Network Architectures
And Services
NET 2010-09-1

SN
SS 2010

Proceeding of the Seminar
Sensor nodes - Operation, Network and Application (SN)
Summer Semester 2010

Munich, Germany, 13./14.07.2010

Editors

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

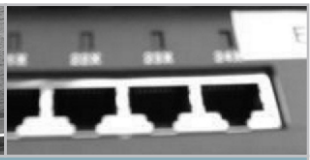
Organisation

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 2010-09-1

SN SS 2010

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

München, Germany, 13./14.07.2010

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 2010
Seminar: Sensorknoten – Betrieb, Netze und Anwendungen

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 2010
Proceeding zum Seminar „Sensorknoten – Betrieb, Netze und Anwendungen“
München, Germany, 13./14.07.2010
Georg Carle, Corinna Schmitt, Alexander Klein,Uwe Baumgarten, Christoph Söllner
ISBN: 3-937201-16-5

ISSN: 1862-7803 (print)
ISSN: 1862-7811 (electronic)
Lehrstuhl Netzarchitekturen und Netzdienste (I8) NET 2010-09-1
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 2010 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 das Thema Wireless Sensor Networks und Vergleich von Knotentechnologien
- Betriebssysteme für Sensorknoten im vergleich
- Sensorknoten – Sicherheit auf Schicht 2
- Sensornetzwerke: Integration in IPv6-Netzwerke und Sicherheit auf Schicht 3
- Sicherheit durch Hardwareeinbindung – secFleck
- Vereinfachtes Schlüsselmanagement in WSNs durch Vertrauensbildung
- Mechanismen der zufallsbedingten Schlüsselvorverteilung in Sensornetzwerken
- Schlüsselmanagement durch vorzeitige Gruppenbildung in WSNs
- Routing in Sensornetzen – Angriffe auf ausgewählte Protokolle und Lösungsansätze
- Routing in Sensornetzen II
- Medium Access Control (MAC) in WSNs
- Vergleich des Energieverbrauches der Protokolle Z-Wave und ZigBee
- Akustische (Unterwasser-) Kommunikation

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, September 2010


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 2010.

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 Wireless Sensor Networks and Comparison of different Technologies
- Comparison of Operating Systems in WSNs
- Sensor Nodes – Security on Layer 2
- Sensor Networks: Integration into IPv6 based networks and security on layer 3
- Security by Using Hardware - secFleck
- Lightweight Key Management in Wireless Sensor Networks by Leveraging Initial Trust
- Mechanisms for pre-distributed Key Management in WSNs
- Key Management in Wireless Sensor Networks focusing on pre-distribution by group deployment
- Routing in Sensor Networks – Attacks on selected Protocols and defence Strategies
- Routing in Sensor Networks II
- Medium Access Control (MAC) in Wireless Sensor Networks
- Comparison of Energy Requirements of Z-Wave und ZigBee
- Acoustic (underwater) Communications

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, September 2010

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/ss10/seminare/

Inhaltsverzeichnis

Session 1: Grundlagen und Anwendungen

Session 2: Sicherheit

Session 3: Protokollanalysen

Einführung in das Thema Wireless Sensor Networks und Vergleich von Knotentechnologien

Benjamin Wiesmüller
Betreuer: Christoph Söllner, Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: benny.w@mytum.de

KURZFASSUNG

Diese Arbeit soll eine Einführung in das weitläufige Themengebiet der Wireless Sensor Networks (WSN) bieten. Der Schwerpunkt liegt hierbei auf einem Überblick über den Aufbau von Sensorknoten, insbesondere einem Vergleich von Anschlusssystemen für die Peripherie. Es folgt auch eine kurze Einführung in das Betriebssystem TinyOS, das häufig auf Sensorknoten verwendet wird. Schließlich wird ein Projekt zur Überwachung eines Vulkans mit WSN vorgestellt, als Beispiel für eine Anwendung dieser.

Schlüsselworte

Wireless Sensor Networks, Sensornetzwerke, Sensorknoten

1. EINLEITUNG

Durch Weiterentwicklungen in der Elektronik wurden Prozessoren und Sensoren immer kleiner, billiger und energiesparender. Dadurch wurde es möglich, kleine Sensorknoten (Abbildung 1) zu fertigen. Diese vereinen auf wenig Platz die Möglichkeit, Messungen und Berechnungen selbstständig über längere Zeit durchzuführen und die Ergebnisse per Funk weiterzuleiten. Das führte zur Entwicklung von großen Sensornetzwerken bei denen mehrere solche Knoten zusammen vernetzt werden.
Ein solches Netz kann nahe um ein zu beobachtendes Phänomen verteilt werden, sich selbstständig vernetzen und vor Ort Messungen durchführen. Denkbar wäre eine großflächige Verteilung in schwer zugänglichem Terrain oder ein mobiler Einsatz des Netzes z.B. beim Krisenmanagement bei großen Katastrophen.
Wireless Sensor Networks sind verschiedenen einschränkenden Faktoren unterworfen. Dadurch ergeben sich besondere Herausforderungen für das Software- und Hardwaredesign. Sensorknoten, die zur Messung der Umwelt eingesetzt werden, müssen dauerhaft mit wenig oder keiner Wartung auskommen und den Umwelteinflüssen widerstehen. Ein sehr wichtiger Faktor ist dabei der Energieverbrauch. Zur Versorgung verfügt ein Sensorknoten meist nur über eine Batterie. Trotz dieses begrenzten Energievorrats soll das Netzwerk lange funktionstüchtig bleiben. Zu bemerken ist hier, dass das Senden von Daten um ein Vielfaches mehr Energie benötigt als die Datenverarbeitung im Prozessor [25].
Zu Beginn der Arbeit wird in Kapitel 2 auf den allgemeinen Aufbau von Sensorknoten eingegangen. Detaillierter wird dort auf die Verbindung der Sensoren mit dem Mikrocontroller eingegangen. Hier erfolgt ein Vergleich verschiedener gängiger Bussyteme.
Danach folgt in Kapitel 3 eine Einführung in das Betriebssystem TinyOS. Dieses wird immer öfter auf Sensorknoten eingesetzt und vereinfacht den Zugriff auf die heterogene Hardware der verschiedenen Sensorknoten und erleichtert allgemein die Softwareentwicklung.
In Kapitel 4 wird ein Projekt vorgestellt, bei dem ein drahtloses Sensornetzwerk eingesetzt wurde, um die Aktivität eines Vulkans zu überwachen. Dort werden die im Projekt verwendete Hardware und die erzielten Ergebnisse der Betreiber vorgestellt.

Abbildung 1: Beispiel für Sensorknoten: MICAz von Crossbow Technology [6].

2. AUFBAU VON SENSORKNOTEN

Sensorknoten bestehen meist aus einem Mikrocontroller, einem Sende- und Empfangsgerät, der Energieversorgung und den eigentlichen Sensoren.
Der Mikrocontroller vereinigt Prozessor, Speicher und Peripheriefunktionen auf kleinem Raum, innerhalb eines Chips. Ein wichtiges Designkriterium bei Mikrocontrollern ist der Energieverbrauch, so dass diese wesentlich weniger Energie verbrauchen, aber auch weniger Rechenleistung bieten, als Standard Desktop CPUs. Oft ist auch ein Sleep-Modus vorhanden, der die CPU-Clock und andere Teile des Controllers vorübergehend abschaltet um Energie zu sparen. Daten- und Programmspeicher sind meist schon auf dem Chip integriert. Dabei wird meist Flash-Speicher verwendet. Da dieser nicht flüchtig ist, bleibt das Programm auch nach Abschalten des Controllers erhalten.
Zur Stromversorgung verfügen die Sensorknoten meist über Batterien oder Akkus.
Kommunikation erfolgt üblicherweise über Funk in den ISM-

Mote Type Year	*WeC* 1998	*René* 1999	*René 2* 2000	*Dot* 2000	*Mica* 2001	*Mica2Dot* 2002	*Mica 2* 2002	*Telos* 2004
								
Microcontroller								
Type	AT90LS8535		ATmega163		ATmega128			TI MSP430
Program memory (KB)	8		16		128			60
RAM (KB)	0.5		1		4			2
Active Power (mW)	15		15		8		33	3
Sleep Power (μW)	45		45		75		75	6
Wakeup Time (μs)	1000		36		180		180	6
Nonvolatile storage								
Chip	24LC256				AT45DB041B			ST M24M01S
Connection type	I^2C				SPI			I^2C
Size (KB)	32				512			128
Communication								
Radio	TR1000				TR1000	CC1000		CC2420
Data rate (kbps)	10				40	38.4		250
Modulation type	OOK				ASK	FSK		O-QPSK
Receive Power (mW)	9				12	29		38
Transmit Power at 0dBm (mW)	36				36	42		35
Power Consumption								
Minimum Operation (V)	2.7		2.7		2.7			1.8
Total Active Power (mW)	24				27	44	89	41
Programming and Sensor Interface								
Expansion	none	51-pin	51-pin	none	51-pin	19-pin	51-pin	10-pin
Communication	IEEE 1284 (programming) and RS232 (requires additional hardware)							USB
Integrated Sensors	no	no	no	yes	no	no	no	yes

Abbildung 2: Überblick über einige Sensorknoten [13].

Bändern. Diese Frequenzbänder sind lizenzfrei zur Funkübertragung für industrielle, wissenschaftliche und medizinische Anwendungen freigegeben [25].
Einen Überblick über die Entwicklung der Technologien für Sensorknoten bietet Abbildung 2. Gezeigt wird eine kleine Auswahl einiger Sensorknoten von 1998 bis 2004 und die jeweils verwendete Hardware.
Im folgenden werden die seriellen Bussysteme 1-Wire, SPI, I^2C und die serielle Schnittstelle RS–232 mit UART vorgestellt. Diese Systeme werden auf Mikrocontrollern bzw. Sensorknoten häufig eingesetzt um die Sensoren und sonstige Peripherie anzuschließen.

2.1 1-Wire

Hierbei handelt es sich um einen asynchronen seriellen Bus, entwickelt von Dallas Semiconductors (inzwischen Maxim Integrated Products) [7]. Dieses Bussystem kommt dabei mit nur einer Leitung aus, die sowohl zur Datenübertragung, als auch zur Stromversorgung der angeschlossenen Geräte verwendet wird. Die Kommunikation erfolgt dabei bidirektional im halbduplex Betrieb. Das heißt, Nachrichten können in beide Richtungen gesendet werden, aber nicht gleichzeitig. Ein Gerät übernimmt die Rolle eines Masters. Dieser kontrolliert den Bus und initiiert die Kommunikation mit den anderen Geräten, die dann Slaves genannt werden. Bei einem Sensorknoten wäre der Master üblicherweise der Mikrocontroller und die Sensoren die Slaves. Die 1-Wire Geräte verfügen über eine eindeutige unveränderbare Identifikationsnummer, die bei der Produktion festgelegt wird. Über diese werden die Slave-Geräte adressiert und auch der Gerätetyp identifiziert.

2.1.1 *Übertragung einer Nachricht*

Abbildung 3 zeigt den Aufbau einer Nachricht des Masters. Zum Start einer Übertragung sendet der Master ein Reset Signal. Danach befinden sich alle Geräte in einem wohl definiertem Zustand und sind bereit für die Übertragung. Anschließend folgt die *ROM Command Sequence*, die unter anderem zur Auswahl des Zielgerätes dient. Dazu mehr im Abschnitt Adressierung. Danach kommt der *Function Command*, der nun abhängig vom Zielgerät ist und die auszuführende Aktion bestimmt [7].

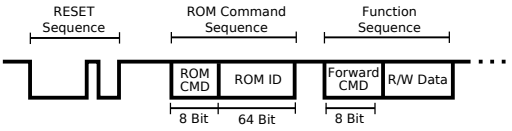


Abbildung 3: Aufbau einer 1-Wire Nachricht.

2.1.2 *Asynchrone Übermittlung der Bits*

Die Datenübertragung erfolgt asynchron. Zur Erkennung der Bits ist ein Zeitslot bestimmter Länge definiert. Erkennt ein Gerät eine abfallende Flanke, tastest es nach einer definierten Zeit das Signal ab. Für eine logische „1“ befindet sich der Pegel der Leitung nur für einen kurzen Zeitabschnitt auf Low und zum Abtastzeitpunkt wieder auf High. Für eine logische „0“ befindet sich der Pegel während des gesamten Zeitslots auf Low. Dadurch, dass jeder Zeitslot durch eine abfallende Flanke eingeleitet wird, synchronisieren sich die Geräte selbst auf jedes Bit. Die Zeitspanne zwischen dem definierten Abtastzeitpunkt nach der fallenden Flanke und dem Ende des Zeitslots, dient als Sicherheitsfenster für die Abtastung um unterschiedlich laufende Uhren in Master und Slave auszugleichen [7].

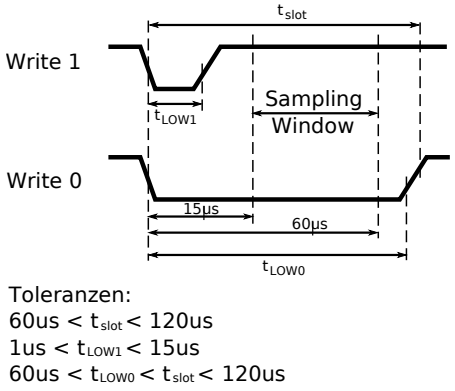


Abbildung 4: Schreiben eines Bits bei 1-wire.

2.1.3 *Adressierung*

Nach dem Reset sendet der Master eine *ROM Command Sequence*, bestehend aus einem 8 Bit *ROM Command* und der 64 Bit langen *ROM ID* des gewünschten Kommunikationspartners. Wie zu Beginn erwähnt wird diese ID schon bei der Fertigstellung des Gerätes fest eingebaut und ist einzigartig.
Die ID besteht aus drei Teilen. Der erste Teil ist der 8 Bit lange *Family Code* und ist spezifisch für den Gerätetyp. Anschließend folgt die 48 Bit lange Seriennummer. Der letzte Teil bildet ein 8 Bit langer CRC.
Von den *ROM Command*s seien hier nur zwei erwähnt. Ein *Match ROM* Befehl ermöglicht es ein bestimmtes Gerät anhand der übertragenen *ROM ID* als Kommunikationspartner auszuwählen. Der *Search ROM* Befehl startet ein Suchverfahren, bei dem alle angeschlossenen Geräte und deren IDs ermittelt werden [7].

2.1.4 *Stromversorgung*

Wenn nicht gesendet wird, befindet sich die Leitung auf dem High Pegel und lädt dabei einen Kondensator in den angeschlossenen Geräten auf. Befindet sich die Leitung auf Low versorgen sich die Geräte durch den im Kondensator gespeicherten Strom. Für Geräte mit geringem Energiebedarf, wie beispielsweise Temperatursensoren, kann dieses Verfahren zur Stromversorgung schon ausreichen [7].

2.1.5 *Erreichbare Übertragungsgeschwindigkeit*

Im regulären Modus sind Übertragungsraten von 15,4kbit/s möglich. Es wurde auch ein *Overdrive* Modus spezifiziert, bei dem Raten von bis zu 126kbit/s möglich sind. Im *Overdrive* Modus sind die Abtastfenster für die einzelnen Bits entsprechend kürzer [7].

2.2 SPI

SPI steht für Serial Peripheral Interface [2]. Ursprünglich wurde SPI von Motorla entwickelt. Eine ähnliche Spezifikation unter dem Namen MicroWire stammt von National Semiconductor [4]. Da es für SPI keine offizielle Spezifikation gibt, ist es nötig die Datenblätter der jeweiligen Geräte genau zu Rate zu ziehen. So sind auch verschiedene Modi für die Datenübertragung üblich und es werden verschieden breite Bitshift-Register verwendet. Bei SPI erfolgt die Datenübertragung seriell und wird durch ein Taktsignal synchronisiert. Dabei werden erneut zwischen Master- und Slave-Geräten unterschieden. Es werden vier Anschlüsse benötigt, zwei davon zur Datenübertragung und zwei zur Steuerung der Übertragung.

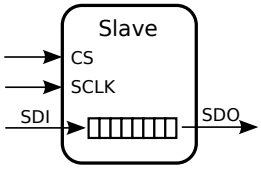


Abbildung 5: SPI-Slave.

2.2.1 *Datenübertragung und Geräteauswahl*

Der Master liefert den Takt für die synchrone Datenübertragung über die SCKL-Leitung (serial clock) [3]. Das Slave-Gerät mit dem er kommunizieren möchte wird über CS (Chip Select) ausgewählt. Über die SDI-Leitung werden die eigentlichen Daten von Master zu Slave übertragen (Input). Über SDO können Daten zurück an den Master gesendet werden werden (Output). Über SDO können auch mehrere Slave-Geräte hintereinander geschaltet werden. Dies wird im nächsten Abschnitt näher beschrieben. Dadurch dass für Input und Output extra Leitungen verwendet werden ist ein vollduplex Betrieb möglich. Das serielle Senden und Empfangen der Daten erfolgt dabei über ein Bitshift-Register, in dem die Daten gepuffert werden. Die einzelnen Bits werden entsprechend dem Takt aus/in dieses Register geschrieben und können dort gelesen/geschrieben werden. Anschließend können die Daten im Mikrocontroller wieder parallel verarbeitet werden.

2.2.2 *Kaskadierte Slaves*

Wie schon erwähnt können die Slave–Geräte kaskadiert werden oder auch direkt mit eigener CS-Leitung angesprochen werden [3]. Die kaskadierte Konfiguration ist in Abbildung 6 dargestellt. Der Output eines Slaves ist direkt mit dem Input des nächsten verbunden. Für eine sinnvolle Steuerung der Slaves müssen diese ein solches Vorgehen unterstützen und die gesendeten Daten teilweise oder ganz – je nach verwendetem Protokoll – zum nächsten Slave weiterleiten.

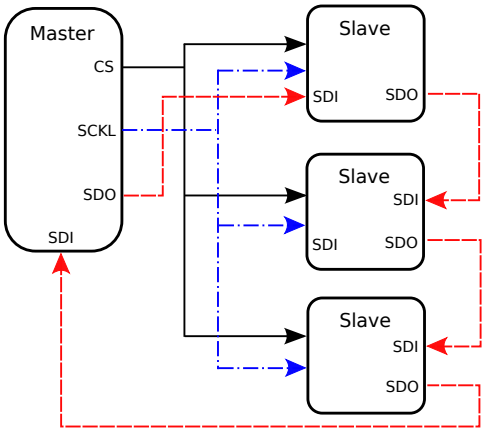


Abbildung 6: SPI mit kaskadierten Slaves.

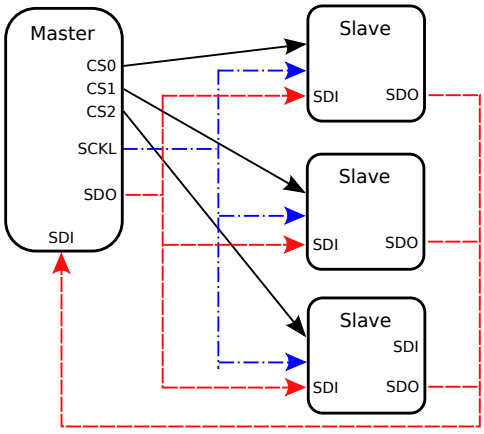


Abbildung 7: SPI mit seperaten CS-Leitungen.

2.2.3 *Einzeln angesprochene Slaves*

Bei dieser Konfiguration — gezeigt in Abbildung 7 — wird der Input durch separate CS–Leitungen auf alle Slave–Geräte gelegt. Ebenso werden die Output-Leitungen zusammengefasst und zurück geführt. Der Host kann hier den gewünschten Kommunikationspartner durch Aktivieren des zugehörigen CS auswählen. Natürlich sind auch Kombinationen aus kaskadierten und einzeln angesprochenen Slaves möglich.

2.2.4 *Erreichbare Übertragungsgeschwindigkeit*

Als erreichbare Übertragungsgeschwindigkeiten, für beispielsweise die M68HC11E Mikrocontrollerfamilie [2], wird zwischen Master und Slave unterschieden. Bei 3MHz Taktfrequenz werden für einen Master 1,5Mbit/s und für einen Slave 3Mbit/s angegeben.

2.3 I²C

Bei I²C (Inter-Integrated Circuit) handelt es sich um ein serielles Bussystem. Es wurde ursprünglich von Philips Semiconductors erfunden [5]. Es verwendet zwei Kabel, *Serial Data* (SDA) und *Serial Clock* (SCL). Jedes Gerät wird durch eine eindeutige Adresse identifiziert und kann als Empfänger oder Sender fungieren (oder beides). Gleichzeitiges Senden und Empfangen ist dabei aber nicht möglich. Die angeschlossenen Geräte werden in Master und Slave eingeteilt, wobei auch mehrere Master–Geräte an einem Bus direkt unterstützt werden. Abbildung 8 zeigt schematisch mehrere mit I²C zusammengeschlossene Geräte.

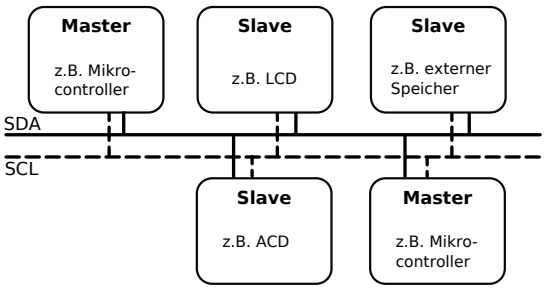


Abbildung 8: Schematischer Zusammenschluss von Master– und Slave–Geräten mit I²C.

2.3.1 *Start einer Übertragung*

Zur Kommunikation werden zwei besondere Signale definiert. Das Start- und das Stoppbit. Diese geben den Beginn und das Ende einer Übertragung bekannt und werden von einem Master erzeugt. Nach einem Startbit ist die Leitung belegt und nach einem Stoppbit wieder frei. Die Master beobachten den Bus und initiieren entsprechend keine Kommunikation bei belegtem Bus. Für den Fall, dass zwei Master fast gleichzeitig zu Senden beginnen, ist ein Arbitrierungsverfahren festgelegt, das im nächsten Abschnitt vorgestellt wird. Beginnt ein Master seine Übertragung, startet er auch mit der Generierung des Taktes zur synchronen Übertragung der folgenden gesendeten/empfangenen Bytes. Bei freiem Bus bleibt die SCL auf hohem Pegel. Durch die dominante Eigenschaft einer logischen Null, was einem niedrigen Pegel entspricht, wäre sonst kein Senden möglich. Diese Eigenschaft wird im nächsten Abschnitt näher erklärt [5].

2.3.2 *Arbitrierung*

Da bei I²C die Möglichkeit besteht, mehrere Master an einem Bus anzuschließen, könnte es dazu kommen, dass zwei solche Master Geräte gleichzeitig versuchen eine Kommunikation zu starten. Deshalb muss es ein Verfahren geben, das den Zugriff auf den Bus regelt. Dies wird allgemein Arbitrierung genannt. Bei I²C funktioniert die Arbitrierung wie im Folgenden beschrieben. Die Datenleitung SDA ist so konzipiert, dass eine logische „0“ dominiert. Das bedeutet wenn zwei Master Daten auf den Bus legen werden alle „1“en von eventuellen „0“en „überschrieben“. Ein Master beobachtet während dem Senden das entstehende Signal auf der Leitung. Stellt er fest, dass das Signal nicht seinem gesendeten entspricht, also eine gesendete „1“ einer „0“ entspricht, weiß er, dass ein anderer Master ebenfalls sendet und beendet seinen Versuch. Der Master, der die Arbitrierung „gewonnen“ hat, merkt davon nichts, da sein Signal unverfälscht übertragen wird und sendet ungestört weiter.
Zur Erklärung, wie die dominante Eigenschaft der logischen „0“ erreicht wird, dient Abbildung 9. Wie in der Abbildung gezeigt, ist der Bus über einen Pullup-Widerstand ständig mit der Versorgungsspannung +VDD verbunden. Um eine logische „0“ zu Senden, wird an Data Out eine Verbindung zur Masse hergestellt, die den ganzen Bus auf Masse „zieht“. Da der Widerstand zur Masse gegen Null geht und die Versorgungsspannung mit einem Widerstand am Bus angeschlossen ist, fließt der gesamte Strom zur Masse und der Bus befindet sich auf einem niedrigen Pegel – auch wenn mehrere Slaves versuchen eine „1“ zu senden.
Über die abgebildete Leitung Data In, kann das Slave–Gerät den Pegel auf dem Bus lesen und so, auch während eigener Übertragung, feststellen was aktuell gesendet wird.
Durch die Eigenschaft der dominanten „0“, kann auch das Taktsignal zweier Master, die gleichzeitig senden, angeglichen werden, so dass die Taktgenerierung nicht gestört wird [5].

2.3.3 *Datenübertragung und Adressierung*

Nach dem Start–Bit beginnt ein Master mit dem Senden der Adresse des Slaves, mit dem er kommunizieren möchte. Diese Adresse ist 7 Bit lang. Es gibt auch eine reservierte Adresse für einen Broadcast an alle angeschlossenen Geräte. Nach der Adresse folgt ein zusätzliches achtes Bit, dass die gewünschte Senderichtung bekannt gibt. Auf diese Weise kann der Master festlegen, ob er in dieser Übertragung Daten an

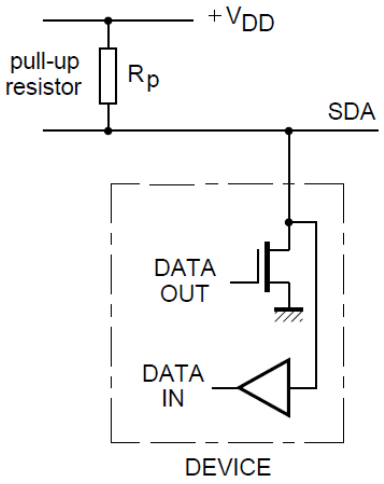


Abbildung 9: Verbindung eines Geräts an den I²C Bus [5].

den Slave senden will, oder von diesem Daten empfangen. Anschließend werden die Nutzdaten übermittelt. Es können beliebig viele Bytes übertragen werden, bis der Master mit einem Stop–Bit, das Ende der Übertragung bestimmt. Jedes Byte muss vom Empfänger mit einem Acknowledgment–Bit bestätigt werden. Durch ein Ausbleiben dieser ACKs können defekte Salves identifiziert werden.
Die Adressen der Geräte bestehen oft aus einem festen Teil und einem programmierbaren Teil. Wie viele Bits dabei programmierbar sind, hängt von der Anzahl der PINs ab, die das Gerät zur Programmierung zur Verfügung stellt. Wenn mehrere Geräte gleichen Typs an einem I²C-Bus angeschloßen sind teilen diese sich den festen Teil ihrer Adresse. Wenn dann drei PINs und damit Bits zur Programmierung zur Verfügung stehen, können maximal $2^3 = 8$ Geräte dieses Typs angeschlossen werden [5].

2.3.4 Clock Stretching

Eine Besonderheit von I²C ist das Clock Stretching. Dabei kann ein Gerät während einer Übertragung die SCL-Leitung auf niedrigem Pegel halten und so die weitere Übertragung von Daten verzögern. Dies ist z.B. nützlich, wenn ein Gerät die Daten nicht schnell genug verarbeitet. Ein Gerät, das beispielsweise gerade einen hoch priorisierten Interrupt bearbeitet und nicht antworten kann, könnte so die Kommunikation verzögern [5].

2.3.5 Spezifikation verschiedener Modi und mögliche Geschwindigkeiten

Im Standard Mode unterstützt I²C 7-Bit Adressen und Geschwindigkeiten von bis zu 100kbit/s. Es gibt verschiedene erweiterte Spezifikationen mit Modi für höhere Geschwindigkeiten oder größerem Adressraum. Beispielsweise den *Fast Mode* mit bis zu 400kbit/s, den *Fast Mode Plus (Fm+)* mit bis zu 1Mbit/s und dem *High Speed Mode* mit bis zu 3.4 Mbit/s. Diese Modi werden allerdings nicht von allen Geräten unterstützt [5].

2.4 UART

Die Bezeichnung UART steht für *Universal Asynchronous Receiver/Transmitter*. Sie dienen der Umwandlung von parallelen Daten in eine serielle Form zur Übertragung an ein anderes Gerät. UARTs werden meist zusammen mit dem Kommunikationsstandard RS–232 verwendet, der die nötigen Anschlüsse und die Signalkodierung festlegt. Durch RS–232 ist ein vollduplex Betrieb vorgesehen. Im Bereich der PCs wurde UART weitgehend von USB verdrängt. Allerdings verfügen Mikrocontroller meist immer noch über einen oder mehrere UART, der zur Kommunikation mit einem Host–PC, zur Programmierung oder zur Kommunikation mit anderen Mikrocontrollern dient. Oft werden auch USARTs verwendet die zusätzlich einen synchronen Übertragungsmodus unterstützen. Hier wird jedoch nur die asynchrone Variante vorgestellt.

2.4.1 UART Rahmen

Zur Umwandlung der parallelen Daten wird ein Bitshift-Register verwendet, dass die zu senden Bits einzeln weitergibt bzw. beim Empfang einzeln speichert und dann als Byte an den Host-Prozessor weiter gibt. Dabei sind auch zusätzliche Empfangs- oder Sendepuffer möglich. Bei UART werden die zu sendenden Daten in einen Rahmen mit Startbit, Stoppbit und Paritätsbit verpackt.Eine schematische Darstellung zeigt Abbildung 10.

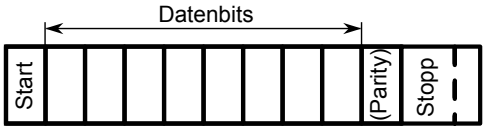


Abbildung 10: UART Rahmen mit 8 Datenbits.

Das Startbit ist dabei ein spezielles Signal um den Beginn einer Übertragung zu kennzeichnen und dem Empfänger die Synchronisation zu ermöglichen. Entsprechend kennzeichnet das Stoppbit das Ende des Rahmens. Das Paritätsbit ist entweder „gerade“, „ungerade“ oder kann ganz weggelassen werden. Durch ein gerades Paritätsbit wird die Gesamtanzahl der „1“en in den Datenbits gerade, durch das ungerade Paritätsbit entsprechend ungerade. Dies dient einer einfachen Fehlererkennung. Erkennt ein Empfänger das Startbit, beginnt er das Signal regelmäßig abzutasten um die einzelnen Bitwerte festzustellen. Die Abtastung erfolgt dabei jeweils ungefähr in der Mitte des Zeitraums, der für ein Bit festgelegt wurde. Dazu müssen Sende- und Abtastrate gleich eingestellt sein und dürfen im Laufe eines Rahmen nicht zu weit auseinander laufen. Zur Übertragung müssen zwei Geräte somit auf die selbe Übertragungsrate und Anzahl von übertragenen Datenbits eingestellt sein. Zusätzlich müssen beide auch auf die Verwendung des selben Typs von Paritätsbit und auf die selbe Art des Stoppbits eingestellt werden. Beim Stoppbit wird zwischen der Länge unterschieden, die ein Bit, ein–einhalb Bit oder zwei Bit betragen kann.

