss bei ScatterWeb der Anwender selbst sicherstellen, dass keine Anschlüsse der Erweiterungsschnittstelle von mehr als einem Modul genutzt werden. Sofern dies beachtet wird, kann allerdings die einmal entwickelte Hardware mitsamt der entsprechenden Treiber sehr einfach für andere Anwendungen wiederverwendet werden.

Da der Energieverbrauch einer Plattform ein entscheidendes Kriterium für die Lebensdauer eines Sensornetzwerks ist, sollte er beim Vergleich verschiedener Plattformen besonders berücksichtigt werden. Maßgeblich für den Energieverbrauch eines Sensorknotens ist die Kombination aus Prozessor und Funkchip. In den vorgestellten Plattformen konnten zwei verschiedene Prozessoren und zwei verschiedene Funkchips in unterschiedlichen Zusammenstellungen vorgefunden werden. Der bei TelosB und ScatterWeb eingesetzte TI MSP430-Prozessor ist erheblich sparsamer als der Atmel ATmega128L der restlichen Plattformen. Bei den Funkchips ist der maximale Stromverbrauch beim Senden des ChipCon CC 1100 geringer als des CC 2420 (siehe Tabelle 1). Umgekehrt ist die Situation beim Empfangen. Allerdings ist die Datenrate des CC 2420 um einiges höher als die des CC 1100, wodurch er dieselbe Datenmenge in viel kürzerer Zeit übertragen kann. Unter Beachtung dieser Kriterien kann sich TelosB von den vorgestellten Plattformen wiederum am Besten behaupten. Sie vereint einen sparsamen Prozessor mit dem Funkchip, der den geringeren maximalen Stromverbrauch besitzt. MICAz verwendet den sparsameren Prozessor und schlechteren Funkchip, bei ScatterWeb verhält es sich genau umgekehrt. MICA2 und BTnode setzen die rein rechnerisch schlechteste Kombination ein. Da die Funkchips im Vergleich zu den Prozessoren sehr viel Strom benötigen, spielt allerdings das Kommunikationsverhalten eines Sensornetzwerks eine extrem wichtige Rolle. Sofern keine Daten gesendet oder empfangen werden, können die Transceiver im Schlafzustand gehalten werden, in dem sie fast keinen Strom verbrauchen. Bei einer geschickten Kommunikationsstrategie könnte dadurch der rechnerischer Nachteil einer Plattform leicht ausgeglichen werden.

In der Praxis wird man ein System zur Waldbranderkennung wahrscheinlich nur an Orten realisieren, die diesbezüglich besonders gefährdet sind. Zweifellos sind dies Orte, die besonders warm und trocken sind. Wie in [24] erwähnt, würde sich daher Energy Harvesting mittels Solarenergie oder des Temperaturgradienten zwischen Tag und Nacht anbieten. Aus diesem Grund könnte der Aspekt des Energieverbrauchs eines Sensorknotens etwas in den Hintergrund rücken.

Zusammenfassend steht die TelosB-Plattorm von den hier vorgestellten Plattformen am besten da. Sie verwendet den stromsparenden Prozessor, mit dem größeren Arbeitsspeicher und setzt auf die Funktechnologie mit den höheren Übertragunsraten. Die Tatsache, dass verglichen mit dem SD-Karten-Anschluss von ScatterWeb „nur“ 1 MB an externem Flash-Speicher zur Verfügung steht, ist kein Nachteil. Im Hinblick auf das Datenvolumen der Messwerte sollte sich dies nicht negativ auswirken.

Die BTnodes scheinen aufgrund des vergleichsweise hohen noch zu leistenden Entwicklungsaufwandes für die Sensoren und ihres Energiebedarfes auf den ersten Blick eher ungeeignet für das hier betrachtetete Anwendungsgebiet der Waldbranderkennung. Allerdings könnten sie sich gerade wegen ihres Bluetooth-Chips doch als nützlich erweisen. Denn ein bisher noch nicht betrachteter Punkt ist, dass die Daten, die das drahtlose Sensornetz erhebt und überträgt, an einer Stelle gesammelt und / oder ausgewertet werden müssen.

doi: 10.2313/NET-2011-07-1_01

Von Bernardo et al. [5] ist angedacht, dass dies ein Wächter oder Feuerwehrmann übernehmen könnte, der z.B. mit einem PDA das waldbrandgefährdete Gebiet patrouilliert. Im Zeitalter von Smartphones, die heutzutage in den meisten Fällen ebenfalls mit Bluetooth ausgestattet sind, ist es naheliegend diesen Übertragungsweg zu nutzen. Dank des CC 1000-Chips der BTnode muss das restliche Sensornetz nicht zwingend aus BTnodes bestehen. Es kann auf eine Hardware-Plattform zurückgegriffen werden, die sich besser für die Sensorik eignet. In diesem Fall wird einmal mehr der Vorteil von Betriebssystemen wie tinyOS deutlich: setzen beide Plattformen dasselbe Betriebssystem ein, sollte sich zum Beispiel die Protokollschicht für die Kommunikation innerhalb des WSN auf beiden Plattformen unverändert nutzen lassen.

5. ZUSAMMENFASSUNG

Die untersuchten Hardware-Plattformen eignen sich grundsätzlich alle zur Anwendung im hier betrachteten Bereich der Waldbranderkennung. Unterschiede existieren in der Verfügbarkeit der geforderten Sensoren, sowie im Energieverbrauch der einzelnen Plattformen. Auch die Möglichkeit fehlende Sensoren nachzurüsten ist unterschiedlich. TelosB konnte sich von den anderen Hardware-Plattformen positiv absetzen. Dieses Ergebnis gilt allerdings nur für das gegebene Beispiel und kann nicht pauschalisiert werden. Dennoch wird klar, dass mit Hilfe der vorgestellten Plattformen viele unterschiedliche Anwendungsgebiete für drahtlose Sensornetze abgedeckt werden können.

6. LITERATUR

[1] Advantic systemas y servicios S.L. homepage. `http://advanticsys.com/`.

[2] Advantic systemas y servicios S.L. Product catalog 2010. `http://advanticsys.com/files/catalog.pdf`.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102 – 114, aug 2002.

[4] Atmel Corporation. ATmega128L datasheet. `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`.

[5] L. Bernardo, R. Oliveira, R. Tiago, and P. Pinto. A fire monitoring application for scattered wireless sensor networks - a peer-to-peer cross-layering approach. In *WINSYS*, pages 189–196, 2007.

[6] R. Bischoff, J. Meyer, and G. Feltrin. Wireless sensor network platforms. In *Encyclopedia of Structural Health Monitoring*, chapter 69. John Wiley & Sons, Ltd, 2009.

[7] BTnode Project. BTnode rev3 hardware reference. `http://www.btnode.ethz.ch/Documentation/BTnodeRev3HardwareReference`.

[8] BTnode Project. BTnode rev3 product brief. `http://www.btnode.ethz.ch/pub/files/btnode_rev3.24_productbrief.pdf`. Dokumentenrevision: rev3.24 2006/03/21.

[9] BTnode Project. Homepage. `http://www.btnode.ethz.ch`.

[10] J. M. Kahn, R. H. Katz, R. H. K. (acm Fellow, and K. S. J. Pister. Next century challenges: Mobile networking for „smart dust“, 1999.

[11] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, 2007.

[12] P. Levis, N. Lee, M. Welsh, and 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, pages 126–137, New York, NY, USA, 2003. ACM.

[13] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In *in Ambient Intelligence*. Springer Verlag, 2004.

[14] Memsic Inc. Memsic completes crossbow technology acquisition. `http://investor.memsic.com/releasedetail.cfm?ReleaseID=439436`. Pressemitteilung vom 22. Januar 2010.

[15] Memsic Inc. MICA2 datasheet. `http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=147%3Amica2`. Dokumentenrevision: 6020-0042-09 Rev A.

[16] Memsic Inc. MICAz datasheet. `http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=148%3Amicaz`. Dokumentenrevision: 6020-0065-05 Rev A.

[17] Memsic Inc. MTS400CC datasheet. `http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=174%3Amts400_420`. Dokumentenrevision: 6020-0065-05 Rev A.

[18] Memsic Inc. TelosB datasheet. `http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=152%3Atelosb`. Dokumentenrevision: 6020-0094-04 Rev A.

[19] ScatterWeb Project. MSB430-H information. `http://cst.mi.fu-berlin.de/projects/ScatterWeb/mod_MSB-430H.html`.

[20] ScatterWeb Project. Scatterweb homepage. `http://cst.mi.fu-berlin.de/projects/ScatterWeb/index.html`.

[21] ScatterWeb Project. ScatterWeb^2 software. `http://cst.mi.fu-berlin.de/projects/ScatterWeb/sof_ScatterWeb2.html`.

[22] ScatterWeb Project. TinyOS support in ScatterWeb. `http://cst.mi.fu-berlin.de/projects/ScatterWeb/sof_TinyOS.html`.

[23] K. Sohraby, D. Minoli, and T. Znati. *Wireless Sensor Networks: Technology, Protocols, and Applications*. Wiley-Interscience, 2007.

[24] H. Soliman, K. Sudan, and A. Mishra. A smart forest-fire early detection sensory system: Another approach of utilizing wireless sensor and neural networks. In *Sensors, 2010 IEEE*, pages 1900 –1904, nov. 2010.

[25] Texas Instruments. MSP430F1611 datasheet. `http://www.ti.com/lit/gpn/msp430f1611`.

[26] E. M. Yeatman. Advances in power sources for wireless sensor nodes. In *Proc. of BSN 2004*, pages 6–7, 2004.

Energieverbrauch von Sensorkomponenten - Optimierung der Sensorkommunikation

Johannes Ziegltrum
Betreuer: Christoph Söllner, Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik
Technische Universität München
Email: johannes_ziegltrum@gmx.de

***Kurzfassung*—**
Die moderne Forschung verspricht sich großen Nutzen von drahtlosen Sensornetzen. Dabei handelt es sich um ein Rechnernetz, das aus einzelnen Knoten aufgebaut ist, die ihre Umgebung mittels Sensoren erfassen, diese Informationen verarbeiten und per Funk weiterleiten. Diese Technologie gestattet eine großflächige, sehr genaue Beobachtung bzw. Überwachung von Phänomenen der realen Welt. In den meisten Fällen sind Sensorknoten batteriebetrieben. Da es oft nicht möglich ist, diese Batterien auszutauschen bzw. wieder aufzuladen besteht eine zentrale Aufgabe der Forschung darin, den Energieverbrauch der Knoten so gering wie möglich zu halten, um die Lebenszeit von Sensornetzen zu verlängern. Deshalb sollte man sich bereits beim Entwurf und der Entwicklung von Netzwerkprotokollen darüber Gedanken machen, welche Möglichkeiten es gibt, um Energie zu sparen.

***Schlüsselworte*—**
Drahtloses Sensornetz, Sensorknoten, Multihop-Routing, Arbeitszyklus, S-MAC, T-MAC, Datenaggregation, DCE

I. Einleitung

Ein drahtloses Sensornetz ist ein Rechnernetz von Sensorknoten. Dabei handelt es sich um Computer, die per Funk miteinander kommunizieren. Aufgrund der in Zukunft anvisierten Größe werden Sensorknoten oft auch als Smart Dust (engl. für intelligenter Staub) bezeichnet.
Ein typischer drahtloser Sensorknoten beinhaltet in der Regel vier Hauptkomponenten: (1) Ein Abtastungs-System, das mittels Sensoren Daten erfasst, (2) ein Verarbeitungs-System mit einem Prozessor und einem Datenspeicher, (3) ein Funk-System zum Senden der Daten, sowie ein Stromversorgungs-System, das die benötigte Energie bereitstellt (in der Regel über eine Batterie).

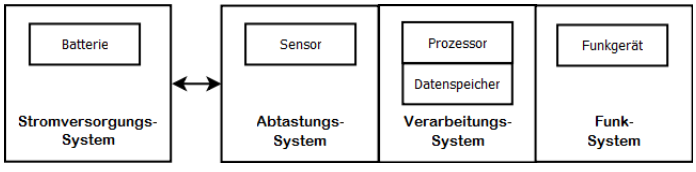


Abbildung 1. Die vier Hauptsysteme eines Sensorknotens [1]

Ein wichtiges Merkmal ist das sogenannte Ad-hoc-Netzwerk, bei dem die einzelnen Sensorknoten in einem Sensorfeld dazu in der Lage sind, selbständig ihre Nachbarn zu finden. Auf diese Weise bildet sich eine dynamische Netzstruktur, die sich automatisch anpasst, wenn sich Knoten bewegen, hinzukommen oder ausfallen. Dabei werden die Daten oft nicht direkt an die sogenannte Basis-Station (Benutzer-Schnittstelle) gesendet, sondern solange an benachbarte Knoten weitergeleitet, bis das Ziel erreicht wird (Multihop-Routing).

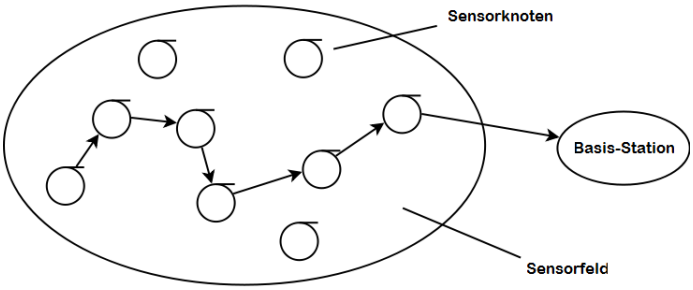


Abbildung 2. Ad-hoc-Netzwerk mit Multihop-Routing [1]

Ziel dieser Netze ist die Beobachtung und Überwachung verschiedenster Vorgänge in der realen Welt, wie beispielsweise Temperaturmessungen für Wetterstationen oder Vibrationsmessungen für Frühwarnsysteme. Dafür werden die Knoten an gewünschter Stelle verteilt, um dort anhand ihrer Sensoren Informationen über ihre Umwelt zu erfassen, diese Informationen dann mittels Prozessor zu verarbeiten und gegebenenfalls im Datenspeicher zu sichern, um sie schließlich per Funk an andere, benachbarte Sensorknoten zu schicken. Ein großes Problem dieser Technologie stellt der Energiebedarf der Sensorknoten dar. In den meisten Fällen wird die benötigte Energie von einer Batterie bezogen, die irgendwann aufgebraucht ist. Da es aber nicht immer möglich ist, diese Batterie auszutauschen oder wiederaufzuladen und man dennoch daran interessiert ist, diese Systeme über Monate oder sogar Jahre hinweg aufrecht zu erhalten, stellt sich die Frage, wie man die Laufzeit solcher Netze verlängern kann [2]–[4].
Zum einen versucht man die Batterietechnologie an sich zu verbessern und neue Methoden zur Energiegewinnung zu erforschen, wie beispielsweise Photovoltaik (Umwandlung von

doi: 10.2313/NET-2011-07-1_02

Sonnenenergie in elektr. Energie mittels Solarzellen), zum anderen gibt es aber auch Möglichkeiten, Energie an den Sensorknoten selbst einzusparen. In dieser Arbeit wird im Besonderen darauf eingegangen, in welchen Bereichen Energie verbraucht wird und welche verschiedenen Möglichkeiten es gibt, den Stromverbrauch zu reduzieren.
Zum Aufbau der Arbeit: In Kapitel II wird zunächst ein kurzer Überblick gegeben bevor in Kapitel III und IV auf konkrete Lösungsvorschläge eingegangen wird. Kapitel III erläutert wesentliche Ursachen für unnötigen Energieverbrauch und präsentiert zwei frühe MAC Protokolle, Sensor Media Access Control (S-MAC) und Timeout Media Access Control (T-MAC). Kapitel IV zeigt, wie das Problem redundanter Daten mittels Datenaggregation gelöst werden kann, bevor in Kapitel V eine kurze Zusammenfassung gegeben wird.

II. Überblick

Vergleicht man die drei Systeme Abtastung, Verarbeitung und Funk bezüglich ihres Energiebedarfs, so stellt sich heraus, dass vor allem die Kommunikation, also das Senden, das Empfangen und das Warten auf Daten, die meiste Energie verbraucht. Allerdings sind einzelne Knoten nicht ununterbrochen an einer Kommunikation beteiligt und auch das Nachrichtenaufkommen ist eher gering, da in der Regel nur selten Ereignisse auftreten. Aus diesem Grund konzentriert man sich bei der Reduzierung des Energieverbrauchs in erster Linie auf die Kommunikation der Sensorknoten. Eine Möglichkeit hierfür ist die Einführung von Arbeitszyklen, die einzelnen Knoten regelmäßiges Abschalten in einen 'Schlafmodus' erlaubt. Das Ziel eines MAC Protokolls ist es nun, trotz Schlaf-Phasen ausreichende Kommunikation zu ermöglichen. Zahlreiche Protokolle lösen diese Optimierung auf unterschiedliche Art und Weise [2].
Aufgrund der Tatsache, dass mehrere Knoten oftmals ähnliche Phänomene beobachten, gibt es viel Redundanz in den erfassten Daten benachbarter Knoten. Gibt es keine direkte Verbindung zur Basis-Station, werden die Daten von Knoten zu Knoten transferiert. Dadurch können Knoten ihre eigenen Daten mit denen, die sie von anderen empfangen haben vergleichen und zusammenfassen. Durch diese Vorarbeit kann die zu sendende Datenlast minimiert werden [1].
Im Folgenden werden allgemein also zwei energiesparende Techniken vorgestellt: Arbeitszyklus und Datenaggregation.

III. Arbeitszyklus

Sind Sensorknoten ununterbrochen in Betrieb, führt das dazu, dass der Energievorrat schnell aufgebraucht ist. Um die Lebensdauer eines drahtlosen Sensornetzes zu verlängern, ist es deshalb sinnvoll, bestimmte Komponenten zeitweise abzuschalten. Solange Daten gesendet oder empfangen werden müssen, befinden sich die entsprechenden Sensorknoten im aktiven Zustand. Sind sie untätig, werden sie in einen passiven Zustand gesetzt, in dem sie sehr viel weniger Energie verbrauchen. Je öfter ein Sensorknoten also aktiv ist, desto mehr Energie verbraucht er.

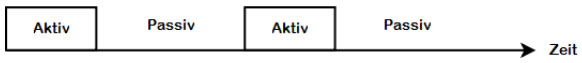


Abbildung 3. Aktiv-Passiv-Zyklus [5]

Folgende Ursachen für unnötigen Energieverbrauch lassen sich identifizieren:

- **Overhearing**: Das Mithören einer Kommunikation, an der ein Knoten nicht beteiligt ist. Ständiges Abhören ist eine einfache Möglichkeit, um herauszufinden, ob ein Medium für den eigenen Datenaustausch zur Verfügung steht. Falls das Medium belegt ist, führt das allerdings oft zu einem unnötigen Zeit- und Energieverlust [5].
- **Idle Listening**: Ein Knoten wartet auf den Empfang von Daten. Findet in einem Netzwerk Kommunikation ohne Vorankündigung statt, so ist Idle Listening die einzige Möglichkeit, Nachrichten zu empfangen. Werden nur wenige Nachrichten gesendet, kann dies zu einem erheblichen Energieverbrauch führen [5].
- **Datenkollision**: Verlust von Daten, weil mehrere Knoten gleichzeitig auf ein gemeinsames Medium zugreifen und somit die Nachrichten auf Empfängerseite nicht mehr zu entziffern sind. Für mögliche Wiederholungen muss dafür zusätzliche Energie aufgebracht werden [5].
- **Kontrollnachrichten**: In vielen Netzwerken entsteht zusätzlicher Mehraufwand durch Kontrollnachrichten, die in der Regel keine für die Anwendung relevanten Informationen beinhalten [5].

Ziel ist es nun, den Sensorknoten passende Arbeitszyklen von aktiven und passiven Zeiten so zuzuordnen, dass sie miteinander kommunizieren können, ohne dass unverhältnismäßig hoher Energieverlust durch Overhearing, Idle Listening, Kollisionen oder Kontrollnachrichten entsteht. Im Folgenden werden nun die gebräuchlichsten Ansätze aufgezeigt, mit denen der Arbeitszyklus organisiert werden kann [6].

A. S-MAC

Das S-MAC-Protokoll (Sensor Media Access Control) beruht auf der Rendezvous-Technik. Will ein Sensorknoten Daten senden, so muss er zunächst eine Sendeanfrage (RTS, request to send) an den betreffenden Empfänger schicken. Antwortet dieser mit einer Sendebestätigung (CTS, clear to send), so weiß der Sender, dass er seine Daten nun übermitteln kann. Nach dem Erhalt der Daten antwortet der Empfänger wiederum mit einer Empfangsbestätigung (ACK, acknowledgement). Damit bei einem Übermittlungsfehler nicht die gesamten Daten erneut versendet werden müssen, verfolgt S-MAC den Ansatz, große Dateneinheiten in kleinere Pakete aufzuteilen [Abb.4].

Beim S-MAC-Konzept wird den Sensorknoten vor Betriebsbeginn ein Aktiv-Passiv-Zyklus vorgegeben, der ihren 'Tagesablauf' bestimmt. Dabei werden die Knoten immer wieder in einen Ruhezustand mit geringem Energieverbrauch versetzt. In diesem Zustand bleibt lediglich eine interne Uhr in Betrieb, die vorgibt, zu welchem Zeitpunkt der Knoten wieder in den

doi: 10.2313/NET-2011-07-1_02

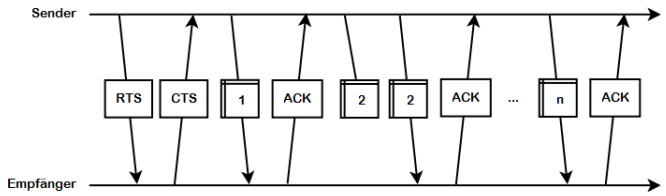


Abbildung 4. Datenübermittlung bei S-MAC [7]

aktiven Zustand übergehen soll. Dabei sollten möglichst viele Knoten dem gleichen Tagesablauf folgen, damit sie in ihrer gemeinsamen Aktiv-Phase miteinander kommunizieren können. Um dies sicherzustellen, gibt S-MAC einen Mechanismus vor, mit dem mehrere Sensorknoten ihre Zeitpläne aufeinander abstimmen können.
Zu Betriebsbeginn wartet jeder Sensorknoten eine gewisse Zeit auf den Empfang eines Synchronisationsimpulses (SYNC). Bleibt dieser aus, weist sich ein Knoten nach einer zufälligen Zeitspanne selbst die Rolle des Synchronizers zu, indem er die Information aussendet, zu welchem Zeitpunkt und für wie lange er in den Ruhezustand übergeht. Wer einen Synchronisationsimpuls empfängt, nimmt die Rolle eines Followers ein und passt seinen eigenen Zeitplan dem im SYNC enthaltenen Tagesablauf an.

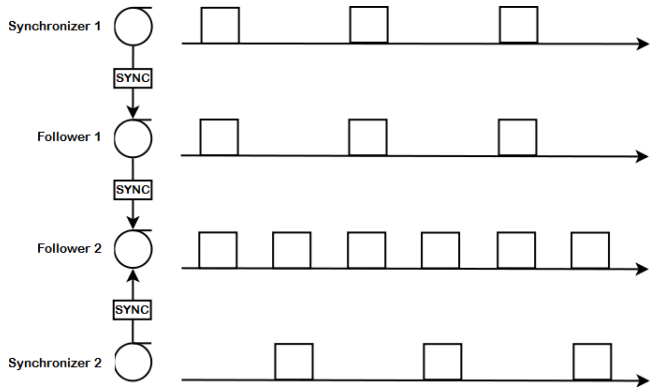


Abbildung 5. Synchronisation der Zeitpläne von Sensorknoten [6]

Allerdings garantiert dieses Verfahren keine netzwerkweite Synchronisation. Werden Synchronisationsimpulse gleichzeitig an verschiedenen Stellen des Netzes gestartet, so können sich Cluster bilden, die jeweils nur in sich synchronisiert sind. Eine Kommunikation zwischen diesen Clustern ist dennoch möglich, indem Knoten im Grenzbereich als Brückenknoten agieren und sich dem Tagesablauf aller benachbarten Cluster anpassen. Dies führt dazu, dass diese Grenzknoten schneller ausfallen, da sie öfter aktiv sind und somit mehr Energie verbrauchen [Abb.6].

Da Uhren niemals völlig gleich gehen, kann es während des Betriebs zu zeitlichen Abweichungen der internen Uhren kommen. Um diese zu korrigieren, teilt S-MAC die Aktiv-Phasen in zwei Unterphasen auf. In der ersten, kürzeren Phase werden Synchronisationsnachrichten gesendet, die zweite, längere Phase ist dann für den eigentlichen Datenaustausch gedacht.

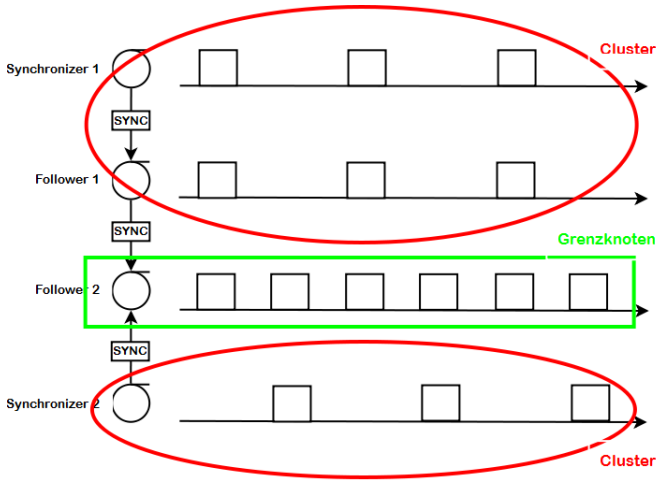


Abbildung 6. Bildung eines Grenzknotens zwischen benachbarten Clustern [6]

Auf diese Weise ist es auch möglich, bei fortlaufendem Betrieb neue Knoten in das bestehende Netz mit aufzunehmen. Nach dem Erhalt einer Synchronisationsnachricht können sie sich zeitlich orientieren und passen sich dem Tagesablauf an.
Zur Reduzierung von Kollisionen wird CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) eingesetzt. Dabei handelt es sich um ein Verfahren, mit dem ein Übertragungsmedium vor einem geplanten Zugriff auf Belegung geprüft werden kann (Trägerprüfung). Um gleichzeitig Overhearing zu vermeiden, enthält jede Sendeanfrage die entsprechende Information über die Dauer einer Sendung. Diese Zusatzdaten werden im NAV (Network Allocation Vector) gespeichert. Mithörende Knoten können anhand dieses Netzbelegungsvektors die Länge einer Kommunikation bestimmen und sich für diese Zeit wieder schlafen legen. Ein Zeitgeber zählt den Wert des Vektors nach und nach herunter. Wird der Wert Null erreicht, so wissen andere Knoten, dass das Medium wieder frei ist [5]–[7].
Um die Verzögerung bei Multihop-Übertragungen zu reduzieren, wurde S-MAC um Adaptive Listening erweitert. Mit dieser Optimierung kann im Idealfall pro Aktiv-Passiv-Zyklus ein zusätzlicher Hop überwunden werden, indem die entsprechenden Knoten nach ihrer eigentlichen Aktiv-Phase erneut für eine bestimmte Zeit aufwachen [8].

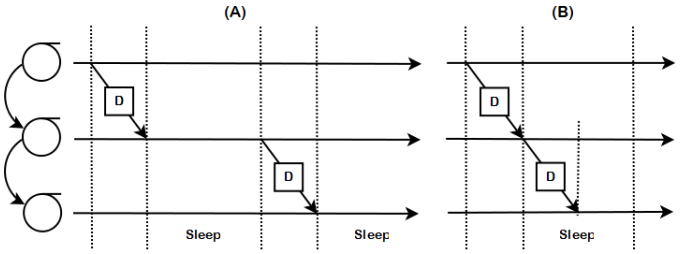


Abbildung 7. Multihop-Routing ohne **(A)** und mit **(B)** Adaptive Listening

Ein ungelöstes Problem beim S-MAC stellt das Early-

doi: 10.2313/NET-2011-07-1_02

Sleeping-Problem dar. Oft kommt es zu der Situation, dass Knoten in den Schlafmodus wechseln, obwohl einem Nachbarknoten Daten zur Übertragung vorliegen. Dieser kann seine Sendeanfrage aber nicht abschicken, da er gerade in eine andere Kommunikation verwickelt ist. Um dieses Problem zu lösen und weitere Effizienzsteigerungen zu erreichen, wurde der Nachfolger T-MAC entwickelt.

B. T-MAC

Das T-MAC-Protokoll (Timeout Media Access Control) basiert auf dem eben dargestellten S-MAC-Verfahren und setzt wie sein Vorgänger auf einen regelmäßig vereinbarten Aktiv-Passiv-Zyklus. Dementsprechend synchronisieren Sensorknoten ihre Tagesabläufe, um möglichst große Gruppen zu bilden, die gemeinsam aktiv werden und somit miteinander kommunizieren können. Anders als beim S-MAC wird die Dauer der aktiven Phase aber nicht fest vorgegeben, sondern kann von Knoten zu Knoten immer wieder variieren. Werden für eine festgelegte Zeitspanne TA keine Nachrichten empfangen, gehen die entsprechenden Knoten für eine bestimmte Zeit in den Ruhezustand über. Dieser Timeout war namensgebend für das Protokoll. Knoten, die vor der Zeitüberschreitung Nachrichten empfangen, setzen ihre aktive Phase fort. Beim T-MAC wachen also alle Knoten eines Clusters gleichzeitig auf, können sich aber innerhalb ihrer aktiven Phase zu unterschiedlichen Zeitpunkten wieder schlafen legen, falls sie an keiner Kommunikation teilnehmen. Somit verkürzen sich die aktiven Zeiten im Vergleich zum S-MAC noch einmal.

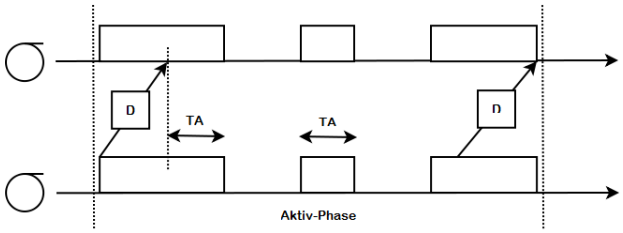


Abbildung 8. Findet für die Zeitspanne TA keine Kommunikation statt, legen sich die Knoten für eine gewisse Zeit schlafen [6]

T-MAC folgt bei der Datenübertragung wie S-MAC dem Rendezvous-Schema mit RTS, CTS und ACK. Allerdings wird dieses Schema bei T-MAC um eine zukünftige Sendeanfrage (FRTS, future request to send) erweitert, um das Early-Sleeping-Problem zu lösen.

In Abb. 9 wird dies an einem Beispiel verdeutlicht. Knoten A sendet Daten an B. Knoten C wiederum würde gerne Daten an D senden. Da C aber die Antwortpakete von B hören kann und die Kommunikation nicht stören will, muss er mit seiner Sendeanfrage an D warten, bis A und B ihre Kommunikation beendet haben und geht für diese Zeit schlafen. Knoten D bekommt von alledem nichts mit und geht ebenfalls nach einer gewissen Zeit schlafen, da er an keiner Datenübertragung beteiligt ist. Nachdem Knoten C wieder aufgewacht ist, versucht er vergebens D zu erreichen und muss schließlich aufgeben und in der nächsten Periode um das Medium kämpfen.

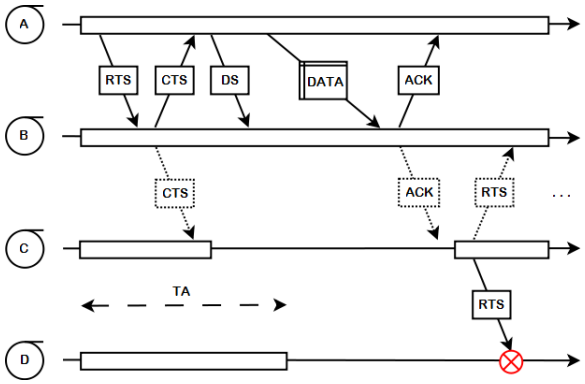


Abbildung 9. Early-Sleeping-Problem [9]

Dieses Problem soll nun gelöst werden, indem man Knoten erlaubt, zukünftige Sendeanfragen zu äußern.

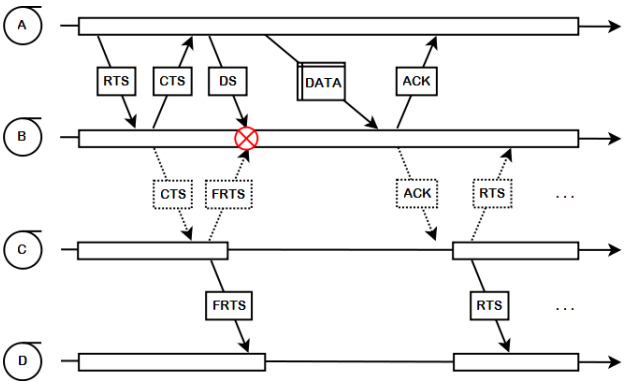


Abbildung 10. FRTS um das Early-Sleeping-Problem zu lösen [10]

Knoten C hört das CTS von B und hat unmittelbar danach die Möglichkeit, eine spätere Sendeanfrage (FRTS) an Knoten D zu äußern. Dadurch weiß Knoten D zum einen von der Sendeabsicht C's, zum anderen können beide nun eine genau bemessene Schlafphase einlegen, da über das FRTS auch die Dauer der Kommunikation zwischen Knoten A und B mitgeteilt wurde. Damit das FRTS-Paket mit keinen Nutzdaten von Knoten A kollidiert, verzögert A die Datensendung, indem es nach dem CTS von B zunächst ein Datensignal (DS, data signal) schickt. Dieses DS-Paket hat dieselbe Größe wie ein FRTS, wodurch es zur Kollision dieser beiden Pakete kommt. Dies ist aber nicht weiter problematisch, da das DS einerseits keine Nutzdaten enthält, sondern dem Empfänger lediglich ankündigt, dass danach die eigentlich Datensendung beginnt und andererseits dafür gedacht ist, das Medium für diesen Zeitraum zu belegen, um die Übernahme anderer Nachbarknoten von A zu verhindern. Die Knoten C und D wachen zum vereinbarten Zeitpunkt auf und können nun ihrerseits Daten austauschen [Abb.10].

Sensorknoten sammeln ihre zu sendenden Daten in einem Nachrichtenausgang. Bekommt ein Knoten selten die Möglichkeit seine Nachrichten abzuschicken, kann es passieren, dass sein Datenpuffer irgendwann voll ist. Um dies zu verhindern,

doi: 10.2313/NET-2011-07-1_02

schlägt T-MAC eine weitere Neuerung vor. Falls sich bei einem Knoten Daten anhäufen, will dieser bevorzugt Nachrichten versenden und bekommt daher die Möglichkeit, seinen Sendewünschen eine höhere Priorität einzuräumen.

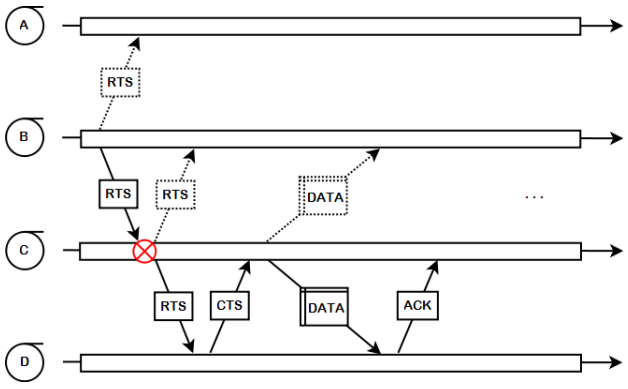


Abbildung 11. Priorisierung durch Sendeanfrage-Ablehnung [9]

Knoten B will Daten an C senden und schickt deshalb ein RTS an C. Da dessen Datenpuffer aber fast voll ist, ignoriert er die Sendeanfrage von B und schickt seinerseits ein RTS an D. Knoten B nimmt das wahr und verzichtet daraufhin auf seine Datenübertragung. Es kommt zur Kommunikation zwischen C und D.
Allerdings ist hierbei Vorsicht geboten. In der Folge kam es oft dazu, dass sich mehrere Nachrichtenausgänge anfüllten und somit viele Knoten Anfragen ablehnten, was den Datendurchsatz stark senkte. Damit bei hohem Datenaufkommen nicht das ganze Netz zum Erliegen kommt, wendet ein Knoten mit fast vollem Puffer diese Technik erst nach zwei fehlgeschlagenen Sendeanfragen an [5], [6], [10].

C. Weitere MAC-Protokolle

Die Verfahren S-MAC und T-MAC sind nur zwei von vielen weiteren Methoden, um die Arbeitszyklen von Sensorknoten festzulegen. Da die ausführliche Erklärung aller Verfahren den Rahmen dieser Arbeit sprengen würde, wird im Folgenden in ein bis zwei Sätzen erklärt, welche grundsätzlichen Strategien andere Protokolle verfolgen:

- **TRAMA**: Um kollisionsfreie Kommunikation zu realisieren, werden bei TRAMA zwei zeitlich voneinander getrennte Kanäle für die Nachrichtenübertragung verwendet. Über einen Kanal mit zufälligem Zugriff erhalten die Knoten Informationen über benachbarte Knoten. Über den zweiten, kollisionsfreien Kanal werden die Nutzdaten gesendet, indem jedem Knoten basierend auf beabsichtigten Sendeaktivitäten feste Zeitschlitze für die Übertragung zugeordnet werden [6].
- **B-MAC**: Ziel des B-MAC-Protokolls ist es, Funktionen wie Netzinitialisierung und -synchronisation in höherschichtige Module auszugliedern. Aus diesem Grund beschränkt sich B-MAC einzig auf die Medienzugriffskontrolle und verwendet ein Rauschunterdrückungsverfahren, um Signalempfang und Trägerprüfung zuverlässiger zu machen [11].
- **D-MAC**: D-MAC wurde für effizienteres Multihop-Routing entwickelt. Da Multihop-Übertragungen jeweils durch Schlafzyklen unterbrochen werden, nutzt D-MAC Topologieinformationen, um die Aktiv-Phasen benachbarter Knoten so anzuordnen, dass sie sich überlappen. Beim Senden von Daten entsteht so eine Kettenreaktion, die die Verzögerung von Blattknoten zur Basis-Station minimiert [5].

IV. Datenaggregation

In drahtlosen Sensornetzen werden Informationen oft von Sensorknoten zu einer Basis-Station geleitet. Bei dieser Kommunikationsform haben Knoten einen ungleichmäßig verteilten Energieverbrauch, da sie unter dem sogenannten 'Trichtereffekt' leiden. Je näher sich die Knoten an der Basis-Station befinden, desto häufiger werden Informationen per Multihop-Routing über sie weitergeleitet und desto höher ist dann auch die Wahrscheinlichkeit, dass sie früher ausfallen.

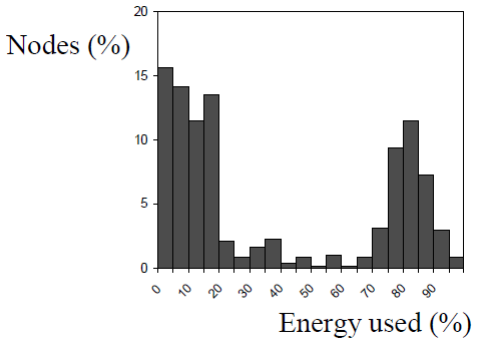


Abbildung 12. Unerwünschtes Energie-Histogramm: Viele Knoten haben noch fast vollen Energievorrat, während einige Knoten beinahe leer sind. [12]

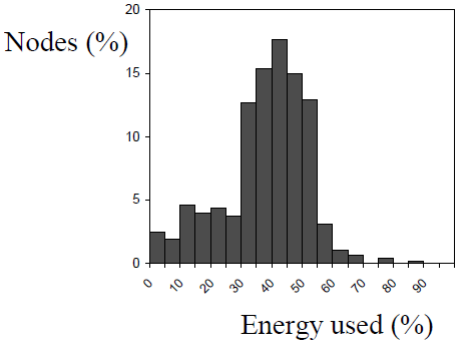


Abbildung 13. Erwünschtes Energie-Histogramm: Die meisten Knoten besitzen noch ähnlich große Energiereserven. [12]

Es hat sich gezeigt, dass der Einsatz von Datenaggregation die negativen Auswirkungen dieses Effekts reduzieren kann. In den erfassten Daten benachbarter Knoten gibt es viel Redundanz, da es oft vorkommt, dass mehrere Knoten dasselbe Phänomen beobachten (z.B. Temperaturmessungen in einem Raum). Um die Lebensdauer eines Sensornetzes zu verlängern, können diese redundanten Daten auf dem Weg von der Quelle

doi: 10.2313/NET-2011-07-1_02

zur Basis-Station zusammengefasst oder komprimiert werden [1].

A. Data Combining Entity

Die Energiekosten für die Übertragung von Daten sind wesentlich höher als die Kosten für die Verarbeitung dieser Daten. Erfassen die Sensoren verschiedener Knoten dasselbe Ereignis, bietet es sich also an, diese Daten zuerst in einem Knoten zusammenzufassen, bevor sie weitergeschickt werden. Bei diesen Knoten handelt es sich also um Einheiten, die verschiedene Daten kombinieren (DCE, data combining entity). Durch die Komprimierung redundanter Daten zu einer einzigen Datei können erhebliche Übertragungskosten eingespart werden.
Welche Knoten zu solchen Kombinierungseinheiten werden, kann explizit oder dynamisch festgelegt werden. Allerdings ist die explizite Festlegung vor allem bei sehr großen Sensornetzen oft zu aufwändig. Eine mögliche Vorgehensweise ist eine dynamische Wahl zur Laufzeit, bei der jeder Knoten je nach Bedarf zu einer DCE wird. Erfassen verschiedene Knoten zur gleichen Zeit Daten, handelt es sich dabei meist um Knoten, die sehr nahe beieinander stehen und dasselbe Ereignis beobachtet haben. Durch ihre räumliche Nähe zueinander fließen die gesendeten Daten irgendwann auf gleichen Pfaden zur nächsten Basis-Station. Knoten, bei denen mehrere Datenströme zusammenfließen, werden zu DCEs. Diese befinden sich meist in der Nähe des Ereignis-Radius [12].

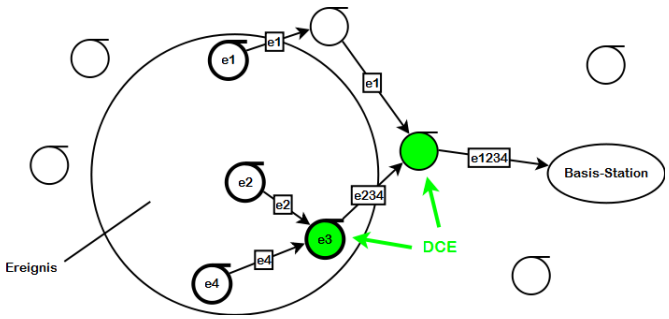


Abbildung 14. Dynamische Bildung von Data Combining Entities [13]

In einem bestimmten Gebiet wird ein Ereignis ausgelöst. Die Knoten, die sich innerhalb des Ereignis-Radius befinden, versuchen ihre erfassten Daten (e1, e2, e3, e4) per Multihop-Routing an die Basis-Station zu senden. Treffen auf dem Weg in einem Knoten Daten vom gleichen Ereignis aufeinander, so werden diese zunächst zusammengefasst (e234, e1234) bevor sie weitergeleitet werden. Die entsprechenden Knoten werden automatisch zu DCEs [13].

V. Zusammenfassung und Ausblick

Wie wir in dieser Arbeit sehen konnten, gibt es verschiedene Möglichkeiten, um energieeffiziente Kommunikation in drahtlosen Sensornetzen zu realisieren. Allerdings haben all diese Techniken Vor- und Nachteile, die man je nach Gegebenheit abwägen muss.
S-MAC versucht in erster Linie die knappen Ressourcen von Sensorknoten zu schonen und gilt als Vorläufer der spezialisierten Sensornetzprotokolle. Heute ist es aber lediglich noch von rein akademischem Interesse und wird gerne zum Vergleich zahlreicher nachfolgender Netzwerkprotokolle herangezogen. Später entwickelte MAC-Protokolle gingen gezielt auf die Schwächen von S-MAC ein und waren erwartungsgemäß effizienter. Bereits der direkte Nachfolger T-MAC konnte den Energieverbrauch durch geringfügige Modifikationen erheblich reduzieren. Die Entwickler von B-MAC wiederum kritisierten, dass S-MAC zu weit gehe, indem es sich um die Netzorganisation, Netzsynchronisation und Datenfragmentierung kümmere. Sie lagerten diese Aufgaben in eigene konfigurierbare Module aus.
Eine weitere Möglichkeit, die wir kennengelernt haben, um die Kommunikation in Sensornetzen zu verbessern, ist die Datenaggregation, die redundante Informationen mittels DCEs zusammenfasst. Mit zunehmender Anzahl an DCEs sinkt die zu sendende Datenmenge und damit der Energieverbrauch. Gleichzeitig steigt aber auch die Verzögerung im Netz, da Knoten mehr Zeit für die Kombinierung der Daten benötigen. Grundsätzlich kann man sagen, dass es nicht eine 'beste' Strategie gibt oder geben wird. Man muss immer einen Kompromiss zwischen Energieeinsparung und Dienstgüte finden. Für spezifische Szenarien müssen spezielle, auf das jeweilige Problem zugeschnittene Lösungen gefunden werden. Und durch den technischen Fortschritt wird man auch in Zukunft immer neue Wege finden, um den Energieverbrauch weiter zu reduzieren.

Literatur

[1] S. Haidan, "Datenzentrisches routing und directed diffusion in drahtlosen sensornetzen," pp. 1–5, 2004.
[2] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, 2009.
[3] S. Mark, "Energiesparende system-und schaltungskonzepte f ür drahtlose sensornetzwerke," Ph.D. dissertation, Universit ätsbibliothek, 2008.
[4] "Sensornetz." [Online]. Available: http://de.wikipedia.org/wiki/Sensornetz
[5] D. Christmann, "Duty cycling in drahtlosen multi-hop-netzwerken," Technical Report 372/09, TU Kaiserslautern, Tech. Rep., 2009.
[6] M. Neugebauer, *Energieeffiziente Anpassung des Arbeitszyklus in drahtlosen Sensornetzen*. J örg Vogt Verlag, 2007.
[7] "S-mac." [Online]. Available: http://de.wikipedia.org/wiki/S-MAC
[8] "Adaptive listening." [Online]. Available: http://www.scribd.com/doc/53349143/14/Adaptive-listening
[9] T. Van Dam and K. Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM, 2003, pp. 171–180.
[10] "T-mac." [Online]. Available: http://de.wikipedia.org/wiki/T-MAC
[11] "Berkeley_media_access_control." [Online]. Available: http://de.wikipedia.org/wiki/Berkeley_Media_Access_Control
[12] C. Schurgers and M. Srivastava, "Energy efficient routing in wireless sensor networks," in *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, vol. 1. IEEE, 2001, pp. 357–361.
[13] S. Shumilov, "Seminar verteilte informationssysteme," pp. 15–17, 2006.

doi: 10.2313/NET-2011-07-1_02

Einführung in Sensornetze - Abhängigkeiten zwischen Protokolldesign und verwendeter Hardware

Andreas Heider
Betreuerin: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: heidera@in.tum.de

KURZFASSUNG

Sensorknoten sollen klein, drahtlos und günstig sein und müssen daher mit begrenzten Ressourcen auskommen. Ihre vielfältigen Anwendungen stellen weitere Anforderungen, die nicht nur einzelne Knoten, sondern das gesamte Sensornetz beeinflussen. Im Rahmen dieser Arbeit werden die wichtigsten Zusammenhänge zwischen Hardware, Einsatzszenarien und Netzwerkstrukturen vorgestellt und gezeigt wie diese im Protokolldesign berücksichtigt werden können.

Schlüsselworte

Sensornetzwerke, Protokolldesign, Anwendungen, Hardware, Protokollstack

1. EINLEITUNG

Ein verteiltes Sensornetzwerk besteht aus hunderten bis zu mehreren tausenden über ein Einsatzgebiet verteilten Sensorknoten [11]. Mittels Sensoren überwachen diese ihre Umwelt und stellen die Messergebnisse dem Nutzer zur Verfügung. Dazu kommunizieren sie drahtlos untereinander.

Jeder Knoten kann dabei sowohl an der Kommunikation als auch an der Datensammlung und -verarbeitung beteiligt sein.

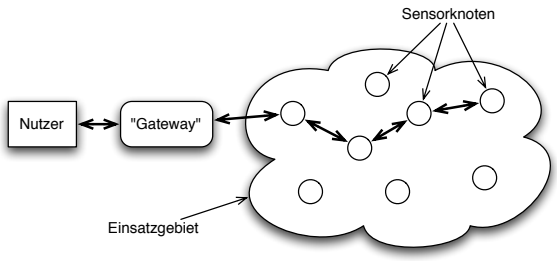


Abbildung 1: Aufbau eines einfachen Sensornetzwerks [6]

In Abbildung 1 ist der typische Aufbau eines einfachen Sensornetzwerks dargestellt. Der Nutzer kann über ein zentrales „Gateway“ auf die Sensorknoten zugreifen.

Von besonderer Bedeutung ist bei Sensornetzen der Energieverbrauch. Da sich die Sensorknoten meist batteriebetrieben sind und sich oft an schwer zugänglichen Orten befinden, steht ihnen nur eine begrenzte Menge an Energie zur Verfügung. Diese vorhandene Energie muss intelligent genutzt werden, um die Aufgabe des Sensornetzes bestmöglich zu erfüllen. Hierfür ist es nötig, sowohl die Anforderungen eines einzelnen Sensorknotens als auch das Zusammenspiel aller Knoten im Sensornetzwerk zu beachten.

Um diese und weitere Anforderungen zu erfüllen, sind klassische Techniken für den Aufbau und Betrieb von Ad-Hoc-Netzwerken nicht ausreichend. Daher werden spezialisierte Protokolle eingesetzt, die die besonderen Eigenschaften drahtloser Sensornetze beachten.

Ziel dieser Arbeit ist, die Besonderheiten drahtloser Sensornetze darzustellen und die Auswirkungen auf das Protokolldesign zu untersuchen.

Hierfür wird zunächst in Abschnitt 2 ein Überblick über die Hardware eines einzelnen Sensorknotens gegeben. In Abschnitt 3 werden typische Anwendungen für Sensornetze vorgestellt und klassifiziert. Darauf folgend wird in Abschnitt 4 die Netzstruktur eines Sensornetzes anhand der verschiedenen Topologien vorgestellt. In Abschnitt 5.1 werden die bisherigen Erkenntnisse zusammengefasst und konkrete Anforderungen an das Protokolldesign formuliert. Schließlich wird gezeigt wie diese im Protokollstack berücksichtigt werden können.

2. HARDWARE

Als nur ein Teil eines großen Netzwerks zeichnet sich ein einzelner Sensorknoten durch die Beschränkung auf das Nötige aus. Um den Energieverbrauch und die Kosten zu minimieren, sollte ein Sensorknoten nur die nötigen Komponenten beinhalten, um Datensammlung, Kommunikation und anwendungsspezifische Aufgaben auszuführen.

Ein Sensorknoten besteht mindestens aus den in Abbildung 2 dargestellten vier Hauptkomponenten: Den je nach Anwendung sehr verschiedenen Sensoren, der Recheneinheit, die die Steuerung übernimmt, einem Transceiver für die Kommunikation mit anderen Knoten und einer Energiequelle, die das gesamte System mit Strom versorgt.

Je nach Anwendungsfall können zusätzlich weitere Komponenten hinzukommen, die beispielsweise die Lokalisation des Knotens im Einsatzgebiet ermöglichen oder für Mobilität sorgen.

doi: 10.2313/NET-2011-07-1_03

Tabelle 1: Vergleich häufig verwendeter Mikrocontroller [9]

Name	Bit	Flash	RAM	Energieverbrauch		
				Aktiv	Idle	Sleep
AT90LS8525	8	8KB	512B	6.4 mA	1.9 mA	<1 uA
ATMEGA 103L	8	128KB	4KB	5.5 mA	1.6 mA	<1 uA
MSP430 F149	16	60KB	2B	400 uA	1.3 uA	<0.1 uA

2.1 Sensoreinheit

Die Sensoreinheit stellt die Verbindung zwischen dem Sensorknoten und seiner Umwelt her. Mittels anwendungsspezifischer Sensoren werden verschiedene Parameter gemessen, mit Aktoren kann ein Sensorknoten seine Umgebung auch beeinflussen.

Die benötigten Sensoren werden durch die zu messenden Umwelteigenschaften festgelegt. Sie stellen wiederum Anforderungen an die sonstige verwendete Hardware und den Aufbau des Sensornetzes. Maßgeblich ist dabei vor allem die zu übertragende Datenmenge, die nötige Rechenleistung für den Betrieb des Sensors und der Energieverbrauch.

Je nach Anwendungsfall können sich die verwendeten Sensoren und damit die Anforderungen stark unterscheiden. Für einen konkreten Anwendungsfall gibt es allerdings oft wenig Wahlmöglichkeiten zwischen verschiedenen Sensortechnologien. Daher muss sich meist die übrige Hardware an den anzusteuernden Sensoren orientieren.

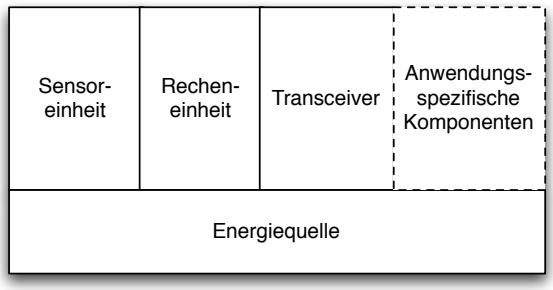


Abbildung 2: Die Hauptkomponenten eines Sensorknotens

2.2 Recheneinheit

Für die Steuerung des Sensorknotens werden meist Mikrocontroller verwendet. Diese vereinen einen Prozessor sowie Speicher auf einem Chip. In Tabelle 1 sind die wichtigsten Daten einiger häufig verwendeter Mikrocontroller dargestellt.

In Hinblick auf das Protokolldesign ist hier vor allem festzustellen: Die Rechenleistung sowie der verfügbare Speicher der Mikrocontroller ist äußerst beschränkt.

Des weiteren ist zu sehen, dass mehrere Energieverbrauchswerte angegeben werden. Moderne Mikrocontroller bieten verschiedene Betriebsmodi an, die sich stark im Energieverbrauch unterscheiden [3]. Mittels „Dynamic Voltage Scaling“ oder dem Abschalten momentan nicht benötigter Komponenten kann dieser deutlich verringert werden.

2.3 Transceiver

Die Kommunikation zwischen den Knoten eines Sensornetzwerks findet typischerweise drahtlos statt. Hierfür wird ein Transceiver verwendet.

Ähnlich den Betriebsmodi der Recheneinheit kann auch der Transceiver bei Nicht-Gebrauch abgeschaltet werden. Während dem Senden (Tx) von Daten wird sehr viel Energie verbraucht.

Daher ist ein Ziel des Sensornetzwerkdesigns, den Transceiver möglichst selten und kurz zu aktivieren. Die meiste Zeit sollte sich der Transceiver daher entweder im Empfangsmodus (Rx) befinden oder ganz deaktiviert sein.

Essentiell für die Wahl des Transceivers, der Antenne sowie des Netzwerkaufbaus ist dabei die Reichweite der drahtlosen Verbindung. Denn die für eine Übertragung benötigte Energie ist exponentiell von der Distanz d zwischen den Knoten abhängig [3]. Während das theoretische Minimum noch d^2 ist, entspricht der tatsächliche Energieverbrauch in der Praxis oft sogar d^3 bis d^4. Daher ist es wichtig beim Netzwerkaufbau sowie der Wahl der einzusetzenden Netzwerktopologie auf eine möglichst geringe Distanz zwischen den miteinander kommunizierenden Knoten zu achten.

Der Energieverbrauch eines Transceivers lässt sich noch weiter untergliedern. Dies wird in [4] am Beispiel des Empfangs einer DVB-T Übertragung dargestellt. Dabei werden folgende Hautverbraucher identifiziert:

Der Analogteil verbraucht etwa 10% der Gesamtenergie und umfasst diverse Filter sowie die Analog-Digital-Umwandlung. Mit 60% entfällt der Hauptteil des Energieverbrauchs auf die digitale Demodulation. Hierbei ist insbesondere die nötige Fouriertransformation (FFT) hervorzuheben, die allein bereits 30% des Energieverbrauchs ausmacht. Die übrige Energie verteilt sich mit 20% auf den „Channel decoder“ sowie mit 10% auf sonstige Verbraucher.

2.4 Energiequelle

Da Sensorknoten oft schwer zugänglich und über ein großes Gebiet verteilt sind, ist es meist nicht möglich diese kabelgebunden mit Strom zu versorgen. Daher ist ein essentieller Bestandteil jedes Sensorknotens seine jeweilige Energieversorgung. Die Zeit, die ein Sensorknoten arbeiten kann, ist abhängig von der vorhandenen Energie sowie dem Energieverbrauch. Die Kapazität der Energiequelle beeinflusst also die Lebenszeit des Sensorknotens entscheidend.

Die einfachste und am häufigsten verwendete Energiequelle ist die Batterie. Mit einer Ladung können Sensorknoten teilweise mehrere Jahre arbeiten bis sie erneuert werden muss. Allerdings besitzt eine Batterie nur eine feste Kapazität.

doi: 10.2313/NET-2011-07-1_03

Tabelle 2: Betriebsmodi eines Sensorknotens [3]

Modus	Recheneinheit	Sensorik	Transceiver
s_0	Aktiv	Aktiv	Tx, Rx
s_1	Idle	Aktiv	Rx
s_2	Sleep	Aktiv	Rx
s_3	Sleep	Aktiv	Inaktiv
s_4	Sleep	Inaktiv	Inaktiv

Um eine längere Laufzeit zu erreichen kann mittels „Energy Harvesting“ Energie aus der Umgebung des Sensorknotens genutzt werden. Ein Sensorknoten kann beispielsweise mit einer Solarzelle ausgestattet werden und so tagsüber einen Akkumulator laden. Wichtig ist hierbei die Verfügbarkeit der Energiequelle, die nötige Größe um genug Strom zu erzeugen sowie die Kosten für den Einsatz.

Idealerweise wird durch „Energy Harvesting“ mehr Energie erzeugt, als der Sensorknoten für den Betrieb benötigt.

Da die Effektivität einiger Energiegewinnungsverfahren von der Tages- sowie der Jahreszeit abhängt, müssen diese Eigenschaften ebenfalls in den verwendeten Protokollen beachtet werden. So ist es denkbar, solarbetriebene Sensorknoten bevorzugt tagsüber bei hoher Sonneneinstrahlung zu aktivieren.

2.5 Betriebsmodi

Da die Stromsparfunktionen der Komponenten voneinander abhängig sind, ergeben sich für einen vollständigen Sensorknoten beispielsweise die in Tabelle 2 aufgelisteten Betriebsmodi. Da nicht alle Kombinationen der Energiesparmodi der einzelnen Komponenten sinnvoll sind, werden hierfür oft nur wenige Kombinationen ausgewählt.

Im Allgemeinen ist es Aufgabe des Betriebssystems, je nach Bedarf zwischen den Betriebsmodi zu wechseln.

Allerdings bringt ein Übergang in einen sparsameren Betriebsmodus auch Kosten mit sich: Jeder Moduswechsel benötigt eine gewisse Zeit um durchgeführt zu werden und verbraucht damit selbst Energie. Daher muss das Betriebssystem möglichst viele Informationen zur Verfügung haben, um gut entscheiden zu können, wann in einen sparsameren Modus gewechselt werden sollte.

Insbesondere für die Kommunikation spielt auch das verwendete Netzwerkprotokoll eine wichtige Rolle. Um den Stromverbrauch zu minimieren, sollte der Transceiver oft ganz ausschaltet sein. Dazu muss sichergestellt werden, dass der Knoten eine gewisse Zeit keine Pakete empfangen muss. Um solche Optimierungen implementieren zu können ist es notwendig, alle Aspekte eines Sensornetzwerks zu beachten.

3. ANWENDUNGEN

Der klassische Anwendungsfall für ein Sensornetzwerk ist das Überwachen von physikalischen Messwerten. Diese werden an ein zentrales „Gateway“ gesendet, über das der Nutzer auf die Messungen zugegreifen kann. Dies entspricht dem in Abbildung 1 dargestellten Fall.

3.1 Klassifikation

Es gibt allerdings noch andere Einsatzfälle für drahtlose Sensornetze. In [8] wird ein Klassifikationsschema für Anwendungen von Sensornetzwerken vorgestellt, das diese nach diversen Gesichtspunkten gruppiert.

3.1.1 Ziel

Ein Kriterium ist dabei das Ziel des Sensornetzes. Hier wird unterschieden zwischen Anwendungen in denen die Sensorknoten nur Messaufgaben übernehmen und solchen, in denen sie mittels Aktoren ihre Umwelt auch beeinflussen.

Ein Beispiel für ein solches drahtloses Sensor- und Aktor-Netzwerk ist die automatische Temperatursteuerung durch zusätzliche Aktorknoten. Anstatt die Klimatisierung des Einsatzgebiets nur zu überwachen, könnte ein solches Netz auch aktiv eingreifen und die Belüftung steuern. Weitere Informationen zu diesem Typ von Sensornetzen sind in [5] zu finden.