2.4.3 UART mit Adressierung: RS–485

Es existiert auch eine adressierte Version des UART. Auf Mikrocontrollern von Atmel [19] befinden sich beispielsweise USARTs, die RS–485 unterstützen. Dort können mehrere Geräte an den selben Kabeln angeschlossen werden und so

Tabelle 1: Vergleich serieller Busse

Bus	Typ	Anzahl Kabel (ohne Masse)	Adressierung	*2.4.2 Datenrate*	Sonstiges
1-Wire	asynchron, halbduplex	1	Adresse (Gerätetyp/Seriennummer in Paket-Header)	15,4kbit/s bis 126kbit/s	Stromversorgung über Datenleitung
I^2C	synchron halbduplex	2	7 Bit Adresse Nur wenige Geräte gleichen Typs	100kbit/s bis 4Mbit/s	Schnelle Betriebsmodi selten unterstützt Direkte Multi–Master Unterstützung
SPI	synchron, vollduplex	Slave: 4 Master: 4–X	Extra Kabel zur Auswahl	Master: 1,5Mbit/s Slave: 3Mbit/s	
U(S)ART RS–232	a-/synchron, vollduplex	spezieller Stecker (RS–232)	Nur zwei Geräte	bis 500kbit/s	
U(S)ART RS–485	a-/synchron, vollduplex	spezieller Stecker (RS–485)	Extra Protokoll	bis 2 MBit/s	Protokoll regelt Adressierung und Arbitrierung

ein serieller Bus geschaffen werden [20]. Üblicherweise wird dabei ein Gerät als Master und die restlichen als Slave konfiguriert. RS–485 selbst definiert nicht das zu verwendende Protokoll. Dieses muss den Zugriff auf den Bus und die Adressierung regeln.

2.5 Vergleich der Technologien

Bei allen hier vorgestellten Bussystemen handelte es sich um serielle Busse. Dort ist die Verkabelung im Vergleich zu parallelen Bussystemen einfacher. Dank hoher Frequenzen bei der Datenübertragung spielt auch der Geschwindigkeitsvorteil von parallelen Systemen keine so große Rolle und ist für die Kommunikation mit Sensoren meist nicht erforderlich. Einen Vergleich der verschiedenen Busse zeigt Tabelle 1. SPI kommt mit einem Kabel weniger als angegeben aus, wenn nur ein Master und Slave verwendet wird und der CS des Slaves permanent auf High gesetzt wird.

3. EINFÜHRUNG IN TINYOS

TinyOS [12] ist ein Betriebssystem, das speziell für Sensorknoten entworfen wurde. Es bietet dazu ein ereignisgesteuertes Ausführungsmodell, das Nebenläufigkeit unterstützt. Eine Anwendung wird erstellt, in dem verschiedene Komponenten miteinander verbunden werden. Die Komponenten sind dabei entweder vom Benutzer erstellt oder stammen aus dem von TinyOS bereitgestellten Katalog. Hardware Ressourcen werden ebenfalls als Komponenten dargestellt. TinyOS wurde in der Programmiersprache NesC geschrieben, die einen Dialekt von C darstellt. NesC biete Sprachkonstrukte, die direkt den modularen Aufbau aus Komponenten und die Kommunikation zwischen diesen unterstützt.

TinyOS ist dabei kein Betriebssystem im traditionellen Sinne. Es ist ein Framework zur Programmierung eingebetteter Systeme und eine Sammlung von Komponenten, die es ermöglichen ein anwendungsspezifisches Betriebssystem in einer Anwendung zu integrieren.

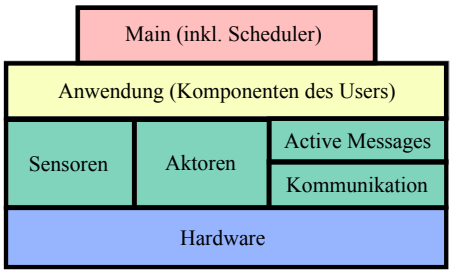


Abbildung 11: Stark vereinfachte Darstellung einer TinyOS Anwendung.

3.1 Komponentenmodell

Wie zu Beginn erwähnt besteht ein TinyOS Programm aus einer Verknüpfung von Komponenten [9]. Jede dieser Komponenten stellt eine eigene Berechnungseinheit, mit einer oder mehreren Schnittstellen, dar. Die Anwendung wird dabei durch eine Wiring Specification beschrieben, die angibt wie die einzelnen verwendeten Komponenten miteinander verbunden sind.
TinyOS abstrahiert alle Hardware Ressourcen als Komponenten. Dabei bietet TinyOS eine große Sammlung an fertigen Komponenten, die einem Anwendungsentwickler zur Verfügung stehen. Darunter befinden sich Abstraktionen für verschiedene Sensoren, Single-Hop Networking, Ad-Hoc Routing, Power Management, Timer und nicht flüchtige Speicher. Der Nachteil vieler kleiner Software Komponenten wird durch den Compiler ausgeglichen, der das entstehende Programm als ganzes analysiert und großen Gebrauch von Inlining macht.

Bei den Interfaces der Komponenten werden solchen unterschieden, die die Komponente anbietet und solche, die von ihr benützt werden. Abbildung 12 zeigt eine vereinfachte Darstellung der TimerM Komponente aus TinyOS, die die Interfaces StdControl und Timer anbietet und HWClock verwendet. Pfeile nach unten repräsentieren dabei Comman-

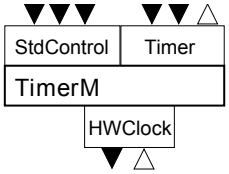


Abbildung 12: Spezifikation und Abbildung der TimerM Komponente [9].

ds, Pfeile nach oben Events.
Im Zusammenhang mit den Komponenten sind drei wichtige Abstraktionen wichtig: Commands, Events und Tasks. Commands und Events dienen dabei der Kommunikation zwischen den Komponenten, während Tasks nebenläufige Berechnungen innerhalb einer Komponente darstellen.
Durch Commands und Events wird der Aufruf eines angebotenen Services einer Komponente in zwei Phasen geteilt. Der Command wird dazu verwendet den Service anzustoßen, beispielsweise das Auslesen eines Sensorwertes, während ein Event die Fertigstellung eines solchen Services signalisiert. Der Command terminiert dabei sofort. Die Events können auch asynchron auftreten, beispielsweise durch Interrupts.
Commands und Event Handler können dabei anfallende teure Berechnungen als Tasks an den TinyOS Scheduler abgeben. Komponenten bleiben so reaktionsfähig, da Commands und Event Handler sofort terminieren. Auf den Scheduler und das Ausführungsmodell wird später näher eingegangen.

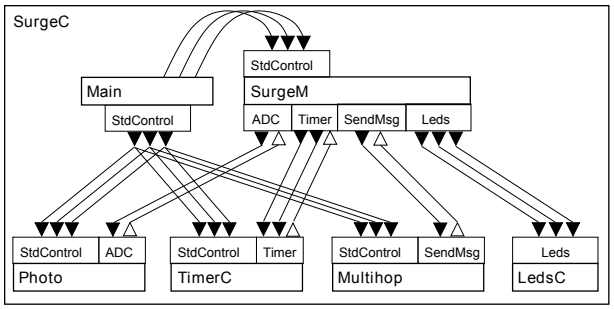


Abbildung 13: Top-level Konfiguration für Surge Anwendung [9].

Eine beispielhafte Vernetzung für eine komplette TinyOS Anwendung ist dabei in Abbildung 13 gezeigt. SurgeC ist ein einfaches Programm, das periodisch (TimerC) Werte eines Lichtsensors (Photo) ausliest und diese zurück an eine Basisstation schickt, wobei Multi-Hop Routing (Multihop) verwendet wird.

3.2 Ausführungsmodell

Sensorknoten reagieren typischerweise auf in ihrer Umgebung auftretende Ereignisse. Diese ereigniszentrierte Architektur erfordert einen feinen Grad an Nebenläufigkeit. Um dies zu unterstützen werden in TinyOS die sogenannten Tasks verwendet [9]. Diese repräsentieren fortlaufende Berechnungen und Interrupt-Handler. Dabei laufen sie einmalig, bis zu ihrer Terminierung. Tasks werden dabei an den Scheduler übergeben, der die Ausführung der Tasks regelt. Der Scheduler verwendet eine nicht preemptive FIFO Strategie. Da die Tasks nicht preemptiv sind, sind sie zueinander atomar. Allerdings gilt das nicht für Interrupt-Handler und von den Tasks angestoßene Commands und Events.
Es ist auch möglich, den Standard Scheduler durch eine eigene Implementierung zu ersetzen.
Um Race Conditions zu vermeiden, kann man kritischen Abschnitte in Tasks umwandeln oder Atomic Sections verwenden. Letztere werden in der aktuellen Implementierung von TinyOS umgesetzt, indem einfach die Interrupts für diesen Abschnitt deaktiviert werden. Außerdem wird sicher gestellt, dass ein solcher Abschnitt keine Schleifen enthält. Der NesC Compiler überprüft dabei zur Compile-Zeit ob Race Conditions auftreten. Dadurch muss man sich bei der Erstellung der Anwendungskonfiguration wenig Gedanken über Nebenläufigkeit machen und muss nicht beachten welche Komponenten Nebenläufigkeit einführen und wie diese untereinander verbunden sind. Dabei sei allerdings erwähnt, dass der Compiler leider nicht alle Race Conditions entdecken kann [9].

3.3 Netzwerkmodell

Ein wichtiger Teil von TinyOS und jedem WSN ist die Netzwerkarchitektur. Die Kommunikation wird von TinyOS durch *Active Messages*[10] abstrahiert [9]. Dabei handelt es sich um kleine 36 Byte Pakete, die mit einer 1 Byte ID versehen sind. Bei Erhalt einer Nachricht wird die Nachricht mit Hilfe eines Events an einen oder mehrere Handler weitergereicht. Diese Handler melden sich zuvor dafür an, Nachrichten diesen Typs zu erhalten. Diese Registrierung erfolgt durch die zuvor genannte Verknüpfung der Schnittstellen der Komponenten. Durch dieses Vorgehen ähnelt die Kommunikation im Netzwerk dem von TinyOS verwendeten Mechanismus der Commands und Events.
Active Messages ist ein unzuverlässiges, Single-Hop Datagram Protokoll und bietet ein einheitliches Kommunikationsinterface, sowohl für das Radio, als auch die eingebauten Seriellen Ports eines Sensorknotens. Höhere Protokolle, die Multi-Hop Kommunikation oder weitere Features anbieten, können auf den Active Messages aufsetzen. Wie erwähnt sind auch hier schon einige Komponenten in TinyOS verfügbar.

3.4 TOSSIM

TOSSIM ist ein Simulator für TinyOS Sensornetze. Mit Hilfe eines Simulators stehen weitere Debugging Möglichkeiten zur Verfügung, um das Design eines Sensornetzwerkes weiter zu untersuchen. So können Fehler gefunden werden, die in der realen Welt schwer nachzuvollziehen sind oder alternative Algorithmen ausprobiert werden.
TOSSIM versucht dabei möglichst einfach und effizient zu arbeiten und trotzdem eine Vielfalt an Interaktionen im Netzwerk abbilden zu können. Prinzipiell wird dabei versucht möglichst treu das Verhalten eines TinyOS Sensorknotens wiederzugeben. So wird die Netzwerkkommunikation auf dem Bit-Level simuliert [17].
TOSSIM simuliert diskrete Ereignisse [16]. Interrupts, Tasks, etc. werden als Ereignisse in einer Warteschlange dargestellt, die nacheinander abgearbeitet werden. Hier sieht man auch schon eine der Vereinfachungen von TOSSIM gegenüber der Realität. Alle Ereignisse werden sofort bearbeitet, Tasks besitzen keine Ausführungszeit und Interrupts treten nie während laufenden Tasks auf [17].
Um die Simulationsergebnisse übersichtlich darzustellen exis-

tieren verschiedene GUIs für TOSSIM [17].
Als Simulator werden in TOSSIM natürlich einige Vereinfachungen vorgenommen, so dass es keinen Ersatz für Tests in der realen Welt darstellt.

3.5 Fazit zu TinyOS

Inzwischen wird TinyOS in vielen WSN Projekten eingesetzt. Es selbst wird ständig weiter entwickelt – die zum Zeitpunkt dieser Arbeit neueste Version 2.1.1 wurde am 6. April 2010 veröffentlicht.
Zum Release der Version im November 2006 unterstützte TinyOS 2.0 die Plattformen EyesIFXv2, Intelmote2, Mica2, Mica2dot, MicaZ, Telosb, Tinynode und Btnode3 [11].

Durch die fertigen Komponenten für den Zugriff auf die Hardware und allgemein der Möglichkeit auf einer höheren Abstraktionsebene zu Programmieren wird durch TinyOS die Entwicklung von WSNs beschleunigt.

4. VORSTELLUNG EINES WSN PROJEKTS

Im Folgenden wird ein Forschungsprojekt vorgestellt, in dem ein drahtloses Sensornetzwerk praktisch angewendet wurde. Dabei wird auf die Ziele des Projekts, die verwendete Hardware und Sensoren und das Fazit der Betreiber eingegangen.

4.1 Projektbeschreibung

Bei diesem Projekt des Harvard Sensor Networks Lab [1], wurde 2004 (und später erneut im Jahre 2007) ein WSN eingesetzt um vulkanische Aktivität am Tungurahua in Ecuador zu messen. Bei der Vulkanforschung werden bisher meist verdrahtete Anordnungen mit mehreren Sensoren, wie akustischen Mikrophonen oder Seismographen, verwendet. In einer typische Anordnung sind mehrere Stationen um den zu beobachtenden Vulkan verteilt. Mit jeder Station sind einige wenige Sensoren in engerem Umkreis verdrahtet – z.B. fünf Stück innerhalb von 100m^2. Die dabei anfallenden großen Datenmengen werden oft in den Stationen digitalisiert und auf Festplatten gespeichert, die manuell abgeholt werden müssen. Die Anzahl und Platzierung der Sensoren an einer Station sind durch Stromverbrauch, Kabellänge und Aufzeichnungsmöglichkeiten der Daten limitiert.
Kleine günstige Sensorknoten könnten über eine größere Fläche verteilt werden. Das würde zu einer besseren Auflösung der Beobachtung führen. Mit einer zusätzlichen leistungsstarken Antenne könnten die Daten an eine entfernte Basisstation übermittelt werden. Ein weitere Vorteil eines WSN ist, dass es leichter neu programmierbar ist, und Wissenschaftler so mit verschiedenen Signalverarbeitungs- und Kompressionsverfahren experimentieren können, um die Qualität der aufgezeichneten Daten zu verbessern.

4.2 Systemarchitektur

Die verteilten Sensorknoten zeichnen mit akustischen Mikrofonen, die bei einem Ausbruch entstehenden niederfrequenten akustischen Wellen (bis zu 50Hz), auf. Die gesammelten Daten werden an einen Aggregationsknoten geschickt, der die Daten über eine drahtlose Verbindung zu einem 9km entfernten Laptop im Vulkanobservatorium weiter leitet. Zur Zeitsynchronisation der Netzwerks ist zusätzlich ein GPS-Empfangsknoten vorhanden. Der Test lief kontinuierlich über 54 Stunden [1].

4.3 Verwendete Hardware

Die Sensorknoten basieren auf einer Mica2-Plattform, bestehend aus einem 7,4MHz ATmega128L Prozessor, 128KB Speicher für den Code, 4KB Speicher für die Daten und einem Chipcon CC1000 Radiosender, der mit einer Frequenz von 433MHz sendet und eine Datenrate von ca. 34Kbps erreicht. Auf den Knoten läuft TinyOS. Als Wetterschutz diente ein einfacher Plastikbehälter in dem die Hardware untergebracht war. Zur Stromversorgung verfügt jeder Knoten über zwei AA Batterien. Der Aggregationsknoten verfügt über eine 12V Autobatterie zum Betreiben der Sendeantenne [1].

4.4 Fazit des Projekts

Die Anordnung lieferte zufriedenstellende Ergebnisse. Die Aufzeichnungen stimmten weitgehend mit denen einer naheliegenden verdrahteten Station überein. Die Verlustraten der Pakete lagen zwischen 1,5% und 5%. Ein dauerhafter Betrieb über längere Zeiträume wäre auf diese Art nicht möglich gewesen. Das kontinuierliche Senden der Daten benötigte zu viel Energie. Deshalb gibt es bei diesem Projekt auch Bestrebungen den Energieverbrauch zu verbessern, in dem ein Teil der Rechenarbeit und Interpretation der Daten auf die Knoten verlagert wird. Da Rechenzeit im Prozessor wesentlich weniger Strom verbraucht als das Senden von Daten, kann durch eine intelligente Auswahl der zu sendenden Daten Leistung gespart werden. Dazu wurde ein verbessertes System vorgeschlagen. Dort würden ein Knoten, sobald er ein „interessantes“ Ereignis beobachten, dies an seine Nachbarn mitteilen. Stimmen mehrere Knoten in dieser Meinung überein, senden sie ihre Daten weiter zum Aggregationsknoten. Insbesondere in einem größeren Netzwerk, bei dem nicht jeder Knoten direkte Verbindung zum Aggregationsknoten hat, soll ein solches Vorgehen für Einsparungen sorgen [1].

5. FAZIT & AUSBLICK

Diese Arbeit konnte nur eine kleine Einführung in das Gebiet der Wireless Sensor Networks bieten. Es wurden verwendete Hardware Technologien der Sensorknoten vorgestellt. Dabei wurde ein Überblick über die gängigsten seriellen Bussysteme geboten um einzuschätzen wann welches am einfachste und am besten geeignet ist und zu zeigen wie diese funktionieren. Wichtig bei der Auswahl ist hier auch vor allem welche Bussysteme und Modi der gewählte Mikrocontroller und die gewünschten Sensoren überhaupt unterstützen. Der Abschnitt über TinyOS zeigte wie mit Hilfe dieses Betriebssystems bzw. Frameworks Sensorknoten auf höherer Abstraktionsebene programmiert werden können. So muss bei der Entwicklung nicht mehr auf die Details der Hardware eingegangen werden und es fällt leichter das WSN als ganzes zu designen und debuggen. Zum Schluss wurde ein Projekt zur Vulkanüberwachung mit WSN gezeigt. Dieses Projekt diente als Beispiel, wie die Möglichkeiten eines WSN genutzt werden können und wo Vorteile gegenüber vorhandener Systeme liegen können.
Im Gebiet der Wireless Sensor Networks hat sich noch viel weiterer Forschungsbedarf gezeigt, um das Potential dieser auszuschöpfen. Im folgenden werden zu den genannten Punkten weiterführende Arbeiten referenziert. Ein wichtiger und interessanter Punkt, der hier nicht näher gezeigt wurde, ist z.B. die Thematik des Power Managements. Der Energieverbrauch der Sensorknoten spielt für den Wartungsaufwand und die Langlebigkeit eines WSN eine große Rolle, so dass

spezielle *Energy Aware* Algorithmen entwickelt werden, die auf geringen Energieverbrauch optimiert sind. Sehr wichtig ist hier die Verteilung der Datenlast im Netzwerk und die Fähigkeit zum ad-hoc Routing [21]. Eine weitere Optimierungsmöglichkeit, ist die *Datenaggregation* [23]. Dabei wird versucht, durch möglichst intelligentes Zusammenfassen und Verarbeiten der Daten auf den Sensorknoten die zu übertragene Datenlast zu senken. Eine andere Methode die Lebenszeit eines Sensorknoten zu erhöhen ist das *Energy Harvesting* [22], bei dem aus verschiedenen Quellen Energie gewonnen wird, um die Batterien zu laden. Beispielsweise können Sonnenlicht, Wind und Vibrationen in Energie umgewandelt werden. Als letztes sei hier die Sicherheitsthematik im Netzwerk erwähnt. Diese spielt für viele Anwendungen eine wichtige Rolle und sorgt für Forschungsbedarf [24].

6. LITERATUR

[1] Geoffrey Werner-Allen, Jeff Johnson, Mario Ruiz, Jonathan Lees, and Matt Welsh: *Monitoring Volcanic Eruptions with a Wireless Sensor Network*, In Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN 05), 2005

[2] freescale semiconductor, *M68HC11E Microcontroller Family - Data Sheet*, Chapter 8: Serial Peripheral Interface, Rev. 5.1, 07/2005, `http://www.freescale.com/files/microcontrollers/doc/data_sheet/M68HC11E.pdf`

[3] MCT Paul & Scherer Mikrocomputertechnik GmbH, *SPI - Serial Peripheral Interface*, `http://www.mct.de/faq/spi.html`

[4] National Semiconductor, *MICROWIRETM Serial Interface*, Application Note 452, January 1992, `http://www.national.com/an/AN/AN-452.pdf`

[5] Philips Semiconductors, *THE I^2C-BUS SPECIFICATION*, Version 2.1, January 2000, `http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf`

[6] *Crossbow Technology*, `http://www.xbow.com/`

[7] Maxim Integrated Products, *1-Wire Devices*, `http://www.maxim-ic.com/products/1-wire/`

[8] *Atendo GmbH & Co KG*, `http://www.atendo.de/`

[9] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer and David Culler: *TinyOS: An Operating System for Sensor Networks*, In Ambient Intelligence (2005), pp. 115-148.

[10] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser: *a mechanism for integrating communication and computation*, In Proceedings of the 19th Annual International Symposium on Computer Architecture, pages 256-266, May 1992.

[11] *TinyOS 2.0 Overview*, `http://www.tinyos.net/tinyos-2.x/doc/html/overview.html`

[12] *TinyOS Community Forum*, `http://www.tinyos.net/`

[13] Joseph Polastre, Robert Szewczyk, Cory Sharp, David Culler: *The Mote Revolution: Low Power Wireless Sensor Network Devices*, In HotChips 2004, `http://www.hotchips.org/archives/hc16/3_Tue/7_HC16_Sess7_Pres2_bw.pdf`

[14] *Willow Technologies Limited*, `http://www.willow.co.uk/TelosB_Datasheet.pdf`

[15] *SEAMONSTER: A terrestrial and marine geoscience research program at the University of Alaska Southeast*, `http://www.robfatland.net/seamonster/`

[16] Philip Levis, Nelson Lee, Matt Welsh, David Culler: *TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications*, `http://www.cs.berkeley.edu/~pal/pubs/tossim-sensys03.pdf`

[17] Philip Levis, Nelson Lee: *TOSSIM: A Simulator for TinyOS Networks*, `http://www.cs.berkeley.edu/~pal/pubs/nido.pdf`

[18] *TinyOS Tutorial – Lesson 5: Simulating TinyOS Applications in TOSSIM*, `http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson5.html`

[19] Atmel Corporation, `http://www.atmel.com/`

[20] Atmel 32-bit AVR Mictrocontrollers Application Note: *AVR32100: Using the AVR32 USART*, `www.atmel.com/dyn/resources/prod_documents/doc32006.pdf`, Rev. 32006A-AVR-04/06

[21] Mike Woo, Suresh Singh, C. S. Raghavendra: *Power-Aware Routing in Mobile Ad Hoc Networks*, 1998

[22] M Rahimi, H Shah, G Sukhatme, J Heidemann, D Estrin: *Studying the Feasibility of Energy Harvesting in a Mobile Sensor Network*, in ICRA´03 (2003)

[23] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong: *TAG: a Tiny AGgregation service for ad-hoc sensor networks*, In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (2002)

[24] Adrian Perrig, John Stankovic, David Wagner, Caren Rosenblatt: *Security In Wireless Sensor Networks*, 2004

[25] O.K. Boyinbode, K.G. Akintola: *Transforming Nigeria Health Care System through Wireless Sensor Networks* In Pacific Journal of Science and Technology, Volume 10. Number 1. May 2009 (Spring), pages 312–313

Betriebssysteme für Sensorknoten im Vergleich

Florian Walter
Betreuer: Christoph Söllner
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: walterfl@in.tum.de

KURZFASSUNG

Die besonderen Anforderungen an Sensorknoten haben zur Entstehung von Betriebssystemen speziell für diesen Einsatzbereich geführt. Die Wahl des richtigen Systems für den eigenen Zweck bedarf einer fundierten Entscheidungsgrundlage. Vor diesem Hintergrund werden in der vorliegenden Abhandlung drei Betriebssysteme mit stark voneinander abweichenden Design-Konzepten vorgestellt: TinyOS, MANTIS OS und Contiki. Für jedes dieser Systeme wird ein kompakter Architekturüberblick einschließlich des resultierenden Laufzeitmodells präsentiert. Ebenso werden für den produktiven Einsatz relevante Aspekte wie Portierbarkeit auf neue Plattformen, Energieeffizienz, Toolchain im Entwicklungsprozess sowie Verfügbarkeit und Qualitiät von Dokumentation abgedeckt. Das klassische Konzept des Multithreadings und das Event-Konzept werden anhand von MANTIS OS und TinyOS verglichen.

Schlüsselworte

Sensorknoten, Betriebssystem, TinyOS, MANTIS OS, Contiki, Portabilität, Energieeffizienz, Dokumentation, Toolchain, event-driven, Multithreading, Protothread

1. EINLEITUNG

Die typischen Aufgaben eines Sensorknotens sind die Erfassung, Verarbeitung und Weiterleitung von Daten. Rechenleistung und Speicherkapazität der zugrundeliegenden Hardware sind beschränkt. Zudem werden die Geräte oft an schwer zugänglichen Orten fernab jeglicher Infrastruktur eingesetzt und über Batterien [25] und Solarzellen [36] mit Strom versorgt. Fortschritte in der Halbleiterfertigung werden sich daher eher in sinkender Energieaufnahme statt steigender Leistung der Sensorhardware niederschlagen [25, 28]. Eigens für Sensorknoten entwickelte Betriebssysteme sind speziell an diese knappen Hardware-Ressourcen angepasst. Sie nehmen dem Programmierer die Verwaltung von Hardware sowie Programmausführung ab und erleichtern dadurch die Anwendungsentwicklung.

In dieser Arbeit werden mit TinyOS, MANTIS OS und Contiki drei Betriebssysteme für Sensorknoten vorgestellt. Trotz ähnlicher Ziele verfolgen diese drei Systeme unterschiedliche Realisierungskonzepte. Wichtig für die erfolgreiche Entwicklung einer Anwendung ist die Wahl des für diese Anwendung am besten geeigneten Betriebssystems. Ebenso entscheidend ist die Unterstützung des Anwendungsentwicklers mit Dokumentation und ausgereiften Entwicklungswerkzeugen. Wissen über beide dieser Aspekte ist Voraussetzung für die begründete Entscheidung für oder gegen eines der genannten Systeme. Die vorliegende Abhandlung will dazu eine fundierte Basis liefern und stellt alle drei Betriebssysteme vor. Zu Beginn wird dargelegt, weshalb der Einsatz eines Betriebssystems auf Sensorknoten sinnvoll ist. In Abschnitt 3 wird TinyOS behandelt. Danach wird MANTIS OS genauer betrachtet. Während Tiny OS strikt den „event-driven“-Ansatz verfolgt, gilt MANTIS OS als Vertreter der Multithreading-Betriebssysteme. Beide Architekturparadigmen werden in Abschnitt 5 kurz verglichen. Das Betriebssystem Contiki wird in Abschnitt 6 als Kompromiss dieser Paradigmen vorgestellt. Die vorliegende Abhandlung schließt mit einer Zusammenfassung der Ergebnisse.

2. BETRIEBSSYSTEME AUF SENSORKNOTEN

2.1 Aufgaben eines Betriebssystems

Die Hauptaufgaben eines Betriebssystems sind die Verwaltung der Hardware und die Steuerung der Programmausführung [12]. Gerade eine effiziente Steuerung der Programmausführung ist auf Sensorknoten von großer Bedeutung. Denn um Energie zu sparen soll der Knoten einerseits möglichst früh in den Ruhezustand versetzt werden. Andererseits ist eine schnelle Reaktion auf eingehende Datenpakte notwendig. Nicht zeitkritische Berechnungen müssen dazu gegebenenfalls unterbrochen und später wieder aufgenommen werden.

2.2 Laufzeitsystem

Ein Betriebssystem setzt direkt auf der Hardware auf und stellt dem Benutzer eine komfortable Schnittstelle zur Verfügung, über die er die Hardware steuern kann. Diese Schnittstelle wird durch das Laufzeitsystem [12] realisiert. Anwendungen greifen nur indirekt über das Laufzeitsystem auf die Hardware zu. Ist also das Betriebssystem – und damit auch dessen Laufzeitsystem – für mehrere Hardware-Plattformen verfügbar, können bestehende Anwendungen beim Wechsel auf eine neue Plattform direkt übernommen werden.

3. TINY OS

TinyOS ist das älteste der hier betrachteten Betriebssysteme. Erste Versionen entstanden im Jahr 1999 an der University of California, Berkeley. Heute ist TinyOS ein Projekt der TinyOS Alliance und steht unter der BSD-Lizenz. Die aktuelle Version TinyOS 2 unterstützt eine Vielzahl von Prozessoren und Plattformen [44]. Im nächsten Abschnitt werden vier davon vorgestellt.

3.1 Unterstützte Hardware

TinyOS läuft auf den Mikrocontrollern der ATmega128-Serie [1] der Firma ATMEL und auf denen der MSP430-Serie [10] von Texas Instruments. Auch leistungsfähigere Prozessoren wie die aus der PXA27x-Serie von Marvell [7] werden unterstützt [40]. Zudem ist das Betriebssystem bereits an eine Vielzahl am Markt verfügbarer Sensorknoten angepasst [40]. Ein Beispiel ist der Sensorknoten MICAz [8] der Firma MEMSIC. Im Wesentlichen handelt es sich dabei um einen Mikrocontroller aus der ATmega128-Serie, an den ein 2,4 GHz Funkmodul des Typs Chipcon CC2420 angebunden ist [41].