Soll das Sensornetzwerk auch reagieren können ergeben sich völlig neue Anforderungen für den Netzaufbau und die verwendeten Protokolle. Es müssen nun einige Knoten die Aufgabe übernehmen, Entscheidungen über das auszuübende Verhalten zu fällen. Um diese Steuerentscheidungen zu den ausführenden Knoten zu übermitteln muss ein Rückkanal vorhanden sein.

3.1.2 Interaktionsmuster

Diese Anforderung führt direkt zu dem zweiten Unterscheidungsfaktor, dem Interaktionsmuster zwischen den Knoten. Der klassische Fall eines Sensornetzwerks mit zentralem „Gateway“ entspricht dabei „many-to-one“: Viele verteilte Sensorknoten schicken Daten an ein „Gateway“.

Kann statt nur über ein zentrales „Gateway“ über viele Knoten auf das Netzwerk zugegriffen werden oder müssen Steuerknoten mit anderen Sensorknoten kommunizieren handelt es sich um ein „many-to-many“-Szenario. Hierbei kann jeder Knoten mit jedem anderen kommunizieren. In seltenen Fällen sind auch „one-to-many“-Sensornetze zu finden, also Netzwerke in denen nur ein Knoten Nachrichten an viele andere schickt.

3.1.3 Mobilität

Wie bereits in Abschnitt 2 angemerkt wurde können Sensorknoten auch beweglich sein. Dies wird im Klassifikationsschma als Mobilität berücksichtigt. Im Gegensatz zu statischen können mobile Sensorknoten ihre Position während der Laufzeit ändern.

Dabei kann zusätzlich zwischen Quellen-mobilen und Senken-mobilen Anwendungen unterschieden werden. In Quellen-mobilen Sensornetzen sind die datensammelnden Sensorknoten beweglich. Für diese Einsatzzwecke ist insbesondere die Verfolgung und Lokalisation der Sensorknoten in dem Einsatzgebiet sowie ein regelmäßiges Aktualisieren der Routingtabellen nötig. Senken-mobile Anwendungen zeichnet dagegen aus, dass die Senke (das „Gateway“) nicht statisch ist und beispielsweise durch einen Roboter, der regelmäßig seine Position ändert realisiert sein kann. In diesem Fall werden besonders hohe Anforderungen an die verwendeten Aggregationsalgorithmen gestellt.

doi: 10.2313/NET-2011-07-1_03

Tabelle 3: Beispiele verschiedener Anwendungen und ihre Klassifizierung [8]

Anwendung	Ziel	Interaktion	Mobilität	Lokalität	Zeit
Gletscherüberwachung	„sense only“	„many-to-one“	statisch	global	periodisch
Verkehrssteuerung	„sense and react“	„many-to-many“	statisch	regional	periodisch
Waldbrandfrühwarnsystem	„sense only“	„many-to-one“	statisch	global	eventbasiert
Tierüberwachung	„sense only“	„many-to-one“	mobile Quellen	global	periodisch

3.1.4 Lokalität

Des weiteren kann die Lokalität der verarbeiteten Daten betrachtet werden: Während in globalen Sensornetzwerken ein Zugriff auf alle Knoten gleichzeitig erforderlich ist, ist es in regionalen Netzwerken ausreichend auf die Daten einiger Sensorknoten in einer bestimmen Umgebung Zugang zu haben.

3.1.5 Zeit

Schließlich kann bei der Datenakquisition noch zwischen periodischer Aufzeichnung, bei der Messungen in einem Intervall durchgeführt werden, und eventbasierten Sensornetzwerken, die auf Ereignisse warten und reagieren, unterschieden werden.

Einige Beispiele für die Einordnung verschiedener Anwendungsszenarien in dieses Klassifikationsschema sind in Tabelle 3 zu finden. Weitere Einordnungen sowie ausführlichere Informationen zu diesem Thema sind in [8] beschrieben.

3.2 Lebenszeit

Aufgrund der begrenzten zur Verfügung stehenden Energiemenge kann ein Sensorknoten meist nur eine bestimmte Zeit ohne Wartung arbeiten. In [10] werden Anwendungen anhand der benötigten Lebenszeit des Sensornetzwerks klassifiziert.

Die typischerweise benötigte Lebenszeit bis zum Batterieaustausch für einige Anwendungen ist in Tabelle 4 dargestellt. Diese kann sich in einem großen Bereich bewegen: Während es in klinischen Anwendungen unproblematisch ist, die Batterie täglich durch Pflegepersonal zu wechseln, sollten die oft an weit abgelegenen Orten verteilten Sensorknoten eines Erdbebenfrüherkennungssystems lange ohne teure Eingriffe zuverlässig arbeiten.

Des weiteren wird die Frequenz der periodisch zu messenden Ereignisse betrachtet. Soll beispielsweise ein Sensorknoten nur Luftdruck und Temperatur messen ist es ausreichend dies stündlich oder noch seltener durchzuführen. Für solch eine Anwendung ist entsprechend ein kleiner und sparsamer Sensorknoten ausreichend. Um Vibrationen in einer Halbleiterfabrik oder Geräusche zu überwachen ist dagegen eine hohe Messfrequenz von über 1 kHz notwendig.

3.2.1 Definition Lebenszeit

Wenn Anwendungen nach der Lebenszeit gegliedert werden sollen, stellt sich die Frage was diese genau aussagt. Denn während die Lebenszeit eines einzelnen Sensorknotens direkt als die Zeit definiert werden kann, in der der Knoten seine Aufgabe im Netzwerk erfüllen kann, ist dies für ein ganzes Sensornetzwerk schwieriger. Denn der Ausfall eines Knotens bedeutet meist noch nicht den Ausfall des gesamten Netzes.

Tabelle 4: Anwendungsgebiete nach Lebenszeit [10]

Anwendung	Lebenszeit	Messfrequenz
Wetterüberwachung	Jahre	Sehr niedrig
Erdbebendetektion	Jahrzehnte	Niedrig-Mittel
Herzratenüberwachung	Tage	Mittel
Warenüberwachung	Monate	Niedrig
Vibrationsüberwachung	Monate	Hoch

Hierfür werden in [7] verschiedene Aspekte der Lebenszeit eines Sensornetzwerks identifiziert:

Üblicherweise wird die Lebenszeit ausgehend von der Anzahl noch aktiver Sensorknoten angegeben. Ein Sensornetzwerk ist dann funktionsfähig, wenn mehr als n Knoten lauffähig sind. Diese einfache Metrik ignoriert allerdings einige funktionale Aspekte.

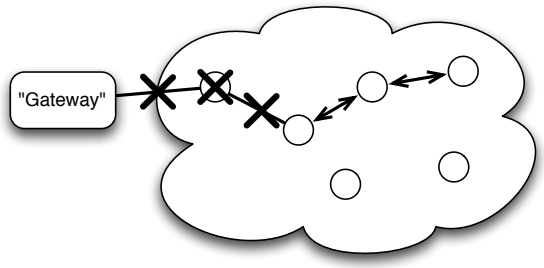


Abbildung 3: Ausfall eines zentralen Knotens

Alternativ kann die Lebenszeit auch über die Abdeckung des Einsatzgebiets oder die Erreichbarkeit des „Gateways“ durch die verbleibenden Sensorknoten angegeben werden. Bei diesen Definitionen wird bereits die Funktionalität des Sensornetzwerks mit einbezogen.

Insbesondere wird nicht jeder Ausfall gleich gewichtet: Fällt ein Knoten aus, der zuvor zentrale Routingaufgaben übernommen hatte, ist dies schwerwiegender als der Ausfall eines Randknotens. Abbildung 3 zeigt solch einen möglichen Extremfall. Da das gesamte Sensornetz nur über einen Knoten mit dem „Gateway“ kommunizieren konnte, führt der Ausfall dieses einen Knotens bereits dazu, dass das Sensornetz seine Aufgabe nicht mehr erfüllen kann.

Als gute Metrik wird schließlich die Angabe der Lebenszeit basierend auf den „Quality-of-Service“-Anforderungen genannt: Die Lebenszeit eines Netzwerks ist die Zeit, in der das Netzwerk durchgehend die Anforderungen der Anwendung erfüllt.

doi: 10.2313/NET-2011-07-1_03

4. TOPOLOGIEN

Die Netzwerktopologie eines Sensornetzwerks gibt an, wie die Knoten untereinander kommunizieren.

4.1 Stern

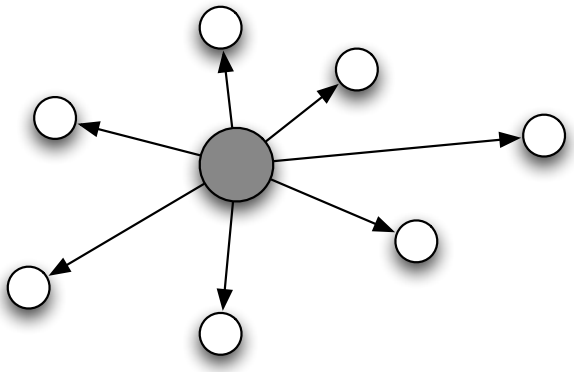


Abbildung 4: Sterntopologie

Die einfachste Topologie ist der Stern. Dabei läuft sämtliche Kommunikation über eine einzige Basisstation [2]. Nur diese kann mit den anderen Knoten kommunizieren, direkte Kommunikation zwischen anderen Knoten ist in einem sternförmigen Netzwerk nicht zulässig.

Durch diesen Aufbau ist die Kommunikation sehr einfach, da jeder Knoten von jedem Knoten mit maximal 2 Hops erreicht werden kann. Bei größeren Netzwerken ergeben sich allerdings einige Probleme. Insbesondere die Reichweite der Drahtloskommunikation limitiert die Einsatzmöglichkeiten. Dabei ist insbesondere der bereits in Abschnitt 2.3 angesprochene exponentielle Zusammenhang zwischen Sendedistanz und Energieverbrauch zu beachten.

Die Beschränkung auf eine einzige Basisstation stellt auch in Bezug auf die Zuverlässigkeit des Netzwerks eine Schwachstelle dar. Fällt diese aus ist im gesamten Netzwerk keine Kommunikation mehr möglich.

4.2 Baum

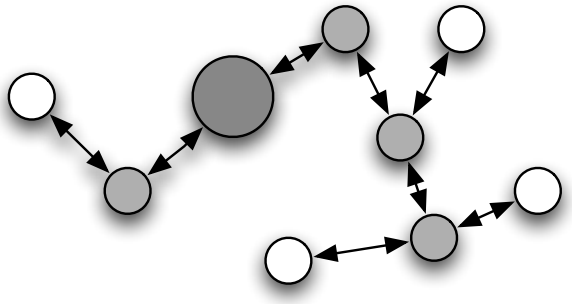


Abbildung 5: Baumtopologie

Eine Erweiterung der Sterntopologie ist die Baumstruktur. Durch zusätzliche Basisstationen kann ein solches Netz auch bei größerer räumlicher Ausdehnung arbeiten.

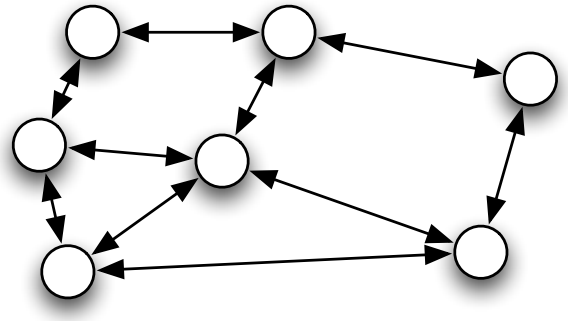


Abbildung 6: Meshtopologie

4.3 Mesh

Das Gegenteil zum zentralen Aufbau eines sternförmigen Netzwerks ist ein Mesh-Netzwerk. Innerhalb eines Meshes können Knoten mit allen anderen in Reichweite liegenden Knoten kommunizieren. Dazu nehmen alle Knoten auch eine Rolle als Router ein und leiten Nachrichten weiter. Durch diesen Aufbau sind Meshnetzwerke sehr störungsresistent. Falls ein Knoten ausfällt, können die Daten über einen anderen Pfad gesendet werden.

Durch die vielfältigen Routingmöglichkeiten wird die Kommunikation allerdings komplexer, was in höherer Latenz und vergrößertem Rechenaufwand resultiert.

4.4 Baum/Mesh-Hybride

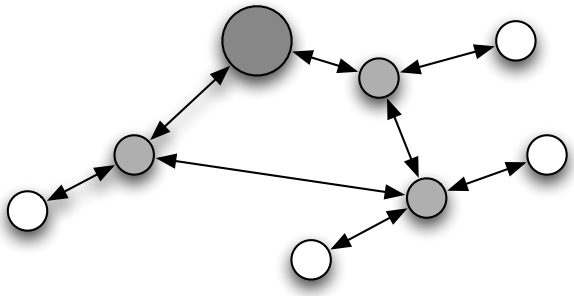


Abbildung 7: Hybride Topologie

Um die Vorteile der einzelnen Topologien zu kombinieren werden in der Praxis meist Hybridsysteme eingesetzt. Ein Beispiel hierfür ist in Abbildung 7 zu sehen. Durch die Kombination einer Baumtopologie mit Meshelementen können so insgesamt bessere Netzwerkeigenschaften erreicht werden.

4.5 Einfluss auf den Energieverbrauch

Je nach Topologie existieren in einem Sensornetzwerk verschiedene Knotentypen. Diese unterscheiden sich vor allem in ihren Kommunikationsaufgaben.

So gibt es in den Baum- und Sterntopologien Endknoten, die nur Daten aufzeichnen. Zentraler gelegene Sensorknoten dagegen übernehmen auch Routingfunktionen. Sie müssen also nicht nur ihre eigenen Messungen übermitteln, sondern auch die anderer Knoten empfangen und weiterleiten.

doi: 10.2313/NET-2011-07-1_03

Daher kann sich der Energieverbrauch der Sensorknoten je nach Lage im Netzwerk unterschieden. Dies ist ein wichtiger Punkt für das Protokolldesign.

So wäre es möglich, für mehr Redudanz bei diesen viel beanspruchten Knoten zu sorgen, so dass falls ein Routingknoten ausfällt Ersatz bereitsteht. Um dies umsetzen zu können muss aber die entsprechende Unterstützung z.B. für die Aktualisierung der Routinginformationen im Protokollstack vorhanden sein.

5. PROTOKOLLDESIGN

5.1 Anforderungen

Aus der Betrachtung der verwendeten Hardware sowie den Anwendungsfällen ergeben sich einige konkrete Anforderungen an das Protokolldesign. In [12] werden diese sowie weitere Ziele und Herausforderungen vorgestellt. Die Wichtigsten sollen an dieser Stelle kurz zusammengefasst werden.

5.1.1 Hardware

Sensorknoten sollen möglichst billig sein und müssen mit einer begrenzten Energiemenge auskommen. Daher steht nur wenig Rechenleistung und Speicher zur Verfügung.

5.1.2 Transceiver

Sensornetze kommunizieren meist drahtlos. Beim Protokolldesign müssen daher die höheren Bitfehlerraten, Latenzzeiten und die schwankende Bandbreite berücksichtigt werden. Ebenso muss robust auf ganz ausgefallene Knoten reagiert werden können. Um den Energieverbrauch gering zu halten, sollten die Abstände zwischen direkt miteinander kommunizierenden Knoten minimiert werden.

5.1.3 Anwendungen

Die vielfältigen Anwendungen für Sensornetze stellen ebenso vielfältige Anforderungen an die verwendeten Protokolle. Diese Anforderungen an die Netzwerkqualität werden in den „Quality of Service“-Anforderungen der Anwendung festgehalten und müssen von dem Protokollstack eingehalten werden.

Über das „Gateway“ soll das Sensornetz mit anderen Netzwerken wie dem Internet integrierbar sein.

5.1.4 Netzwerkaufbau

Sensornetze bestehen aus einer großen Anzahl an Knoten, die mobil sein können und einfach austauschbar sein müssen. Dazu sollte sich das Netzwerk selbst organisieren können und sehr flexibel sein.

Das Netzwerk muss robust gegenüber Störungen sein. Dies beinhaltet Fehlertoleranz sowie Sicherheit gegenüber Angriffen. Da die drahtlose Kommunikation leicht mitschneidbar ist sollten wichtige Informationen verschlüsselt übertragen werden.

Des weiteren ist das richtige Timing für viele Sensorfunktionen kritisch. Daher sollte das Netzwerk einen Zeitsynchronisierungsmechanismus anbieten. Dieser wird beispielsweise für einige Sicherheitsmechanismen sowie zur exakten Zeitmessung von Ereignissen benötigt.

5.1.5 Unterschiede zu klassischen Ad-Hoc-Netzen

Einige dieser Anforderungen werden bereits von allgemeinen Protokollen für drahtlose Ad-Hoc-Netzwerke erfüllt. Es bestehen aber einige gravierende Unterschiede zwischen drahtlosen Sensornetzwerken und Ad-Hoc-Netzwerken [6].

Durch die geringeren Kosten können und werden in Sensornetzwerken deutlich mehr Knoten eingesetzt als in traditionellen Ad-Hoc-Netzwerken. Daher müssen die eingesetzten Techniken gut skalieren. Aus den Energieversorgungsproblemen ergibt sich, dass der Ausfall eines Sensorknotens keine Ausnahme ist und das Netzwerk entsprechend robust darauf reagieren muss. Nicht nur aufgrund von Ausfällen sondern auch durch Knotenmobilität oder manuelle Eingriffe kann sich auch die Topologie des Sensornetzwerks häufig ändern.

Letztendlich ist noch die Interaktionsweise zu beachten. Wie bereits in Abschnitt 3 vorgestellt ist die am häufigsten verwendete Kommunikationsform der Broadcast von Messdaten an das „Gateway“. Dies steht im Gegensatz zu der in traditionellen Rechnernetzwerken meist genutzten Punkt-zu-Punkt-Kommunikation.

5.2 Protokollstack

Um die besonderen Anforderungen drahtloser Sensornetzwerke erfüllen zu können, wird der klassische am ISO/OSI Schichtenmodell orientierte Protokollstack in [6] erweitert. Wie in Abbildung 8 zu sehen sind für Sensornetze zusätzliche Ebenen relevant.

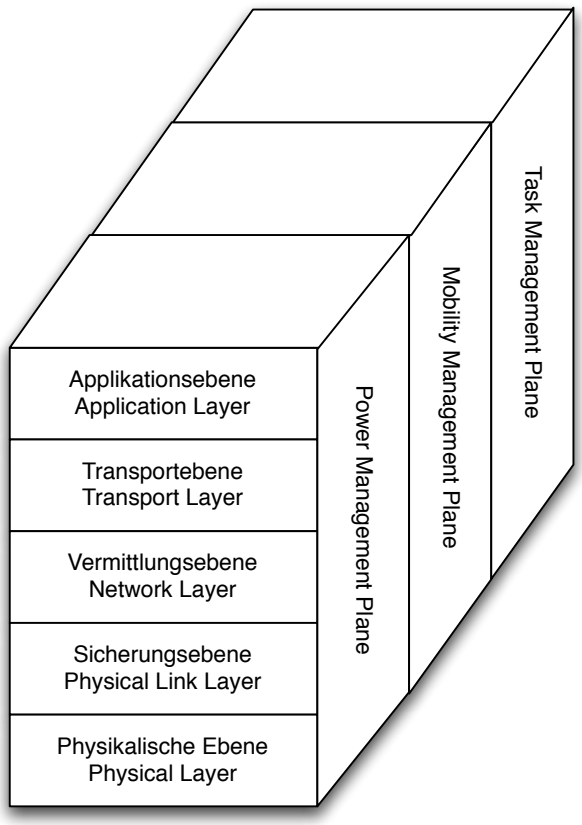


Abbildung 8: Protokollstack für Sensornetze [6]

doi: 10.2313/NET-2011-07-1_03

Um die Energieknappheit zu berücksichtigen wird die „Power Management Plane“ eingeführt. Sie regelt den Energieverbrauch der Sensorknoten und hat zum Ziel, die Lebenszeit des Sensornetzes zu erhöhen. Dies kann beispielsweise geschehen indem Knoten mit wenig verbleibender Laufzeit ihre Routingfunktionalität einschränken und die restliche Energie nur noch für Messungen verwenden.

Die „Mobility Management Plane“ verfolgt die Bewegungen der Knoten und kann dadurch Energieverbrauch und Aufgabenverteilung besser steuern.

Die Verteilung von Aufgaben auf verfügbare Sensorknoten wird durch die „Task Management Plane“ geregelt. In Abbildung 9 ist ein Anwendungsfall hierfür dargestellt. In jedem der vier Sektoren des Einsatzgebiets soll eine Messung durchgeführt werden, es sind allerdings teilweise deutlich mehr als dafür nötige Sensorknoten verfügbar. Daher muss nun die Aufgabe auf die Sensorknoten aufgeteilt werden. Indem nicht benötigte Knoten temporär deaktiviert werden kann Energie gespart und so die Lebenszeit des Gesamtsystems erhöht werden.

Eine mögliche Aufteilung ist mit ausgefüllten Knoten für aktive und leeren Knoten für deaktivierte Knoten in Abbildung 9 zu sehen.

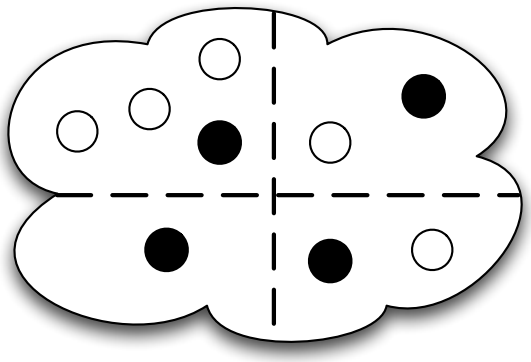


Abbildung 9: Energiesparen durch Abschalten von Knoten

Je nach dem Verhältnis von aktiven zu abgeschalteten Knoten ergeben sich daraus unterschiedliche Lebenszeiten für das Gesamtsystem. Es ist sogar denkbar, das Verhältnis dynamisch nach Bedarf anzupassen. Damit könnten dann genauere Messdaten zur Verfügung stehen wenn sie benötigt werden und in der übrigen Zeit Energie gespart werden.

Bereits an diesem einfachen Beispiel wird klar, dass ein gutes Protokoll viele Eigenschaften eines Sensornetzes beachten muss, um effizient zu arbeiten.

5.3 „Cross-Layer Design“

In Architekturen wie dem ISO/OSI Schichtenmodell wird das Netzwerkdesign in hierarchische Schichten aufgeteilt, die bestimmte Dienste anbieten. Direkte Kommunikation zwischen nicht direkt aneinanderliegenden Schichten ist nicht erlaubt [1].

In Bezug auf Sensornetze bezeichnet „Cross-Layer Design“ hauptsächlich das Aufweichen dieser strikten Schichtentrennung. Dazu werden Mechanismen eingeführt um über mehrere Schichten hinweg Informationen auszutauschen. Insbesondere bei drahtloser Kommunikation können so die vorhandenen Ressourcen besser genutzt werden.

Dies ist besonders für schichtübergreifende Optimierungen hilfreich, wie der in Abschnitt 2.5 vorgestellten.

6. ZUSAMMENFASSUNG

Abschließend lässt sich feststellen, dass eine Vielzahl von Abhängigkeiten zwischen der Anwendung, der verwendetem Hardware und dem Protokolldesign existiert. Zentral für die Planung des Sensornetzes ist dabei die Anwendung. Von dieser ausgehen kann erst die Hardware sowie die verwendeten Protokolle festgelegt werden.

Die Anwendung gibt auch vor, welche dieser Abhängigkeiten für einen konkreten Einsatz tatsächlich relevant sind. Je nach Einsatzzweck kann der Schwerpunkt auf verschiedenen Aspekten liegen.

In jedem Fall ist es essentiell, bei der Planung das Sensornetzwerk in seiner Gesamtheit zu betrachten. Nur so kann ein effizientes Protokoll und damit ein gut funktionierendes Sensornetzwerk entwickelt werden.

7. LITERATUR

[1] V. Srivastava, M. Motani. Cross-layer design: a survey and the road ahead. *IEEE Communications Magazine*, 43(12):112 – 119, 2005.

[2] Chris Townsend, Steven Arms. Wireless Sensor Networks: Principles and Applications, 2004.

[3] Hao Zhou. Energieeffiziente, drahtlose Sensornetze - Auswirkung der Hardware auf das Protokolldesign, 2003.

[4] Heinrich Meyr. Minimizing the total energy consumption in wireless sensor networks, 2011.

[5] Ian Akyildiz Ismail, Ian F. Akyildiz, Ismail H. Kasimoglu. Wireless Sensor and Actor Networks: Research Challenges, 2004.

[6] I.F. Akyildiz, Weilian Su, Y. Sankarasubramaniam, E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102 – 114, 2002.

[7] Isabel Dietrich, Falko Dressler. On the lifetime of wireless sensor networks. *ACM Trans. Sen. Netw.*, 5:5:1–5:39, 2009.

[8] Luca Mottola, Gian Pietro Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv.*, 43:19:1–19:51, 2011.

[9] M.A.M. Vieira, C.N. Coelho Jr., D.C. da Silva Jr., J.M. da Mata. Survey on wireless sensor network devices. In *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*, pages 537 – 544 vol.1, 2003.

[10] Mark Hempstead, Michael J. Lyons, David Brooks, Gu-Yeon Wei. Survey of Hardware Systems for Wireless Sensor Networks. *J. Low Power Electronics*, 4(1):11–20, 2008.

[11] Sanjay Shakkottai, Peter C. Karlsson. Cross-Layer

Design for Wireless Networks. *IEEE Communications Magazine*, 41:74–80, 2003.

[12] V. C. Gungor, G. P. Hancke. Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches. *IEEE Transactions on Industrial Electronics*, 56(10):4258–4265, 2009.

doi: 10.2313/NET-2011-07-1_03

Betriebssysteme für Wireless Sensor Network Motes

Markus Dauberschmidt
Betreuer: Christian Sauter
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: daubersc@in.tum.de

KURZFASSUNG

In dieser Arbeit werden drei Betriebssysteme näher vorgestellt, die vor allem auf „Wireless Sensor Network Motes“ (WSN Mote), also auf Rechenknoten eines Sensor-Netzwerkes eingesetzt werden. Die vorliegende Arbeit geht hier auf die Besonderheiten und Unterschiede dieser Betriebssysteme ein und arbeitet die für WSNs wichtigen Aspekte heraus.

Schlüsselworte

Wireless sensor networks (WSN), Mikrocontroller, TinyOS, Contiki, cocoOS

1. EINLEITUNG

Diese Arbeit befasst sich mit Betriebssystemen für Wireless Sensor Network (WSN) Motes. Im Abschnitt „Wireless Sensor Networks“ wird beschrieben, was ein WSN bzw. ein „Mote“ ist und wofür WSNs eingesetzt werden können.

Im Abschnitt „Betriebssysteme für WSN-Motes“ wird diskutiert, warum überhaupt ein Betriebssystem auf einem WSN Mote sinnvoll ist, und welche Funktionen es bieten sollte.

In den Abschnitten „TinyOS“, „Contiki“ und „cocoOS“ werden die drei genannten Betriebssysteme näher vorgestellt. Dabei werden die jeweiligen Besonderheiten dargelegt, sowie die Aspekte, die besonders für WSNs von Belang sind.

Diese Arbeit schliesst mit einer Zusammenfassung der Besonderheiten der verschiedenen Betriebssysteme.

2. WIRELESS SENSOR NETWORKS

Das Forschungsgebiet der Wireless Sensor Networks, kurz WSNs, hat in den letzten Jahren immens an Bedeutung gewonnen. Das primäre Einsatzgebiet eines WSN ist es, Daten an verschiedenen geograpischen Punkten des Netzes zu erheben, ggf. weiterzuverarbeiten und dann konsolidiert an einen „Data-Sink“ bzw. Root-Knoten des Netzwerkes weiterzureichen [2]. Ein Knoten in einem solchen Netz werden durch „Motes“ gebildet (siehe Abb. 1), kleinen autarken System, bestehend aus einem Mikrocontroller, einem oder mehreren Sensoren, einer i.d.R. autarken Stromversorgung aus einer Batterie, Solarzelle oder einer „Energy Harvesting“ Einheit und einer Möglichkeit zur Datenübertragung und Kommunikation, i.d.R. einem Funksender. Seltener wird auch ein GSM, WLAN oder Ethernet-Modul zur Kommunikation verwendet. Ist es nicht möglich, die Daten weiter zu senden, so sollte der Mote in der Lage sein, die Messergebnisse lokal aufzuzeichnen und später erneut versuchen diese zu übertragen. Was ein Mote messen und übertragen soll, ist so individuell wie vielseitig. Es folgen drei Beispiele für den Einsatz eines WSNs.

Abbildung 1: Ein MICAZ Mote (Quelle: [18])

- In einem Industriekomplex wird eine Fertigungsstraße durch Sensorknoten überwacht. Die Knoten sitzen an neuralgischen Punkten der Anlage und senden regelmäßig Messdaten über Temperatur, Druck und Luftfeuchtigkeit der verschiedenen Fertigungsstationen an eine zentrale Messstation, die diese Daten aufbereitet und der Leitwarte zur Verfügung stellt. Die Sensoren können auch an unzugänglichen Stellen angebracht sein, an denen eine Verkabelung zu aufwändig wäre oder möglicherweise die Fertigungsprozesse stören könnte.

- In einem waldbrandgefährdeten Gebiet werden im Abstand von mehreren hundert Metern Motes an Baumstämmen befestigt. Diese messen die Luftfeuchtigkeit, Windgeschwindigkeit und Umgebungstemperatur und übertragen diese Daten zusammen mit ihrer eigenen Position an eine zentrale Station. Sollte durch große Trockenheit ein Feuer ausbrechen, schlagen die Sensoren Alarm. Mit Hilfe der Sensordaten kann sofort eine Übersicht über den Brandherd in der Zentrale erstellt werden, sowie die Ausbreitungsrichtung also auch die Geschwindigkeit live verfolgt werden (siehe [8]).

- In einem Naturpark in Afrika werden Zebras mit Motes versehen, die regelmäßig ihre aktuelle Position über einen GPS Sensor ermitteln und speichern. Die Daten werden hierbei über Wochen oder gar Monate gesammelt. Ein Austausch der gespeicherten Informationen erfolgt, sobald sich zwei mit Sensoren versehene Tiere

doi: 10.2313/NET-2011-07-1_04

nahe genug kommen. Dieses Projekt wurde unter dem Namen „ZebraNet“ [9] bekannt.

2.1 Mikrocontroller

Ein Mote beinhaltet in der Regel ein Rechensystem auf Basis eines Mikrocontrollers. Ein Mikrocontroller ist eine Recheneinheit, vergleichbar mit der CPU in einem herkömmlichen IT-System, nur ungleich einfacher. Im Allgemeinen sind es, im Gegensatz zu CPUs, keine „All-Purpose“-Recheneinheiten, sondern sind auf das Wesentliche reduziert, um möglichst stromsparend ihre eigentliche Aufgabe zu erledigen. Diese liegt zumeist im Steuer- und Regelbereich und darauf sind Mikrocontroller optimiert. Mikrocontroller finden sich heute überall in eingebetteten Systemen. Egal, ob in der Waschmaschine, in der Kaffeemaschine, im Handy oder der Fernbedienung für den Fernseher: überall arbeiten ein oder mehrere Mikrocontroller im Inneren.

Für die Programmierung von Mikrocontrollern kommen verschiedene Sprachen zum Einsatz: Für sehr echtzeitkritische Aufgaben kann die Programmierung hardwarenah in Assembler erfolgen, in der Regel wird jedoch C, selten auch C++ verwendet. Auch existieren bereits erste Lösungen, die mit den Sprachen des Microsoft .NET Micro Frameworks programmiert werden können ([21]). Auch wenn die Programmierung nicht in Assembler erfolgt, benötigt der Programmierer dennoch genaue Kenntnisse über die verwendete Mikrocontroller-Plattform, deren Möglichkeiten und Verhalten.

Die Kommunikation mit der Umwelt erfolgt über serielle Schnittstellen, USB, Funk, Netzwerkinterfaces oder entsprechende Anzeigeperipherie wie LEDs oder LCDs. Entweder ist die Unterstützung dieser Technologien von den Herstellern in dem jeweiligen Mikrocontroller von Haus aus vorgesehen, oder die entsprechenden Peripheriegeräte werden über einen eigenen Bus (z.B. SPI, i2C) angesprochen.

3. BETRIEBSSYSTEME FÜR WSN-MOTES

Die Knoten eines WSNs besitzen in der Regel einen oder mehrere Sensoren und sind durch Mikrocontroller bzw. daran angeschlossene Peripherie realisiert. Während die ersten WSNs nur statisch Daten sammelten und zu einem zentralen Gateway weiterleiteten, der die Daten dann agreggiert zur Weiterverarbeitung zur Verfügung stellte, erlauben es moderne WSNs auch, die einzelnen Knoten zu steuern, das Routing innerhalb des Netzwerkes zu ändern oder die Knoten zur Laufzeit mit einer neuen Steuersoftware zu versehen. Da WSN aus hunderten von Knoten bestehen können, kommt der Koordination der einzelnen Knoten und dem kontrollierten Abfragen bzw. Einsammeln von Daten eine zentrale Bedeutung zu.

Für die Kommunikation mit der Außenwelt gibt es eine Reihe von Protokollen und Standards. So sind aktuell z.B. IPv6 (implementiert in dem Protokoll „6LoWPAN“ [20]), oder Funk-Technologien wie 802.15.4 oder „ZigBee“ ([1]) ein Gebiet der aktiven Forschung.

Die oben genannten Aspekte machen deutlich, dass für eine effiziente Umsetzung einer Projekt- oder Produktidee sehr viel Breitenwissen rund um das Umfeld, in dem man sich bewegt, vorhanden sein muss. Es ist nicht ausreichend, nur die eigentliche Anwendung zu entwickeln, sondern man muss sich auch mit den Besonderheiten befassen, die die Entwicklung für WSNs nach sich zieht.

Daher ist es wenig verwunderlich, dass in den letzten Jahren, bedingt durch die immer mächtigeren Controller-Generationen, auch erste Betriebssysteme für Mikrocontroller entwickelt worden sind, die dem Entwickler einen Teil dieser Arbeiten abnehmen.

Bei den folgenden Punkten handelt es sich um wesentliche Anforderungen, die an Betriebssysteme im Bereich Wireless Sensor Networks gestellt werden (vgl. auch [16]):

- Das Betriebssystem sollte den Umstand berücksichtigen, dass nur sehr begrenzte Energieressourcen zur Verfügung stehen, Stromsparmodi des Controllers verwenden und nach dem Ausführen des Codes so schnell wie möglich wieder in einen Ruhezustand zu wechseln.
- Es sollte von der zugrundeliegenden Hardwareplattform abstrahieren (HAL=Hardware Abstraction Layer). Code, der für das Betriebssystem auf Plattform A geschrieben wurde, sollte nach Austauschen der HAL auch auf Plattform B verwendbar sein.
- Die Unterstützung für Netzwerkprotokolle sollte direkt im Betriebssystem vorhanden sein oder über Erweiterungen zur Verfügung stehen. Idealerweise unterstützt es Standards wie ZigBee oder TCP/IP und bietet die Nutzung für Anwenderprogramme über entsprechende Schnittstellen (zum Kernel) an.
- Das Betriebssystem sollte möglichst „light-weight“ sein, d.h. es sollte so viele Ressourcen wie möglich dem Anwenderprogramm zur Verfügung halten und selbst möglichst wenig Speicher verbrauchen.
- Mechanismen zur Synchronisation des Zugriffs auf geteilte Ressourcen sollten vom Betriebssystem bereitgestellt werden (z.B. Semaphore, Locks).
- Wichtigere Programmteile sollten gegenüber unwichtigeren Programmteilen bei der Abarbeitung bevorzugt werden (Task-Scheduling, unterschiedliche Schedulingverfahren). Idealerweise unterstützt die eingesetzte Zielplattform bereits hardware-seitig die Prioritätenvergabe, so dass diese nicht in Software nachgebildet werden muss.

In den folgenden drei Abschnitten werden die drei Betriebssysteme TinyOS, Contiki und cocoOS vorgestellt, welche explizit für den Einsatz auf Mikrocontrollern entwickelt wurden und Unterstützung für gängige Anforderungen im Bereich der Wireless Sensor Networks mitbringen.

3.1 TinyOS

TinyOS [30] ist das älteste der drei betrachteten Betriebssysteme. Es ist ein Open Source Betriebssystem und wurde an der Berkeley University von Dr. D. Culler für die dort genutzten WSNs mit Berkeley Motes entwickelt. Inzwischen unterstützt es aber auch etliche weiter Motes wie die Modelle epic, mulle und shimmer2. Es ist zu berücksichtigen,

doi: 10.2313/NET-2011-07-1_04

dass TinyOS standardmäßig *nur* auf den Plattformen eingesetzt werden kann, für die entsprechende Build-Umgebungen existieren! Es ist nicht ohne Weiteres möglich, TinyOS auf einem selbstgebauten Mote einzusetzen, selbst wenn dieser den gleichen Mikrocontroller verwendet, wie ein unterstützter Mote. [13] beschreibt, wie eine Portierung auf eine neue Hardware durchgeführt werden kann. Anwendungen können zunächst auch für den mitgelieferten TOSSIM Emulator compiliert und der Ablauf simuliert werden, bevor die Software produktiv auf die Motes aufgespielt wird.

Es bietet native Unterstützung für 6LoWPAN, dem IPv6 Layer für WSNs.

Die erste Version wurde im Jahr 2006 veröffentlicht. TinyOS ist aktuell in der Version 2.1.1 verfügbar. Auf der TinyOS Homepage steht ein VMWare Image zum Download bereit, in dem alle notwendigen Entwicklungstools bereits vorhanden und vorkonfiguriert sind. Möchte man die Entwicklungsumgebung direkt in ein existierendes System integrieren, so sind hierfür Anleitungen für die Plattformen Windows, Linux und MacOS vorhanden.

TinyOS war ursprünglich in einem C geschrieben, welches jedoch nicht besonders gut für das ereignisgesteuerte Programmiermodell geeignet war [11], das bei TinyOS zum Einsatz kommt. Aus der Notwendigkeit heraus, dieses Programmiermodell bestmöglich zu unterstützten, wurde eine neue Sprache mit dem Namen „nesC“ geschaffen, welche besser für die Programmierung von Sensorknoten geeignet ist [6]. TinyOS wurde daraufhin mit Release der Version 2 nochmals komplett in nesC neu programmiert.

nesC ist eine Sprache, in der die Sprachelemente nicht von Grund auf neu entwickelt wurden, sondern es handelt sich dabei um einen C-Dialekt. Das heisst, dass viele Dinge aus C übernommen wurden und nur spezielle Funktionen für die Besonderheiten in TinyOS (siehe unten) hinzugekommen sind. Man hat sich dagegen entschieden, diese Konzepte rein über Makros in C umzusetzen, da diese nur bedingt durch Compiler optimiert werden können [6]. Da es für nesC aber eigens entwickelte Compiler bzw. Tool-Chains gibt, können zum einen schon zur Compilezeit detaillierte Code-Überprüfungen vorgenommen werden, zum anderen auch viele Optimierungen realisiert werden [6].

TinyOS beinhaltet im Lieferungszustand bereits eine große Anzahl fertig nutzbarer Komponenten (wie z.B. Timer, Scheduler) und bietet auch Simulator-Tools und etliche Beispielanwendungen, die die verschiedenen Konzepte verdeutlichen.

Das Betriebssystem ist dafür gedacht, leichtgewichtige Anwendungen ablaufen zu lassen, wie sie für WSNs üblich sind. Das heisst, es geht um das Erfassen von Daten, ggf. noch deren Aufbereitung und die Weiterleitung der Daten.

TinyOS unterstützt die TI MSP430 (16 Bit) Mikrocontroller Familie[27], sowie die Atmel ATmega128 (8 Bit) [24] und den eher weniger bekannten Intel px28ax Controller. Darüber hinaus werden die gängigen Funkchips wie die Chipcon CC1000 (ISM Band, 433/868 MHz) und CC2420 (2,4 GHz) und Atmel AT86RF212 (868 MHz) und AT86RF230 (2,4 GHz) unterstützt. Wird der Funkchip CC2420 von Texas Instruments eingesetzt, kann TinyOS hier auch die in die Hardware intergrierten AES-Verschlüsselungsfunktionen nutzen, um den Funkverkehr abzusichern [23].

Der von TinyOS bereitgestellte Arbiter schaltet Funkübertragungs-Hardware immer nur für kurze Zeitspannen an, um z.B. neue Pakete zu empfangen, und lässt die Funkgruppen nie dauerhaft aktiviert. Daraus resultiert eine drastische Stromeinsparung, ohne für die Aussenwelt erkennbar zu machen, dass das Gerät nicht dauerhaft auf Empfang ist [29]. Allerdings wird dieser Vorteil mit einer höheren Latenz erkauft.

Zur Zeit-Synchronisation der Motes im Netzwerk verwendet TinyOS das *Flooding Time Synchronization Protocol* [12], welches Synchronisation bis zu Abweichungen im Millisekundenbereich ermöglicht [26].

Das für den Versand von Daten verwendete Protokoll „Collection Tree Protocol“ (CTP) ist sehr unempfindlich gegenüber Störungen und Ausfällen der Infrastruktur, so dass es als sehr robustes Routingprotokoll gilt [7].

Das Aufspielen von neuer Firmware auf Motes zur Laufzeit ist mit Hilfe des *Deluge* Protokolls möglich. Hierbei wird ein Komplettimage zu dem Mote übertragen, der mit einer neuen Software versehen werden soll. Der selektive Austausch nur einiger Komponenten eines Knotens, der unter TinyOS läuft, ist nicht möglich, da es keine Trennung zwischen Betriebssystem und Anwenderprogramm gibt.

Für die Datenverteilung zu den Knoten unterstützt TinyOS die Protokolle Drip, DIP und DHV. Bei Neuentwicklungen sollte dem modernen Protokoll DHV der Vorzug gegeben werden, da dieses die Verteilung am effizientesten gestaltet [25].

Programme in TinyOS bestehen aus *Components*, welche vom Typ *Module* oder *Configuration* sein können.

Eine *Component* in TinyOS besteht wiederum aus zwei Teilen: einem *interface* Teil und einem *implementation* Teil. Komponenten stellen selbst Interfaces bereit und nutzen externe Interfaces.

In *Modules* wird die Implementierung eines Interfaces vorgenommen, während in *Configurations* die Verbindung zwischen Komponenten und deren Interfaces definiert wird. Jede nesC Anwendung besteht aus wenigstens einer *Configuration* Komponente, die die einzelnen Teile der Anwendung beschreibt.

Die einzelnen *Components* werden dann vom TinyOS Scheduler ereignisgesteuert stanardmäßig nach dem FIFO-Prinzip aufgerufen. Die zentrale Motivation dieses Ansatzes ist, das System so schnell wie möglich wieder in den Schlafmodus zu schicken, um dadurch Energie zu sparen. Der Schedulingmechanismus lässt sich jedoch auch austauschen [14], um z.B. einen prioritäts-basierten Scheduler zu verwenden. Idealerweise bietet die unterstützte Mikrocontroller-Plattform hardware-seitig auch direkt unterschiedliche Prioritäten für Interrupts an. Diese Anforderung ist sowohl bei den MSP430 als auch den Atmel AVR Mikrocontrollern erfüllt, allerdings

doi: 10.2313/NET-2011-07-1_04

werden nur Hardware-Interrupts wie Timer, ADC, UART, ermöglicht. Diese Prioritäten sind auch fest vom Mikrocontroller vorgegeben und können nicht vom Entwickler geändert werden. Die Verwendung von vom Entwickler definierten „Software-Interrupts“ ist nicht möglich. Ein „Vectored Interrupt Controller“, bei dem den einzelnen Interrupts definierte Prioritäten zugewiesen werden können, existiert beispielsweise beim ARM Cortex M3 Mikrocontroller. Mit dem „Egs“ Mote existiert eine Plattform auf ARM Cortex M3 Basis, für die TinyOS bereits portiert wurde ([10]).

Das TinyOS Entwicklungssystem besteht aus Kommandozeilen-Tools, mit denen die Programme compiliert werden und auf die verschiedenen Motes übertragen werden. TinyOS bringt hierfür ein eigenes Buildsystem mit.

Im Folgenden werden diese Konzepte anhand der Beispielanwendung „BlinkC“ vorgestellt. Diese Anwendung lässt eine LED eines Motes mit einer festen Frequenz blinken und verdeutlicht die Nutzung der Timer, die das Betriebssystem abstrahiert zur Verfügung stellt. Die Anwendung besteht aus einer *Configuration* Komponente und einer *Module* Komponente.

Zunächst zur *Configuration* Komponente

```
1  configuration BlinkAppC {
   }
3
   implementation {
5    components MainC, BlinkC, LedsC;
     components new TimerMilliC() as Timer0;
7
     BlinkC -> MainC.Boot;
9    BlinkC.Timer0 -> Timer0;
     BlinkC.Leds -> LedsC;
11 }
```

Zeile 1-2 definiert diese Komponente als *configuration*. Die ersten zwei Zeilen im Abschnitt „implementation“ definieren die zu nutzenden Interfaces anderer Komponenten, namentlich `MainC`, `BlinkC` und `LedsC`. MainC und LedsC sind Dateien des TinyOS Betriebssystems, BlinkC ist unsere eigene Komponente. Desweiteren wird ein Timer mit den Namen Timer0 im System bekannt gemacht. Die letzten drei Zeilen der Datei verbinden die Elemente unserer Komponente BlinkC mit den anderen angeführten Komponenten.

Die Datei BlinkC, implementiert als *Module* die eigentliche Anwendung und ist wie folgt aufgebaut:

```
1  #include "Timer.h"

3  module BlinkC @safe() {
     uses interface Timer<TMilli> as Timer0;
5    uses interface Leds;
     uses interface Boot;
7  }

9  implementation {
     event void Boot.booted() {
11     call Timer0.startPeriodic( 250 );
     }
13   event void Timer0.fired() {
       call Leds.led0Toggle();
15   }
   }
```

Das Schlüsselwort „module“ zu Beginn der Datei deklariert diese Komponente als *Module*. Durch den Zusatz „@safe“ wird der Compiler angewiesen, zusätzliche Checks in den Programmcode einzubauen. Dadurch ist es möglich, z.B. Zugriffe ausserhalb der definierten Grenzen eines Arrays zu erkennen und das Programm kontrolliert abstürzen zu lassen, anstatt dass es zum ungewollten Überschreiben von Speicherbereichen kommt. Die „uses“ Schlüsselwörter in den folgenden 3 Zeilen besagen, dass diese Komponente die genannten 3 Komponenten nutzt. Die Semantik ist vergleichbar zu dem „implements“ Schlüsselwort in Java. Dadurch, dass unsere Komponente diese Interfaces nutzen will, muss sie zeitgleich auch alle geforderten Events implementieren. In unserem Beispiel ist dies das „fired()“ Event der genutzten Timer-Komponente. Der Rest des Programms ist selbsterklärend und illustriert die einfache Art wie in TinyOS ein Timer konfiguriert und eine LED umgeschaltet werden kann, ohne konkrete Kenntnis über die zugrundeliegende Hardware zu haben. Der Programmierer spricht lediglich „Led0“ an, muss aber nicht wissen, an welchem Port des Mikrocontrollers diese hängt. Die tatsächliche Zuordnung der Alias-Namen zur Hardware geschieht in den Header-Dateien von TinyOS für die jeweilige Zielplattform. Das „call“ Schlüsselwort ist nur notwendig, wenn Funktionen von Interfaces aufgerufen werden sollen. Der Aufruf von Funktionen innerhalb der Komponente ist, wie bei C üblich, ohne weiteren Aufwand direkt durch Angabe des Funktionsnamens möglich.

Ein Blick in die TinyOS System-Dateien bestätigt den erwarteten Aufbau:

Beispiel: Timer.nc

```
   tos/lib/timer/Timer.nc:
2  interface Timer {
     // basic interface
4    command void startPeriodic( uint32_t dt );
     command void startOneShot( uint32_t dt );
6    command void stop();
     event void fired();
8    ...
   }
```

Die drei „command“ Definitionen sorgen dafür, dass die Methoden `startPeriodic`, `startOneShot` und `stop` auf dem Interface ausgeführt werden können. Um das Interface auch nutzen zu können, muss ein Handler für das Event „fired“ implementiert werden, was in der Datei BlinkC.nc geschehen ist.

Das letzte Beispiel zeigte bereits ein Programm, welches asynchrone Aufrufe nutzte. Natürlich können in TinyOS auch synchrone Aufrufe verwendet werden, aber davon wird abgeraten, da synchroner Code immer ohne Unterbrechung ausgeführt wird und dies mitunter dazu führt, dass das System nicht mehr zeitnah reagieren kann.

Stattdessen macht man sich in TinyOS sogenannte „Split-

doi: 10.2313/NET-2011-07-1_04

Phase“ Operationen zu nutze, wo das Absetzen eines Befehls von dessen Erledigungsmeldung entkoppelt ist.

Um diese asynchronen Arbeiten auszuführen, hat TinyOS das Konzept der „Tasks“ vorgesehen. Tasks werden nicht sofort ausgeführt, sondern erst, wenn das System dafür Zeit hat.

Ein Task wird in TinyOS durch das Schlüsselwort „task“ definiert. Tasks haben einen „void“ Rückgabewert und nehmen keine Parameter entgegen. Dies erinnert an die Art und Weise, wie Threads in Java definiert werden. Auch hier gibt es eine `run`-Methode die implementiert werden muss, die aber keinen Rückgabewert gibt und keine Argumente nimmt.

Unser Beispiel kann wie folgt umgebaut werden:

```
task void computeTask() {
  uint32_t i;
  for (i = 0; i < 400001; i++) {}
}

event void Timer0.fired() {
  call Leds.led0Toggle();
  post computeTask();
}
```

Um einen Task zu starten wird der Aufruf „`post taskname()`“ verwendet.

Für das Task-Management verwendet TinyOS intern eine FIFO-Queue. Jeder Task wird durch ein Status-Byte auf dem Stack beschrieben. Die Länge ist auf 255 Tasks begrenzt. Wird ein Task aus der Queue gestartet, so läuft dieser ohne Unterbrechung bis zum Ende. Aus diesem Grund sollten Tasks nicht zu aufwändig sein, sondern sich wiederum ggf. in Subtasks splitten, falls größere Berechnungen durchgeführt werden müssen. Die Verwendung einer FIFO-Queue bedingt, dass alle Tasks gleichberechtigt sind und ihre Ausführungsreihenfolge ausschließlich von der Position innerhalb der Queue abhängt. Es ist nicht möglich, Tasks mit Prioritäten zu versehen und dadurch eine bevorzugte Abarbeitung zu erzwingen. Der Scheduler ist allerdings selbst eine TinyOS Komponente und kann als solche ersetzt bzw. angepasst werden um z.B. eine andere Scheduling Policy zu ermöglichen. Das Einhalten gewisser Vorgaben, wie beispielsweise „Fairness“ zur Verhinderung des „Verhungerns“ eines Tasks, sind vom Scheduler selbst sicherzustellen.

3.1.1 Persistente Speicherung von Daten

TinyOS abstrahiert auch das Einbinden von persistentem Speicher. In der Regel besitzen die verschiedenen Motes zusätzlichen On-board Speicher in Form von Flash-Speicher oder auch ein EEPROM. TinyOS nutzt plattform-spezifische Implementierungen der Komponenten `ConfigStorageC`, `LogStorageC` und `BlockStorageC`, die den Zugriff auf diese Komponenten abstrahieren [28, 3].

Der zur Verfügung stehende Flashspeicher kann von TinyOS in einzelne (logische) Volumes separiert werden, ähnlich den Partitionen einer Festplatte.

Das Konzept von Dateien ist TinyOS nicht bekannt. Es gibt kein Dateisystem im üblichen Sinne. Logdaten werden je nach Implementierung entweder sequentiell als Abfolge von sog. Records geschrieben oder über einen Ringbuffer persistiert. Beide Varianten werden von TinyOS in Form der Komponente „LogRead/LogWrite“ abstrahiert.

3.1.2 Ressourcenarbitrierung

Ressourcen in TinyOS können einer von drei Kategorien angehören: *dedicated*, *virtualized* oder *shared*. Die Kategorie bestimmt die Art und Weise, wie auf diese Ressource (etwa das Funksystem) zugegriffen werden kann. Eine `dedicated` Ressource steht der nutzenden Komponente exklusiv zur Verfügung, während `virtualized` Ressourcen die gleichzeitige Nutzung durch mehrere Komponenten ermöglichen: Jeder Client nutzt die Ressource, als würde sie ihm exklusiv zur Verfügung stehen, tatsächlich wird der Zugriff aller Komponenten durch das TinyOS gemultiplexed. Virtualisierte Ressourcen stellen den Komponenten keine Möglichkeit zur Verfügung, den Powerstate der Ressource zu kontrollieren, während `dedicated` Komponenten dies anbieten. `Shared` Ressourcen wiederum setzen auf `dedicated` Ressourcen auf, wobei der Zugriff über einen Arbiter reglementiert wird. Fragen mehrere Komponenten eine Ressource an, so ist es Aufgabe des Arbiters, diese Ressource entsprechend den aufrufenden Komponenten zur Verfügung zu stellen. Sobald ein Client eine Ressource im Zugriff hat, wird vorausgesetzt, dass er diese selbst wieder nach Gebrauch zurückgibt („Cooperative behaviour“). Ein Arbiter kann selbst keine Ressourcen entziehen - nur zuteilen. Der Arbiter kümmert sich mit Hilfe eines PowerManagers auch um die Stromversorgung der geteilten Ressourcen.

Neben den Power-State-Änderungen des Powermanagers versucht TinyOS natürlich auch die Stromspar-Funktionen der zugrundeliegenden Hardware zu nutzen. Jedesmal, wenn die Task-Queue leer ist, wird der bestmögliche Strom-Spar-Zustand ermittelt und der Controller in diesen Zustand versetzt, bis er durch ein externes Ereignis (z.B. das Empfangen eines Pakets über die Funkgruppe) wieder geweckt wird.

3.1.3 Datensammlung und -Verteilung

TinyOS beinhaltet im Lieferumfang bereits *Collector* und *Disseminator* Komponenten, die den Anwendungsentwickler dabei unterstützen, Daten im WSN zu verteilen (disseminate), bzw. einzusammeln (collect). Der Aufbau eines Routing-Trees geschieht dabei vollautomatisch im Hintergrund.

Die Auswahl des Dissemination-Verfahrens (TinyOS unterstützt die Verfahren Drip, DIP und DHV, die sich hinsichtlich ihrer Effizienz unterscheiden), geschieht erst auf Ebene des Makefiles. Rein vom Code sind zur Nutzung alle drei Verfahren (fast) identisch. Weitere Verfahren die TinyOS anbietet, sind etwa das Routingprotocol DYMO, mit dem on-demand Routen zwischen den Motes ermittelt werden können.

Um große Daten zu übertragen - das schließt auch Update-Images für die Motes selbst ein - kann das *Deluge T2* Protokoll verwendet werden. Damit ist es möglich, eine neue Firmware Version an einer zentralen Stelle in das Netzwerk einzuspielen und diese mit Hilfe des Protokolls auf alle Motes verteilen zu lassen [17]. Mit Deluge T2 ist es allerdings nur möglich, den gesamten Mote zu aktualisieren.

doi: 10.2313/NET-2011-07-1_04

Mit *blib* gibt es auch eine 6lowpan/IPv6 Implementierung in TinyOS. Der TCP Stack ist jedoch noch immer im experimentellen Stadium und es werden noch nicht alle Mote Typen unterstützt. Mit „nwprog“ gibt es auch hier eine Möglichkeit, die Motes über ein IP Netzwerk neu zu programmieren (via Deluge).

3.1.4 Energiesparen

TinyOS optimiert das Abschalten zur Zeit nicht genutzter Hardware-Komponenten selbständig. Für das Funkmodul steht die Funktionsart *LPL* (Low Power Listening) zur Verfügung, welches die Funkgruppe nur für die absolut notwendige Zeit aktiviert, um zu prüfen, ob ein Paket übertragen wird, welches für den aktuellen Mote bestimmt ist. Welcher Powersave-Mode des Mikrocontrollers aktuell am Besten geeignet ist, ermittelt TinyOS anhand der Status- und Controlregister, welche chip-spezifisch ausgewertet werden, einem „dirty bit“, welches die Notwendigkeit einer neuen Power-Save-Berechnung signalisiert und einem „Power-State override“. Die Berechnungen finden innerhalb der TinyOS core scheduling loop statt. Finden Ereignisse statt, die zu einer Veränderung des Powersave-Modes führen können, wird dies durch das Dirtybit signalisiert und der Zustand daraufhin reevaluiert.

3.1.5 TOSThreads

Ab TinyOS 2.1 können nun auch präemptive Threads mithilfe der TOSThreads Bibliothek erstellt werden [22]. Der Kernel selbst läuft als hoch-priorer Thread. Eine Besonderheit ist, dass die TOSThreads Library es gestattet, die Anwendung in reinem C zu schreiben und kein nesC verwenden zu müssen. Dies ist möglich, da TOSThreads einen Wrapper über die Split-Phase-Operationen legt. Die Anwendung wird dennoch effizient in TinyOS ausgeführt. Threads bieten `start()`, `stop()`, `pause()`, `resume()`, `sleep()`, `run()` und `join()` Funktionen, wie sie aus z.B. POSIX kompatiblen OS bekannt sind. Desweiteren werden auch die folgenden Synchronisations-Primitive unterstützt: Mutex, Semaphore, Barrier, Condition variable, Blocking reference counter.

3.2 Contiki

Contiki sieht sich in direkter Konkurrenz zu TinyOS und will vieles besser machen. Als ein Vorteil wird von den Entwicklern angeführt, dass es bei der Nutzung von Contiki nicht notwendig ist, einen neuen C-Dialekt (nesC) wie bei TinyOS zu lernen. An der Contiki Entwicklung sind u.a Firmen wie SAP, Cisco, Atmel, sowie die TU München beteiligt.

Wie bei TinyOS gibt es auch hier ein VM Image, das alles enthält, was man für die Entwicklung benötigt. Desweiteren gibt es einen Simulator namens Cooja, der genutzt werden kann, wenn keine echten Motes verfügbar sind, und der z.B. auch graphisch die Kommunikationsbeziehungen zwischen Teilnehmern eines WSNs darstellen kann. Die Hardware, die von Contiki unterstützt wird, ist die gleiche, die auch von TinyOS unterstützt wird. Wie bereits im Abschnitt über TinyOS erwähnt, kann Contiki aber nur auf Geräten eingesetzt werden, für die ein entsprechender Port existiert. Contiki wurde bereits für eine ganze Reihe von Systemen portiert und ist keinesfalls nur auf Mikrocontroller beschränkt. So gibt es Ports für den C64, Apple II, Atari ST, Gameboy, Playstation und viele andere Klassiker der Computerära der 80er und 90er Jahre.

3.2.1 Programmiermodell

Als Ablaufmodell kommt bei Contiki ebenso wie bei TinyOS ein eventbasiertes Modell zum Einsatz, da dieses den Memory-Overhead für echte Multi-Threading Systeme vermeidet. Bei Event-basierten Systemen ist es nicht notwendig, Stackspeicher für jeden Thread zu reservieren, was den eingeschränkten Ressourcen von Mikrocontrollern zu Gute kommt.

Ein großer Vorteil gegenüber TinyOS ist bei Contiki die Möglichkeit, Applikationen zur Laufzeit zu laden bzw. zu entladen. Contiki bietet auch Features, die nur von größeren Betriebssystemen bekannt sind, wie Interprozesskommunikation (IPC) via „Message Passing“ und Multithreading innerhalb einer Anwendung. Realisiert wird dies in Contiki über sogenannte „Protothreads“. Hierbei handelt es sich um eine sehr leichtgewichtige Thread-Implementierung, die für die Datenspeicherung über den Context-Wechsel hinweg keinen Stack, sondern globale Variablen nutzt. Pro Thread fallen dafür nur 2 Byte Speicher für die Verwaltung an (vgl. [5]).

Hauptmotivation für die Entwicklung der Protothreads war die Tatsache, dass eventbasierte Programmierung voraussetzt, dass die Anwendungen wie Zustandsmaschinen aufgebaut sind. Es kann aber, je nach Anwendung, alles andere als trivial sein, die gewünschte Logik korrekt als Zustandsmaschine abzubilden und erfordert oft ein Umdenken der Entwickler. Da solche Zustandsmaschinen oftmals ohne eingehende Vorabplanung umgesetzt werden, kommt es hierbei zu höheren Aufwänden beim Testen und Debuggen. Protothreads vereinfachen die Umsetzung, da sie nach außen hin keine Zustandsmaschinen zu nutzen scheinen. Tatsächlich wird intern der Kontrollfluss jedoch auf Schritte einer Zustandsmaschine abgebildet. Protothreads-Programme sind zumeist auch kompakter als entsprechende Statemachine-Implementierungen. Detaillierte Informationen zur Implementierung von Protothreads findet sich in [5].

Ein Protothread wird über das Makro „`PT_THREAD`“ definiert und enthält den Code zwischen den beiden Markern „`PT_BEGIN`“ und „`PT_END`“. Zentrales Element ist der Befehl „`PT_WAIT_UNTIL`“, welches eine Bedingung übergeben bekommt und den Thread so lange blockiert, bis die Auswertung der Bedingung zu „true“ erfolgt. Protothreads können *nur* an diesen Stellen blockieren und die Kontrolle abgeben, andere Möglichkeiten gibt es nicht. Dies erleichtert aber auch gleichzeitig dem Programmierer die Fehlersuche, da die Punkte im Code, an denen blockiert wird, sofort erkennbar sind.

Um dieses Konzept zu verdeutlichen, wird im folgenden Codeausschnitt das im Kapitel „TinyOS“ vorgestellte Beispielprogramm mit Protothreads für Contiki umgeschrieben.

```
1 #include "pt.h"

3 struct pt pt;
  struct timer timer;
5
  PT_THREAD(example(struct pt *pt))
7 {
    PT_BEGIN(pt);
9
```


doi: 10.2313/NET-2011-07-1_04

```
    while(1) {
11      timer_start(&timer);
        PT_WAIT_UNTIL(pt,
13          timer_expired(&timer));
        call_toggle_led();
15
    }
17  PT_END(pt);
}
```

3.2.2 Persistente Speicherung von Daten

Für das persistente Schreiben von Daten bietet Contiki das „Coffee“ Filesystem, welches Funktionen zum Lesen, Schreiben, Positionieren (Seek), Anhängen und Löschen von Dateien bietet. Ebenso werden auch Verzeichnisse unterstützt. Im Gegensatz zu TinyOS hat der Entwickler hier die Möglichkeit, „echte“ Dateien und Verzeichnisse zu erzeugen. Contiki abstrahiert die Zugriffe hierbei von der eigentlichen Ressource, d.h. prinzipiell kann als Schreibziel jedes Medium verwendet werden, für das ein entsprechender Treiber in Contiki existiert (Netzwerkshare, SD-Karte, ...)

3.2.3 Netzwerkstack

Contiki kommt mit zwei fertigen Kommunikations-Stacks: „uIP“ und „Rime“. Bei uIP handelt es sich um eine vollwertige Implementierung eines TCP/IP Stacks, welcher speziell für die begrenzten Betriebsmittel eines Mikrocontrollers entwickelt wurde. uIP bietet eine API an, die an die Posix Sockets angelehnt ist und u.a. Funktionen zum DNS Lookup bietet, sowie die Protokolle IP, IPv6, UDP, TCP, ARP und 6lowpan unterstützt. „Rime“ ist der Versuch, ein einheitliches Protokoll zu implementieren, welches von der zugrundeliegenden Hardware abstrahiert und dem Anwendungsentwickler die Mühe abnimmt, für jede Funktechnologie, jedes Funk-Hardwaremodul oder jedes Übertragungsprotokoll seine Anwendung erneut anpassen zu müssen.

Als Media Access Control (MAC) kommt das CSMA/CD ([15]) Verfahren zum Einsatz. Um den geringen Ressourcen von Motes Rechnung zu tragen, können verschiedene „Radio Duty Cycling Layer“ (RDCs) ausgewählt werden, die zur Compilezeit gebunden werden. Implementierungen, die die Funkgruppe des Mote nur alle paar hundert Millisekunden aktivieren, um zu prüfen, ob neue Daten vorliegen, verbrauchen gegenüber einem Mote, dessen Funkgruppe permanent aktiviert ist, natürlich weniger Strom. Erkauft wird dies dann mit einer größeren Latenz, bis der Mote auf eine Nachricht reagiert (vgl. auch Abschnitt TinyOS).

3.2.4 Speicher-Management

Contiki nutzt einen Speichermanager, der genutzt werden kann, um fragmentierungsfrei dynamisch Speicher zu allokieren. Dieser verschiebt freigegebene bzw. genutzte Speicherbereiche bei jeder Änderung, so dass sich möglichst große zusammenhängende Flächen ergeben und einer Fragmentierung entgegen gewirkt wird.

Dies wiederum setzt allerdings voraus, dass Anwendungen nie direkt auf den Speicher zugreifen, sondern über einen Umweg über den Contiki Speichermanager, damit ein Zugriff auf einen vormals allokierten Speicherbereich nicht in einem inzwischen anderweitig genutzten Bereich endet.

3.2.5 Laden und Entladen von Anwendungen

Contiki nutzt ein DLL (Dynamic linking and loading) Laufzeitsystem, um Applikationen dynamisch zu laden und zu entladen. Da WSN-Knoten häufig im Produktiveinsatz an unzugänglichen Stellen platziert werden (vgl. die verschiedenen Szenarien in Abschnitt 1), wird die Fähigkeit benötigt, Teile des Systems zur Laufzeit durch „over-the-air“ Programmierung zu aktualisieren, da ein Abbau des Sensors und der Anschluss an eine Programmierstation in der Regel zu aufwändig ist. Software-Updates können hier einerseits Fehlerkorrekturen an den vorhandenen Anwendungen sein, es können aber auch zusätzliche Anwendungen eingespielt werden. Contiki setzt für die Binaries das ELF Objektformat ein, welches z.B. im Linux Umfeld weit verbreitet ist.

Im Gegensatz zu TinyOS geht Contiki bei den „over-the-air“ Updates einen anderen Weg und erlaubt das selektive Updaten bzw. Hinzufügen von Applikationen. Dies ist sinnvoll, denn das Ersetzen des kompletten Firmware-Images ist nur selten notwendig und wirkt sich negativ auf den Stromverbrauch und die Dauer des Updates aus.

Ein Full-System-Replacement ist relativ einfach durchzuführen, da keine Vorverarbeitung des Images auf dem Zielknoten stattfinden muss. Alle im Code enthaltenen Adressen sind absolut und das Image wird einfach in den Flashspeicher eingespielt und überschreibt den Speicher komplett. Der Ansatz in Contiki, ein dynamisches Linken von Modulen einzuführen, war relativ neu, da der mit dem Linken verbundene Berechnungsaufwand bis zu diesem Zeitpunkt als zu aufwändig und damit für WSN Motes als unwirtschaftlich angesehen wurde. Beim dynamischen Linken werden alle in dem Modul deklarierten Funktionsaufrufe und Symbole (z.B. aus dem Kernel) aufgelöst. Erst, wenn alle Referenzen erfolgreich verknüpft worden sind, ist das Modul nutzbar. Funktionsaufrufe müssen dabei nicht nur Aufrufe aus anderen Modulen oder dem Kernel sein, sondern sind auch die Aufrufe im eigenen Modul: Da das Modul dynamisch in den Speicher geladen wird, sind alle im Code verwendeten Adressen Offsets oder Alias-Adressen, die erst beim Laden auf die physikalischen Speicheradressen gesetzt werden. Diesen Vorgang nennt man „Relocation“.

Dies ist das gleiche Verhalten, wie es DLLs in der Windows-Welt oder SO (Shared Objects) in Linux Betriebssystemen aufweisen.

Objektdateien im ELF Format beinhalten unter anderem den eigentlichen Programm-Code, Symboltabellen und Relokierungstabellen, also alle Informationen, die für ein dynamisches Laden des Programmstückes notwendig sind. Ein großer Nachteil des ELF-Formats ist jedoch, dass es den Anspruch hat, sowohl unter 32 Bit als auch 64 Bit Betriebssystemen ohne Änderungen genutzt werden zu können. Daher werden auch grundsätzlich nur 64 Bit kompatible Datentypen verwendet, was bedeutet, dass die ELF-Dateien mehr Speicherplatz in Anspruch nehmen, als auf einem Mote nötig wäre, was wiederum zu längeren Übertragungszeiten und größerem Speicherplatzbedarf führt. Daher hat man das CELF Format („Compressed ELF“) eingeführt, welches diese Kompatibilität wieder entfernt, und native 8 und 16 Bit Datentypen unterstützt. Die CELF Datei ist im Schnitt nur noch halb so groß wie die normale ELF Datei und wird

doi: 10.2313/NET-2011-07-1_04

automatisch während des Compiler-Vorgangs erzeugt.

Ein Contiki System besteht aus dem Core Bereich und dem Bereich für die ladbaren Module. Der Core Bereich ist im Wesentlichen das, was man den Kernel nennt. Er bietet Datenstrukturen und die C-Bibliotheksfunktionen an. Ein ladbares Modul kann die Funktionen des Kernels aufrufen und darüber auch mit anderen Modulen Daten austauschen. Diese Teilung macht es möglich, nur die ladbare Applikation auszutauschen und den Kernel unangetastet zu lassen. Darüber hinaus ist es möglich auch den Core zur Laufzeit auszutauschen, jedoch wird von diesem Feature eher selten Gebrauch gemacht.

3.3 cocoOS

Das dritte Betriebssystem, welches im Rahmen dieser Ausarbeitung betrachtet wird, ist cocoOS [19]. cocoOS erfährt aktuell die meisten Änderungen, ist aber auch ein recht neues Projekt, welches erst 2010 gestartet wurde.

Im Vergleich zu TinyOS und Contiki fällt sofort auf, dass cocoOS noch in den Kinderschuhen steckt. Bislang sind lediglich OS Primitive wie Tasks, Events, Messages und Semaphore realisiert. Aktualisierungen der Firmware über Funk, Strom-Spar-Modi, IP stacks oder ähnliches sind für dieses Betriebssystem noch Fremdwörter.

cocoOS ist derzeit nur für die AVR Plattform verfügbar, ein Port für MSP430 ist jedoch ebenfalls vorgesehen.

Wie auch bei TinyOS und Contiki OS wird bei cocoOS das Prinzip eines kooperatives Multitasking eingesetzt. Die „Task procedures“ werden stets komplett ausgeführt. Es ist keine Präemption vorgesehen.

Die bisher vorhandenen Primitive sind sehr einfach zu nutzen.

3.3.1 Tasks

Bis zu 16 Tasks können im System vorhanden sein. Mittels der API Funktion `task_create` kann ein neuer Task dem System hinzugefügt werden. Beim Erzeugen eines Tasks wird ein Pointer auf die Taskprozedur übergeben, es kann eine Priorität vergeben werden und eine MessageQueue für den entsprechenden Task.

Die Taskroutine selbst wird durch zwei Makros eingekleidet, ähnlich wie bei ContikiOS:

```
void task(void) {
    task_open();
    ...
    task_close();
}
```

Auf äußere Ereignisse kann mittels des Aufrufes `event_wait` gewartet werden. Das Task blockiert bis zum Eintreffen des Events und ein anderer Task wird vom cocoOS Scheduler als aktiver Task ausgewählt.

Ein Beispiel für einen Task, welcher endlos ausgeführt wird und auf ein Ereignis von der seriellen Schnittstelle wartet, ist im folgenden Listing gegeben:

```
void task(void) {
    uint8_t data;
    task_open();
    for (;;) {
        event_wait(rxEvt);
        uart_get(&data);
        ...
    }
    task_close();
}
```

3.3.2 Events

Im obigen Beispiel kam bereits ein Aufruf von `event_wait` vor. Mittels dieses API Calls können Tasks auf das Eintreffen eines Ereignisses warten. Zunächst muss jedoch ein Event mittels der Funktion `event_create` erzeugt worden sein. Im Code selbst kann dann mittels `event_signal` oder, falls man sich in einer Interrupt Service Routine (ISR) befindet, mit `event_ISR_signal` das entsprechende Event signalisiert werden. Das Warten auf mehrere Events (Synchronisationsprimitiv „Barriere“) ist ebenfalls möglich und zwar über die Funktion `event_wait_multiple`.

Unser bereits bekanntes „Blinkprogramm“ könnte in cocoOS wie folgt implementiert werden:

```
Evt_t timerExpiredEvent;

void main(void) {
    system_init();
    os_init();
    timerExpiredEvent = event_create();
    task_create(task,1,NULL,0,0);
    setup_timer();
    clock_start();
    os_start();
    return 0;
}
void task(void) {
    uint8_t data;
    task_open();
    for (;;) {
        event_wait(timerExpiredEvent);
        toggle_led();
        ...
    }
    task_close();
}
```

3.3.3 Messages

Sollen mehr als einfache Signale verschickt werden, können „Messages“ verwendet werden. Messages sind C structs beliebigen Inhalts, welche zwischen Tasks versendet werden können und in einer Message-Queue gepuffert werden (siehe API Call `task_create` weiter oben). Auch das periodische Versenden von Nachrichten ist möglich.

Der Zugriff auf die Message-Queue erfolgt über `msg_q_get` und `msg_q_give`, welche intern als Mutex die Message-Queue für den exklusiven Zugriff locken.

doi: 10.2313/NET-2011-07-1_04

Über `msg_post` bzw. `msg_receive` erfolgt dann der Versand bzw. Empfang der Nachrichten.

Ein Beispiel:

```
static void task1(void) {
    static Msg_t msg;
    msg.signal = 10MS_SIG;
    task_open();
    for (;;) {
        task_wait( 10 );
        msg_q_get( task2 );
        msg_post( task2, msg );
        msg_q_give( task2 );
    }
    task_close();
}
```

3.3.4 Semaphore

Über `sem_bin_create` bzw. `sem_counting_create` kann ein binäres Semaphor (mutex) oder ein zählendes Semaphor erzeugt werden. Mittels `sem_wait` und `sem_signal` werden die Operationen „P“ („Probeer“) bzw. „V“ („Vrijgeven“) umgesetzt (Nähere Informationen zu Semaphoren unter [4]).

Damit sind die Möglichkeiten von cocoOS bereits erschöpft. cocoOS bietet kein Modulkonzept, bringt keine Treiber für etwaige Funkmodule mit und bietet auch keine Netzwerk-Protokolle. Damit entfällt auch die Möglichkeit Teile des Systems zur Laufzeit auszutauschen. Der Funktionsumfang von cocoOS ist somit wesentlich geringer als der der anderen beiden vorgestellten Betriebssysteme TinyOS und Contiki.

4. ZUSAMMENFASSUNG

In der vorliegenden Arbeit wurden drei Betriebssysteme für Wireless Sensor Netzwerkknoten vorgestellt: Tiny OS, Contiki und cocoOS. TinyOS, das älteste der drei vorgestellten Betriebssysteme, zeichnet sich besonders durch den stabilen Einsatz aus. Contiki bietet durch den Einsatz von Protothreads neue Möglichkeiten, die so bei TinyOS bislang nicht möglich waren und erst durch TOSThreads nachträglich eingebaut wurden. Es wurde das dynamische Laden von Anwendungen unter Contiki vorgestellt, ein Aspekt, der gerade im WSN Umfeld sehr positiv anzusehen ist. Sowohl TinyOS als auch Contiki OS verbergen erfolgreich die Komplexität der Hardware vor dem Programmierer und erlauben die Entwicklung von Anwendungen, die auf Motes unterschiedlicher Hersteller laufen. cocoOS hingegen ist ein sehr neues Projekt, welches aktuell erst einige wenige Datenstrukturen und Basis-Funktionen eines Betriebssytems bietet, und sich um Dinge wie eine HAL (Hardware Abstraction Layer) bislang nicht gekümmert hat. Durch das Studium des noch überschaubaren cocoOS Sourcecodes ist es jedoch noch gut möglich, ein Verständnis für diese grundlegenden Betriebssystem-Objekte zu entwickeln, ein Aspekt der bei TinyOS und Contiki durch die inzwischen erreichte Komplexität der Projekte mit deutlich mehr Aufwand verbunden ist.

Die meiste Dokumentation zum Betriebssystem ist aktuell bei TinyOS vorhanden. Hier werden dem Entwickler in mehreren Tutorials die grundlegenden Konzepte des Betriebssystems nahegebracht. Die Dokumentation in Contiki ist als oberflächlich zu bezeichnen. Entwickler, die genauere Informationen benötigen, sind darauf angewiesen, das Verhalten in den Source-Dateien von Contiki nachzuvollziehen. cocoOS besitzt von allen drei Betriebssystemen die kürzeste Dokumentation, was aber in Anbetracht der einfachen Datenstrukturen, und dem geringen Funktionsumfang nachvollziehbar ist.

Betriebssysteme auf Mikrocontrollern befinden sich auf einer wesentlich niedrigeren Evolutionsebene als andere bekannte Betriebssysteme aus dem Desktop- und Server-Umfeld. Dies ist in erster Linie den stark beschränkten Hardware-Ressourcen geschuldet, die für die Entwicklung von Programmen auf den Controllern zur Verfügung stehen. Zudem handelt es sich um ein relativ spezielles Einsatzgebiet, welches von großen Unternehmen bislang nicht als Markt betrachtet wird. Durch die vielen Aktivitäten im Bereich der Wireless Sensor Networks ist jedoch davon auszugehen, dass alle drei vorgestellten Betriebssystemen stetig weiterentwickeln werden.

5. FAZIT

Alle drei vorgestellten Betriebssysteme ermöglichen die Entwicklung von Anwendungen und entlasten den Entwickler davon, wichtige Komponenten wie Ereignisbehandlung und Netzwerk-Konnektivität selbst implementieren zu müssen. Sowohl TinyOS als auch Contiki bieten eine breite Unterstützung für vielfältige Netzwerkprotokolle. Eine generelle Empfehlung für eines der Betriebssysteme ist schwer zu geben, da sie sich in bestimmten Punkten (z.B. Programmiermodell, Sprachsyntax, Netzwerk-Schichten, Filesystem) teils deutlich unterscheiden.

6. LITERATUR

[1] ZigBee Alliance. ZigBee Alliance. `http://www.zigbee.org`, 2011.

[2] Dargie, W. and Poellabauer, C. *Fundamentals of wireless sensor networks: theory and practice*. John Wiley and Sons, 2010.

[3] Jonathan Hui David Gay. TEP 103: Permanent Data Storage. `http://www.tinyos.net/tinyos-2.x/doc/html/tep103.html`.

[4] Edsger W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages: NATO Advanced Study Institute*, pages 43–112. Academic Press, 1968.

[5] Adam Dunkels, Oliver Schmidt, and Thiemo Voigt. Using Protothreads for Sensor Node Programming. In *Proceedings of the REALWSN'05 Workshop on Real-World Wireless Sensor Networks*, Stockholm, Sweden, June 2005.

[6] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David 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, pages 1–11, New York, NY, USA, 2003. ACM.

[7] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, and Philip Levis. CTP: Robust and efficient collection through control and data plane integration. February 2008.

doi: 10.2313/NET-2011-07-1_04

[8] Mohamed Hefeeda and Majid Bagheri. Wireless sensor networks for early detection of forest fires. 2007.

[9] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. *SIGPLAN Not.*, 37:96–107, October 2002.

[10] Jeonggil Ko, Qiang Wang, Thomas Schmid, Wanja Hofer, Prabal Dutta, and Andreas Terzis. Egs: A Cortex M3-based Mote Platform.

[11] Philip Levis. TinyOS Programming Manual. `http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf`, 2006.

[12] Miklos Maroti, Branislav Kusy, Simon, and Akos Ledeczi. The flooding time synchronization protocol. pages 39–49. ACM Press, 2004.

[13] Martin Leopold. Creating a new platform for TinyOS 2.x. `http://www.tinyos.net/tinyos-2.1.0/doc/html/tep131.html`, 2007.

[14] Philip Levis, Cory Sharp. Schedulers and Tasks. `http://www.tinyos.net/tinyos-2.x/doc/html/tep106.html`, 2011.

[15] Tom Sheldon. CSMA/CD explained. `http://www.linktionary.com/c/csma.html`, 2001.

[16] Tanenbaum, Andrew S. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[17] TinyOS Project. Deluge T2 - Tutorial. `http://docs.tinyos.net/index.php/Deluge_T2`, 2010.

[18] TUM, LS Knoll. Multifunk. `http://www6.in.tum.de/pub/Main/ResearchMultifunk/micaz.PNG`, 2011.

[19] Various. cocoOS. `http://www.cocoos.net`.

[20] Various. IPv6 over Low power WPAN (6lowpan). `http://datatracker.ietf.org/wg/6lowpan/charter/`.

[21] Various. Netduino. `http://netduino.com`.

[22] Various. TOSThreads Tutorial. `http://docs.tinyos.net/index.php/TOSThreads_Tutorial`, 2009.

[23] Various. CC2420 Security Tutorial. `http://docs.tinyos.net/tinywiki/index.php/CC2420_Security_Tutorial`, 2010.

[24] Various. ATmega128 Product page. `http://www.atmel.com/dyn/products/product_card.asp?part_id=2018`, 2011.

[25] Various. Dissemination. `http://docs.tinyos.net/tinywiki/index.php/Dissemination`, 2011.

[26] Various. FAQ - TinyOS Documentaiton Wiki. `http://docs.tinyos.net/tinywiki/index.php/FAQ`, 2011.

[27] Various. MSP430G2xx 16 bit Microcontroller. `http://focus.ti.com/paramsearch/docs/parametricsearch.tsp?familyId=1937&sectionId=95&tabId=2662&family=mcu`, 2011.

[28] Various. Storage Tutorial. `http://docs.tinyos.net/index.php/Storage`, 2011.

[29] Various. TinyOS - Resource Arbitration and Power Management. `http://docs.tinyos.net/index.php/Resource_Arbitration_and_Power_Management`, 2011.

[30] Various. TinyOS Home Page. `http://www.tinyos.net`, 2011.

doi: 10.2313/NET-2011-07-1_04

Event Detection in WSNs - Vehicle Tracking

Michael Stuber
Seminar Sensorknoten: Betrieb, Netze & Anwendungen, SS 2011
Institut für Informatik
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Technische Universität München
stuberm@in.tum.de

KURZFASSUNG

In dieser Arbeit wird die Ereigniserkennung in kabellosen Sensornetzwerken untersucht. Eine Verwendungsmöglichkeit der Ereigniserkennung ist das Tracken von Fahrzeugen. Die besonderen Anforderungen des Fahrzeugtrackens erfordern besondere Algorithmen zur Ereigniserkennung und Klassifizierung. Es werden zwei Ansätze zur Realisierung eines solchen Sensornetzwerkes vorgestellt und miteinander verglichen: zum einen das energieeffiziente Überwachungsnetzwerk VigilNet, zum anderen ein für das SensIT-Projekt entwickelter Ansatz.

Schlüsselworte

WSN, Event Detection, Vehicle Tracking, VigilNet, SensIT

1. EINLEITUNG

In dieser Arbeit geht es darum, die Möglichkeiten zu untersuchen, Events mit Hilfe von Sensorknoten zu erkennen. Dafür werden viele Sensorknoten in einem Gebiet verteilt, die diese Events erkennen sollen. Ein solches Ereignis kann beispielsweise das Vorbeifahren eines Fahrzeuges sein. Generell werden mehrere Schritte durchgeführt, um ein Fahrzeug zu tracken.

Zuerst muss das Ereignis erkannt werden, dass sich etwas vorbeibewegt hat. Anschließend wird das, was sich vorbeibewegt hat, genauer spezifiziert (es hat sich ein Fahrzeug, eine Person, etc. vorbeibewegt). Zuletzt können noch bestimmte Attribute wie Geschwindigkeit oder Position bestimmt werden.

2. GRUNDSÄTZLICHES

Sensornetzwerke zur Ereigniserkennung haben bestimmte Eigenschaften und stellen gewisse Anforderungen an das Design des Systems.

2.1 Bedingungen

Für ein gutes System müssen die Sensorknoten bestimmte Eigenschaften erfüllen, um mit den Einsatzbedingungen zurecht zu kommen.

Meistens werden sehr viele Sensorknoten benötigt, um akzeptable Ergebnisse zu erzielen. Diese Knoten werden auf eine große Fläche verteilt. Daraus ergeben sich besondere Anforderungen an die Kommunikation. Die Knoten müssen über eine weite Distanz senden und empfangen können und der Kommunikationsverkehr steigt proportional zur Anzahl der Knoten. Deshalb ist es erstrebenswert, die Informationen schon zu bearbeiten oder zu filtern, bevor sie von den Knoten weiterverschickt werden. Denn das Versenden von Informationen an Kommunikationspartner kostet relativ viel Energie. Ein effizientes System versucht also das Kommunikationsaufkommen zu minimieren (vgl. [2]).

Für die Erkennung und Klassifizierung können komplexe Algorithmen aus der Signalverarbeitung nützlich sein. Aus Kostengründen und Energieeffizienz sind die meisten Sensorknoten jedoch relativ leistungsarm. Es wird ein schwacher Prozessor mit geringem Speicher verbaut. Ein Beispiel wäre ein Knoten der Berley mica Serie, der einen 8-Bit Mikroprozessor ohne Unterstützung für Fließkomma-Operationen und 4 kB Speicher hat. Die meisten Algorithmen der Signalverarbeitung sind schlichtweg zu aufwendig für die schwachen Mikrocontroller. Deshalb müssen die Algorithmen für die Ereigniserkennung und Klassifizierung möglichst simpel, speichereffizient und damit schnell ausführbar sein (vgl. [2]).

Die Echtzeitanforderungen sind applikationsabhängig stringent. In den meisten Fällen ist es wünschenswert, dass das System ein Objekt so schnell erkennt, dass eine Reaktion auf das Objekt noch möglich ist. Da sich das Objekt fortbewegt, kann es passieren, dass sich das Objekt bei zu langsamer Verarbeitung der Daten auf dem Sensorknoten bereits aus dem Sensorfeld heraus bewegt hat, bevor eine Reaktion möglich ist. Daher sollten leistungsfähige Prozessoren und einfache Algorithmen verwendet werden (vgl. [2]).

Für genaue Messergebnisse der Sensoren werden hohe Samplingraten benötigt. Hohe Samplingraten haben zur Folge, dass viele Daten in kurzer Zeit aufgezeichnet werden. Dies erfordert eine schnelle und speichergünstige Verarbeitung der Daten, da nicht viel Speicher zur Verfügung steht. (vgl. [2])

Schließlich werden die Knoten in die echte Umgebung ausgesetzt. Die Knoten stehen unter realen Umwelteinflüssen und sind dem Wetter und anderen Störfaktoren ausgesetzt. Dies erfordert eine hohe Fehlertoleranz und eine Resistenz gegen Störungen. So führt beispielsweise ein Temperaturanstieg dazu, dass sich die Grenzwerte der Infrarot-Sensoren für die Bewegungserkennung verschieben. Wegen den teilweise sensiblen Sensoren ist diese Anforderung eine große Herausforderung. Sensornetze, die nur bei milden Temperaturen und Windstille akkurate Ergebnisse liefern, finden nur bedingt Einsatzmöglichkeiten (vgl. [2]).

2.2 Verwendungszweck

Es gibt sowohl militärische als auch zivile Verwendungszwecke für das Fahrzeugtracking mit Hilfe von Sensorknoten. Für das Militär sind die Informationsgewinnung und Überwachung eines bestimmten Gebietes extrem wichtig. Beispiele für getrackte Objekte im militärischen Bereich sind die Überwachung von Panzern, Fregatten oder Drohnen.

Im zivilen Bereich können solche Sensornetzwerke beispielsweise für intelligente Verkehrssysteme verwendet werden. So wird die Position von Geldtransportern überwacht, es existieren Notrufsysteme, die über GPS die aktuelle Position mitteilen, und eine Kooperation zwischen dem Mobilfunknetzbetreiber Vodafone und dem Navigationsgerätehersteller TomTom ermöglicht es, dass Staumeldungen anhand der Position der Handybesitzer an die Navigationssysteme gemeldet werden können.

doi: 10.2313/NET-2011-07-1_05

2.3 Lösungsmöglichkeiten

Der herkömmliche Weg (Methode 1), wie man bei der Fahrzeugerkennung bei kabellosen Sensornetzwerken vorgehen kann, ist die zentrale Berechnung aller Daten auf einer Basisstation. Die Knoten zeichnen die Daten ihrer Sensoren auf und schicken diese unangetastet an eine zentrale Basisstation. Die Zentrale führt die notwendigen Berechnungen zur Ereigniserkennung durch. Der Vorteil dieses Ansatzes ist die einfache Realisierung eines solchen Systems. Die Basisstation hat viel Rechenleistung, einen großen Speicher und kann komplexe Algorithmen schnell durchführen. Allerdings ist das Kommunikationsaufkommen sehr hoch, da die Daten ungefiltert direkt verschickt werden. Ein hohes Kommunikationsaufkommen hat zur Folge, dass sich die Laufzeit der Sensorknoten stark reduziert, da das Senden der Daten vergleichsweise viel Energie kostet. Außerdem müssen für die Ereigniserkennung ausreichend Daten an der Basisstation vorliegen, was eine Verzögerung mit sich bringt. Es dauert bis genügend Informationen gesammelt und verschickt werden, damit die Ereigniserkennung durchgeführt werden kann.

Der andere Weg (Methode 2) verfolgt den Ansatz, die aufgezeichneten Daten zuerst zu verarbeiten, um danach nur relevante Informationen zu verschicken. Dazu muss jeder Knoten entscheiden, ob ein Ereignis (Objekt fährt vorbei) eingetreten ist. Das Ereignis wird klassifiziert (Fahrzeug ist vorbeigefahren) und bestimme Attribute werden berechnet (Geschwindigkeit oder Position). Versendet werden nur die verarbeiteten Informationen, was das Kommunikationsaufkommen deutlich reduziert. Ein reduziertes Kommunikationsaufkommen hat eine längere Laufzeit der Knoten zur Folge. Jedoch ist die Realisierung dieser verteilten Intelligenz wesentlich aufwendiger. Die Entscheidung, ob ein Ereignis eingetreten ist, wird von einem einzigen Knoten getroffen, was sich auf die Fehlertoleranz auswirkt. Projekte, die Methode 2 anwenden, werden im Nachfolgenden näher charakterisiert.

3. VIGILNET

VigilNet ist ein Sensor-Netzwerk für die energieeffiziente Überwachung [1]. Es wird von der Technologieabteilung Defense Advanced Researchs Projects Agency (DARPA) des amerikanischen Verteidigungsministeriums gesponsert. Die allgemeine Aufgabe von VigilNet ist es, einen militärischen Kommandanten auf interessante Ereignisse in einer feindlichen Region aufmerksam zu machen. Interessante Ereignisse sind in diesem Fall das Vorbeifahren von unterschiedlichen Fahrzeugen.

3.1 Beschreibung

Bei VigilNet werden 200 ExScal Sensorknoten verwendet, die mit einem Mikrofon, vier passiven Infrarotsensoren (PIR), mit denen die Bewegungen von Objekten erkannt werden können, und einem Magnetometer, der die magnetische Flussdichte misst, ausgestattet sind. Abbildung 1 zeigt einen solchen Sensorknoten.

Das gesamte System ist jedoch darauf konzipiert, um mit mindestens 1000 Knoten auf einem Gebiet von mindestens 100 x 1000 Quadratmetern akzeptable Ergebnisse zu liefern (vgl. [1]).

Abbildung 1: ExScal Knoten ([2])

Es können Fahrzeuge, Personen und Personen mit metallischen Gegenständen erkannt werden.

Ein erkanntes Objekt wird einem externen Gerät gemeldet. Ein externes Gerät kann ein mächtigerer Knoten oder ein anderes Gerät sein, das die Informationen weiterverarbeitet. Wird ein Objekt erkannt, dann wird es klassifiziert und Attribute wie Geschwindigkeit und Position berechnet und periodisch an das externe Gerät weitergeleitet (vgl. [2]).

Das Ziel ist es, möglichst viele Objekte zu erkennen und möglichst wenig Objekte fälschlicherweise zu erkennen, die eigentlich gar keine realen Objekte sind.

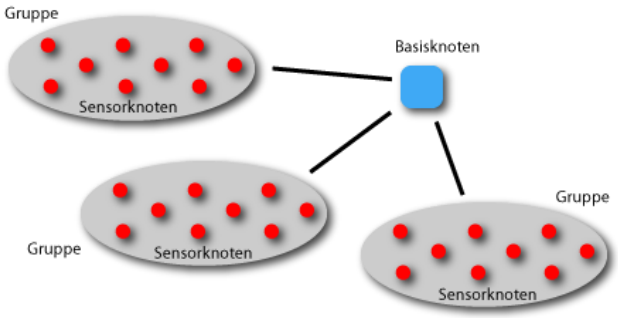


Abbildung 2: Hierarchische Struktur

Um die Fehlertoleranz zu erhöhen, ist es ratsam, dass mehrere Sensorknoten kooperieren können. Die Sensorknoten werden hierarchisch angeordnet. Auf einem Sensorknoten sind mehrere Sensoren verbaut. Diese unterste Ebene umfasst die verschiedenen Abtastalgorithmen für die einzelnen Sensordaten. Die gesammelten Daten werden an die nächste Schicht weitergereicht: die Sensorknoten selbst. Die Sensorknoten-Schicht verarbeitet die bereits gefilterten Daten verschiedener Sensortypen. Defekte Sensoren werden auf dieser Ebene erkannt. Mehrere Sensorknoten werden zu einer Gruppe zusammengefasst. Die Gruppen werden von einer Middleware verwaltet, welche Methoden zur dynamischen Gruppenorganisation zur Verfügung stellt. Es wird zum Beispiel ein Leader-Knoten bestimmt, der die Daten der restlichen Gruppenknoten einsammelt und Klassifizierungs-

doi: 10.2313/NET-2011-07-1_05

algorithmen auf Gruppenebene durchführt. Die Gruppen leiten ihre Ergebnisse an einen zentralen Basisknoten, dem höchsten Element in der Hierarchie, weiter. Dort werden beispielsweise Attribute (Geschwindigkeit) der Objekte berechnet. Abbildung 2 illustriert die hierarchische Struktur des Netzwerkes (vgl. [2]).

3.2 Algorithmen

Die Sensoren haben unterschiedliche Aufgaben und verwenden dafür unterschiedliche Algorithmen, die detailliert beschrieben sind in [2].

3.2.1 Magnetometer

Die Aufgabe des Magnetometers ist es, Fahrzeuge und Personen mit metallischen Gegenständen zu erkennen. Mit dem verwendeten Sensor kann die Ablenkung des magnetischen Felds gemessen werden, welche durch die Bewegung von metallischen Objekten entsteht. Der Magnetometer ist ein schwieriger Sensor, da zum Beispiel das elektromagnetische Rauschen des Boards das Signal - Rausch - Verhältnis (Verhältnis des Nutzsignals zur mittleren Rauschleistung des Störsignals) vermindert oder Änderungen der Umgebungstemperatur die Sensorwerte rapide ändert. Es kann sogar vorkommen, dass sich eine Temperaturänderung wie ein metallisches Objekt verhält. Grundlage für die Erkennung sind die Sensordaten, die in mehreren Schritten transformiert werden, um sie besser verarbeiten zu können. Unter anderem wird ein großer Anteil des Rauschens aus dem Signal herausgefiltert. Wird ein bestimmter Grenzwert der gefilterten Daten überschritten, so wird davon ausgegangen, dass ein Ereignis (in diesem Fall vorbeifahrendes Objekt) stattgefunden hat.

3.2.2 Passive Infrarot-Sensoren

Die passiven Infrarot-Sensoren (oder auch pyroelektrische Sensoren, abgekürzt PIR) ermöglichen die Bewegungserkennung. Hierbei wird die thermische Strahlung der Objekte in Form von infraroter Strahlung gemessen, ohne dass die Sensoren selbst Energie an die Umgebung abgeben. Das infrarote Licht prallt auf die Elektronen eines Substrats, die erkannt und in ein Signal verwandelt werden können [3].

Die PIR-Sensoren werden am stärksten vom Wetter beeinflusst, im Speziellen Wind, Temperatur und Luftfeuchtigkeit. Deshalb ist ein möglichst Wetter-resistenter Algorithmus zur Objekterkennung erstrebenswert. VigilNet verwendet einen einfachen Hochpassfilter, der hohe Frequenzen ungeschwächt passieren lässt und tiefe Frequenzen abdämpft. Die Spitzen der gefilterten Daten lassen dann auf ein sich bewegendes Objekt schließen. Jedoch variiert der Grundpegel des Frequenzspektrums je nach Wetterverhältnissen, was eine Anpassung des Grenzwertes für die Bewegungserkennung erforderlich macht (vgl .[2]).

3.2.3 Mikrofon

Der akustische Sensor wird für die Unterscheidung zwischen einem Menschen und einem Fahrzeug benötigt. Der Aufwand ist aufgrund der hohen Sampling- und Verarbeitungsrate hoch. Algorithmen der Signalverarbeitung, wie die Schnelle-Fourier Transformation zur Frequenzanalyse, sind nicht implementierbar, da die Leistung der Sensorknoten für eine zeitnahe Berechnung nicht ausreicht. Deshalb verwendet VigilNet einen schnelleren Ansatz. Bei jedem neuen akustischen Sample wird ein gewichteter Durchschnittswert der Akustik-Signale aktualisiert. Dieser Wert wird benötigt, um eine Variable zu berechnen, die mit der unmittelbaren akustischen Energie zusammenhängt. Dieser zur Energie relative Faktor wird wiederum für die Berechnung der Grenzwerte verwendet, mit denen die Klassifizierung der Objekte möglich ist. Falls die aktuelle akustische Energie (Et) großer als der berechnete Grenzwert ist, wird davon ausgegangen, dass sich ein Fahrzeug vorbeibewegt hat.

3.2.4 Geschwindigkeits- und Positionsberechnung

Die Basisstation ist verantwortlich für die Berechnung von Geschwindigkeit und Position. Dort liegen die Ergebnisse der Berechnungen aller Sensorknoten vor, somit wird eine globale Sicht auf die erkannten Ereignisse der getrackten Objekte ermöglicht. Dazu merkt sich die Basisstation den Verlauf der kürzlich erhaltenen Kalkulationsergebnisse. Bekommt sie beispielsweise zwei Meldungen von Objekten, die in der unmittelbaren Umgebung liegen, so wird davon ausgegangen, dass die beiden Meldungen von dem selben Objekt stammen. Anhand der Positionsdifferenz und den unterschiedlichen Zeitpunkten kann die Geschwindigkeit des sich bewegenden Objektes bestimmt werden.

3.3 Weiterverarbeitung der Daten

Mehrere Sensorknoten bilden eine Gruppe mit einem Gruppenleiter. Die auf den einzelnen Sensorknoten gesammelten Daten werden periodisch an den jeweiligen Gruppenleiter in Form von Berichten geschickt. Ein Bericht kann beispielsweise Knoteninformationen (welcher Knoten sendet), Gruppen-informationen (Gruppen-ID) und Ereignisinformationen (Position, Geschwindigkeit) enthalten. Die Gruppenleiter sammeln diese Berichte und leiten sie gebündelt an die Basisstation weiter (vgl. [2]).

An der Basisstation laufen die Informationen der Gruppen zusammen. Dort wird die endgültige Klassifizierung bestimmt und die Werte der Attribute können berechnet werden. Dazu wird für jedes getrackte Objekt eine Datenstruktur angelegt, die die Position, einen Zeitstempel, die Sensorwerte und einen Verweis zu dem letzten Objekt von dem Bericht enthält. Falls ein neuer Bericht mit einer Position eintrifft, die sich in der Nähe eines vom System bereits erkannten Objekts befindet, wird dieser Bericht genau diesem Objekt zugeordnet. Ist dem System kein Objekt in der Nähe der Position des neuen Berichtes bekannt, so wird ein neues Objekt angenommen. Anhand vom Verlauf der Daten kann man Attribute wie Geschwindigkeit berechnen, die sich von der Position und den Zeitstempeln ableiten lässt (vgl. [2]).

3.4 Bewertung

VigilNet ist ein komplexes Sensornetzwerk zur Erkennung von Ereignissen und zum Tracken von Fahrzeugen und anderen Objekten. Die verwendeten Algorithmen zur Ereigniserkennung sind aufgrund der Anforderungen simpel und damit schnell berechenbar. Bemerkenswert ist, dass VigilNet ohne eine Analyse im Frequenz-Spektrum auskommt, da auf die Anwendung der Fourier-Transformation verzichtet wird. Dies unterstützt den Fokus des Projekts auf die Energieeffizienz.

4. SENSIT-PROJEKT

Das Sensor Information Technology - Projekt (SensIT) ist ein von der DARPA gefördertes Projekt zur Entwicklung einer neuen Software-Klasse für verteilte Mikrosensoren [5]. Durch SensIT wird die Entwicklung von Sensornetzen für das Fahrzeugtracking vereinfacht. Anhand der Förderung des Projekts durch die DARPA erkennt man die hauptsächliche militärische Anwendung von Sensornetzwerken zum Tracken von Fahrzeugen.

doi: 10.2313/NET-2011-07-1_05

4.1 Beschreibung

Beim SensIT-Projekt wurde eine Versuchsstrecke aufgebaut, an der verschiedene Teams ihre Methoden zu Fahrzeugerkennung testen und demonstrieren konnten. Es wurden insgesamt 75 Sensorknoten vom Typ WINS NG 2.0 (vgl. Abbildung 3) auf dem Marine Corps Air Ground Combat Center in Twenty-Nine Palms in Kalifornien verteilt.

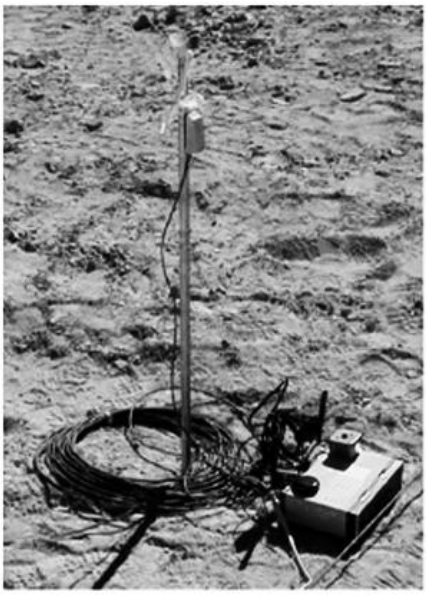


Abbildung 3: Ein WINS NG 2.0 Knoten ([4])

Jeder Knoten ist mit einem akustischen, seismischen und passivem Infrarot-Sensor zur Bewegungserkennung ausgestattet. Ein echtzeitfähiger digitaler Signalprozessor erlaubt die Verwendung von aufwendigen Signalverarbeitungsalgorithmen, wie der schnellen Fourier-Transformation. Das Projekt dauerte zwei Wochen lang und die Sensoren wurden auf einem Gebiet von 900 x 300 m² verteilt. Die Fahrzeuge haben sich auf zwei Straßen und einer Kreuzung der beiden Straßen fortbewegt: die Ost-West-Route bildete zusammen mit der Nord-Süd-Route an deren Kreuzung eine Art umgedrehtes "T" (vgl. Abbildung 4).

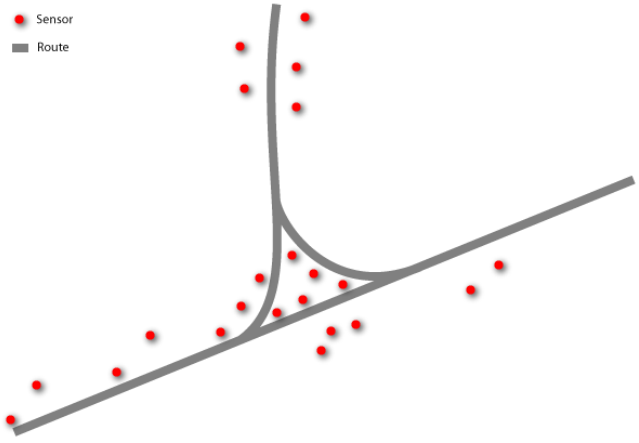


Abbildung 4: Feld Layout

Vier verschiedene Fahrzeugklassen sind drei verschiedene Fahrrouten entlang gefahren. Zu den ausschließlich militärischen Fahrzeugen zählten ein Assault Amphibian Vehicle (AAV), ein Main Battle Tank (M1), ein High Mobility Multipurpose Wheeled Vehicle (HMMWV) und ein Dragon Wagon (DW). Die Routen waren von Westen nach Norden, von Norden nach Osten und von Osten nach Westen.

4.2 Vorgehensweise

[4] präsentieren anhand im Rahmen von SensIT die Ereigniserkennung und Klassifizierung der verschiedenen Fahrzeuge. Grundlage ist die Kalibrierung der Sensorknoten vor der Benutzung des Systems.

4.2.1 Ereigniserkennung

Der erste Schritt des Vehicle Trackings ist, zu erkennen, dass sich ein Objekt vorbeibewegt hat. Durch die Erkennung dieser Ereignisse kann das Datenaufkommen erheblich reduziert werden, da die irrelevanten Sensordaten, bei denen sich kein Objekt vorbeibewegt hat, nicht weiterverarbeitet werden müssen. Bei dem hier vorgestellten Ansatz wird das anhand des akustischen Energiepegels entschieden. Eine solche Entscheidung wird alle 0,75 Sekunden herbeigeführt. Die Spitzen der akustischen Signale stellen ein vorbeifahrendes Objekt dar. Zur Klassifizierung und Berechnung von bestimmten Attributen werden die Datensätze bestehend aus akustischen, seismischen und infraroten Sensordaten behalten. Anhand dieser Daten wird die Klassifizierung des Ereignisses mittels verschiedener Algorithmen durchgeführt (vgl. [4]).

4.2.2 Klassifizierung

Für die Klassifizierung werden bestimmte Eigenschafts-Vektoren anhand der Sensordaten berechnet. Explizit werden die akustischen und seismischen Aufzeichnungen hierfür verwendet. Es wird eine schnelle Fourier-Transformation durchgeführt, um das Frequenzspektrum analysieren zu können. Eine der einfachsten Klassifizierungsmethode ist der k-nächste Nachbar (k-NN) Klassifizier, der auf der Annahme basiert, dass Objekte mit ähnlichen („benachbarten“) Sensorwerten zu derselben Objektklasse gehören. Das System wird zu Beginn eines Aufzeichnungslaufs mit Daten gefüttert, damit die Knoten kalibriert werden können. Ein Objekt wird vom k-NN Klassifizierer erkannt, wenn während der Kalibrierungsphase genügend Objekte derselben Klasse aufgezeichnet wurden (vgl. [4]). Trotz der Einfachheit liefert der k-NN Algorithmus akkurate Ergebnisse. Jedoch benötigt dieser Ansatz viel Zeit, da die Ähnlichkeit des aktuellen Samples mit den Kalibrierungssamples vergleichen werden muss. Die Zeit, die für die Klassifizierung benötigt wird, ist also proportional zur Anzahl an aufgezeichneten Kalibrierungsdatensätzen.

Tabelle 1: Ergebnisse der Klassifizierung mit k-NN zeigt die Ergebnisse eines Testlaufs von der Klassifizierung über k-NN. Die Klassifizierungsrate ist dabei die Anzahl an korrekt klassifizierten Objekten geteilt durch die gesamte Anzahl an Objekten. Die Ablehnungsrate entspricht den fälschlicherweise nicht erkannten Objekten. Eine hohe Klassifizierungsrate und eine niedrige Ablehnungsrate sind erstrebenswert.

Tabelle 1: Ergebnisse der Klassifizerung mit k-NN ([4])

Objekt	Klassifizierungs-rate	Ablehnungsrate
AAV3	73,33 %	6,25 %
AAV6	100,00 %	2,86 %

doi: 10.2313/NET-2011-07-1_05

AAV9	84,31 %	0,00 %
DW3	85,71 %	0,00 %
DW6	100,00 %	0,00 %
DW9	75,86 %	17,14 %
DW12	65,63 %	25,58 %
Gesamt	**83,55 %**	**7,40 %**

4.3 Bewertung

Der hier vorgestellte Ansatz hat einen starken Fokus auf die Klassifizerung der getrackten Objekte. Die Hardware-Anforderungen an die Sensorknoten sind relativ hoch, da eine schnelle Fourier-Transformation der Sensorwerte zur Frequenzanalyse durchgeführt werden muss. Grundlage des Systems ist eine Kalibrierung der Sensorknoten. Damit können bessere Messergebnisse erzielt werden.

5. VERGLEICH

Tabelle 2 gibt einen Überblick über den Vergleich der Ansätze in VigilNet und im SensIT-Projekt.

Tabelle 2: Vergleich VigilNet und SensIT

	VigilNet	**SensIT**
Anzahl Knoten	200	75
Art der Knoten	ExScal	WINS NG 2.0
Sensoren	1 x Mikrofon, 4 x Infrarot, 1 x Magnetometer	1 x Mikrofon, 1 x Infrarot, 1 x Seismisch
Fläche		900 x 300 m²
Objekte	Fahrzeuge, Personen, Personen mit metallischen Gegenständen	Assault Amphibian Vehicle, Main Battle Tank, High Mobility Multipurpose Wheeled Vehicle, Dragon Wagon
Prozessor-leistung	niedrig	hoch
Kalibrierung	nicht unterstützt	unterstützt

Die Ziele beider Projekte sind ähnlich: sich bewegende Objekte im Freien sollen mit Hilfe von vielen Sensorknoten erkannt und Attribute bestimmt werden. Die Lösungsansätze unterscheiden sich jedoch in bestimmten Punkten. Die Art der Sensoren ist unterschiedlich. Auf der einen Seite werden seismische Sensoren, auf der anderen Seite Magnetometer verwendet. Die Art der Objekte, die erkannt werden, differiert ebenfalls und reicht von Personen zu großen, militärischen Fahrzeugen. Die Algorithmen unterscheiden sich stark. Bemerkenswert hierbei ist, dass der Ansatz bei VigilNet ohne eine Form der Fourier-Transformation auskommt und geringe Anforderungen an die Rechenleistung des Sensorknoten stellt.

Im Gegensatz dazu führen die im SensIT-Projekt verwendeten Sensorknoten schnelle Fourier-Transformationen zur Frequenzanalyse durch. Die Anforderungen an die Prozessoren sind deutlich höher. Der Ansatz bei VigilNet beschreibt die hierarchische Organisation der Sensorknoten recht ausführlich. Die Knoten werden auf verschiedenen Eben gruppiert. Der SensIT-Ansatz ermöglicht eine Kalibrierung des Systems für bessere Ergebnisse. Dieses Feature fehlt bei VigilNet. Stattdessen werden statische Grenzwerte verwendet. (vgl. [6])

6. FAZIT

In dieser Arbeit wurden zwei Ansätze zur Realisierung des Fahrzeugtrackings mit Hilfe eines Sensornetzwerkes untersucht. Die beiden Varianten sind durchaus unterschiedlich, verfolgen aber das gleiche Ziel. Mit Hilfe von einer großen Anzahl an Sensorknoten sollen verschiedene, sich in einer realen Umgebung bewegende Objekte erkannt und Attribute spezifiziert werden. Unterschiedliche Algorithmen zur Ereigniserkennung und Klassifizierung von getrackten Objekten kommen zum Einsatz. Der Entwurf von solchen Systemen ist aufgrund der Anforderungen nicht trivial. Energieeffizienz steht an oberster Stelle der nicht-funktionalen Anforderungen. Deshalb werden intelligente Lösungen zur Verminderung des Kommunikationsaufkommens und zur Reduzierung der Rechenlast benötigt.

7. LITERATUR

[1] VigilNet Website, http://www.cs.virginia.edu/wsn/vigilnet/index.html, aufgerufen am 26.06.2011

[2] L. Gu, D. Jia, P. Vicaire, T. Yan, L. Luo, A. Tirumala, Q. Cao, T. He, J. A. Stankovic, T. Abdelzaher, B. H. Krogh. Lightweight Detection and Classification for Wireless Sensor Networks in Realistic Environments. In Proc of 3rd Intl. Conf. on Embedded Networked Sensor Systems (SenSys '05), San Diego, CA, USA, Nov. 2005.

[3] How PIR Motion Sensors Work, http://www.gadgetshack.com/motionsensor.html, aufgerufen am 27.06.2011

[4] M. F. Duarte and Y. H. Hu. Vehicle Classification in Distributed Sensor Networks. Journal of Parallel and Distributed Computing, 64(7), July 2004.

[5] Srikanta Kumar, David Shepherd, DARPA- ITO: SensIT: Sensor Information Technology for the warfighter

[6] Wittenburg, Dziengel, Wartenburger, Schiller. A System for Distributed Event Detection in Wireless Sensor Networks.

MAC - Vergleich von Präambel-basierten Kanalzugriffsprotokollen

Joseph Wessner
Betreuer: Dr. Alexander Klein
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: wessnerj@in.tum.de

KURZFASSUNG

Mit die wichtigste Anforderung an drahtlose Sensornetze (WSN) ist die Energieeffizienz. Da die Kommunikation oftmals den größten Energieverbraucher darstellt, ist hier ein sparsamer Umgang mit den Ressource enorm wichtig. Die Wahl eines geeigneten Protokolls spielt dabei eine große Rolle um den Protokolloverhead möglichst gering zu halten. In diesem Paper werden daher die verschiedene Präambelbasierten Kanalzugriffsprotokolle LPL, X-MAC und BPS-MAC vorgestellt und miteinander verglichen. Die Schwerpunkte liegen dabei auf den Bereichen der Interferenz, Overhead, Paketverlust, Durchsatzrate, Verzögerung und natürlich der Energieeffizienz.

Schlüsselworte

LPL, X-MAC, BPS-MAC, Energieeffizienz, MAC Protokolle

1. EINLEITUNG

Media Access Control (MAC) Protokolle, im folgenden auch Kanalzugriffsprotokolle genannt, werden für die Kommunikation in WSN benötigt. In dem OSI-Schichtenmodell reihen sie sich dabei in die Sicherungsschicht direkt über der physikalischen Schicht ein. Ziel der MAC Protokolle ist die Energieeffizienz um eine möglichst lange Laufzeit der Sensorknoten zu ermöglichen. Da Sensorknoten meist über keine externe Stromversorgung verfügen, sondern auf ihre Batterien angewiesen sind muss der Energieverbrauch so weit wie möglich gesenkt werden. Um dieses Ziel bestmöglich zu erreichen spielt neben der Hardware auch die Wahl eines geeigneten MAC Protokolls eine wichtige Rolle. In dieser Arbeit werden im folgenden verschiedene MAC Protokolle und deren Vor- und Nachteile vorgestellt.

2. ENERGIEVERSCHWENDUNG

Natürlich gibt es viele verschiedene Arten unnötig Energie zu verbrauchen, die wichtigsten davon werden hier kurz vorgestellt.

2.1 Idle Listening

Mit Idle Listening bezeichnet man den Umstand, den Kommunikationskanal abzuhören, obwohl keine Kommunikation stattfindet. Da das Empfangen von Daten in der Regel mehr Energie benötigt als das Senden von Daten, stellt das Idle Listening einen nicht zu vernachlässigbaren Teil der Energieverschwendung dar. Um dieses Problem zu umgehen werden Sensorknoten oft in einen Schlafzustand versetzt, in welchem sie das Empfangsmodul deaktivieren, um möglichst viel Energie einzusparen.
Um das Idle Listening ganz zu vermeiden, müsste man also vorab wissen wann Daten gesendet werden. Dieses Problem lässt sich zwar theoretisch mit Hilfe von Zeitslots lösen, ist aber in der Praxis aufgrund zu ungenauer Zeitsynchronisation der Sensorknoten nur schwer zu realisieren.
Um die Zeit des Idle Listenings trotzdem möglichst gering zu halten, lassen viele MAC Protokolle das Empfangsmodul nur kurzzeitig aufwachen, um zu überprüfen ob momentan eine Kommunikation stattfindet. Falls keine Kommunikation stattfindet wird das Empfangsmodul wieder ausgeschaltet um Strom zu sparen. [5]

2.2 Overhearing

Anders als beim Idle Listening findet beim Overhearing eine Kommunikation statt, allerdings sind die Daten für einen anderen Empfänger bestimmt. Es werden also Daten empfangen mit denen der Sensorknoten nichts anfangen kann. Den Empfänger während des Datenstroms festzustellen erweist sich als recht schwierig. Die Folge dessen ist, dass meist erst am Ende des Datenstroms festgestellt werden kann, dass die Daten für einen anderen Empfänger bestimmt waren und dann verworfen werden.
Um das unnötige und stromraubende Empfangen nicht benötigter Daten zu vermeiden wird oft ein Header mit der Adresse des Empfängers vorausgeschickt. Wird also nach dem Empfang des Header bemerkt, dass die Daten für einen anderen Empfänger bestimmt sind, kann das Empfangsmodul sofort wieder in den Schlafzustand zurückkehren. Das unnötige Empfangen eines Headers stellt im Vergleich zu dem überflüssigen Empfangen der kompletten Daten einen recht geringen Stromverbrauch dar. [5]

2.3 Kollisionen

Kollisionen entstehen, wie der Name schon vermuten lässt, wenn zwei oder mehr Teilnehmer gleichzeitig Daten versenden. Aufgrund der Überlagerung beider Signale ist das Empfangen der Daten nur sehr schwer oder meist gar nicht mehr möglich.
Kollisionen stellen eine der größtmöglichen Formen der Energieverschwendung dar, da die Pakete erneut gesendet werden müssen. Das Erkennen einer Kollision ist erst nach dem Senden des kompletten Daten möglich, weshalb im Falle einer Kollision auch die komplette Übertragung erneut gesendet werden muss. Die Kollision wird entweder durch das Fehlen der Bestätigung über das Ankommen des Datenpaketes vom Empfänger erkannt, oder aber durch einen belegten Kanal

doi: 10.2313/NET-2011-07-1_06

nach der Übertragung.
Die Wahrscheinlichkeit einer Kollision hängt natürlich hauptsächlich von der Menge der zu übertragenden Daten und der Menge der vorhandenen Sensorknoten ab.
Um die Wahrscheinlichkeit von Kollisionen zu verringern setzten die verschiedenen Protokolle auch verschiedene Lösungsansätze um. Die einfachste Variante besteht darin, vor dem Senden zu überprüfen ob der Kanal bereits belegt ist. Dieser triviale Ansatz stößt allerdings auch sehr schnell an seine Grenzen, beispielsweise wenn 2 Sender gleichzeitig mit der Übertragung beginnen oder sich die beiden Sender gegenseitig nicht in Reichweite befinden, der Empfänger aber im Sendebereich beider Sender liegt. [5]

2.4 Auslastungsschwankungen

WSN sind fast immer großen Lastschwankungen ausgesetzt. Oft werden WSN für Messungen eingesetzt. Eine Veränderung an einem einzigen Messpunkt ist sehr unwahrscheinlich. Ändern sich aber die Messungen an vielen Sensorknoten gleichzeitig, versuchen die Sensorknoten ihre neuen Messdaten auch alle gleichzeitig zu senden. So tritt schnell eine Lastspitze auf, während in der anderen Zeit, wenn sich Messwerte nicht oder kaum ändern, fast kein Traffic entsteht.
Die eingesetzten Protokolle müssen also einerseits in der Lage sein diese vergleichsweise großen Lastspitzen zu handhaben und andererseits bei einer geringen Auslastung möglichst viel Energie sparen. [5]

2.5 Protokolloverhead

Der Protokolloverhead spielt in WSN eine große Rolle. Meistens werden in WSN nur sehr geringe Mengen an Daten versendet, so macht sich auch ein relativ geringer Overhead im Vergleich zu den Nutzdaten sehr schnell negativ bemerkbar. In anderen drahtlosen Netzwerken, wie beispielsweise dem von Laptops und Computern genutzten WLAN spielt der Overhead eine im Vergleich geringe Rolle, da in diesen Netzen der Datendurchsatz wesentlich höher ist als in WSN.
Um den Overhead zu reduzieren werden die Daten meistens gesammelt bevor sie versendet werden. Das Aggregieren der Daten erhöht in der Regel die Latenz, verringert aber auch den Overhead enorm. Da in vielen Fällen eine leichte Erhöhung der Latenz verschmerzbar ist, wird das Aggregieren der Daten oftmals eingesetzt. [5]

2.6 Over-emitting

Over-emitting beschreibt das Senden von Daten ohne dass der Empfänger die Daten empfängt. Dies tritt auf wenn sich der Empfänger im Schlafzustand befindet und sein Empfangsmodul deaktiviert hat.
Die Daten müssen also erneut gesendet werden, sobald bemerkt wird, dass sie beim Empfänger nicht angekommen sind. Im schlimmsten Fall, wenn das Ausbleiben des Empfangs nicht bemerkt wird, gehen die Daten sogar verloren. Durch unnötiges oder wiederholtes Senden von Daten, wird der Kanal auch länger belegt. Dies hat zur Folge, dass andere Knoten nicht senden können und sich so die Latenz des ganzen WSN erhöht.
Die naivste Lösung wäre natürlich, dass die Sensorknoten ihre Empfangsmodul ständig aktiviert haben und auf dem Kanal lauschen, so dass sie keine Übertragung verpassen. Aus Energieeffizienzgründen ist dies aber in den meisten Fällen keine akzeptable Lösung.
Viele MAC Protokolle setzten auf eine Präambel, die die Datenübertragung ankündigt. Durch die Ankündigung der Übertragung soll sichergestellt werden, dass der Empfänger sein Empfangsmodul nicht deaktiviert und auch wirklich auf dem Kanal lauscht wenn die eigentliche Übertragung stattfindet. Dabei wird aber vorausgesetzt, dass der Empfänger die Präambel empfängt. Dieses Problem wiederum versucht man beispielsweise mit einer sehr langen Präambel zu lösen. Die lange Präambel muss dabei so lange andauern, dass der Empfänger sie auch sicher empfängt.
Eine anderen Möglichkeit Over-emitting zu vermeiden besteht darin die einzelnen Knoten betreffend ihrer Aufwachzeitpunkte zu synchronisieren. Eine Synchronisierung des WSN stellt allerdings einen komplexeren Prozess dar, der viel Bandbreite und Rechenzeit benötigt.
So bewerten die verschiedenen MAC Protokolle die Vor- und Nachteile der Synchronisation unterschiedlich, weshalb auch verschieden aufgebaute Protokolle existieren.

3. MAC PROTOKOLLE

Alle existierenden MAC Protokolle aufzulisten würde den Rahmen dieser Arbeit sprengen, um dennoch einen Einblick in die MAC Protokolle zu erlangen werden in diesem Kapitel 3 verschiedene MAC Protokolle vorgestellt. Im folgenden werden die Protokolle LPL, X-MAC und BPS-MAC erläutert und deren Unterschiede aufgezeigt.