3.2 Systemarchitektur

TinyOS ist kein Betriebssystem im klassischen Sinn. Vielmehr ist es eine Sammlung vieler einzelner Module, von denen jedes einen bestimmten Dienst anbietet. Im Kontext von TinyOS werden diese funktionalen Basiseinheiten *Komponenten*[1] genannt. Sie bilden die Basis für Architektur und Laufzeitmodell des Betriebssystems. Abbildung 1 gibt einen schematischen Überblick über den allgemeinen Aufbau einer Komponente.

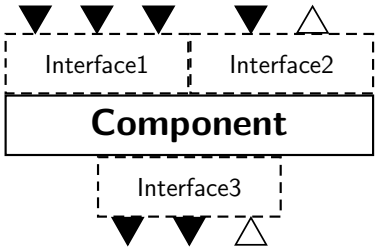


Abbildung 1: TinyOS-Komponente

3.2.1 Interfaces

Wie bereits erwähnt, kapselt jede Komponente bestimmte Dienste. Die Schnittstellen, über die diese zur Verfügung gestellt werden, heißen *Interfaces*. Im Beispiel aus Abbildung 1 bietet die Komponente `Component` Dienste über die Interfaces `Interface1` und `Interface2` an. Jedem Interface einer Komponente wird ein *Interface-Type* zugeordnet, der die Schnittstelle eindeutig beschreibt. Mit Interfaces werden nicht nur von Komponenten angebotene Dienste spezifiziert sondern auch Abhängigkeiten von anderen Diensten modelliert. Zur besseren Unterscheidung werden Interfaces letzteren Typs im Folgenden als *ausgehende Interfaces* bezeichnet. Im Beispiel aus Abbildung 1 benötigt die Komponente `Component` Zugriff auf eine andere Komponente, die `Interface3` implementiert.

3.2.2 Komponenten & Wiring

Wie das Betriebssystem selbst werden auch Anwendungen als Komponente oder als Sammlung von Komponenten entwickelt. Es gibt zwei Klassen von Komponenten: *Module*[2] bilden die kleinsten Einheiten im System und stellen eine Implementierung für die gekapselte Funktionalität bereit. *Konfigurationen*[3] verknüpfen mehrere Komponenten zu einer größeren Komponente. Ein lauffähiges Betriebssystem entsteht durch Verbindung aller notwendigen Komponenten über deren Interfaces. Der Bauplan, der angibt, wie die einzelnen Teile des Systems zu verknüpfen sind, heißt *wiring specification*. Dank dieses Bauplans werden immer nur unbedingt notwendige Komponenten in das Betriebssystem eingebunden.

3.3 Laufzeitmodell

Eng mit der Architektur verbunden ist das Laufzeitmodell von TinyOS. Wie bereits zu Beginn erwähnt, ist TinyOS kein typisches Multithreading-Betriebssystem, sondern arbeitet auf Basis von Events[4]. Als weitere Besonderheit wird die dynamische Allokation von Speicher während der Laufzeit nicht unterstützt [28].

3.3.1 Commands, Events und Tasks

Die Ausführung des Betriebssystems wird durch drei Grundprimitive gesteuert, die jeder Komponente zur Verfügung stehen und die im Folgenden genauer beschrieben werden.

- Über ausgehende Interfaces können *Commands* an andere Komponenten abgesetzt werden.
- *Events* werden von der Komponente gesendet, die einen Dienst zur Verfügung stellt, und werden über ausgehende Interfaces empfangen.
- Für Berechnungen, die ausschließlich innerhalb einer Komponente stattfinden, kennt TinyOS das Konzept der *Tasks*. Tasks repräsentieren Rechenaufträge für den Prozessor.

Letztlich handelt es sich bei Commands und Events um Funktionaufrufe [28]. Der Interface-Type enthält die Header der Command- und Event-Funktionen. Commands müssen von der Komponente implementiert werden, die einen Dienst anbietet und Events von der, die den Dienst nutzt.

3.3.2 Laufzeitverhalten

Die gerade beschriebenen Grundprimitive bestimmen maßgeblich das Laufzeitverhalten von TinyOS, welches nun dargestellt wird. Dienstanfragen, wie zum Beispiel die Erfassung eines Messwertes oder die Ausführung einer bestimmten Berechnung, werden als Command an die dafür bestimmte Komponente übermittelt. Anfragen, die wenig Rechenaufwand erfordern, können direkt in der Handler-Funktion des Commands bearbeitet werden. Umfangreichere Rechenaufgaben werden in Tasks ausgelagert. Jeder Task wird in eine zentrale Warteschlange eingereiht. Ein nicht-präemptiver Scheduler arbeitet diese Warteschlange gemäß einer First-Come-First-Served-Semantik ab. Eine Priorisierung von Tasks ist somit nicht möglich. Nur Commands, Events und Interrupt-Handler können einen aktiven Task unterbrechen. Mit Events werden von Tasks berechnete Ergebnisse übertragen oder Ereignisse wie Interrupts signalisiert.

[1] engl.: components
[2] engl.: modules
[3] engl.: configurations
[4] Dieses Architekturmodell heißt *event-driven*.

3.4 Beispiel

Die gerade beschriebenen Mechanismen werden nun anhand einer einfachen Anwendung [23] illustriert. Ziel ist die Entwicklung einer Diebstahlsicherung für Sensorknoten. Abbildung 2 zeigt alle beteiligten Komponenten und die Verknüpfung ihrer Interfaces, also die *wiring specification*. Commands eines Interfaces werden als schwarze und Events als weiße Pfeile dargestellt. `MainC` ist eine Konfiguration, die neben

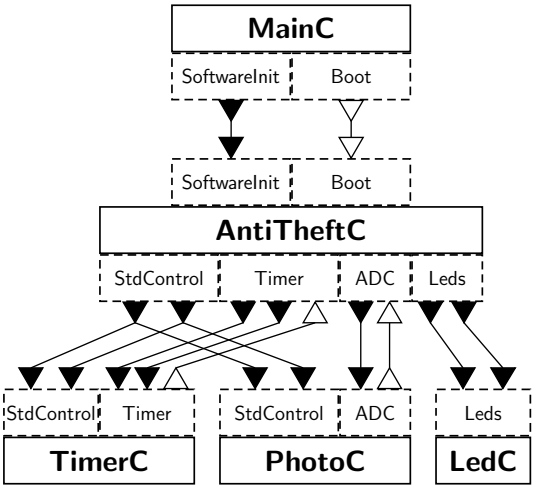


Abbildung 2: Diebstahlsicherung

dem Scheduler für den Systemstart notwendige Komponenten enthält [37]. Die Interfaces `SoftwareInit` und `Boot` steuern die Initialisierung der Komponente `AntiTheftC`. Diese wiederum steuert die angebundenen Sensoren über das Interface `StdControl`. Erfahrungsgemäß stecken Diebe gestohlene Sensorknoten in ihre Tasche. Der Messwert des durch `PhotoC` gesteuerten Helligkeitssensors fällt dann unter einen bestimmtem Schwellwert. Die Helligkeit wird daher regelmäßig gemessen. Die Fälligkeit der nächsten Messung wird durch `TimerC` signalisiert. `AntiTheftC` fragt daraufhin über das Interface `ADC` die aktuellen Sensordaten ab und vergleicht sie mit dem Schwellwert. Wurde ein Diebstahl erkannt, wird über das Interface `Leds` eine LED am Sensorknoten aktiviert, die den lichtscheuen Dieb abschrecken soll. Andernfalls wird keine Aktion ausgeführt oder die LED bei erfolgreicher Abschreckung gegebenenfalls deaktiviert.

3.5 Portabilität

Bei der großen Anzahl verfügbarer Sensorplattformen ist die Portabilität des Betriebssystems und der dafür geschriebenen Anwendungen von großer Bedeutung. Durch Kapselung von Diensten in Komponenten [28] können TinyOS selbst und für TinyOS entwickelte Anwendungen leicht an andere Plattformen angepasst werden. Denn bei der Portierung einer TinyOS-Anwendung müssen nur Komponenten mit direktem Hardwarezugriff neu implementiert werden. Solange die neue Implementierung die gleichen Interfaces wie die ursprüngliche Komponente bereitstellt, kann die Anwendung einfach durch Austausch der entsprechenden Komponenten an die neue Hardware-Plattform angepasst werden. Die durch das Interface-Konzept erwirkte strikte Trennung von Dienst und der Implementierung des Dienstes erlaubt zudem eine unkomplizierte Integration neuer Hardwarefunktionen in die bestehende Anwendung. Beispielsweise kann eine alte Komponente, die einen Kryptographiealgorithmus in Software realisiert, durch eine neue Komponente mit gleichen Interfaces ersetzt werden, die auf die Kryptographieeinheit der neuen Sensorhardware zurückgreift [28].

3.6 Energieeffizienz

TinyOS sieht mehrere Mechanismen für das Energie-Management vor. Mikrocontroller und Peripheriegeräte werden getrennt behandelt [39].

3.6.1 Mikrocontroller

Der Betriebszustand des Mikrocontrollers wird durch den Scheduler verwaltet. Sobald sich keine Tasks in der Warteschlange befinden, wird der Mikrocontroller in einen der verfügbaren Energiesparmodi versetzt. Die Berechnung und Aktivierung des am besten geeigneten Sparmodus erfolgt in der plattformspezifischen Komponente `McuSleepC` [38]. Optional können beliebige Komponenten die Wahl tieferer Sparmodi durch das Setzen einer unteren Schranke unterbinden.

3.6.2 Peripheriegeräte

Im Gegensatz zu Prozessoren kennen Peripheriegeräte meist lediglich die Zustände „An“ und „Aus“. Den Wechsel zwischen diesen Zuständen steuern die Anwendungen selbst[5] über Interfaces wie `StdControl` [39]. Beim Zugriff mehrerer Komponenten auf ein Gerät kann dieses Verfahren zu Konflikten führen. Für solche Szenarios können in TinyOS Richtlinien für die Steuerung des Geräts realisiert werden[6]. Die Komponente `PowerManager` setzt Richtlinien für das Energiemanagement durch. Wird ein Gerät von keiner anderen Komponente benötigt, wird es dem `PowerManger` zugeteilt. Dieser kann dann den Betriebszustand dieses Geräts anpassen. Sobald eine Komponente auf das Gerät zugreift, gibt der `PowerManager` die Kontrolle wieder ab.

3.7 Toolchain

3.7.1 nesC

TinyOS baut auf einer eigenen Programmiersprache namens *nesC* auf. nesC erweitert die Programmiersprache C um spezielle Konstrukte für das Architekturmodell von TinyOS. Allerdings werden weder die dynamische Allokation von Speicher noch Funktionspointer unterstützt [28]. Abbildung 3 zeigt die *wiring specification* der Beispielanwendung `AntiTheftApp` aus Abschnitt 3.4. Die Diebstahlsicherung verknüpft mehrere Komponenten und ist daher eine Konfiguration. Im Block **`configuration`** werden, falls vorhanden, die Interfaces der Konfiguration aufgeführt. Im Beispiel ist dieser Block leer. Im Abschnitt **`implementation`** werden zunächst alle Komponenten der Konfiguration aufgelistet. Anschließend werden die Interfaces dieser Komponenten mittels des Operators `->` verknüpft. Dabei steht auf der linken Seite des Operators die Komponente, die auf die Dienste der Komponente auf der rechten Seite des Operators zugreift. Der nesC-Compiler enthält Mechanismen zur Erkennung von Race Conditions[7] [24]. Zudem wird das Komponenten-Modell zur Programmoptimierung ausgenutzt [28].

[5] explizites Energiemanagement
[6] implizites Energiemanagent
[7] Konkurrierender Zugriff mehrerer Operationen auf die gemeinsam genutzte Variable führt zu unvorhersehbarem Ergebnis.

```
configuration AntiTheftApp {
}

implementation {
  components MainC, AntiTheftC, TimerC,
             PhotoC, LedC;

  MainC.SoftwareInit –>
                AntiTheftC.SoftwareInit;
  MainC.Boot –> AntiTheftC.Boot;
  AntiTheftC.StdControl –> TimerC.StdControl;
  AntiTheftC.Timer –> TimerC.Timer;
  AntiTheftC.StdControl –> PhotoC.StdControl;
  AntiTheftC.ADC –> PhotoC.ADC;
  AntiTheftC.Leds –> LedC.Leds;
}
```

Abbildung 3: Diebstahlsicherung in nesC

3.7.2 Buildsystem

Die Übersetzung des fertigen Codes und die Übertragung auf den Sensorknoten werden durch das GNU Buildsysem unterstützt. Ein mit TinyOS geliefertes Makefile steuert die Übersetzung des Quelltextes für die gewünschte Hardware-Plattform und die anschließende Übertragung der Software auf den Sensorknoten [42]. Noch bessere Unterstützung bei der Entwicklung von Anwendungen für TinyOS versprechen speziell auf dieses Betriebssystem abgestimmte Plugins für die IDE Eclipse. „NESCDT“ [9] erweitert Eclipse um Editierfunktionen für nesC. Das Plugin „Yeti 2“ [15] von der ETH Zürich unterstützt den gesamten Entwicklungsprozess einer TinyOS-Anwendung vom Schreiben des Codes bis hin zur Übertragung der fertigen Anwendung in den Flash-Speicher des Sensorknotens.

3.7.3 Simulator

Ein weiteres wichtiges Glied in der Toolchain von TinyOS ist der Simulator TOSSIM [43]. Mit TOSSIM können Sensornetzwerke mit hunderten von Knoten simuliert werden [27]. Durch Setzen einer Option des nesC-Compilers kann TinyOS für TOSSIM kompiliert werden. Dabei werden Komponenten mit Hardwarezugriff durch Komponenten von TOSSIM ersetzt. Je nach Anforderung können so auch ganze Mikrocontroller simuliert werden. Die eigentliche Simulation erfolgt gemäß einer übergebenen Konfiguration und umfasst auch eine drahtlose Netzwerkanbindung mit anpassbarer Fehlerrate [27].

3.8 Dokumentation

TinyOS und alle zugehörigen Tools sind umfassend dokumentiert. Zentraler Ausgangspunkt jeglicher Recherche ist die Projekt-Homepage [11]. Von dort aus gibt es Zugriff auf ein Dokumentations-Wiki. Eine Reihe von Tutorials ermöglicht den problemlosen Einstieg in TinyOS. Die Themen reichen von der Installation einer geeigneten Entwicklungsumgebung auf Windows, Mac OS X und Linux bis hin zur Nutzung des Simulators TOSSIM. TEPs[8] enthalten umfassende Beschreibungen der APIs von TinyOS. Das Git-Repository des Projekts enthält neben dem Quelltext des Betriebssystems zusätzlich viele Programmbeispiele. Insgesamt ist TinyOS hervorragend dokumentiert. Anwender mit grundlegenden Kenntnissen der Programmierung von Mikrocontrollern finden daher abgesehen vom neuen Programmiermodell einen leichten Einstieg.

[8]TinyOS Enhancement Proposals

4. MANTIS OS

Das nächste in dieser Abhandlung betrachtete Betriebssystem heißt MANTIS OS und ist im Jahr 2003 [14] an der University of Colorado at Boulder entwickelt worden. Die Software steht unter der BSD-Lizenz [32]. MANTIS steht als Akronym für „MultimodAl Networks of In-situ Sensors“.

4.1 Unterstützte Hardware

MANTIS OS unterstützt die Sensorplattformen MICA2, MICAz, und TelosB der Firma MEMSIC [32]. Der Sensorknoten MICA2 basiert ebenso wie der in Abschnitt 3.1 vorgestellte MICAz auf einem ATmega 128-Mikrocontroller. Bis auf das verwendete Funkmodul sind diese beiden Sensorknoten im Wesentlichen identisch. Der TelosB baut auf dem Mikrocontroller MSP430 von Texas Instruments auf. Neben einem Funkmodul und einer USB-Schnittstelle sind bereits mehrere Sensoren zum Messen von Lichteinstrahlung, Temperatur und Luftfeuchtigkeit auf dem Knoten integriert. Im Projektverzeichnis von MANTIS OS befindet sich zusätzlicher Code für die Unterstützung weiterer Hardware wie etwa der PXA27x-Prozessoren von Marvell [31].

4.2 Systemarchitektur

Das Design von MANTIS OS orientiert sich an klassischen UNIX-Betriebssystemen [13] und unterstützt präemptives Multitasking. Jede Anwendung ist daher ein eigener Thread. Freier Hauptspeicher wird als Heap verwaltet. Wie in TinyOS gibt es jedoch keine Möglichkeit zur dynamischen Allokation von Speicher durch den Programmierer. Lediglich das Betriebssystem selbst hat Zugriff auf den Heap [13]. Einen Überblick über die Architektur gibt Abbildung 4.

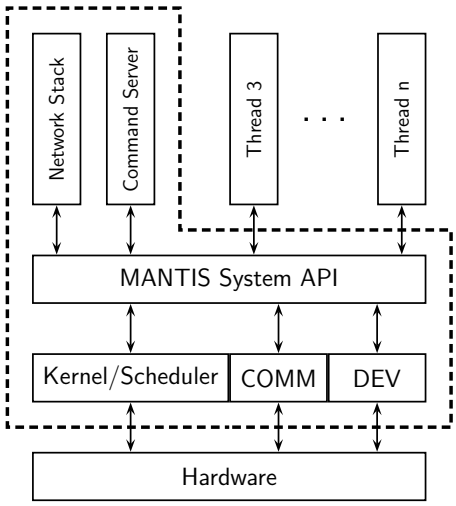


Abbildung 4: Systemarchitektur von MANTIS OS [13]

4.2.1 Kernel & Scheduler

MANTIS OS ist in mehreren Schichten organisiert. Anwendungsprogramme und der Netzwerk-Stack [13] laufen in Form von Threads auf der obersten Systemschicht. Die Kommunikation mit der darunterliegenden Schicht erfolgt über die API des Betriebssystems. Diese Schicht verfügt über direkten Hardwarezugriff und besteht aus Kernel und Treibern. Innerhalb des Kernels verwaltet ein Scheduler die Threads in einer statischen Tabelle [13]. Als Folge der festen Tabellengröße muss die maximale Anzahl gleichzeitiger laufender Threads bereits beim Kompilieren feststehen. Standardmäßig können maximal 10 Threads gleichzeitig gestartet sein [31]. In der Tabelle werden nur grundlegende Verwaltungsinformationen wie etwa die Priorität eines Threads gespeichert. Alle weiteren Kontextinformationen werden beim Threadwechsel auf dem Stack des verdrängten Threads gesichert. Der Stack eines Threads wird zur Laufzeit bei Bedarf auf dem Heap angelegt und nach Beendigung des Threads wieder freigegeben.

4.2.2 Synchronisation von Threads

Der unkontrollierte Zugriff mehrerer Threads auf gemeinsam genutzte Systemressourcen führt zu Race Conditions. MANTIS OS stellt zur Synchronisation solcher konkurrierender Threads Semaphore und Mutexe [13] zur Verfügung. Auf ein Semaphor wartende Threads werden in einer Liste innerhalb dieses Semaphors verwaltet [13]. Gleiches gilt für Mutexe [31].

4.2.3 Gerätetreiber

Gerätetreiber arbeiten wie der Kernel auf der untersten Schicht des Betriebssystems. Asynchrone Ein-/Ausgabe erfolgt über die COMM-Schicht, in der beispielsweise Funk- und USB-Verbindungen verwaltet werden. Diese Schicht verwaltet gemeinsam genutzte Datenpuffer, die in MANTIS OS `comBuf` genannt werden [13]. Anwendungen schreiben zu sendende Daten in einen solchen Puffer. Die COMM-Schicht nimmt diesen Puffer entgegen und übernimmt den eigentlichen Versand. Umgekehrt werden die empfangenen Daten ebenfalls in Puffern an die entsprechende Anwendung übergeben. Synchrone Ein-/Ausgabe wie zum Beispiel das Auslesen von Sensoren wird in der DEV-Schicht abgewickelt. Die API dieser Schicht lehnt sich an den POSIX-Standard an und bietet elementare Funktionen zur einheitlichen Ansteuerung der angebundenen Peripherie [13, 30]. Für jeden Gerätetreiber müssen lediglich vier Funktionen zur Gerätesteuerung sowie zum Lesen und Schreiben von Daten implementiert werden. MANTIS OS unterhält für jedes Gerät eine Tabelle, in der Zeiger auf diese vier Funktionen gespeichert sind. Interrupts werden grundsätzlich von den Gerätetreibern und nicht vom Kernel bearbeitet [13]. Lediglich der Thread-Wechsel durch den Scheduler wird durch einen Hardware-Timer ausgelöst. Software-Interrupts werden nicht unterstützt [13].

4.3 Laufzeitmodell

Jeder Thread in MANTIS OS ist einer von fünf Prioritätsstufen zugeteilt [13] und befindet sich immer in genau einem der folgenden Zustände [31]: Rechnend, rechenbereit, blockiert, schlafend. Der Scheduler verwaltet für jede Prioritätsstufe eine eigene Warteschlange. Threads gleicher Priorität werden in einem Round Robin-Verfahren abgearbeitet [34]. Eine Warteschlange wird nur dann bearbeitet, wenn die Warteschlangen aller höheren Prioritätsstufen leer sind. Die Priorität gibt also nicht an, in welchem Verhältnis die Rechenzeit unter den Threads aufgeteilt wird, sondern welche Threads zuerst vollständig abgearbeitet werden [33]. Gibt es keine rechenbereiten Threads, wird ein spezieller Idle-Thread ausgeführt [13]. Auf dessen Bedeutung wird bei der Diskussion der Energiesparmechanismen von MANTIS OS noch näher eingegangen. Ein Thread wechselt durch Aufruf der Funktion `mos_thread_sleep(uint16_t msecs)` für eine festgelegte Zeit in den Zustand „schlafend“. Dabei wird er aus seiner Warteschlange entfernt und in eine separate Warteschlange für schlafende Threads eingereiht [13]. Nach Ablauf der definierten Zeitspanne kehrt der Thread wieder in den Zustand „rechenbereit“ zurück und wechselt die Warteschlange. Der Zugriff auf ein belegtes Semaphor verläuft ähnlich: Der aufrufende Thread betritt die Warteschlange des Semaphors und geht in den Zustand „blockiert“ über. Bei Freigabe des Semaphors wird der erste Thread in dessen Warteschlange wieder in den Zustand „rechenbereit“ versetzt [31] und kehrt in seine ursprüngliche Warteschlange im Scheduler zurück.

4.4 Portabilität

Der Code von MANTIS OS ist gemäß der drei Teile der untersten Systemschicht unterteilt [31]. Architekturspezifischer Code ist von plattformunabhängigem Code getrennt [31]. Bei der Portierung des Kernels auf eine neue Prozessorarchitektur muss daher nur der architekturspezifische Teil des Codes neu implementiert werden. Jedoch enthält auch der eigentlich plattformunabhängige Teil des Kernels an einigen Stellen plattformspezifischen Code. Die Berücksichtigung dieser Sonderfälle und die Teils stark unterschiedliche Struktur des architekturspezifischen Codes dürfte die Portierung von MANTIS OS im Vergleich zum wesentlich klarer gegliederten TinyOS erschweren. Ein ähnliches Bild ergibt sich bei den bereits vorhandenen Treibern. Beispielsweise befinden sich die UART-Treiber für mehrere Sensorplattformen in einer einzelnen Datei. Erst der Präprozessor des Compilers entscheidet anhand von `ifdef`-Direktiven, welche Teile des Codes kompiliert werden sollen. Neue Treiber können dank der einfachen und standardisierten API jedoch leicht implementiert werden. Insgesamt wird deutlich, dass MANTIS OS nicht über die hohe Flexibilität über TinyOS verfügt. Trotzdem unterstützt die in vielen Fällen durchgesetzte Kapselung von hardwareabhängigem Code die Portierung des Systems auf neue Hardware-Plattformen.

4.5 Energieeffizienz

Der bereits erwähnte Idle-Thread ist maßgeblich für das Energiemanagement von MANTIS OS verantwortlich [13]. Da dieser Thread die niedrigste Priorität besitzt, wird er vom Scheduler nur dann ausgewählt, wenn die Warteschlangen aller anderen Prioritäten leer sind. Das ist dann der Fall, wenn kein Thread läuft oder alle Threads sich in den Zuständen „blockiert“ oder „schlafend“ befinden. Sobald der Idle-Thread läuft, werden die Parameter für den Energiesparmodus gewählt. Der Thread, der den Zustand „schlafend“ zuerst verlassen wird, bestimmt die Dauer. Zusätzlich wird abhängig von den Einstellungen der schlafenden Threads einer von zwei möglichen Energiesparmodi ausgewählt. Das differenzierte Energiemanagement von Mikrocontrollern wie dem ATmega128 [1], der sechs verschiedene Schlafzustän-

de anbietet, wird von MANTIS OS nicht unterstützt. Peripherie wird getrennt vom Prozessor über die Gerätetreiber durch die Anwendungen gesteuert.

4.6 Toolchain

4.6.1 Buildsystem

Wie bereits zu Beginn erwähnt, werden in der Architektur von MANTIS OS viele Konzepte aus UNIX-Systemen aufgegriffen. Das Betriebssystem und die API sind in der Programmiersprache C implementiert [13]. Das Thread-Konzept von MANTIS sowie die synchrone Ein-/Ausgabe über die DEV-Schicht orientieren sich am POSIX-Standard [13]. Dies erlaubt die Verwendung gängiger Entwicklungswerkzeuge. Zum Kompilieren des Codes unterstützt MANTIS OS SCons und das GNU Buildsystem [29].

4.6.2 Virtuelle Sensorknoten

Dank der Anlehnung der API von MANTIS OS an den POSIX-Standard ist zum Testen von Anwendungen kein spezieller Simulator notwendig. Stattdessen kann das gesamte Betriebssystem als UNIX- oder Windows-Anwendung kompiliert werden [13]. Aufrufe von Funktionen der MANTIS API werden werden über eine Zwischenschicht, den *„POSIX Shim Layer“*, in API-Aufrufe des zugrundeliegenden Betriebssystems übersetzt. Ein als UNIX-Anwendung laufendes MANTIS OS ist ein virtueller Sensorknoten, der in beliebige Sensornetzwerke eingebunden werden kann. So werden heterogene Netzwerke ermöglicht, die nicht allein auf reale Sensorknoten beschränkt sind. Abbildung 5 zeigt ein Beispiel für solch ein Netzwerk. In diesem Beispiel verbin-

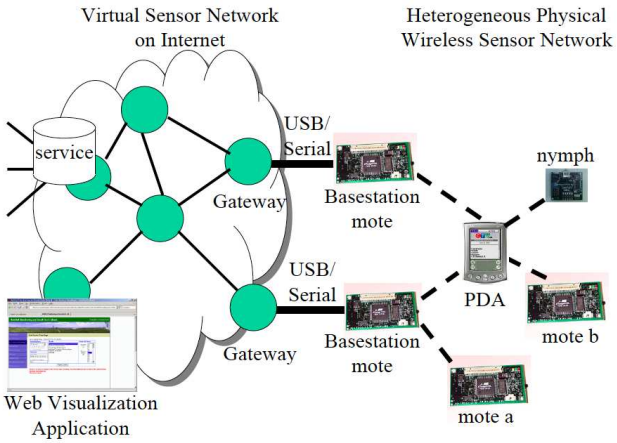


Abbildung 5: Heterogenes Sensornetzwerk mit virtuellen Sensorknoten im Internet [13]

den spezielle Bridge-Knoten das physische Sensornetzwerk über zwei Gateways mit den virtuellen Sensorknoten auf Servern im Internet. Virtuelle Sensorknoten sind nicht auf die API von MANTIS OS beschränkt. Dies ermöglicht die Einbindung von Anwendungen auf dem Host des virtuellen Knotens in das Netzwerk. Ein Nachteil dieser Flexibiltiät ist, dass aufgrund der Übersetzung der API im „Posix Shim Layer“ einige Teile von MANTIS OS wie etwa der Scheduler nicht getestet werden können [13]. Stattdessen muss in solchen Fällen auf Simulatoren wie Avrora [2] zurückgegriffen werden.

4.7 Dokumentation

Einen Großteil der Dokumentation zu MANTIS OS enthält ein ACM-Artikel aus dem Jahr 2005 [13]. Informationen über die Nutzung des Systems und über die Implementierung eigener Anwendungen liefert die Rubrik „Documentation“ auf der Homepage des Projekts [32]. Das SVN-Repository enthält eine genauere Beschreibung der API in Form des kommentierten Source-Codes. Zudem sind dort Beispielprogramme und C-Bibliotheken zu finden. Insgesamt ist die Dokumentation des Systems knapp. Die große Ähnlichkeit zu UNIX und die Verwendung der Programmiersprache C erlauben dennoch einen leichten Einstieg. Insbesondere muss anders als bei TinyOS kein neues Programmierkonzept erlernt werden.

5. EVENTS UND MULTITHREADING IM VERGLEICH

Wie gerade ausführlich erläutert wurde, unterscheidet sich das Event-Konzept von TinyOS deutlich vom Multithreading in MANTIS OS. Obwohl beide Konzepte gleich mächtig sind [26], ist die Entwicklung mit Threads oft leichter, da Programme bei der Verwendung von Events als Zustandsmaschinen programmiert werden müssen [21]. Zudem kann die rein sequentielle Abarbeitung von Tasks in TinyOS zu Problemen führen. Ein Beispiel ist der Empfang von Daten aus dem Netzwerk, die in einem Puffer zwischengespeichert werden [13]. Muss ein Task, der die Daten aus dem Puffer abarbeiten soll, längere Zeit auf einen anderen Task warten, kann der Puffer überlaufen. Ohne präemptives Multithreading lässt sich dieses Problem nur durch Elimination größerer Tasks umgehen. Dies ist jedoch nicht ohne genaues Verständnis der in einer Anwendung eingesetzten Algorithmen möglich und unter Umständen sehr aufwendig [13]. Die sequentielle Programmierung mit Threads ist meist einfacher. Ein genauerer Vergleich von TinyOS und MANTIS OS [16] hat ergeben, dass TinyOS weniger Speicher belegt und weniger Energie verbraucht als MANTIS OS. Andererseits reagiert MANTIS OS dank Multithreading bei umfangreicheren Rechenaufgaben schneller auf Anfragen aus dem Netzwerk als TinyOS.

6. CONTIKI

Contiki ist im Jahr 2003 am Swedish Institute of Computer Science entstanden. Wie die beiden anderen vorgestellten Betriebssysteme steht es unter der BSD-Lizenz. Aktuell ist die Version 2.4 verfügbar [5]. Einige der unterstützen Hardware-Plattformen werden im Folgenden vorgestellt.