3.1 LPL

Bei dem Low Power Listening (LPL) Protokoll prüfen die einzelnen Knoten in gewissen Zeitabständen, ob der Übertragungskanal belegt ist. Ist der Übertragungskanal frei, können sie sofort wieder in den Schlafzustand übergehen. Falls der Übertragungskanal aber belegt ist, warten die Knoten so lange bis das Startsignal der Datenübertragung übertragen wird. Nachdem das Startsignal empfangen worden ist, können die Knoten entscheiden für welchen Empfänger die darauf folgenden Datenpakete bestimmt sind und sich gegebenenfalls wieder in den Schlafzustand begeben.
Diese Funktionsweise setzt voraus, dass die Präambel vor dem Startsignal länger andauert als die Zeitspanne zwischen zwei Aufwachvorgängen der beteiligten Knoten.

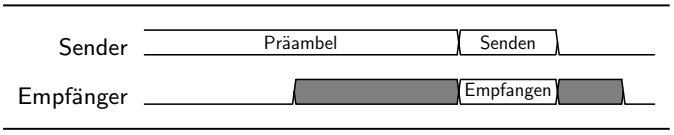


Abbildung 1: LPL Protokoll

Wie in Abbildung 1 zu sehen, beginnt der Sender damit die Präambel zu senden. Der Empfänger beginnt während der Präambel auf dem Kanal zu lauschen und erkennt dadurch, dass momentan eine Präambel gesendet wird und bald Datenpakete folgen werden, die möglicherweise für ihn selbst bestimmt sind. Dadurch ist sichergestellt, dass der Empfänger während der Datenübertragung sein Empfangsmodul aktiviert hat und dadurch die Daten auch wirklich empfangen kann. Nach dem Senden und Empfangen der Datenpakete, lauscht der Empfänger noch für eine kurze Zeitspanne ob eventuell noch weitere Datenpakete oder sogar eine andere Präambel (von einen 2. Sender) gesendet werden. Ist dies

doi: 10.2313/NET-2011-07-1_06

nicht der Fall beginnt der Knoten (Empfänger) wieder mit seinem Schlafzyklus.
Das LPL Protokoll hat zwei offensichtliche Nachteile.
Erstens wird durch die lange Präambel die aktive Sendedauer des Senders, um die meist geringen Datenmengen zu übertragen sehr lang, da Senden Strom kostet ist ein großer Overhead durch eine (im Vergleich zur Datenmenge) lange Präambel natürlich nicht wünschenswert. Auf der Empfängerseite tritt ein sehr ähnlicher unerwünschter Effekt ein. Alle möglichen Empfänger müssen die ganze Präambel abwarten bis sie entscheiden können ob sie die darauf folgenden Daten überhaupt empfangen sollen. Außer dem hohen Strombedarfs auf Senderseite, fällt auch ein hoher Strombedarf bei allen anderen Knoten an, da während der Präambel der gewünschte Empfänger nicht ermittelt werden kann.
Der zweite offensichtliche Nachteil besteht darin, dass durch die lange Präambel der Funkkanal unnötig lange blockiert wird, so dass in dieser Zeit kein weiterer Sender aktiv werden kann. [4, 1]

3.2 X-MAC

Bei der Entwicklung von X-MAC wurde besonders auf dessen Energieeffizienz geachtet. Im Gegensatz zu LPL setzt X-MAC nicht auf eine lange Präambel, um dem Empfänger die Übertragung anzukündigen, sondern auf mehrere kurze Präambelübertragungen.

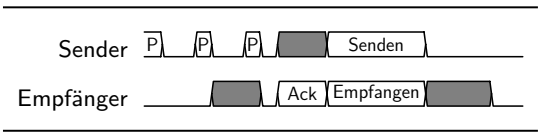


Abbildung 2: X-MAC Protokoll

Wie in Abbildung 2 dargestellt, werden mehrere kurze Präambel (P) gesendet, der zeitliche Abstand dieser Präambel muss kürzer als eine Wachphase (dunkelgrau dargestellt) des Empfängers sein, damit sichergestellt ist, dass dieser eine Präambel empfängt. Wird die Präambel vom Empfänger gelesen, schickt dieser eine Bestätigung (Ack) zurück zum Sender. Der Sender kann daraufhin mit der Übertragung der Daten beginnen. Der Empfangsmodus des Empfängers wurde durch den Bestätigungscodes Ack bereits zugesagt.
Damit nicht jeder Sensorknoten im Netzwerk den Bestätigungscode Ack sendet und damit eventuell verhindert, dass der gewünschte Empfänger über die Übertragung informiert wird, enthalten die kurzen Präambeln (P) bereits die Zieladresse der Datenpakete. Zusätzlich zur Gewährleistung, dass nur der Empfänger das Ack Signal sendet, ist es den anderen Sensorknoten auch möglich sofort nach dem Empfang einer kurzen Präambel (P) zurück in den energiesparenden Schlafzustand zu wechseln.
Das Entwicklungsziel möglichst energiesparend zu sein, wurde vor allem durch zwei Eigenschaften erreicht.
Die Sensorknoten müssen keine lange Präambel abwarten um entscheiden zu können, ob die Daten für sie oder für einen anderen Sensorknoten bestimmt sind. So können sich alle bis auf den Empfängerknoten sofort nach der Präambel wieder in den Schlafzustand begeben. Der Empfängerknoten verkürzt die Dauer der Präambelphase durch das Senden der Bestätigung.
Aber auch auf der Senderseite wird Energie eingespart. Durch die verkürzte Präambelphase wird die Sendedauer und damit die benötigte Energie verringert.
Als weiterer Vorteil sollte noch die geringere Belegungsdauer des (Funk)kanals genannt werden. Dadurch wird es weiteren Sender ermöglicht schneller auf den Kanal zuzugreifen. [1, 4]

3.3 BPS-MAC

3.3.1 Einzelne Präambel

Das Backoff Preamble-based MAC Protokoll (BPS-MAC Protokoll) hingegen ist mehr auf WSNs mit vielen Knoten ausgelegt. Da die Wahrscheinlichkeit von Kollisionen zu nimmt, je mehr Sensorknoten im Netzwerk beteiligt sind, liegt das Hauptanliegen des BPS-MAC Protokolls im Vermeiden von Kollisionen.
Trotz der Optimierung zur Vermeidung von Kollisionen kommt das BPS-MAC Protokoll ohne Synchronisation von Zeitslots oder der Speicherung größerer Datenmengen aus. Durch diesen Umstand eignet sich das BPS-MAC Protokoll auch für Sensorknoten mit begrenzter Rechen- und Speicherkapazität. So können die Kosten für BPS-MAC geeigneten Sensorknoten gering gehalten werden, was sich bei einer großen Anzahl an Netzwerkknoten, für die das Protokoll konzipiert ist, durchaus lohnt.
Im BPS-MAC Protokoll prüft der Sender zuerst ob der Kanal frei ist, ist dies der Fall sendet er eine Präambel zufälliger Länge und prüft wieder ob der Kanal frei ist. Ist der Kanal wieder frei kann der Sender problemlos seine Daten verschicken. Diese vereinfachte Darstellung stellt natürlich den optimalen Fall dar und nicht den wahrscheinlichsten.
Senden 2 (oder mehr) Sender gleichzeitig ihre Präambeln, da beide Sender den unbelegten Kanal erkannt haben, kann der Sender mit der kürzeren Präambel den anderen Sender erkennen. Wird ein anderer Sender erkannt darf die Datenübertragung natürlich nicht erfolgen, da sonst eine unerwünschte Kollision auftreten würde. Zusätzlich muss noch die Umschaltdauer vom Empfangsmodus (Rx) zum Sendemodus (Tx) und umgekehrt berücksichtigt werden. Wir nehmen im folgenden an, dass sowohl die Umschaltdauer in den Sendemodus als auch die Umschaltdauer in den Empfangsmodus jeweils eine Zeiteinheit in Anspruch nehmen. Daraus folgt, dass der (Funk)kanal mindestens für 3 Zeiteinheiten als frei erkannt werden muss um davon ausgehen zu können, dass dieser wirklich frei ist und nicht nur vorübergehend.
In Abbildung 3 werden 2 Sender dargestellt, die versuchen gleichzeitig zu Senden:

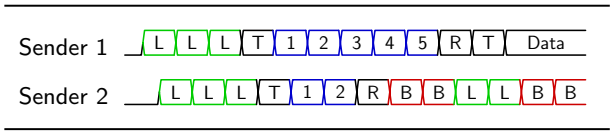


Abbildung 3: BPS-MAC mit einer Präambel

Sender 1 lauscht (L) 3 Zeiteinheiten auf dem Kanal, da der Kanal frei ist wechselt er in den Sendemodus (T) und sendet eine Präambel, die 5 Zeiteinheiten andauert (1 - 5), daraufhin wechselt er wieder in den Empfangsmodus (R), merkt dass der Kanal immer noch oder wieder frei ist, wechselt zurück in den Sendemodus (T) und beginnt mit der Datenübertragung (Data).

doi: 10.2313/NET-2011-07-1_06

Sender 2 lauscht (L) ebenfalls 3 Zeiteinheiten lang auf dem Kanal, wechselt in den Sendemodus (T) und sendet eine Präambel, die im Gegensatz zur Präambel von Sender 1 nur 2 Zeiteinheiten lang ist. Nach der Präambel wechselt auch Sender 2 wieder in den Empfangsmodus (R) zurück. Da die Präambel von Sender 1 aber noch andauert bemerkt Sender 2, dass der Kanal noch belegt ist (B). In der Zeit, die Sender 1 benötigt um in den Empfangsmodus und wieder zurück in den Sendemodus zu wechseln, wird der Kanal von Sender 2 zwar als frei wahrgenommen (L), aber kurz darauf wenn die Datenübertragung beginnt wird der Kanal wieder als belegt (B) erkannt.
So kann die Datenübertragung von Sender 1 ohne Kollision mit Sender 2 erfolgen. Sender 2 beginnt wieder mit dem Senden einer zufällig langen Präambel, sobald die Datenübertragung von Sender 1 abgeschlossen ist. Das Beispiel lässt sich natürlich auch auf mehr als nur 2 Sensorknoten, die gleichzeitig senden wollen übertragen.
Trotzdem kann es immer noch zu Kollisionen kommen, allerdings nur wenn 2 Sender gleichzeitig beginnen eine gleich lange Präambel zu senden, wie in Abbildung 4 veranschaulicht.

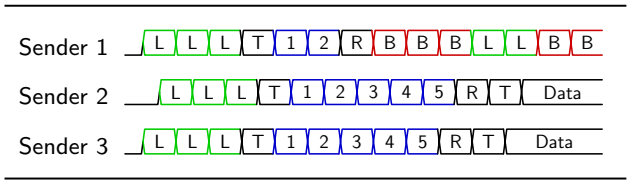


Abbildung 4: Kollision im BPS-MAC

In diesem Beispiel versuchen 3 Sender gleichzeitig Daten zu senden. Sender 1 wählt eine Präambeldauer von 2, während Sender 2 und 3 beide eine Präambeldauer von 5 wählen. Sender 1 ist also in der Lage den durch Sender 2 und 3 belegten Kanal zu erkennen und seine eigene Datenübertragung zu verschieben um eine Kollision zu vermeiden.
Sender 2 und 3 allerdings können sich gegenseitig nicht als sendewillig erkennen, da sie beide innerhalb des gleichen Zeitslots mit einer gleich langen Präambel begonnen haben. Durch diese identische Präambel erscheint beiden Sendern der Kanal als frei und sie beginnen mit ihrer Datenübertragung, die unweigerlich zur Kollision führt.
Die optimale maximale Präambeldauer hängt von der Umgebung ab. Die Anzahl der beteiligten Sensorknoten sowie die Menge der übertragenden Daten spielen dabei eine wichtige Rolle. Dennoch gilt auch bei BPS-MAC, die Wahrscheinlichkeit einer Kollision steigt mit der Anzahl der Sensorknoten und der Menge der übertragenden Daten. [3]

3.3.2 Mehrere Präambeln

Um die Wahrscheinlichkeit einer Kollision weiter zu verringern, werden ähnlich dem Schritt vom LPL zum X-MAC Protokoll, mehrere kürzere Präambel statt einer langen Präambel verwendet.

Sender 1, 2 und 3 wollen innerhalb des gleichen Zeitslots Daten versenden. Nachdem jeder Sender drei Zeitslots gelauscht (L) und keine Übertragung feststellen konnte, wechseln alle drei Sender in den Sendemodus (T). Sender 1 und

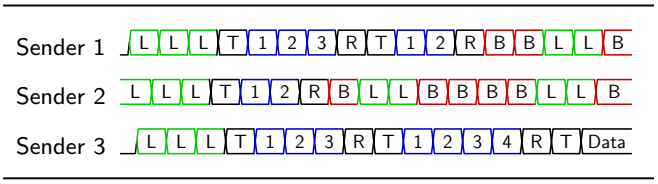


Abbildung 5: BPS-MAC mit mehreren Präambeln

3 würfeln eine Präambeldauer von 3 Zeiteinheiten (1 - 3), Sender 2 dagegen würfelt eine Präambeldauer von 2 Zeiteinheiten (1 - 2). Dadurch ist es Sender 2 möglich die Präambeln von Sender 1 und 3 auf dem Kanal festzustellen.
Sender 1 und 3 können die Präambel des jeweils anderen nicht feststellen, da beide die gleiche Präambeldauer gewählt haben.
Allerdings unterscheidet sich die Dauer der 2. Präambel von Sender 1 und 3, so dass Sender 1 die längere Präambel von Sender 3 empfängt. Durch die 2. Präambel wird in diesem Beispiel die Kollision verhindert.
Natürlich können auch hier noch Kollisionen auftreten, falls zwei Sender für jede Präambel die gleiche Dauer auswürfeln und im selben Zeitslot mit der Übertragung beginnen wollen. Allerdings sinkt die Wahrscheinlichkeit einer Kollision durch mehrere kurze Präambel im Vergleich zu einer längeren Präambel deutlich. Mit den Einstellungsmöglichkeiten für die Anzahl der Präambeln sowie der minimalen und maximalen Präambeldauer lässt sich das Protokoll für das jeweilige WSN optimal konfigurieren. Die Wahrscheinlichkeit für eine Kollision ist daher sehr gering.

4. VERGLEICH DER MAC PROTOKOLLE

In diesem Kapitel werden MAC Protokolle LPL, X-MAC und BPS-MAC bezüglich Interferenz, Auslastung (Overhead), Energieeffizienz, Paketverlust und Verzögerung miteinander verglichen.

4.1 Interferenz

Der Vergleich bezüglich der Interferenz wird in drei Teilbereichen aufgeteilt. Zuerst die klassische Kollision, bei der sich die Übertragungen von zwei oder mehr Sendern überlagern, so dass eine Unterscheidung der verschiedenen Übertragungen nicht mehr möglich ist. Außerdem wird in diesem Abschnitt noch das Hidden-Node und das Exposed-Node Problem behandelt.

4.1.1 Kollision

Kollisionen treten auf sobald ein Sender eine Übertragung beginnt obwohl bereits eine Übertragung im Gange ist. Natürlich versuchen alle 3 Protokolle eine Kollision zu vermeiden. Das LPL Protokoll benutzt eine lange Präambel um den Kanal für die darauf folgende Datenübertragung zu reservieren, so erkennt im Idealfall ein 2. Sender die Präambel und verschiebt seine eigene Übertragung nach hinten um eine Kollision mit der anderen zu vermeiden. Da aber nur eine Präambel verwendet wird, ist die Wahrscheinlichkeit einer Kollision gegenüber den anderen beiden Protokollen mit mehreren Präabeln recht hoch. BPS-MAC mit den unterschiedlich langen Präambeln bietet die höchste Wahrscheinlichkeit einer kollisionsfreien Übertragung, vor allem WSN mit höherer Knotendichte.

doi: 10.2313/NET-2011-07-1_06

Bei allen drei Protokollen steigt die Kollisionswahrscheinlichkeit mit der Anzahl der Sender und der Menge der Übertragungen.

4.1.2 Hidden-Node Problem

Das Hidden-Node Problem beschreibt folgendes Problem. Sender A und Sender B liegen in dem Sendreradius des jeweils anderen. Ebenso liegen Sender B und Sender C im gegenseitigen Senderadius. (Abbildung 6) Allerdings ist Sender C für Sender A nicht erreichbar und Sender A für Sender C nicht erreichbar. Sender A und C können also mit Sender B kommunizieren, aber nicht direkt miteinander. Daraus folgt, dass Sender C eine Kommunikation von Sender B zu Sender A erkennt, aber Sender C keine Kommunikation von Sender A zu Sender B erkennt, da sich Sender C nicht im Senderadius von Sender A befindet.

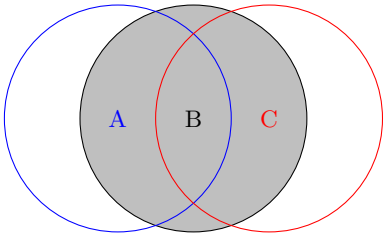


Abbildung 6: Hidden-Node Problem

Beginnt Sender C mit einer Datenübertragung, da für ihn der Kanal frei ist, aber schon eine Datenübertragung von Sender A zu B im Gange ist, kommt es zu Interferenz. Bei Sender B kommen nun gleichzeitig die Datenpakete von Sender A und Sender C an. Das Unterscheiden welche Pakete von welchem Sender kommen ist dann nahezu unmöglich. Das Ergebnis ist dann, dass weder die Übertragung von Sender A noch die von Sender C erfolgreich ist.
Für das Hidden-Node Problem bietet keines der drei vorgestellten MAC Protokolle eine zufriedenstellende Lösung, weshalb hier auch keine Empfehlung für ein bestimmtes Protokoll gegeben werden kann.
Allerdings sei angemerkt, dass ein stärkeres Signal ein schwächeres Signal oft auslöscht. Als Richtwert gilt hier eine Differenz von mehr als 3dBm um das stärkere Signal noch empfangen zu können. [2]

4.1.3 Exposed-Node Problem

Das Exposed-Node Problem beschreibt ein ähnliches Problem wie das Hidden-Node Problem, allerdings bleibt beim Exposed-Node Problem eine Übertragung aus, die eigentlich möglich wäre. Um das Exposed-Node Problem besser zu verstehen ordnen wir vier Sender so an, das jeder Sender nur seine Nachbarn erreichen kann (siehe Abbildung 7).

Sender A erreicht B, Sender B erreicht A und C, Sender C erreicht B und D, und Sender D erreicht wiederum nur Sender C. Besteht eine aktive Übertragung von Sender B zu Sender A, wird Sender C den Kanal als belegt erkennen, da sich auch Sender C im Einflussbereich von Sender B befindet. Sender C wird also keine Übertragung zu Sender D starten, da der Kanal scheinbar von Sender B belegt ist. Allerdings würde die Übertragung von Sender C zu D die andere Übertragung nicht beeinträchtigen, da Sender A nicht

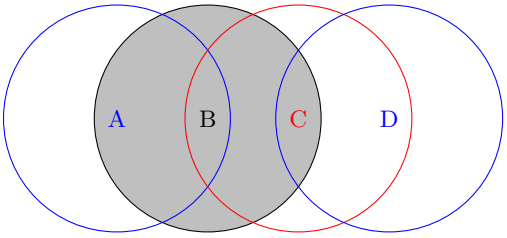


Abbildung 7: Exposed-Node Problem

im Sendebereich von Sender C liegt. Also wurde Sender A weiterhin nur die Daten von Sender B empfangen und auch Sender D wurde nur die Daten von Sender C empfangen.
Die beiden Übertragungen würden sich also nicht beeinflussen und trotzdem kommt die 2. Übertragung nicht zu stande, da Sender C den Kanal als belegt erkennt.
Auch für das Exposed-Node Problem kennt keines der 3 Protokolle eine Lösung.

4.2 Overhead

Als Overhead bezeichnet man Daten die zusätzlich zu den Nutzdaten übertragen werden. Overhead beinhaltet beispielsweise Routinginformation oder Empfängeradressen die notwendig sind um sicherzustellen, dass die eigentlichen Datenpakete auch beim Empfänger ankommen. Je größer der Overhead desto schlechter die maximale Auslastung, denn je mehr Zusatzinformationen übertragen werden müssen, umso weniger Bandbreite bleibt für die eigentlichen Daten übrig. Für ein Protokoll bedeutet dies: so wenig Overhead wie nötig.
Für WSN Protokolle ist der Overhead besonders wichtig, da hier oft mehr Overhead als Nutzdaten übertragen wird.
Vergleicht man das LPL mit dem X-MAC Protokoll wird schnell klar, dass das X-MAC Protokoll deutlich weniger Overhead produziert als das LPL Protokoll mit seiner sehr langen Präambel. Die lange Präambel des LPL kann sogar länger andauern als die darauf folgende Datenübertragung. Dieser große Overhead des LPL Protokolls führt zu einer deutlich schlechteren Auslastung des Übertragungskanals.
Obwohl das X-MAC Protokoll im Gegensatz zum LPL Protokoll schon eine deutlich bessere Auslastung bietet, eignet sich für ein höhere Übertragungsdichte BPS-MAC am besten. Da sich beim BPS-MAC Protokoll die einzelnen Sensorknoten nicht im Schlafzustand befinden, muss hier nur auf die Vermeidung von Kollisionen Rücksicht genommen werden, nicht aber auf die Aufwachzyklen. Durch das Wegfallen der Wartezeit bis der Empfängerknoten empfangsbereit ist, ist auch der Overhead beim BPS-MAC Protokoll geringer, womit sich dieses Protokoll am besten eignet, falls mit großeren Datenübertragungen zu rechnen ist.

4.3 Energieeffizienz

Wie in der Einleitung und in Kapitel 2 schon erwähnt wurde ist die Energieeffizienz oft die wichtigste Eigenschaft in WSN. Die verschiedenen Arten von Energieverschwendung wurden in Kapitel 2 schon aufgezählt.
Da das BPS-MAC Protokoll für hohes Trafficaufkommen entworfen wurde, spielt bei dem BPS-MAC Protokoll die Energieeffizienz eine untergeordnete Rolle. So lauschen alle Knoten ständig dem Funkkanal und schicken ihr Empfangsmodul nie in den Schlafzustand. Für den Energieverbrauch

doi: 10.2313/NET-2011-07-1_06

ist dieser Umstand natürlich nicht wünschenswert, deshalb schneidet das BPS-MAC Protokoll im Vergleich bezüglich der Energieeffizienz auch am schlechtesten ab.
Sowohl das LPL als auch das X-MAC Protokoll setzten auf Aufwachzyklen der Empfangsmodule. Es nutzen auch beide Protokolle Präambeln, um Datenübertragungen anzukündigen. LPL setzt dabei auf eine sehr lange Präambel, X-MAC dagegen auf mehrere Kurze. Der Vorteil der kurzen Präambeln besteht darin, dass die Präambelphase durch Senden einer Bestätigungsnachricht (Ack) des Empfängerknotens abgekürzt wird.
So wird durch die kürzere Präambelphase auf Sender- und auf Empfängerseite Energie eingespart. Außerdem enthalten die kurzen Präambeln des X-MAC Protokolls im Gegensatz zur Präambel des LPL Protokolls schon die Adresse des Empfängers, so dass alle Knoten, die nicht Empfänger sind die Präambelphase nicht abwarten müssen. Für das WSN bedeutet dies eine zusätzliche Energieeinsparung.
Im Bereich der Energieeffizienz schneidet also das X-MAC Protokoll am besten ab, weshalb es den anderen beiden Protokolle vorzuziehen ist, falls die Energieeffizienz das oberste Ziel darstellt. [1, 3]

4.4 Paketverlust

Packet Loss bezeichnet das Verhältnis zwischen den generierten und der erfolgreich übertragenen Datenpakete. Das X-MAC Protokoll bietet in fast allen Fällen eine bessere (niedrigere) Packet Loss Rate als das LPL Protokoll. [1] Die Erfolgswahrscheinlichkeit einer erfolgreichen Datenübertragung beim BPS-MAC Protokoll mit einer einzigen Präambel sinkt bereits bei einer Knotenanzahl von 10 unter 20%. Erhöht man aber die Anzahl der Präambeln des BPS-MAC Protokolls auf 5, dann beträgt die Erfolgswahrscheinlichkeit einer erfolgreichen Datenübertragung über 90 %. Liegt das Verhältnis zwischen übertragender und generierter Daten bei 10 Sendern und dem X-MAC Protokoll bei ca. 95%, erreicht das BPS-MAC Protokoll mit 5 Präambeln nahezu 100%.
Aus Sicht eines besseren Packet Loss Verhältnisses ist für dichtere Sensornetze daher sicherlich BPS-MAC die erste Wahl. [1, 3]

4.5 Verzögerung

Die Paketverzögerung spielt in WSN insoweit eine Rolle, da die Sensornetzwerke häufig eingesetzt werden um ein Ereignis zu protokollieren. Um dieses Ereignis auswerten zu können werden alle Messwerte benötigt die durch die Sensorknoten geliefert werden können. Damit die verschiedenen Messwerte zugeordnet werden können müssen sie zur gleichen Zeit ausgewertet werden. Werden nun Messwerte aufgrund der Latenz im Netz ausgebremst wird eine Zuordnung und damit verbundene Auswertung der Daten erschwert.
Wieder schneidet das LPL Protokoll am schlechtesten ab. Durch die lange Präambel wird nicht nur der eigene Messwert verzögert versendet sondern es werden auch andere Sensorknoten durch den belegten Kanal daran gehindert ihre Daten sofort weiter zu reichen.
Die verkürzte Präambelphase des X-MAC Protokolls verringert die Verzögerung deutlich. Noch weniger Latenz erreicht man mit dem BPS-MAC, da hier nicht die nächste Wachphase des Empfängers abgewartet werden muss.

5. FAZIT

Zusammenfassend kann man sagen, dass es nicht das perfekte MAC Protokoll gibt. Jedes Protokoll hat seine Vor- und Nachteile, auch wenn das BPS-MAC Protokoll in dem Vergleich als eindeutiger Sieger erscheint hat es seine größte Schwäche in der Energieeffizienz.
Da die Energieeffizienz wohl meistens das Hauptanliegen an das MAC Protokoll sein wird, greift man hier wohl besser zum X-MAC Protokoll. Das LPL Protokoll hingegen scheint nur Nachteile zu bieten und dürfte wohl in keinem Einsatzszenario die beste Wahl darstellen. Man darf dabei aber nicht vergessen, dass das LPL Protokoll das älteste der drei hier vorgestellten Protokolle ist und die anderen beiden Protokolle auf dem LPL Protokoll aufbauen.
Mittlerweile gibt es eine Vielzahl an verschiedenen MAC Protokolle, jedes für andere Anforderungen optimiert, doch selbst wenn man ein geeignetes Protokoll gefunden hat, gilt es die einzelnen Optionen, wie beispielsweise die minimale und maximale Präambeldauer im Auge zu behalten und gegebenenfalls erneut anzupassen, falls sich beispielsweise die Anzahl der Sensorknoten im Netzwerk ändert.

6. LITERATUR

[1] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks. In *SenSys '06: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, pages 307–320, New York, NY, USA, 2006. ACM.

[2] A. Kiryushin, A. Sadkov, and A. Mainwaring. Real-world performance of clear channel assessment in 802.15.4 wireless sensor networks. In *Proc. Second International Conference on Sensor Technologies and Applications SENSORCOMM '08*, pages 625–630, August 2008.

[3] A. Klein, J. Klaue, and J. Schalk. BP-MAC: a high reliable backoff preamble MAC protocol for wireless sensor networks. *Electronic Journal of Structural Engineering (EJSE): Special Issue on Sensor Network for Building Monitoring: From Theory to Real Application*, -:35–45, December 2009.

[4] K. Langedoen. The MAC alphabet soup served in wireless sensor networks, http://www.st.ewi.tudelft.nl/~koen/macsoup/, 2006.

[5] K. Langedoen. *Medium Access Control in Wireless Networks*, volume 2, chapter 20, pages 535–560. Nova Science Publishers, July 2008.

doi: 10.2313/NET-2011-07-1_06

Akustische Unterwasserkommunikation

Markus Grimm
Betreuer: Christoph Söllner
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: grimmm@in.tum.de

KURZFASSUNG

Die drahtlose Unterwasserkommunikation eröffnet ein weites Anwendungsgebiet im Bereich der zivilen und militärischen Überwachung und Erkundung der Ozeane. Sensornetze, bestehend aus unter und über Wasser ausgesetzten Sensorknoten, nutzen dabei Ultra- und Infraschall zur Kommunikation, um Daten im Wasser zu übertragen. Dabei werden sowohl stationäre Sensorknoten (Bojen und Bodenmesseinheiten) als auch mobile autonome Unterwasserfahrzeuge verwendet.
Diese Arbeit gibt einen Überblick über die Probleme, die mit der akustischen Kommunikation unter Wasser verbunden sind und stellt entsprechende Lösungsansätze vor. Es wird zudem auf die gängigen Modulationsverfahren eingegangen und anschließend ein Überblick über aktuelle Anwendungen gegeben.

Schlüsselworte

Akustische Unterwasserkommunikation, Modulationsverfahren, ASK, FSK, PSK, OFDM, Unterwasser Sensornetze, Unterwassersensoren, Tsunami-Frühwarnsystem

1. EINLEITUNG

Schallwellen sind für die Kommunikation im Ozean von hoher Bedeutung, denn im Unterschied zu elektromagnetischen Wellen absorbiert Wasser Schallwellen kaum. Elektromagnetische Wellen haben in Wasser nur eine Reichweite von wenigen hundert Metern, Schallwellen hingegen können abhängig vom Einsatzszenario Reichweiten von über 1000 km erreichen. Wegen der elektrischen Leitfähigkeit von Salzwasser benötigt man zur Kommunikation mit elektromagnetischen Wellen unter Wasser viel Energie. Nur mit relativ niedrigen Frequenzen unter 300 Hz kann über weitere Strecken kommuniziert werden [16]. Hierfür werden jedoch relativ große Antennen benötigt. Hochfrequente elektromagnetische Wellen hingegen werden schon in geringen Tiefen weitestgehend abgeschirmt. Daher finden unter Wasser in der Regel Schallwellen Anwendung zur Kommunikation, zur Ortung und zum Zwecke der Navigation [16].
Die drahtlose Kommunikation unter Wasser eröffnet eine Reihe von militärischen und zivilen Anwendungsbereichen: Sie wird eingesetzt bei der Überwachung von klimatischen, biologischen und seismographischen Veränderungen in den Ozeanen, zur Erkundung der Meeresböden u.a. für die Erschließung neuer Ölvorkommen, aber auch zur Erkennung von feindlichen U-Booten und Minen. Nicht zuletzt durch die großen Tsunami Katastrophen von 2004 und 2011 ist ein reaktionsschnelles Tsunami-Frühwarnsystem von globalem Interesse, dessen Grundlage schnelle und effiziente Kommunikationsstrukturen sind.
Oftmals werden hierfür Sensornetze eingesetzt, die aus verteilten Sensorknoten, ausgestattet mit den jeweiligen Messeinrichtungen und Vorrichtungen zur Kommunikation, bestehen, um kollaborativ Monitoring-Aufgaben auszuführen.
In dieser Arbeit werden zunächst die physikalischen Grundlagen der Ausbreitung von Schallwellen unter Wasser behandelt. Im Anschluss wird die Übertragung von Daten unter Wasser näher betrachtet und dabei auf den Aspekt der Modulation mit Hilfe eines Unterwassermodems eingegangen. In Abschnitt 4 werden darauf aufbauend die Bestandteile von Unterwassersensornetzen beschrieben. Nach der Betrachtung der Grundlagen werden zwei Anwendungen von Schallwellen unter Wasser vorgestellt und im Speziellen ein System zur Tsunami-Früherkennung vorgestellt. Als Abschluss werden die wichtigsten Aspekte bei der akustischen Unterwasserkommunikation noch einmal zusammengefasst und ein kurzer Ausblick auf die zukünftige Entwicklung in diesem Bereich gegeben.

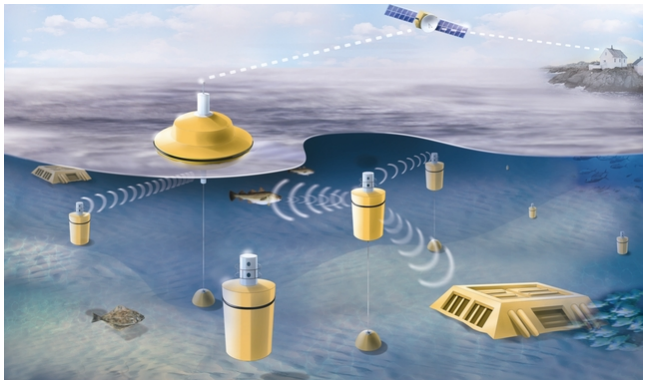


Abbildung 1: Konzeptzeichnung eines Unterwassersensornetzes [3]

2. GRUNDLAGEN DES WASSERSCHALLS

In diesem Abschnitt werden die physikalischen Grundlagen der Schallausbreitung im Wasser behandelt, um einen Überblick über die Anforderungen an Systeme zur akustischen Kommunikation im Ozean zu geben.
Schall breitet sich in Wellenform aus, unter Wasser erfolgen die Schwingungen in Ausbreitungsrichtung (Longitudinalwelle).
Der für die Kommunikation benutzte Frequenzbereich liegt zwischen 10Hz und 1 MHz [5]. Diese Frequenzen liegen in-

doi: 10.2313/NET-2011-07-1_07

nerhalb der Frequenzbereiche des Infraschalls (< 16 Hz), des Hörschalls (16 Hz bis 20 kHz) und des Ultraschalls (20kHz bis 1,6 GHz).

2.1 Schallgeschwindigkeit im Wasser

Die Geschwindigkeit von Schallwellen unter Wasser hängt im Wesentlichen von drei Faktoren ab: Druck, Wassertemperatur und Salzgehalt. Bei einer Temperatur von 20°C und unter atmosphärischem Druck beträgt die Schallgeschwindigkeit in Wasser 1484 m/s, wohingegen sich der Schall in Luft nur mit einer vergleichsweise geringen Geschwindigkeit von ungefähr 340 m/s bewegt. Bei zunehmender Tiefe (bzw. Druck), Salzgehalt oder Temperatur nimmt die Schallgeschwindigkeit zu [13].
Abhängig von der geografischen Breite und damit der Wassertemperatur in den oberen Wasserschichten wird der durch die Druckzunahme bedingte Geschwindigkeitsanstieg bis in etwa 1km Tiefe durch die Temperaturabnahme überlagert. Ab dieser Tiefe ist die Wassertemperatur konstant bei etwa 4°C [19]. Das daraus resultierende Geschwindigkeitsprofil in Abhängigkeit zur Tiefe ist schemenhaft in Abbildung 2 zu sehen.
Nach dem Brechungsgesetz wird eine Schallwelle immer in Richtung der niedrigeren Ausbreitungsgeschwindigkeit hin gebrochen. Schallwellen, die im Bereich der minimalen Schallgeschwindigkeit emittiert wurden, werden daher innerhalb dieser Wasserschicht „fixiert". Dadurch können sich die Schallwellen innerhalb dieses Kanals (dem sog. SOFAR-Kanal) über weite Strecken ausbreiten (vgl. Abbildung 2).
Direkt unterhalb der Meeresoberfläche ist die Temperatur häufig am höchsten und nimmt dann bis in ca. 50 m Tiefe relativ schnell ab. Dies hat zur Folge, dass Schallwellen, die sich unter- bzw. oberhalb dieser Grenze bewegen, aufgrund von Brechung die jeweils andere Schicht nicht erreichen können. Diese sog. akustische Schattenzone beeinflusst die Kommunikation zwischen diesen Wasserschichten stark [13].

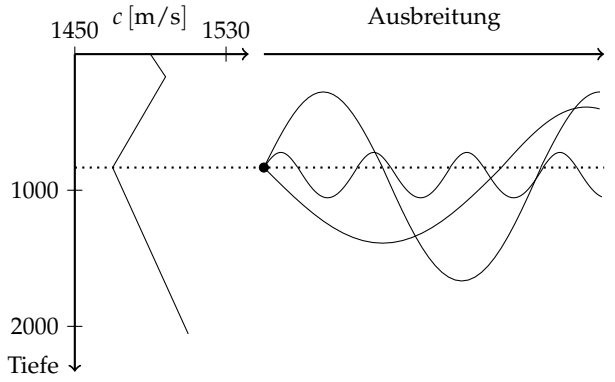


Abbildung 2: Links: Schematisiertes Geschwindigkeitsprofil, Rechts: Schallausbreitung im SOFAR Kanal, Gepunktet: SOFAR Kanal [13, 19]

Die Schallgeschwindigkeit in Wasser ist zudem von der Frequenz abhängig, wodurch es zu Dispersionserscheinungen kommt. Dispersion beschreibt die Abhängigkeit der Phasengeschwindigkeit von der Frequenz einer Welle. Dieser Effekt kann, je nach relativer Bandbreite der Signale, zu erheblichen Laufzeit- bzw. Pulsformänderungen führen [13].

2.2 Mehrwegausbreitung

Infolge von Reflexionen am Meeresboden bzw. an der Wasseroberfläche kommt es zu unterschiedlich langen Laufwegen der Signale (die sog. Mehrwegausbreitung). Die dadurch auftretenden Amplituden- und Phasenschwankungen nehmen mit wachsender Entfernung von der Schallquelle zu, was sich auf die Anforderungen an das Modulationsverfahren und die mögliche Datenrate auswirkt. Man spricht hierbei von Intersymbolinterferenz [13].

2.3 Doppler-Effekt

Bewegungen von Sender oder Empfänger führen aufgrund des Doppler-Effekts zu Frequenzverschiebungen und damit zu einer Ausdehnung der Frequenzbandbreite. Dieser Effekt tritt nicht nur bei Tauchrobotern durch kontrollierte Bewegungen auf, sondern auch bei Bojen und Unterwassermesseinheiten, die Seegang und Strömungen ausgesetzt sind. Die Größenordnung des Doppler- Effekts ist proportional zum Verhältnis der Bewegungs- zur Ausbreitungsgeschwindigkeit [18]. Aufgrund der im Vergleich zu elektromagnetischen Wellen relativ niedrigen Geschwindigkeit der Schallwellen wird das Signal durch die Frequenzverschiebung deutlich stärker verzerrt als dies z.B. bei terrestrischem Funk der Fall wäre.

2.4 Unterwasserlärm

Als Unterwasserlärm werden Schallwellen bezeichnet, die nicht direkt vom Sender erzeugt werden. Dabei handelt es sich zum einen um natürliche Umgebungsgeräusche, erzeugt durch Wind, Strömungen, Regen, seismologische Aktivitäten oder Meeresbewohner. Den größeren Teil des Unterwasserlärms stellen dabei aber künstlich erzeugte Geräusche von Schiffen, Ölplattformen oder Pumpen dar.
Unterwasserlärm ist am stärksten im niederfrequenten Bereich und nimmt mit zunehmender Frequenz ab. Je nach verwendetem Frequenzband können diese Geräusche das Signal-Rausch Verhältnis dadurch unterschiedlich stark beeinflussen [19]. Bei der Verwendung von höheren Frequenzbändern nimmt der Effekt demnach ab, was wiederum mit anderen Nachteilen wie der höheren Dämpfung verbunden ist.

2.5 Dämpfung

Die Absorptionsdämpfung (Umwandlung von Schallenergie in Wärme) lässt sich im Meerwasser zum größten Teil auf die Relaxationsdämpfung zurückführen. Relaxationsdämpfung tritt bei verzögerter Einstellung eines chemischen Gleichgewichts bei Druckänderung auf. Da im Meerwasser viele Komponenten in verschiedenen chemischen Zuständen vorliegen und sich das Verhältnis zueinander durch druckabhängige Gleichgewichtsreaktionen nur verzögert einstellt, wird der Schallwelle dadurch Energie entzogen.
Im kHz-Bereich wird dieser Effekt hauptsächlich durch Borsäure und Magnesiumsulfat hervorgerufen [13]. In [8] wurde dieser Zusammenhang experimentell bestätigt und eine empirische Formel für den Energiedämpfungskoeffizient angegeben. Abbildung 3 zeigt dessen schnellen Anstieg bei zunehmender Frequenz.
Absorption tritt im Wasser außerdem bei Reflexionen am Meeresboden, an Eis, Luftblasen oder anderen Hindernissen auf. Die Schallintensität nimmt zudem mit dem Quadrat der Entfernung zur Schallquelle ab (Divergenz). Alle diese Effekte führen zu einer Abnahme der Energie einer Schallwelle und damit deren Reichweite.

doi: 10.2313/NET-2011-07-1_07

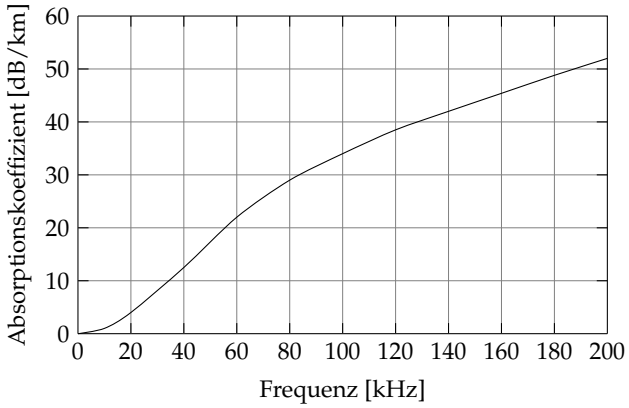


Abbildung 3: Energiedämpfungskoeffizient von akustischen Wellen im Wasser [19]

2.6 Latenzzeit

Die Übertragungsgeschwindigkeit, die mit elektromagnetischen Wellen erreicht werden kann, entspricht der Lichtgeschwindigkeit und liegt demnach bei etwa 300.000 km/s. Die damit verbundenen Latenzzeiten sind für viele Anwendungen dabei vernachlässigbar klein. Bei der akustischen Unterwasserkommunikation hingegen sind die Latenzzeiten aufgrund der sehr viel niedrigeren Geschwindigkeit der Schallwellen um ein Vielfaches größer.
Eine Angabe der *round trip time* (RTT, Laufzeit eines Datenpakets von der Quelle zum Empfänger und zurück) für ein Paket ist vor allem aufgrund der hohen Variabilität der Schallgeschwindigkeit, hervorgerufen durch die Mehrwegausbreitung, kaum möglich [16]. Daher sind viele bekannte Transportschicht-Protokolle für die Anwendung unter Wasser ungeeignet, da diese eine genaue Abschätzung der RTT benötigen.

2.7 Reichweite

Die Reichweite von Schallwellen unter Wasser hängt wie in Abschnitt 2.5 bereits angedeutet, stark von den jeweiligen Frequenzen ab. Je nach Anwendung und gewünschter Reichweite muss entsprechend das Frequenzband gewählt werden. In Referenz [5] werden die Frequenzbänder nach Reichweite in fünf Kategorien aufgeteilt, zu sehen in Tabelle 1.
Die Datenrate hängt maßgeblich von der verwendeten Frequenz ab. Daher sind für hohe Datenraten hohe Frequenzen wünschenswert, diese können jedoch nur über kurze Distanzen verwendet werden.

	Reichweite [km]	Trägerfrequenz [kHz]
Sehr lang	1000	< 1
Lang	$10 - 100$	$2 - 5$
Mittel	$1 - 10$	≈ 10
Kurz	$0.1 - 1$	$20 - 50$
Sehr kurz	< 0.1	> 100

Tabelle 1: Signalreichweiten [5]

3. NACHRICHTENÜBERTRAGUNG

Für die akustische Datenübertragung unter Wasser müssen die zuvor vorgestellten Einschränkungen und Hindernisse beachtet und adäquate Lösungen gefunden werden.
Um Nachrichten unter Wasser zu senden oder zu empfangen, benötigt man einen Wasserschallwandler (auch Unterwassermodem genannt). Als Wandlungseffekt wird vorwiegend die piezoelektrische und die magnetostriktive Wandlung genutzt [13]. Die Aufgabe eines Wasserschallwandlers besteht darin, die zu sendenden Daten auf eine Trägerfrequenz zu modulieren und die Schallwellen abzusetzen, sowie Signale zu empfangen und daraus Daten wiederherzustellen (Demodulation).
Die Anforderungen bei der Konstruktion eines Wasserschallwandlers sind hoch: Sie müssen neben hoher Beständigkeit gegen äußere Einflüsse, wie z.B. Dichtigkeit, Korrosionsbeständigkeit und Druckfestigkeit, eine gute Anpassungsfähigkeit an das Übertragungsmedium Wasser aufweisen [13]. Für eine genauere Beschreibung der Funktionsweise eines Wasserschallwandlers wird an dieser Stelle auf die Patentschrift von Troin und Cazaoulou [20] verwiesen.
Im folgenden Abschnitt werden einige gängige Modulationsverfahren und eine Einteilung der Signale vorgestellt.

3.1 Modulationsverfahren

Unter Modulation versteht man das Verändern eines Trägersignals in Abhängigkeit eines informationstragenden Signals. Dem Trägersignal wird dadurch die Information aufgeprägt, die am Empfänger möglichst ohne Verfälschung aus dem Signal rekonstruiert werden kann. Durch Modulation wird außerdem versucht, die Kanalkapazität besser auszunutzen (Multiplexing). Gerade bei der akustischen Kommunikation unter Wasser soll so die netto Datenrate erhöht werden.

3.1.1 Amplitudenumtastung

Die Amplitudenumtastung (ASK) ist ein Amplitudenmodulationsverfahren, das die Signalzustände mit Hilfe von unterschiedlichen Amplituden moduliert. Bei der binären Amplitudenumtastung (BASK) werden hierfür zwei verschiedene Amplituden verwendet, die die Zustände 0 und 1 kodieren [15]. In Abbildung 4 ist beispielhaft die Modulation der Signalfolge „1010“ dargestellt.

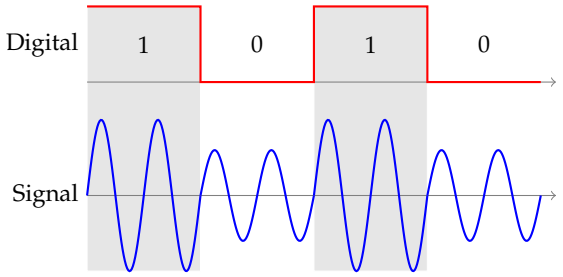


Abbildung 4: Binäre Amplitudenumtastung[15]

3.1.2 Frequenzumtastung

Bei der Frequenzumtastung (FSK) wird die Trägerfrequenz einer periodischen sinusförmigen Schwingung zwischen einem Satz unterschiedlicher Frequenzen verändert, welche die einzelnen zu sendenden diskreten Zustände (wie 0 oder

doi: 10.2313/NET-2011-07-1_07

1) darstellen. Die zu sendenden Signale werden durch Aneinanderreihung der jeweiligen Frequenzen zusammengesetzt [15]. In Abbildung 5 ist die Form der binären Frequenzumtastung (BFSK) mit zwei Zuständen für die Signalfolge „1010“ dargestellt.

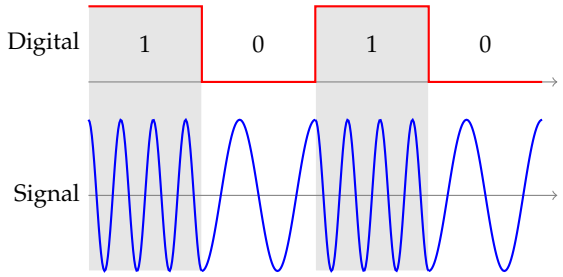


Abbildung 5: Binäre Frequenzumtastung [15]

3.1.3 Phasenumtastung

Die Phasenumtastung (PSK) nutzt zur Modulation des Nachrichtensignals Phasensprünge bzw. -verschiebungen des Trägersignals. Der einfachste Fall der binären Phasenumtastung (BPSK) lässt sich anschaulich als Multiplikation des Trägers mit ± 1 beschreiben ($+1$ für die logische 1, -1 für die logische 0 des Datensignals) [15]. Dadurch wird die Phase des Trägersignals um 0° bzw. 180° verschoben.
Das Problem beim Empfangen eines PSK-Signals ist die Phasensynchronisation. Um die Zuordnung von Phase zu Signalwert herzustellen wird beim Verbindungsaufbau ein spezielles Synchronisationswort übertragen. Andernfalls könnte bei falscher Zuordnung, wenn also der Empfänger auf der falschen Phase einrastet und damit die Phase um π verschoben interpretiert wird, die Bitfolge beim Empfänger invertiert werden [15].
Bei der Phasendifferenzmodulation, Differential Phase Shift Keying (DPSK), werden die Bits durch die Änderung der Phase kodiert. Bei der Verwendung von zwei Symbolen kodiert eine Änderung der Phase um 0° eine 0, die Änderung der Phase um 180° kodiert 1. Die Information ist hierbei in der Differenz der Phasenlagen aufeinanderfolgender Schritte kodiert, während bei der klassischen PSK die Information direkt in der Phase kodiert ist. Zwei aufeinanderfolgende Einsen würden bei der einfachen binären PSK keine Phasenverschiebung hervorrufen, während bei der binären DPSK die zweite 1 durch einen (zusätzlichen) Phasensprung um 180° kodiert wird [15].
Diese Variante der Kodierung hat zwei wesentliche Vorteile: Zum einen ist die Messung einer Phasenänderung auf Empfängerseite deutlich einfacher realisierbar, da kein gemeinsamer Referenzzeitpunkt benötigt wird. Zum anderen wird durch diese Art der Kodierung eine automatische Synchronisation auch nach Verbindungsabbrüchen gewährleistet, da die Information spätestens ab dem zweiten gelesenen Bit korrekt interpretiert wird [15].
In Abbildung 6 ist die Phasenumtastung am Beispiel der Signalfolge „0110“ abgebildet.

3.1.4 Quadraturamplitudenmodulation

Die Quadraturamplitudenmodulation (QAM) ist ein Modulationsverfahren, das durch eine Kombination von Phasenumtastung und Amplitudenumtastung realisiert ist. Das di-

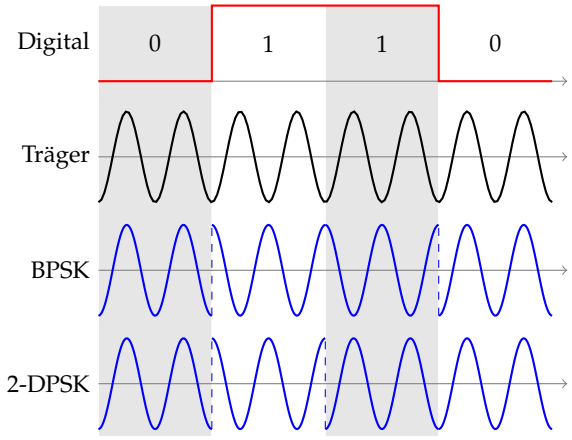


Abbildung 6: Phasenumtastung [15]

gitale Datensignal wird dabei auf zwei Signale gleicher Frequenz aufgeteilt, die addiert dem Träger (ebenfalls gleiche Frequenz) aufmoduliert werden. Die zwei Signale, die Inphase I und die zu I um 90° verschobene sog. Quadratur Q, können beide benutzt werden, um Informationen zu übertragen. Hierzu werden beide unabhängig voneinander mittels Amplitudenmodulation moduliert. Die beiden Signale stehen orthogonal zueinander und stören sich deswegen gegenseitig nicht.

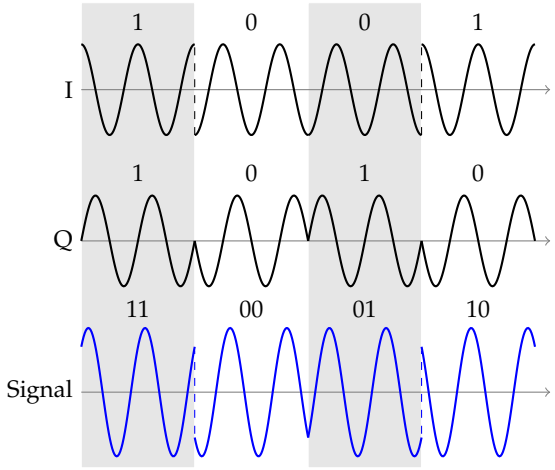


Abbildung 7: Quadraturamplitudenmodulation (4-QAM)

Zur anschaulichen Darstellung drückt man Amplitude und Phase des addierten Signals zum Zeitpunkt der Abtastung in Polarkoordinaten in einem Konstellationsdiagramm aus, wobei die Amplitude dem Abstand vom Ursprung entspricht und die Phase dem Winkel relativ zum Schnittpunkt der I/Q Achsen. Diese Art der Darstellung ist möglich, da I und Q orthogonal zueinander stehen. Die Anzahl der darstellbaren Symbole, die Punkte in der komplexen Ebene darstellen, wird in Form einer Zahl ausgedrückt. In Abbildung 8 ist ein solches Konstellationsdiagramm für die QAM mit vier Symbolen gegeben, das die Anordnung der Datensymbole zeigt. Abbildung 7 zeigt die Komponenten der 4-QAM und das addierte Signal, wobei hier zu beachten ist, dass die Modulation der Amplitude durch Multiplikation mit 1

doi: 10.2313/NET-2011-07-1_07

bzw. -1 geschieht. Prinzipiell ist es möglich, die Anzahl der darstellbaren Symbole beliebig zu erhöhen, doch erschwert dies zunehmend die Demodulation. Bei der Demodulation muss der rekonstruierte Punkt im Konstellationsdiagramm den vorher definierten Symbolen zugeordnet werden. Wird die Anzahl der darstellbaren Symbole erhöht, werden gleichzeitig die Toleranzbereiche zwischen den Symbolen verkleinert und damit eine eindeutige Zuordnung erschwert [15].

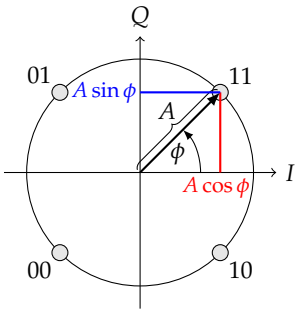


Abbildung 8: Konstellationsdiagramm für 4-QAM [15]

3.1.5 *Orthogonales Frequenzmultiplexverfahren*

Das orthogonale Frequenzmultiplexverfahren (OFDM) ist ein sog. Multicarrier-Modulationsverfahren, das mehrere orthogonale Trägersignale zur Datenübertragung verwendet. Dieses Verfahren eignet sich besonders für drahtlose Kommunikation, da es weniger anfällig für schmalbandige Störungen ist, da diese, falls sie nur einen Teil der Trägersignale betreffen, nur einen Teil der Daten betreffen. Treten Störungen bei einem Teil der Trägersignale auf, können diese zudem einfach von der Datenübertragung ausgeschlossen werden [12, 15].
Der zu übertragende Datenstrom mit hoher Datenrate wird bei der OFDM in Subdatenströme niedriger Datenrate aufgeteilt, die jeweils einzeln mit einem der zuvor vorgestellten Modulationsverfahren auf Subträger moduliert werden. Die durch die Trägersignale beschriebenen Funktionsräume der einzelnen Subsignale sind dabei orthogonal zueinander zu wählen, damit auf Empfängerseite die Trägersignale bei der Demodulation unterschieden werden können.
Das OFDM Signal wird aus den einzelnen Subsignalen durch eine komplexrechnende inverse diskrete Fouriertransformation erzeugt. Auf Empfängerseite kann das Signal anschließend mittels der schnellen Fouriertransformation wieder in die einzelnen Subsignale separiert werden [12].
OFDM wird in der Funktechnik bei WLAN, DVB-T und LTE eingesetzt. Es kommt außerdem bei ADSL zur Anwendung. Abbildung 9 zeigt eine schematische Darstellung des Modulationsvorgangs. Die zu übertragenden Daten wurden hier bereits in Subsignale aufgeteilt und einzeln auf die Trägerfrequenzen f_1 bis f_n moduliert.

3.2 Signaltypen

Datendurchsatz und Zuverlässigkeit eines Unterwassersensorknotens werden durch die physikalischen Einschränkungen stark limitiert. Demnach ist es sinnvoll, die Anforderungen an ein solches System je nach Anwendung zu klassifizieren. In [6] wird eine solche Einteilung der zu übertragenden

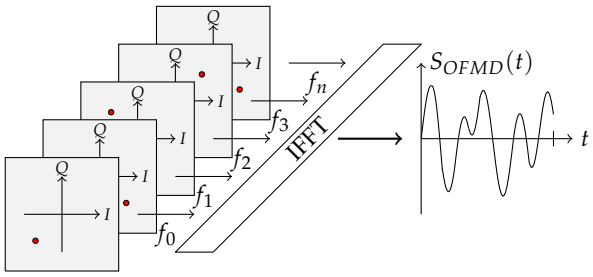


Abbildung 9: Schematische Darstellung einer Modulation mit OFDM [1]

Signale in Kontroll-, Messwert- und Videosignale vorgenommen, die im Folgenden näher betrachtet wird:

3.2.1 *Kontrollsignale*

Als Kontrollsignale werden Signale zur Steuerung, Navigation und Statusübertragung bezeichnet. Sie müssen mit hoher Zuverlässigkeit übertragen werden, benötigen aber eine geringe Bandbreite von unter 1 kBit/s.

3.2.2 *Messwertsignale*

Für die Übertragung von Messdaten und Bildern mit niedriger Auflösung benötigt man hingegen Bandbreiten bis etwa 10 kBit/s. Hierbei sind geringe Übertragungsfehler im Bereich von 10^{-4} bis 10^{-3} vertretbar.

3.2.3 *Videosignale*

Die Übertragung von hochauflösenden Bild- und Videodaten benötigt die größte Bandbreite. Bei einer Auflösung von einem Megapixel ($\approx$ 8 MBits) benötigt die Übertragung des Bildes selbst bei einer Bandbreite von 500 KBit/s mehrere Sekunden. In diesem Fall ist eine Fehlerrate bis 10^{-4} akzeptabel.

4. SENSORKNOTENSYSTEME

Sensorknotensysteme zur Überwachung von Aktivitäten im Ozean bestehen aus einer Kombination von verschiedenen Sensor- und Kommunikationseinheiten, wie in Abbildung 10 beispielhaft zu sehen ist.

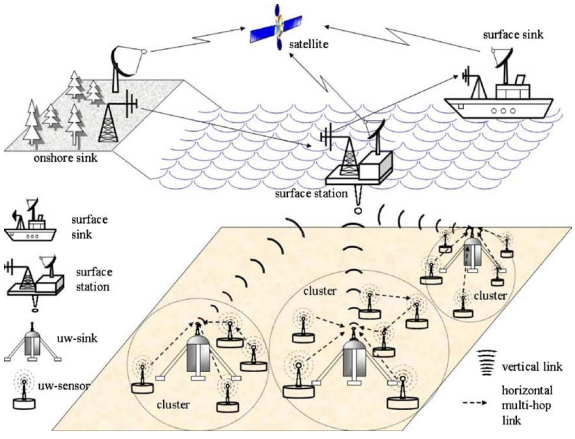


Abbildung 10: Architektur eines Unterwassersensornetzwerks [5]

doi: 10.2313/NET-2011-07-1_07

4.1 Unterwasser Sensorknoten

Unterwasser Sensorknoten werden häufig am Meeresgrund verankert und kommunizieren direkt oder über einen speziellen Sensorknoten mit einer an der Oberfläche schwimmenden Plattform, welche wiederum Daten per Satellit oder Funk zur Auswertung an Land oder Schiff weitergibt.
Es gibt eine Vielzahl von Anwendungsfällen, für die der Einsatz von Sensorknoten am Meeresboden in Frage kommt. Ein Sensorknoten kann z.B. eingesetzt werden, um Temperatur, Wasserqualität, Strömungsverhalten oder seismologische Aktivitäten zu messen.
Die Stromversorgung eines Unterwassersensorknotens wird durch Batterien gewährleistet. Eine eigene Stromversorgung mit Solarpanelen ist aufgrund der Dunkelheit am Meeresboden nicht möglich. Bei niedrigem Ladezustand oder für Wartungsarbeiten kann der Sensor zudem über einen speziellen Bergungsmechanismus zum Auftreiben gebracht werden [5].

4.2 Bergungsmechanismus

Der Bergungsmechanismus dient dazu, den Sensor vom Anker zu trennen und zum Auftreiben zu bringen. Dieser Vorgang wird in der Regel aus der Ferne aktiviert und funktioniert ebenfalls akustisch, meist mittels eines frequenzmodulierten Signals, da diese Art der Modulation bei niedriger Datenrate am zuverlässigsten ist [11]. Es gibt verschiedene Ansätze, wie der Sensor zuverlässig vom Anker getrennt werden kann. In [9] ist ein Mechanismus beschrieben, der hierfür den verbindenden Draht zwischen Anker und Sensor mit einer hohen Spannung zum Schmelzen bringt. Der Vorteil dieser Variante ist, dass sie ohne Motor auskommt.

4.3 Autonome Unterwasserfahrzeuge

Als Ergänzung zu den mehr oder weniger statisch verankerten Sensorknoten wird in den letzten Jahren verstärkt auf die Entwicklung von autonomen Unterwasserfahrzeugen (AUV) gesetzt. Ihr Vorteil liegt in dem relativ großen Arbeitsbereich, der nur durch die Reichweite des akustischen Signals beschränkt wird [10, 17]. Durch den Einsatz von autonomen Unterwasserfahrzeugen entstehen zusätzliche Anforderungen an die Kommunikationstechnik, da hier der Doppler-Effekt stärker zum Tragen kommt [19]. Abbildung 11 zeigt ein AUV der Firma Saab Seaeye, das zum Umwelt-Monitoring eingesetzt wird.