6.1 Unterstützte Hardware

Contiki läuft auf den bereits bekannten ATmega 128-Mikrocontrollern und der MSP430-Serie von Texas Instruments. Unterstützt wird neben dem Sensorknoten MICAz von MEMSIC auch die an der Freien Universität Berlin entwickelte Plattform MSB430 [4]. Dieser Sensorknoten basiert auf einem MSP430-Prozessor und verfügt über ein Funkmodul sowie Sensoren für Beschleunigung, Temperatur und Luftfeuchtigkeit. Daten können auf einer SD-Karte gespeichert werden. Contiki wurde auch auf ältere Heimcomputer wie den Apple II oder den Commodore C64 portiert [17].

6.2 Systemarchitektur

Anders als die bisher vorgestellten Betriebssysteme verfügt Contiki über einen Programm-Lader, mit dem Anwendungen zur Laufzeit aus dem Speicher oder über ein Funkmodul geladen werden können [20]. Dies führt, wie Abbildung 6 zeigt, zur Gliederung des Betriebssystems in zwei Teile. Der

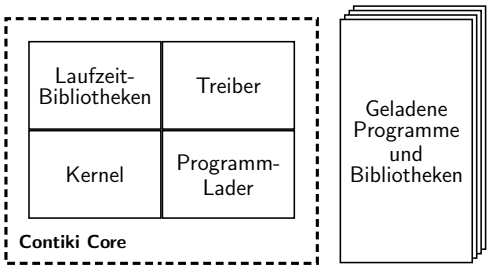


Abbildung 6: Systemarchitektur von Contiki [20]

Core ist ein Basissystem, das alle für den Betrieb wesentlichen Komponenten wie Kernel, Programm-Lader, Bibliotheken und Gerätetreiber enthält. Während des Betriebs können nur die geladenen Programme verändert werden. Änderungen am Core sind im Allgemeinen nicht vorgesehen und nur mit einem speziellen Bootloader möglich [20]. Die konkrete Aufteilung des Systems in Core und ladbare Programme wird beim Kompilieren des Systems entschieden und hängt von der Hardware-Plattform ab [20]. Gerätetreiber werden als Bibliotheken implementiert [20]. APIs für einige Geräteklassen wie etwa Funkmodule erlauben standardisierten Hardware-Zugriff [3]. Als einziges der hier vorgestellten Betriebssysteme unterstützt es die dynamische Allokation von Speicher [3].

6.3 Laufzeitmodell

Im Laufzeitmodell von Contiki werden die beiden Konzepte von MANTIS OS und TinyOS vereint. Einerseits besitzt das Betriebssystem einen Kernel, der Prozesse verwaltet. Andererseits basiert der Kernel auf dem Event-Konzept [20].

6.3.1 Events in Contiki

Contiki-Prozesse sind sogenannte *Protothreads* ohne eigenen Stack [4] und kommunizieren über *synchrone* und *asynchrone* Events mit anderen Prozessen. Eine Warteschlange im Kernel speichert alle eintreffenden asynchronen Events. Die Scheduling-Funktion des Kernels läuft nach Systemstart in einer Endlosschleife [3]. In jedem Durchlauf wird ein Event aus der Schlange entnommen und an den Empfänger-Prozess weitergeleitet. *Synchrone* Events werden ohne Umweg über die Warteschlange direkt an den Empfänger-Prozess zugestellt [20]. Die Abarbeitung eines Events in einem Prozess kann grundsätzlich nur durch Hardware-Interrupts unterbrochen werden [20]. Anders als in TinyOS werden größere Rechenaufträge nicht in Tasks ausgegliedert, sondern direkt im Prozess bearbeitet. Contiki stellt dazu mehrere Primitive zur Verfügung, mit denen die Berechnung gesteuert werden kann. Beispielsweise blockiert `PROCESS_WAIT_EVENT()` den aufrufenden Prozess, bis zum Eintreffen eines Events. Mit `PROCESS_WAIT_UNTIL(c)` wartet der Prozess, bis die Bedingung `c` erfüllt ist. Mit `PROCESS_YIELD()` wird ein Prozess unterbrochen. Im Gegensatz zum Komponentenmodell von TinyOS können Anwendungen mit Prototheads in sequentieller Form programmiert werden. Die meist komplizierte Aufteilung einer Berechnung in Commands, Tasks und Events entfällt [21].

6.3.2 Polling

Neben Events unterstützt Contiki *Polling*. Jeder Protothread kann zusätzlich zum Event-Handler einen Poll-Handler implementieren. Polls sind Events mit hoher Priorität. Sie sind besonders bei der Abarbeitung von Hardware-Interrupts wichtig, da Interrupts-Handler keine Events, sondern nur Polling-Anfragen absetzten dürfen [20]. Nach der Bearbeitung eines asynchronen Events testet der Scheduler ein Polling-Flag. Ist dieses Flag gesetzt, werden die Poll-Handler aller Prozesse aufgerufen, für die eine Polling-Anfrage vorliegt.

6.3.3 Multithreading

Über eine Bibliothek unterstützt Contiki neben Protothreads auch normale Threads mit eigenem Stack. Diese Threads können in einem Prozess erzeugt werden und werden nicht durch den Scheduler des Betriebssystems verwaltet. Stattdessen muss die Ausführung der Threads durch den Prozess selbst gesteuert werden [19].

6.4 Portabilität

Wie bei MANTIS OS ist in Contiki architekturspezifischer Code von plattformunabhängigem Code getrennt [3]. Jedoch ist diese Unterteilung bei Contiki deutlich konsequenter durchgesetzt. Die bei der Portierung anzupassenden Teile des Betriebssystems sind klar definiert: Bootprozess, architekturspezifische Teile des Programm-Laders und der Multithreading-Bibliothek [20]. Auch Treiber müssen auf die neue Plattform angepasst werden. Da Contiki nur für wenige Hardware-Komponenten standardisierte APIs enthält, muss der Entwickler eventuell auch Anwendungen an die Treiber der neuen Plattform anpassen.

6.5 Energieeffizienz

Contiki bietet keinerlei Unterstützung für das Energie-Management von Mikrocontroller und Peripherie. Stattdessen sollen die Anwendungen selbst entsprechende Mechanismen implementieren [20]. Die einzige Unterstützung, die Contiki dem Programmierer bei der Entwicklung von Energiesparmechanismen bietet, ist die Möglichkeit, die Anzahl der Events in der Warteschlange des Schedulers abzufragen [20].

6.6 Toolchain

6.6.1 Buildsystem

Contiki ist wie MANTIS OS in der Programmiersprache C geschrieben. Das Konzept der Protothreads erfordert wie bereits gezeigt spezielle Anweisungen. Anders als bei TinyOS werden die neuen Konstrukte nicht über eine Spracherweiterung von C sondern über Makros für den Präprozessor des Compilers realisiert. Abbildung 7 zeigt das Grundgerüst eines „Hello World“-Prozesses in Contiki. Als Makros realisierte Anweisungen enthalten nur Großbuchstaben. Die Verwendung von C erlaubt den Einsatz gängiger Editoren und Werkzeuge. Die Erstellung eines lauffähigen Systems wird durch das GNU Buildsystem unterstützt.

```
#include "contiki.h"
#include "stdio.h"

PROCESS(hello_world_process,
                "Hello world process");

PROCESS_THREAD(hello_world_process, ev, data)
{
  PROCESS_BEGIN();

  printf("Hello, world\n");

  PROCESS_END();
}
```

Abbildung 7: Aufbau eines Prozesses in Contiki [3]

6.6.2 Simulatoren

Zum Testen von Anwendungen enthält das Projekt-Verzeichnis von Contiki zwei Simulatoren. *MSPsim* [22] ist ein Simulator für Sensorknoten auf Basis des MSP430 von Texas Instruments, der Code für diesen Prozessor nativ ausführen kann. Neben dem Prozessor selbst werden auch Peripheriegeräte des Sensorknotens simuliert. Das Verhalten von Sensornetzen kann mit mit *COOJA* [35] getestet werden. COOJA kann entweder wie MSPsim die Hardware eines Knotens exakt nachbilden oder nur dessen Verhalten simulieren. Beides ist in einem simulierten Netzwerk zur gleichen Zeit möglich. Alle hier vorgestellten Tools sind auch in *Instant Contiki* [6] enthalten. Instant Contiki ist eine komplette Entwicklungsumgebung für Contiki die als Image für den VMWare Player heruntergeladen werden kann.

6.7 Dokumentation

Die gesamte technische Dokumentation des Betriebssystems ist über den Punkt „Documentation“ von der Projekthomepage [5] aus erreichbar. Anleitungen zu Installation und Benutzung sind ebenfalls direkt auf der Homepage verlinkt. Genaue technische Details zu Konzept und Realisierung von Contiki findet man in diesen Quellen jedoch nicht. Insbesondere gibt es keine Artikel, die einen Überblick über den Aufbau des Systems beschreiben. Der einzige vom Autor auffindbare Artikel, der Contiki näher beschreibt [20], ist veraltet. Beispielsweise wird dort noch das verworfene [18] Service-Konzept beschrieben. Insgesamt liegt der Schwerpunkt der Dokumentation auf der praktischen Anwendung von Contiki. Mit dem Quelltext werden viele Beispiele, Bibliotheken und fertige Anwendungen geliefert, die den Einstieg in die Programmierung erleichtern.

7. ZUSAMMENFASSUNG & FAZIT

In dieser Abhandlung wurden drei Betriebssysteme für Sensornetzwerke vorgestellt: TinyOS, Mantis OS und Contiki. Trotz vieler Gemeinsamkeiten setzen TinyOS, MANTIS OS und Conikti unterschiedliche Schwerpunkte.

7.1 TinyOS

TinyOS ist in erster Linie auf geringen Ressourcenverbrauch optimiert. Davon zeugt neben durchdachten Energiesparmechanismen vor allem das Event-Konzept, das die vom Betriebssystem beanspruchte Rechenzeit minimiert. Die aktive Weiterentwicklung des Systems durch die TinyOS Alliance macht TinyOS zusammen mit der Unterstützung einer Vielzahl von Sensorknoten zum attraktivsten System in diesem Vergleich. Die hervorragende Dokumentation macht das Betriebssystem auch für kommerzielle Anwendungen besonders interessant.

7.2 Contiki

Contiki versucht die Vorteile von Multithreading und Events in Protothreads zu vereinen. Damit fällt der Einstieg leichter als bei TinyOS, erfordert aber mehr Aufwand als bei MANTIS OS. Negativ fällt auf, dass keinerlei Maßnahmen zur Reduzierung des Energieverbrauchs implementiert sind. Unter den unterstützten Hardware-Plattformen finden sich nur wenige Sensorknoten. Die Dokumentation ist umfangreich, enthält aber keine Details zur Realisierung des Betriebssystems. Insgesamt nimmt Contiki so nur den zweiten Rang innerhalb dieses Vergleichs ein.

7.3 MANTIS OS

Die Architektur von MANTIS OS realisiert präemptives Multithreading. Dies führt zu einem im Vergleich zu TinyOS geringfügig höheren Ressourcenverbrauch. Vorteile ergeben sich vor allem bei rechenlastigen Anwendungen. Die große Ähnlichkeit zu typischen UNIX-Systemen erlaubt trotz sehr knapp gehaltener Dokumentation einen einfachen Einstieg in die Anwendungsentwicklung. Im Vergleich zum streng modular aufgebauten TinyOS ist MANTIS OS deutlich schlechter portierbar. Es gibt keine nennenswerte Entwicklergemeinschaft und die Weiterentwicklung scheint zu stagnieren. Vor allem deswegen belegt MANTIS OS in diesem Vergleich den letzten Platz.

8. LITERATUR

[1] ATMEL ATmega128A. http://www.atmel.com/dyn/products/product_card_v2.asp?PN=ATmega128A.

[2] Avrora. http://avrora.sourceforge.net/.

[3] Contiki 2.4 Source Code. http://www.sics.se/contiki/download.html.

[4] Contiki Dokumentation. http://www.sics.se/~adam/contiki/docs/.

[5] Contiki Projekt-Homepage. http://www.sics.se/contiki/.

[6] Instant Contiki. http://www.sics.se/contiki/instant-contiki.html.

[7] Marvell PXA-Prozessoren. http://www.marvell.com/products/processors/applications/pxa_family/.

[8] MEMSIC Wireless Modules. http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html.

[9] NESCDT. http://docs.tinyos.net/index.php/NESCDT.

[10] Texas Instruments MSP430. http://www.ti.com/ww/de/msp430.htm.

[11] TinyOS Homepage. http://www.tinyos.net/.

[12] U. Baumgarten and H. Siegert. *Betriebssysteme: Eine Einführung*. Oldenbourg, 6., überarb., aktualis. u. erw. A. edition, Dec. 2006.

[13] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantis OS: An embedded multithreaded

operating system for wireless micro sensor platforms. In *ACM/Kluwer Mobile Networks & Applications (MONET), Special Issue on Wireless Sensor Networks*, page 2005, 2005.

[14] Q. Cao and T. Abdelzaher. The liteos operating system: Towards unix-like abstractions for wireless sensor networks.

[15] Distributed Computing Group, ETH Zürich. Yeti 2. `http://tos-ide.ethz.ch/wiki/index.php`.

[16] C. Duffy, U. Roedig, J. Herbert, and C. Sreenan. A comprehensive experimental comparison of event driven and multi-threaded sensor node operating systems.

[17] A. Dunkels. About Contiki. `http://www.sics.se/contiki/about-contiki.html`.

[18] A. Dunkels. Contiki Changelog. `http://www.sics.se/contiki/changelog.html`.

[19] A. Dunkels. Using Multi-Threading in Contiki. `http://www.sics.se/contiki/developers/using-multi-threading-in-contiki.html`, 2007.

[20] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors, 2004.

[21] A. Dunkels and O. Schmidt. Protothreads - Lightweight, Stackless Threads in C, 2005.

[22] J. Eriksson, A. Dunkels, N. Finne, F. Österlind, and T. Voigt. Poster Abstract: MSPsim – an Extensible Simulator for MSP430-equipped Sensor Boards.

[23] D. Gay. TinyOS 2.0: A wireless sensor network operating system. `http://hinrg.cs.jhu.edu/git/?p=tinyos-2.x.git;a=blob;f=apps/AntiTheft/tutorial-slides.pdf`, 2005.

[24] D. Gay, P. Levis, D. Culler, and E. Brewer. *nesC 1.1 Language Reference Manual*. TinyOS.net, 2003.

[25] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy. The platforms enabling wireless sensor networks. *Communications of the ACM*, 47:41–46, 2004.

[26] H. C. Lauer and R. M. Needham. On the duality of operating system structures. In *Operating Systems Review*, pages 3–19, 1979.

[27] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications, 2003.

[28] 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.

[29] MANTIS Group. Building and running an example application. `http://mantisos.org/index/tiki-read_article.php%3FarticleId=6.html`.

[30] MANTIS Group. MANTIS Device Layer. `http://mantisos.org/index/tiki-index.php%3Fpage=ReferenceDevLayer.html`.

[31] MANTIS Group. MANTIS OS 1.0 Beta Source Code. `http://mantisos.org/index/tiki-read_article.php%3FarticleId=1&page=2.html`.

[32] MANTIS Group. MANTIS OS Projekt Homepage. `http://mantisos.org/`.

[33] MANTIS Group. MANTIS Thread Priorities. `http://mantisos.org/index/tiki-index.php%3Fpage=ReferenceThreadPriority.html`.

[34] MANTIS Group. MANTIS Threads. `http://mantisos.org/index/tiki-index.php%3Fpage=ReferenceThreadPriority.html`.

[35] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Crosslevelsensor network simulation with cooja. InLCN, 2006.

[36] W. K. Seah, Z. A. Eu, and H. P. Tan. Wireless sensor networks powered by ambient energy harvesting (WSN-HEAP)-Survey and challenges. In *1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009. Wireless VITAE 2009*, pages 1–5, 2009.

[37] TinyOS.net. TEP 107. `http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep107.html`.

[38] TinyOS.net. TEP 112. `http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep112.html`.

[39] TinyOS.net. TEP 115. `http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep115.html`.

[40] TinyOS.net. TinyOS Hardware-Unterstützung. `http://docs.tinyos.net/index.php/Platform_Hardware`.

[41] TinyOS.net. TinyOS MICAz-Unterstützung. `http://docs.tinyos.net/index.php/MicaZ`.

[42] TinyOS.net. TinyOS Toolchain. `http://docs.tinyos.net/index.php/TinyOS_Toolchain`.

[43] TinyOS.net. TOSSIM. `http://docs.tinyos.net/index.php/TOSSIM`.

[44] TinyOS.net. TinyOS git-Repository. `http://hinrg.cs.jhu.edu/git/?p=tinyos-2.x.git;a=tree;f=tos/chips`, Stand: 07.06.2010.

Akustische Unterwasserkommunikation

Jun Chen
Betreuer: Christoph Söllner
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: chenjun@in.tum.de

KURZFASSUNG

In unserem heutigen Zeitalter spielen die Kommunikation und der dabei stattfindende Informationsaustausch eine sehr große Rolle. Neben der Kommunikation über elektromagnetische Wellen im Medium Luft und im Vakuum verwendet man akustische Wellen unter Wasser. Im dichten Medium Wasser können sich elektromagnetische Wellen nur schwer ausbreiten, wogegen aber Schallwellen keine so hohe Abschwächung erleiden [17]. Daher sind sie mit ihren Eigenschaften und Reichweite für die Kommunikation unter Wasser sehr von Bedeutung. Neben den Fragen der Modulation und Verwendung gibt es hierbei ebenso Grenzen und Probleme durch Störungen, welche im Verlaufe der Arbeit erläutert werden.

Schlüsselworte

Wasserschall, akustische Datenübertragung, FSK, ASK, PSK, QAM, Unterwassersensoren, Sensorennetze, Tsunami-Warnsystem, D.A.R.T. II

1. EINLEITUNG

Die Kommunikation mit Funkwellen unter Wasser erfordert sehr viel Energie. Ebenso erreicht sie nur geringe Reichweite und Datenrate. Akustische Wellen können sich hingegen in Unterwasserumgebungen besonders gut ausbreiten [17].
Die Bedeutung von Unterwasserkommunikation wurde zuerst im Militärbereich für den Wasserkampf mit U-Booten entdeckt. Heutzutage wird sie jedoch immer mehr für kommerzielle und private Anwendungen interessant [17]. Die Anwendung der akustischen Unterwasserkommunikation zur Fernsteuerung von Unterwasserroboter eröffnete neue Möglichkeiten zum Erforschen von unbekannte Tiefen oder bewahrt Menschen vor unangenehme Unterwasserarbeiten wie z.B. bei Ölplattformen.
Durch die Tsunami-Katastrophe im Dezember 2004 wurde die Nachfrage nach effektiver akustischer Kommunikation für Tsunami-Warnsystemen nochmals erhöht [10].
Diese Arbeit ist nach folgendem Schema gegliedert: Als erstes werden die physikalische Gegebenheiten sowie deren Einschränkungen erläutert. Anschließend wechseln wir unseren Fokus in Richtung Datenübertragung und betrachten den Aspekt der Reichweiten und Frequenzen, sowie die Modulation mit Hilfe des Unterwassermodems. Darauf werden zwei Typen von Sensoren mit deren Vor- und Nachteilen betrachtet, gefolgt von der Applikation in Sensorennetzen. Nach der Betrachtung der Grundlagen diskutieren wir diese in deren Praxiseinsatz in einem Tsunami-Warnsystem. Zum Abschluss folgt die Zusammenfassung in der die wichtigsten Punkte nochmals herausgestellt werden.

2. WASSERSCHALL

Die akustische Kommunikation unter Wasser wird mit sogenanntem Wasserschall realisiert. Wasserschall ist Verdichtung und Verdünnung von Wasser, die zu Dichteveränderung führt, weshalb Schall sich allgemein ohne ein Medium nicht ausbreiten kann [18]. Wasserschall wird von einem Sender erzeugt und von einem Empfänger aufgrund dieser Veränderung erkannt.

2.1 Schallgeschwindigkeit unter Wasser

Die Ausbreitung des Wasserschalls hat einen Distanz-Zeitfaktor, die Geschwindigkeit. Sie ist unter Wasser durch Tiefe (bzw. Druck), Temperatur und Salzgehalt bedingt. Mit der Zunahme der Temperatur, der Tiefe und dem Salzgehalt des Wassers nimmt diese ebenfalls zu. In Abbildung 1 sind einige dieser Beziehung nochmals verdeutlicht.
Im Meerwasser beträgt die Schallgeschwindigkeit rund 1450 m/s - 1540 m/s was mehr als ein 4-faches der Geschwindigkeit in der Luft (343 m/s) ist [8]. Eine Formel zur genauen Bestimmung kann [19] entnommen werden.

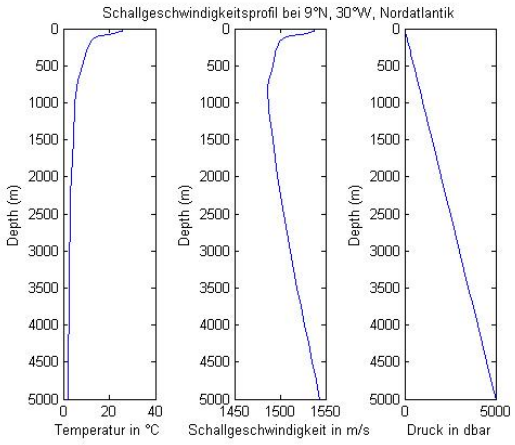


Abbildung 1: Abhängigkeiten der Schallgeschwindigkeit [21]

2.2 Ausbreitung und Frequenzen

Schall breitet sich in Wellenform aus. Im Wasser verbreitet er sich in Longitudinalwellen, d.h. die Schwingung erfolgen in Ausbreitungsrichtung der Welle [13].

Schallwellen haben eine Frequenz die beschreibt wie oft sich diese Welle pro Sekunde wiederholt. Wasserschall nutzt Frequenzbereiche von etwa 10 Hz bis 1 MHz. In diesen Frequenzbereich liegen drei Arten von Schall: Infraschall, Hörschall und Ultraschall (siehe Abbildung 2).

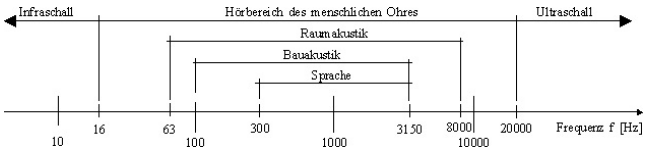


Abbildung 2: Schallarten und deren Frequenzen [14]

2.2.1 Infraschall

Infraschall ist Schall im Frequenzbereich das unter 16 Hz liegt und nicht von Menschen wahrgenommen werden [7].

2.2.2 Hörschall

Hörschall nennt man Frequenzen zwischen 16 Hz - 20 kHz. Wie der Name schon sagt, liegen in diesem Frequenzbereich die Geräusche die der Mensch auch wahrnehmen kann [14].

2.2.3 Ultraschall

Ultraschall gegenüber Infraschall hat einen viel höheren Frequenzbereich das zwischen 20 kHz - 1 GHz liegt und sich oberhalb des vom Menschen wahrnehmbaren Bereich befindet [7].

3. PROBLEME DER UNTERWASSERKOMMUNIKATION

Bei einer Kommunikation unter Wasser kommt es zu einigen Einschränkungen durch Störungen. Störungen schwächen oder verändern das Signal, sodass es zu schlechten Signal-Rausch-Verhältnissen bzw. Signalverlust beim Empfänger kommt. In diesem Abschnitt werden diese Probleme vorgestellt.

3.1 Absorption

Die Absorption ist die Hauptursache für die Reichweitenbeschränkung der akustischen Unterwasserkommunikation.
Die Intensität einer Schallwelle nimmt mit jeder zurückgelegten Einheit der Distanz ab. Die Energie wird von der Umgebung die sie durchdringt teilweise absorbiert, d.h. ein Teil der Schallenergie wird in Wärmeenergie umgewandelt [17]. Die Absorption nimmt mit steigende Frequenz zu [3]. Hierbei ist die Beziehung der Distanz und der Frequenz sehr wichtig. Die Abnahme der Intensität, von rot zu schwarz, bei 2 kHz ist der Abbildung 3 zu entnehmen. Man erkennt dass diese mit der Distanz abnimmt.

3.2 Reflexion und Streuung

Das Meer ist begrenzt durch die Meeresoberfläche und den Meeresboden. Diese beiden Grenzen dienen nicht nur zum Differenzieren des Meeres von andere Schichten der Erde, sondern setzt ebenso Grenzen für den Schall im Wasser [17]. Beim Auftreffen einer Schallwelle von unten auf die Meeresoberfläche wird dieser bei ruhiger See fast komplett um 180° reflektiert. Durch den Seegang wird dies beeinflusst und kann auch zur Streuung führen. Beim Meeresboden wird diese Eigenschaft durch die Beschaffenheit des Bodens (z.B. Gesteinstyp oder Ebenheit) bestimmt. In Abbildung 3 sind diese Reflexionen oben und unten zu erkennen.
Zusätzlich treten Reflexion und Streuung auch bei Wasserblasen und größeren Objekten wie Fischen auf. Wegen den verschiedenen Temperaturverhältnissen an unterschiedlichen Teilen des Meeres werden Schallwellen gebrochen, sodass sie gegen die Meeresoberfläche oder -boden gelenkt werden.
Reflexion und Streuung verursachen das Problem der Mehrwegausbreitung. Bei der Mehrwegausbreitung werden Signale aufgrund der verschieden langen Wege zeitlich versetzt und erzeugen einen Nachhall was für Unverständlichkeit beim Empfänger sorgt. Ebenso führt diese Eigenschaft zu Intersymbolinterferenzen (ISI) die bei Datenübertragungen durch Überlagerungen der Frequenzen zu Informationsverlust führen [17].
Neben der Absorption ist dies eine der größten Hürden der Unterwasserkommunikation. In Abbildung 4 ist die Reflexion und der Mehrwegausbreitung nochmals grafisch dargestellt.
Die Stärke und Häufigkeit dieses Phänomens wird von der Tiefe, Entfernung und Frequenz der Übertragung bestimmt [17]. Sie ist stärker bei Verbindungen in vertikaler Richtung als in horizontaler Richtung.

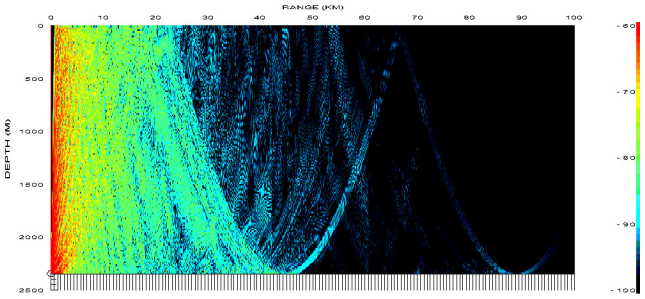


Abbildung 3: Energieverlust bei 2 kHz über 100 km [9]

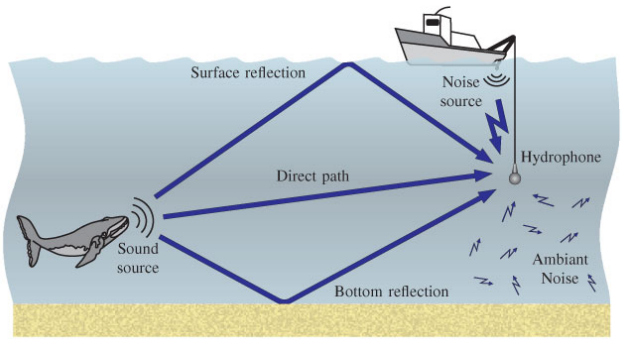


Abbildung 4: Störungen unter Wasser [1]

3.3 Unterwasserlärm

Unterwasserlärm bezeichnet Schallwellen die nicht vom Sender der Verbindung stammen. Der Lärm stört Signale vom Sender und verschlechtert das Signal-Rausch-Verhältnis (SNR) beim Empfänger.
Im Laufe der letzten Jahrezehnte hat der Unterwasserlärm

Tabelle 1: Reichweiten mit ihre Frequenzen [17]

	Reichweite (km)	Frequenz (kHz)
Lang	20 - 2000	0.5 - 10
Mittel	1 - 20	10 - 100
Kurz	< 1	> 100

durch die Erhöhung der Meeresaktivitäten, hauptsächlich des Schiffsverkehrs, drastisch zugenommen. Diese stören nicht nur die akustische Unterwasserkommunikation, sondern auch Meereslebewesen wie z.B. Wale und Delphine die Schallwellen zur Navigation und Kommunikation verwenden [2].
Durch die Aggregation von [19] lässt sich der Lärm in zwei Geräuschekategorien aufteilen:

3.3.1 Künstliche Geräusche
Ursprung direkt oder indirekt von Menschen, z.B. Schiffe, Ölplattformen, Pumpen, Taucher.

3.3.2 Umgebungsgeräusche
Ursprung von Meeresflora und -fauna, geologische Aktivitäten, Strömungen, Regen.

3.4 Zeitliche Verzögerung
Bei der elektromagnetischen Kommunikation basiert die Übertragungsgeschwindigkeit auf der Lichtgeschwindigkeit von ca. 300.000 km/s. Bei einem Erdumfang von ca. 40.000 km kann ein Signal innerhalb von 1 Sekunde diese 7.5x umrunden. Im Vergleich dazu beträgt die Wasserschallgeschwindigkeit ca. 1500 m/s was $5 \cdot 10^{-6}$ davon darstellt.
Durch diese viel niedrigere Geschwindigkeit muss man bei der akustischen Unterwasserkommunikation die Übertragungszeit viel mehr in Betracht ziehen, was man bei Lichtgeschwindigkeit fast vernachlässigen kann. Aus diesem Grund haben akustische Kommunikationen viel höhere Latenzzeiten als die der Elektromagnetischen. Durch die vielen physikalischen Variablen (Dichte, Temperatur, Salzgehalt) auf lange Distanzen ist es kaum möglich eine genaue Paketumlaufzeit bzw. *round trip time* (RTT) zu bestimmen.

4. REICHWEITEN
Wie bereits im vorherigen Abschnitt beschrieben, hängen die Reichweiten der akustischen Unterwasserkommunikation sehr von den Frequenzen ab. Daher müssen unterschiedliche Anwendungen mit verschiedenen Reichweiten ebenso auch verschiedene Frequenzen verwenden. Die Tabelle 1 beschreibt laut [17] die drei typischen Kategorien. Beispielanwendungen für lange Reichweite findet man bei Ozean-Tomographie, mittlere Reichweite meistens bei Kommunikation zwischen Sensorenknoten und Meeresoberfläche und kurze Reichweite haben Kommunikation zu Wartungsrobotern von Ölplattformen.
Generell werden für kurze Entfernungen meist Ultraschall und für lange Entfernungen Infraschall verwendet.
Um eine hohe Datenrate zu erziehen müssen hohe Frequenzen verwendet werden. Doch aufgrund der beschriebene Eigenschaft der Absorption muss man mit steigender Distanz, die verwendete Frequenz senken [17]. Damit wird die Reichweite des Signals erhöht und das SNR optimal gehalten, sodass eine zuverlässige und effiziente Verbindung aufgebaut werden kann.

5. UNTERWASSERMODEM
Das Überwinden der nun kennengelernten physikalischen Hürden macht die akustische Kommunikation unter Wasser möglich. Zum Senden und Empfangen von Daten wird ein Unterwassermodem benötigt. Ein Unterwassermodem hat die Aufgabe Daten in eine Trägerfrequenz zu modulieren und zu versenden, sowie Signale zu empfangen und die Daten daraus zu rekonstruieren (Demodulation).
Die genaue Funktionalität eines solchen Modems kann dem Patent von Troin und Cazaoulou entnommen werden [4].
Im folgenden Abschnitt werden einige Modulationsverfahren gefolgt von den Daten- und Fehlerraten der typischen Signaltypen vorgestellt.

5.1 Modulation
Neben der Abhängigkeit der Datenrate von physikalischen Gegebenheiten, ist es ebenfalls möglich Datenraten zwischen Sender und Empfänger mit Hilfe von Modulationsverfahren zu steigern. Die Modulation ist die Veränderung einer Trägerfrequenz durch die Daten- bzw. Nutzfrequenz. Einfache binäre Modulationsverfahren benötigen zwei Zustände bzw. Sendesymbole zur Darstellung von 0 und 1. Durch komplexere Variation dieser ist es möglich in einen Sendesymbol mehr als nur ein Bit zu übermitteln, wodurch die Datenrate dadurch vervielfacht werden kann. Für die akustische Kommunikation sind hierbei vier Modulationsverfahren besonders interessant.

5.1.1 Frequenzumtastung (FSK)
Die Frequenzumtastung (*frequency-shift keying*) ist ein Frequenzmodulationsverfahren (FM) bei der diskrete Zustände wie 0 und 1 durch diskrete Frequenzänderung dargestellt werden.
In Abbildung 5 ist eine Form dieses Verfahrens grafisch dargestellt. Diese spezielle Form mit zwei Zuständen wird auch 2-FSK bzw. *binary* FSK (BFSK) genannt [11]. Jeder Zustand 0 und 1 (oben) wird durch eine jeweilige Frequenz (unten) dargestellt. Die zu sendende Signale werden durch die Aneinanderreihungen der repräsentativen Frequenzen zusammengesetzt. Wie im Abbildung 5 zu sehen ist wird "1010" dargestellt.
Diese Modulation ist unter Wasser wegen der Abhängigkeiten der Absorption und 1-Bit pro Sendesymbol durch niedrige Datenrate geprägt.

5.1.2 Amplitudenumtastung (ASK)
Die Amplitudenumtastung (*amplitude-shift keying*) ist ein Amplitudenmodulationsverfahren (AM) welches analog zu FSK die Zustände mit Hilfe von unterschiedliche Amplituden moduliert.
Bei einer binären ASK (BASK) bzw. 2-ASK werden zwei verschiedene Amplituden verwendet [11]. An diesen unterschiedlichen Amplituden können dann die zwei Zustände 0 und 1 erkannt werden (Abbildung 6 mitte).
In einer Umgebung mit minimale bzw. keine Störungen kann dies auch einfach durch ein Ein- und Ausschalten des Signals umgesetzt werden. Diese Form der Umsetzung nennt man auch On-Off Keying (Abbildung 6 unten) [11].

5.1.3 Phasenumtastung (PSK)
Die Phasenumtastung (*phase-shift keying*) ist ein Phasenmodulationsverfahren (PM) das Phasenlagen statt diskreter

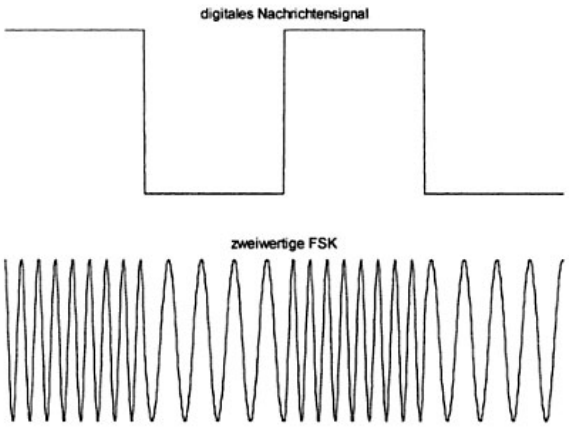


Abbildung 5: Binäre Frequenzumtastung (BFSK) [11]

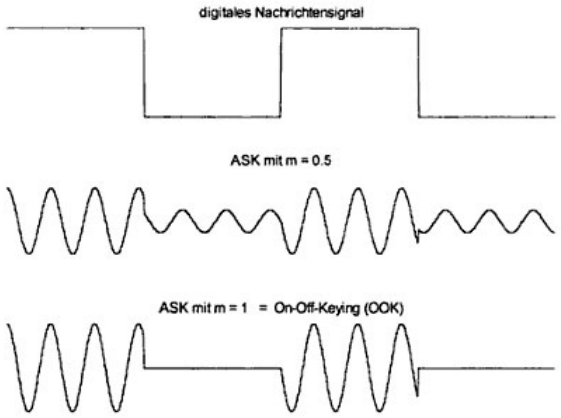


Abbildung 6: Binäre Amplitudenumtastungumtastung (BASK) [11]

Frequenzen zu Modulation nutzt. Dafür verwendet es eine Sinus- oder Kosinus-Trägerfrequenz und moduliert die Zustände als Phasenverschiebung bzw. -sprünge dieser Trägerfrequenz [11].
Abbildung 7 veranschaulicht auch hier die binäre Variante dieses Verfahren. Da diese Variante zwei Zustände benötigt wird die Trägerfrequenz von 0° um 180° verschoben. Nehmen wir nun an die Trägerfrequenz bestehen aus normalen Sinusschwingungen, dann starten die 0-Phasen bei der steigende Kurve und die 1-Phasen bei der fallende Kurve, d.h. die ”0” wird mit der normalen und die ”1” mit der verschobene Sinusschwingung dargestellt. In unseren Beispiel ist die Sequenz ”110100” zu erkennen.
Bei diesem Verfahren bestehen die zu sendenden Signalen aus Aneinanderreihungen der Schwingungen aus verschiedenen verschobenen Phasen der Trägerfrequenz. Die Demodulation dieses Verfahrens muss hierbei kohärent sein, d.h. Sender und Empfänger müssen auf der selben Frequenz operieren, da die Phasen sonst nicht genau erkennbar sind [11].

Quadraturphasenumtastung (QPSK)
Die bis jetzt kennengelernten Verfahren machen es möglich

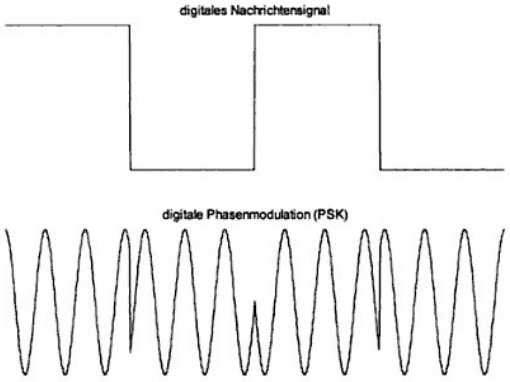


Abbildung 7: Binäre Phasenumtastung (BPSK) [11]

Binärdaten auf drei verschiedene Arten zu versenden. Alle drei sind nicht sehr effizient da sie pro Sendesymbol nur ein Bit übertragen. Eine praktische Variante der PSK ist die Quadraturphasenumtastung (*quadrature phase-shift keying*) bzw. 4-PSK. In dieser Variante besitzt das Verfahren vier Sendesymbole in der jeweils zwei Informationsbits enthalten wodurch die Datenrate verdoppelt werden kann [11]. Ebenso ist sie ein mehrdimensionales Verfahren wie wir gleich sehen werden.
Um diese Variante umzusetzen werden die zu übertragende Bits sozusagen nummeriert und in zwei Teile aufgeteilt, dem Realteil I und Imaginärteil Q. Die Namen der Teile haben keinen Bezug auf die komplexen Zahlen aus der Mathematik, sondern bezeichnen die beiden Dimensionen. I besteht aus den Bits mit ungerader Nummerierung und Q aus die der Geraden. I und Q werden regulär mit BPSK moduliert. Die Besonderheit dabei ist jedoch, dass Q gegenüber I um 90° verschoben ist (Kosinus), welches dadurch orthogonal zu I steht wodurch sie sich nicht gegenseitig stören. Anschließend werden beide modulierte Teile zusammenaddiert und stellt somit das QPSK Signal dar.
Die Abbildung 8 zeigen die beiden Teile aller vier Sendesymbole von QPSK. Auch hier bestehen die zu sendenden Signale aus den Aneinanderreihungen dieser Schwingungen. Die Demodulation eines Sendesymbols kann mit Hilfe eines Konstellationsdiagramms wie in Abbildung 9 veranschaulicht werden. Das QPSK Signal wird am Demodulator zuerst wieder in ihre Bestandteile I und Q getrennt. Da I aus Kosinus-Schwingungen und Q aus Sinus-Schwingungen bestehen ist die Realisierung technisch einfach. Jeder Bestandteil wird darauf parallel abgetastet. Die beiden erhaltene Werte werden im Diagramm an ihre Achsen nachverfolgt. Dadurch erhält man wieder die modulierten Bits.
Die Anordnung der Punkte in diesem Konstellationsdiagramm ist nach dem Gray-Code Schema, bei der die benachbarten Zustände sich jeweils nur um ein Bit unterscheiden. Dadurch gibt es bei gestörten Übertragungen meist nur Fehler in einem Bit, wodurch Übertragungen robuster und Fehlerkorrekturen einfacher werden [11].
Die Kreislinie in Abbildung 9 gibt die Plazierungsmöglichkeiten der Punkte an. Da PSK zwar verschiedene Phasen haben, aber immer die gleiche Amplitude, kann es sich deswegen nur auf der Kreislinie bewegen.

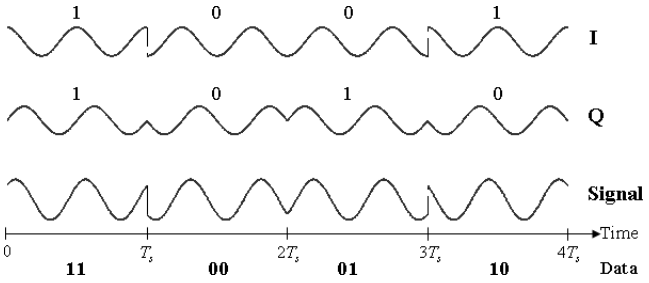


Abbildung 8: Quadraturphasenumtastung (4-PSK oder QPSK) [22]

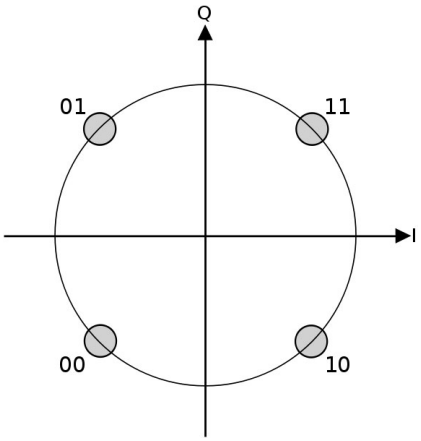


Abbildung 9: Konstellationsdiagramm der QPSK und 4-QAM (ohne Kreisbegrenzung) [23]

5.1.4 Quadraturamplitudenmodulation (QAM)

Quadraturamplitudenmodulation (*Quadrature amplitude modulation*) ist wie die ASK ein Amplitudenmodulationsverfahren. Sie kombiniert dabei ASK und PSK.
Die Implementation erfolgt fast analog zu QPSK mit den Unterschieden, dass die beiden Teile (I und Q) nach der Trennung mit ASK statt PSK moduliert werden und anschließend I mit Kosinus und Q mit dem Minus-Sinus multipliziert werden [5]. Zum Abschluss werden die beiden Teile durch die Summe wieder zusammengefügt und repräsentieren das QAM Signal.
Das Konstellationsdiagramm der 4-QAM ist das selbe wie bei QPSK, da keine verschiedene Amplituden in dem Fall nötig sind. Die Datenrate wird ebenfalls durch die Modulation von zwei Bits je Sendesymbol verdoppelt. Noch höhere Datenraten sind durch Erhöhung der Anzahl der Zustände möglich. Die Erhöhung erfolgt in Schritten der Zweierpotenzen. Ein Beispiel ist das 16-QAM welches 16 Zustände verwendet und somit pro Sendesymbol 4 Bits ($2^4 = 16$) übertragen kann. Theoretisch ist die Erhöhung fast beliebig, doch jede Erhöhung verengt die Räume zwischen den Punkten und somit auch die Toleranzbereiche für die Demodulation.
Die Konstellationsdiagramme mit 16 und 32 Zustände sind der Abbildung 10 zu entnehmen. In den Diagrammen kann man erkennen, dass die Amplitude eine wichtige Rolle spielt. Da sich die Punkte nicht wie bei QPSK immer auf einer Kreislinie befinden, können sie unterschiedliche Amplituden besitzen. Ein Beispiel sind die Zustände ”0011” und ”0000” der 16-QAM welche die selbe Phase, aber unterschiedliche Amplituden besitzen.

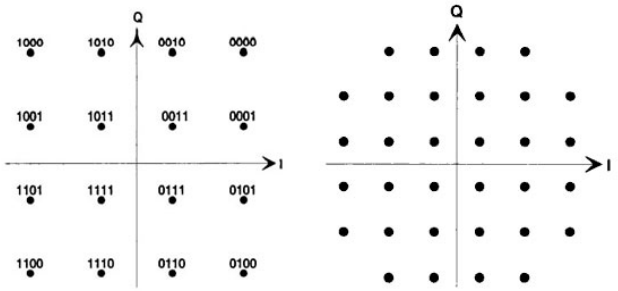


Abbildung 10: Konstellationsdiagramm der 16-QAM (links) und 32-QAM (rechts) [11]

5.2 Signaltypen mit Datenraten und Fehlertoleranzen

Je nach Anwendung hat ein Unterwassermodem verschiedene Datenraten mit den zugehörigen akzeptablen Fehlerraten. Laut [17] gibt es drei Signaltypen die unter Wasser übertragen werden.

5.2.1 Kontrollsignale

Diese Signale dienen der Wartung und Kontrolle der Sensoren. Sie erfordern eine hohe Zuverlässigkeit mit niedriger Fehlerrate, sodass eine Datenrate von unter 1 kb/s ausreichen

5.2.2 Messdaten

Messdaten haben dagegen andere Anforderung. Neben den vielen Zahlenwerten beinhalten sie ebenfalls sehr kleine Bilddaten weshalb eine Datenrate ab 10 kb/s nötig sind. Gegenüber der Kontrollsignale besteht hier keine so hohe Anforderung an die Zuverlässigkeit, sodass eine Fehlerrate von $10^{-3} - 10^{-4}$ akzeptabel sind.

5.2.3 Multimediasignale

Die höchsten Datenraten werden für Multimediasignale benötigt. Wegen der hohen Qualität bzw. hohe Datenmengen werden Datenraten von 10 - 500 kb/s benötigt. Hierbei ist eine Fehlerrate bis 10^{-4} vertretbar.

6. UNTERWASSERSENSOREN

Unterwassersensoren werden verwendet um über eine lange Zeit einen Bereich des Meeres zu überwachen und Daten zu sammeln, die von Wissenschaftler ausgewertet werden. Die Anwendungen sind sehr vielfältig. Beispielsweise können Sensoren die Strömung und die Temperatur vom Golf von Mexiko protokollieren oder regelmäßig Wasserproben entnehmen und deren Bestandteile messen um den Verlauf der Verschmutzung herauszufinden. Die Vielfalt ist allein durch die Vorstellungskraft des Menschen begrenzt.
Generell bestehen Hightech Unterwassersensoren aus vier Hauptkomponenten:

- Sensorenknoten
- Anker

- Bergungsmechanismus
- Treibeinheit

Im folgenden Abschnitt werden zwei Arten von Sensorenknoten und deren Vor- und Nachteile betrachtet.

6.1 Akustischer Bergungsmechanismus

Ein wichtiger Mechanismus den beide Sensorentypen gemeinsam haben, ist der Mechanismus zum Bergen. Dafür wird über die akustische Kommunikation ein sogenannter Release-Befehl an das Bergungsmodul des Sensorenknotens gesendet, der nach Verifikation des Befehls die Befestigung des Ankers löst oder durchtrennt, sodass der Knoten durch die Treibeinheit an die Meeresoberfläche treiben kann.
Abbildung 11 stellt ein solches Bergungsmodul dar. Bei diesem Mechanismus wird der rote Teil eingezogen und somit den Anker gelöst.
Solche Systeme verwenden nur Kontrollsignale und arbeiten je nach Entfernung auf niedrige bis mittelhohe Frequenzen.

Abbildung 11: Bergungsmechanismus der Firma Sonardyne [15]

6.2 Traditionelle Sensorenknoten

Eine einfache Methode um Messdaten vom Meeresgrund zu erhalten sind die traditionellen Sensorenknoten [16]. Diese Sensoren werden bei der Verwendung an einen zu überwachenden Bereich des Gewässers transportiert und dort versenkt und später wieder geborgen.
Während der Sensorenknoten im Wasser ist sammelt er die vorgesehenen Daten und speichert sie auf seinem Datenträger. Sind die benötigten Daten komplett oder die Kapazität der Batterie bzw. der Datenträger des Knotens erschöpft, wird der Knoten geborgen und die Wissenschaftler können die gesammelten Daten auswerten.

6.2.1 Vorteile

Die traditionelle Methode ist viel kostengünstiger als die folgende Echtzeitvariante. Durch das Versenken und die Bergung von Sensorenknoten entstehen nur hierfür, sowie den Transport, Kosten.
Wegen der Abgeschiedenheit ohne Kommunikation in der Messphase ist der Energiebedarf konstant klein sodass der Energievorrat relativ lange nutzbar ist.

6.2.2 Nachteile

Wie schon vorher angedeutet hat der Sensor begrenzte Kapazitäten an Energie und Speicherplatz. Das Ausnutzen von Solarenergie für den Energiebedarf kann aufgrund der Dunkelheit in der Tiefe des Meeres nicht verwendet werden.
Durch die späte Interaktion und Überprüfung des Sensorenknotens beim Bergen können eventuelle Probleme durch Beschädigung oder Fehlkonfiguration erst am Ende der Messphase erkannt werden. Zu diesem Zeitpunkt kann es bereits zu spät sein und somit das gesamte Projekt zum Scheitern bringen [16].

6.3 Echtzeit-Sensorenknoten

Die neuere Methode basiert auf Sensorenknoten die über eine konstante Verbindung zu den Wissenschaftlern der Kontrollstationen verfügen [16].
Für die Verbindung benötigt man neben dem Sensorenknoten noch eine Relay-Boje zwischen der Oberfläche und dem Meer, sowie einen Satelliten. Der Satellit ist meist nötig da die zu überwachende Bereiche weit vom Land entfernt sind und dadurch ebenfalls die Übertragung beschleunigt werden kann.
Die generelle Funktionalität lässt sich wie folgt beschreiben: Zuerst sendet der Sensorenknoten Daten über die akustischen Kommunikation zur Relay-Boje. Diese leitet es über eine elektromagnetische Kommunikation zum Satelliten weiter. Vom Satelliten aus werden die Daten dann an die Kontrollstationen am Land geschickt. Eine grafisches Darstellung (Abbildung 12) dieser Methode kann später am Praxisbeispiel in einem Tsunami-Warnsystem entnommen werden.

6.3.1 Vorteile

Mit der konstanten Verbindung ist eine Echtzeitüberwachung der Sensoren, sowie Messdaten möglich. Durch die bidirektionale Verbindung können Wissenschaftler von der Kontrollstation aus direkt die Sensoren warten und neu konfigurieren [16].
Da die Messdaten über die Verbindung sofort weitergeleitet werden, benötigt es nicht unbedingt Datenträger zum Speichern von Messdaten, wodurch die Speicherprobleme von traditionellen Sensorenknoten beseitigt sind.

6.3.2 Nachteile

Zu den Nachteilen zählt auch bei dieser Methode die fehlende Möglichkeit der Nutzung von Solarenergie, sodass die Energiekapazitäten begrenzt sind.
Wegen den zusätzlichen benötigten Geräte für die Verbindung zur Kontrollstation entstehen viel höhere Kosten im Gegensatz zur traditionellen Methode.
Die Möglichkeit der Kommunikation während der Messphase erzeugt ebenso einen höheren Energiebedarf was höhere Anforderungen an die Energieversorgung stellt.

6.4 Fazit

Beide Systemen haben ihre Vor- und Nachteile. Die Wahl wird meist durch die Kosten- und Zeitfaktoren getroffen. Die Echtzeit-Sensorenknoten haben hohe Flexibilität und erzeugen schnelle Resultate wogegen traditionelle Sensorenknoten niedrige Kosten und schnelle Umsetzung bringen.

7. UNTERWASSER SENSORENNETZE

Ein Sensorenknoten hat sehr begrenzte Energiereserven und Messreichweiten. Um eine größere Fläche abzudecken werden oft mehrere solcher Sensorenknoten benötigt. Mit Echt-

zeit-Sensorenknoten, die eine lokale bidirektionale Kommunikation erlauben, entstehen sogenannten Sensorennetze.
Vorteile dieser Netze liegen in den Erweiterungen der Messreichweite durch Synchronisation untereinander, der Verbindungsreichweite und Energiereserven mit Hilfe von Routingprotokollen die Hops erlauben. Durch höhere Anzahl an Sensorenknoten entsteht jedoch mehr Datenverkehr was zu gegenseitigen Störungen führt. Daher ist es besonders sinnvoll in einen Unterwassernetz Routingknoten einzusetzen, die horizontal auf kurze Entfernung mit den Sensorenknoten kommunizieren und Daten vertikal auf mittellanger Entfernung zur Boje weiterleiten. Dadurch können die Übertragungen von Sensoren durch höhere Datenraten verkürzt, sowie die Energiereserven geschont werden.

8. PRAXISANWENDUNG

Unterwasserkommunikation findet in vielen Bereichen Anwendungen. Neben dem Einsatz in Sonar- oder Echolotgeräten, wurde sie in der Vergangenheit hauptsächlich in Militärbereich verwendet.
Eines dieser Militärprojekte ist das SOSUS (Sound Surveillance System) der USA welches in den 50er Jahren zur Überwachung der sowjetische U-Boote verwendet wurde. Sie nutzt Hydrophone die auf den Meeresgrund installiert sind um passiv nach U-Boot-Geräuschen zu lauschen. Die Verbindung zur Kontrollstation kam mit Hilfe eines langen Unterwasserkabels zustande [20].
Neben dem Einsatz der Unterwasserkommunikation im Militärbereich ist deren praktische Anwendung in lebensrettenden Tsunami-Warnsystemen sehr von Bedeutung [6]. Im nächsten Abschnitt werden die Verwendung der vorher eingeführten Unterwassersensoren anhand eines Tsunami-Warnsystems vorgestellt.

8.1 Tsunami-Warnsystem: D.A.R.T. II

Ein Tsunami-Warnsystem soll eine drohende Tsunami-Welle schon weit vorm Erreichen der Ufer erkennen und die Menschen vor dieser Drohung frühzeitig warnen. Eine der essentiellen Komponenten ist die Kommunikation zwischen den Drucksensoren auf dem Meeresgrund und der Relay-Boje an der Meeresoberfläche welche die Information an bemannte Kontrollstationen am Land weiterleitet. Diese zeitliche Anforderungen erfordern Echtzeit-Sensorenknoten.

8.1.1 Einleitung

Eines der aktuellen Tsunami-Warnsysteme ist das D.A.R.T. II (Deep ocean Assessment and Reporting of Tsunamis) Projekt vom US National Oceanic and Atmosperic Administration. Dieses Projekt überwacht vor allem das Pazifische Meer, sowie den amerikanischen Teil des Atlantik.

8.1.2 Spezifikation

Die Sensoren besitzen einen 32 Bit, 3.3V Motorola 68332 Mikrocontroller mit 4MB Flash Speicher und 512 Byte RAM. Die Verbindung zwischen Sensoren und die Boje operiert auf Frequenzen 9 - 14 kHz mit einer Geschwindigkeit von 600 Baud. Zur Übertragung der Daten wird eine Variante der FSK verwendet. Zur Energieversorgung besitzen die Sensoren zwei Batteriepacks (Modem 1.560 Wh, Mikrocontroller 2.000+ Wh) die ungefähr vier Jahre ausreichen, je nach Auslastung auch weniger. Die Energiereserve der Boje besteht ebenso aus zwei Batteriepacks (Modem 1.800 Wh, Mikrocontroller und Satellitenverbindung 2.569 Wh), aber halten nur zwei Jahre [6].

Zusammenfassung:

- Mikrocontroller: 32 Bit, 3.3V Motorola 68332
- Speicher: 4 MB Flash und 512 Byte RAM
- Unterwassermodem: Benthos ATM-880
  - Frequenzen: 9 - 14 kHz
  - Datenrate: 600 Baud (Symbole pro Sekunde)
  - Modulation: MFSK (Multifrequenzumtastung)
- Energieversorgung: Batteriepacks Typ D (Mono)
  - Sensoren 4 Jahre
  - Boje 2 Jahre

8.1.3 Funktionaliät

Die Funktionalität dieses Systems ist in Abbildung 12 skizziert. Zuerst werden Drucksensoren in seismologisch kritischen Gebieten, z.B. an Kontinentalplattenrändern, plaziert. Nahe jedem dieser Sensoren wird ebenfalls eine Boje, zur Weiterleitung an Satelliten, installiert. Die Drucksensoren messen kontinuierlich Vibrationen der Erde und Wellengang. Dafür besitzen sie auf der Sensoren-Seite zwei automatische Benachrichtigungsmodi [6].

Standard Modus: Bei regulären Aktivitäten befinden sich die Sensorenknoten in einer Art Bereitschaftsmodus mit niedrigem Energieverbrauch. Ohne ungewöhnliche Aktivitäten leiten sie alle 6 Stunden Messdaten, wie z.B. durchschnittliche Wellenhöhe, Energiestand und Zeitstempel, an die Kontrollstationen weiter, welche daran erkennen können ob das System korrekt arbeitet.

Event Modus: Kommt es zu einer größeren Erschütterung bzw. übersteigen die Messewerte einen Schwellenwert, so alarmieren die Sensoren selbstständig die Kontrollstationen. Dazu senden sie diese Messdaten über die akustische Kommunikation an die Boje, welches diese darauf über einen Satelliten per elektromagnetische Kommunikation an die Kontrollstationen weiterleitet. Nach einer kurzen Überprüfung auf Fehlerfreiheit entscheiden die Mitarbeiter der Station ob eine Tsunami-Warnung ausgesandt werden muss.

Neben den beiden automatischen Modi, besitzen die Sensoren ebenfalls einen manuellen Modus mit der Wissenschaftler auf der Kontrollstation-Seite außerplanmäßige Abfragen und Wartungsarbeiten tätigen können.

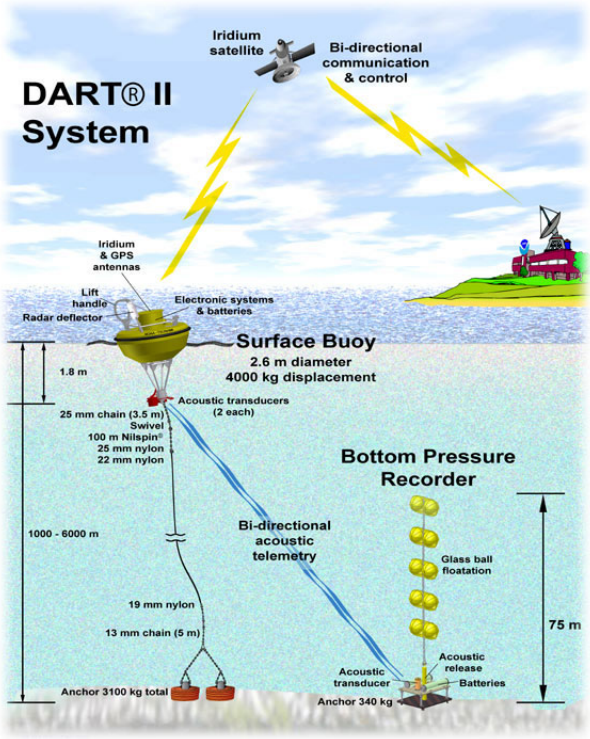


Abbildung 12: Funktionalität eines Tsunami-Warnsystems [12]

9. ZUSAMMENFASSUNG

Damit der Mensch auch die Möglichkeit hat, die unerreichbaren Tiefen der Gewässer zu erkunden wird die drahtlose Unterwasserkommunikation benötigt. Wegen der hohen Abschwächung der elektromagnetischen Signale unter Wasser musste eine andere Möglichkeit der Kommunikation gefunden werden. Dafür bieten sich die akustische Signale an die wir in dieser Arbeit näher kennengelernt haben.
Zuerst haben wir die physikalischen Grundlagen von Wasserschall betrachtet. Die Geschwindigkeit, deren Ausbreitung und die drei relevanten Frequenztypen.
Die Probleme der Unterwasserkommunikation durch Absorption, Reflexion, Streuung, Mehrwegausbreitung, Unterwasserlärm und die zeitlichen Verzögerungen waren Thema des folgenden Kapitels. Die Absorption ist die Umwandlung von einem Teil der Schallenergie in Wärmeenergie. Sie steigt mit der Frequenz und Distanz. Reflexion und Streuung treten beim Auftreffen von Schallwellen auf Meeresoberfläche und -boden, sowie große Objekte auf. Sie führen zu Mehrwegausbreitungen die zeitlich versetzte Signale verursachen und ISI erzeugen. Neben der Störung durch unerwünschten Unterwasserlärm, sind die Latenzzeiten der Verbindung recht hoch.
Die Reichweiten hängen stark von der verwendeten Frequenz ab. Bei hoher Reichweite benötigt man Infraschall und für niedrige Reichweiten ist Ultraschall nutzbar.
Um Daten senden zu können benötigt man ein Unterwassermodem das die Daten zum Senden in Trägerfrequenzen moduliert. Dafür haben wir mehrere Modulationsverfahren (FSK, ASK, PSK, QPSK und QAM) kennengelernt. Durch Modulation von mehreren Bits in einem Sendesymbol können höhere Datenraten erzielt werden.
Das Unterwassermodem wird in Unterwassersensoren verwendet und dient bei traditionellen Sensorenknoten nur zur Bergung. Echtzeit-Sensorenknoten nutzen es aktiver. Sie haben eine konstante Verbindung zur Kontrollstation und können in Echtzeit Daten übermitteln und Befehle empfangen. Die aktive Nutzung führt auch zum Aufbau von Sensorennetzen welche Messbereiche, Energiereserven und Verbindungsreichweiten erweitern können.
Zum Abschluss wurden Praxisanwendungen vorgestellt. Dazu haben wir speziell ein Tsunami-Warnsystem mit seiner Funktionsweise und Spezifikation betrachtet.