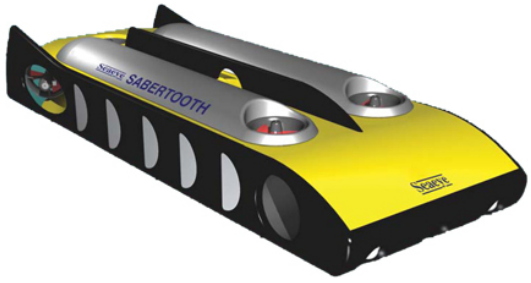


Abbildung 11: AUV *Seaeye Sabertooth* [2]

4.4 Unterwasser Sensornetze

Die Architektur eines Unterwasser Sensornetzes hat große Auswirkungen auf die Faktoren Energieverbrauch, Kapazität und Zuverlässigkeit [5]. Die Zuverlässigkeit ist aufgrund der oft hohen Kosten eines Unterwassersensornetzes hierbei das entscheidende Kriterium. In [5] werden drei verschiedene Referenzarchitekturen vorgestellt, die je nach Anwendung diese Faktoren unterschiedlich gewichten.
Der Datenverkehr innerhalb eines Sensornetzes nimmt mit der Anzahl der beteiligten Sensorknoten rapide zu. Daher sind effiziente Routing- und Kollisionsverhinderungsmechanismen nötig, um den Datenverkehr in dem gemeinsamen Übertragungsmedium Wasser zu regeln. Häufig werden dazu Routingknoten eingesetzt, die horizontal mit den benachbarten Sensorknoten kommunizieren und die gesammelten Daten vertikal an die über der Wasseroberfläche liegenden Sendeplattform weiterleiten [16].

5. ANWENDUNGEN

Die Anwendungsgebiete der Kommunikation unter Wasser sind vielseitig. Seit Beginn des 20. Jahrhunderts und der Erfindung der ersten Wasserschallwandler wurde sich intensiv mit der akustischen Kommunikation auseinandergesetzt.
Den Anstoß für die größten Entwicklungen gab jedoch der zweite Weltkrieg. Zu dieser Zeit wurde das Sonar zur Ortung von U-Booten entwickelt. Außerdem entstand das erste System zur Unterwassertelefonie (Deckname „Gertrude“). Weitere Geräte zur Ortung und Navigation folgten. Neben dem Einsatz zur militärischen Kommunikation und Ortung ist die praktische Anwendung von Schallwellen unter Wasser heute auch im zivilen Bereich von großem Interesse. Im Folgenden werden zwei Systeme, die sich die Schallausbreitung unter Wasser zu Nutze machen, vorgestellt.

5.1 Sound Surveillance System

Das SOSUS (*SOund SUrveillance System*) ist ein US amerikanisches Geräuschüberwachungssystem, ursprünglich entwickelt zur Überwachung von U-Boot Aktivitäten in den Ozeanen. Es handelt sich hierbei um ein Netzwerk aus fest installierten und verkabelten Unterwassersensoren, die, ausgestattet mit Unterwassermikrofonen, Geräusche aufnehmen und diese per Kabel zur Auswertung an Land übertragen. Die hochsensiblen Sensoren erlauben es, Geräusche mit einer akustischen Leistung von weniger als einem Watt über mehrere hundert Kilometer Entfernung zu entdecken. Nach dem Ende des kalten Krieges wurden Teile des Systems abgeschaltet. Einige Unterwassersensoren werden weiterhin genutzt um Aktivitäten von Walen zu verfolgen [4].

5.2 Tsunami Frühwarnsystem GITEWS

Das deutsch-indonesische System zur Tsunami Früherkennung GITEWS ist ein komplexes System verschiedener Sensortypen wie Seismometer, Ozeaninstrumenten und GPS-Sensoren, das seit 2008 im indischen Ozean eingesetzt wird.

5.2.1 Systemkomponenten

Um schnell und zuverlässig eine sich ausbreitende Tsunami-Welle erkennen zu können, wurden eine Reihe unterschiedlicher Systeme kombiniert. An Land erfassen Seismometer und GPS-Stationen kleinste Bewegungen der Kontinentalplatten. Dadurch kann innerhalb kürzester Zeit die Stärke, die Bruchrichtung und das Epizentrum einer Erdverschiebung bestimmt werden, was für die Bewertung des Tsunami-Risikos von essenziellem Wert ist. Auf hoher See erfassen Messbojen und Unterwasser-Drucksensoren Erderschütterungen und Veränderungen des Wasserdrucks. Die Unterwassersensoren kommunizieren akustisch mit Oberflächen-

doi: 10.2313/NET-2011-07-1_07

bojen, die die gesammelten Daten per Satellitenverbindung zur Auswertung an Land übertragen. Zusätzlich registrieren Detektoren in Küstennähe Veränderungen des Wasserstands, um eine Vorhersage treffen zu können, wo die Woge auf die Küste treffen wird [7].

5.2.2 *Unterwassersensoren*

Für die Erkennung einer sich ausbreitenden Tsunami-Flutwelle misst ein Unterwassersensor den Druck der auf ihm lastenden Wassersäule, um dadurch den Wellengang an der Oberfläche zu bestimmen. Hierfür werden präzise Drucksensoren benötigt, die selbst aus einer Tiefe von mehreren Kilometern den Wellengang zentimetergenau bestimmen können. Denn eine Tsunami-Welle hat die Eigenschaft, dass sie im tiefen Wasser oft nur einige Zentimeter hoch ist und dabei eine Wellenlänge von 100 bis 500 km besitzt [14].
Zu diesem Zwecke wurden für das GITEWS Projekt zwei unabhängige Bodensensoren (PACT und OBU) entwickelt und in seismologisch kritischen Gebieten wie z.B. an den Kontinentalplattenrändern ausgesetzt. In der Nähe einer jeden Bodeneinheit wurde zusätzlich eine Oberflächenboje installiert. Zur Kommunikation verwenden alle drei Systeme (PACT, OBU und Boje) das Unterwassermodem *Ham.node* der Firma Develogic. Das Modem verwendet zur Modulation je nach Anwendung entweder OFDM-mDPSK für hohe Datenraten oder n-mFSK (nicht kohärente Variante der FSK) für niedrigere Datenraten mit höherer Zuverlässigkeit. Die Aufgabe des PACT Systems ist es, den Wasserdruck am Meeresboden zu messen. Die Energiereserven betragen dafür ca. 3000 Wh. bei einer vorraussichtlichen Betriebsdauer von 29 Monaten. Das OBU System misst zusätzlich zum Wasserdruck seismologische Aktivitäten und speichert diese auf einer internen Festplatte. Die Energiereserven von 12500 Wh. ermöglichen eine Laufzeit von etwa 12 Monaten [7].
PACT und OBU messen beide alle 15 Sekunden den Druck der Wassersäule und übertragen die gesammelten Daten im regulären Betrieb alle 4 bzw. 6 Stunden. Wird vom System ein Tsunami erkannt, werden automatisch alle zwei Minuten Daten übertragen. Die von OBU gesammelten seismologischen Daten können extern getriggert in Echtzeit übertragen werden. OBU verwendet hierfür OFDM-mDPSK (11.2-19.2 kHz) zur Modulation, PACT hingegen n-mFSK (9-12.8 kHz) aufgrund der niedrigeren Datenmenge (100 Byte pro Nachricht im Gegensatz zu 50 bis 500 KB bei OBU).

6. ZUSAMMENFASSUNG

Zur Erforschung der Tiefen unserer Ozeane benötigt man zuverlässige und leistungsfähige Kommunikationsmittel. Für die drahtlose Kommunikation eignet sich die Kommunikation über elektromagnetische Wellen hierfür nur bedingt, da diese nur begrenzte Reichweiten besitzen. Eine andere und besser geeignete Möglichkeit der Kommunikation besteht in der Anwendung von akustischen Signalen, die in dieser Arbeit näher betrachtet wurden.
Die physikalischen Grundlagen und limitierenden Faktoren wurden zu Beginn vorgestellt, ebenso wie die Einflussfaktoren auf die Reichweite und Geschwindigkeit der akustischen Signale. Durch die vielen unerwünschten Einflüsse auf die Qualität der Übertragung bestehen besondere Anforderungen an die Art der Signalübertragung. Hierfür wurden mehrere Modulationsverfahren (ASK, FSK, PSK, QAM, OFDM) vorgestellt, die alle unter Wasser ihre Anwendung finden. Um das Thema abzuschließen wurden zwei Anwendungsszenarien von Unterwassersensornetzen kurz vorgestellt. Die akustische Kommunikation wird auch in Zukunft eine wichtige Rolle bei der Kommunikation unter Wasser spielen. Gerade im Zusammenhang mit dem globalen Klimawandel ist die Beobachtung der Ozeane von großem Interesse.
Nicht betrachtet wurden im Zuge dieser Arbeit Verfahren zur Zugriffskontrolle und Kollisionsverhinderung. Der interessierte Leser sei an dieser Stelle auf den Artikel [16] verwiesen, der sich mit diesen Themen und anderen Aspekten, die beim Design von Netzwerkprotokollen unter Wasser von Bedeutung sind, beschäftigt.

7. LITERATUR

[1] *An Introduction to Orthogonal Frequency Division Multiplex Technology*. `http://ieee.li/pdf/viewgraphs/introduction_orthogonal_frequency_division_multiplex.pdf`, 7 2011

[2] *Autonomous Undersea Vehicle Applications Center*. `http://auvac.org/configurations/view/95`, 7 2011

[3] *Underwater Acoustic Network*. `http://www.ua-net.eu/?q=gallery&g2_itemId=33`, 7 2011

[4] ABILEAH, R. ; MARTIN, D. ; LEWIS, S. ; GISINER, B. : Long-range acoustic detection and tracking of the humpback whale Hawaii-Alaska migration. In: *OCEANS '96. MTS/IEEE. 'Prospects for the 21st Century'. Conference Proceedings* Bd. 1, 1996, S. 373 –377 vol.1

[5] AKYILDIZ, I. F. ; POMPILI, D. ; MELODIA, T. : Underwater acoustic sensor networks: research challenges. In: *Ad Hoc Networks* 3 (2005), Nr. 3, S. 257 – 279

[6] BAGGEROER, A. : Acoustic telemetry–An overview. In: *IEEE Journal of Oceanic Engineering* 9 (1984), 10, Nr. 4, S. 229 – 235

[7] BOEBEL, O. ; BUSACK, M. ; FLUEH, E. R. ; GOURETSKI, V. ; ROHR, H. ; MACRANDER, A. ; KRABBENHOEFT, A. ; MOTZ, M. ; RADTKE, T. : The GITEWS ocean bottom sensor packages. In: *Natural Hazards and Earth System Sciences* 10 (2010), 8, Nr. 8, S. 1759–1780

[8] BREKHOVSKIKH, L. ; LYSANOV, I. : *Fundamentals of ocean acoustics*. AIP Press/Springer, 2003 (AIP series in modern acoustics and signal processing)

[9] FLAGG, M. ; GOLDSTEIN, A. ; HARVEY, A. ; HOLTMEIER, G. : *Device for remotely decoupling coupled objects with a fusible link underwater*. 11 2006

[10] HYAKUDOME, T. : Design of Autonomous Underwater Vehicle. In: *International Journal Of Advanced Robotic Systems* 8 (2011), Nr. 1, S. 131 – 139

[11] KILFOYLE, D. ; BAGGEROER, A. : The state of the art in underwater acoustic telemetry. In: *Oceanic Engineering, IEEE Journal of* 25 (2000), 1, Nr. 1, S. 4 –27

[12] KNÖRZER, S. : *Funkkanalmodellierung für OFDM-Kommunikationssysteme bei Hochgeschwindigkeitszügen*. Univ.-Verl. Karlsruhe, 2009 (Karlsruher Forschungsberichte aus dem Institut für Hochfrequenztechnik und Elektronik)

[13] LERCH, R. ; SESSLER, G. ; WOLF, D. : Unterwasserschall (Hydroakustik). In: *Technische Akustik*. Springer Berlin Heidelberg, 2009, S. 537–571

[14] LEVIN, B. ; NOSOV, M. : *Physics of tsunamis*. Springer, 2008. – ISBN 9781402088551

doi: 10.2313/NET-2011-07-1_07

[15] MEYER, M. : *Kommunikationstechnik*. Vieweg, 2002. – ISBN 9783528138653

[16] POMPILI, D. ; AKYILDIZ, I. : Overview of networking protocols for underwater wireless communications. In: *IEEE Communications Magazine* 47 (2009), 1, Nr. 1, S. 97 –102

[17] SOZER, E. ; STOJANOVIC, M. ; PROAKIS, J. : Underwater acoustic networks. In: *IEEE Journal of Oceanic Engineering* 25 (2000), 1, Nr. 1, S. 72 –83

[18] STOJANOVIC, M. : Underwater Acoustic Communications: Design Considerations on the Physical Layer. In: *Wireless on Demand Network Systems and Services, 2008. WONS 2008. Fifth Annual Conference on*, 2008, S. 1 –10

[19] STOJANOVIC, M. ; PREISIG, J. : Underwater acoustic communication channels: Propagation models and statistical characterization. In: *Communications Magazine, IEEE* 47 (2009), 1, Nr. 1, S. 84 –89

[20] TROIN, G. ; CAZAOULOU, C. : *Underwater acoustic transmission method and equipment to improve the intelligibility of such transmissions*. 11 1996

doi: 10.2313/NET-2011-07-1_07

Collection Tree Protocol

Christan Dietz
Betreuer: Dr. Alexander Klein
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: dietzc@in.tum.de

KURZFASSUNG

Das Sammeln von Messdaten gehört zu den Hauptaufgaben von drahtlosen Sensornetzwerken. Die von den Sensorknoten erfassten Messwerte sollen möglichst vollständig und über mehrere Hops zu einer zentralen Sammeleinheit weitergeleitet werden. Die in drahtlosen Sensornetzwerken auftretenden Beeinträchtigungen, wie etwa stark asymmetrische Verbindungsqualitäten und häufige Topologieänderungen durch Knotenmobilität oder -ausfälle, stellen hierbei hohe Anforderungen an das Routingprotokoll. Vorliegende Arbeit stellt das Collection Tree Protocol (CTP) vor, ein für dieses Einsatzgebiet entwickeltes Routingprotokoll, welches die oben genannten Probleme adressiert. Ausgehend von der populären CTP-Implementierung des Sensorknotenbetriebssystems TinyOS wird die Funktionsweise von CTP detailliert erläutert.

Schlüsselworte

Collection, CTP, Routing, Sensornetze

1. EINLEITUNG

Drahtlose Sensornetzwerke finden aufgrund sinkender Kosten und Fortschritte in der Forschung immer weitere Verbreitung. Besonders im Bereich der Überwachung bieten sich zahlreiche Anwendungsgebiete: Sie werden beispielsweise genutzt, um die Wasserqualität in einem großem Gelände zu kontrollieren [5] oder um den Stromverbrauch in Bürogebäuden zu überprüfen [11]. Tierforscher erlangen mithilfe drahtloser Sensornetzwerke Aufschluss über das Gruppenverhalten von wilden Tieren [14].

Bei vielen dieser Einsatzgebiete wird eine große Anzahl an Sensorknoten, die in regelmäßigen Abständen Messungen vornehmen, im zu untersuchenden Bereich verteilt. Diese Messwerte müssen anschließend an zentrale Knoten weitergereicht werden, da diese mehr Ressourcen zur Verfügung haben, um die Daten zu verarbeiten und zu speichern. Wegen der großen räumlichen Ausdehnung der Senornetzwerke muss der Übertragungsweg mehrere Zwischenknoten beinhalten, welche die Nachrichten weiterleiten. Aufgabe des Routingprotokolls ist es nun, möglichst kostengünstige und stabile Pfade zu finden und zu verwalten.

Viele Gegebenheiten in einem drahtlosen Sensornetzwerk erschweren das: Da die Sensorknoten aus Gründen des Stromverbrauchs und wegen der Anschaffungskosten nur Funkmodule mit niedriger Leistung besitzen, sind die Verbindungsqualitäten ständigen Schwankungen unterworfen. Knoten können auch ausfallen, oder wechseln zeitweise in einen Schlafmodus. In vielen Sensornetzen sind die Knoten zudem mobil. Dies führt zu häufigen Änderungen der Topologie, auf die das Routingprotokoll zeitnah reagieren muss.

Das in dieser Arbeit erläuterte CTP geht durch mehrere Ansätze auf diese Probleme ein: Es verwendet den modernen Four-Bit Link Estimator um möglichst gute Abschätzungen über Verbindungsqualitäten zu erhalten und so für stabile Routen zu sorgen. Durch eine variable Beaconrate wird versucht, den Control Overhead gering zu halten, um die Luftschnittstelle zu entlasten. Interferenzen werden zusätzlich durch künstliche Verzögerungen beim Weiterleiten von Nachrichten gemieden. CTP besitzt des Weiteren einen Mechnismus, um Routingschleifen zu erkennen und aufzulösen.

CTP ist mittlerweile weit verbreitet und in vielen Betriebssystemen für Sensorknoten verfügbar. Dazu zählen unter anderem Contiki OS [3], Mantis OS [1] und TinyOS [6]. Die verschiedenen Implementierungen unterscheiden sich aber in Details. Deshalb behandelt diese Arbeit speziell die CTP-Implementierung von TinyOS, da es sich um die bekannteste handelt. Beschrieben wird das Verhalten von CTP in TinyOS 2.1.1, die aktuellste Version zum Zeitpunkt der Anfertigung dieser Arbeit.

Die Arbeit gliedert sich folgendermaßen: Zuerst wird in Kapitel 2 vorgestellt, welche Netzwerktopologie CTP bildet und welche Nachrichten dabei ausgetauscht werden müssen, um diese zu erreichen. Kapitel 3 beschreibt, auf welche Art und Weise CTP die Kosten von Verbindungen schätzt und Kapitel 4 behandelt die Routenauswahl. Das Senden und Weiterleiten von Datenpaketen wird in Kapitel 5 erläutert. Das darauf folgende Kapitel 6 erklärt, wie CTP auf Topologieänderungen reagiert. Kapitel 7 fasst zwei Leistungsbewertungen von CTP zusammen. Ein Fazit bildet in Kapitel 8 den Schluss.

2. METRIK/BEACONS

CTP ist ein proaktives Distance Vector Routing Protokoll. Jeder Knoten wählt aus seinen Nachbarn einen Elternknoten so aus, dass seine Kosten zum Wurzelknoten möglichst gering sind. Die Kosten eines Knotens resultieren aus den Kosten des Elternknotens plus denen der Verbindung zwischen den beiden [10]. Alle Wurzelknoten haben die Kosten 0. Dadurch baut CTP eine baumartige Topologie auf. Es können auch mehrere Wurzelknoten existieren, wodurch sich ein Wald von Bäumen ergibt. Datenpakete werden entlang dieses Kostengefälles zu den Wurzelknoten weitergeleitet. Jeder Knoten nimmt Pakete von Kindknoten entgegen und schickt sie an seinen Elternknoten [4]. Wurzelknoten selbst leiten keine Nachrichten im Sensornetzwerk weiter, sondern

doi: 10.2313/NET-2011-07-1_08

fungieren als reine Datensenken. Sie besitzen auch keine Elternknoten. Es gibt in CTP keine Möglichkeit, bestimmte Datensenken direkt zu addressieren. Die Daten werden zu dem, bezogen auf den Kostenabstand, nächsten Wurzelknoten geleitet. Es handelt sich in diesem Sinne um ein Anycast-Protokoll. Unterschiedliche Instanzen des gleichen Datenpaketes können aber auch mehrere Wurzelknoten erreichen, falls es zu Paketduplikaten kommt. CTP garantiert hier weder, dass alle Pakete an den Datensenken ankommen, noch, dass sie nur an jeweils einer Datensenke ankommen. Obwohl CTP diese Verlässlichkeit nicht bietet, wird versucht, möglichst viele Datenpakete auszuliefern und Paketduplikate zu unterdrücken (vgl. Abschnitt 5).
Als Kostenmetrik verwendet die TinyOS-Implementierung von CTP Expected Transmissions (ETX), also die Anzahl der Sendeversuche, die im Mittel durchgeführt werden müssen, damit eine Nachricht erfolgreich übertragen wird.

2.1 Beacons

Durch regelmäßige Beacons informieren Knoten Nachbarn über ihren Elternknoten und ihre Kosten. Solche Routing Frames, wie in Abbildung 1 gezeigt, enthalten den gewählten Next Hop (parent) und die Kosten bis zum Wurzelknoten (ETX).

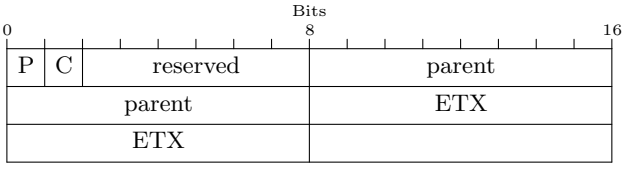


Abbildung 1: CTP Routing Frame

Nachbarknoten, die solche Beacons empfangen, fügen das Tupel (Knotenadresse, parent, ETX) in ihre Routingtabelle ein oder aktualisieren die Werte parent und ETX, falls es für diese Knotenadresse schon einen Eintrag in der Tabelle gibt. Knoten, die noch keine gültige Route besitzen, setzen das „parent"-Feld in ihren Beacons auf INVALID_ADDR (0xffff).
Das Pull-Bit (P) wird von Knoten gesetzt, welche keinen Elternknoten besitzen, also neu zum Netzwerk hinzukommen oder kurzzeitig von den anderen Sensorknoten isoliert waren. Sie fordern damit Nachbarknoten, die den Beacon mit dem gesetzten Pull-Bit empfangen, auf, ihre Beaconrate zu erhöhen, damit der isolierte Knoten schnell in das Netzwerk eingegliedert werden kann.
Das Congestion-Bit (C) bot die Möglichkeit, Nachbarknoten über eine Überlastung zu informieren. Es wurde von Knoten gesetzt, deren Sendepuffer zu viele Pakete enthielt. Knoten mit gesetztem C-Bit wurden nicht als Elternknoten gewählt. Dies hatte allerdings, wie auf der TinyOS-Mailingliste diskutiert, insgesamt negative Folgen auf die Leistung des Routingprotokolls, weshalb das C-Bit in der aktuellen TinyOS-Version 2.1.1 ignoriert wird.

2.2 Trickle

Im Gegensatz zu anderen Collection-Protokollen wie MultiHopLQI [2] versendet CTP Beacons nicht in gleichmäßigen Zeitabständen, sondern passt das Beaconintervall dynamisch an die Gegebenheiten im Netzwerk an [10]. Um die Zeitpunkte für Routing Frame Broadcasts zu bestimmen, verwendet CTP eine Abwandlung des Trickle-Algorithmus [13]. Der Zeitabstand zwischen den Beacons schwankt zwischen τ und $2 \cdot \tau$. Die genaue Intervalldauer wird jedes Mal zufällig gewählt, um eine ungewollte Synchronisation der Sensorknoten zu vermeiden. τ wird nach jedem Beacon verdoppelt, bis es den Wert τ_h erreicht, wodurch die Zeit zwischen den Beacons exponentiell ansteigt und die Beaconrate abnimmt. In TinyOS bewegt sich das Intervall τ zwischen $\tau_l = 64ms$ und $\tau_h = 256s$. Abhängig vom eingesetzten Sensornetzwerk muss darauf geachtet werden, dass der Minimalwert τ_l nicht zu niedrig gewählt ist. Bei dichter Knotenverteilung kann ein zu kleiner Wert von τ_l viele Kollisionen hervorrufen, was die Leistung des Netzwerkes negativ beeinflusst. Richtig eingestellt hat das dynamische Beaconintervall den Vorteil, dass wenig Control Overhead erzeugt wird, nachdem sich das Netzwerk stabilisiert hat. Um dennoch schnell auf Topologieänderungen reagieren zu können, wird τ bei bestimmten Ereignissen auf den ursprünglichen Wert τ_l zurückgesetzt.
Dies geschieht, wenn ein Routing- oder Data-Frame empfangen wird, bei dem das Pull-Bit gesetzt ist. Das hilft, die Routingtabelle des anfragenden Knoten zeitnah zu füllen, damit dieser einen Next-Hop wählen kann. Solange Knoten keinen gültigen Elter besitzen, setzen sie nicht nur das Pull-Bit in ihren Paketen, sondern halten τ auch konstant auf dem Minimalwert τ_l.
Knoten setzen das Beaconintervall auch zurück, wenn sie Inkonsistenzen oder Änderungen in der Netzwerktopologie feststellen, um das Netzwerk schnell wieder in einen stabilen Zustand zurückzuführen. CTP reagiert auf diese Weise, wenn Schleifen erkannt werden oder die Kosten eines Knoten drastisch sinken.

3. LINK ESTIMATION

Der Link Estimator ist in CTP dafür zuständig, die Kosten von Verbindungen zwischen Knoten zu schätzen. Diese Kosten werden auch *1-hop*-ETX, ETX_{1hop} [4] oder single-hop ETX [6] genannt. Der ETX-Wert einer Verbindung, bei der keine Pakete verloren gehen ist 1. Für eine Verbindung mit 50% Erfolgswahrscheinlichkeit auf Hin- und Rückweg gilt $\text{ETX}_{1hop} = \frac{1}{0.5 \cdot 0.5} = 4$. CTP verwendet sowohl Beacons (ETX_b) als auch Unicastpakete (ETX_d), um diese Kosten zu schätzen. Beide Schätzungen werden nicht direkt als Wert von ETX_{1hop} übernommen. Damit kurzzeitige Schwankungen in der Verbindungsqualität nicht zu drastischen Änderungen der Kosten führen, wird der Prozess geglättet, indem der alte ETX-Wert mit einfließt:

$$\text{ETX}_i = \alpha \cdot \text{ETX}_{i-1} + (1 - \alpha) \cdot \text{ETX}_x \qquad (1)$$

ETX_i steht hier für den *1-hop*-ETX-Wert nach der i-ten Aktualisierung und ETX_x für Kostenschätzungen entweder durch Beacons (ETX_b) oder durch Datenpakete (ETX_d). Der Gewichtungsfaktor α hat in TinyOS den Wert 0,9.
Beacon-basierte und Datenpaket-basierte ETX-Schätzungen beeinflussen die *1-hop*-ETX durch Gl. 1 auf gleiche Weise. Allerdings aktualisieren sie ETX_{1hop} unterschiedlich oft. ETX_b wird nach Ankunft einer gewissen Anzahl an Beacons und ETX_d nach dem Senden von Datenpaketen erneuert. In einem stabilen Netzwerk mit niedriger Beaconrate überwiegt der Datenverkehr und ETX_d hat dann mehr Einfluss auf die Kostenschätzung als ETX_b [6]. Untersuchungen

doi: 10.2313/NET-2011-07-1_08

haben ergeben, dass die Schätzung der Verbindungskosten durch Unicastpakete eine deutliche Verbesserung gegenüber der reinen Beacon-basierten Schätzung darstellt. Sowohl im Hinblick auf Paketverlust [15] als auch auf die durchschnittliche Anzahl der Hops, die ein Paket auf dem Weg zum Wurzelknoten passiert [7].
Der Link Estimator verwaltet eine Tabelle mit bekannten Nachbarknoten und den *1-hop*-ETX-Werten der dazugehörigen Verbindungen.

3.1 Linkestimation durch Beacons

Sensorknoten berechnen ihre Eingangsqualitäten durch Beacons, die sie von Nachbarknoten empfangen und Beacons, die verloren gehen. Die Eingangsqualität eines Knotens K_1 bezüglich des Knotens K_2 ist gleichzeitig die Ausgangsqualität von K_2 bezüglich K_1 [8]. Die Eingangsqualität wird durch das Link Estimation Exchange Protocol (LEEP) [8] benachbarten Knoten mitgeteilt, sodass diese ihre Ausgangsqualität kennen. Dazu werden an CTP Routing Frames ein LEEP-Header und ein LEEP-Footer angefügt, wie Abbildung 2 zeigt. Der CTP Beacon wird also immer als Payload in einem LEEP Frame übertragen.

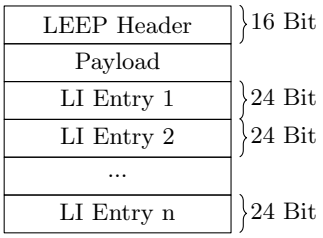


Abbildung 2: LEEP Frame

Der LEEP Header (Abbildung 3) enthält neben einer Sequenznummer (seqno) die Anzahl (nentry) der angefügten Link Information Entries (Abbildung 4). Der Footer besteht aus 0 bis n LI Entries, wobei n durch die maximale Paketgröße auf Layer 2 limitiert ist: Der gesamte LEEP Frame darf nicht größer sein als die maximale Payloadlänge von Verbindungsschichtpaketen [8]. In jedem dieser LI Entries wird die Adresse eines Nachbarknoten (node id) und die Eingangsqualität bezüglich dieses Nachbarn (link quality) übertragen. Knoten, die einen solchen LEEP Frame empfangen, lesen ihre Ausgangsqualität aus dem entsprechenden LI Entry mit ihrer Knotenadresse aus.

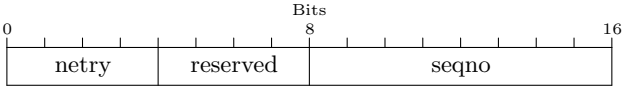


Abbildung 3: LEEP Header

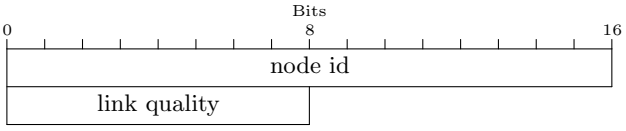


Abbildung 4: LEEP Link Information Entry

Wird ein Beacon von einem Knoten empfangen, der sich noch nicht in der LE-Tabelle befindet, wird ein neuer Eintrag für diesen angelegt. Dabei wird das „mature“-Flag auf 0 gesetzt, was bedeutet, dass es noch keine zuverlässige Kostenschätzung für diesen Knoten gibt und er nicht als Elternknoten verwendet werden darf. Folgende Beacons von diesem Knoten werden gezählt und anhand der Sequenznummer im LEEP-Header wird die Anzahl verlorener Beacons festgestellt. Nach einem Fenster von BLQ_PKT_WINDOW (5) Beacons wird Q_{in} bestimmt durch das Verhälnis der empfangenen Beacons b_{rcv} und der gesamten Beacons, die dieser Nachbarknoten versendet hat b_{sent}:

$$Q_{in} = \frac{b_{rcv}}{b_{sent}} \tag{2}$$

Liegt diese erste Schätzung der Eingangsqualität vor, wird dieser Knoten in der LE-Tabelle als „mature“ deklariert und so zur Verwendung als Next-Hop freigegeben. Weiterhin wird Q_{in} alle BLQ_PKT_WINDOW Routing Frames aktualisiert. Allerdings wird von nun an der alte Wert von Q_{in} gewichtet durch α mit einbezogen:

$$Q_{in,i} = \alpha \cdot Q_{in,i-1} + (1-\alpha) \cdot \frac{b_{rcv}}{b_{sent}} \tag{3}$$

Der LE-Tabellen-Eintrag wird zurückgesetzt, falls die Differenz der Sequenznummern zweier hintereinander empfangenen LEEP Frames größer ist als MAX_PKT_GAP (10). Dieser Nachbar verliert seinen „mature“-Status und die Verbindungsqualität zu ihm muss neu geschätzt werden.
Ein- und Ausgangsqualität bezüglich eines Knotens werden schließlich verwendet, um den Beacon-gestützten ETX-Wert zu bestimmen:

$$\mathrm{ETX}_b = \frac{1}{Q_{in} \cdot Q_{out}} \tag{4}$$

3.2 Linkestimation auf Datenebene

Sobald eine gültige Route durch einen Elternknoten verfügbar ist und Unicastpakete zu diesem Next-Hop gesendet werden, wird auch die Datenebene in die Linkschätzung mit einbezogen:

$$\mathrm{ETX}_d = \frac{d_{sent}}{d_{acked}} \tag{5}$$

Das Verhältnis aus gesendeten Unicastpaketen zu einem Knoten d_{sent} und den erfolgreich übertragenen d_{acked} ist ein direktes Maß für den ETX-Wert dieser Verbindung [10]. ETX_d wird aktualisiert, nachdem DLQ_PKT_WINDOW (3) Pakete jeweils durch ein entsprechendes ACK quittiert wurden oder durch Timeouts als verloren gelten. Sollten innerhalb eines solchen Paketfensters keine ACKs empfangen werden ($d_{acked} = 0$), so wird $\mathrm{ETX}_d = d_{sent}$ gesetzt [7]. CTP versendet ACKs nicht selbst, sondern erwartet, dass Schicht 2 diese Funktionalität anbietet. Es muss daher eine Sicherungsschicht verwendet werden, welche synchrone ACKs für Unicast-Pakete unterstützt [6].

3.3 Four-Bit Link Estimator

Seit September 2007 (Revision 3604) wird in TinyOS der von Fonseca und Gnawali in [7] vorgeschlagene Four-Bit Link Estimator verwendet, wodurch sich einige Abwandlungen von dem oben beschriebenen, alten Verfahren zur Verbindungskostenschätzung ergeben. Der Four-Bit Link Estimator nutzt Informationen aus weitereren Schichten, um *1-hop*-ETX-Werte zu bestimmen. Diese Informationen, namentlich White-, ACK-, Compare- und Pin-Bit, werden von der physikalischen Schicht (White-Bit), der Verbindungsschicht

doi: 10.2313/NET-2011-07-1_08

(ACK-Bit) und der Netzwerkschicht (Compare- und Pin-Bit) bereitgestellt.
Das White-Bit ist ein Hinweis auf eine gute Verbindungsqualität. Es wird dann gesetzt, wenn der Kanal laut Schicht 1 bei Empfang eines Paketes eine gute Qualität aufweist. Viele der in drahtlosen Sensornetzwerken eingesetzten Funkmodule wie das CC1000 [16] oder CC2420 [17] unterstützen diese Funktionalität.
Das ACK-Bit wird bei Paketen gesetzt, die erfolgreich durch ein ACK quittiert wurden. Es ermöglicht eine genaue Schätzung der bidirektionalen Verbindungsqualität. TinyOS implementiert das ACK-Bit durch die in Abschnitt 3.2 beschriebene Funktionalität.
Mit dem Pin-Bit kann die Netzwerkschicht dem Link Estimator mitteilen, dass ein Eintrag nicht aus der LE-Tabelle entfernt werden darf. Bei TinyOS wird das Pin-Bit bei dem Knoten gesetzt, der derzeit als Elternknoten verwendet wird und aus diesem Grund nicht ersetzt werden soll. Es wird auch bei LE-Tabellen-Einträgen gesetzt, deren zugehörige Nachbarknoten Wurzelknoten mit ETX-Wert 0 sind [4], da direkte Verbindungen zu Wurzelknoten denen über mehrere Hops vorgezogen werden.
Das Compare-Bit ermöglicht es dem Link Estimator, mit Hilfe der Netzwerkschicht festzustellen, ob ein Nachbarknoten vorteilhaft erscheint. Es wird gesetzt, falls die Kosten dieses Knotens geringer sind, als die von mindestens einem Eintrag der Routingtabelle. Compare- und White-Bit werden bei der Ersetzungsstrategie von Einträgen in der LE-Tabelle verwendet.
Beim Berechnen der Beacon-basierten Verbindungskosten, ETX_b, verzichtet der Four Bit Link Estimator auf die Ausgangqualität. Die Eingangqualitäten werden zwar durch das Link Estimation Exchange Protocol (siehe Abschnitt 3.1) an Nachbarknoten verteilt, um mit der CTP-Spezifikation [6] kompatibel zu bleiben. Sie werden aber von empfangenden Knoten ignoriert. Die Eingangsqualität wird lediglich als erste, grobe Schätzung der Verbindungsqualität verwendet, welche später durch Unicastpakete (Ack-Bit), wie in Abschnitt 3.2 beschrieben, genauer geschätzt werden kann.

3.4 Verwalten der LE-Einträge

Speicher ist auf Sensorknoten eine stark limitierte Ressource, wodurch die Link Estimation-Tabelle nur eine begrenzte Anzahl an Einträgen verwalten kann. Gleichzeitig ist es für den Link Estimator wichtig, Informationen über Nachbarknoten zu speichern, da die Verbindungsqualitäten zu ihnen nur über einen längeren Zeitraum genau geschätzt werden können. Es ist also von großer Bedeutung, nur die besten und vielversprechendsten Nachbarn in die LE-Tabelle mit aufzunehmen. Eine unvorteilhafte Auswahl an Nachbarknoten kann fatale Auswirkungen auf die resultierende Netzwerktopologie haben [12]. Aus diesem Grund verwendet CTP relativ viel Energie auf die Ersetzungsstrategie.
Falls noch Platz in der LE-Tabelle ist, kann ein neu entdeckter Nachbarknoten einfach eingefügt werden.
Ist die Tabelle voll, versucht CTP den schlechtesten Eintrag mit „mature“-Status zu finden, dessen *1-hop*-ETX-Wert oberhalb des Schwellwertes EVICT_ETX_THRESHOLD (TinyOS: 6,5) liegt. Gelingt es, einen solchen Eintrag zu finden, so wird er durch den Neuen ersetzt. Einträge, deren Pin-Bit gesetzt ist (der derzeitige Elternknoten und Wurzelknoten), werden hierbei nicht beachtet, da sie nicht aus der Tabelle entfernt werden dürfen.
Scheitert auch dieser Versuch, kann der Knoten nicht in die Tabelle aufgenommen werden, es sei denn, der Four-Bit Link Estimator wird verwendet. In diesem Fall überprüft CTP, ob das White- und das Compare-Bit bei dem Beaconpaket, durch das der neue Knoten sich meldet, gesetzt sind. Sind beide Bits gesetzt, wird ein zufälliger, unpinned, nicht-„mature“ Eintrag aus der LE-Tabelle entfernt und durch den neuen Knoten ersetzt, da dieser laut Netzwerkschicht (Compare-Bit) und physikalischer Schicht (White-Bit) sehr gute Kosten und Verbindungsqualitäten aufweist. Gibt es keinen solchen Eintrag, wird der Knoten nicht in die LE-Tabelle eingefügt.

4. ROUTING

In regelmäßigen Zeitabständen (8 Sekunden in TinyOS) wählen Kindknoten in CTP ihren Vaterknoten neu. Die Auswahlprozedur wird auch immer dann ausgeführt, bevor ein Beacon gesendet wird oder wenn der derzeitige Elternknoten nicht mehr erreichbar ist.
Aus der Routingtabelle wird der Knoten mit den geringsten Kosten bestimmt. Die Summe aus Knoten-ETX und *1-hop*-ETX zu diesem Knoten ist hierbei ausschlaggebend. In Betracht gezogen werden hierbei nur Nachbarknoten, die einen gültigen Elternknoten besitzen. Des Weiteren muss der zugehörige Eintrag in der LE-Tabelle den Status „mature“ besitzen und die Verbindungsqualität zu diesem potentiellen Next-Hop muss sich oberhalb eines gewissen Schwellwertes befinden: $ETX_{1hop} < 5$. Es werden auch keine direkten Kindknoten als Elternknoten zugelassen, da sich sonst Schleifen der Länge 2 bilden würden [4].
Wird ein Knoten gefunden, auf den obige Eigenschaften zutreffen, entscheidet CTP, ob es einen Wechsel zu diesem neuen Knoten gibt, oder ob der alte Next-Hop beibehalten wird. Damit nur geringfügig schwankende Verbindungsqualitäten nicht zu häufigen Topologieänderungen führen, wird nur zu einem neuen Elternknoten gewechselt, wenn der Pfad über diesen „signifikant“ besser ist. „Signifikant“ bedeutet nach [10], einen um 1,5 Übertragungen niedrigereren ETX-Wert zu besitzen.

5. DATENPAKETE

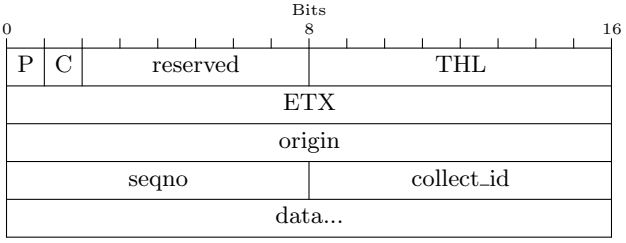


Abbildung 5: CTP Data Frame

CTP verwendet für Datenpakete Header wie in Abbildung 5 gezeigt. Der Header enthält wieder die aus dem Routing Frame (Abbildung 1) bekannten Pull- und Congestion-Bits. Ihre Funktion ist die gleiche wie in Abschnitt 2.1 beschrieben. Das Time-Has-Lived-Feld (THL) ist eine Art invertiertes TTL-Feld. Es startet mit dem Wert 0 und wird von jedem Hop inkrementiert. „origin“ enthält die Adresse des Ursprungsknotens und „seqno“ ist die Sequenznummer für dieses Paket. „collect_id“ kann von höheren Schichten verwendet werden, um dem Datenpaket einen bestimmten Typ

doi: 10.2313/NET-2011-07-1_08

zuzuweisen. Die Felder „origin“, „seqno“ und „collect_id“ werden von dem Sensorknoten gesetzt, welcher das Paket in Umlauf bringt, und von weiterleitenden Knoten nicht verändert [6]. In „ETX“ werden die Kosten des sendenden oder weiterleitenden Knotens übertragen. Diese Information wird benötigt, um Routingschleifen aufzulösen (siehe Abschnitt 6).

5.1 Sendepuffer

Jeder Sensorknoten verwaltet einen Sendepuffer, in welchem zu sendende Nachrichten nach FIFO-Strategie zwischengespeichert werden. Dieser Puffer hat in TinyOS eine Kapazität von 12 Einträgen für weiterzuleitete Pakete. Er besitzt zusätzlich Speicherplatz für jeweils ein Paket pro lokalem Client, der via CTP Daten versenden möchte. Pakete, die gesendet oder weitergeleitet werden sollen, werden an das Ende der Sendepuffers angefügt. Ist kein Speicher für weitere Pakete verfügbar, so wird das Paket verworfen.
Den Kopf dieses FIFO-Puffers bildet das Paket, das gerade versendet werden soll. Das Senden eines Paketes erfolgt synchron. Das heißt, es wird kein weiterer Sendevorgang ausgeführt, bis entweder ein ACK empfangen wird, oder ein Timeout eintritt. Wird das Paket durch ein entsprechendes ACK quittiert, so gilt es als erfolgreich übertragen und wird aus dem Puffer entfernt. Nach einer Zeitspanne t_{ack} wird der Sendevorgang für das nächste Paket im Sendepuffer eingeleitet. Konnte allerdings das ACK oder das Paket nicht erfolgreich übertragen werden, wird nach einem Timeout versucht, das Paket bis zu 30 weitere Male zu versenden. Zwischen den Sendeversuchen bleibt wieder eine Zeitspanne t_{nack}.
Die Verzögerungen t_{ack} und t_{nack} sind wichtig, um Interferenzen mit eigenen, früheren Paketen, die von Nachbarknoten weitergleleitet werden, zu vermeiden. Abbildung 6 veranschaulicht dies. Hier möchte Knoten K_1 zwei Pakete P_1 und P_2 an Knoten K_2 übertragen, der diese an K_3 weiterleitet. Die Kreise symbolisieren die jeweilige Funkreichweite von K_1 und K_3. Nachdem K_1 das erste Paket verschickt hat, muss er mit dem Sendevorgang von Paket P_2 mindestens solange warten, bis K_3 P_1 versendet hat, da sonst, falls K_1 und K_3 gleichzeitig senden, bei K_2 Interferenzen auftreten und K_2 P_2 nicht empfangen kann.

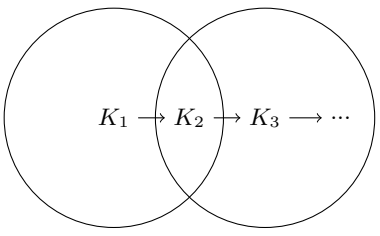


Abbildung 6: Mögliche Interferenz bei K_2

5.2 Umgang mit Paketduplikaten

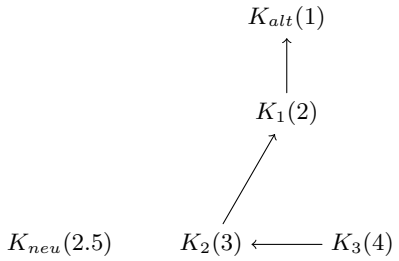
Durch stark asymmetrische Links oder kurzzeitige Schwankungen in der Linkqualität kann der Fall auftreten, dass ein Datenpaket beim Empfangsknoten ankommt, dieser das Paket durch ein ACK quittiert, das ACK aber nicht vom Sendeknoten empfangen wird. Da der Sendeknoten keine Quittierung erhält, muss er davon ausgehen, dass auch sein Paket verloren gegangen ist, weshalb er das Datenpaket erneut sendet.
Solche Paketduplikate stellen ein Problem für das drahtlose Sensornetz dar, da sie im ungünstigsten Fall bei jedem Hop verdoppelt werden und ihre Zahl somit exponentiell mit der Anzahl weiterleitender Knoten wächst [10]. Aus diesem Grund versucht CTP, duplizierte Pakete zu erkennen und ihre Weiterleitung zu unterdrücken.
Zwei Pakete gelten als Duplikate, wenn sie in den Feldern „origin“, „seqno“, „collect_id“ und „THL“ übereinstimmen. Die Überprüfung des Time-Has-Lived-Feldes ist notwendig, da es aufgrund von Schleifen vorkommen kann, dass ein Paket einen Knoten mehrmals passiert. Damit in einem solchen Fall das Paket nicht fälschlicherweise als Duplikat identifiziert und verworfen wird, muss auch der THL-Wert mit einbezogen werden. Denn falls die Schleifenlänge nicht ein Vielfaches von 256 beträgt, unterscheidet sich der THL-Wert nach jedem Schleifendurchgang.[10]
Bevor ein Paket zum Weiterleiten in den Sendepuffer eingefügt wird, überprüft CTP zunächst, ob das Paket bereits darin vorliegt. Wird eine Übereinstimmung in den obigen Header-Feldern gefunden, so wird das Paket verworfen.
Weil der Sendepuffer im optimalen Fall nicht sehr voll ist und es sein kann, dass das gesuchte Paket schon erfolgreich übertragen wurde, verwaltet CTP zusätzlichen Speicher für versendete Pakete, den sogenannten Transmit Cache [10]. Diese FIFO-Schlange enthält die für eine Paketinstanz charakteristischen Felder („origin“, „seqno“, „collect_id“ und „THL“) der n zuletzt weitergeleiteten Pakete. Dadurch kann eine Übereinstimmung mit früheren Paketen geprüft werden. TinyOS speichert auf diese Weise $n = 4$ Pakete. In [9] wird ein Verfahren beschrieben, bei dem nicht die charakteristischen Tupel der zuletzt weitergeleiteten Pakete, sondern ein Hashwert gespeichert wird. TinyOS 2.1.1 verwendet aber direkt die Tupel, wie in [10] erläutert.
Nur wenn weder im Transmit Cache noch im Sendepuffer Paketduplikate gefunden werden, kann das Paket zum Versenden in den Sendepuffer aufgenommen werden.

6. TOPOLOGIEÄNDERUNGEN

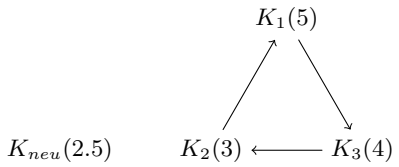
Sensornetzwerke unterliegen ständigen Topologieänderungen: Die Knoten haben oft nur sehr wenig Batteriekapazität zur Verfügung, wodurch sie immer wieder ausfallen und ersetzt werden müssen. Auch die Verbindungsqualitäten zwischen den Knoten können durch Knotenbewegung oder abschirmende Objekte schwanken.
Der von CTP verwendete Link Estimator kann durch seine kleinen Aktualisierungsfenster schnell auf solche Veränderungen reagieren, sodass die Netzwerktopologie die physikalischen Gegebenheiten zeitnah widerspiegelt.[10]
Verbessert sich der ETX-Wert eines Knotens um mehr als 1,5 (2,0 in TinyOS) Übertragungen, stellt er möglicherweiße einen günstigereren Next-Hop für seine Nachbarn dar. Aus diesem Grund verringert er das Beaconintervall auf τ_l, um seine Nachbarknoten innerhalb kurzer Zeit über die neuen Kosten zu informieren.[10]
Distance Vector Routing Protokolle wie CTP sind sehr anfällig gegenüber Schleifenbildung. Routingschleifen können entstehen, wenn Knoten aufgrund von Topologieänderungen, die noch nicht von allen Nachbarknoten wahrgenommen wurden, Pfade wählen, welche einen eigenen Kindknoten enthalten.
Die Abbildungen 7a und 7b veranschaulichen dies. In diesem Beispiel fällt die Verbindung von Knoten K_1 zu K_{alt} aus. K_1 wählt nun K_3 als seinen Elternknoten und es entsteht eine

doi: 10.2313/NET-2011-07-1_08

Schleife. Da K_2 noch veraltete Kosteninformationen über K_1 besitzt, die unter denen von K_{neu} liegen, wählt er keinen neuen Elternknoten. Die Zahlen in Klammern stehen für beispielhafte Knotenkosten.



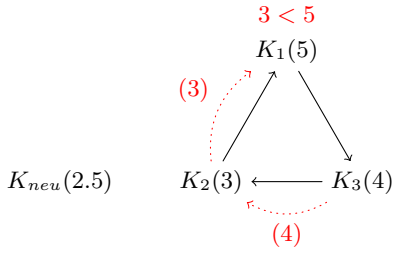
(a) Die Verbindung $K_1 \rightarrow K_{alt}$ fällt aus



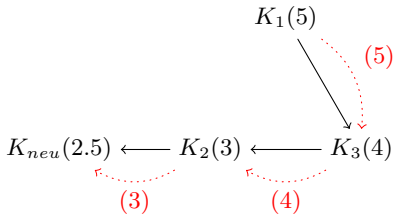
(b) Eine Schleife hat sich gebildet

Abbildung 7: Entstehung von Routingschleifen

CTP kann solche Schleifen erkennen und aktiv auflösen. Dazu wird im Header eines jeden Datenpaketes (siehe Abbildung 5) der ETX-Wert des weiterleitenden Knotens übertragen. Empfängt ein Sensorknoten ein Paket mit einem ETX-Wert, der geringer ist als sein eigener, deutet das auf eine Schleife hin. Denn in einer konsistenten Topologie muss der ETX-Wert von Hop zu Hop in Richtung Wurzelknoten monoton abnehmen.



(a) K_1 stellt eine Schleife fest



(b) K_2 wählt einen neuen Vaterknoten

Abbildung 8: Auflösen einer Schleife

Der Knoten, der eine solche Inkonsistenz feststellt, setzt sein Beaconintervall auf den Minimalwert τ_l zurück, um benachbarte Knoten schnell zu informieren. Um den Routing Frames Vorrang zu gewähren, werden eine kurze Zeitspanne lang keine Datenpakete versendet.
CTP verwirft Datenpakete nicht, bis die Schleife beseitigt ist, sondern nutzt sie, um die Topologie aktiv zu reparieren: Die Datenpakete zirkulieren so lange in der Schleife, bis ein Knoten einen Next-Hop wählt, mit dem ein Weg aus der Schleife gefunden wird. Sie veranlassen dabei die Hops auf ihrem Pfad, die Beaconrate zu erhöhen, um die Reparatur zu beschleunigen.[10]
Abbildung 8 zeigt ein Beispiel dieses Vorgangs. Die Knoten K_1 bis K_3 bilden eine Schleife. Das von K_3 in Umlauf gebrachte Datenpaket erreicht schließlich K_1. Dieser stellt eine mögliche Schleife fest, da das empfangene Paket einen geringeren ETX-Wert aufweist als er. Er sendet mit erhöhter Frequenz Beacons und informiert dadurch unter anderem K_2 über seine gestiegenen Kosten. Das Paket zirkuliert währenddessen weiter in der Schleife. Bei der nächsten Ankunft bei K_2 besitzt dieser Knoten aber aktualisierte Informationen über die Kosten von K_1 und hat deshalb K_{neu} als seinen Elternknoten gewählt. Das Datenpaket verlässt über die Verbindung $K_2 \rightarrow K_{neu}$ die Schleife.

7. LEISTUNGSBEWERTUNG

Gnawali, Fonseca et al. dokumentieren in [10] ihre Experimente mit CTP. Dabei wurden 12 Testbeds unterschiedlicher Größe von $6m$ x $4m$ bis $50m$ x $25m$ x $10m$ untersucht. Die Testbeds enthielten jeweils 35 bis 310 Sensorknoten vom Typ Tmote, TelosB, Mica2dot, MicaZ, Epic-Quanto, eyesIFXv2 oder Blaze. Durch die unterschiedliche Verteilung und Dichte hatten die Knoten einen Grad von 6 bis 305. Die Experimente liefen für mindestens 3 Stunden, in denen die Sensorknoten alle 16 Sekunden Daten an die Wurzel sendeten. In allen Fällen kamen mindestens 90% der auf den Sensorknoten generierten Daten bei den Wurzelknoten an. In mehr als der Hälfte der Testkonfigurationen lag der Wert sogar über 99%. Hierbei ist allerdings anzumerken, dass CTP so konfiguriert war, nach einem Paketverlust die Nachricht bis zu 30 weitere Male erneut zu versenden. Das führt einerseits zu einer hohen Verlässlichkeit, dass Datenpakete die Wurzelknoten erreichen, kann bei hohem Verkehrsaufkommen aber auch die Leistung des Protokolls beeinträchtigen. Viele erneute Sendeversuche führen nicht nur zu einer temporären Überlastung der Luftschnittstelle, sondern erzeugen auch Paketstau auf den Knoten.
Verglichen mit dem ebenfalls sehr bekannten Protokoll MultiHopLQI erzielte CTP bessere Ergebnisse. MultiHopLQI wies einen niedrigereren Bruchteil an erfolgreich übertragenen Datenpaketen von durchschnittlich 85% auf, welcher zudem größeren Schwankungen unterworfen war. Durch seine konstante Beaconrate benötigte MultiHopLQI 73% mehr Control Overhead als CTP, das die Beaconrate dynamisch anpasst.
Die simulative Leistungsbewertung von Colesanti und Santini in [4] bestätigt die Ergebnisse von Gnawali et al. größtenteils. In einem simulierten Testbed der Größe $250m$ x $250m$ wurden 100 Knoten gleichmäßig verteilt und folgendes Experiment durchgeführt. Alle Knoten wachen in regelmäßigen Intervallen gleichzeitig auf, erzeugen die Baumtopologie und senden Daten zum Wurzelknoten. Anschließen gehen sie wieder in einen Schlafzustand über. Jeder der Knoten nimmt in

doi: 10.2313/NET-2011-07-1_08

der Wachphase aktiv am Erstellen der Netzwerktopologie teil. Ob er aber in einem Zyklus auch Daten versendet, wird durch eine vordefinierte Wahrscheinlichkeit p bestimmt. Die Schlafphase dauert $44s$ und das Zeitfenster, in dem die Knoten aktiv sind, ist unterteilt in $11s$ für das Erstellen der Topologie und $5s$ um die Daten weiterzureichen. Colesanti und Santini werteten auf diese Weise 50 Topologien mit jeweils unterschiedlicher Sendewahrscheinlichkeit p aus. Sie kamen zu dem Ergebnis, dass mit $p = 0{,}5$ mehr als 95% der gesendeten Pakete die Datensenke erreichen. Mit höherer Sendewahrscheinlichkeit $p = 1$ werden allerdings in manchen Fällen nur 80% oder weniger erreicht. Im Durchschnitt kamen aber für alle Werte von p über 95% der gesendeten Datenpakete bei den Wurzelknoten an.[4]

8. FAZIT

In dieser Arbeit wurde das Routingprotokoll CTP beschrieben. Der Schwerpunkt lag dabei darauf, wie es mit den Herausforderungen in drahtlosen Sensornetzwerken umgeht. Denn CTP besitzt eine Vielzahl an Eigenschaften, die auf dieses Einsatzgebiet hin zugeschnitten sind. Dies zeigt, dass Routing in solchen Netzwerken kein einfaches Thema ist.
Dennoch ist es mit CTP möglich, in einem hohen Grad verlässliches Routing zu betreiben, was die weite Verbreitung des Protokolls erklärt. Mit seinen Ideen hat CTP auch die Entwicklung des IPv6 Routingprotokolls RPL wesentlich beeinflusst. RPL adaptiert beispielsweise das in CTP erprobte dynamische Beaconintervall.
Dabei gibt es für CTP und allgemein auf dem Gebiet Routing in drahtlosen Sensornetzwerken durchaus noch deutlichen Forschungs- und Verbesserungsbedarf. So wäre zum Beispiel im Fall von CTP eine stabil funktionierende Methode, überlastete Knoten zu umgehen, wünschenswert.

9. LITERATUR

[1] *Mantis OS*. `http://mantisos.org/documentation/api/html-unstable/ctp_8h-source.html`, Okt. 2007.

[2] *MultiHopLQI Routing Protocol*. `http://www.tinyos.net/tinyos-2.1.0/tos/lib/net/lqi/`, Aug. 2008.

[3] *Contiki 2.5 Release Candidate 1*. `http://www.sics.se/contiki/news/contiki-2.5-release-candidate-1-avaliable.html`, Nov. 2010.

[4] Colesanti, U. und S. Santini: *A Performance Evaluation of the Collection Tree Protocol Based on its Implementation for the Castalia Wireless Sensor Networks Simulator*. Technical Report 681, Department of Computer Science, ETH Zurich, Zurich, Switzerland, Aug. 2010.

[5] Dinh, T. L., W. Hu, P. Sikka, P. Corke, L. Overs und S. Brosnan: *Design and Deployment of a Remote Robust Sensor Network: Experiences from an Outdoor Water Quality Monitoring Network*. In: *32nd IEEE Conference on Local computer Networks, 2007. LCN 2007.*, S. 799 –806, Okt. 2007.

[6] Fonseca, R., O. Gnawali, K. Jamieson, S. Kim, P. Levis und A. Woo: *TEP 123: The Collection Tree Protocol*. `http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html`, Feb. 2007.

[7] Fonseca, R., O. Gnawali, K. Jamieson und P. Levis: *Four Bit Wireless Link Estimation*. In: *Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets VI)*, Nov. 2007.

[8] Gnawali, O.: *TEP 124: The Link Estimation Exchange Protocol (LEEP)*. `http://www.tinyos.net/tinyos-2.x/doc/html/tep124.html`, Feb. 2007.

[9] Gnawali, O., R. Fonseca, K. Jamieson und P. Levis: *CTP: Robust and Efficient Collection through Control and Data Plane Integration*. Technical Report, The Stanford Information Networks Group (SING), Stanford University, Feb. 2008.

[10] Gnawali, O., R. Fonseca, K. Jamieson, D. Moss und P. Levis: *Collection tree protocol*. In: *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, S. 1–14, Nov. 2009.

[11] Kazandjieva, M., O. Gnawali, B. Heller, P. Levis und C. Kozyrakis: *Identifying Energy Waste through Dense Power Sensing and Utilization Monitoring*. Technical Report, Computer Science Lab, Stanford University, März 2010.

[12] Langendoen, K., A. Baggio und O. Visser: *Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture*. In: *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, S. 8 pp., Apr. 2006.

[13] Levis, P., N. Patel, D. Culler und S. Shenker: *Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks*. In: *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, S. 15 – 28. USENIX Association, März 2004.

[14] Romer, K. und F. Mattern: *The design space of wireless sensor networks*. Wireless Communications, IEEE, 11(6):54 – 61, Dez. 2004.

[15] Rothery, S., W. Hu und P. Corke: *An empirical study of data collection protocols for wireless sensor networks*. In: *Proceedings of the workshop on Real-world wireless sensor networks*, REALWSN '08, S. 16–20, 2008.

[16] Texas Instruments: *CC1000 Datasheet*. `http://www.ti.com/lit/gpn/cc1000`, Feb. 2007.

[17] Texas Instruments: *CC2420 Datasheet*. `http://www.ti.com/lit/gpn/cc2420`, März 2007.

doi: 10.2313/NET-2011-07-1_08

RPL: IPv6 Routing Protocol for Low Power and Lossy Networks

Tsvetko Tsvetkov
Betreuer: Alexander Klein
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS 2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: tsvetko.tsvetkov@gmail.com

ABSTRACT

Today, Low Power and Lossy Networks (LLNs) represent one of the most interesting research areas. They include Wireless Personal Area Networks (WPANs), low-power Power Line Communication (PLC) networks and Wireless Sensor Networks (WSNs). Such networks are often optimized to save energy, support traffic patterns different from the standard unicast communication, run routing protocols over link layers with restricted frame-sizes and many others [14].
This paper presents the IPv6 Routing Protocol for Low power and Lossy Networks (RPL) [19], which has been designed to overcome routing issues in LLNs. It implements measures to reduce energy consumption such as dynamic sending rate of control messages and addressing topology inconsistencies only when data packets have to be sent. The protocol makes use of IPv6 and supports not only traffic in the upward direction, but also traffic flowing from a gateway node to all other network participants.
This paper focuses on the employment of RPL in WSNs and gives a brief overview of the protocol's performance in two different testbeds.