10. LITERATUR

[1] A. Q. Arnaud Jarrot, Cornel Ioana. Denoising underwater signals propagating through multi–path channels, 6 2010. http://www.ensieta.fr/e3i2/images/E3I2/Publi/publi/JARROT_05a.pdf.

[2] S. Donner. Lärm unter wasser: Von wegen stiller ozean. Online Artikel, 10 2009. http://www.spiegel.de/wissenschaft/natur/0,1518,652873,00.html.

[3] R. E. Francois and G. R. Garrison. Sound absorption based on ocean measurements: Part i: Pure water and magnesium sulfate contributions. *The Journal of the Acoustical Society of America*, 72(3):896–907, 1982.

[4] M.-S. F. Georges Troin and N. F. Claude Cazaoulou. Underwater acoustic transmission method and equipment to improve the intelligibility of such transmissions. Patent, 11 1996. US 5572485.

[5] P. D.-I. habil. Günter Söder. Underwater acoustics. Vorlesung, 6 2010. www.lnt.ei.tum.de/download/2010_NT1_Soeder/NT1_Kapitel6.pdf.

[6] S. E. S. Hugh B. Milburn, Christian Meinig and A. I. Nakamura. Real-time deep-ocean tsunami measuring, monitoring, and reporting system: The noaa dart ii description and disclosure. Technische Spezifikation, 6 2010. http://www.ndbc.noaa.gov/dart/dart_ii_description_6_4_05.pdf.

[7] I. K. Jürgen Kiefer. *Allgemeine Radiologie: Strahlenanwendung, Strahlenwirkung, Strahlenschutz.* Georg Thieme Verlag, 2003.

[8] P. D.-I. D. Kraus. Underwater acoustics. Vorlesung, 6 2010. http://homepages.hs-bremen.de/~krausd/iwss/uwa.html.

[9] M. G. Lee Freitag, Milica Stojanovic and S. Singh. Acoustic communications for regional undersea observatories. In *In Proc. Oceanology Intl*, 2002.

[10] N. Magazine. U.s. announces plan for an improved tsunami detection and warning system. Ankündigung, 6 2010. http://www.noaanews.noaa.gov/stories2005/s2369.htm.

[11] M. Meyer. *Kommunikationstechnik. Konzepte der modernen Nachrichtenübertragung.* Vieweg Verlag, 2002.

[12] U. N. Oceanic and A. Administration. Dart ii system, 6 2010. http://www.ndbc.noaa.gov/dart/dart_mooring.jpg.

[13] D. D. Russell. Longitudinal and transverse wave motion, 6 2010. http://paws.kettering.edu/~drussell

/Demos/waves/wavemotion.html.

[14] U.-P. D. M. J. Setzer. Schall. Online Buch, 6 2010. http://www.uni-due.de/ibpm/BauPhy/Schall /indexschall.htm.

[15] Sonardyne. Acoustic release transponders. Produktkatalog, 6 2010. http://www.sonardyne.co.uk/Products /ReleaseActuation/systems/acoustic_releases.html.

[16] E. Sozer, M. Stojanovic, and J. Proakis. Underwater acoustic networks. *Oceanic Engineering, IEEE Journal of*, 25(1):72 –83, jan 2000.

[17] M. Stojanovic. Underwater acoustic communications. In *Electro/95 International. Professional Program Proceedings.*, pages 435 –440, 21-23 1995.

[18] Trikustik. Akustik lexikon. Lexikon, 6 2010. http://www.trikustik.at/wissen/akustik.html.

[19] R. J. Urick. *Principles of Underwater Sound*. McGraw Hill, New York, 1975.

[20] E. C. Whitman. Sosus the "secret weapon" of underwater surveillance. Online Artikel, 6 2010. http://www.navy.mil/navydata/cno/n87/usw/issue_25 /sosus.htm.

[21] Wikipedia. Beispiel eines schallgeschwindigkeitsprofils, 6 2010. http://upload.wikimedia.org /*wikipedia*/*de*/2/26/Schallgesch_nordatl.jpg.

[22] Wikipedia. Timing diagram for qpsk, 6 2010. http://upload.wikimedia.org/wikipedia/commons/b /be/QPSK_timing_diagram.png.

[23] Wikipedia. Timing diagram for qpsk, 6 2010. http://upload.wikimedia.org/wikipedia/commons/8/8f /QPSK_Gray_Coded.svg.

Sensorknoten - Sicherheit auf Schicht 2

Tobias Niedl
Betreuer: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: niedl@in.tum.de

KURZFASSUNG

Die sichere Kommunikation in Sensornetzen war und ist Gegenstand einiger Forschungsarbeiten. Diese Arbeit stellt die Systeme SPINS, TinySec und TinyPK vor, die in den letzten Jahren entwickelt wurden und Sicherheit auf Schicht 2 des OSI-Modells bieten. Alle Systeme arbeiten auf vergleichbaren Hardware-Plattformen und nutzen oder erweitern das Open-Source-Betriebssystem TinyOS für Sensor-Knoten. Das Hauptproblem für alle drei vorgestellten Systeme ist dabei die geringe Rechen- und Speicherkapazität der Sensorknoten.
Während SPINS und TinySec symmetrische Kryptographie-Verfahren verwenden, geht TinyPK einen Schritt weiter und versucht asymmetrische Verfahren zu implementieren, die pauschal mehr Ressourcen benötigen.
SPINS war einer der ersten Ansätze sichere Kommunikation in Sensornetzen zu ermöglichen. TinySec nutzte einige Erkenntnisse aus SPINS und kann heute offiziell zusammen mit TinyOS aus dem Internet heruntergeladen und verwendet werden. TinyPK zeigt, dass asymmetrische Kryptographie — zumindest teilweise — auf Sensorknoten implementiert und verwendet werden kann.

Schlüsselworte

Sensor Network Security, Link Layer Security, SPINS, TinySec, TinyPK

1. EINLEITUNG

Als Sensornetz wird ein Netzwerk von drahtlos kommunizierenden Rechnern (sog. Knoten) verstanden. Die Knoten sind batterie- bzw. akkubetrieben, verfügen über nur geringe Rechen- und Speicherkapazitäten und sind entsprechend günstig in der Produktion. Die Knoten können je nach Produktionsart für viele verschiedene Aufgaben verwendet werden, beispielsweise zur Überwachung von Umwelt-Werten. Die erfassten Messwerte können drahtlos zwischen den Knoten ausgetauscht oder an eine Basisstation, die als Gateway fungiert, geleitet werden. Aufgrund der Energieversorgung mittels Batterien sind die zur Verfügung stehenden Ressourcen stark begrenzt. Die Taktrate der Prozessoren liegt im einstelligen Megaherz-Bereich. Der verfügbare Speicher (Flash, RAM, EEPROM) liegt zwischen einigen hundert Byte und einigen Kilobyte. Aufgrund der begrenzten Ressourcen können herkömmliche Algorithmen und Protokolle, wie sie beispielsweise in IPsec [7] oder TLS [4] verwendet werden, nicht zum Schutz von Sensornetzen genutzt werden (vgl. [6]). Entsprechend erfolgte die Entwicklung der hier vorgestellten Systeme SPINS, TinySec und TinyPK immer unter dem Gesichtspunkt der mangelnden Rechenkapazitäten und dem begrenzen Energievorrat [10, 6, 13].

Für Sensorknoten existiert ein Open-Source-Betriebssystem namens TinyOS [8]. Unter TinyOS sind keine kryptographischen Funktionen implementiert um die Datenübertragung zwischen den Knoten zu sichern. Die vorgestellten Systeme erweitern oder basieren jedoch auf TinyOS und implementieren Funktionen zur Sicherung der Kommunikation auf dem sog. Link-Layer, also Schicht 2 nach dem OSI-Referenzmodell [3].

Da die Kommunikation zwischen Knoten unter TinyOS standardmäßig im Klartext erfolgt, sind Angriffe gegen ein Sensornetz sehr einfach. Ein Angreifer kann die übertragenen Pakete mitlesen, abfangen, modifiziert weiterleiten und eigene Pakete ins Netz senden. Die vorgestellten Systeme versuchen jeweils einige dieser Angriffe zu erkennen bzw. zu unterbinden.

Das Ziel von TinySec und SPINS ist es, eine sichere Kommunikation zwischen den Knoten (TinySec) bzw. einem Knoten und einer der Basisstation (SPINS) zu ermöglichen. TinyPK widmet sich der Problemstellung, ein nicht zum Sensornetz gehörendes Gerät, die „External Party“ (EP), zu authentisieren und ihr sicher einen gemeinsamen Schlüssel zu übermitteln, damit sie an der Kommunikation im Sensornetz teilnehmen kann.

In Abschnitt 2 werden zunächst einige wichtige Begriffe im Zusammenhang mit sicherer Kommunikation erläutert. Abschnitt 3 stellt SPINS vor, das aus den Protokollen SNEP und µTESLA besteht. Abschnitt 4 erläutert das neuere TinySec-System und Abschnitt 5 stellt TinyPK vor. In Abschnitt 6 folgt ein Vergleich der genannten Systeme und in Abschnitt 7 werden einige Angriffe aufgeführt, die auch bei Verwendung der vorgestellten Systeme möglich sind.

2. SICHERHEIT

2.1 Schutzziele

Für eine sichere Kommunikation gilt es, bestimmte Schutzziele zu erreichen. Diese lauten: Geheimhaltung, Integrität, Authentizität und Frische, sowie Verfügbarkeit, Verbindlichkeit und Anonymität (vgl. [5]). Die vorgestellten Systeme erreichen jedoch bei weitem nicht alle davon. Entsprechend werden im folgenden nur die Schutzziele näher erläutert, die wenigstens eines der Systeme auch erreicht.

2.1.1 Geheimhaltung (Confidentiality)

Die Geheimhaltung dient dazu, den tatsächlichen Inhalt einer Nachricht (den Klartext) gegenüber unautorisierten Parteien zu verschleiern. Dies wird meist durch eine Verschlüsselung der Nachricht erreicht. Idealerweise wird der selbe Klartext P in verschiedenen Nachrichten zu verschiedenen Kryptotexten C und C' verschlüsselt. Diese Eigenschaft wird als *semantic security* bezeichnet. Dadurch wird verhindert, dass ein Angreifer aus vielen abgehörten, verschlüsselten Nachrichten etwas über den Inhalt lernen kann. D.h. ein Angreifer soll nur mit einer 50%-igen Wahrscheinlichkeit erraten können, ob ein Bit des Kryptotextes für eine 0 oder eine 1 im Klartext steht [6].

2.1.2 Integrität (Integretiy)

Der Schutz der Integrität bedeutet, dass der Empfänger einer Nachricht erkennen kann, ob die Nachricht während der Übermittlung modifiziert wurde.

2.1.3 Authentizität (Authentication)

Authentizität verstärkt das Schutzziel der Integrität. Beim Empfang einer authentisierten Nachricht ist der Empfänger in der Lage zu prüfen, ob die Nachricht modifiziert wurde (Integrität) und ob die Nachricht wirklich vom angegebenen Absender stammt.

2.1.4 Frische (Freshness)

Die Zusicherung der Frische einer Nachricht verhindert Replay-Angriffe gegen Netzwerk-Knoten. Beispielsweise wäre es einem Angreifer möglich ein verschlüsseltes und authentisiertes Pakete abzuhören und später erneut ins Netz einzuspielen (injection). Durch die Verwendung von *Nonces* (*Number used once*, relativ großen Zufallszahlen), die in den Paketen mit übertragen werden, werden solche Replay-Angriffe verhindert.

2.2 Symmetrische und asymmetrische Kryptographie

Um die jeweiligen Schutzziele zu erreichen, verwenden die Systeme SPINS, TinySec und TinyPK zwei verschiedene Ansätze. SPINS und TinySec nutzen symmetrische, TinyPK asymmetrische Kryptographie. Auf die mathematischen Hintergründe soll an dieser Stelle nicht näher eingegangen werden. Stattdessen sollen nur die Komponenten erläutert werden, die in den Systemen verwendet werden.

2.2.1 Symmetrische Kryptographie

Symmetrische Kryptographie ist im Gegensatz zur asymmetrischen Kryptographie schneller und benötigt weniger Ressourcen bzw. kürzere Schlüssel. Schlüssellängen über 100 Bit gelten hier noch als sicher [2]. Allerdings ist es bei symmetrischen Verfahren nötig, dass zwei Partner über den selben Schlüssel K verfügen, um kommunizieren zu können. Daraus resultiert ein großes Problem und ein großer Nachteil der symmetrischen Kryptographie: Das Problem der Schlüssel-Verteilung. Es muss sichergestellt werden, dass zwei Partner den Schlüssel auf „sicherem Weg“ bekommen, also so, dass ein Angreifer ihn nicht abfangen bzw. mitlesen kann.

In der symmetrischen Kryptographie wird das Schutzziel der Authentizität mit Hilfe sog. Message Authentication Codes (MACs) erreicht. MACs werden durch kryptographische Hashfunktionen unter Einbeziehung des gemeinsamen Schlüssels erzeugt. Eine kryptographische Hashfunktion h bildet eine beliebig lange Bitfolge B auf eine Bitfolge S fester Länge ab: $S = h(B)$. Dabei werden gleiche Bitfolgen zu gleichen Hashwerten S (siehe auch [5]). Ein MAC ist so nichts weiter als ein Hashwert einer Hashfunktion h bei dem als Argument neben der Bitfolge B noch ein geheimer Schlüssel K verwendet wurde, also $MAC = h(B|K)$. Wenn Sender und Empfänger den Schlüssel K kennen, so kann der Empfänger die Korrektheit der empfangenen Nachricht prüfen, indem er selbst den MAC-Wert der Nachricht berechnet und mit dem im Paket enthaltenen MAC-Wert vergleicht.

SPINS und TinySec nutzen, wie bereits erwähnt, jeweils symmetrische Kryptographie-Verfahren.

2.2.2 Asymmetrische Kryptographie

Asymmetrische Kryptographie ist deutlich rechenintensiver und benötigt längere Schlüssel. Heute gelten 1024 Bit noch als sicher [2]. Jeder Kommunikationsteilnehmer besitzt ein Schlüsselpaar; einen privaten Schlüssel K_{Priv} und einen öffentlichen Schlüssel K_{Pub}. Der private Schlüssel muss geheim bleiben, wohingegen der öffentliche Schlüssel offen im Netz verbreitet werden darf. Wenn zwei Partner A und B kommunizieren möchten, werden vier Schlüssel benötigt: $K_{A,Pub}$, $K_{A,Priv}$, $K_{B,Pub}$ und $K_{B,Priv}$. D.h. für eine gesicherte Kommunikation wird kein gemeinsamer Schlüssel verwendet. Dadurch gibt es bei asymmetrischen Verfahren das Problem der sicheren Schlüsselverteilung nicht. Im weiteren ist es durch die Verwendung dieses Verfahrens auch möglich, das Schutzziel der Verbindlichkeit zu erreichen. D.h. ein Empfänger kann sich sicher sein, dass eine Nachricht wirklich vom angegebenen Absender stammt. Dieses Ziel wird von den vorgestellten Systemen jedoch nicht weiter verfolgt.

Die genannten vier Schlüssel werden für eine gesicherte Kommunikation wie folgt verwendet (vgl. [1, 5]):

- A möchte B eine verschlüsselte Nachricht senden
  - A verschlüsselt die Nachricht mit dem öffentlichen Schlüssel von B $K_{B,Pub}$
  - B empfängt die Nachricht und kann diese *nur* mit seinem privaten Schlüssel $K_{B,Priv}$ entschlüsseln
- A möchte seine Nachricht an B signiert versenden. Durch eine Signatur wird das Schutzziel der Authentizität erreicht
  - A berechnet einen Hashwert S der Nachricht und verschlüsselt diesen mit seinem privaten Schlüssel $K_{A,Priv}$. Dies entspricht der Signatur der Nachricht: $Sig(h; K_{A,Priv})$
  - B empfängt die Nachricht und berechnet ebenfalls den Hashwert h'
  - B „entschlüsselt“ die Signatur mit dem öffentlichen Schlüssel von A $K_{A,Pub}$ und vergleicht $h = h'$? Bei Übereinstimmung wurde die Nachricht nicht modifiziert und stammt von A
- Für die umgekehrte Kommunikation von B nach A erfolgt das Vorgehen analog

TinyPK nutzt mit RSA [11] ist ein bekanntes und weit verbreitetes asymmetrsiches Kryptographie-Verfahren.

3. SPINS – SECURITY PROTOCOLS FOR SENSOR NETWORKS

SPINS wurde 2001 an der „University of California“ entwickelt [10]. Das Ziel von SPINS ist es, eine „viele-zu-einer“-Kommunikation zu ermöglichen. Im Zentrum steht eine Basisstation, die auch als Gateway fungiert und über mehr Ressourcen als ein gewöhnlicher Knoten verfügt. SPINS bietet zwei Protokolle an; SNEP und µTESLA. SNEP bietet Schutz für die bidirektionale Kommunikation zwischen zwei Knoten durch Verschlüsselung, Authentisierung, Integrität und Frische. µTESLA ist eine Abänderung des TESLA-Protokolls [9] und dient dazu authentisierte, aber unverschlüsselte Broadcast-Nachrichten im Sensornetz zu verteilen.

3.1 SNEP – Sensor Network Encryption Protocol

Die Verschlüsselung erfolgt mit dem RC5-Algorithmus [12] im Counter-Mode (CTR). Dies hat mehrere Gründe:

- RC5 ist ein symmetrisches Verfahren. Es ist dadurch schneller und benötigt kürzere Schlüssel als asymmetrische Verfahren [12]
- RC5 verwendet keine S-Boxen für Permutationen und benötigt dadurch weniger Speicher [12]
- Der erzeugte Kryptotext hat die selbe Größe wie der zugehörige Klartext [10], was aufgrund der geringen Bandbreite im Netzwerk vorteilhaft ist

Der Counter-Mode (CTR) funktioniert wie folgt: In mehreren Durchläufen wird aus einem Zähler i und dem symmetrischen Schlüssel K ein temporärer Schlüssel K_i erzeugt. Insgesamt wird ein sog. Schlüsselstrom $K_0, K_1, \ldots, K_n$ generiert. Der Schlüsselstrom wird dann mit dem Klartext mittels XOR verknüpft, wodurch der Kryptotext entsteht (siehe Abb. 1).
Durch Verwendung des CTR wird die sog. *semantic security* Eigenschaft erreicht. D.h. gleiche Klartexte P werden zu verschiedenen Kryptotexten C und C' verschlüsselt.
Um Paket-Overhead einzusparen wird der Zähler-Wert nicht mit jeder Nachricht übertragen, sondern implizit von den kommunizierenden Knoten gespeichert und bei jedem Paket einer Kommunikation zwischen zwei Partnern erhöht.

Die Authentizität, Integrität und Frische von Nachrichten wird mittels MACs erreicht. In die MAC-Berechnung gehen neben der Nachricht und dem gemeinsamen Schlüssel von Sender und Empfänger auch der Zähler-Wert und ggf. eine *Nonce* ein.

3.2 µTESLA

µTESLA ist eine Abwandlung des TESLA-Protokolls [9] und dient dem authentisierten Versenden von Broadcasts. Die Nachrichten-Authentisierung erfolgt dabei wie bei SNEP durch MACs, die in der Nachricht mit übertragen werden.

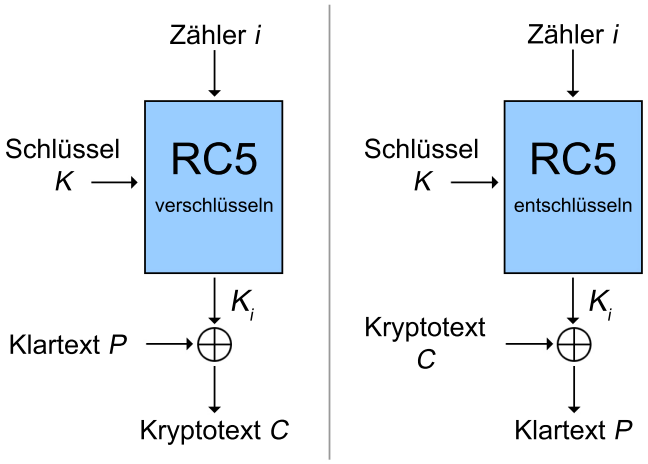


Abbildung 1: Ver- und Entschlüsselung mit RC5 im CTR-Mode

Wie bereits erläutert, geht in den MAC-Wert einer Nachricht ein geheimer Schlüssel ein. Würde für jede Broadcast-Nachricht der selbe Schlüssel verwendet, so könnte ein Angreifer jedoch leicht gültige Broadcasts versenden, sobald er im Besitz des Schlüssels wäre (was z.B. durch einen physikalischen Zugriff auf einen Knoten erfolgen könnte). Zur Lösung dieses Problems arbeitet µTESLA wie auch TESLA mit einer *key chain*.

Der Sender von Broadcast-Nachrichten erzeugt zunächst einen Zufalls-Bytestrom K_n. Anschließend wird darauf eine Hashfunktion h angewandt, womit ein anderer Pseudo-Zufallswert K_{n-1} erzeugt wird: $K_{n-1} = h(K_n)$ Durch wiederholtes anwenden der Hashfunktion auf den jeweils zuvor erzeugten Wert wird in jedem Durchlauf ein neuer Bytestrom erzeugt: $K_{n-1}, \ldots, K_2, K_1, K_0$. Diese Folge von Byteströmen entspricht der *key chain* und die Einträge K_i entsprechen den Schlüsseln, die in die spätere MAC-Berechnung der Broadcasts eingehen. Diese erzeugten Schlüssel sind zunächst nur dem Sender bekannt.

Nun wird die Zeit in Epochen $0, 1, 2 \ldots, n-1, n$ eingeteilt. Jeder Epoche i wird dabei ein Schlüssel K_i zugeteilt. Schickt der Sender in Epoche i eine Nachricht, so berechnet er den zugehörigen MAC mit dem Schlüssel K_i. Die Schlüssel der *key chain* werden also in der Reihenfolge n bis 0 erzeugt und in der Reihenfolge 0 bis n verwendet (siehe Abb. 2).

Empfängt ein Knoten eine Broadcast-Nachricht, so ist er zunächst nicht in der Lage die Nachricht zu authentisieren, da er den Schlüssel (der aktuellen Epoche) nicht kennt. Er muss die Nachricht daher zwischenspeichern. Nach Ablauf der Epoche i, in der die Nachricht gesendet wurde, teilt der Sender den Empfängern den Schlüssel K_i der letzten Epoche mit. Durch Erhalt des Schlüssels kann der Empfänger dann die gespeicherte Nachricht authentisieren. In einer Epoche können auch mehrere Nachrichten versendet werden. Der Empfänger muss dann alle Nachrichten speichern, bis der zugehörige Schlüssel gesendet wird. Aufgrund der knappen Ressourcen der Knoten darf eine Epoche nicht zu lange dauern, da ein Empfänger sonst nicht alle Nachrichten puffern kann. Außerdem müssen Sender und Empfänger

über „schwach synchronisierte“ Uhren verfügen [10].

Die Sicherheit dieses Verfahrens ist durch die Einweg-Eigenschaft der Hashfunktion gegeben: Der Sender berechnet den MAC in Periode i mit K_i. Wenn ein Angreifer gültige MAC-Werte berechnen wollte, so müsste er aus dem Schlüssel der vorherigen Epoche K_{i-1} den aktuellen Schlüssel K_i berechnen: $K_i = h^{-1}(K_{i-1})$. Das ist aber gerade die Umkehrung einer kryptographischen Hashfunktion, die nicht möglich ist. Abbildung 2 stellt die Erzeugung und Verwendung der *key chain* nochmals graphisch dar.

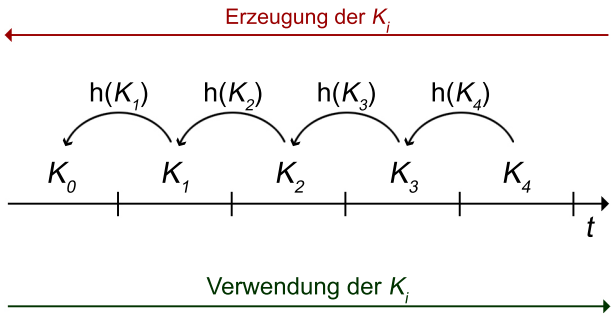


Abbildung 2: Erzeugung und Verwendung der K_i in der *key chain*

Wegen der knappen Ressourcen der Knoten können diese nach [10] selbst keine *key chain* berechnen und speichern. Diese Funktion bleibt der Basisstation vorbehalten. Muss ein Knoten einen Broadcast senden, so beschreibt [10] dazu zwei Möglichkeiten:

1. Der Knoten sendet die Broadcast-Nachricht mittels SNEP verschlüsselt an die Basisstation, welche dann einen Broadcast erzeugt
2. Der Knoten sendet den Broadcast. Die zugehörige *key chain* wird aber von der Basisstation erzeugt und verteilt. Vor dem Senden eines Broadcasts teilt die Basisstation dem Broadcast-Knoten den aktuellen Schlüssel via SNEP mit

4. TINYSEC

Wie in [6] kritisiert, wurde SPINS weder komplett spezifiziert noch vollständig implementiert. Außerdem ist SPINS wegen der Verwaltung von Zählern für jede Kommunikation zwischen Paaren von Knoten relativ komplex.
TinySec [6] erweitert ebenfalls TinyOS um kryptographische Funktionen. Es ist vollständig implementiert und kann zusammen mit TinyOS im Internet heruntergeladen werden. TinySec wurde 2004 an der „UC Berkeley“ vorgestellt. Das Ziel von TinySec ist es, eine sichere Kommunikation zwischen zwei beliebigen Knoten im Sensornetz zu ermöglichen. Eine Basisstation gibt es nicht. TinySec bietet drei mögliche Kommunikationsprotokolle an:

1. Klartext: unsichere Kommunikation mittels Nachrichten wie von TinyOS definiert (Kompatibilität zu TinyOS)
2. TinySec-Auth (authenticated): Nachrichten sind unverschlüsselt, aber authentisiert
3. TinySec-AE (authenticated encryption): Nachrichten sind verschlüsselt und authentisiert

4.1 TinySec-Auth

Eine Nachricht besteht aus einem 5 Byte Header, 0-29 Byte Nutzdaten, sowie einem 4 Byte Trailer. Der Header besteht aus drei Feldern: Zieladresse (Dest), Nachrichten-Typ (AM) und Nachrichtenlänge (Len). Der Trailer enthält den 4 Byte langen MAC. In den MAC geht neben den Nutzdaten auch der Header ein (siehe Abb. 3).

4.2 TinySec-AE

Die Nachricht besteht hier aus einem 8 Byte Header, ebenfalls 0-29 Byte Nutzdaten und ebenfalls einem 4 Byte Trailer für den MAC. Der Header enthält hier fünf Felder: Empfänger-Adresse (Dest), Nachrichten-Typ (AM), Länge (Len), Sender-Adresse (Src) und Counter (Ctr). Der Nutzdatenteil der Nachricht ist verschlüsselt (siehe Abb. 3).

4.3 Verschlüsselung

TinySec nutzt den Skipjack Algorithmus im Cipherblock-Chaining Mode (CBC). Im Gegensatz zu RC5, der in SNEP verwendet wird, ist Skipjack nicht patentiert (vgl. [6]).

Der CBC-Modus funktioniert wie folgt: Der Klartext wird in Blöcke gleicher Größe aufgeteilt: $P_0, P_1, \ldots, P_n$. Ein Block P_i wird unter Verwendung des Schlüssels K und eines zuvor erzeugten Kryptotextblocks C_{i-1} zu einem Kryptotextblock C_i verschlüsselt (siehe Abb. 4).
Bei der Verschlüsselung des ersten Blocks P_0 gibt es noch keinen Kryptotextblock, der in die Verschlüsselung eingehen könnte. Daher wird ein sog. Initialisation Vector (IV) verwendet.

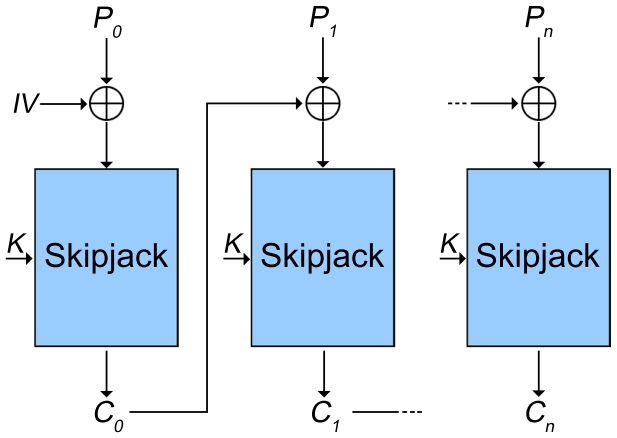


Abbildung 4: Verschlüsselung im CBC-Modus mit Skipjack

Der IV ist nicht geheim, sollte aber möglichst zufällig und einmalig sein. Außerdem sollte sich der IV nicht wiederholen, da es dann möglich ist, dass gleiche Klartexte zu gleichen Kryptotexten verschlüsselt werden.
Damit der Empfänger einer Nachricht diese entschlüsseln

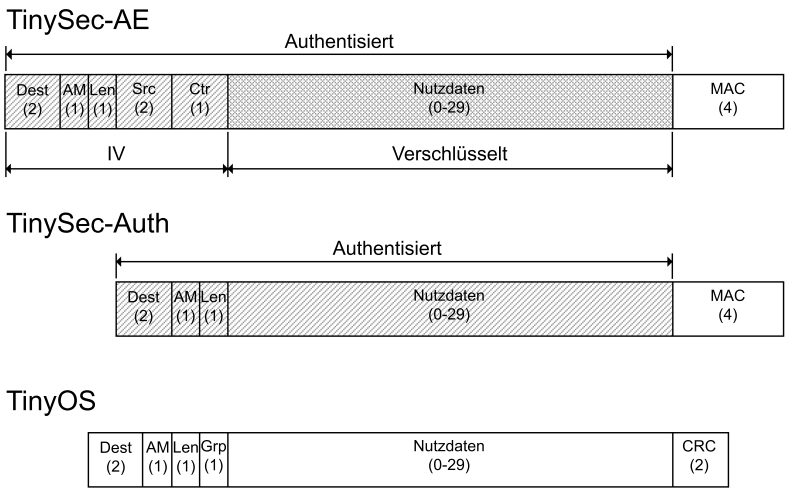


Abbildung 3: Nachrichten-Formate für TinySec-AE, TinySec-Auth und TinyOS [6]

kann, muss er den IV kennen. Da der IV aber möglichst groß sein sollte (um eine 2-malige Verwendung zu vermeiden), ergibt sich das Problem, dass viele zusätzliche Bytes für jedes Paket übertragen werden müssen. In TinySec-AE wird das Problem gelöst, indem der IV wie folgt zusammengesetzt wird: IV=Dest|AM|Len|Src|Ctr. Diese Werte entsprechen genau den Header-Feldern von TinySec-AE Nachrichten: Ziel-Adresse, Nachrichten-Typ, Länge, Quell-Adresse, Counter.
TinySec-AE ist hier effizient, da es für den IV die Felder verwendet, die ohnehin in jedem Nachrichten-Header enthalten sein müssen (außer Counter). Counter dient als fortlaufende Nummer dazu, eindeutige IVs zu erzeugen und ist zwei Byte groß. Es müssen für den IV also nur zwei zusätzliche Byte übertragen werden, obwohl der IV insgesamt 8 Byte lang ist.

4.4 Authentisierung

Für die Authentisierung von Nachrichten wird auch unter TinySec ein MAC verwendet. Zur Berechnung des MAC wird jedoch nicht extra eine kryptographische Hashfunktion wie MD5 oder SHA-1 verwendet, sondern der bereits implementierte Verschlüsselungs-Algorithmus Skipjack im CBC-Modus. Der Vorteil dieser Vorgehensweise wird durch die Code-Einsparung begründet [6].

Die Generierung von MACs durch eine CBC-Verschlüsselung funktioniert prinzipiell wie folgt: Die Nachricht P wird wie bei der Verschlüsselung in Teilblöcke $P_0, P_1, \ldots, P_n$ zerlegt. Die Blöcke werden nacheinander zu den C_i verschlüsselt. Die Kryptotexte C_0 bis C_{n-1} werden abschließend verworfen. C_n stellt den MAC-Wert der Nachricht P dar.

Da für Verschlüsselung und Authentisierung aus Sicherheitsgründen zwei verschiedene Schlüssel verwendet werden (jeder Knoten ist im Besitz dieser zwei Schlüssel), kann der C_n Block der durch die Verschlüsselung entsteht, nicht als MAC-Wert verwendet werden. Entsprechend muss der Verschlüsselungs-Algorithmus bei Tiny-AE-Nachrichten zwei mal durchlaufen werden, was relativ viel Energie benötigt.

5. TINYPK

Die bereits vorgestellten Systeme SPINS und TinySec benutzen symmetrische Kryptographie um eine gesicherte Kommunikation zu ermöglichen. TinyPK implementiert mit RSA [11] ein bekanntes asymmetrisches Kryptographie-Verfahren und wurde 2004 von „BBN Technologies“ vorgestellt [13]. TinyPK verfolgt drei Ziele:

1. Ein außenstehendes Gerät, die *External Party* (EP), die zunächst nicht zum Sensornetz gehört, zu authentisieren
2. Der EP sicher den symmetrischen Schlüssel K zu übermitteln, der im Sensornetz zur gesicherten Kommunikation genutzt wird (z.B. von TinySec)
3. Einen Knoten aus dem Netz gegenüber der EP zu authentisieren

Auf den Knoten sind nur die Public-Key-Operationen *Verschlüsselung* und *Prüfen einer Signatur* implementiert, da diese im Vergleich zu den Private-Key-Operationen *Entschlüsseln* und *Erzeugen einer Signatur* relativ wenig Ressourcen benötigen [13].

5.1 Verwendete Techniken

TinyPK verwendet zwei Techniken, die in Netzen mit leistungsstärkeren Clients ebenfalls weit verbreitet sind und die im folgenden kurz erläutert werden sollen.

5.1.1 Certification Authority (CA)

Eine CA ist eine Instanz, der ein Client bzw. Nutzer vertraut. Eine CA bürgt mittels Zertifikat dafür, dass ein öffentlicher Schlüssel zu einer bestimmten Partei gehört, beispielhaft: „Ich, die CA XYZ, versichere, dass $K_{A,Pub}$ dem Benutzer A gehört.“ Diese Zusicherung wird in Form von Zertifikaten, wie sie auch in SSL/TLS verwendet werden, ausgedrückt. Ein Zertifikat ist im wesentlichen eine Datenstruktur, die den Namen einer Partei, deren Public-Key und

eine Signatur enthält. Die Signatur ist von der CA ausgestellt. Mit Hilfe des Public-Key der CA kann die Signatur und damit die Echtheit des Zertifikates überprüft werden.

5.1.2 Diffie-Hellman Verfahren (DH)

Das Diffie-Hellman Verfahren dient dazu, einen gemeinsamen Schlüssel zwischen zwei Teilnehmern A und B zu etablieren, ohne dass der Schlüssel dabei übertragen wird. Dies ist vor allem dann nötig, wenn der Schlüssel über einen potentiell (kryptographisch) unsicheren Kanal ausgehandelt werden sollen. Das Verfahren ist weit verbreitet und gut dokumentiert, wie [5] entnommen werden kann.

5.2 Authentisierung einer External Party

Es werden folgende Annahmen und Voraussetzungen gemacht (vgl. [13]):

- Die Sensorknoten kommunizieren untereinander gesichert mittels symmetrischer Kryptographie-Verfahren, z.B. mit TinySec. Damit kennen alle, oder zumindest einige, einen gemeinsamen symmetrischen Schlüssel K
- Die Knoten kennen eine Certification Authority (CA), der sie vertrauen. Die Knoten sind im Besitz des öffentlichen Schlüssels der CA $K_{CA,Pub}$
- Die External Party kann zu Beginn nicht mit dem Netz kommunizieren, da sie den Schlüssel K nicht kennt
- Die EP kann — im Gegensatz zu den Knoten — auch Private-Key Operationen durchführen, da sie über mehr Rechenleistung verfügt
- Die EP besitzt eine Signatur $Sig(K_{EP,Pub}; K_{CA,Priv})$ ihres öffentlichen Schlüssels $K_{EP,Pub}$, die von der CA ausgestellt, (d.h. von der CA mit $K_{CA,Priv}$ signiert, wurde

Die Authentisierung der EP läuft wie folgt ab: Die EP sendet zwei Nachrichten:

1. Die Signatur ihres Public Key $Sig(K_{EP,Pub}; K_{CA,Priv})$
2. Eine Signatur $Sig((Nonce, Checksum); K_{EP,Priv})$ über eine *Nonce* und eine Prüfsumme von $K_{EP,Pub}$. Diese Signatur wurde nicht von der CA, sondern von der EP erstellt

Der Knoten „entschlüsselt“ die erste Signatur mit dem öffentlichen Schlüssel der CA $K_{CA,Pub}$. Anschließend ist der Knoten im Besitz des öffentlichen Schlüssels der EP $K_{EP,Pub}$. Mit diesem kann der Knoten nun die zweite Signatur „entschlüsseln“ und erhält so die *Nonce* und die Prüfsumme. Die *Nonce* dient dazu eine Replay-Attacke zu erkennen, womit das Schutzziel der Frische erreicht wird. Verläuft die Prüfung positiv, ist die EP authentisiert.

5.3 Übermittlung des Sensornetzschlüssels

Der Knoten sendet nun eine Antwortnachricht: $Enc((Nonce, K); K_{EP,Pub})$. Diese enthält den symmetrischen Schlüssel K und die empfangene *Nonce*. Die Nachricht ist mit dem öffentlichen Schlüssel der EP $K_{EP,Pub}$ verschlüsselt.

5.4 Authentisierung eines Knoten

Da die Knoten keine Private-Key Operationen und somit auch keine Signaturberechnungen durchführen können, besitzt jeder Knoten einige statische Informationen, die zur Authentisierung verwendet werden:

- Einen ID-Text, der in bestimmter Formatierung unter anderem folgende Informationen enthält: Serien-Nummer, Konstruktionsdatum, Initial-Daten
- Einen statischen, öffentlichen Diffie-Hellman-Wert $DH_{Node,Pub}$
- Eine Signatur des Diffie-Hellman-Wertes, die von der CA ausgestellt wurde: $Sig(DH_{Node,Pub}; K_{CA,Priv})$

Die EP sendet nun erneut eine Nachricht an den Knoten, diesmal jedoch nicht im Klartext, sondern verschlüsselt im TinySec-AE-Format: $Enc(Nonce, DH_{EP,Pub}; K)$. Die Nachricht enthält eine *Nonce* und einen temporären DH-Wert $DH_{EP,Pub}$. Der Knoten empfängt die Nachricht und berechnet aus den DH-Werten einen neuen Schlüssel K_{DH}. Anschließend berechnet er einen MAC-Wert aus der empfangenen *Nonce* und dem neu erstellten Schlüssel K_{DH}: $MAC(Nonce; K_{DH})$. Der MAC-Wert wird zusammen mit dem signierten statischen DH-Wert des Knotens $Sig(DH_{Node,Pub}; K_{CA,Priv})$, sowie dem ID-Text zurück gesandt.

Die EP empfängt die Antwort und ist damit im Besitz des öffentlichen DH-Schlüssels des Knotens $DH_{Node,Pub}$. Damit kann die EP ebenfalls den Schlüssel K_{DH} berechnen und mit diesem den empfangenen MAC-Wert prüfen. Verläuft die Prüfung positiv, wird der ID-Text des Knoten zusammen mit dem erzeugten DH-Schlüssel K_{DH} in einer Datenbank gespeichert.

Sendet zukünftig ein Sensornetz-Knoten Daten an die EP, muss er für jede Nachricht einen MAC berechnen und mitsenden, in den der vereinbarte DH-Schlüssel K_{DH} eingeht. Die EP prüft dann die MAC anhand des DH-Schlüssels. Ist der MAC gültig, ist die Nachricht bzw. der Absender authentifiziert und die Nachricht wird akzeptiert. Abbildung 5 stellt den Protokollablauf nochmals graphisch dar.

6. VERGLEICH UND BEWERTUNG

Die vorgestellten Systeme SPINS, TinySec und TinyPK verwenden bzw. erweitern das Open-Source-Betriebssystem TinyOS, das an sich keine Sicherheitsmechanismen für die Übertragung bietet. Alle drei Systeme verfolgen verschiedene Ziele. SPINS und TinySec implementieren dazu die effizienteren symmetrischen Kryptographie-Verfahren, während TinyPK asymmetrische Verfahren umsetzt.

SPINS war einer der ersten Versuche um die Kommunikation in Sensornetzen zu schützen. Der Schwerpunkt liegt in der sicheren Kommunikation zwischen einem Knoten und einer Basisstation (SNEP), bzw. in der gesicherten Übertragung von Broadcasts (µTESLA). TinySec wurde später entwickelt. Der Schwerpunkt liegt hier in der gesicherten Kommunikation zwischen beliebigen Knoten. Das Konzept einer Basisstation bzw. von Broadcast-Nachrichten fehlt. In

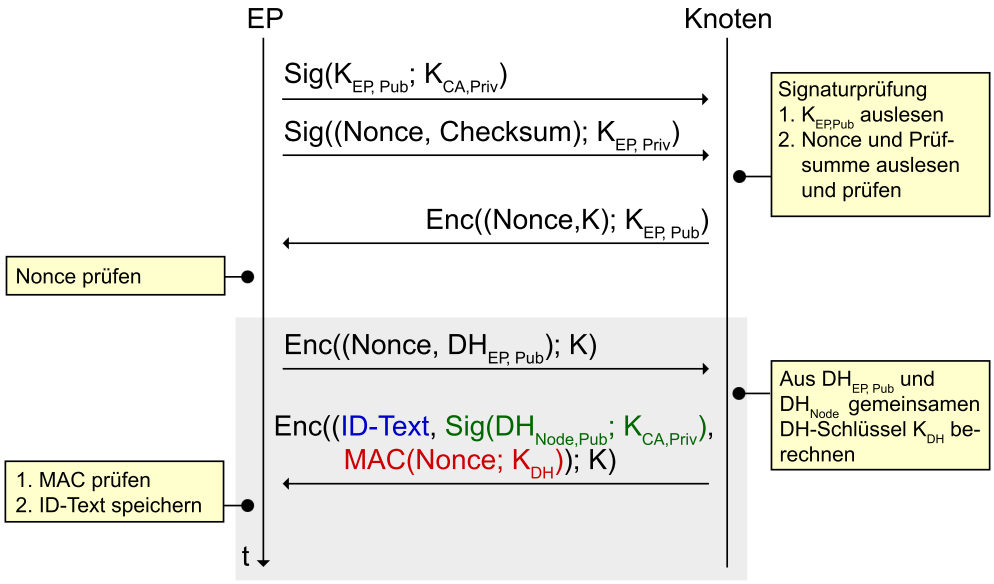


Abbildung 5: Protokollablauf der TinyPK Authentisierungen

SPINS besitzt jeder Knoten einen eigenen Schlüssel, welcher der Basisstation bekannt ist. In TinySec teilen sich alle Knoten eines Netzes einen gemeinsamen Schlüssel.

SPINS und TinySec verwenden jeweils einen Zähler, der in die Verschlüsselung bzw. den MAC eingeht. SPINS verwaltet diese Zähler lokal auf jedem Knoten. D.h. für jede Kommunikation, die ein Knoten mit einem anderen Knoten, bzw. der Basisstation, betreibt, muss er einen Zähler-Wert speichern und verwalten (stateful Protokoll). Es sei jedoch angemerkt, dass es die Grundidee von SPINS ist, eine gesicherte Kommunikation von Knoten zu Basisstation bzw. umgekehrt zu ermöglichen. Es ist zwar auch möglich, dass zwei Knoten direkt miteinander „sprechen“, das ist jedoch nicht der Schwerpunkt des Konzepts.
Anders hingegen in TinySec. Hier soll jeder Knoten mit jedem anderen Knoten sicher kommunizieren können. Ein Knoten speichert den Zähler für den Nachrichten-Schutz daher nicht lokal, sondern überträgt ihn im Header jeder Nachricht, wodurch der Aufwand zur Verwaltung der Zähler-Werte entfällt. Dadurch wird allerdings für jedes Paket ein größerer Overhead übertragen, was mehr Energie benötigt und somit die Laufzeit der Knoten verkürzt.
Die Authentisierung bzw. der Integritätsschutz von Nachrichten erfolgt in SPINS wie in TinySec analog. Beide Systeme verwenden MACs die in jeder Nachricht mit übertragen werden.
SPINS bietet gegenüber TinySec mit µTESLA die Möglichkeit authentisierte Broadcast-Nachrichten zu versenden. Diese Funktion ist in TinySec nicht implementiert.

Für die „Authentisierung“ wird in TinySec ein MAC-Wert genutzt, der durch einen Schlüssel berechnet wird, der im Netz allgemein bekannt ist. Dadurch kann ein Empfänger sicherstellen, dass die Nachricht während der Übertragung nicht verändert wurde. Ein Empfänger kann aber nicht nachvollziehen, ob die Nachricht wirklich von einer vertrauenswürdigen Quelle stammt. Jeder Knoten, der im Besitz des Schlüssels ist, kann „authentisierte“ Nachrichten versenden. TinySec erreicht daher zwar das Schutzziel der Integrität, das der echten Authentizität aber nicht.
Im weiteren wird in TinySec-Auth Nachrichten kein Wert übertragen, mit dem ein Replay-Angriff erkannt werden könnte. Entsprechend wird mit TinySec-Auth das Schutzziel der Frische nicht erreicht.

Sowohl SPINS als auch TinySec verwenden ausschließlich symmetrische Kryptographie mit der Begründung asymmetrische Verfahren benötigen zu viele Ressourcen [10, 6]. Daher wird in beiden Systemen ein gemeinsamer Schlüssel für die Kommunikation benötigt. In TinySec teilen sich alle Knoten den selben Schlüssel. In SPINS teilt sich jeder Knoten einen Schlüssel mit der Basisstation. Es gibt in keinem der beiden Systeme die Möglichkeit Schlüssel automatisch zu verteilen. Das stellt einen großen Nachteil dar, da ein Schlüssel nur für eine begrenzte Zeit verwendet werden sollte. Im Idealfall wird für jede neue Kommunikation zwischen zwei Partnern ein neuer Schlüssel verwendet. Da die Systeme SPINS und TinySec keine automatische Schlüsselverteilung vorsehen, müsste ein neuer Schlüssel durch physikalischen Zugriff auf den oder die jeweiligen Knoten verteilt werden. Dieses Vorgehen ist aufwendig und wird folglich nicht oft durchgeführt werden.

Das Primäre Ziel von TinyPK ist nicht die sichere Kommunikation zwischen Knoten zu ermöglichen, sondern eine Authentisierung zwischen einer External Party (EP), die nicht zum Sensor-Netz gehört, und einem Sensornetz durchzuführen. Nach erfolgreicher Authentisierung wird der symmetrische Schlüssel K, der vom Sensornetz für die sichere Kom-

munikation verwendet wird, zur EP übertragen. TinyPK ist also als Erweiterung für andere Systeme wie TinySec gedacht. TinyPK nutzt eingeschränkt asymmetrische Kryptographie, benötigt zur Vertrauensbildung jedoch eine Certification Authority (CA). Die External Party muss — ähnlich wie die Basisstation in SPINS — über mehr Rechenleistung als ein gewöhnlicher Knoten verfügen, da sie auch in der Lage sein muss, Private-Key-Operationen durchzuführen.

7. ANGRIFFSMÖGLICHKEITEN

Alle vorgestellten Systeme arbeiten mit Funk-Kommunikation. Entsprechend sind alle Systeme durch gezielte Störung der jeweiligen Frequenzen angreifbar (DoS-Angriff). Schutz vor dieser Angriffsart wird durch die vorgestellten Systeme explizit nicht adressiert.

In TinySec teilen sich alle Knoten einen gemeinsamen Schlüssel zur sicheren Kommunikation. Das Vertrauen der Knoten untereinander basiert lediglich auf dem Besitz des Schlüssels. Damit hängt die Sicherheit des Systems stark vom physikalischen Schutz der einzelnen Knoten ab. Schafft es ein Angreifer den Schlüssel aus einem Knoten auszulesen, ist das gesamte System ausgehebelt und wirkungslos.

In SPINS tritt dieses Problem zunächst nicht so drastisch auf, da jeder Knoten einen eigenen Schlüssel besitzt, den außer ihm nur die Basisstation kennt. Erlangt ein Angreifer physikalischen Zugriff auf einen Knoten um so den Schlüssel auszulesen, ist zunächst nicht die Kommunikation des gesamten Netzes gefährdet. Allerdings ist ein Angreifer in der Lage die Kommunikation zwischen Basisstation und dem kompromittierten Knoten zu stören bzw. abzuhören. Im Weiteren ist es einem Angreifer auch möglich, sich als Knoten gegenüber der Basisstation auszugeben (spoofing). Auf diesem Weg kann ein Angreifer dann nicht nur gefälschte Nachrichten an die Basisstation senden, sondern über diese auch Broadcast-Nachrichten an das gesamte System verteilen.
Gelingt es einem Angreifer die Basisstation zu kompromittieren, so ist es prinzipiell auch möglich die Schlüssel aller Knoten auszulesen. Damit wäre das System wirkungslos.

Ein weiterer Angriff auf SPINS-Knoten ist möglich, wenn sich ein Angreifer als Basisstation ausgibt und mittels µTESLA Broadcast-Nachrichten sendet. Knoten müssen diese Nachrichten solange zwischenspeichern, bis sie einen Schlüssel erhalten haben, um die Nachrichten nachträglich zu authentisieren. Ein Angreifer kann hier viele Nachrichten senden, ohne je einen Schlüssel zu verteilen. Die Knoten werden so mit Paketen geflutet (flood-Angriff).

8. ZUSAMMENFASSUNG

Sichere Kommunikation in Sensor-Netzen ist trotz geringer Bandbreite und geringen Ressourcen möglich. Die vorgestellten Systeme SPINS, TinySec und TinyPK verfolgen unterschiedliche Ziele und erreichen diese mit unterschiedlichen Methoden. SPINS geht davon aus, dass im Sensornetz viele Knoten zu einer Basisstation „sprechen“, bzw. umgekehrt. TinySec kennt keine zentrale Basisstation. Hier sollen alle Knoten untereinander sicher kommunizieren können. SPINS und TinySec verwenden ähnliche Lösungsansätze mit symmetrischen Kryptographie-Verfahren. TinyPK implementiert Teile von Public-Key Verfahren auf Knoten, benötigt jedoch immer einen Kommunikationspartner (EP), der über mehr Rechenleistung als ein gewöhnlicher Knoten verfügt. Das Ziel von TinyPK ist die sichere Authentisierung einer EP gegenüber dem Sensornetz und umgekehrt. TinyPK kann daher beispielsweise als Erweiterung zu TinySec verwendet werden.

9. LITERATUR

[1] *M 3.23 Einführung in kryptographische Grundbegriffe*, `https://www.bsi.bund.de/` Bundesamt für Sicherheit in der Informationstechnik (BSI), 2006.

[2] *G 4.35 Unsichere kryptographische Algorithmen*, `https://www.bsi.bund.de/` Bundesamt für Sicherheit in der Informationstechnik (BSI), 2009.

[3] H. Zimmermann. OSI Reference Model-The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.

[4] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol - Version 1.2. RFC 5246, IETF, August 2008.

[5] C. Eckert. *IT-Sicherheit*. Oldenbourg Wissensch.Vlg, 2006.

[6] C. Karlof, N. Sastry, and D. Wagner. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks In *SenSys*, Baltimore, Maryland, USA, November 2004.

[7] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, IETF, November 1998.

[8] 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 *Ambient Intelligence*. Springer Berlin / Heidelberg, 2005.

[9] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The TESLA Broadcast Authentication Protocol. *CryptoBytes*, 5(2):2–13, 2002.

[10] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. In *Mobile Computing and Networking*, Rome, Italy, 2001.

[11] R. Rivest, A. Shamir, and L. Adleman. A Method for obtaining Digital Signatures and Public Key Cryptosystems. *Communication of the ACM*, 21(2):120–126, 1978.

[12] R. L. Rivest. *Fast Software Encryption*, volume 1008 of *Lecture Notes in Computer Science (LNCS)*, chapter The RC5 Encryption Algorithm, pages 86–96. Springer Berlin / Heidelberg, 1995.

[13] R. Watro, D. Kong, S. fen Cuti, C. Gardiner, C. Lynn, and P. Kruus. TinyPK: Securing Sensor Networks with Public Key Technology. In *SASN*, Washigton, DC, USA, October 2004.

Sensornetworks: Integration into IPv6 based networks and security on layer 3

Andreas Scheibleger
Betreuer: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: scheible@in.tum.de

ABSTRACT
Sensor networks are becomming more and more important. They are no more only used internally in companies but get gradually connected to the internet. As there are that much sensor nodes it is sensible to use the IPv6 protocol, since it provides a 128 bit address space. This technique however needs some adaptation on the link layer as well as on the network layer. Therefore there arise the same security risks which need to be solved as on typical stationary devices. Solving the integration and adding security on the network layer is going to be discussed in this paper.

Keywords
IPv6, 6LoWPAN, MiniSec, Security, IEEE 802.15.4

1. INTRODUCTION
Sensor nodes get more and more connected to already established conventional networks. For that to work there was the need to develop methods, which allow the usage of standard protocols on devices which have limited memory and processor resources. In favor of that the "IPv6 over Low power Wireless Personal Area Networks" (6LoWPAN [9]) architecture was developed. This architecture describes a mechanism to encapsulate the data and compress the headers in order to send and receive IPv6 packets over a IEEE802.15.4 [5] network. In the area of sensor networks the usual security criterias, namely "confidentiality", "integrity" and "accountability" need to be guaranteed as well as the criteria of "freshness". In addition there is the need for low energy consumption. For this purpose the "MiniSec" [8] protocol got developed. Those two topics are going to be covered in the following.

2. CLASSIFICATION
This document refers to security on layer 3 whereas layer 3 in the OSI-model is meant. To make it clear the model is shown in fig.1. By that one can see that it is not about how packets are sent or received, but the type or content of the packet is of interest. The network layer, loosely speaking, is responsible for accepting packets destined for the device and for forwarding packets if the current device is used as an intermediate node, i.e. routing.
The first part of this document deals with integration of sensor nodes into an IPv6 network. IPv6 (Internet Protocol version 6) is settled as well on this layer, as is the actually more widely used IPv4.

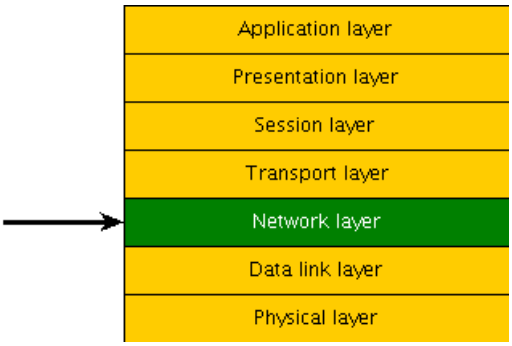


Figure 1: OSI-model

3. 6LOWPAN
3.1 Motivation
By extending LoWPAN or 802.15.4 [5] based networks so that it is possible to transmit encapsulated IPv6 [4] packets, one can make a fully into the IP network integrated device out of a simple sensor node. The problem is that the maximum packet size for 802.15.4 is limited on the physical layer (see fig.1) to 127 bytes. The maximum overhead for the frame and for the security on the link layer amount 25 and 21 bytes, respectively. With a default size of 40 bytes for the IPv6 header and 8 bytes for the UDP header, there only remain 33 bytes for the payload. Thus it becomes clear that a procedure must be introduced to shrink the large IPv6 header as well as the UDP header in order to transmit the packet over a LoWPAN network. In addition this modified packet needs to be encapsulated into the existing LoWPAN packet. For that an adaptation layer is inserted at first which is needed for mesh-routing, fragmentation and message identification. In connection it will be shown how in accordance to RFC4944 [9] the IPv6 header can be compressed in the extreme case from 40 down to 2 bytes and the UDP header from 8 down to 4 bytes. Further how stateless address autoconfiguration is done in order for every node being reachable via IPv6.

3.2 Adaptation header
Fig.2 shows a full adaptation header including the fields for mesh-routing as well as the fields for fragmentation. Those are optional and are only used if necessary to avoid unnecessary overhead.

Figure 2: full adaptation header

Mesh type: This type field with the corresponding header field should only be inserted if the connection is not a direct one, but needs mesh-routing. The mesh-type starts as shown in fig.3 with the bit sequence 10 followed by two bits (V, F) which indicate the adress type. Subsequently a 4 bit hops-left counter which is decreased at every hop is appended. If this counter reaches a value of 0 the packet should be dropped. V specifies the address type of the originator

Figure 3: Mesh type and header

address and should be set to 0 if it is a IEEE extended 64-bit address or otherwise to 1 in case of a short 16-bit address. F is used the same way for the destination address.

fragment type: If the whole payload fits into a single 802.15.4 frame, this type field and its corresponding header field should be left away. In case of needed fragmentation the header of the first fragment should be built according to fig.4, the header of the subsequent fragments according to fig.5. The datagram size is 11 bits long and should be the

Figure 4: First fragment

Figure 5: Subsequent fragments

same for all correlated fragments indicating the size of the whole packet. The tag is 16 bits wide and should be the same for all correlated fragments, either. It should be reset to zero if it reaches the maximum possible value. The 8 bit wide offset is present with all fragments but the first one. The offset for the first fragment is implicitly set to 0.

Dispatch type: This field is mandatory and starts as shown in fig.6 with the bit sequence 01 follwed by a 6 bit wide selector which indicates the header type of the subsequent header. Thereby the values for the selector are defined as listed in table 1.

Figure 6: Dispatch type and header

00 xxxxxx	NALP	The following data is not part of the LoWPAN encapsulation
01 000001	IPv6	An uncrompressed IPv6 header is next
01 000010	LOWPAN_HC1	A LOWPAN_HC1 compressed header is next
01 ...	reserved	
01 010000	LOWPAN_BC0	A LOWPAN_BC0 header for mesh broadcasting/multicasting is next
01 ...	reserved	
01 111111	ESC	Another 8 bit selector is following in order to have values greater than 63

Table 1: Selector values

3.3 Address autoconfiguration

The interface identifier for an IEEE 802.15.4 interface cannot directly be used as interface identifier for IPv6 as this one is 2 bytes longer. After optaining such a compatible interface identifier (see 3.3.1) we can generate a IPv6 link local address (see 3.3.2).

3.3.1 Interface identifier

IEEE 802.15.4 devices may have an IEEE EUI-64 address or a short 16 bit address. If it is a short 16 bit address then one has to derive a IEEE EUI-64 bit address out of this very short address. This is achieved by firstly generating a 32 bit prefix which is built out of the concatenation of the 16 bit PAN ID (Personal area network Identifier) and a sequence of 16 zero bits. This prefix is prepended to the 16 bit short address which gives a 48 bit pseudo interface identifier. This is then equally long as a default built in ethernet interface identifier. In accordance with RFC2464 [3] this 48 bit long address is then enlarged to an EUI-64 address as described in the following. Let the 48 bit address be the one shown in fig.7. To form the EUI-64 interface identifier

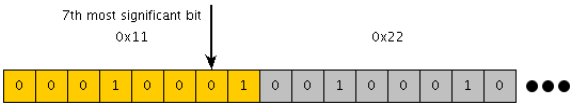


Figure 7: original interface identifier (11:22:33:44:55:66)

needed, the first three bytes are used as company identifier and are copied into the EUI-64 identifier. The next two bytes are statically set to FFFE hexadecimal. Afterwards the last three bytes are appended to the EUI-64 identifier. The last thing to be done is to set the Universal/Local bit to 1 as this derived EUI-64 identifier is in general not universally unique. This bit is the seventh most significant bit. Applying this we get the following:

13:22:33:FF:FE:44:55:66

Now that we either have a EUI-64 built in address or a derived one we can go further an build the interface identifier

out of it. This is simply achieved by complementing the Universal/Local bit. This needs to be done as global uniqueness in EUI-64 is indicated by a 0 and global uniqueness in the IPv6 interface identifier is signified by a 1. To go on with the example above we end up with the following IPv6 interface identifier:

11:22:33:FF:FE:44:55:66

3.3.2 *IPv6 link local address*

Now having obtained the interface identifier as described in 3.3.1 we can form the IPv6 link local address out of it to the prefix FE80::/64 which identifies an IPv6 link local address. It is built as shown in fig.8 by appending the 64 bit interface identifier to the already mentioned IPv6 link local prefix FE80::/64.