Keywords

RPL, Sensor Network, Low-Power Network, Lossy Link, Routing, Data Collection, Data Dissemination

1. INTRODUCTION

Over the last years WSNs have become a very important and challenging research field. Such networks consist of spatially distributed autonomous devices which usually operate untethered and additionally have limited power resources. This limits all aspects of their construction, architecture and communication capabilities. Several studies such as [2] and [11] reveal the impact of wireless lossy links on the overall reliability, power efficiency and maximum achievable throughput. There are cases where a network can only achieve approximately the half of the throughput of the corresponding lossless network. Moreover, lossy links effect the power consumption due to packet retransmissions and broadcasting. Zhao and Govindan [20] have estimated the impact of such links and concluded that 50% to 80% of the communication energy is wasted in overcoming packet collisions and environmental effects in indoor and outdoor scenarios.
Such LLNs are additionally characterized by connections that are not restricted to two endpoints. Many scenarios may include Point-to-Multipoint (P2MP) or Multipoint-to-Point (MP2P) traffic patterns. Such networks are also known for their asymmetric link properties. The communication is realized by using a separate uplink and downlink. Because each unidirectional link provides only one way traffic, the bandwidths in the two directions may differ substantially, possibly by many orders of magnitude.
In order to meet these requirements and challenges, the Internet Engineering Task Force (IETF) ROLL Working Group designed a new routing protocol, called RPL [18]. The highest goal of RPL is to provide efficient routing paths for P2MP and MP2P traffic patters in LLNs. The protocol successfully supports the latest version of the Internet Protocol which results from the research made by different organizations.
The IP for Smart Objects (IPSO) Alliance has made a great effort to promote the use of IP for small devices [4]. It is the leading organization for defining the *Internet of Things* and supports the use of the layered IP architecture for small computers. The cooperation with the IETF organization further accelerates the adoption of IPv6 on LLNs. IETF has specified the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) standard [12] which supports the idea of applying IPv6 even to the smallest machines. In this way, devices with limited hardware resources are able to participate in the Internet of Things. This standard also enables the use of standard web services without application gateways.
The rest of this paper is organized as follows. Section 2 gives an overview of RPL's basic features and describes the terminology of the protocol. Section 3 discusses topics such as topology construction and structure of the used control message. An introduction to RPL's loop avoidance and detection mechanisms is presented in Section 4. Section 5 gives information about the different routing metrics. Section 6 describes how the support of P2MP traffic is realized and Section 7 gives an overview of the protocol's performance. Finally, the paper is concluded in Section 8.

2. RPL DESIGN OVERVIEW

RPL is a distance vector routing protocol for LLNs that makes use of IPv6. Network devices running the protocol are connected in such a way that no cycles are present. For this purpose a Destination Oriented Directed Acyclic Graph (DODAG), which is routed at a *single* destination, is built. The RPL specification calls this specific node a DODAG root. The graph is constructed by the use of an Objective

doi: 10.2313/NET-2011-07-1_09

Function (OF) which defines how the *routing metric* is computed. In other words, the OF specifies how routing constraints and other functions are taken into account during topology construction.
In some cases a network has to be optimized for different application scenarios and deployments. For example, a DODAG may be constructed in a way where the Expected Number of Transmissions (ETX) or where the current amount of battery power of a node is considered. For this reason, RPL allows building a logical routing topology over an existing physical infrastructure. It specifies the so-called RPL Instance which defines an OF for a set of one or more DODAGs.
The protocol tries to avoid routing loops by computing a node's position relative to other nodes with respect to the DODAG root. This position is called a *Rank* and increases if nodes move away from the root and decreases when nodes move in the other direction, respectively. The Rank may be equal to a simple hop-count distance, may be calculated as a function of the routing metric or it may be calculated with respect to other constraints.
The RPL specification defines four types of control messages for topology maintenance and information exchange. The first one is called DODAG Information Object (DIO) and is the main source of routing control information. It may store information like the current Rank of a node, the current RPL Instance, the IPv6 address of the root, etc. The second one is called a Destination Advertisement Object (DAO). It enables the support of down traffic and is used to propagate destination information upwards along the DODAG. The third one is named DODAG Information Solicitation (DIS) and makes it possible for a node to require DIO messages from a reachable neighbor. The fourth type is a DAO-ACK and is sent by a DAO recipient in response to a DAO message. The RPL specification defines all four types of control messages as ICMPv6 information messages with a requested type of 155. This new type has been officially confirmed by IANA [6]. Note that the last two are not further described in this paper.
Another important fact about the protocol's design is the maintenance of the topology. Since most of devices in a LLN are typically battery powered, it is crucial to limit the amount of sent control messages over the network. Many routing protocols broadcast control packets at a fixed time interval which causes energy to be wasted when the network is in a stable condition. Thus, RPL adapts the sending rate of DIO messages by extending the Trickle algorithm [10]. In a network with stable links the control messages will be rare whereas an environment in which the topology changes frequently will cause RPL to send control information more often.

3. UPWARD ROUTING

Upward routing is a standard procedure which enables network devices to send data (e.g. temperature measurements) to a common data sink, also called sometimes a gateway or root node. In a typical WSN scenario, nodes periodically generate data packets (e.g. each minute) which have to find their way through the network. In this section, the RPL topology construction process is discussed and the structure of a DIO message is presented.

3.1 DIO Message Structure

As previously mentioned, a DIO message is the main source of information which is needed during topology construction. Figure 1 represents the structure of the message.

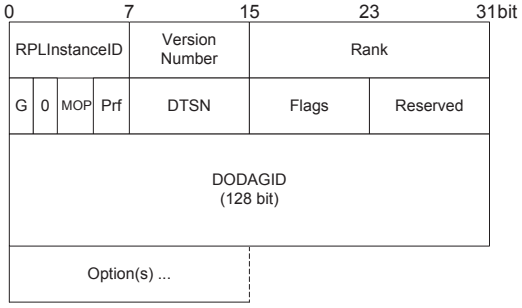


Figure 1: DIO Message Structure

A DIO first allows a node to discover the RPL Instance by storing the corresponding one in the first data field. The second and the third field include the DODAG Version and the Rank of the sender of the message. Section 3.2 describes the purpose of the RPL Instance, version number and the Rank update process. The next byte includes the 'G' flag which defines whether a DODAG is grounded. Grounded means that it can satisfy an application-defined goal. If it is not set, the DODAG is said to be floating. This may happen when a DODAG is disconnected from the rest of the network and supports only connectivity to its nodes. The MOP field (size of 3 bits) is set by the DODAG root and defines the used mode of operation for downward routing. Section 6 gives more information about it. The Prf field (size of 3 bits) defines how preferable the root node is compared to other root nodes. Such a node is identified by the DODAGID field. The last used field is DTSN and it is needed for saving a sequence number. Such a number is maintained by the node issuing the DIO message and guarantees the freshness of the message.
A DIO message may be extended by the use of options. In this paper, only the DODAG Configuration option is discussed, since it plays a crucial role for parameter exchange. Figure 2 outlines its structure.

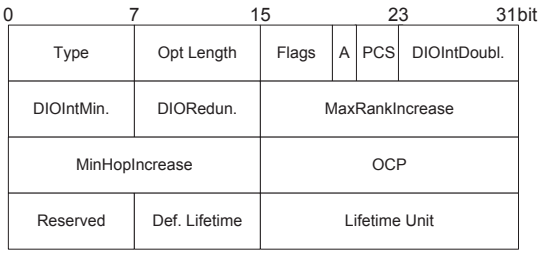


Figure 2: DODAG Configuration Option

The first two bytes always include the type (`0x04`) and the option's length (14 bytes). The next byte includes the 'A' flag, the Flags field and the PCS field. They will be not further discussed since they are not important for this paper. The next two bytes define the maximum timer value τ_{max} and the minimum timer value τ_{min} needed for the trickle

doi: 10.2313/NET-2011-07-1_09

timer setup. More information can be found in Section 3.2. The DIORedun. field defines a constant k used for suppressing DIO messages. If a node receives more than k DIOs, it may suppress them. The MaxRankIncrease field defines an upper limit for the Rank and is further discussed in Section 4. The MinHopIncrease field stores the minimum increase of the Rank between a node and any of its parent nodes. It creates a trade-off between the maximum number of hops and the hop cost precision. The DODAG Configuration Option concludes with the OCP field (OF identification), the Default Lifetime for all routes and the Lifetime Unit. The latter one defines in seconds the length of a time unit.

3.2 Constructing Topologies

In general, there are three[1] types of nodes in a RPL network. The first type are root nodes which are commonly referred in literature as gateway nodes that provide connectivity to another network. The second type are routers. Such nodes may advertise topology information to their neighbors. The third type are leafs that do not send any DIO messages and have only the ability to join an existing DODAG.
The construction of the topology starts at a root node that begins to send DIO messages. Each node that receives the message runs an algorithm to choose an appropriate parent. The choice is based on the used metric and constraints defined by the OF. Afterwards each of them computes its own Rank and in case a node is a router, it updates the Rank in the DIO message and sends it to all neighboring peers. Those nodes repeat the same steps and the process terminates when a DIO message hits a leaf or when no more nodes are left in range. A possible Rank computation is shown in Equation 1. In this example, *floor(v)* evaluates v to the greatest integer less than or equal to v.

$$DAGRank(rank) = floor\left(\frac{rank}{MinHopIncrease}\right) \qquad (1)$$

However, in most sensor node deployments several data collection points (root nodes) are needed. Thus, three values have to be considered in order to uniquely identify a DODAG: (1) RPL Instance ID for identification of an independent set of DODAGs, optimized for a given scenario; (2) DODAG ID which is a routable IPv6 address belonging to the root; (3) DODAG version number which is incremented each time a DODAG reconstruction is needed. The RPL specification defines the combination of those three values as a DODAG Version. An example is shown in Figure 3. Each of the three constructed topologies may be positioned in different rooms where the construction of a single DODAG is impossible. Due to the fact that the three graphs belong to the same instance, their construction is realized in a similar way (e.g. by considering ETX values).
The RPL specification distinguishes between three logical sets when building upward routes. First, the candidate neighbor set which includes all reachable nodes. They may belong to different DODAG Versions. For example, in Figure 3 node 10 may store node 11 in its candidate neighbor set. Second, the parent set which is a subset of the candidate neighbor set. It includes only nodes that belong to the same DODAG Version. When a node stores a neighbor into the parent set, it becomes attached to the given DODAG.

[1]Virtual roots are not considered in this paper.

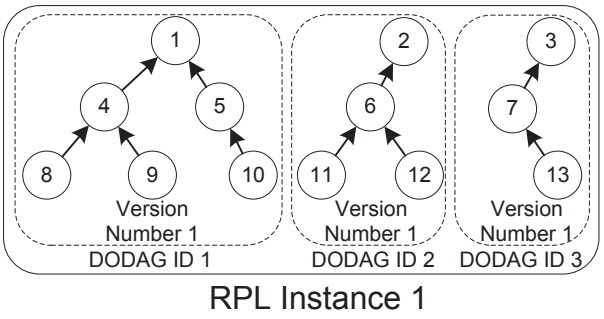


Figure 3: RPL Topology Partition

The last one contains one[2] element: the most preferred next hop taken from the parent set. Note that a node may belong to more than one RPL Instance. In this case, it must join a DODAG for each RPL Instance.
As previously mentioned, RPL dynamically adapts the sending rate of its control DIO messages. To achieve this, two values need to be used: one for defining the minimum sending time interval τ_{min} and another for defining the maximum sending interval τ_{max}. Whenever the sending timer expires, RPL doubles it up to the maximum value τ_{max}. Whenever RPL detects an event which indicates that the topology needs active maintenance, it resets the timer to τ_{min}. Such events are a found inconsistency when forwarding a data packet, joining of a new node, leaving the current DODAG and triggering topology repair.

4. ROUTING LOOPS

The formation of routing loops is a common problem in all kinds of networks. Due to topology changes caused by failure or mobility, a node may pick a new route to a given destination. If the new route includes a network participant which is a descendant, loops may occur. This leads to network congestion, packet drops, energy waste and delays. However, a quick and reliable detection of such topology inconsistencies is not a sufficient solution for LLNs. For example, even in a complete static sensor node deployment a malfunctioning antenna of a node may cause frequent changes of the node's distance to the root. Child nodes may be picked as next hops by their parents and a topology repair mechanism may be triggered. This leads to further energy consumption and waste of bandwidth. Therefore, a routing protocol for LLNs has to define a loop avoidance strategy considered during topology construction.

4.1 Avoidance Mechanisms

So far, it was only said that a Rank represents a node's position in the graph and that router nodes forward DIO control messages for topology maintenance. However, such messages are sent in a multicast manner to the neighboring nodes. If each node in range accepts such messages and takes the sender of the DIO into account for computation of the parent set, it may happen that child nodes are selected as best next hops. Consider the example shown in Figure 4. The topology is constructed by taking ETX into account. Further, the Rank value equals the ETX metric.

[2]RPL also allows multiple equally preferred parents.

doi: 10.2313/NET-2011-07-1_09

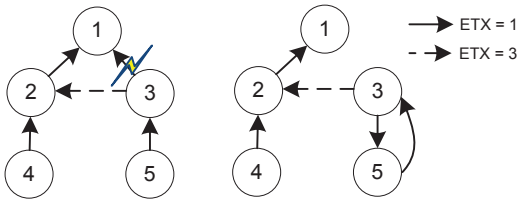


Figure 4: Loop Creation

If node 3 processes a DIO control message from node 5 it will consider it as a valid possible parent. If the link between the root node and node 3 fails, node 3 will simply pick its child node as next hop and a loop will occur, since the ETX cost to the root is 3. Compared to the other possible way to the root (through node 2) the ETX cost is 4.
Therefore, a RPL node does not process DIO messages from nodes deeper (higher Rank) than itself because such nodes may belong to its sub-DODAG. Even if node 4 is reachable for node 3, it should not be considered as next hop. Instead, node 3 has to declare an invalid state, poison its routes and join the DODAG Version again.
RPL also limits the movement of a node within a DODAG Version. Since moving up in a DODAG does not present the risk of creating a loop but moving down might, the RPL specification suggests that a node must never advertise within a DODAG Version a Rank higher than $Rank_{Lowest} + Rank_{MaxInc}$. $Rank_{Lowest}$ is the lowest Rank the node has advertised within a DODAG Version and $Rank_{MaxInc}$ is a predefined constant received via a DIO. Figure 5 illustrates a simple topology where node 1 is the DODAG root.

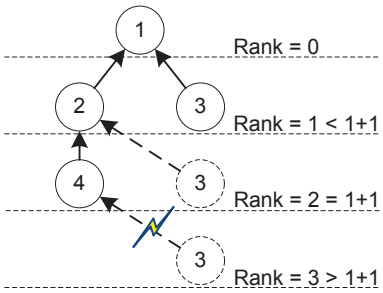


Figure 5: Movement Limitation within a DODAG Version

Let node 2 and 3 have a Rank of 1 and node 4 a Rank of 2. In addition, let $Rank_{MaxInc} = 1$. If node 3 moves away and increases its Rank to 2, it is allowed to choose node 2 as next hop. However, if node 3 moves even further away and increases its Rank to 3, it must not pick node 4 as a next hop. The RPL document does not specify what node 3 should do in this situation. A possible solution is to join another DODAG or advertise a DODAG rebuild request.

4.2 Detection Mechanisms

Usually, in LLNs nodes rarely generate data traffic which makes keeping the topology consistent all the time wasteful in terms of energy consumption. For example, Koen Langendoen et al. introduced in their paper [9] a large-scale experiment in which several sensor nodes were disseminated over a potato field programmed to send data every 60 seconds. Even for this relatively large time interval they have experienced loss of battery power after three weeks of operation. If the topology is kept consistent all the time, it may happen that nodes experience lack of energy after a shorter period of time. Thus, changes in connectivity need not to be addressed until data packets are present.
RPL loop detection uses additional information that is transported in the data packets. It places a RPL Packet Information in the IPv6 option field which is updated and examined on each hop. There are five control fields within the RPL Packet Information. The first one indicates whether the packet is sent in a upward or downward direction. The second one reports if a Rank mismatch has been detected. It is used to indicate whenever the Rank of the sender, stored in the packet, is lower than the Rank of the receiver. The RPL specification suggests that packets should not be immediately dropped if such a inconsistency is detected. Instead, the packet should be forwarded. However, if an inconsistency is detected on the packet for the second time, it must be dropped and the trickle timer must be reset. In this way, route repair is triggered. The third one is the forwarding error field and is used by a child node to report that it does not have a valid route to the destination of the packet. The last two are the Rank of the sender and the RPL Instance ID.

5. RPL METRICS

Many of today's routing protocols use link metrics that do not take a node's current status into account. The status includes typical resources such as CPU usage, available memory and left energy. This may be crucial for LLNs where network devices are usually battery powered and have limited hardware resources. For example, if a chain topology occurs in a sensor network deployment, the last node before the root will usually experience a higher traffic load and forwarding overhead than the others. If in Figure 6 all nodes frequently generate data packets and send them to the root, node 2 may fail very quickly due to the lack of energy. Even if the link between node 3 and 7 is not an optimal solution, it may be reasonable to send data packets through node 7 since it may offer a more stable node condition. Otherwise, if node 2 fails it may take some time until node 3 picks node 7 as next hop and packet drop may occur.

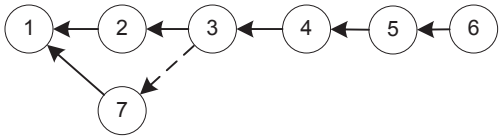


Figure 6: Chain Topology

Therefore, the ROLL Working Group defined several metric categories in [7] that may be considered when selecting the next hop. Some of the metrics defined in the document are carried in messages that are optional from the point of view of a RPL implementation. They have to be additionally specified and are not further described in this paper. In the following two possible metric computations from the RPL metrics document will be shortly discussed.

doi: 10.2313/NET-2011-07-1_09

5.1 Node Energy Consumption

This method suggests that a node should consider the energy level of its neighbors before picking them as possible parents. For this purpose, two units of information are used: (1) the type of the node which indicates how it is supplied with power and (2) the Energy Estimation (EE). The RPL metric specification defines three possible states for the first information field: powered, on batteries and scavenger. If a network device is powered it means that it may be the root node connected to a PC or it may be some sort of special data collector (e.g. cluster-heads in hierarchical routing). Such nodes may report a maximum EE value and, in general, are preferable during parent selection. If a node is on batteries, it has to compute its EE value by using Equation 2. The $Power_{now}$ value is the remaining energy and $Power_{max}$ is the power estimation reported at boot up.

$$EE = \frac{Power_{now}}{Power_{max}} \cdot 100 \tag{2}$$

However, if a node derives energy from external sources [13] it may report EE as a quantity value that is computed by dividing the amount of power the node has acquired by the power consumed. This may be a rough estimation of how much load a node experiences for a given period of time.

5.2 ETX

This metric is an approximation of the expected number of transmissions until a data packet reaches the gateway node. A node that is one hop away from the root, with perfect signal strength and very little interference, may have an ETX of 1. Another node with a less reliable connection to a root may have a higher ETX.
ETX is a bidirectional single-hop link quality computation between two neighbor nodes [1]. For the computation a metric called Packet Reception Rate (PRR) is used. PRR is calculated at the receiver node for each window ρ of received packets, as follows:

$$PRR(\rho) = \frac{Number\ of\ received\ packets}{Number\ of\ sent\ packets} \tag{3}$$

In literature the value computed in Equation 3 is also defined as *in-quality*, which is the quality from node A to node B measured by node B by counting the successfully received packets from A among all transmitted. In this paper it will be called PRR_{down}. For the actual ETX estimation the *out-quality* is further needed. This is the in-quality estimated by node A and is defined as PRR_{up} at node B. In this way node B can calculate ETX as shown in Equation 4:

$$ETX = \frac{1}{PRR_{down} \cdot PRR_{up}} \tag{4}$$

6. DOWNWARD ROUTING

The support of downward routing is another important key feature of the protocol. By supporting P2MP traffic it is possible for a network administrator to control nodes that are even not in range. This is very useful for performance evaluation purposes where usually several hundred nodes are spread over a large area. If such traffic is not supported, even the slightest changes, such as a timer value, may require to find the node, disconnect it from the network and upload a new code image. Moreover, if the idea of the Internet of Things is considered, P2MP becomes a must for LLN routing protocols [17].
The RPL specification defines two modes of operation for supporting P2MP. First, the non-storing mode which makes use of source routing. In this mode each node has to propagate its parent list up to the root. After receiving such topology information, the root computes the path to the destinations. Second, the storing mode which is fully stateful. Here, each non-root and non-leaf network participant has to maintain a routing table for possible destinations. Note that any given RPL Instance is either storing or non-storing.

6.1 DAO Message Structure

As mentioned in Section 2, DAO messages are used by RPL nodes to propagate routing information in order to enable P2MP traffic. Figure 7 represents the structure of a DAO message.

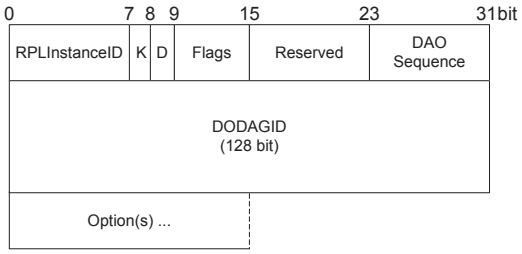


Figure 7: DAO Message Structure

Similar to the DIO message, the DAO message includes an RPL Instance ID. This is the same one that the node has learned from a received DIO. The next field is the Flags field where only the first two bits are used. The first one is the 'K' flag which indicates whether the sender of the DAO expects to receive a DAO-ACK in response. The second one is the 'D' flag which indicates if the DODAGID field is present. Due to the fact that the DODAGID field represents the IPv6 address of the root it may be omitted if there is only one root node present. The DAO Sequence field is a sequence number that is incremented for each outgoing DAO message by the sender. The sequence number ensures the freshness of a DAO message and is echoed back by the parent when DAO-ACKs are used.
The message can be further extended by the use of options. In this paper, only two will be discussed: the Target option and the Transit Information option. The first one is used to indicate a target IPv6 address, prefix or multicast group. In terms of routing, it represents reachability information. RPL defines the structure of it, as shown in Figure 8.

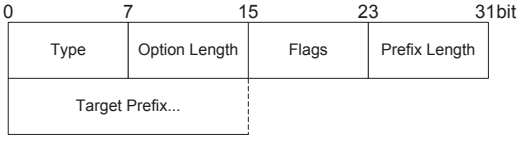


Figure 8: DAO Target Option

doi: 10.2313/NET-2011-07-1_09

First of all, it stores the type of the option (0x05). Afterwards, the length of the option and the length of the prefix are included. In the latter case, the number of valid leading bits of the routing prefix is meant. The last field is the target prefix and it may identify, for example, a single node or a whole group of nodes that can be reached via a common prefix. The Flags field is experimental and is not used.
The second type is the Transit Information option. It is used to indicate attributes for a path to one or more destinations. Destinations are indicated by one or more Target options that precede the Transit Information option(s). Figure 9 outlines its structure.

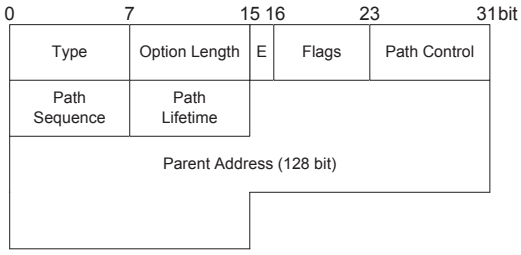


Figure 9: DAO Transit Information Option

The first field represents the type of the option and is always set to 0x06. The second field is the Option Length field which indicates if the Parent Address field is present. Note that in case of storing mode the IPv6 address of the parent may be omitted. The next field is the Flags field where only the first bit is used. The 'E' flag indicates whether the parent router redistributes external targets into the RPL network. Such targets may have been learned by an external protocol. However, they do not play a crucial role for this paper. The next three fields are needed for reachability control. First, the Path Control field is used to limit the number of parents to which a DAO message may be sent. Second, the Path Sequence field indicates if a Target option with updated information has been issued. Third, the Path Lifetime defines how long a prefix for a destination should be kept valid. The time is measured in Lifetime Units and is implementation specific.

6.2 Non-Storing Mode

In the non-storing mode each node generates a DAO message and sends it to the DODAG root. The time generation interval in which DAO messages are sent depends on the implementation. However, the RPL specification suggests that the needed delay between two DAO sending operations may be inversely proportional to the Rank. In this way, if a node is far away from the root it will generate DAOs more often than a node that is closely positioned to the gateway. Furthermore, each node has to extend the DAO message by using the aforementioned Transit Information option. In the Parent Address field the IPv6 address of a parent node is stored. It should be kept in mind that a typical non-storing node may use multiple Transit Information options in order to report its complete parent set to the root node. The resulting DAO message is sent directly to the DODAG root along the default route created during parent selection. Figure 10 illustrates this process.

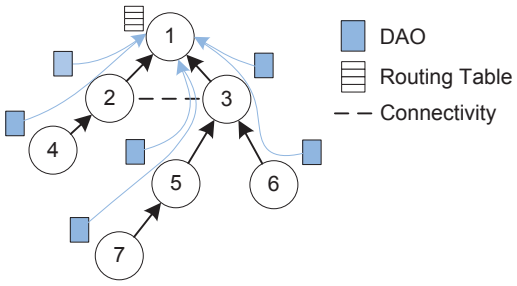


Figure 10: RPL Non-Storing Mode

Usually, intermediate nodes inspect DAO messages and compare the stored path sequence number with the last seen. In this way, a node can distinguish between stale and up-to-date routing information.
After collecting the needed information, the root pieces the downward route together. If it needs to send a data packet to a given destination the IPv6 Source Routing header is used. Thus, network nodes can easily forward a data packet until it reaches the given destination or the IPv6 Hop Limit reaches 0.

6.3 Storing Mode

Similar to the non-storing mode, the storing mode also requires the generation of DAO messages. The configuration of the timer triggering such messages may be implemented in the same way as it was mentioned above. However, a DAO is no longer propagated to the DODAG root. Instead, it is sent as unicast to all parent nodes which maintain additional downward routing tables. Figure 11 gives an overview of this process.

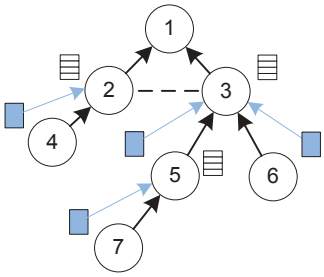


Figure 11: RPL Storing Mode

When a node sends a DAO message it has to keep the Parent address field in the Transit Information option empty since a node's responsibility is not to advertise its parent set, but to announce prefixes that are reachable through it. If the device is a router it has to use the Target option in order to advertise a prefix. In case it has multiple prefixes to advertise, it must extent the DAO by multiple Target options.
If a data packet is sent from the DODAG root node, it must be sent only to all one-hop neighbors. Afterwards, through table look-up the packet is routed downwards until it reaches its destination or the Hop Limit in the IPv6 header reaches 0. Such networks may also be hierarchically organized and route aggregation may be performed. Moreover, a sub-DODAG may run another RPL Instance which makes it possible to

doi: 10.2313/NET-2011-07-1_09

combine storing and non-storing mode. In this way, nodes with constraints regarding memory may be grouped together and operate only in non-storing mode.

7. PERFORMANCE EVALUATION

Gnawali et al. [8] introduced a RPL implementation, called TinyRPL. In their work they use the Berkeley Low-power IP (BLIP) stack in TinyOS 2.x [15] which interacts with their protocol implementation. More precisely, the control plane of TinyRPL communicates with the BLIP stack which offers a forwarding plane implementation. It should be kept in mind that BLIP also offers an implementation in TinyOS of a number of IP-based protocols such as TCP and UDP. In this way, during one test run several transport protocol configurations can be evaluated and compared.
TinyRPL is further compared with the Collection Tree Protocol (CTP) [5], the de-facto routing protocol standard for TinyOS. For this purpose, a 51-node TelosB [3] testbed scenario is build where only one node is acting as a root. Overall, they use two network configurations: one that has a data packet generation interval of 5 seconds and another that has a data packet time interval of 10 seconds. For each of them the protocols are tested against each other and an estimation of the packet reception ratio and the average number of control packets is made. Each testbed run lasts 24 hours. Since CTP uses ETX for topology maintenance, the OF of TinyRPL is set to use the same method for metric computation. In order to make a fair comparison the downstream routing options are disabled since the standard CTP configuration does not define a mechanism similar to the one realized with DAO messages.
In all test runs both protocols managed to achieve a reception rate of approximately 99 %. The amount of control traffic reported also stays at the same level since both protocols make use of the trickle timer and are evaluated in static scenarios.

8. CONCLUSION

LLNs and, in particular, WSNs are rapidly emerging as a new type of distributed systems, with applications in different areas such as target tracking, building environment, traffic management, etc. However, to achieve a reliable communication, to guarantee a high delivery ratio and to be at same time energy efficient requires special mechanisms realized at the network layer.
As a result, RPL was specified and developed in order to overcome these requirements. The protocol is an end-to-end IP-based solution which does not require translation gateways in order to address nodes within the network from the outside world. Moreover, the use of IPv6 allows deploying RESTful web services for sensor networks [16]. In this way, a client (e.g. PC connected to the root) may initiate requests by using HTTP to nodes within the network which will return the appropriate responses. It is even easier to use such a feature, since RPL defines in its specification the support of downward traffic. Because of P2MP a root node can easily propagate such requests to the nodes.
It should also be mentioned that RPL may run on nodes that have limited energy and memory capabilities. The protocol dynamically adapts the sending rate of routing control messages which will be frequently generated only if the network is in a unstable condition. In addition, the protocol allows the use of source routing when P2MP is needed which reduces the memory overhead on intermediate nodes.
RPL also allows optimization of the network for different application scenarios and deployments. For example, it may take into account the link quality between nodes or their current amount of energy which makes it an efficient solution for WSN deployments.

9. REFERENCES

[1] N. Baccour, A. Koubaa, M. B. Jamaa, H. Youssef, M. Zuniga, and M. Alves. A comparative simulation study of link quality estimators in wireless sensor networks. Technical Report TR-09-03-30, open-ZB, 2009.

[2] A. Cerpa, J. Wong, L. Kuang, M. Potkonjak, and D. Estrin. Statistical model of lossy links in wireless sensor networks. In *Information Processing in Sensor Networks, IPSN 2005*, pages 81–88, 2005.

[3] Crossbow Technology. TelosB datasheet. http://www.willow.co.uk/TelosB_Datasheet.pdf, 2010.

[4] A. Dunkels and J. P. Vasseur. Why IP. IPSO Alliance White Paper #1, 2008.

[5] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. Technical Report SING-09-01, Stanford University, 2009.

[6] Internet Assigned Numbers Authority (IANA). Internet Control Message Protocol version 6 (ICMPv6) Parameters. http://www.iana.org/assignments/, 2011.

[7] M. Kim, J. P. Vasseur, K. Pister, N. Dejean, and D. Barthel. Routing Metrics used for Path Calculation in Low Power and Lossy Networks. Internet draft, http://datatracker.ietf.org/wg/roll/, 2011.

[8] J. Ko, S. Dawson-Haggerty, O. Gnawali, D. Culler, and A. Terzis. Evaluating the performance of RPL and 6LoWPAN in TinyOS. In *Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks (IP+SN)*, 2011.

[9] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *The International Workshop on Parallel and Distributed Real-Time Systems*, 2006.

[10] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks. In *Proceedings of the USENIX NSDI Conference*, pages 15–28, San Francisco, CA, USA, 2004.

[11] Y. Li, J. Harms, and R. Holte. Impact of lossy links on performance of multihop wireless networks. In *Computer Communications and Networks, 2005. ICCCN 2005*, pages 303–308, 2005.

[12] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. RFC 4944: Transmission of IPv6 packets over IEEE 802.14 networks, 2007.

[13] S. J. Roundy. *Energy scavenging for wireless sensor nodes with Focus on vibration to electricity conversion.* PhD thesis, University of California at Berkeley, 2003.

[14] Routing Over Low power and Lossy networks (ROLL). Description of Working Group. http://datatracker.ietf.org/wg/roll/charter/, 2011.

doi: 10.2313/NET-2011-07-1_09

[15] The TinyOS Alliance. Poster abstract: TinyOS 2.1, adding threads and memory protection to TinyOS. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, Raleigh, NC, USA, 2008.

[16] TinyOS community. BLIP tutorial. http://docs.tinyos.net/index.php/BLIP_Tutorial, 2010.

[17] J. P. Vasseur. The Internet of Things: Dream or Reality. SENSORCOMM 2010: The Fourth International Conference on Sensor Technologies and Applications, 2010.

[18] J. P. Vasseur, N. Agarwal, J. Hui, Z. Shelby, P. Bertrand, and C. Chauvenet. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. IPSO Alliance, 2011.

[19] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. P. Vasseur. RPL: IPv6 Routing Protocol for Low power and Lossy Networks. ROLL Working Group, 2011.

[20] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, pages 1–13, New York, USA, 2003.

doi: 10.2313/NET-2011-07-1_09

IP-based Communication in Wireless Sensor Network

Christian Fuchs
Betreuer: Dr. Alexander Klein
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: fuchsch@in.tum.de

ABSTRACT

IP is the standard communication protocol in the Internet. Several attempts have been made to port the TCP/IP to Wireless Sensor Networks, because it would ease data acquisition from external applications. This would make it possible to integrate applications more easily into existing networks. On the other hand, IP was initially designed for non resource constraint systems. This causes a significant decrease in bandwidth mostly because of its large header data. This paper will look first at the 6loWPAN specification and afterwards at two widespread implementations of the IP stack for sensor nodes, uIP and BLIP, and compare their compatibility with other implementations as well as their impact on the bandwidth in an example setup.

Keywords

IP over 802.15.4,WSN,uIP,BLIP,6loWPAN

1. INTRODUCTION

Handling communication efficiently and with low energy consumption is one of the most important tasks, when it comes to the programming of nodes that are to be used in Wireless Sensor Networks(WSNs). Up to date there are different approaches on how to handle communication in such networks, like protocols specifically designed for the use on resource constraint nodes. Despite this there are also urges to use TCP/IP on these nodes to make them directly accessible via the Internet[9][5][14].

Using IP in Wireless Sensor Networks holds many opportunities for reasons of direct communication with WSNs over the Internet as well as remote access to the Sensors' data. On the other hand, most platforms used in WSNs are highly resource constraint in terms of memory as well as computing power. These constraints pose various challenges when implementing IP on these platforms, because IP was initially developed for platforms without these constraints. Therefore, the standard IP-Stack implementations used in Linux or Windows can not be directly adapted for the use in WSNs[7], since they require RAM in order of some megabytes, an amount that is not available on typical sensor motes. Also TCP/IP communication consists of point-to-point flows between arbitrary nodes, whereas most traffic in WSNs is for collecting data from the sensors and sending it to a remote server.

Also most WSNs use IEEE 802.15.4 radio for communication which only offers a limited bandwidth compared to other WLAN technologies. Again, IP was not designed with such constraints in mind leading to IP headers, which seem to be too large to be efficiently used in environments with limited bandwidth. Due to this, the IETF has presented 6loWPAN in [15] which describes techniques to cut down the large overhead caused by using IPv6, to enable its usage in the context of an IEEE 802.15.4 environment.

This leads to different approaches on how to bring TCP/IP to WSNs ranging from the use of a proxy server and smaller versions of the IP protocol to the development of smaller IP stacks [2].

In the following sections this paper will take a closer look at two well known implementations of this standard, uIP which has been created by Adam Dunkels as a part of the Contiki operating system[7] and BLIP which was created for TinyOS [5]. The paper will continue in the following way: Section 2 will give a brief overview of the IEEE 6loWPAN standard for IPv6 in Wireless Sensor Networks. Section 3 and 4 will discuss the uIP and BLIP implementations in greater detail. After this there will be some Case Studies with these protocol stacks in Section 5 and finally section 6 concludes the paper.

2. 6LOWPAN

Since Wireless Sensor Networks consist of mostly battery powered motes, energy efficiency is a very important requirement for the used hard- and software. Therefore, the IEEE 802.15.4 standard for data transmission in *low power wireless personal area networks*(loWPAN) was developed. Because typical packets sent in such a network are quite small and for the reason that most power is consumed by sending packets, this standard proposes a data rate of only 250 kbit/s. While this is perfectly satisfactory for specifically designed protocols, which add only a small overhead to the transmission, it issues a challenge to the use of IP in WSNs. Especially IPv6 would consume most of the available bandwidth just with its headers, if used unmodified. To face this challenge 6loWPAN, an adaptation layer between the link and network layer, was introduced by the IETF in [15]. Its main contribution is the introduction of an alternative header format which supports header compression as well as fragmentation of IPv6 packets into multiple link-level frames [12].

6loWPAN consists of several distinct headers, each serving a different purpose. The headers can be stacked on each other, making it possible to skip the unneeded headers. The dispatch header indicates which network layer protocol is used, like plain IPv6 or 6loWPAN. The fragmentation header is used if a single IPv6 packet is too large to be sent over radio in one step. It contains information on how the packet

doi: 10.2313/NET-2011-07-1_10

was fragmented and should be reassembled. And the mesh addressing header serves for layer 2 routing.

2.1 Header Compression

The most important step to efficiently use IPv6 in WSNs is to cut down the transmission overhead caused by the seemingly large IPv6 headers. The standard IPv6 header without any extension headers is 40 bytes large. The largest part of this header are the source and destination addresses, having 16 bytes each. To reduce header sizes either stateless or stateful compression techniques could be used. Using states takes advantage of the fact that certain fields do not change their value in a communication flow between two senders[13]. On the other hand, stateless header compression uses common values of certain fields and information that is stored redundantly across layers to reduce header sizes[15]. Hui and Culler explained in [12] that compression techniques using states are not very effective in loWPANs, since it only pays of in long transmission flows, which typically do not occur in loWPANs. This is the major reason why 6loWPAN uses the stateless header compression format HC1. To denote what kind of header is used for the particular packet, an 1 byte large dispatch header is inserted before the layer-3 header.
One major technique is the use of common values in certain fields[19]. For example, in most packets only TCP,UDP and ICMP are used as upper layer protocols. With this knowledge it is possible to reduce the size of the next header field to only two bits. To manage cases where fields do not carry such a common value, there are reserved values to denote an uncompressed field in the header[12].
To further lessen the header size, 6loWPAN exploits the fact that some information carried in the header can be derived from fields in other layers. For example, the payload length field or the interface identifiers can be determined by the fields in the 802.15.4 header. The only field which must always be carried inline is the 8-bit hops-left field[15], to avoid packets circulating in the network infinitely.
With the exploitation of all these redundancies and common values the IPv6 header can be cut down to 2 bytes only, 1 byte denoting the common values in the original header and 1 byte for the hops-left field. Additionally, an 1 byte large dispatch header is added before the IP header, to denote if the following header is a compressed IP header or native an IPv6 header. If, however, some of the fields cannot be compressed they must be carried inline. For example, when the interface identifier cannot be derived from the layer-2 address, it must be carried inline. In these cases the compressed header would be bigger than 2 bytes. Figure 1 shows the HC1 header with inline IP fields.
By eliding the length field 6loWPAN can also compress UDP to 4 bytes, if source and destination port have a predefined well known value.

2.2 Fragmentation

IPv6 also relies on a Maximum Transmission Unit(MTU) of at least 1280 Bytes, which means that packets smaller than this can be sent without fragmentation. On the other hand, the maximum length of 802.15.4 frames is 127 bytes due to the 7-bit length field in the 802.15.4 frame header. This means there has to be a mechanism for fragmentation on link-layer transparent to higher layers.
As described in [15] 6loWPAN achieves this by splitting up the packets in several frames. Each of these frames has an additional fragmentation header, containing the actual size of the packet, an unique identifier, which is the same for all frames that belong to the same packet, and an offset field, denoting which part of the packet the frame contains. To further reduce the size of the header, the first frame elides the offset field[12]. This results in an additional overhead of 4 Bytes for the first frame respectively 5 bytes for each following frame.

2.3 Routing

Another issue that has to be considered when using IPv6 in a loWPAN, is that sometimes link layer routing is required. Normally, routing would be done on the network layer(layer 3). Since the HC1 format for header compression is optimized for link-local communication[12], it is desirable to do the routing on layer 2.
To solve this issue 6loWPAN has a concept named mesh under. It adds the mesh addressing header, which simply holds the packets source and destination address as well as an hops-left-counter that is decreased by every forwarding node[15]. With this it is possible to route packets over the link layer transparently to the upper layers. In this case the network topology looks like all nodes are in a single broadcast domain to the network layer. Hence, only link-local addresses are needed for communication within the network. This header adds another 5 bytes of overhead.

2.4 Compressed Header Sizes

6loWPAN compresses the IPv6 header to only 2 bytes. On the other side it adds only an 1-2 byte dispatch header. This means an IPv6 packet using UDP as a transport protocol, which would add 44 bytes overhead to the raw payload, if transmitted uncompressed, is compressed by 6loWPAN to only 7 bytes(1 dispatch + 2 compressed IPv6 + 4 UDP)[12]. Even if the other headers for fragmentation and mesh addressing are added, the overhead would not exceed 17 bytes, which means still half of the overhead size can be saved. Figure 1 shows a frame with mesh routing, fragmentation and header compression.

3. UIP

uIP was developed by Adam Dunkels[7] along with the Contiki operating system for tiny sensor networks[8] in 2003. It was one of the first fully working IP stacks for resource constraint systems. In the first version uIP already supported all mandatory IPv4 features as requested by [17], and omits only some of the lesser used features like IP options, thus being fully compatible to any other IP-Stack like the BSD stack implementation[7]. In 2008 uIP has been extended to support IPv6 as well[10]. Up to now uIP is the smallest implementation of a complete IP stack.

3.1 Attributes

In order to cut down memory consumption to an absolute minimum, the uIP implementation uses a shared buffer for incoming and outgoing packets. This buffer is only big enough to hold one packet of maximum size. uIP holds one packet in the buffer at a time and overwrites the buffer every time a new packet arrives, but not before the application has processed the data. As most radio or ethernet

doi: 10.2313/NET-2011-07-1_10

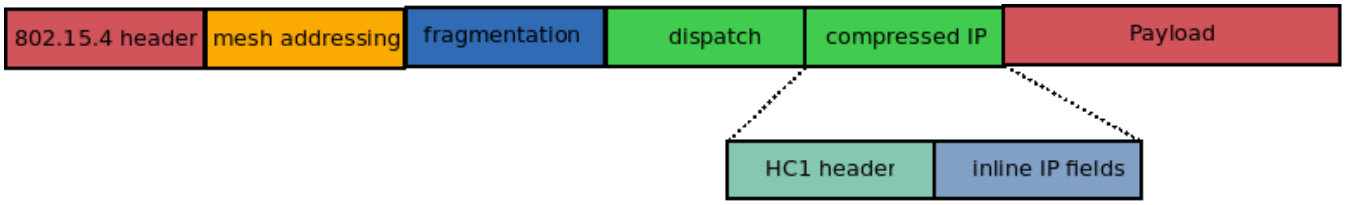


Figure 1: 802.15.4 frame with all 6loWPAN headers references [12]

controllers have internal memory to buffer about 5 packets, this behavior will not lead to packet loss, if more than one packet arrives, before the application is able to process the data[7]. Another consequence is that outgoing packets may be overwritten right after being sent. Furthermore, uIP does absolutely no dynamic memory allocation.
uIP consists of an implementation of IP as a network layer and UDP or TCP as a transport layer. The link layer is not specified and can be chosen independently. It is possible to use 6loWPAN as well as any other implementation.
uIP is fully compatible with other TCP/IP implementations since it supports all features that are needed for host-to-host communication as required by RFC 1122[3]. Additionally, uIP can reassemble only one fragmented packet at a time, for the reason that reassembling has to be done in a separate buffer. On the other hand, not all features for interfacing between the application and the network stack are implemented. As a result of this uIP does not support soft error reporting or dynamically configurable type-of-service bits[7]. For routing uIP uses the RPL protocol explained in the next section.
For ICMP just the minimal subset of operations, that are mandatory for interoperability, is supported. Meaning that uIP is solely capable of sending echo reply messages, by swapping the source and destination address[7] of incoming echo request messages.
As a transport layer uIP supports both TCP and UDP. Like any other part of the uIP stack the TCP implementation is done to ensure full interoperability, while it omits all features, which are not mandatory, to save memory and quicken the network operations. Listening and connecting ports are supported as well as sending data works normally. But there is no support for a sliding window mechanism, because as mentioned above the packet buffer can only hold a single packet at a time. Additionally, as no sent packets are buffered and any incoming packet would overwrite the packet buffer, retransmission has to be done manually. This means that TCP reports to the application, if a packet was not acknowledged correctly and the application then has to ensure the packet is retransmitted. Congestion Control is elided as it is not needed, too[7].

3.2 Routing

uIP uses the Routing Protocol for Low Power and Lossy Networks(RPL) [1]. This protocol is especially designed for large networks of resource constraint nodes managed by a few central *supernodes* and optimized for multipoint-to-point traffic[4]. RPL is a distance vector protocol that uses a destination orientated directed acyclic graph(DODAG) for routing. One of the *supernodes* serves as the root of the DODAG and all other nodes build up multipoint-to-point routes to this root. Every node in the DODAG has a rank, denoting its distance from the root node, and a set of parent nodes, which are closer to the root. One of this parents serves as the preferred parent.
To create the DODAG each node that is part of the DODAG sends DODAG information objects(DIO) via link-local multicast containing its own rank. A new node, which has received some of this messages, can determine its own rank, which has to be greater than its parents rank, and starts itself sending DIOs. To avoid count-to-infinity problems a node that is part of the DODAG can only lower its rank if it receives a DIO from a parent with a lower rank.
After the DODAG has been formed, messages can be sent to the root by forwarding to the nodes preferred parent. To send messages from the root to nodes in the networks DODAG advertisement objects(DAO) are sent towards the root, while every node that forwards the object adds its address. With this the root node has a source route to the node, that issued the DAO, once it arrives at the root. For node to node communication messages are sent first to the root from where the messages are sent towards the destination nodes via the source routes.

3.3 Usage

Since uIP is designed for the use together with Contiki as an operating system, there are two ways of programming applications that use uIP for communication.
The first way to program with uIP is the event driven interface, which was initially the only way to program with uIP. Here the application is invoked every time an event on the IP stack occurs, like incoming data or a new connection request[7]. After processing the incoming data the program has to hand control back to the uIP stack, so that outgoing packets can be sent and new arriving packets can be processed. This means that the application has to be built like a state-machine, testing on each incoming event what caused the event to be posted. For the purpose of testing which event has occurred the API defines functions such as uip_newdata(), uip_acked() or uip_connected(). On every new event each of this functions has to be called to determine which is the appropriate action to be taken. While this is perfectly sufficient for small programs, it would cause too much overhead in more complex applications. These would be easier to develop and more efficient with an API that allows sequential code. Listing 1 shows a short example code using the event driven API.

```
void example() {
  example_state state;
  if(uip_connected()) {
    uip_send("Welcome\n",9);
  }
```


doi: 10.2313/NET-2011-07-1_10

```
  if (uip_acked()) {
    if (state == WELCOME_SENT) {
      state = WELCOME_ACKED;
    }
  }
  if (uip_newdata()) {
    uip_send("ok",2);
  }
}
```

Listing 1: example program with event driven API references[1]

For this purpose protosockets, which can be thought of as lightweight versions of the BSD sockets, were provided[1]. With this API the networking code can be written sequentially, similar to networking applications on a PC. Like with normal sockets every connection is associated with an protosocket and the protosocket has to be handed over to the underlying operating system every time an action like sending or receiving data has to be carried out. In opposite to the event driven API, where control must be handed back to the stack after processing a packet, protosockets also support blocking calls for receiving data, which is shown in listing 2.

```
PT_THREAD example() {
  PSOCK_BEGIN(s);
  PSOCK_READTO(s,'\n');
  if (strcmp(inputbuffer,"HI") != 0) {
    PSOCK_CLOSE(s);
    PSOCK_EXIT(s);
  }
  PSOCK_SEND(s, "ok", 2);
  PSOCK_CLOSE(s);
  PSOCK_EXIT(s);

}
```

Listing 2: example program with protosocket API references[1]

3.4 Memory Footprint

uIP is highly scalable at compile time. The programmer can for example decide on the number of maximum simultaneously open connections as well as the amount of RAM reserved for the packet buffer. Furthermore, support for TCP, UDP and IPv6 can be deactivated, if not needed, to save further memory.
In a minimal configuration it is possible to run uIP with only 200 bytes of RAM but with this configuration throughput is very low and it is usable only in simple environments that send only very small packets. A configuration that is usable in a more generic scenario would consume about 2 kbytes of RAM. The stack implementation needs about 5.1 kbytes of ROM[7].

4. BLIP

BLIP stands for Berkley low power IP and is a IPv6 stack developed for TinyOS by David Culler[11]. It was first developed under the name b6loWPAN and later renamed to BLIP, because BLIP was at this time already more than just an implementation of 6loWPAN but instead a fully working IP stack. As a part of TinyOS it was written in nesC, an enhancement to the language C especially written for TinyOS. There are implementations of BLIP for micaZ, TelosB and iMotes[16].

4.1 Attributes

Unlike uIP BLIP implements not only network and transport layer, but it also uses b6loWPAN as an adaptation layer between link and network layer. Therefore, BLIP must always use IPv6 for communication. BLIP also supports mesh under routing, explained in section 2.3, meaning that routing is done transparently by the adaptation layer[18].
The network layer uses a routing protocol called HYDRO, which is explained in the following section. Additionally, the network layer is able to do neighbor discovery, using ICMPv6, which is fully supported, as opposed to uIP, which only supports echo messages. Also BLIP can configure link-local addresses and global addresses, either via stateless auto-configuration or via DHCPv6, if a router is reachable[18][11].
The network layer is additionally responsible for retransmitting packets that get lost over a single hop. This enables the stack to reroute the packet if the network topology changed during transmission.
BLIP defines basic Quality of Service(QoS) classes. The most important are to indicate high priority data and contrary to denote latency-tolerant packets, which can be buffered and sent in large bulks of packets for energy efficiency[11].
As a transport protocol it currently supports UDP, but there is already a prototype for TCP[16].

4.2 Routing

BLIP uses the HYDRO routing protocol[6]. It was designed to combine the support for many-to-one traffic, which is needed for data collection, and one-to-one traffic, needed to propagate commands in the network. For the HYDRO protocol there are two kinds of communication points. The sensor nodes, which acquire data and send them to a remote server or communicate in the network, and at least one border router, which connects the network to the outside world. The border router has to store an overview over the complete network and should install routes onto the nodes. This network overview is created from topology reports it gets from the nodes. To hold traffic for creating and maintaining the network overview as little as possible, the topology reports are sent piggybacked with the sensor data. Such a topology report holds information about the node's neighbors and the estimated transfer costs to this neighbor. However, the construction of this global network topology is complicated, due to the fact that the sensor nodes have not enough memory to hold a complete list of their neighbors. This means that the created network topology does not represent the complete network with all its links, but only a subset, which is nevertheless sufficient for routing.
The border router stores this subset of the complete network topology in the so called link state database. If there is more than one border router, they have to be connected in order to share their particular link state database. From this database a best effort route between every two nodes in the network can be derived. This routes are installed in the corresponding node to improve point-to-point communication.
The nodes on the other hand are to constraint in terms of memory to hold a complete routing table. Instead they use

doi: 10.2313/NET-2011-07-1_10

a distributed DAG(directed acyclic graph) to provide them with a reliable route to the next border router. In this distributed DAG every node holds the address of a node nearer to the border router. In practice more than one route to the border router is stored to improve reliability[6]. The nodes also maintain flow tables, which hold the installed routes from the border router.
The nodes can forward packets in 3 steps:

1. If the packet contains a valid source route it is used
2. If there is an entry in the flow table for the desired destination, the previously installed route is used
3. If neither a source route is provided nor a matching entry in the flow table exists, the packet is redirected to the border router, from where it is sent to the correct node

4.3 Usage

The BLIP API is closely related to programming with sockets on UNIX systems. The structs containing the address are defined like in Linux:

```
struct in6_addr
  {
    union
      {
      uint8_t    u6_addr8[16];
      uint16_t  u6_addr16[8];
      uint32_t  u6_addr32[4];
      } in6_u;
#define s6_addr       in6_u.u6_addr8
#define s6_addr16     in6_u.u6_addr16
#define s6_addr32     in6_u.u6_addr32
  };

struct sockaddr_in6 {
  uint16_t sin6_port;
  struct in6_addr sin6_addr;
};
```

Listing 3: structs for IPv6 addresses and port

For programming with UDP BLIP provides the three functions. error_t bind(uint16_t port) is for binding a socket to a specific port and the functions error_t send_to() and error_t recv_from() serve for sending and receiving data.
The TCP implementation is still experimental and cannot accept more than one connection. The API is similar to BSD sockets, with functions to actively connect to an port and passively listening for incoming connection requests.

4.4 Memory Footprint

Together with the HYDRO routing protocol BLIP needs about 2.5 kilobytes of RAM and 9.4 kilobytes of ROM[6][11]. Especially in code size BLIP is significantly bigger than uIP. This is because BLIP also contains the underlying link layer, whereas uIP only implements the network and transport layer. Furthermore, BLIP fully supports ICMPv6 and DHCPv6. Table 1 briefly compares the features supported by uIP and BLIP and table 2 shows the memory usage of both stacks.

	uIP	BLIP
OS	Contiki	TinyOS
IP	IPv4 and IPv6	IPv6 only
TCP	YES	Prototype
UDP	YES	YES
ICMP	echo only	YES
Mesh Under	NO	YES
Route over	NO	YES

Table 1: Comparison between the main features of uIP and BLIP references [18]

	uIP	BLIP
RAM	200 byte - 2 kbyte	1 kbyte
ROM	5.1 kbyte	9.4 kbyte

Table 2: Memory footprint of uIP and BLIP

5. EVALUATION

This section will show the usage of IP in two simple network environments, to examine its impact on throughput and the effectively available bandwidth. For the evaluation we assume typical IEEE 802.15.4 standard parameters, namely a bandwidth of 250 kbit/s and all frames should be of maximum frame length, hence 127 byte. In the following sections plain IPv6 without any compression is compared against compression according to the 6loWPAN standard. UDP is used as transport protocol.
The following formula is used to compute the maximal achievable data rates: $(1 - \frac{headersize}{127byte}) \cdot 250\frac{kbit}{s}$. The term $1 - \frac{headersize}{127byte}$ is the percentage of the frame that can be used for the payload. Multiplied with the available maximum data rate this gives us the data rate for payload transfers.

5.1 Single Hop

The first network layout simulates a link-local transmission from node 1 to node 2. Figure 2 depicts this scenario. Plain IPv6 together with UDP headers add 48 bytes of overhead to every transferred frame. This equals to 37.7 % of the whole frame. Thus the total available data rate is reduced by 34.6 % leaving a total of 155 kbit/s for payload transfer. Opposed

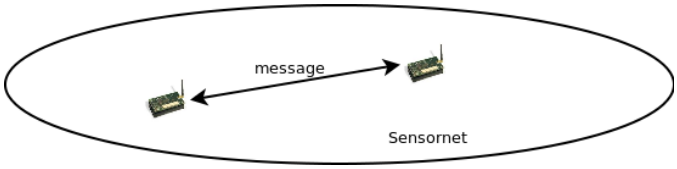


Figure 2: First setup with just two nodes

to this transmission with 6loWPAN adds only 7 bytes of overhead as discussed in section 2.4. This is only 5.5 % of the total frame length, meaning that a data rate of 236 kbit/s is still available for payload transmission. Considering a packet larger than 127 bytes, the packet must be fragmented, which adds the 5 bytes fragmentation header to each frame. With this there are 226 kbit/s left for payload transmission.
These values show that using 6loWPAN pays off even in a single hop transfer, since uncompressed IPv6 consumes nearly one third of the available frame length.

doi: 10.2313/NET-2011-07-1_10

5.2 Multi Hop

In this scenario we consider 4 nodes in a string connection, as shown in figure 3, and we assume a message is sent from node 1 to node 4 over two hops(nodes 2 and 3). In this setup the node's bandwidth is not only consumed by them-selves but also by the next node, which blocks the communication medium while it transmits the message to the next hop. Furthermore, interference plays a role as two nodes may not be close enough to directly communicate with each other, but one node sending may still cause interference on the other node. This leads to another limitation of the effectively usable bandwidth.

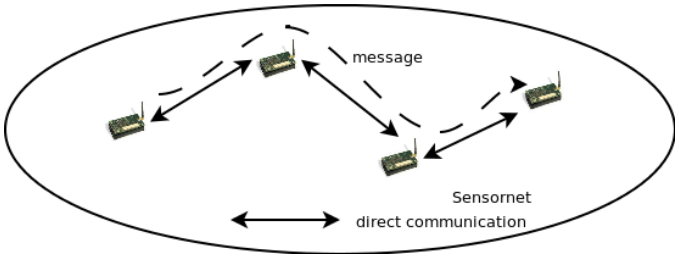


Figure 3: Message being sent over 3 Hops

Normal Interference Radius

Here we assume that transmission and interference radius are the same and a node only blocks the communication channel for the nodes it can directly communicate with. In this scenario the bandwidth for the initial sending node is halved, because for every packet, the node sends, the next hop also has to transmit the packet. Therefore, the node can effectively use only half of its original bandwidth, hence 125 kbit/s.
At the nodes that serve as hops, but not as communication endpoints, each packet consumes three times the bandwidth that one transmission of the packet would cost. This is because at first the packet must be transferred from the previous hop to the node. After that the node has to forward the packet and at last the next hop is also retransmitting the packet, blocking the communication channel for another length of the packet.
Using IPv6 and UDP without any compression would consume 37.7 % of the total frame length. Leaving a data rate of only 78 kbit/s for the payload. With 6loWPAN 118 kbit/s remain. A fragmented packet consumes another 5 bytes for overhead as well as a packet that is routed on the link layer. In this case the data rate effectively available for payload transmission is 113 kbit/s. If both cases, fragmentation and mesh under routing, apply the needed headers consume 13.3 % if the frame length, leaving a data rate of 108 kbit/s.

Double Interference Radius

In a more realistic scenario interference would not only occur between nodes, which could directly communicate with each other, but have a far wider influential radius. For this example setup we consider the sphere of interference to be twice as big as the radius in which actual communication is possible. As shown in figure 4 this would mean that node 1 would not only block the communication channel for node 2 but also for node 3, despite there is no direct communication possible between node 1 and 3. In this scenario the initially

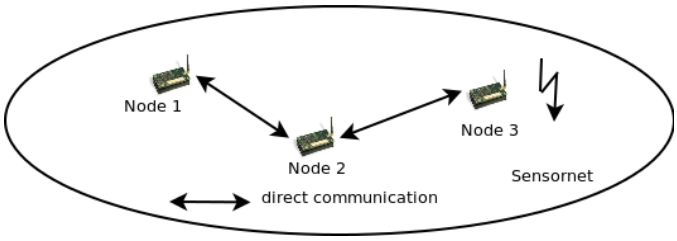


Figure 4: Example for doubled interference radius

sending node has only one third of its original bandwidth for transmission available, because the communication channel is not only blocked from the next node sending, but also from the node hereafter. This means the node has only 83 kbit/s of effective usable bandwidth.
Here plain IPv6 would leave only 51 kbit/s for payload transmission. 6loWPAN on the other hand leaves 78 kbit/s for not fragmented packets, 75kbit/s for fragmented packets and 71 kbit/s for fragmented packets that are routed mesh under.
The different data rates achievable with and without 6loWPAN are summarized in table 3.

6. CONCLUSION

Since the usage of IP in WSNs and their integration into the Internet brings many advantages, a lot of effort has been put to overcome the challenges that using IP in WSNs poses. The 6loWPAN standard made it feasible to use IPv6 in low throughput environments by compressing the relatively large overhead of IPv6 to a few bytes.
Another issue was that most IP stacks were too big to be used on resource constraint nodes with less than 100 kilobytes of RAM. Here uIP and BLIP have shown that it is possible to implement a complete IP stack, which uses only a few kilobytes of RAM and ROM, while being fully interoperable with other IP stacks.
Since traffic in WSNs happens to be mostly multipoint-to-point traffic, in opposite to classic networks, where most traffic is point-to-point, there is still a need for specialized routing protocols. Recently there has been some work in this area, especially by the IETF which has brought up RPL to create a standard routing protocol for WSNs.

7. REFERENCES

[1] http://www.sics.se/contiki/.
[2] J. A. H. R. Adam Dunkels, Thiemo Voigt and J. Schiller. Connecting wireless sensornets with tcp/ip networks. In *Proceedings of the Second International Conference on Wired/Wireless Internet Communications (WWIC2004)*, 2004.
[3] R. T. Braden. Rfc 1122: Requirements for internet hosts — communication layers, Oct. 1989. Status: STANDARD.

doi: 10.2313/NET-2011-07-1_10

	Single Hop	Multiple Hops	Multiple Hops Double Interference Radius
IPv6	155 kbit/s	78 kbit/s	51 kbit/s
6loWPAN	236 kbit/s	118 kbit/s	78 kbit/s
6loWPAN & Fragmentation	226 kbit/s	113 kbit/s	75 kbit/s
6loWPAN & Mesh Under	/	113 kbit/s	75 kbit/s
6loWPAN & Mesh Under & Fragmentation	/	108 kbit/s	71 kbit/s

Table 3: Comparison of data rates in different environments

[4] T. Clausen and U. Herberg. Multipoint-to-point and broadcast in rpl. In *Proc. 13th Int Network-Based Information Systems (NBiS) Conf*, pages 493–498, 2010.

[5] P. A. C. da Silva Neves and J. J. P. C. Rodrigues. Internet protocol over wireless sensor networks, from myth to reality. *Journal of Communications*, 5(3), March 2010.

[6] S. Dawson-Haggerty, A. Tavakoli, and D. Culler. Hydro: A hybrid routing protocol for low-power and lossy networks. In *Proc. First IEEE Int Smart Grid Communications (SmartGridComm) Conf*, pages 268–273, 2010.

[7] A. Dunkels. Full tcp/ip for 8-bit architectures. In *Proceedings of The First International Conference on Mobile Systems, Applications, and Services*, May 2003.

[8] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *First IEEE Workshop on Embedded Networked Sensors*, November 2004.

[9] A. Dunkels, T. Voigt, and J. Alonso. Making tcp/ip viable for wireless sensor networks. In *Proceedings of the First European Workshop on Wireless Sensor Networks (EWSN2004)*, January 2004.

[10] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels. Making sensor networks ipv6 ready. In *Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008), poster session*, Raleigh, North Carolina, USA, Nov. 2008. Best poster award.

[11] J. W. Hui, A. R. Corporation, and D. E. Culler. Ip is dead, long live ip for wireless sensor networks. In *The 6th International Conference on Embedded Networked Sensor Systems (SENSYS'08)*, pages 15–28. ACM, 2008.

[12] J. W. Hui and D. E. Culler. Extending ip to low-power, wireless personal area networks. *Internet Computing, IEEE*, 12(4):37–45, July-Aug. 2008.

[13] V. Jacobson. Rfc 1144: Compressing tcp/ip headers for low-speed serial links, Feb. 1990. Status: PROPOSED STANDARD.

[14] W. F. Karl Mayer. Ip-enabled wireless sensor networks and their integration into the internet. In May, editor, *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, 2006.

[15] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of ipv6 packets over ieee 802.15.4 networks. RFC 4944 (Proposed Standard), Sept. 2007.

[16] K. Nithin. Blip: An implementation of 6lowpan in tinyos, November 2010.

[17] J. Postel. Rfc 791: Internet protocol, Sept. 1981. Status: STANDARD.

[18] F. B. Ricardo Silva, Jorge S Silva. Evaluating 6lowpan implementations in wsns. In *CRC '09: Proceedings of the 9Âł ConferÃłncia sobre Redes de Computadores*, 2009.

[19] C. Westphal and R. Koodli. Stateless ip header compression. In *Proc. IEEE Int. Conf. Communications ICC 2005*, volume 5, pages 3236–3241, 2005.

TCP/IP communication in a WSN

Oliver Gasser
Betreuerin: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: gasser@in.tum.de

ABSTRACT
In this paper, the current state of TCP/IP adaption in wireless sensor networks (WSNs) is surveyed. WSNs are becoming more and more ubiquitous for all kinds of monitoring applications and also for ad-hoc networking. TCP/IP on the other hand is the de-facto networking standard. In its pure form, however, the protocol suite does not perform well in a WSN in terms of energy consumption and other factors. Fortunately there are several approaches that try to adapt TCP/IP to sensor networks which will be presented in this paper.

Keywords
TCP, TCP/IP, WSN, Sensor Network, Caching, Energy Efficiency, Link Quality, MSS

1. INTRODUCTION
Wireless Sensor Networks are becoming ubiquitous not only in the scientific world but also more and more in your home (e.g. monitoring the temperature of rooms, alerting of an upcoming rain shower, etc.). Thus the communication between sensor networks and other networks is getting more important too. The de-facto networking standard protocol suite is TCP/IP. While it is possible to run TCP/IP on sensor nodes [7] it is not feasible (in terms of energy consumption) to run pure TCP/IP on them. The main factor for this is that TCP/IP was not designed to be used in wireless environments but rather in wired ones. The error rate in wireless environments is much higher and TCP cannot distinguish between losses due to congestion and losses due to bit errors. Another drawback of TCP/IP in WSNs are end-to-end retransmissions which are a huge energy waster if they are not kept to a minimum. That is why TCP/IP can and should be adapted to sensor networks. In this paper approaches to tailor TCP/IP to the characteristics of a WSN are presented.
The remainder of this paper is structured as follows. Section 2 explains in more detail the specifics of TCP/IP and wireless sensor networks. Section 3 surveys the challenges which are faced when making sensor nodes TCP/IP-ready and Section 4 lists and analyzes proposed approaches to solve the problems discussed in the previous section. Finally, this paper is concluded in Section 5, where the proposed approaches are rated.

2. TCP/IP IN A WSN
2.1 Wireless Sensor Networks
Wireless Sensor Networks (WSNs) are becoming more and more omnipresent. Nodes in WSNs are mainly used for monitoring purposes such as environment monitoring (air pollution, temperature, humidity etc.) and machine monitoring (observing a machine's condition or health). Another application are mobile ad-hoc networks, where two computers (or similar devices) communicate with each other via WSN nodes without a router or any other kind of access point.
Due to their compact nature nodes in these networks have certain characteristics and constraints:

- Nodes should function unattendedly (except changing the battery every year or so).
- Energy consumption (CPU, sending, receiving) needs to be kept at a minimum to save battery power.
- Resources (CPU power, memory, energy) are scarce.
- The network must adapt to churn (i.e. nodes joining and leaving due to failure).
- A node's location can possibly change affecting the network's topology.

These characteristics on the other hand limit the nodes' universal applicability, i.e. sensor nodes are not able to run resource-consuming software as a generic PC would be able to. That is why the software for sensor needs to be specifically tailored to their abilities.

2.2 TCP/IP
For communication in the Internet and many other networks the Internet Protocol Suite is most often used. The two most important protocols from this family are the Transmission Control Protocol (TCP)[18] and the Internet Protocol (IP) [17]. Those two protocols are commonly referred to as TCP/IP and they are the de-facto standard protocols for Internet communication. TCP/IP is one reason why the Internet has become a big success story and is nearly ubiquitous today.
However, due to various design issues laid out in detail in Section 3, TCP/IP is not mainly used for WSN communication. Instead nodes run other WSN specific protocols. Despite that fact there are certain scenarios where it would be preferable to use TCP/IP in a WSN. The main gain of

doi: 10.2313/NET-2011-07-1_11

running TCP/IP on sensor nodes is the ability to easily communicate from any other TCP/IP compatible device with any single node. This could be used to download gathered sensor data, reconfigure the node without directly attaching it to a PC or updating the software running on the node "over the air". Another scenario could be to use the WSN as an ad-hoc network to connect two TCP/IP devices which are otherwise unable to communicate with each other as they are not directly connected and have no access to the internet.

Internet Protocol

The Internet Protocol is an unreliable, connectionless layer 3 (i.e. network layer) protocol. That means that no connetion setup is performed and it is not guaranteed that packets arrive at the destination. The Internet Protocol exists in two versions: IPv4 and IPv6 [6]. Version 4 is predominantly used as of 2011, but the adoption of IPv6 is continuously increasing. IPv6 was primarily introduced to solve the problem of the exhaustion of IPv4 addresses. Since IPv4 addresses are 32 bits long therer are around 4 billion possible addresses. IPv6 on the other side features 128 bit long addresses which makes a total of 10^{38} different addresses. This increase in addresses, however, did not come for free: The IPv6 header is double the size of an IPv4 header, without extensions or options: 40 bytes vs. 20 bytes. In wireless sensor networks where most of the time only a few bytes are transferred and the link layer segment size is limited to a little more than 100 bytes, the 40 bytes of the IPv6 header alone would be too great of an overhead. Fortunately compression algorithms can reduce the IPv6 header's size to about 20 bytes [13]. Should the packet still not fit in a link layer frame it can be fragmented using the LoWPAN protocol layer [14].

Transmission Control Protocol

The Transmission Control Protocol is a reliable, connection-oriented protocol, in contrast to IP. It operates on layer 4 (transport layer) and establishes a connection between two TCP endpoints. Once the connection is established it needs to be managed. This management effort consists of making sure that all TCP segments are reliably transmitted from source to destination (retransmission on loss), detecting transmission errors, potential reordering of packets before delivering them to the destination process, detecting and reacting to network congestion, etc. Although these operations work very well on wired information systems and on wireless systems with no energy scarcitiy (e.g. home WiFi network), they do not work well in WSNs. The reasons for that are the sensor nodes' limited resources and the high error rates in wireless networks. An end-to-end retransmission after a bit error wastes too much energy since the packet has to be sent and received over the whole path again. TCP's reaction to packet loss is not optimal as it is interpreted as network congestion and the sending rate is reduced. This are just two of the problems, which will be discussed in the following section, of operating TCP in a WSN.

3. ISSUES OF ADOPTING TCP/IP IN A WSN

There are several issues which need to be solved, before TCP/IP is a viable protocol combination to be used in a WSN.

One issue is the header overhead. TCP and IP headers combined have a minimal size of 40 bytes: 20 bytes TCP header plus 20 bytes IPv4 header, without any additional options. If we look at the maximum size of link layer frames a significant part of the message is being occupied by header data. The IEEE 802.15.4 standard [11] for example limits the maximum size of link layer frames to 127 bytes. That leaves a mere 87 bytes of TCP payload even without taking the link layer header into account. The headers therefore occupy more than 30 % of the total maximum possible data which can be sent in one frame. The nodes' scarce energy resources are thusly not utilized in an optimal way. Additionally it should be noted that larger payloads can be fragmented into many packets. Fragmentation and reassembly, however, are themselves energy consuming processes.

The greatest hurdle which hinders TCP/IP from being widely adopted in wireless sensor networks is TCP's flow and congestion control mechanism. TCP is unable to differentiate between a lost segment due to congestion and a lost segment due to bit errors. Whenever a segment is lost, i.e. no acknowledgment is received for that particular segment and a timeout event is triggered, this loss is believed to be due to congestion in the network. Consequently the sending rate is reduced to avoid further segment losses. While this might be a good strategy in wired networks it certainly is not appropriate for wireless sensor networks, where bit error rates are orders of magnitude higher (up to double digit percentage package error rates [1]). Despite the fact that the loss occured due to bit errors the sending rate is reduced nevertheless. This leads to a less than ideal throughput.

Another issue with TCP in WSNs is TCP's connection management. Connections are maintained between the two endpoints of communication. If a packet gets lost in transit, a timeout occurs at the senders side or a duplicate acknowledgemt is received. The sender then retransmits the original packet to the receiver and hopes that it will go through this time. This behaviour is wasting the nodes' energy by forcing expensive retransmission on the whole path from sender to receiver.

In traditional IP networks unique IP addresses are assigned to each network interface based on the network's topology. This process of address assignment is either done manually or automatically (e.g. via DHCP). Assigning addresses this way is not very practical for sensor nodes. Furthermore sensor networks often prefer data-centric routing mechanisms instead of traditional address based routing [5]. A receiver simply announces its interest in a certain kind of data instead of nodes directly addressing a data sink.

Finally the energy, memory and CPU resources of a sensor node are very limited and it may be unfeasible to run a full TCP/IP stack on them. This, however, was proven to be possible [3, 7].

4. POSSIBLE SOLUTIONS

In this section possible solutions to the challenges faced in TCP/IP and WSN, which were discussed in Section 3, are presented.

4.1 Different scenarios and approaches

Dunkels et al. [8] listed three possible ways to connect sensor networks with TCP/IP networks: (1) via a proxy, (2) via DTN overlays or (3) implementing TCP/IP directly on

doi: 10.2313/NET-2011-07-1_11

sensor nodes.
With the first method (see Figure 1) a proxy which resides on a gateway machine "translates" all TCP/IP packets directed to the sensor network to conform to the sensor network protocols. The exact opposite is done for packets sent from the WSN to the TCP/IP networks. Although this approach does not implement TCP/IP on the sensor nodes themselves it still makes it possible to access WSNs from a TCP/IP network. This approach has the advantage of being relatively simple to set up. Moreover security policies can easily be enforced on the gateway proxy. Drawbacks of this setup are the single point of failure nature of the proxy machine and the fact that different WSNs need different proxy implementations.

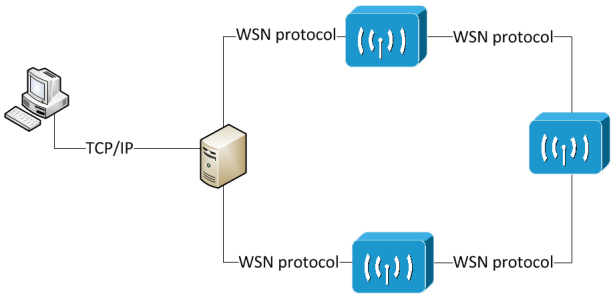


Figure 1: Proxy architecture.

Delay Tolerant Networks (DTNs) [10] are especially designed for environments with high bit error rates, long and changing delays, high churn and asymmetrical data connections. DTNs implement a network overlay and transmit messages (called "bundles") based on store-and-forward switching, i.e. a bundle is held available until the next hop confirms its reception. This avoids costly end-to-end retransmissions. A DTN is partitioned into regions and each region has one DTN gateway. This gateway is responsible for sending messages to other regions and to nodes in its own region. The DTN architecture can be seen as a generalization of the proxy approach.
To enable seamless integration between a TCP/IP network and a WSN the TCP/IP protocol stack should directly run on each sensor node. No gateways or other special nodes are needed in this approach. A visual representation can be seen in Figure 2. This approach, however, faces some issues which need to be addressed: Host-centric routing and addressing, large header overhead for TCP/IP, bad performance over links with a high bit error rate, end-to-end retransmissions. Possible solutions for these issues will be presented in the rest of this section.

Kuorilehto et al. [12] also list three different possibilities of integrating TCP/IP into WSN: (1) direct TCP, (2) proxy TCP and (3) native TCP.
The first approach is the same as the last method mentioned by Dunkels et al., Kuorilehto's second method corresponds to the first method by Dunkels et al. (see above).
Lastly the native TCP architecture transports TCP/IP traffic as a payload of the WSN protocols (see Figure 3). Their implementation called TUTWSN uses Time Division Multiple Access (TDMA, sending is only allowed in dedicated time slots) to avoid collisions which would lead to expensive re-

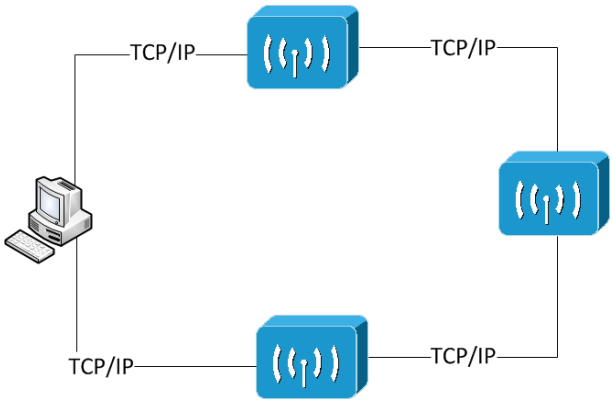


Figure 2: Direct TCP/IP implementation on sensor nodes.

transmissions. This setup allows for communications of two or more TCP/IP endpoints over a WSN with a gateway at each entry point. Direct TCP/IP communication with sensor nodes is not possible. In TUTWSN the sensor network is clustered, each cluster having one "cluster headnode". These headnodes control the communication within the cluster and cluster-to-cluster communication. The other nodes are called "subnodes" and cannot communicate directly with each other but rather via a cluster headnode. Each cluster communicates on a dedicated cluster channel which does not overlap with neighboring clusters. Routing decisions are made based on cost-gradients to a gateway. These costs could depend on the number of hops, remaining energy, number of associated nodes, power necessary to send a packet to the next hop, etc. Kuorilehto et al. suggest that the TDMA's idle periods could also be used for sending data which would result in a higher throughput. This, however, leads to more energy being consumed.

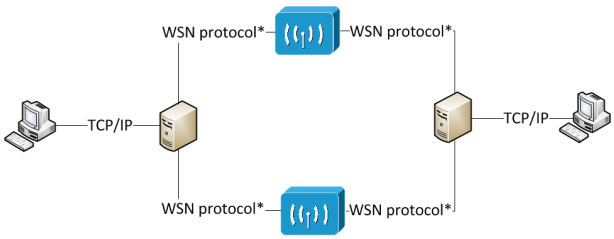


Figure 3: Native TCP/IP implementation transporting TCP/IP packets as payload (marked by a star).

4.2 Distributed TCP Caching

In this subsection and the following one, two similar approaches to make TCP in sensor networks more energy efficient [5] are presented: Distributed TCP Caching (DTC) [9] and TCP Support for Sensor networks (TSS) [4] which are both inspired by the Snoop protocol [2]. These approaches reduce the number of retransmissions by allowing intermediate nodes to cache packets and in case of packet loss do local retransmissions. The performance of both approaches is discussed in Section 4.4.

The basic idea in DTC is that intermediate nodes cache seg-

doi: 10.2313/NET-2011-07-1_11

ments on their way and retransmit them locally to avoid costly end-to-end retransmissions. In the best case each node would cache all bypassing segments. In reality, however, this cannot be achieved as the nodes have only a very limited amount of storage. Thusly it is vital that the nodes very carefully select which segment to cache, hopefully caching those which will be lost. If a node receives a segment with the highest segment number seen until then it caches this segment with a certain probability. If the segment number is lower it is not cached. That behavior assures that not only the newest but also older segments are kept in the cache. If a TCP segment gets lost in transit to the next hop, the segment is locked in the cache. It will therefore not be overwritten by TCP segments with a higher sequence number. Link layer acknowledgements are used in order to detect segment loss. This could also be done with "overhearing", i.e. listening if the next hops forwards the segment. A cached segment is unlocked as soon as a TCP ACK acknowledging this very segment is received or when the segment times out. To avoid that a packet loss triggers a end-to-end retransmission DTC detects the loss before the TCP endpoint does. Each node maintains a soft state for passing TCP connections. This state saves the round-trip time (RTT) to the receiving node and sets the local retransmission timeout to $1.5 \cdot RTT$. The local retransmission timeout values are smaller for nodes close to the TCP receiver and get larger the nearer you come to the TCP sender. Since the local retransmission timeout is believed to be smaller than the TCP sender timeout local retransmissions kick in as a packet is lost and end-to-end retransmissions are avoided. Whenever a sensor node locks a segment in the cache the local retransmission timer is started. If it ends before the packet is unlocked, the segment is retransmitted.
To detect packet loss and for signaling purposes DTC uses the TCP selective acknowledgement (SACK) option [15]. If a node receives a TCP ACK there are two possibilities:

1. The acknowledged segment number is larger or equal to the cached segment, then the cache can be cleared.
2. The acknowledged segment number is smaller than the cached sequence number, then there are two subpossibilities:
   (a) If the cached segment's sequence number (`cached`) is not in the SACK block, the cached segment is retransmitted and `cached` is added to the SACK block. If all sequence numbers in the SACK block are contiguous the TCP ACK is dropped altogether, as all missing segments have been retransmitted and forwarding the ACK to the sender would trigger an unnecessary retransmission. If the SACK block is not contiguous the ACK is forwarded in the direction of the TCP sender.
   (b) If `cached` is in the SACK block, the node's cache can be cleared since the TCP receiver either already got this segment or it is locked in the cache by a node which is closer to the TCP receiver. Finally the TCP ACK is forwarded in direction to the TCP sender.

DTC has the ability to locally regenerate TCP acknowledgements without caching or otherwise storing them. If a node receives a TCP segment for which it already saw a TCP ACK the TCP segment is dropped and a TCP ACK is locally regenerated using the TCP connection's state information.

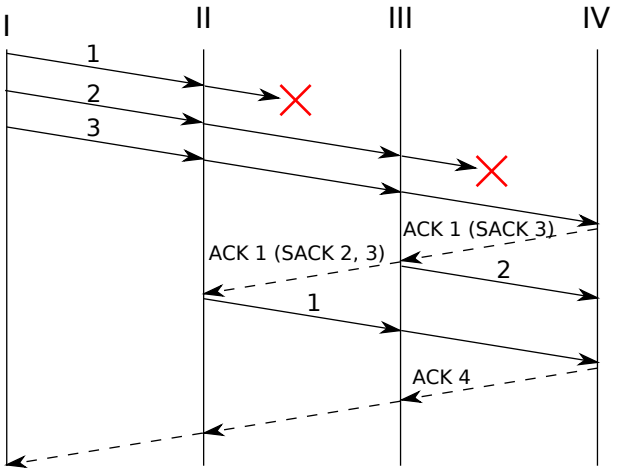


Figure 4: DTC example sending 3 packets over 4 nodes.

DTC example

The DTC example in Figure 4 shows the principle of the local recovery mechanism combined with the selective acknowledgement. The first packet gets lost in the network between node II and III but before that happens it gets cached by node II. Since node II does not receive a link level ACK from node III for the first packet, it gets locked in the cache. When the second packet arrives at node 2 it is not even considered to be cached as the cache is already occupied by a locked packet (the first packet). The second packet then gets cached at node III and is lost in transit from III to IV. With no link level ACK received, node III locks the second packet in its cache. The third packet arrives at the final node IV without being lost (it did not get cached at any node since all intermediate nodes had their caches locked). As node IV receives the third packet it sends the following TCP acknownledgement in direction to node I: `ACK 1 (SACK 3)`. Node IV is still waiting for the first packet but it already received the third packet. Upon reception of this TCP ACK message node III finds out that the ACK number is lower than its cached packet's and its cached packet's sequence number is not present in the SACK block either. In consequence node III restransmits the cached second packet, adds 2 to the TCP ACK and forwards it: `ACK 1 (SACK 2, 3)`. When node II receives the TCP ACK it does the same reasoning as node III did and restransmits its cached first packet. However, the TCP ACK is discarded as the SACK block now forms a contiguous sequence. As node IV receives the first packet it sends a TCP acknowledgement to node I indicating it is now waiting for the fourth packet: `ACK 4`.

4.3 TCP Support for Sensor Nodes

As DTC TCP Support for Sensor nodes tries to reduce the number of retransmissions by caching packets, retransmitting them locally, regenerating TCP acknowledgements and

doi: 10.2313/NET-2011-07-1_11

a mechanism that avoids forwarding packets if the successor node has not received previously sent packets. Overhearing is used to notice if packets are received by the successor node, link level acknowledgements are not needed. As another benefit TCP segments arrive in sequence, hence no reordering or selective acknowledgements are required.
Unlike DTCs caching decision, TSS' is not based on a probability but completely deterministic. A segment which has not yet been forwarded or acknowledged by the successor is always cached. Nodes are listening to their neighbors' transmissions. If a node overhears that a packet has been forwarded by its successor it can remove this packet from the cache. The same holds if a TCP ACK sent from the TCP receiver to a neighboring node and acknowledging the cached packet is overheard. In addition to the buffer to store cached packets another packet buffer for temporal packet storage is needed. This buffer holds packets which have not yet been forwarded to the successor node but need to wait for the confirmation that the previously sent packet was received.
Local retransmissions are triggered by timeouts. The timeout is set to $1.5 \cdot RTT$ and is started after a package was completely sent. Then the node listens if the successor forwards the packet (overhearing). If it does not overhear anything before the timeout is triggered, the packet is locally retransmitted from the cache. To avoid unnecessary retransmissions during network problems the number of local retransmissions is limited to four. If the timeout is triggered by mistake (e.g. due to a bit error in the packet forwarded by the successor) and the TCP segment is retransmitted locally, the successor filters and drops this packet. End-to-end retransmissions are not to be filtered.
It is crucial for TSS that TCP ACKs are not lost since RTT estimation, retransmssions and caching depend on them. Therefore as with DTC TCP ACKs are regenerated locally and additionally an aggressive TCP acknowledgement recovery mechanisms is in place. For this mechanism to work each node measures the time between sending of an ACK and forwarding by the successor node. If after twice this average value the successor did not forward the ACK, it is recovered using the highest acknowledgement number stored in the TCP connection's state information.
TSS avoids packet forwarding in congestion situation and waits until a bit-error packet has been recovered. Hence a node does not forward more packets until it knows that the successor has received and forwarded all previously sent packets. Consequently, if this situation occurs not only does this node stop forwarding packets but all its predecessor nodes too. This is called the backpressure mechanism. The additional packet buffer avoids packets from being lost because the other buffer is full. When the successor of the head of the line node recovers the packet and forwards it, all other nodes will resume their usual modus operandi.

TSS example

Figure 5 shows the main principle of TSS' local retransmissions. While the first packet is received by node V without any problems (resulting in the ACK), a transmission error occurs with the second packet between node III and node IV. Node II, however, overheard node III forwarding the second packet and thusly forwards the third packet to node III. This is where the additional packet buffer comes into play: Node

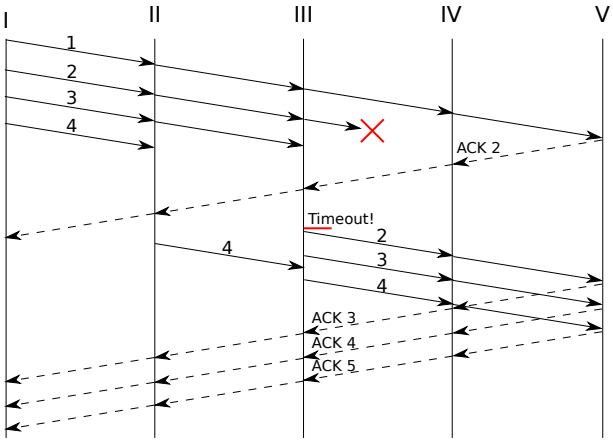


Figure 5: TSS example sending 4 packets over 5 nodes.

III stores the third packet received from node II in the additional buffer and does not forward any more packets. This leads to the so called backpressure mechanism: Node II does not forward any more packets itself, as it did not overhear node III's forwarding of the third packet. Only when the timeout at node III is triggered the packet forwarding continues. Node III retransmits the second packet to node IV, overhears node IV forwarding the same packet right away and continues with forwarding the third packet (which was stored in the additional packet buffer). Node II on the other hand overhears node III's forwarding of the third packet and forwards the fourth packet. From there on all packets are received by node V, the TCP endpoint, which then sends TCP ACKs for all received packets.

4.4 DTC and TSS performance

For high packet error rates (10 – 15 %) normal TCP performs extremly poor. DTC's and TSS' performance is similarly well, while DTC is doing slightly better with high error rates. For a packet error rate of 15 %, 500 TCP segments and 11 hops DTC transmitted about 14,500 TCP segments and ACKs and TSS 13,500 segments and ACKs. Generic TCP on the other hand used over 45,000 TCP transmissions. Assuming that transmission and reception of packets is the main source of energy consumption DTC as well as TSS can reduce that energy consumption of TCP/IP in wireless sensor networks by nearly 70 %.
Despite the differences between TSS and DTC (SACK vs. overhearing, 1 buffer vs. 2 buffers, etc.) both approaches perform similarly well. It should be noted that overhearing should not have a big impact on energy consumption as most packets are forwarded right away and the overhearing period is thusly very short [5].

4.5 Link Quality Estimated TCP

Link Quality Estimated TCP is another variation to the Snoop protocol [2] presented by Ponmagal et al. [16]. The Snoop protocol needs theoretically infinite buffer space at the gateway where all packets are cached. To address this issue Link Quality Estimated TCP employs a selective caching policy thus utilizing the gateway's buffer space more efficiently. TCP segments and ACKs flowing through the gateway to the TCP destination are cached only if the channel's

doi: 10.2313/NET-2011-07-1_11

condition is thought to be bad, i.e. when the probability of a transmission error is high. If this probability exceeds a certain threshold the packet is cached. The probability of a transmission error can be calculated as

$$P_{err} = 1 - (1 - P_1)F \cdot (1 - P_2)F$$

when transmitting a packet in two fragments with F being the fragment size and P_1 and P_2 being the expected bit-error rate for the first fragment and the second fragment respectively. It is left unexplained [16] why the two fragments would have different bit-error rates. Furthermore there is no indication on how to estimate the bit-error rate. This could be done through monitoring previous transmissions and averaging the loss rate. This mechanism, however, would be very slow in reacting to sudden changes in channel conditions. If a duplicate TCP ACK from the TCP destination is received at the gateway the corresponding packet (if cached) is retransmitted locally and all following packets are discarded. If the local retransmission is triggered by a timeout following packets are not discarded.
The rate-controlled wireless link selects its link rate according to some performance objective taking into account the current channel conditions. That means that the highest possible link rate which is below a specified bit-error rate threshold is selected.
Moreover the TCP window which is calculated as the minimum of the congestion window and the receiver window is modified. Along the way of ACKs returning from the receiver to the sender intermediate nodes adjust the receiver window to be as close to the bandwith-delay product as possible. This makes the TCP window adaptable to bottleneck situations in the network.
Ponmagal et al. [16] claim that their approach reduces TCP data transmission times by 5 % compared to normal TCP. Also end-to-end retransmissions are reduced from 18 to 4. This comes at a cost, however, as the gateway needs to store 14 packets whereas with normal TCP no storing is needed. Unfortunately the performance analysis setup (network topology, error rates, file sizes, number of runs,...) is not specified which makes it very hard to draw a consistent conclusion.

4.6 MSS Tuning

The link layer frame size limit in WSNs is relatively small (e.g. 127 bytes in IEEE 802.15.4 [11]). Therefore a TCP/IP packet may have to be fragmented before it can be transmitted in the network. Ayadi et al. are surveying the impact of fragmentation and other TCP parameters on energy consumption [1]. IETF's 6LoWPAN [14] introduces a new protocol layer between the IPv6 layer and the MAC layer. The 6LoWPAN layer compresses the IPv6 header and fragments the IPv6 packet to fit shorter MAC frames. The main source of energy consumption in a WSN is the transmission and reception of data. Thence it directly depends on the total number of bits sent by the whole network. The total number of bits sent depends on several factors: the number of hops between TCP source and TCP destination, the bit-error rate, the TCP maximum segment size (MSS), the maximum number of attempts on the link layer and forward error correction (FEC) redundancy ratio. FEC adds additional information to sent packets which helps recover the original packet if only a few bits are flipped. Therefore a retransmission can be avoided. However, adding FEC data adds an overhead to the packet.
When looking at one-hop transmissions there are three possible outcomes:

1. Failure: The data frame is lost. The sender will initiate a retransmission after a timeout.
2. Partial failure: The data frame is received correctly, but the ACK frame is lost. The sender will (uselessly) initiate a retransmission after a timeout.
3. Success: Both data and ACK frame are received correctly.

The expected total number of bits sent can be calculated analytically by applying probability theory to the different variables [1].
In multi-hop scenarios there are two potential outcomes:

1. End-to-end failure: After the maximum number of attempts on the link layer a node was still unable to forward a frame to the next hop.
2. End-to-end failure: The frame arrives at the TCP destination. Partial failures (i.e. less than the maximum number of retransmissions on the link layer) are possible.

In experiments Ayadi et al. compared the energy footprint of two different TCP maxium segment size choices: MSS = 64 bytes and MSS = 512 bytes. With the former no fragmentation is needed, whereas with the latter MSS the 6LoWPAN layer divides each segment into 8 frames. Using compression an IPv6 header can be shrunk to 2 bytes (from its initial 40 bytes) [13]. Their results showed that for a high bit-error rate it is recommended to use short TCP segments. This is due to the fact that when a segment is lost (due to the high error rate) all of its fragments need to be retransmitted from end to end. If the bit-error rate is rather low it is better to use larger MSS sizes. This leads to fewer acknowledgement messages sent which reduces energy consumption. If packets are fragmented due to a large MSS it is also advisable to increase the maximum number of link layer retransmissions. Thusly the number of costly end-to-end retransmissions can be reduced. On the other side, however, network or node problems are then not detected as quickly as with fewer retransmissions.
When looking at forward error correction they found that an optimal value for the FEC ratio exists [1]: Below this value adding redundancy reduces the loss probability leading to a lower energy consumption. Above this value the redundancy overhead is greater than the expected reduction in data loss. With a high FEC ratio packets with a large MSS perform better than with a small MSS (due to the reduction in transmission errors).
Finally their last finding is that with an increasing number of hops it is better to use small MSSs. This is due to the fact that the overall probability of a transmission error increases with a longer source to destination path.

doi: 10.2313/NET-2011-07-1_11

5. CONCLUSION

In this paper an overview of the current status of TCP/IP in WSNs was given. Optimizing TCP/IP in a WSN is a very active research topic as TCP/IP faces many challenges before being fully adoptable in a sensor network. Various proposals exist for different scenarios. The most promising seem to be Distributed TCP Caching (DTC) [9] and TCP Support for Sensor networks (TSS) [4]. Those two approaches try to make TCP more energy efficient by tweaking the caching process, local retransmissions and introducing additional mechanisms. DTC as well as TSS can reduce the number of sent TCP segments by up to 70 % in respect to normal TCP. Fragmentation is also crucial in lossy networks and should be kept to a minimum when the error rate is high and the end-to-end path is rather long.

Since TCP/IP is the de-facto networking standard it is undoubtful that this protocol suite will be further adapted in such ways that small sensor nodes can cope with it. It is, however, unclear how long it will take to make the final push for TCP/IP adaption in WSNs.

6. REFERENCES

[1] A. Ayadi, P. Maillé, and D. Ros. TCP over low-power and lossy networks: tuning the segment size to minimize energy consumption. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1–5. IEEE.

[2] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP performance over wireless networks. In *Proceedings of the 1st annual international conference on Mobile computing and networking*, pages 2–11. ACM, 1995.

[3] blip authors. blip — Berkeley IP implementation for low-power networks. `http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip`, 2011.

[4] T. Braun, T. Voigt, and A. Dunkels. TCP support for sensor networks. In *Wireless on Demand Network Systems and Services, 2007. WONS'07. Fourth Annual Conference on*, pages 162–169. IEEE.

[5] T. Braun, T. Voigt, and A. Dunkels. Energy-efficient TCP operation in wireless sensor networks. *PIK Journal Special Issue on Sensor Networks*, 28(2):93–100, 2005.

[6] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6), 1995.

[7] A. Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 85–98. ACM, 2003.

[8] A. Dunkels, J. Alonso, T. Voigt, H. Ritter, and J. Schiller. Connecting wireless sensornets with TCP/IP networks. *Wired/Wireless Internet Communications*, pages 583–594, 2004.

[9] A. Dunkels, T. Voigt, J. Alonso, and H. Ritter. Distributed TCP caching for wireless sensor networks. In *Proc. of the Mediterranean Ad Hoc Networking Workshop (MedHoc-Net)*. Citeseer, 2004.

[10] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34. ACM, 2003.

[11] IEEE Computer Society. 802.15.4 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), 2003.

[12] M. Kuorilehto, J. Suhonen, M. Kohvakka, M. Hannikainen, and T. Hamalainen. Experimenting TCP/IP for low-power wireless sensor networks. In *Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on*, pages 1–6. IEEE, 2006.

[13] N. Kushalnagar, G. Montenegro, D. Culler, and J. Hui. Transmission of IPv6 Packets over IEEE 802.15. 4 Networks. 2007.

[14] N. Kushalnagar, G. Montenegro, C. Schumacher, et al. 6lowpan: Overview, assumptions, problem statement and goals. *draft-ietf-6lowpan-problem-01. txt, IETF Internet Draft (Work in progress)*, 2005.

[15] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options: RFC 2018. *Internet Request For Comments*, 1996.

[16] R. Ponmagal and V. Ramachandran. Link quality estimated TCP for wireless sensor networks. *International Journal of Recent Trends in Engineering*, 1(1):495–497, 2009.

[17] J. Postel. Internet protocol. RFC 791, IETF, September 1981.

[18] J. Postel. Transmission control protocol. RFC 793, IETF, September 1981.

„Random Key Distribution“ Verfahren in WSNs – Vergleich verschiedener probabilistischer Ansätze

Sebastian Bendeich
Betreuerin: Corinna Schmitt
Seminar Innovative Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: sebastian.bendeich@mytum.de

KURZFASSUNG

Verschlüsselte paarweise Kommunikation und Authentifizierungsmöglichkeiten spielen eine wichtige Rolle in drahtlosen Sensornetzwerken (WSN), da die Sensoren meist ungeschützt im Einsatzgebiet verteilt sind und ggf. sensible Daten transportieren. Aufgrund von Einschränkungen durch zur Verfügung stehende Ressourcen wie Rechenleistung, Speichergröße und Energie, sowie der Resistenzanforderungen gegenüber Angriffen und Attacken, welche auf WSNs einwirken können, spielen die verwendeten Schlüsselverteilungsstrategien eine wichtige Rolle in der Umsetzung und Ermöglichung von Abhörsicherheit und Authentifizierung. Die Authentifizierung wird insbesondere benötigt, um Manipulationsversuche erkennen und Gegenmaßnahmen, wie Ausschluss des entsprechenden Sensors aus dem Netzwerk, ergreifen zu können. Chan et al [2] und Di Pietro et al [3] haben sich mit probabilistischen Verfahren beschäftigt, die hier erläutert und verglichen werden, sowie auf Ihre Resistenz bzw. Robustheit gegenüber Angriffen beleuchtet werden.

Schlüsselworte

WSN, Random Key Schemes, Probalisitc Key Schemes, Verschlüsselung, Verschlüsselungssicherheit, Authentifizierung, DoS-Attacken, Key Managment Protokolle

1. EINLEITUNG

Der Einsatz von drahtlosen Sensornetzwerken (WSN) wird immer häufiger und umfasst immer mehr Einsatzgebiete. Da es je nach Einsatzgebiet, zum Beispiel bei der Gebäude- oder Geländeüberwachung, aber nicht zuletzt auch im militärischen Bereich, bedeutsam ist, ob die übertragenen Daten abhörsicher und verlässlich (z.B. gegenüber Manipulation) sind, bekommt auch der Aspekt der Verschlüsselung und Authentifizierung in WSNs immer größere Bedeutung. Jedoch lassen sich nicht alle, der in den üblichen EDV-Einsatzgebieten bereits verbreiteten Verfahren, wie „Public Key“-Verschlüsselung, in WSNs einsetzen [1]. Dies ist der Tatsache geschuldet, dass die einzelnen Sensoren nur begrenzte Energie, Rechenleistung und insbesondere begrenzten Speicherplatz zur Verfügung haben, die nicht zuletzt das Speichern der teils langen Schlüssel für „Public Key“-Verfahren einschränken bis ganz unmöglich machen. Denn selbst bei der Verschlüsselung von Webseiten (SSL) wird der „Public Key“ nur zum Finden und Austauschen eines gemeinsamen, symmetrischen Schlüssel genutzt und dies, obwohl die Rechenleistung von Computern und die Bandbreite der Internetanbindungen im Vergleich zu denen der Sensoren in WSNs relativ hoch ist.

In diesem Artikel werden die von Chan et al im Artikel „Random Key Predistribution Schemes for Sensor Networks“ [2] beschriebenen probabilistischen Methoden zur Schlüsselverteilung und daraus folgenden Eigenschaften und Protokolle für die Kommunikation zwischen Sensoren mit denen von Di Petro et al im Artikel „Random Key-Assignment for Secure Wireless Sensor Networks“ [3] verglichen. Insbesondere wird auf die Tauglichkeit der jeweiligen Methoden zur Verschlüsselung und Authentifizierung von Sensoren innerhalb eines WSN wertgelegt und die mit den jeweiligen Verfahren bzw. Protokollen verbundenen Schwachstellen herausgearbeitet. Es ist also an der Verschlüsselung, dem Mithörer nicht die Chance zu geben, den mitgeschnitten Datenverkehr dechiffrieren zu können. Aufgrund der Zugänglichkeit zu den Sensoren, sollte man sich bewusst sein, dass Sensoren eingesammelt oder direkt vor Ort ausgelesen oder gar manipuliert werden können und somit ggf. im Speicher des Sensors liegende Schlüssel dem Angreifer bekannt werden können. Nicht zuletzt spielt auch die mögliche Absicht eines Angreifers, das Sensornetzwerk zu stören bzw. durch Erzwingen von vermehrter Funk- oder CPU-Aktivität, die Energiespeicher der Sensoren vorzeitig zum Erliegen zu bringen eine wichtige Rolle, sodass zum Einsatz kommende Verfahren versuchen sollten, den Einfluss auf die Kommunikationseigenschaften und die lokalen Ressourcen durch derartige Attacken zu minimieren. Detailliertere Beschreibungen der genannten und weiterer Angriffsmotive und –verfahren können u.a. dem Artikel „Security for wireless sensor networks“ von Avancha et al [4] und den Ausführungen in „A survey of security issues in mobile ad hoc and sensor networks“ von Djenouri et al [5] entnommen werden.

Zunächst nun ein paar einführende Definitionen, bevor mit der Vorstellung der Verfahren begonnen wird.

2. DEFINITION SYMMETRISCHE VERSCHLÜSSELUNG

Unter symmetrischer Verschlüsselung versteht man ein Kryptografieverfahren, bei dem die Nachricht mit demselben Schlüssel ver- und auch wieder entschlüsselt wird. Im Gegensatz zu asymmetrischen Verschlüsselung („Public Key“) bei der verschiedene Schlüssel zum Ver- und Entschlüsseln existieren,

doi: 10.2313/NET-2011-07-1_12

sodass der Verschlüsselungsschlüssel frei bekannt gegeben werden kann. Bei der symmetrischen Verschlüsselung ist es also von Vorteil einen Schlüssel nur für eine paarweise Kommunikation einzusetzen, falls sichergestellt werden soll, dass nur Sender und Empfänger in der Lage sind, die Nachricht entschlüsseln zu können. Dies führt auch schon zum Kernproblem von symmetrischen Verschlüsselungsverfahren: auf welche Weise sollen die Schlüssel den Sensoren zugewiesen werden. Dynamische Verfahren, die generisch einen Schlüssel generieren und diesen über einen asymmetrisch verschlüsselten Kanal zum Kommunikationspartner übertragen, entfallen, aufgrund der schon eingangs erwähnten Nichteignung von WSNs für „Public Key"-Verschlüsselungsverfahren.

3. DEFINITION PROBABILISTISCHE VERTEILUNG

Eine mögliche Alternative ist das Vorverteilen eines generierten Schlüsselpools auf die auszubringenden Sensoren, d.h. die Schlüssel werden zusammen mit der Software aufgespielt. Bei probabilistischen Verfahren, wird jedem Sensor zufällig eine Teilmenge der verfügbaren Schlüssel im Pool zugewiesen. Zwei Sensoren können nun direkt miteinander kommunizieren, falls sie mindestens einen Schlüssel gemeinsam haben. Da dies neben einigen Parametern wie Pool- und Teilmengengröße von zufälligen Ereignissen abhängt, spricht man von probabilistischen Verteilungsansätzen. Eschenauer und Gligor veröffentlichten eines der ersten probabilistischen Verfahren [6].

Generell unterscheidet man bei probabilistischen Verfahren drei Phasen (nach dem Aufteilungsschema von Di Pietro et al [3]): die erste Phase (Predeployment) bezeichnet die Zeit, in der (z.B. im Labor) der Speicher des Sensor vor Ausbringung in das eigentliche Einsatzgebiet mit Software oder anderweitigen Daten (z.B. Schlüsseln) bestückt werden kann. In der zweiten Phase (Key Discovery) erkunden die Sensoren, nachdem Sie im Einsatzgebiet ausgebracht wurden, ihre Umgebung. Dabei ermitteln sie mit welchen der Sensoren sie direkt kommunizieren können (Funkreichweite) und welche Schlüssel sie mit ihnen teilen. In der dritten Phase (Kanalaufbau) einigen sich zwei in der Phase zuvor gefundenen Sensoren auf einen gemeinsamen Schlüssel für eine sichere, paarweise Kommunikation.

4. VERFAHREN VON DI PIETRO ET AL

Di Pietro et al entwickelten zwei Protokolle für die Schlüsselverteilung [3], die aufeinander aufbauen. Das erste Verfahren „direktes Protokoll" genannt und die Erweiterung „kooperatives Protokoll", mit deren das Sicherheitslevel der Kommunikation innerhalb des WSN dynamisch verändert werden kann.

4.1 Das „direkte Protokoll"

In der Predeployment-Phase werden zunächst P zufällige Schlüssel generiert.

$$P = \{v_1, \ldots, v_p\}$$

Dabei ist jedem Schlüssel ein Index (z.B. fortlaufende Nummer) zugeordnet. Jeder Sensor a soll am Ende dieser Phase eine k Element große Teilmenge des Schlüsselpools P in sich tragen. Die Zuweisung der k Schlüssel zu einem Sensor erfolgt nun nicht vollständig zufällig, wie im Verfahren von Eschenauer und Gligor [6], sondern durch ein sog. Pseudo-zufälliges Verfahren, bei welchem ein, allen Sensoren bekannter Ausgangswert (Seed) genutzt wird, um den Zuteilungsgenerator zu initialisieren. Der Zuteilungsgenerator wählt nun anhand des Ausgangswerts und der Sensor ID k Indizes aus. Die Schlüssel mit den entsprechenden Indizes werden anschließend dem Sensor zugeteilt, dies wiederholt man mit allen n Sensoren. Die Wahl eines Pseudo-Zufallsgenerators stellt damit den einzigen wirklichen Unterschied zum von Eschenauer und Gligor [6] vorgestellten Verfahren dar. Jedoch macht sich dieser Unterschied in den Eigenschaften stark bemerkbar, wie im Folgenden ersichtlich wird.

Aufgrund dass jedem Sensor der Ausgangswert und der Zuteilungsgeneratoralgorithmus bekannt sind, kann jeder Sensor a berechnen, welche Schlüssel (bzw. vielmehr die Indizes der Schlüssel) ein Sensor mit ID b haben muss.

Der Vorteil der pseudozufälligen Zuteilung offenbart sich nun in der Key Discovery Phase, da nun zum Ermitteln der gemeinsamen Schlüssel nicht die Schlüssel selbst über einen Broadcast gesendet werden müssen (und damit Gefahr gelaufen wird, dass ein möglicher Mithörer, die Schlüssel abgreifen kann), sondern es ausreicht, die eigene Sensor ID zu senden, welche keine sicherheitskritische Information ist und somit auch unverschlüsselt übertragen werden kann. Dies wirkt sich unter anderem auch positiv auf den Overhead und damit auf den Energieverbrauch aus. Empfängt Sensor a nun den Broadcast von Sensor b, so weiß a, dass er sich in Sendereichweite von b befindet und damit, dass b höchstwahrscheinlich auch in der Sendereichweite von ihm liegt.

```
getSitzungsschlüssel(p: sensorID)
Input: ID des Kommunikationspartners (beim Empfänger
Sender-ID und vice versa)

k_a,b = 0;
partner_indizes = calculateIndizes(p);
found=false;
for each partner_index ∈ partner_indizes {
        if partner_index ∈ meine_indizes {
                k_a,b = k_a,b ⊕ meine_keys[partner_index];
                found = true;
        }
}
if( !found) {
        error();
} else {
        return k_a,b;
}
```

Tabelle 1 : Pseudocode der Schlüsselberechnung beim „direkten Protokoll"

Um eine verschlüsselte Nachricht von Sensor a an Sensor b zu senden (Kanalaufbauphase), wird der Sitzungsschlüssel $k_{a,b}$ wie folgt berechnet: aus den beiden vorherigen Schritten kennt a die Schlüsselindizes aller Schlüssel die b besitzt. Nach einem Durchsuchen der eigenen Schlüsselmenge, kennt a nun alle Schlüsselindizes (und Schlüssel!), die er mit b gemeinsam hat. An dieser Stelle könnte sich auch herausstellen, dass a und b keine Schlüssel teilen und somit eine direkte Kommunikation nicht möglich ist. Falls die gefundene Schnittmenge jedoch nicht leer war, so werden die gemeinsamen Schlüssel nun mittels der XOR-

doi: 10.2313/NET-2011-07-1_12

Operationen verbunden, das Ergebnis ist der Sitzungsschlüssel $k_{a,b}$.

$$k_{a,b} = \bigoplus_{Vi \in Pa \cap Pb} V_i$$

Dieser Vorgang ist in Pseudocode-Implementierung auch nochmals in Tabelle 1 veranschaulicht. Wenn Sensor *b* nun eine Nachricht von a empfängt, berechnet *b* nun auf die gleiche Art und Weise, welche Schlüssel er mit *a* teilt und daraus $k_{b,a}$, was offensichtlich identisch ist mit $k_{a,b}$. Um aber die Verlässlichkeit der Berechnung zu erhöhen, bzw. diese prüfen zu können. Sendet Sensor *a* als erste Nachricht zu *b* eine *key_estimate* Nachricht, welche mittels $k_{a,b}$ verschlüsselt ist. Falls *b* in der Lage ist, diese zu entschlüsseln, also $k_{a,b}$ durch Sensor *a* korrekt berechnet wurde, antwortet *b* mit einer *key_estimate_confirm* Nachricht.

4.2 Das „kooperative Protokoll“

Das oben beschriebene Verfahren hat den Nachteil, dass es nicht resistent gegenüber einer hohen Anzahl korrupter Sensoren ist, da sobald der Angreifer es geschafft hat, die Schnittmenge der Schlüssel von Sensor *a* und *b* in Erfahrung zu bringen (zum Beispiel durch Kapern und Auslesen von Sensoren), er nun in der Lage ist, die gesamte Kommunikation zwischen *a* und *b* abzuhören und zu manipulieren.

Im kooperativen Verfahren wird die Kanalaufbauphase modifiziert. Falls nun Sensor *a* mit Sensor *b* kommunizieren möchte, so wählt *a* eine Menge *C* der Kardinalität *m* von Sensoren

$$C = \{c_1, \ldots, c_m\} \text{ mit } m > 0 \text{ und } a,b \notin C$$

als kooperierende Partner aus, wobei *a* und *b* nicht in *C* enthalten sein dürfen. An jeden Sensor c_i wird eine dementsprechende Anfrage gesendet, wobei diese Kooperationsanfrage mittels $k_{a,ci}$ (gemäß dem direkten Verfahren) verschlüsselt ist und als Nutzdaten die ID von Sensor *b* enthält. Sensor c_i berechnet daraufhin zunächst den gemeinsamen Schlüssel $k_{ci,b}$ (gemäß dem direkten Verfahren) und sendet schließlich einen Hashwert (sei HMAC die gewählte Hashfunktion) aus der ID von *a* und dem Schlüssel $k_{ci,b}$ an den anfragenden Sensor *a* mittels $k_{a,c}$ zurück.

$$HMAC(\mathrm{ID}(a), k_{ci,b})$$

Sobald Sensor *a* alle angefragten Hashwerte vorliegen, bildet er den letztendlichen Sitzungsschlüssel $k^C_{a,b}$ durch Anwenden der XOR-Operation auf Schlüssel $k_{a,b}$ und die empfangenen Hashwerte (auch jeweils durch XOR verbunden).

$$k^C_{a,b} = k_{a,b} \oplus (\bigoplus_{c \in C} \mathrm{HMAC}(\mathrm{ID}(a), k_{ci,b}))$$

Sensor *a* sendet nun zu Beginn der Sitzung an *b* verschlüsselt mittels $k_{a,b}$ die Liste der gewählten Sensoren *C* zusammen mit dem Hashwert des Schlüssels $k^C_{a,b}$. Das Verfahren auf Seite von Sensor *a* ist nochmals in der Pseudocode-Implementierung in Tabelle 2 dargestellt.

Daraufhin kann Sensor *b* aus der übertragenen Sensorliste ohne Versenden weiterer Nachrichten, den Schlüssel $k^C_{a,b}$ berechnen und ihn mittels des mit gesendeten Hashwerts validieren, was in Tabelle 3 mittels einer Pseudocode-Implementierung dargestellt ist.

Da Sensor *b* hierzu mit keinem der Sensoren *C* kommunizieren muss, kann *a* auch Sensoren wählen, die nicht in Sende- und Empfangsreichweite von *b* liegen.

Es ist anzumerken, dass das kooperative Protokoll dem direkten entspricht, falls *m=0* gewählt wird. *m* kann insbesondere der jeweiligen Sicherheitslage entsprechend gewählt werden, sodass auch mit zunehmender Anzahl von korrumpierten Sensoren eine sichere Verschlüsselung gewährleistet werden kann.

```
prepareSend(b: sensorID, c[]: sensorID)
Input: ID des Zielsensors, Liste der kooperierenden Sensoren

k^C_a,b = getSitzungsschlüssel(b);
for each koop_id ∈ c {
        //sendCoopRequest Methode liefere die
        //empfangene Antwort zurück
        answer = sendCoopRequest(koop_id);
        k^C_a,b = k^C_a,b ⊕ answer;
}

//sendet Liste c mittels dem direkten Protokoll an Sensor b
sendViaDirectProtocol(b, getSitzungsschlüssel(b), c);

return k^C_a,b;
```

Tabelle 2 : Pseudocode der Sitzungsschlüsselberechnung auf Verbindungsinitiatorseite beim „kooperativen Protokoll“

```
prepareReceive(a: sensorID, c[]: sensorID)
Input: ID des Sendersensors, Empfange Liste der
kooperierenden Sensoren

k^C_a,b = getSitzungsschlüssel(a);
for each koop_id ∈ c {
        //Folgende zwei Kommandos wurden auch bei
        //jedem kooperierenden Sensor ausgeführt, um
        //die Antwort (answer) zu berechnen
        koop_key = getSitzungsschlüssel(koop_id);
        hashedValue = HMAC(a, koop_key);

        k^C_a,b = k^C_a,b ⊕ hashedValue;
}

return k^C_a,b;
```

Tabelle 3 : Pseudocode der Sitzungsschlüsselberechnung auf Verbindungszielseite beim „kooperativen Protokoll”

Abschließend kann man feststellen, dass sowohl im direkten als auch im kooperativen Protokoll eine sichere Authentifizierung von Sensor *b* durch *a* und umgekehrt gewährleistet werden kann, solange auch eine sichere paarweise Kommunikation zwischen beiden besteht. Da mithilfe des Pseudozufallsgenerators jeder Sensor *a* ermitteln kann, welche Schlüssel ein anderer Sensor mit ID *b* haben muss. Ein Angreifer kann also eine ID *b* nur fälschen und sich mit diesem falschen Sensor erfolgreich bei *a* authentifizieren, wenn dem Angreifer alle Schlüssel bekannt waren, die *a* und der echte Sensor *b* gemeinsam hatten.

doi: 10.2313/NET-2011-07-1_12

5. VERFAHREN VON CHAN ET AL

Chan et al [2] schlagen in ihrem Werk „Random Key Predistribution Schemes for Sensor Networks‘“ insgesamt drei Verfahren zur Predeployment-Schlüsselverteilung vor. Wobei die im Folgenden zuerst erläuterte „q-Composite Schlüsselverteilungsstrategie“ eine Möglichkeit für den Grundstock der „Multipfad-Schlüsselverstärkung“ bildet, die anschließend kurz erläutert wird. Schlussendlich wird ein drittes Verfahren, die „zufällige, paarweise Schlüsselstrategie“, vorgestellt, welche einen komplett verschiedenen Ansatz zu den vorherigen beiden aufweist, dafür jedoch einen Authentifizierungsmechanismus enthält.

5.1 q-Composite Schlüsselverteilungsstrategie

Das q-Composite Verfahren variiert das zugrundeliegende Verfahren von Eschenauer und Gligor [6] im Grunde nur in der Tatsache, dass für die Kommunikation zwischen zwei Sensoren, verlangt wird, dass diese mindestens *q* Schlüssel gemeinsam haben, d.h. das zugrundegelegte Verfahren erhält man durch Setzen von *q=1*.

Für das bestehende Sicherheitsproblem während der Key Discovery-Phase, dass ein böswilliger Mithörer, die gebroadcasteten Schlüsselindizes nutzen kann, um gezielt Sensoren einzusammeln und auszuwerten, sodass er den vollständigen Schlüsselpool in seinen Besitz bringen kann, wird ein Verfahren vorgeschlagen, dass von Merkle [7] erarbeitet wurde. Hierbei wird nicht der Schlüsselindex im Klartext gebroadcastet, sondern stattdessen wird für jeden Schlüssel *i*, den der Sensor *a* besitzt, ein „Puzzle“ generiert und gesendet. Ein Sensor *b* der in Empfangsreichweite ist und das Puzzle entschlüsseln kann (also durch Probieren seiner eigenen Schlüssel, die empfangene Nachricht dechiffrieren konnte und somit ebenfalls diesen Schlüssel besitzt), sendet daraufhin die passende Antwort zu dem Puzzle zurück. Sensor *a* weiß anschließend über alle Sensoren *B*, welche die richtige Antwort zurückgesendet haben, dass sie den verwendeten Schlüssel *i* kennen.

Nach abgeschlossenem „puzzeln“, kennt jeder Sensor *a* seine Nachbarn *B* mit denen er mindestens *q* Schlüssel gemeinsam hat und mit denen somit eine direkte Kommunikation möglich bzw. erlaubt ist.

Falls Sensor *a* nun eine Nachricht an einen der Sensoren *B* senden möchte, so bildet er aus den (gemeinsamen mit *b*) Schlüsseln einen Hashwert. Dieser Hashwert ist nun der Kanalschlüssel von Sensor *a* und Sensor *b* für alle zu sendenden Nachrichten, d.h. er bleibt über die gesamte Betriebszeit des WSN gleich.

Welche Hashfunktion angewendet wird, wurde von Chan et al nicht genauer spezifiziert und bleibt dem jeweiligen Anwender überlassen.

5.2 Multipfad-Schlüsselverstärkung

In diesem Verfahren nehmen wir an, dass bereits eine paarweise Kommunikation über das Verfahren von Eschenauer und Gligor [6] oder mittels der q-Composite Schlüsselverteilungsstrategie ermöglicht wurde. Allerdings möchte man nun in der Kanalaufbauphase, die eigentlichen Nachrichten besser verschlüsseln, um zum Beispiel sicherzustellen, dass falls ein Angreifer in Besitz der gemeinsamen Schlüssel von Sensor *a* und *b* gelangt ist, damit nicht automatisch die gesamte

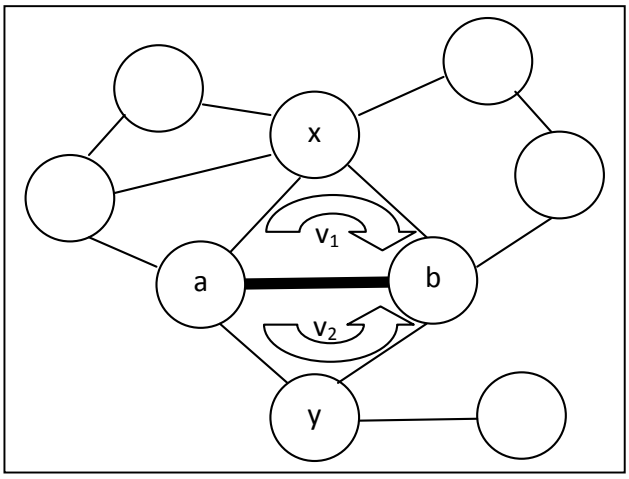


Abbildung 1 : Multipfad-Schlüsselverstärkung Beispiel mit *j=2* und *h=1*

Kommunikation zwischen diesen beiden Sensoren dechiffrieren kann.

Hierzu wird für jeden Kanalaufbau ein eigener Sitzungsschlüssel zwischen Sensor *a* und *b* verabredet. Offensichtlich kann dieser Sitzungsschlüssel nicht einfach direkt von *a* nach *b* gesendet werden, da sonst diese Information im oben genannten Fall für den Angreifer ebenfalls ersichtlich wäre und der Sitzungsschlüssel somit seine Bedeutung und Funktion verloren hätte. Wir nehmen an, es wären genug Routinginformationen bekannt, sodass Sensor *a* alle (Sensor-)disjunkten Pfade (welche in der Key Discovery-Phase entdeckt wurden) von sich zu Sensor *b* kennt, die aus *h* oder weniger Sprüngen (Hops) bestehen. Nehmen wir nun weiter an, es gäbe *j* solcher disjunkten Pfade, so werden *j* Zufallszahlen von Sensor *a* generiert, wobei jede Zufallszahl die gleiche Länge wie die Verschlüsselungsschlüssel aufweist. Jede dieser Zufallszahlen sendet *a* nun über einen anderen Pfad zu Sensor *b*. Wenn *b* alle Zahlen empfangen hat, können Sensor *a* und *b* den Sitzungsschlüssel durch XOR-Anwendung berechnen.

$$k' = k \oplus v_1 \oplus v_2 \oplus \ldots \oplus v_j$$

Für das Finden der Pfade, schlagen Chan et al für den Spezialfall *h=2* die Vorgehensweise vor, dass nach erfolgter Key Discovery-Phase die Sensoren *a* und *b* eine Liste ihrer jeweiligen Nachbarn austauschen. Die Schnittmenge der beiden Listen, sind dann die zu nutzenden Intermediäre. Die Anzahl der Pfade entspricht dann der Kardinalität dieser Schnittmenge. Da bei nur zwei Sprüngen nur genau ein intermediärer Sensor auf dem Pfad liegt, ist es offensichtlich, dass alle so gefundenen Pfade automatisch disjunkt sind. Ein Beispielfall für die Anwendung der Multipfad-Schlüsselverstärkung ist in Abbildung 1 dargestellt, wobei die dick dargestellte Verbindung zwischen Sensor *a* und *b* durch die Teilschlüssel gesichert werden soll. Die zwei Teilschlüssel v_1 und v_2 werden hierbei über die disjunkten Pfade via Sensor *x* bzw. *y* übertragen. Wobei hier angenommen wurde, dass kein Teilschlüssel direkt von *a* nach *b* übertragen wird, da dies wider dem Sinn der Multipfad-Schlüsselverstärkung wäre, worauf im Kapitel „Vergleich“ noch eingegangen wird.

doi: 10.2313/NET-2011-07-1_12

5.3 Zufällige, paarweise Schlüsselstrategie

Die bisherigen Verfahren von Chan et al, besaßen noch nicht die Möglichkeit der Authentifizierung der jeweiligen in Kommunikation stehenden Sensoren. Da im Gegensatz zu dem Verfahren von Di Pietro et al [3] die Schlüssel durch einen, für die Sensoren unbekannten, Zufallsalgorithmus verteilt wurden, sodass es sein kann, dass die Schnittmenge der gemeinsamen Schlüssel von Sensor *a* und *b*, auch in Sensor *c* vorhanden sind und somit keine Möglichkeit für Sensor *a* besteht zu Prüfen, ob er nun wirklich mit Sensor *b* oder Sensor *c* kommuniziert.

Mit der zufälligen, paarweisen Schlüsselstrategie stellen Chan et al nun eine Variante vor, die auch eine Authentifizierung zulässt. Sie basiert auf der naiven Variante, zur Erinnerung: bei einem *n* Sensoren großen Netzwerk hält jeder Sensor *n-1* Schlüssel, wobei jeder Schlüssel in genau zwei Sensoren vorkommt. Das Verfahren führt nun für die einzelnen Sensoren IDs ein, mit jedem Schlüssel der lokal im Sensor gespeichert wird, wird auch die ID des anderen Sensors gespeichert, der denselben Schlüssel besitzt. Die Authentifizierung nutzt also die Tatsache, dass falls eine sichere Kommunikationsverbindung unter Nutzung des Schlüssels *k* zustande kam, dass es sich bei dem Gesprächspartner um den Sensor mit der entsprechenden ID handeln sollte. Aus der in Chan et als Werk zitierten Erdös und Rényis Formel geht hervor, dass eine vollständige Vernetzung des Sensornetzwerks der Größe *n* gegeben ist, in der die kleinste Wahrscheinlichkeit *p* für die Möglichkeit, dass zwei beliebige Sensoren kommunizieren können, sodass der gesamte Sensorknotengraph zusammenhängend ist mit Wahrscheinlichkeit *c*, jeder Sensor nur *np*, statt *n-1* paarweise Schlüssel speichern muss, was dem begrenzten Speicherplatz Rechnung trägt.

In der Predeployment-Phase wird nun ein *n* gewählt, welches der maximalen Größe (bei gleichbleibender Verbindungswahrscheinlichkeit) des Sensornetzwerks entspricht. Es müssen hierbei am Anfang nicht unbedingt auch genauso viele Sensoren ausgebracht werden, sodass Reserven für später hinzuzufügende Sensoren bleibt. Für jeden Sensor *a* wird nun eine ID generiert und mit *np* anderen IDs (Partnern) in Verbindung gebracht. Für jedes so entstandenes Paar aus Sensor a und einem Partnersensor wird nun ein Schlüssel generiert und zusammen mit der ID des Partners in Sensor *a* gespeichert, bzw. zusammen mit der ID von Sensor *a* im Partnersensor gespeichert.

In der Key Discovery-Phase broadcastet jeder Sensor *a* nun seine ID zu seinen unmittelbaren Nachbarn. Die Nachbarn prüfen nun, ob sie die empfangene ID in ihrem Speicher finden, wenn dies der Fall ist, wird der damit im Speicher aufgefundene Schlüssel genutzt, um einen verschlüsselten Handshake mit dem Sendersensor *a* auszuführen. Falls dieser Handshake erfolgreich war, ist sichergestellt, dass *a* auch wirklich der Sensor ist, für den er sich beim Broadcast ausgegeben hat, da er sonst nicht in Besitz des richtigen Schlüssels gewesen wäre, um den Handshake zu dechiffrieren bzw. die Antwort auf den Handshake zu chiffrieren.

6. VERGLEICH

Nachdem nun die von Di Pietro et al und Chan et al entwickelten Verfahren vorgestellt wurden, soll an dieser Stelle ein Vergleich der jeweiligen Verfahren erfolgen mit dem Schwerpunkt auf:

- Overhead im Datenverkehr
- Resistenz gegenüber DoS-Attacken
- Abhörsicherheit

6.1 Overhead im Datenverkehr

Aufgrund der begrenzten Ressourcen in typischen WSNs und des Energiehungers der Funkschnittstelle, sollte der Datenoverhead auf jeden Fall gering gehalten werden. Er ist aufgrund dieser Konstellation auch ein wichtiges Merkmal, der über die Lebensdauer eines einzelnen Sensorknotens entscheidet. Da Datenoverhead auf jeder Ebene der Kommunikation entsteht (OSI-Modell), in diesem Artikel aber nur verschiedene Verschlüsselungsstrategien behandelt wurden, wird hier auch nur der bis zum erfolgten Kanalaufbau entstehende Overhead berücksichtigt. Es ist jedoch anzumerken, dass natürlich die gewählte Schlüssellänge und die Verarbeitungsweise der Daten, also das Verfahren zur De-/Chiffrierung der Nutzdaten mithilfe des Schlüssels auch zum Gesamtoverhead beitragen.

6.1.1 Direktes Protokoll nach Di Pietro et al

Dieses Verfahren weißt den geringsten Overhead auf, da für die Key Discovery-Phase nur einmalig ein Broadcast gesendet werden muss, welches als Nutzdaten ausschließlich die Sensor-ID beinhaltet. Für den Kanalaufbau besteht der Overhead nur in der *key_estimate* und *key_estimate_confirm* Nachricht, welche aber prinzipiell auch weggelassen werden könnte, da sie nur zur Prüfung eingesetzt werden.

6.1.2 Kooperative Protokoll nach Di Pietro et al

Offensichtlich, steigt der Overhead in diesem Verfahren gegenüber dem „direkten Protokoll“ zugunsten der Sicherheit an. Je höher das Sicherheitslevel gewählt wird, also je mehr kooperierende Sensoren mit einbezogen werden, desto größer der Overhead, da an jeden kooperierenden Sensor eine Nachricht mit Aufforderung zur Kooperation und der ID des Zielsensors *b* gesendet werden muss, sowie die ausgewählten Sensoren das Ergebnis an *a* zurücksenden müssen. Folglich zwei Nachrichten pro kooperierenden Sensor plus der Nachricht mit der Liste der kooperierenden Sensoren an Zielsensor *b* zusätzlichen Overhead gegenüber dem „direkten Protokoll“. Hinzukommt, dass für verschiedene Sitzungen verschiedene, kooperierende Sensoren gewählt werden können, sodass das Prozedere dementsprechend pro Sitzung wiederholt werden muss.

6.1.3 q-Composite Strategie nach Chan et at

In diesem Verfahren muss jeder Sensor eine Liste mit seinen Schlüsselindizes senden (also *n* Nachrichten insgesamt), was identisch zur Anzahl bei der Key Discoveryphase im „direkten Protokoll“ nach Di Pietro et al wäre. Um aber die Sicherheit zu erhöhen (s.o.), ist es empfehlenswerter, für jeden Schlüssel den ein Sensor *a* besitzt ein Puzzle zu generieren und zu senden. Hierbei würde jeder Sensor also *K* (= Kardinalität der Schlüsselteilmenge pro Sensor) Nachrichten senden. Da nun aber auf das Puzzle geantwortet werden muss, kommen noch maximal *K* Antworten hinzu. Insgesamt also $(n * k^2)$ Nachrichten, die gesendet werden müssen.

6.1.4 Multipfad-Schlüsselverstärkung nach Chan et al

Dieses Verfahren baut auf einem erfolgten Kanalaufbau auf (deswegen der Name „Verstärkung“), es sind also die schon erzeugten Overheadkosten aufzuaddieren, welche vom gewählten Verfahren abhängig sind. Die zusätzlichen Kosten variieren nach

doi: 10.2313/NET-2011-07-1_12

der Anzahl der Sprünge bzw. der verwendeten Methode zum Finden der Pfade. Legen wir an dieser Stelle das von Chan et al vorgeschlagene Vorgehen zugrunde, so würden alle Sensoren eine Liste mit ihren Nachbarn verschlüsselt an jeden Nachbar senden. Dies wären also *n* Nachrichten. Im Maximalfall hat jeder Sensor *n-1* Nachbarn gefunden. Folglich gäbe es *j=n-2* disjunkte Pfade von Sensor *a* nach Sensor *b*. Damit sendet Sensor *a* folglich *j* Nachrichten und jeder der *j* Sensoren sendet wieder eine Nachricht an Sensor *b*. Insgesamt wären es also insgesamt $(n + (n-2)^2)$ zusätzliche Nachrichten. Damit ist der Overhead des Verfahrens für *h=2* vergleichbar mit dem des „kooperativen Protokolls" von Di Pietro et al. Wobei allerdings auch der eigentlich passive Empfangsknoten zu Beginn eine Liste der Nachbarn senden musste, wohin gegen beim „kooperativen" Protokoll der Zielknoten keine weitere Nachricht senden oder empfangen muss.

6.1.5 Zufällige, paarweise Schlüsselstrategie nach Chan et al

Bei diesem Verfahren haben wir zunächst *m* Nachrichten mit der ID des Sendesensors die gebroadcastet werden. Anschließend muss jeder Sensor für die maximal *m-1* Nachbarn noch einen Handshake ausführen, bestehend aus einer Nachricht plus Antwort. Insgesamt benötigt dieses Verfahren also *(m+ 2*(m-1))* Nachrichten. Dieses Verfahren ähnelt im Punkt des Datenoverheads folglich am stärksten dem des „direkten Protokolls" von Di Pietro et al.

6.2 Resistenz gegenüber DoS-Attacken

Ein Angreifer kann als Ziel nicht nur das Ausspionieren oder Manipulieren von Daten haben, sondern auch daran interessiert sein, dass Sensornetzwerk lahm zu legen, wie im Artikel von Sen et al [1] ausführlich erläutert wurde, z.B. indem er ein vorzeitiges Erschöpfen der Energieressourcen vorursacht. Eine DoS-Attacke kann zum einen die Luftschnittstelle betreffen (Jamming) oder Angriffsmöglichkeiten auf der Anwendungsschicht adressieren. Da in diesem Paper nur ein Schwerpunkt auf die Verschlüsselungsstrategie bzw. den Kanalaufbau gelegt wird, werden nur DoS-Angriffsmöglichkeiten berücksichtigt, die genau daran ansetzen. Wie wir sehen werden, spielt der Netzwerkoverhead, den die verschiedenen Strategien mit sich bringen, eine nicht unbedeutende Rolle.

6.2.1 Direktes Protokoll nach Di Pietro et al

Während der Key Discovery-Phase sendet jeder Sensor nur einmalig seine ID, d.h. die Häufigkeit mit der eine Key Discovery-Phase von außen eingeleitet werden kann, gibt die obere Schranke für erzwungenen, unnötigen Netzwerkoverhead an. Während der Kanalaufbauphase kann nur durch Senden einer *key_estimate* Nachricht Aktivität erzwungen werden, wobei eine Antwort nur erfolgt, falls es sich um den konformen Schlüssel gehandelt hat. Folglich kann der attackierte Sensor in dieser Phase nur dann zum Senden gezwungen werden, falls eine verschlüsselte Verbindung mit diesem Sensor vorliegt, dann bietet zugleich die Anwendungsschicht meist das effizientere Ziel.

6.2.2 Kooperative Protokoll nach Di Pietro et al

Aufgrund, dass das kooperative auf dem direkten Protokoll aufbaut, erbt dieses Verfahren die soeben vorgestellten Schwachstellen. Nehmen wir hier an, dass in der Menge der gewählten, kooperierenden Sensoren, ein oder mehr Sensoren korrumpiert wären. Der korrumpierte Sensor *c* hat nun drei Möglichkeit auf die Kooperationsanfrage zu reagieren: ignorieren (1), mit dem richtigen Schlüssel $k_{c,b}$ (2) oder dem falschen Schlüssel $k'_{c,b}$ (3) zu antworten, wie bereits Di Pietro et al in Ihrem Artikel [3] ausführten.

Fall (1) würde der Situation entsprechen, falls ein zur Kooperation hinzugezogener Sensor nicht mehr erreichbar wäre (z.B. zerstört oder Energievorrat erschöpft), sodass nach einem gewissen Timeout Sensor *a* einfach einen anderen Sensor wählt. Folglich kann gerade einmal eine Nachricht als unnötigen Overhead erzwungen werden, da sich *a* merken kann, welcher Sensor nicht mehr zur Verfügung stand und ihn für weitere Kooperationsanfragen nicht mehr berücksichtigen muss. Fall (2) würde ein Angreifer wählen, falls er beabsichtigt, die Kanalverschlüsselungssicherheit herabzusetzen, da ihm in diesem Fall ggf. bereits genug Informationen (über andere bei der Kooperation beteiligten Schlüssel) vorliegen, um den Kanal zwischen *a* und *b* nun abzuhören, dazu später mehr. Im Fall (3) würde es im Anschluss Sensor *a* nicht gelingen einen Kanal zu *b* aufzubauen, da der Sitzungsschlüssel von *a* nun nicht mehr mit dem von *b* berechneten übereinstimmt. Gegenmaßnahmen kann auch hier das erneute wählen von anderen Kooperationspartnern sein. Mit diesem Verhalten würde es jedoch für Sensor *a* ersichtlich, dass das Netzwerk angegriffen wird und es können ggf. vorgesehene Gegenmaßnahmen eingeleitet werden, die ggf. die Basisstation mit einbeziehen, um den detektieren, böswilligen Sensor zu isolieren, wie Di Pietro et al in Ihrem Artikel [3] kurz skizzieren.

6.2.3 q-Composite Strategie nach Chan et at

In diesem Verfahren besteht ähnlich zum „direkten Protkoll" nach Di Pietro et al nur die Angriffsmöglichkeit, eine Key Discovery-Phase auszulösen.

6.2.4 Multipfad-Schlüsselverstärkung nach Chan et al

In diesem Verfahren wird ähnlich zum „kooperativen Protokoll" nach Di Pietro et al ein Sitzungsschlüssel vereinbart. Hierbei nehmen wir nun an, dass ein oder mehrere Pfade korrumpierte Sensoren beinhalten. Diese Sensoren haben wieder die bereits oben genannten drei Möglichkeiten: Fall (1) entspricht zwar wieder dem Fall, dass ein gewählter Sensor ausgefallen ist, jedoch kann je nach Pfadlänge, der erzeugte Overhead deutlich größer sein, als beim kooperativen Verfahren, da Sensor *a* nicht direkt mitbekommt, dass ein Teil des Schlüssels nicht bei *b* ankam und somit das Scheitern erst nach Senden einer chiffrierten Nachricht an *b* (welche *b* nun nicht entschlüsseln kann) bemerkt. Desweiteren ist im von Chan et al vorgeschlagenen Verfahren zum Finden der Intermediäre nicht vorgesehen, was passieren soll, wenn einer der Sensoren aus der Schnittmenge sich zwar durch Nachbarschaftserkundung auffinden lässt, aber keine Nachrichten weiterleitet. Sodass dieses Verfahren im Gegensatz zum vergleichbaren „kooperativen Protokoll" von Di Pietro et al, noch weitere Spezifikationen in der Implementierung erfordert, um gegen Attacken geschützt zu sein. Das gleiche gilt für Fall (3), wenn ein falscher Wert weitergeleitet wird. Im Fall (2), d.h. der Sensor hat den korrekten Wert weitergeleitet, besteht die gleiche Auswirkung auf die Abhörsicherheit wie beim „kooperativen Protokoll", mehr dazu später.

doi: 10.2313/NET-2011-07-1_12

6.2.5 Zufällige, paarweise Schlüsselstrategie nach Chan et al

Da dieses Verfahren die Paarbildung bereits in der Predeploymentphase festlegt, gibt es auch praktisch keine Angriffspunkte für eine DoS-Attacke auf der in diesem Artikel behandelten Ebene.

6.3 Abhörsicherheit

Das wohl wichtigste Merkmal einer Verschlüsselungsstrategie, stellt die Abhörsicherheit dar. In Sensornetzwerken, bei welchen die Sensoren meist physisch zugänglich sind und der (teilweisen) Verwendung von Pre-Shared Keys ist dies nochmals eine besondere Herausforderung, da einzelne Sensoren korrumpiert sein können oder durch Einsammeln und Auslesen des Speichers, der Angreifer in Besitz der Pre-Shared Keys gelangen kann.

6.3.1 Direktes Protokoll nach Di Pietro et al

Aufgrund der Bedeutsamkeit der Abhörsicherheit für dieses Thema, wurden im Artikel von Di Pietro et al [3] auch die dort vorgestellten Verfahren auf diesen Punkt hin untersucht. Es ist leicht einzusehen, dass eine Verbindung zwischen Sensor *a* und *b* nicht mehr abhör- und damit auch manipulationssicher, sobald der Angreifer in den Besitz aller Schlüssel, die *a* und *b* gemeinsam haben, gekommen ist. Da dieses Verfahren im Gegensatz zu dem q-Composite Verfahren von Chan et al [2] keine Mindestanzahl an gemeinsamen Schlüsseln vorschreibt, kann es auch sein, dass der Kanal nur durch ein einzelnen Pre-Shared Key verschlüsselt ist.

6.3.2 Kooperative Protokoll nach Di Pietro et al

Im „kooperativen Protokoll“ erhöht sich die Anzahl der verwendeten Schlüssel durch die kooperierenden Sensoren. Nimmt man an, dass die kooperierenden Sensoren nur teilnehmen, falls sie tatsächlich auch einen Schlüssel mit Sensor *b* teilen, so stellt die Anzahl der kooperierenden Sensoren eine untere Schranke für die zusätzlich hinzukommenden Schlüssel dar. Wobei diese untere Schranke nur gilt, falls Sensor *a* eine intelligente Auswahl an Kooperationssensoren getroffen hat. Wobei intelligent hierbei bedeutet, dass der Kooperationssensor mindestens ein Schlüssel mit *b* gemeinsam hat, welcher weder in Sensor *a* noch in einem anderen kooperierenden Sensor vorhanden ist. Das hierfür nötige Wissen kann Sensor *a* aus der Anwendung des Pseudo-Zufallsgenerators beziehen. Di Pietro et al erwähnen für Ihr „kooperatives Protokoll“ noch die Möglichkeit als kooperierende Sensoren nicht nur lokale Nachbarn hinzuziehen, sondern ggf. nur über mehrere Sprünge zu erreichende Sensoren mit einzubeziehen. Dies schwächt die Auswirkungen einer lokal begrenzten Infiltrierung durch korrumpierte Sensoren ein. Da Sensor *b* bei diesem Verfahren weder aktiv noch passiv direkt in Kommunikation mit den Kooperationssensoren treten muss, steigt hierbei auch der Overhead nicht so stark an, wie es bei der „Multipfad-Schlüsselverstärkung“ von Chan et al [2] der Fall wäre. Folglich kann das „kooperative Protokoll“ bei der gleichen Anzahl an Sprüngen weiter entferntere Sensoren mit einbeziehen, als dies bei der „Multipfad-Schlüsselverstärkung“ möglich wäre.

6.3.3 q-Composite Strategie nach Chan et at

Durch die Tatsache, dass zunächst eine gewisse Mindestanzahl *q* an Schlüsseln bei beiden Sensoren gemeinsam vorhanden sein müssen, erhöht zunächst die Sicherheit, wie bereits oben erwähnt. Jedoch muss bei diesem Verfahren die Größe des Schlüsselpools (Predeploymentphase) bei angenommener gleichbleibender Speichergröße in den Sensoren verringert werden. Das hat die Konsequenz, dass durch Einsammeln und Auswerten von Sensoren, der Angreifer bei der gleiche Anzahl an ausgewerteten Sensoren einen größeren Teil des Schlüsselpools in Erfahrung bringen konnte, worauf hin bereits Chan et al in Ihrem Artikel [2] selbst hinweisen. Es läuft hierbei auf den folgenden Trade-Off hinaus: je größer *q* gewählt wird, desto Resistenter ist das Netzwerk gegenüber das Auswerten/Korrumpieren einer kleinen Anzahl an Sensoren, wird jedoch Anfälliger gegen Angriffe/Sabotageakte auf eine Vielzahl von Sensoren.

6.3.4 Multipfad-Schlüsselverstärkung nach Chan et al

Aufgrund dass bei diesem Verfahren, ein völlig von den Pre-Shared Keys unabhängiger Sitzungsschlüssel generiert wird, der aus *j* Teilschlüsseln besteht, wobei jeder Teilschlüssel auf einem Sensordisjunkten Pfad zu Sensor *b* übertragen wird, kann ein Angreifer die Verbindung zwischen Sensor *a* und *b* nur dann erfolgreich abhören, wenn er in Besitz aller *j* Teilschlüssel gelangen ist. Dies erhöht also die Sicherheit auf den ersten Blick enorm, gegenüber allen oben genannten Verfahren, da der Angreifer, selbst wenn er im Besitz des gesamten Schlüsselpools gelangt ist, den Sitzungsschlüssel immer noch nicht vorhersagen kann. Allerdings wäre er somit in der Lage, die generieten Teilschlüssel bei der Übertragung von Sensor *a* erfolgreich abzuhören und somit den Sitzungsschlüssel zu erhalten. Auch wenn der Angreifer nicht alle Pre-Shared Keys des Schlüsselpools kennt, er jedoch in der Umgebung von Sensor *a* und *b* ausreichend viele korrumpierte Sensoren hat, kann er ebenfalls Kenntnis über einen Großteil oder gar den gesamten Sitzungsschlüssel erhalten.

6.3.5 Zufällige, paarweise Schlüsselstrategie nach Chan et al

Diese Strategie bietet die größte Abhörsicherheit an, da die Paarbildung bereits in der Predeploymentphase festgelegt wurde und damit keine Schlüsselinformationen ausgetauscht werden müssen. Selbst wenn ein Angreifer Sensoren auswertet, fallen ihm dabei nur Schlüssel in die Hände, die ausschließlich von diesem Sensor genutzt werden und in keinem weiteren Verbindungspaar zweier Sensoren eine Rolle spielen.

7. ZUSAMMENFASSUNG

Zusammenfassend fällt auf, dass die Verfahren von Di Pietro et al [3] dynamischer wirken als die von Chan et al [2], da aufgrund der Idee der Nutzung eines Pseudozufalls-Zuteilungsgenerator jeder Sensor weiß (bzw. vielmehr berechnen kann), welche Schlüssel (anhand der Indizes) ein anderer Sensorknoten aufweisen muss. Wobei hierfür nur der Algorithmus und der Initialisierungswert gespeichert werden müssen. Ein weiterer großer Vorteil der Verfahren von Di Pietro et al, betrifft die Authentifizierung, welche einem durch das Verfahren praktisch geschenkt wird. Allerdings könnte man das „direkte Protokoll“ von Di Pietro um eine Mindestanzahl an gemeinsamen Schlüsseln erweitern, indem man die Forderung aus dem „q-Composite“-Verfahren von Chan et al mit übernehmen würde. Gegebenenfalls erweitern könnte man das direkte oder kooperative Protokoll, um die Nutzung eines Sitzungsschlüssels, der unabhängig von den Pre-Shared Keys ist, z.B. unter Einbeziehung des

doi: 10.2313/NET-2011-07-1_12

Schlüsselverstärkung nach Chan et al auf einer höheren Protokollebene über dem direkten/kooperativen Protokoll. Die Dynamik der Protokolle von Di Pietro et al zeichnet sich auch gerade in mobilen WSN aus, bei der sich die Nachbarn der Sensoren ständig ändern und abwechseln, nicht zuletzt auch, da die Key Disovery-Phase aus nur einem einzigen Broadcast (pro Sensor) besteht.

Für statische Sensornetzwerke dürfte die „zufällige, paarweise Strategie“ von Chan et al, klar im Vorteil sein, da sie für diesen Einsatzzweck, den geringsten Overhead mit sich bringen würde (gut für die Lebensdauer der Energievorräte) und auch beim Thema Abhörsicherheit durch ihre Effizienz besticht.

Generell gilt jedoch für probabilistische Verfahren, dass es mit einer gewissen Wahrscheinlichkeit passieren kann, dass einzelne Sensoren, obwohl sie in Funkreichweite mit ein oder mehreren Nachbarn sind, nicht in das Netzwerk integriert werden können, da sie mit ihren unmittelbaren Nachbarn keine Schlüssel teilen (bzw. nicht ausreichend viele, je nach Verfahren). Dies sollte bei der Planung der Funkreichweite bzw. der Sensordichte, also der Abschätzung der durchschnittlichen Anzahl an Nachbarn, für die Ausbringung berücksichtigt werden.

8. LITERATUR

[1] Jaydip Sen, „A Survey on Wireless Sensor Network Security”, International Journal of Communication Networks and Information Security (IJCNIS), Vol. 1, No. 2, August 2009

[2] Haowen Chan, Adrian Perrig und Dawn Song, „ Random Key Predistribution Schemes for Sensor Networks”, Proceedings of the 2003 IEEE Symposium on Security and Privacy (SP’03), 2003

[3] Roberto Di Pietro, Luigi V. Mancini und Alessandro Mei, „Random KeyAssignment for Secure Wireless Sensor Networks”, Proceedings of the 1st ACM Workshop Security of Ad Hoc and Sensor Networks Fairfax, Virginia, 2003

[4] Sasikanth Avancha, Jeffrey Undercoffer, Anupam Joshi und John Pinkston, „Security for Wireless Sensor Networks”, Kluwer Academic Publishers Norwell, MA, USA, 2004

[5] Djamel Djenouri und Lyes Khelladi, „A survey of security issues in mobile ad hoc and Sensor Networks“, IEEE Communications Surveys & Tutorials, Fourth Quarter, 2005

[6] Laurent Eschenauer undVirgil D. Gligor, „A key-management scheme for distributed sensor networks“, Proceedings of the 9th Computer Communication Security (CCS 2002), Washington, MA, November 2002

[7] R. Merkle, „Secure communication over insecure channels“, Communications of the ACM, 21(4):294–299, 1978

doi: 10.2313/NET-2011-07-1_12

Sicherheit in WSNs – Node Replication Angriffe

Johannes Schlicker
Betreuerin: Corinna Schmitt
Seminar Innovative Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: schlicke@in.tum.de

KURZFASSUNG

In dieser Ausarbeitung wird beschrieben, wie Angriffe auf drahtlose Sensornetzwerke abgewehrt werden können. Es gibt verschiedene Verfahren, solche Angriffe zu erkennen. Im Folgenden wird erklärt, wie „Node Replication Attacks“ ablaufen und sowohl auf verbreitete Verfahren wie „Randomized Multicast“ [1] und „Line-Selected Multicast“ [1], aber auch detaillierter auf ein neues Verfahren eingegangen [2]. Dieses neue Verfahren versucht die Nachteile und Einschränkungen, wie hohen Rechenaufwand und immer noch vorhandene Sicherheitslücken, etablierter Verfahren zu verringern.

Schlüsselworte

Sensornetzwerk, Angriff, Node Replication Attack, Sicherheitsprotokoll

1. EINLEITUNG

Drahtlose Sensornetzwerke (WSNs – Wireless Sensor Networks) werden heute hauptsächlich zur Überwachung von eingegrenzten Gebieten eingesetzt. Der Vorteil der Verwendung von Sensornetzwetzen ist die Skalierbarkeit und einfache Erweiterbarkeit. Neue Sensorknoten (oder auch nur Knoten genannt) können ohne Änderungen an der Konfiguration des Netzwerks als „neue Generation“ hinzugefügt werden. Ein wunder Punkt vieler Protokolle zur Abwehr von Angriffen ist der Vorgang der Erweiterung des Netzwerks, da nicht sichergestellt werden kann, dass keine feindlichen Knoten ins Netzwerk aufgenommen werden [1]. Feindliche Knoten sind im Falle von Node Replication Angriffen Kopien von Sensorknoten, die nach erfolgreichem Einklinken ins Sensornetz, Informationen auslesen oder sogar Schaden anrichten können.

2. ANGRIFFE AUF SENSORNETZE

Bei der hier behandelten Bedrohung für Sensornetzwerke – den Node Replication Attacks – gibt es verschiedene Angriffspunkte, die im Folgenden kurz erläutert werden. Abbildung 1 zeigt schematisch den Überblick über ein Sensornetzwerk, welches durch neue Knoten erweitert wurde. Ein Knoten wurde kompromittiert und könnte sowohl ein Klon eines bereits vorhanden, als auch eines neu eingesetzten Knotens sein kann. Aus diesem Szenario ergeben sich zwei Angriffspunkte: Klonen eines vorhandenen Knotens und Klonen eines neuen Knotens [1][2]. Beide Vorgehen müssen von einem Sicherheitsprotokoll abgefangen werden können.

Um Abwehrmechanismen verstehen zu können, muss zunächst erklärt werden, wie Angriffe ablaufen.

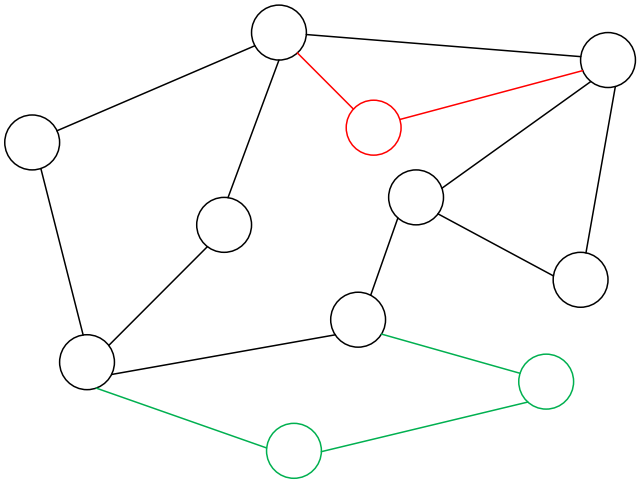


Abbildung 1: Aufbau eines Sensornetzwerks nach dem Hinzufügen neuer Knoten

2.1 Ablauf eines Angriffs

Der erste Schritt bei einem Node Replication Attack ist das Klonen eines Sensorknotens. Dabei werden sowohl Kommunikationsprotokolle, als auch eventuell vorhandene Sicherheitsmechanismen und -zertifikate kopiert [2]. Zusätzlich ist der bösartige Knoten aber natürlich mit Schadsoftware bestückt. Eine wichtige – wenn auch offensichtliche – Eigenschaft duplizierter Knoten ist, dass die Signatur dieses Knotens doppelt im Sensornetz vorhanden ist. Die Signatur ist meistens ein Schlüssel, der zur Authentifizierung gegenüber dem Sensornetz dient.

Des Weiteren ist in jedem Sensornetzwerk ein Protokoll vorhanden, welches die Erweiterung eines Netzwerkes definiert. Ein kompromittierter Knoten, muss diesen Protokolldurchlauf aber nicht zwingend durchführen, da er vorgeben kann, bereits Mitglied einer vorher eingesetzten Generation zu sein. Individuelle Schlüssel, die für die Kommunikation verwendet werden, müssen in diesem Fall anderweitig besorgt werden.

Ist ein neuer (noch nicht im Netzwerk registrierter) Knoten der Angriffspunkt, muss ein Mechanismus durchlaufen werden, der den geklonten Knoten als legitim eingesetzten Knoten identifiziert.

Das technische Ziel eines jeden Angriffs auf ein Sensornetzwerk ist das erlangen eines Schlüssels, der für die Kommunikation mit anderen Knoten unerlässlich ist. Dieser Schlüssel dient dem Ver- und Entschlüsseln zu sendender oder empfangener Nachrichten.

doi: 10.2313/NET-2011-07-1_13

Welches Schlüsselprotokoll verwendet wird, soll hier nicht genauer beschrieben werden. Es sind jedoch sowohl symmetrische als auch asymmetrische Verfahren denkbar.

2.2 Annahmen

Um Angriffe abwehren zu können, müssen zusätzlich Annahmen getroffen werden. Es wird beim Entwickeln eines Protokolls davon ausgegangen, dass nur ein bestimmter Prozentsatz von Knoten kompromittiert wird. Wird die Kontrolle über jeden einzelnen Knoten übernommen, können Sicherheitsmechanismen einfach ausgeschaltet werden [1]. Außerdem wird die Annahme getroffen, dass feindliche Knoten den Protokollen weiterhin folgen und Informationen ausschließlich von kompromittierten Knoten lesen können [1]. Des Weiteren hat jeder Knoten, der einmal eingesetzt wurde eine feste, unveränderbare Position.

3. ZENTRALISIERTE PROTOKOLLE

Bisherige Ansätze waren meist zentralisiert. Das bedeutet, dass ein zentraler Punkt für die Sicherheit des Sensornetzes verantwortlich war. Auf diese Weise kann relativ einfach sichergestellt werden, dass Angriffe erkannt werden. Eine zentrale Station vergleicht die Signaturen aller Knoten und kann nach Erkennen einer doppelt vorhandenen ID das Sensornetz sofort warnen. Dies hat jedoch mehrere entscheidende Nachteile.

Zum einen entsteht ein hoher Kommunikationsaufwand, da alle Knoten im Netzwerk Informationen zu einem zentralen Knoten weiterleiten müssen. Basierend auf der Tatsache, dass nicht alle Knoten direkt miteinander verbunden sind, müssen Sensoren nicht nur ihre eigenen Daten versenden, sondern zusätzlich auch die von weiter entfernten Sensoren weiterleiten. Ausgehend von einer durchschnittlichen Pfadlänge von $O(\sqrt{n})$ zur Base Station, wobei n die Anzahl der Knoten ist, müssen $O(n\sqrt{n})$ Übertragungen vorgenommen werden, bis alle Informationen versendet wurden [1]. Diesem Kommunikationsaufwand ist ein großes Sensornetzwerk nicht gewachsen, da die begrenzten Energieressourcen vor Allem bei Knoten, die sehr viel senden müssen, sehr schnell erschöpft sind.

Ein weiterer Nachteil ist, dass die Base Station zum „Single Point of Failure“ wird. Das bedeutet, dass ein Ausfall oder die Kompromittierung dieser Base Station das komplette Sensornetzwerk lahmlegen und somit zu einem beliebten Angriffsziel werden kann [1].

Abgesehen von diesen Nachteilen, besteht zusätzlich die Gefahr, dass verteilte Angriffe nicht erkannt werden. Das bedeutet, dass ein Angriff, der auf mehrere Knoten gleichzeitig abzielt und somit möglicherweise einen Teil eines Netzwerkes abtrennt, nicht entdeckt werden kann, da ein kompromittierter Verbindungsknoten Informationen verfälscht weitergibt.

4. DEZENTRALISIERTE PROTOKOLLE

In diesem Abschnitt werden verschiedene Protokolle erklärt, die die beiden in Abschnitt 2 beschriebenen Angriffspunkte schließen sollen und den Annahmen aus Abschnitt 2.2 folgen. Außerdem werden die Nachteile zentralisierter Ansätze aus Abschnitt 3 minimiert.

Grundsätzlich sind dezentralisierte Protokolle nicht an Base Stations gebunden, was offensichtlich den „Single Point of Failure“ entfernt.

In den folgenden Abschnitten wird zunächst die grundlegende Idee dezentralisierte Protokolle erläutert, um danach zwei etablierte Ansätze – das Randomized Multicast Protokoll und das Line-Selected Multicast Protokoll – genauer zu erläutern. Schließlich wird noch ein neuer Ansatz beschrieben, der den Energiebedarf bei gleicher Sicherheit nochmals reduziert.

4.1 Idee

Die Idee einer dezentralen Lösung ist, dass an die Position geknüpfte Signaturen der einzelnen Knoten nicht an eine zentrale Station übertragen werden, sondern an andere gleichwertige Knoten im Netzwerk. Wenn jeder Knoten über jeden anderen Bescheid weiß, können – unter der Voraussetzung, dass die Broadcasts jeden Sensor erreichen – alle doppelten Positionsangaben erkannt werden. Allerdings ist dabei der Kommunikationsaufwand im Bereich von $O(n^2)$ noch höher, als bei der zentralisierten Herangehensweise [1].

Um diesen Ansatz zu verbessern, werden die Positions- und Signaturnachrichten nur an eine bestimmte Gruppe von Knoten gesendet. Die Knoten werden nach einem Algorithmus basierend auf der Signatur ausgewählt, um sicherzustellen, dass geklonte Knoten ihre Positionsinformation an die gleiche Gruppe wie auch der kompromittierte Sensor weiterleiten. Das birgt allerdings wieder eine entscheidende Sicherheitslücke: Kennt der Angreifer die Signatur eines Knotens und den Algorithmus, mit dem die sogenannten „Witness Nodes“ ausgewählt werden, kann er versuchen auch alle diese Witness Nodes zu übernehmen, um einen neuen Sensor unbemerkt einzuschleusen [1].

Das Randomized Multicast Protokoll und das Line-Selected Multicast Protokoll gehen daher noch einen Schritt weiter und setzen als Witness Nodes nicht direkte Nachbarknoten ein und randomisieren außerdem die Auswahl.

4.2 Das Randomized Multicast Protokoll

Das Randomized Multicast Protokoll setzte voraus, dass jeder Knoten im Sensornetz seine Position kennt und, dass jeder einzelne Sensor Signaturen erstellen kann, die ihn eindeutig identifizieren [1][2].

Die Idee des Protokolls ist das Ausnutzen des sogenannten „Geburtstags-Paradoxon“ [3]. Ein kurzes Beispiel erklärt dieses Problem am besten: „Sind mindestens 23 Personen in einem Raum, so ist die Wahrscheinlichkeit, dass zwei oder mehr am selben Tag im Jahr Geburtstag haben, größer als 50%.“ Bei 50 Personen liegt die Wahrscheinlichkeit sogar bei über 97%. Die Tatsache dieser relativ geringen Menge an Personen wird hier auf das Sensornetz übertragen, wobei sich hier natürlich die Frage stellt, wie viele zufällige Witness Nodes von Nöten sind, um mit hoher Wahrscheinlichkeit einen Node Replication Angriff zu erkennen.

doi: 10.2313/NET-2011-07-1_13

4.2.1 Beschreibung des Protokolls

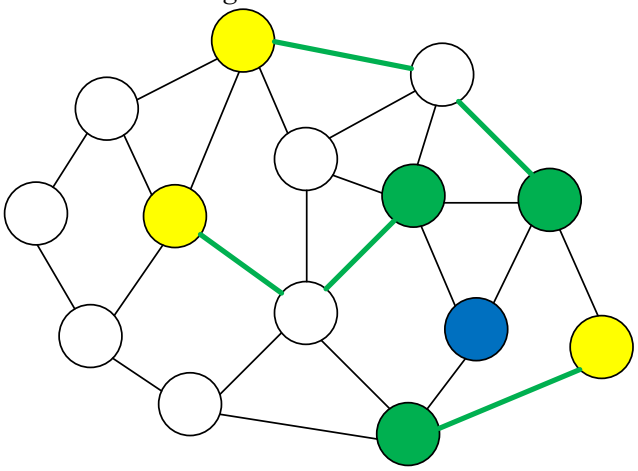


Abbildung 2: Randomized Multicast Protokoll

Die angesprochene Erweiterung der dezentralisierten Idee besteht darin, dass Knoten nicht direkte Nachbarn als Witness Nodes wählen, sondern zufällige Knoten aus dem Sensornetz. In diesem Fall trifft die Wahl der zufälligen Knoten aber nicht der Knoten, dessen Authentizität überprüft werden soll, sondern dessen direkte Nachbarn. In Abbildung 2 ist dies schematisch dargestellt. Der blaue Knoten sendet seine Signatur und seine Position an alle direkten Nachbarn, die grün dargestellt sind. Diese Nachbarn suchen dann zufällig eine bestimmte Anzahl an geographischen Punkten im Netzwerk (hier im Beispiel ist die Anzahl nur 1). Durch geographisches Routing werden dann die Knoten, die diesen Punkten am nächsten liegen, als Zeugenknoten ausgewählt und die Signatur/Position-Kombination an diese weitergeleitet. Am Beispiel des Geburtstagsparadoxon kann man erkennen, dass eine relativ geringe Anzahl an Zeugenknoten ausreicht, um sicherzustellen, dass – trotz der zufälligen Auswahl der Zeugenknoten – mit sehr hoher Wahrscheinlichkeit eine doppelt vorhandene Signatur mit unterschiedlichen Positionsangaben entdeckt wird.

Ein weiterer Punkt, der im Protokoll beachtet werden muss, ist die Integrität und vor Allem Authentizität der Nachrichten, die versendet werden. Um diese zu gewährleisten liegt dem Sensornetz ein Signaturverfahren zugrunde. Jeder Knoten kann seine Identität mit einem privaten Schlüssel garantieren. Die Nachrichten die versendet werden, haben folgendes Format:

$$\langle ID, l, \{H(ID, l)\}_{KP} \rangle$$

Dabei ist *ID* die Signatur des sendenden Knotens, *l* die Positionsangabe und $\{H(ID, l)\}_{KP}$ ein Hashwert über den Nachrichteninhalt, der mit dem privaten Schlüssel *KP* des Knotens verschlüsselt ist. Dieser Hashwert dient dabei der Sicherstellung der Integrität der Nachricht. Ist die Verschlüsselung des Hashwertes falsch, so schlägt die Integritätsprüfung fehl und die Nachricht wird verworfen [1].

4.2.2 Sicherheit und Effizienz

Das Maß an Sicherheit des Protokolls ist gleichzusetzen mit der Wahrscheinlichkeit, dass erkannt wird, wenn mindestens zwei Knoten mit der gleichen Signatur zwei verschiedene Positionen haben. Ohne hier genauer auf die Herleitung einzugehen stößt man bei der Analyse des Problems auf folgende Formel, die genau diese Wahrscheinlichkeit angibt:

$$P \geq 1 - e^{\frac{-p^2 d^2 g^2 L(L-1)}{n \quad 2}}$$

Dabei ist n die Anzahl der Knoten im Sensornetz, p die Wahrscheinlichkeit, dass ein Nachbarknoten Positionsinformationen doppelt weiterleitet, g die Anzahl der Zeugenknoten, d der Grad jedes einzelnen Knotens (also die durchschnittliche Entfernung) und L die Anzahl der Klone des kompromittierten Knotens [1][3].

Spielt man dieses Szenario mit realistische Zahlen durch, so kommt man mit $n = 10000$, $g = 100$, $d = 20$ und $p = 0{,}05$ auf eine Wahrscheinlichkeit von 63%, wenn ein Knoten einmal geklont wurde, bei nur zwei Klonen liegt die Wahrscheinlichkeit, dass mindestens einer erkannt wird schon bei über 95%.

Problematisch bei diesem Protokoll ist allerdings wieder der Kommunkationsaufwand von $O(n^2)$. In der Realität wird allerdings nicht immer eine perfekte Erkennung gefordert. Oftmals reicht es aus, wenn nicht jeder einzelne Klon erkannt wird, sondern das System erst bei mehreren Klonen alarmiert wird. Dafür kann beispielsweise die Anzahl der Zeugenknoten reduziert werden [1]. Durch weitere kleinere Optimierungen kann die Anzahl der Nachrichten pro Knoten auf $O(\sqrt{n}\, p\, g)$ reduziert werden.

4.3 Das Line-Selected Multicast Protokoll

Das Line-Selected Multicast Protokoll ist eine Erweiterung des Randomized Multicast Protokolls [2]. Ziel dieser Erweiterung ist es, den Kommunikationsaufwand weiter zu reduzieren, da dieser hohe Energiekosten erzeugt und somit die Lebensdauer eines Sensornetzwerks beträchtlich mindert.
Wie man in Abbildung 2 sehen kann, kreuzen die Nachrichten der Nachbarknoten mehrere unbeteiligte Knoten (grüne Pfade). Das hier beschriebene Protokoll nutzt die Tatsache aus, dass einzelne Sensoren im Netzwerk sowohl als Router, als auch als Sensor fungieren und somit die Nachrichten, die sie in ihrer Funktion als Router eigentlich nur weiterleiten, auch speichern können [1]. Somit ist die Signatur-/Positionsinformation eines Knotens nicht nur auf den Zeugenknoten, sondern auch auf jedem Knoten auf dem Weg zu den Zeugenknoten vorhanden, ohne den Kommunikationsaufwand dabei erhöht zu haben.

4.3.1 Beschreibung des Protokolls

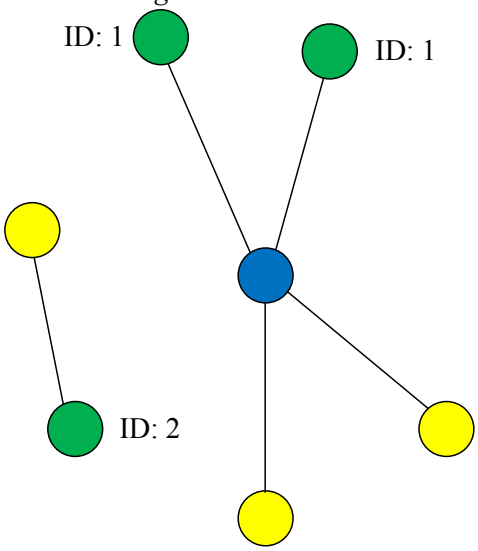


Abbildung 3: Line Selected Multicast Protokoll

doi: 10.2313/NET-2011-07-1_13

Das in Abbildung 3 (zur Vereinfachung wurden die Nachbarknoten nicht mit in die Zeichnung aufgenommen) dargestellte Szenario ist, dass die oberen beiden grünen Knoten doppelt vorhanden sind. Einer dieser beiden Knoten muss also ein bösartiger Klon sein. Bei Anwendungen des Randomized Multicast Protokolls würden Signatur- und Positionsinformationen zu jeweils einem der beiden grünen Knoten nur bei den beiden unteren gelben Knoten vorliegen. Dass die Knoten doppelt vorliegen würde man also nicht erkennen.

Man kann aber erkennen, dass sich die Pfade über die die Nachrichten verschickt wurden, bei dem blauen Knoten kreuzen. Wird hier das Line-Selected Multicast Protokoll verwendet, ist der Ablauf folgender: Die Nachricht des linken Knotens (mit ID 1) wird an den rechten Zeugenknoten (gelb) versendet und auf jedem Knoten, der auf dem Pfad dorthin liegt zwischengespeichert. Sendet der rechte Knoten (mit ID 1) eine Nachricht mit gleicher Signatur aber anderen Positionsinformationen, so muss diese Nachricht nicht bis zum Zeugenknoten weitergeleitet werden, da die Kollision (also das Auftreten verschiedener Positionsinformationen zu gleichen IDs) bereits beim blauen Knoten erkannt wird.

4.3.2 Sicherheit und Effizienz

Um die beiden Protokolle genauer vergleichen zu können, wird auch hier kurz auf die Fehlererkennungswahrscheinlichkeit und den Kommunikationsaufwand eingegangen.

Beim Line-Selected Multicast Protokoll kommt wiederum die Lösung eines anderen Problems zum Einsatz. Sylvesters sogenanntes „Four-Point Problem“ besagt, dass die Wahrscheinlichkeit, dass vier zufällig in einem Kreis gewählte Punkte eine konvexe Hülle bilden, $\frac{35}{12\pi^2}$ ist. Ausgehend von dieser Wahrscheinlichkeit lässt sich berechnen, wie wahrscheinlich eine Kreuzung von Pfaden und somit auch wie wahrscheinlich ein früheres Erkennen einer Kollision ist.

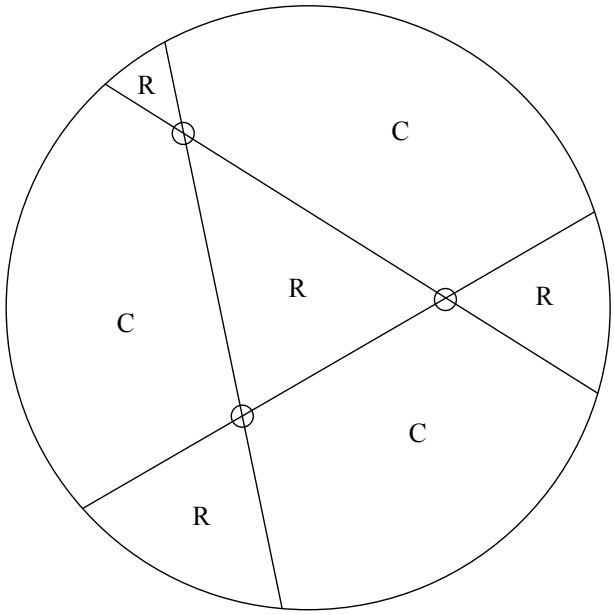


Abbildung 4: Das "Four-Point Problem"

Wählt man beispielsweise drei beliebige Punkte in einer kreisförmigen Fläche, berechnet sich die Wahrscheinlichkeit, dass ein vierter Punkt eine Kreuzung verursacht wie folgt [1][4]:

$$P = \frac{1}{3}\left(1 - \frac{35}{12\pi^2}\right) \approx 0{,}235$$

Das Ergebnis der Klammer ist dabei die Wahrscheinlichkeit, dass der vierte gesetzte Punkt eine Form mit einer Mulde produziert (im Gegensatz zu einer konvexen Hülle). Der vierte Punkt muss dabei in eine in Abbildung 4 mit „R“ beschriftete Region fallen. Um ausgehend vom Punkt oben links eine Kreuzung zu erreichen, muss der vierte Punkt als in den unteren mit „C“ beschrifteten Bereich fallen [1].

Im Falle eines Sensornetzes ist die Annahme allerdings komplizierter. Man geht davon aus, dass alle Pfade mehr oder weniger radial von der Mitte des Kreises nach außen laufen [1]. Diese Annahme beruht auf der Tatsache, dass die Pfade von einem Knoten – oder besser – dessen Nachbarn ausgehen. Durch Simulationen kann man berechnen, dass die Wahrscheinlichkeit, dass sich mindestens zwei Pfade überschneiden bei nur fünf Pfaden bei ungefähr 95% liegt [1/4].

Ausgehend davon, dass jeder Pfad eine Länge von $O(\sqrt{n})$ hat, liegt der Kommunkationsaufwand nur bei $O(n\sqrt{n})$, was eine wesentliche Verbesserung darstellt.

5. FORSCHUNG VON BEKARA UND LAURENT-MAKNAVICIUS

Um den Kommunikationsaufwand der vorigen Protokolle zu minimieren haben Chakib Bekara und Maryline Laurent-Maknavicius ein neues Protokoll entwickelt. Dieses Protokoll basiert auf einer anderen Idee. Deshalb gibt es zusätzliche Annahmen, die es sicherzustellen gilt.

So existiert eine zentrale Base Station, die nicht kompromittiert werden kann. Diese Annahme ist insofern denkbar, da die Basisstation nur zum Einsetzen neuer Generationen verwendet wird und in dieser Zeit überwacht und ansonsten abgeschaltet werden kann. Diese Basisstation setzte einzelne Knoten in fortlaufenden Generationen ein. Außerdem besitzt jeder einzelne Knoten Kenntnis über die Nummer der höchsten Generation.

Des Weiteren verwendet das Protokoll ein symmetrisches Verschlüsselungsverfahren. Sendet ein Knoten mit einem validen Schlüssel, gilt er als nicht kompromittiert. Die Zeit, die für den Schlüsselaustausch mit den direkten Nachbarn aufgebracht wird liegt unter T_{est}. Ein Angreifer braucht allerdings $T_{comp} > T_{est}$, um einen Knoten zu kompromittieren [2]. Diese Annahme ist realistisch, da zusätzlich zu der Zeit, die der Schlüsselaustausch in Anspruch nimmt, noch der Klonvorgang vorgenommen werden muss.

Zuletzt muss sichergestellt sein, dass die internen Uhren aller Knoten mit der Basisstation synchron sind. Dazu wird ein authentifizierter „Beacon“ verwendet [2] – eine kleine Nachricht, die in regelmäßigen Abständen die Uhren abgleicht.

5.1 Aufbau und Erweiterung des Netzwerks

Wie bereits erwähnt werden neue Sensoren in Gruppen eingesetzt. Eine neue Gruppe wird als Generation bezeichnet. Der erste Schritt, den jeder Knoten ausführt, ist ein Schlüsselaustausch mit seinen direkten Nachbarknoten. Erhält ein Sensor also eine Anfrage zum Schlüsselaustausch, geht dieser davon aus, muss die Anfrage von einem Knoten einer Generation größer oder gleich der eigenen Generation stammen, da nur neue Knoten Schlüssel anfragen. Ein erstes Aussortieren maligner Knoten kann also bereits hier getroffen werden. Denn wenn ein bereits etablierter Knoten geklont wird und einen Schlüssel anfragt, stammt die

doi: 10.2313/NET-2011-07-1_13

Anfrage von einer alten Generation und kann als bösartig identifiziert werden. Auf die Kenntnis über die höchste Generation wird später in Abschnitt 5.3 eingegangen [2].

5.2 Beschreibung des Protokolls

5.2.1 Schritt 1 – Hello Nachricht

Wenn ein Knoten *u* eingesetzt wird, sendet dieser zunächste eine „Hello“ Nachricht an alle direkten Nachbarn:

$$\langle Hello, j, Id_u, N_u \rangle$$

N ist dabei ein zufällig steigender Wert, der garantiert, dass keine bereits versendeten Nachrichten wiederverwendet werden können. *j* beschreibt die Generation des sendenden Knotens. *Id* ist eine eindeutige Signatur.

5.2.2 Schritt 2 – Schlüsselgenerierung

Jeder Empfänger einer solchen *Hello* Nachricht führt nun folgende Überprüfung durch:

Zunächst wird überprüft, ob $j = i + 1$ ist. *i* ist dabei die höchste bisher eingesetzte Generation. Schlägt diese Überprüfung fehl, wird die Anfrage verworfen, da der anfragende Knoten bereits eingesetzt ist.

Wenn der überprüfende Knoten zu einer Generation größer als *i* gehört, berechnet er K_{VU} mit Hilfe einer Funktion, die aus dem Hash Wert von $i + 1$ konkateniert mit der *Id* einen Schlüssel berechnet. Dieser Schlüssel wird dann wie folgt verpackt zurückgesendet.

$$\langle z, Id_v, N_v, MAC_{K_{VU}}(z, Id_v, N_v, N_u) \rangle$$

Ist jedoch $j = z = i + 1$ und der von *v* gesetzte Timer, der auf T_{est} gesetzt ist, noch nicht abgelaufen, wird wie im eben beschriebenen Fall vorgegangen. Dieser Fall deckt den Schlüsselaustausch zwischen zwei Knoten ab, die beide der neuesten Generation angehören. Ist der Timer abgelaufen wird die Anfrage verworfen, da davon ausgegangen wird, dass der anfragende Knoten kompromittiert wurde.

5.2.3 Schritt 3 – Schlüsselbestätigung

Im dritten Schritt bestätigt Knoten *u* den Empfang des Schlüssels und kann danach am normalen Netzwerkverkehr teilnehmen. Dazu wird zunächst K_{UV} berechnet. Die Berechnung verwendet dabei die gleiche Funktion wie der Partnerknoten, übergibt der Hashfunktion aber diesmal die Generationsnummer des verifizierenden Knotens konkateniert mit dessen Signatur Id_v. Folgende Nachricht wird anschließend zurückgeschickt [2]:

$$\langle ok, MAC_{K_{UV}}(ok, N_v) \rangle$$

Beide Knoten verifizieren dabei die Authentizität des jeweils anderen Knotens mit Hilfe der Schlüsselberechnungen. Stimmen die Ergebnisse nicht überein liegt ein Fehler vor.

Die paarweise authentifizierten Knoten sind nun sicher legal Teilnehmer und können sich mit den paarweise generierten Schlüsseln Nachrichten senden.

5.3 Berechnen der höchsten Generation

Bei der Verifizierung, ob ein neuer Knoten der höchsten Generation angehört kommt wieder die Basisstation zum Einsatz. Diese setzt dabei ein statisches Verfahren ein, das nach jeder Zeitspanne *T* neue Generationen einsetzt. Dabei wird beim Einsetzen der ersten Generation die Zeit T_1 auf 0 gesetzt. Für jede weitere Generation wird der Timer sukzessive erhöht. Jeder Knoten kennt die aktuelle Zeit, die Zeit T_i, zu der er eingesetzt wurde und die Zeitspanne *T*. Er kann damit verifizieren, dass ein Knoten aus einer neuen Generation stammt, indem er folgende Überprüfung durchführt [2]:

$$0 < t_{current} - (i - 1) * T < T$$

$t_{current}$ ist dabei die aktuelle Zeit und *T* eine von der Basisstation festgesetzte und im Sensornetz publizierte Zeitspanne nach welcher neue Generationen eingesetzt werden. *i* bezeichnet wieder die neueste Generation.

5.4 Sicherheit

Das vorgestellte Protokoll hat allerdings in manchen Fällen noch Probleme, Angreifer zu erkennen.

Der erste Fall ist: Ein neu eingesetzter Knoten ist ein Klon eines neu eingesetzten Knotens. Wenn der Knoten also noch keinen Schlüsselaustausch vorgenommen hat, aber schon geklont ist, läuft der Austausch natürlich innerhalb der erlaubten Zeitspanne T_{est} ab. Abhilfe schafft hierbei die Einführung einer neuen Zeitspanne T_{max}, mit $T_{est} < T_{max} < T_{comp}$. T_{max} ist dabei die maximal zulässige Zeit nach dem Einsetzen einer neuen Generation, in welcher ein Schlüsselaustausch stattfinden darf. Der Schlüsselaustausch darf als nicht nur maximal T_{est} dauern, sondern muss innerhalb eines Zeitraums T_{max} durchgeführt worden sein.

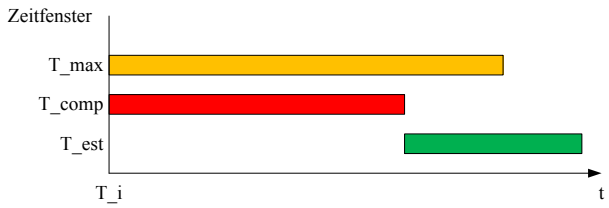


Abbildung 5: Kompromittierung eines neuen, noch nicht eingesetzten Knotens

Abbildung 5 verdeutlicht diesen Fall. T_i ist die Zeit zu der eine neue Generation *i* eingesetzt wurde. Ab diesem Zeitpunkt kann ein Angreifer aktiv werden, da vorher die neuen Knoten noch nicht existieren. Der zweite Balken visualisiert T_{comp}. Der erste Balken T_{max} und der unterste schließlich T_{est}. Ohne T_{max} könnte ein geklonter Knoten bereits vor dem Versuch eines Schlüsselaustausches geklont worden sein und den Austausch legitim vornehmen. Mit T_{max} würde der Gesamtvorgang (Klonen und Schlüssel austauschen) allerdings die maximale Zeitspanne überschreiten und verworfen werden [2]. Diese Lösung kann aber auch dazu führen, dass „langsame“ authentische Knoten verworfen werden.

Ein weiterer schwer zu identifizierender Fall ist, wenn ein alter Knoten geklont wurde und sich Zugang zum Netzwerk über einen neuen Knoten verschaffen will, der legitim dem restlichen Netzwerk beitritt. Der geklonte Knoten hat bis zu diesem Zeitpunkt keine andere Verbindung zum Sensornetz. Um dieses Problem zu lösen müsste man erkennen, dass ein Nachbarknoten eines neuen Knotens existiert, zu dem aber sonst keine sichere Verbindung besteht. Träte so ein Fall auf, könnte man den geklonten Knoten als bösartig erkennen und verwerfen [2].

5.5 Effizienz

Ausgehend davon, dass wie bisher jeder Knoten mit maximal $\sqrt{n}$ weiteren Knoten verbunden ist liegt sowohl der Kommunikationsaufwand, als auch der Speicheraufwand im Bereich von $\sqrt{n}$.

doi: 10.2313/NET-2011-07-1_13

6. ZUSAMMENFASSUNG

Diese Arbeit hat sich mit verschiedenen Methoden der Abwehr von Node Replication Angriffen auseinandergesetzt. Dabei hat sich herausgestellt, dass zentralisierte Herangehensweisen einen „Single Point of Failure“ haben, dessen Kompromittierung das ganze Netzwerk lahmlegen kann. Dezentralisierte Protokolle haben dieses Problem nicht, allerdings ist der Speicher-, Rechen- und Kommunikationsaufwand zur Abwehr von Angriffen um ein vielfaches größer.

Es wurden drei Protokolle vorgestellt, die sich im Maße der Sicherheit und des Aufwands unterscheiden. Vergleicht man die drei genauer beschriebenen Protokolle in Bezug auf die aufwändigsten Operationen Kommunikation und Speichernutzung, so kommt man auf folgendes Ergebnis:

Tabelle 1: Vergleich der Protokolle

	Kommunikation	Speicher
Randomized Multicast	$O(n^2)$	$O(\sqrt{n})$
Line-Selected Multicast	$O(n\sqrt{n})$	$O(\sqrt{n})$
Verbesserung Abschnitt 5	$O(\sqrt{n})$	$O(\sqrt{n})$

Im Vergleich zu zentralisierten Protokollen unterscheiden sich dezentrale Ansätze vor Allem im Speicheraufwand, da einzelne Sensoren keinen Speicher für Node Replication Erkennung belegen müssen.

Man kann nicht sagen, welches Protokoll allgemein das Beste ist. Vielmehr muss beim Einsatz die gewünschte Sicherheit und die Größe des Netzwerks in Betracht gezogen werden, da vor Allem bei dezentralisierten Sicherheitsmechanismen der Aufwand stark von der Anzahl der teilnehmenden Knoten abhängt.

Was sich jedoch zeigt ist, dass die Sicherheit einen sehr engen Zusammenhang zu Kommunikations- und Speicheraufwand hat. Je ressourcenintensiver ein Protokoll ist, desto sicherer scheint es auch zu sein. Das in Abschnitt 5 vorgestellte Protokoll erkauft sich ein hohes Maß an Sicherheit allerdings durch häufigeres Auftreten von Falscherkennungen legitim eingesetzten Sensoren.

7. REFERENZEN

[1] Parno, B., Perrig, A. und Gligor, V. 1993. Distributed Detection of Node Replication Attacks in Sensor Networks

[2] Bekara, C., Laurent-Maknavicius, M. Defending Against Nodes Replication Attacks on Wireless Sensor Networks

[3] Kirchner s., Geschichte des Geburtstagsparadoxon, https://groups.google.com/group/de.sci.mathematik/msg/6ffd85fbcb15647a?hl=de&&pli=1, 2005

[4] Weisstein, E. W. "Sylvester's Four-Point Problem.", http://mathworld.wolfram.com/SylvestersFour-PointProblem.html

doi: 10.2313/NET-2011-07-1_13

ISBN 3-937201-23-8
DOI 10.2313/NET-2011-07-1

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)