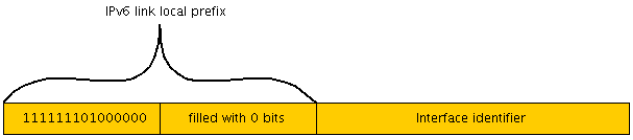


Figure 8: IPv6 link local address

3.4 Header compression

Given the assumption that the LoWPAN devices are in the same network they share some information that is not needed to be transmitted, so these informations can be left away in order to compress the IPv6 header. So the following fields can be left away or considered to be compressable:

- Version: is automatically set to IPv6
- both IPv6 source and destination addresses are considered to be link local and in addition can be derived from the link layer interface identifiers.
- packet length can be inferred either by the datagram_size in the fragmentation header or by the frame length on the link layer.
- traffic class and flow label are considered to be zero
- the next header is UDP, ICMP or TCP

So the only field that needs to be transmitted as a whole is the hop limit. This is of course the best case scenario. For example by changing the link layer interface identifier by software for security reasons the source and destination addresses cannot be derived that easily. Packets following the assumption to be compressible can be compressed via the LOWPAN_HC1 format by using a dispatch-type of LOWPAN_HC1 followed by a "HC1 encoding" field as shown in fig.9 and additionally the uncrompressible header fields. Thereby the encoding field is defined as described in tables 2,3,4 and 5, where PI indicates that the prefix is carried uncompressed, PC stands for the prefix being compressed that means it is a link local prefix, II stands for the interface identifier being carried uncompressed and IC stands for the

Figure 9: LOWPAN_HC1 encoding

00	PI, II
01	PI, IC
10	PC, II
11	PC, IC

Table 2: HC1 encoding: IPv6 source (bits 0 and 1) and destination (bits 1 and 2) address

0	not compressed
1	both are zero

Table 3: HC1 encoding: Traffic class and flow label (bit 4)

00	not compressed
01	UDP
10	ICMP
11	TCP

Table 4: HC1 encoding: next header (bits 5 and 6)

0	no more header compression
1	HC2 encoding is following the HC1 encoding header

Table 5: HC1 encoding: HC2 encoding (bit 7)

interface identifier being elided as it is derivable from the link layer interface identifier. Analysing this one can see that in the ideal case having a HC1 encoding of "11111011" leads to an IPv6 header of as low as 2 bytes as 1 byte is used for the encoding field and the other one is the hop limit. All other fields are given implicitly.

By setting anything but 00 in the two next header bits as shown in table 4 and in addition setting the HC2 encoding bit to 1 as shown in table 5 it is possible to compress the next header as well. RFC4944 [9] describes a HC2 encoding for UDP packets called HC_UDP. This encoding mechanism allows to compress the source as well as the destination port and the length field. This modified header is of the form shown in fig.10, where the bits 3 through 7 are reserved for future use. The compression can be achieved according to tables 6 and 7. If a port is compressed, it is compressed to 4 bits and the full 16 bits port number is calculated by adding this short port to the fixed port 61616.

Figure 10: HC_UDP encoding

3.5 Integration

So far we have seen how in general IPv6 packets can be encapsulated into 802.15.4 packets (3.2). Then we have seen how those IPv6 packets can be compressed for the sensor nodes to be able to transmit them. But there is a last point

0	not compressed
1	compressed

Table 6: HC_UDP encoding: source (bit 0) and destination (bit 1) port

0	not compressed
1	compressed. Can be calculated from the IPv6 payload length

Table 7: HC_UDP encoding: length (bit 2)

that needs to be achieved and that is the one of integrating the sensor nodes into an existing IPv6 network. For that to be done we need a topology similar to the one shown in fig.11 where the sensor nodes can directly communicate with one another and if they want to open a connection with a full IPv6 station they have to route their packets over an edge router which translates those 802.15.4 packets into real IPv6 packets and vice versa.

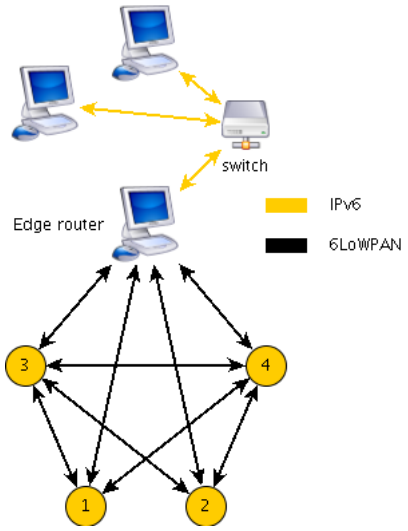


Figure 11: 6LowPAN network integrated into IPv6 network

4. MINISEC

4.1 Basics

MiniSec was developed to address and guarantee the needed security criterias "confidentiality", "authenticity" and "replay-protection" as well as freshness and low energy consumption. Thereby the two modes of communication must be addressed seperately as the replay protection cannot be guaranteed the same way. Namely this is unicast and broadcast which get addressed by MiniSec-U as described in 4.2.1 and MiniSec-B as described in 4.2.2, respectively. Authenticity is warranted in this protocol by the use of OCB (Offset Codebook) [12] as block cipher mode. Same is used to ensure confidentiality. Replay protection depends on the communicationmode, as already stated. As initialisation vector MiniSec uses a counter which is different depending on the mode of communication as well.

The key distribution system is not specified with MiniSec so any system can be applied to fit the individual needs for security.

4.1.1 OCB

In order to describe OCB [7] we need to define some notation first. The notations used are listed in table 8. It must be noted that MiniSec-U uses a directed key, i.e. there is a key for the communication from A to B as well as for the communication from B to A. In contrast Minisec-B uses one network-wide key per sender.

M	the message to be encrypted
H	an optional header that is correlated with M
E	a block cipher (e.g. AES)
K	the key to be used for the communication
N	a nonce
$S \ll 1$	Left-shift S by 1 bit, eliminating the first bit and appending a "0"
λ	$E_K(N)$ is the encryption of N with the cipher E and key K
ζ	$E_K(0)$
2S	$S \ll 1$ if S starts with a 0, otherwise $(S \ll 1) \oplus 0x000...087$
3S	$2S \oplus S$
5S	$(2(2S)) \oplus S$
S0*	S with appended 0s, such that the desired length is achieved
S10*	S with an appended 1 followed by 0s such that the desired length is achieved
len	the binary representation of the length represented with the desired amount of bits

Table 8: OCB notation

In the beginning M is divided into equally sized blocks while the last one may be of shorter size. The block size is chosen such that the block cipher used is capable to process the blocks. Additionally an arbitrary nonce N of the same size is required. The encryption works as shown in fig.12. The XOR of Pad and M_n means that only the first len(M_n) Bits of Pad are used. The checksum is calculated as follows: $M1 \oplus M2 \oplus ... \oplus M_{n-1} \oplus (C_m 0^* \oplus Pad)$.

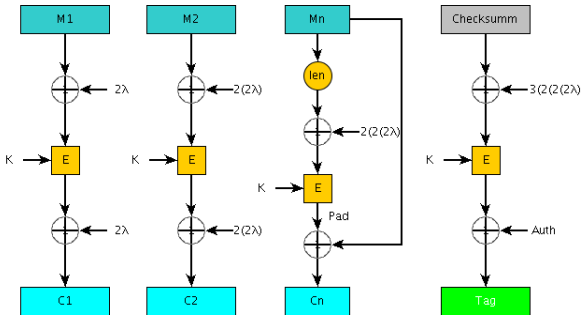


Figure 12: OCB main scheme

Auth can be neglected if no additional header H was handed over as in this case Auth only consists of 0 bits. Otherwise Auth is calculated according to fig.13. For that the header is divided into equally sized blocks while the last may again be

of shorter size. If H_n is as big as the other blocks, then ζ_{End} is equal to $3(2(2(2(5\zeta))))$. Otherwise H_n is padded with a 1 bit and as many 0 bits as needed (H_n10^*) and ζ_{End} is equal to $5(2(2(2(5\zeta))))$.

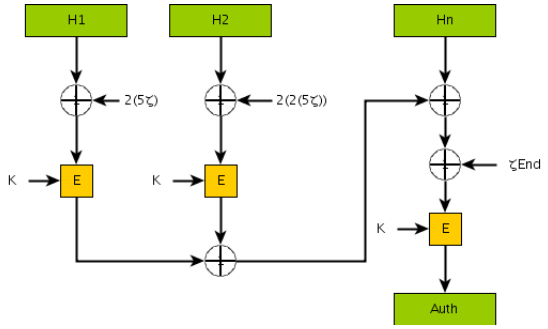


Figure 13: OCB auth calculation

The outcome then is the tupel containing the cipher-text C and the authentication tag "Tag". In order to prove authenticity and integrity the receiver only has to apply the inverse OCB on the cipher-text and calculate the tag on its own. The message was mangled or injected, if the received and calculated tags do not match.
The advantage directly arising for sensor networks is that authenticity and encryption are achieved in one step and therefore less computational resources are needed. In addition no unnecessary bits are transmitted as a shorter block is not enlarged to fit a given block size.

4.2 The protocol

4.2.1 MiniSec-U

MiniSec-U is the variant used if the connection is a unicast one. At this a message is sent by exactly one sender and is destined for exactly one receiver. For this connection a key $K_{A,B}$ is used for messages sent from A to B and a different key $K_{B,A}$ for the opposite direction. For each key a monotonous increasing synchronized counter $C_{A,B}$ or $C_{B,A}$ is needed in addition. This counters are used as initialisation vectors between the two parties A and B. This vector is not transmitted as a whole with every message, but only the last Δ bits as shown in fig.14. Thus no unnecessary in-

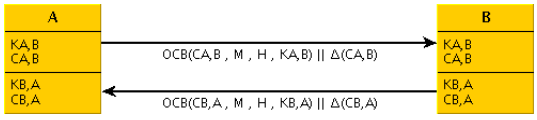


Figure 14: MiniSec-U communication scheme

formations about the initialisation vector are transmitted. This is of benefit for sensor networks since transmission of data is the most energy consuming operation. Each of the parties increases the internally stored counter with each sent or received message. The receiver in addition can adjust its counter based on the last Δ bits received with the message, if the counter is no more in sync. If too many messages were lost in transmission and so simply adjusting according to the last few bits does not work, it is still possible to adjust the counter by gradually increasing the counter by 2^{Δ}. If this does not work after repeated attempts, resynchronisation will be needed which is quite energy consuming.

4.2.2 MiniSec-B

This variant is used if broadcast messages need to be transmitted. It is clear that in a broadcast environment it is not possible to use the already exchanged unicast keys. Therefore a network-wide key is used for every principal that wants to send a broadcast message. This key is used to encrypt the message with OCB. As it is not possible for low memory reasons that every node holds a counter for every possible sender, a different approach must be used. Nevertheless it is possible to guarantee replay protection by combining two Methods. These are the sliding-windows and bloom-filters [2].

With the sliding-windows approach a finite time t_e is defined which is used to determine the length of a so called epoch. Afterwards the time is divided into epochs of length t_e. By a loose time synchronisation between the nodes the current epoch E is known to every participant. t_e is chosen such that it equals twice the time of the maximum possible time synchronisation error Δ_S plus the maximum network latency Δ_L ($t_e = 2 * \Delta_S + \Delta_L$). By using the current epoch number as initialisation vector for OCB, packets from past epochs could automatically be detected as replays. But since by the time synchronisation error and network latency a correctly sent message can reach the receiver in a wrong epoch, two epochs will be put into consideration. As shown in fig.15 based on the time of arrival either the previous or the following epoch will be considered in addition. Let t be the

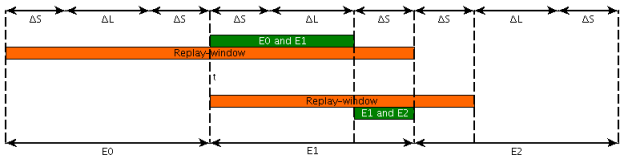


Figure 15: Sliding-window approach and Replay-attack window

time at the beginning of epoch E1 as shown in fig.15. Then epochs E0 and E1 will be considered if and only if the arrival time t_A is between t and $t + \Delta_S + \Delta_L$. Epochs E1 and E2 are considered accordingly if and only if t_A is between the rest of the current epoch, i.e. between $t + \Delta_S + \Delta_L$ and $t + 2 * \Delta_S + \Delta_L$.
To validate that this division is correct, we consider the least possible time and the highest possible time the message could be sent such that it arrives in the respective time window.

Case 1: Packet arrives such that E0 and E1 are put into consideration. In this case the least possible time s_{min} is $t - \Delta_S - \Delta_L$ lying in E0. The maximum possible time s_{max} is $t + \Delta_S + \Delta_L + \Delta_S$ lying in E1. Hence this part is correct.

Case 2: Packet arrives such that E1 and E2 are put into consideration. In this case the least possible time s_{min} is $t + \Delta_S + \Delta_L - \Delta_S - \Delta_L = t$ and lyes therefore in E1. The

maximum sent time s_{max} is $t + 2 * \Delta_S + \Delta_L + \Delta_S$ lying in E2. Hence this part is correct either.

This shows directly that packets captured at the beginning of an epoch can be replayed during this whole epoch and in addition up to $\Delta_S + \Delta_L$ in the following epoch.
In order to guarantee replay protection during this time window, the sender holds an internal counter which is increased with every sent packet. The counter is reset at the end of every epoch and can therefore be very short. The advantage of a short counter is that it can be transmitted as a whole and the receivers do not have to save the state of the counter themselves. In addition bloom-filters are used.
Bloom-filters are probabilistic datastructures which can be used to check whether an element is present or not and that in a very memeory and processor efficient way. The only two possible operations are adding elements and checking whether an element is present. Deleting is not possible. When checking for the presence of an element false-positives are possible false-negatives in contrast are not. For this to achieve an array is used which is filled initially with 0 bits. By adding an element this element gets mapped by different hash-functions to different indices which are then set to 1. In order to check whether an element exists one has to calculate all the hash-functions and check whether all fields in the array that the indices refer to are set to 1. If even one such array field is set to 0 it is sure that the given element was not yet added. This procedure is shown schematically in fig.16. The initialisation vector is constructed by the concatenation

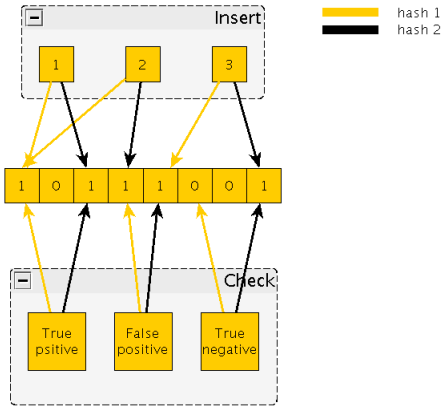


Figure 16: Bloom-Filter

of the senders ID, the counter of the sender and the current epoch number. Since the epoch number is never reset this vector will never be reused. The receiver has to manage two separate bloom-filter in which the received packets are inserted. As key for the insertion the concatenation of the senders counter and the senders ID is used. One has to use two separate bloom-filters since as shown in fig.15 two epochs are checked at every time. Further the counter of the sender is reset at the end of every epoch and therefore an assignment to the epoch is necessary. It would be possible to add the epoch number to the key when inserting into the bloom-filter, but by that many elements would be added to the filter very quickly. Hence the false-positive rate would increase steadily. By resetting the bloom-filter after its validity and reusing it for a new epoch the problem of an increasing false-positive rate is elided.
One of the two filters always belongs to the current epoch and the other one belongs either to the previous or the following epoch. Being within $\Delta_S + \Delta_L$ from the beginning of the epoch the second filter will be used for the previous epoch. When leaving this time slot the filter is reset and used for the following epoch.
As a valid message must belong to any of the considered epochs and the counter as well as the ID of the sender is sent in plain text, one can determine the correct epoch by decrypting the message. To accomplish this one simply has to decrypt the ciphertext with both epoch numbers one after another and check whether the decryption succeeded. Afterwards it can be checked whether the message already exists in the corresponding bloom-filter. If this check returns true it is most likely that the message was replayed and can therefore be dropped. Otherwise it is for sure that it is new and can be accepted and inserted into the bloom-filter.

4.3 Security analysis

As stated in [14] in order to analyse security it is at least necessary to consider the criterias confidentiality, authenticity and in the case of sensor networks freshness. In the following im going to split authenticity further into accountability and integrity.

Confidentiality: Confidentiality is given in both MiniSec-U as well as MiniSec-B by applying encryption utilising OCB (4.1.1). This procedure is as safe as the underlying block cipher that is used. So by using a cipher that has no yet known security threats this criteria can be considered as proofed. Semantic security is given in case of MiniSec-U by using a counter as IV for every direction. In case of MiniSec-B this is achieved by using the epoch number and a shorter counter.

Integrity: Integrity is directly given in both variants by the tag. Here the security directly depends on the length of the tag. The probability to guess a correct tag with a length of 32 bit is 2^{-32} and this can be considered as inveasible.

Accountability: MiniSec-U directly provides accountability by using a seperate key for every node and direction. Without the knowledge of this key it is inveasible to calculate a correct tag and can therefore not masquerade as another node. MiniSec-B in contrast uses a network wide key that every receiver knows. By that every node connected to the network that has received this key can masquerade as the sender since all other parameters like the counter are known as well. This could be remedied by not allowing to use a network wide key a second time.

Freshness: In the variant of MiniSec-U monotonous increasing counters are used as initialisation vectors which are kept internally by both parties. Hence the receiver can check at any time whether the packet is fresh or replayed. In MiniSec-B it would be possible to replay packets during a certain time window if only sliding-windows were applied.

By additionally utilising bloom-filters these packets can be detected. Replayed packets from old epochs are detected by the sliding-windows and replayed packets that would be accepted by the sliding-windows are detected by the bloom-filters. Hence freshness is guaranteed with both variants.

4.4 Comparision to similar protocols

4.4.1 TinySec

As stated in [6] TinySec uses CBC (Ciffer Block Chaining) as block cipher mode and in addition for authentication a CBC-MAC. The great disadvantage with that is that in order to calculate the authentication code the whole encryption algorithm has to be recalculated a second time. One cannot simply use the same key for encryption and authentication as this would lead to great security threats [1]. In this case it would be possible to change the ciphertext without the possibility to notice the modification. Further CBC-MAC uses an initialisation vector 0 and therefore the first encryption round cannot be reused. TinySec uses a 8 bytes long initialisation vector which is used as a counter as MiniSec does in order to prevent a quick repetition. However this initialisation vector is appended to every packet which means an overhead of 8 bytes minus a few bits in comparison to MiniSec. The last block is not enlarged to a given block size but kept as short as possible by using "ciphertext stealing" [13]. TinySec uses a network wide key even for unicast communications. This arises the advantage that no key-exchange algorithms have to be performed neither in unicast nor in broadcast communications. This is achieved for the cost of lower security as accountability can not be guaranteed at any degree. Further, none of the receivers saves anything about the received packets or the state of the last initialisation vector. So freshness can't be guaranteed as well.

4.4.2 SNEP & μTESLA

Another possible protocol combination is SNEP (Sensor network encryption protocol) [11] for unicast connections and μTESLA [11] for broadcast authentication.

SNEP is a procedure which guarantees confidentiality, authenticity and freshness for unicast connections. Confidentiality is achieved by using symmetric cryptography as MiniSec does, but SNEP only uses one key per communication pair. As initialisation vector there is a counter for each direction which is managed by both parties like it is managed in MiniSec. In comparison to MiniSec this counter is not at all transmitted with the packets and thus reduces the energy consumption. The drawback is that in lossy networks the counter cannot be resynchronized without the influence of the other party and the resynchronization consumes much energy. By using a non repeating counter as initialisation vector, freshness is given out of the box as no packet can be replayed. Authenticity is achieved by a separate MAC. This is as already discussed in 4.4.1 a drawback in so far as the whole message needs to be taken into account another time.

In order to authenticate broadcast messages one can use a simpler variant of TESLA [10] called μTESLA. This is a procedure which can be used for authentication but not for encryption. It is again necessary that the parties are loosely time synchronized in order to divide the time in parts as it is done in MiniSec-B (4.2.2). In each of these parts a separate key K_i is used. For this purpose the sender generates a key K_n and calculates based on that subsequent keys K_{n-i} by using a one-way-function F as shown in fig.17. Based on this function every node can calculate the key K_{m-i} out of K_m but never K_{m+1}. By negotiating a key K_0 between the sender and all the receivers, every receiver can proofe the authenticity of a key with a higher index. The sender

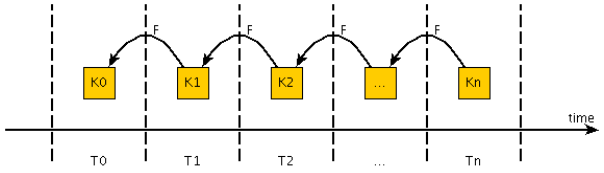


Figure 17: μTESLA representation

authenticates in the time slot T_i every packet with the key K_i and the receiver buffers the packets in the first instance in memory. After a certain defined amount of time the sender discloses the key K_i whereupon the receiver can check the authenticity of this key by subsequently utilising the one-way-function F, whereas the following must apply: $K_0 = F^i(K_i)$. Is this the case all the packets buffered for the time slot T_i can be authenticated with the Key K_i and this key can now be set to be the new trusted key so that the applied one-way-function chain is kept as short as possible. The advantage in comparison to MiniSec is that no other node can masquerade as the sender as keys with a higher index cannot be calculated and old keys are no more valid after disclosing it. The drawback however is that this can nearly not be used for sensor nodes to send broadcast messages, but only to receive them. This is because the full key chain cannot be saved in the limited memory of such nodes and recalculation of the keys all the time is very computationally expensive.

4.4.3 Common security protocols

Common security protocols have the big advantage that they are widely used and are therefore tested by a wide variety of users and got analysed by security experts. The drawback however is that they are not built to be used in an environment with limited resources. Common security protocols often implement very complex algorithms to achieve a very high rate of security. However this is not possible on sensor nodes as they are very limited in memory and computing power. Further most of the protocols rely on TCP which needs many packets to be received and sent which consumes much of the limited energy. Asymmetric cryptography would be very secure but considering for example a 2048 bit key would impose that every node saves the public keys of its neighbours and this is nearly impossible because of the very limited memory. In addition assymmetric encryption algorithms are way more computationally expensive than symmetric ones. Another criteria is as discussed in [6] that in conventional networks most connections are end-to-end ones and the intermediate nodes (e.g. routers) do not check the packets but simply forward them as transmission does not matter. In sensor networks however data transmission is the most energy consuming factor and therefore such a procedure should not be chosen.

5. CONCLUSION

As we have seen 6LoWPAN is fairly well suited for the integration of 802.15.4 capable devices into existing IPv6 networks. The specification not only allows these devices to be integrated but in addition defines procedures that are optimized for the usage in environments with very limited resources and therefore can be used without any doubt even with very cheap and simple sensor nodes.

MiniSec is, as well as 6LoWPAN is, highly optimized for the use with sensor nodes and networks. This protocol tries to eliminate all the overhead that is not absolutely needed to guarantee the security requirements. But it does not simply leave away every conceivable overhead but tries to find the golden mean between letting informations away and sending that much that no additional negotiation is needed. Nevertheless at least the unicast method fully assures the security requirements in wireless networks. The broadcast variant ensures the main criterias namely confidentiality, integrity and freshness as well, but not accountability. So this protocol is quite well suited for sensor networks particularly as it is not vulnerable to any known attack except brute-forcing, which is always a possible attack.

6. REFERENCES

[1] M. Bellare, J. K. T, and P. Rogaway. The security of the cipher block chaining message authentication code, 2001.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.

[3] M. Crawford. RFC2464 transmission of ipv6 packets over ethernet networks. http://www.faqs.org/rfcs/rfc2464.html, December 1998.

[4] S. Deering and R. Hinden. RFC2460 internet protocol, version 6 (ipv6) specification. http://tools.ietf.org/html/rfc2460, December 1998.

[5] IEEE Computer Society. Ieee std. 802.15.4-2003, December 2003.

[6] C. Karlof, N. Sastry, and D. Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 162–175, New York, NY, USA, 2004. ACM.

[7] T. Krovetz and P. Rogaway. The ocb authenticated-encryption algorithm. http://www.cs.ucdavis.edu/ rogaway/papers/draft-krovetz-ocb-00.txt, March 2005.

[8] M. Luk, G. Mezzour, A. Perrig, and V. Gligor. Minisec: a secure sensor network communication architecture. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 479–488, New York, NY, USA, 2007. ACM.

[9] G. Montenegro, J. Hui, and D. Culler. RFC4944 transmission of ipv6 packets over ieee 802.15.4 networks. http://tools.ietf.org/html/rfc4944, September 2007.

[10] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The tesla broadcast authentication protocol. *RSA CryptoBytes*, 5:2002, 2002.

[11] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. E. Culler. Spins: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, September 2002.

[12] P. Rogaway, M. Bellare, and J. Black. Ocb: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, 2003.

[13] B. Schneier. Applied cryptographie, second edition. John Wiley and Sons, 1996.

[14] L. Tobarra, D. Cazorla, F. Cuartero, and G. Diaz. Analysis of security protocol minisec for wireless sensor networks. In *Proceedings of the IV Congreso Iberoamericano de Seguridad Informatica (CIBSI 2007)*, pages 1–13, November 2007.

Sicherheit durch Hardwareeinbindung - secFleck

Thomas Riedmaier
Betreuerin: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: riedmaie@in.tum.de

KURZFASSUNG

Drahtlose Sensornetzwerke wie das Fleck™-WSN halten Einzug in immer mehr Anwendungsgebiete. Im Generellen befinden sich die Sensorknoten hierbei in einer unbewachten Umgebung und sind deshalb zahlreichen Angriffsszenarien ausgesetzt. Das in dieser Arbeit vorgestellte secFleck-System begegnet dieser Problematik, indem es das Standard-Fleck™ -System um einen TPM-Chip erweitert. Dank diesem ist es möglich sowohl asymmetrische als auch symmetrische Verschlüsselungsroutinen effektiv auf einem Sensorknoten zu berechnen. Diese Kombination ermöglicht eine Reihe von Sicherheitsfeatures, von denen einige - wenn auch bei weitem nicht alle - in der secFleck-API realisiert wurden. Viele Features, die der TPM-Chip zur Verfügung stellen würde, bleiben jedoch ungenutzt.

Schlüsselworte

Fleck, SecFleck, Trusted Platform Module (TPM), Wireless Sensor Network (WSN)

1. EINLEITUNG

Drahtlose Sensornetzwerke (WSN) haben schon lange ihre militärischen Wurzeln hinter sich gelassen und werden inzwischen auch zu rein zivilen Zwecken genutzt. Zu diesen gehören unter Anderen wissenschaftliche Forschungsprojekte wie das sog. „Habitat monitoring“ [25], welches Ökologen hilft Einflüsse auf Lebensräume besser verstehen zu lernen. Überdies kommen WSNs auch in kommerziellen Anwendungen wie der Überwachung von Bergwerken zum Einsatz [18].

Da die Sensorknoten eines Netzwerkes typischerweise nur über sehr beschränkte Energieressourcen und Rechenkapazitäten verfügen, wurde bisher dem Sicherheitsaspekt von WSNs nur eine geringe Bedeutung beigemessen. Gerade in den kommerziellen Anwendungen ist die Sicherheit der Systeme jedoch ein verkaufskritisches Feature. Hierbei geht es nicht nur um die Vertraulichkeit eventuell geschäftskritischer Daten, sondern auch um die Glaubwürdigkeit der vom Sensornetzwerk ermittelten Daten. Ein von einem Angreifer absichtlich ausgelöster falscher Alarm könnte beispielsweise mehr Kosten verursachen als der Einsatz des WSNs an Kosten einspart.

Es liegt jedoch in der Natur der drahtlosen Sensornetzwerke, dass die Sensorknoten weit verteilt und deshalb oft unbewacht installiert sind. Dies setzt das Netzwerk einer Reihe von Angriffen aus, die sich nicht nur auf das Eindringen in den Funkverkehr beschränken. Auch der Fall, dass einzelne Knoten unauthorisiert umprogrammiert werden ist nicht auszuschließen.

Diese Arbeit stellt mit secFleck eine auf dem TPM-Chip basierende Technologie vor, die Lösungen für eine Reihe von brennenden Sicherheitsproblemen der WSN-Technologie verspricht. Sie gliedert sich dabei wie folgt:

Zunächst wird in Kapitel 2 mit Fleck™ diejenige Drahtlos-Netzwerk-Technologie vorgestellt, aus der das secFleck-System hervorgegangen ist. Anschließend wird in Kapitel 3 eine generelle kurze Einführung in die Sicherheit verteilter Systeme gegeben, welche in der Vorstellung der TPM-Technologie mündet. Auf diesen Grundlagen aufbauend wird daraufhin in Kapitel 4 secFleck vorgestellt und evaluiert. Abgeschlossen wird mit einem kurzen Überblick über verwandte Arbeiten in Kapitel 5 sowie einem Ausblick in die Zukunft in Kapitel 6.

2. FLECK

Fleck™ [22] bezeichnet eine aus der Zusammenarbeit zwischen dem australischen CSIRO [1] und der Datacall® Company [2] hervorgegangene Drahtlos-Sensor-Netzwerk-Technologie, deren Platine in Abb. 1 zu sehen ist. Sie ermöglicht das Sammeln von Informationen über große Gebiete hinweg, und bietet so beispielsweise für große Agrar- oder Minengesellschaften eine einfache aber dennoch verlässliche Möglichkeit ihren Betrieb zu überwachen [26].

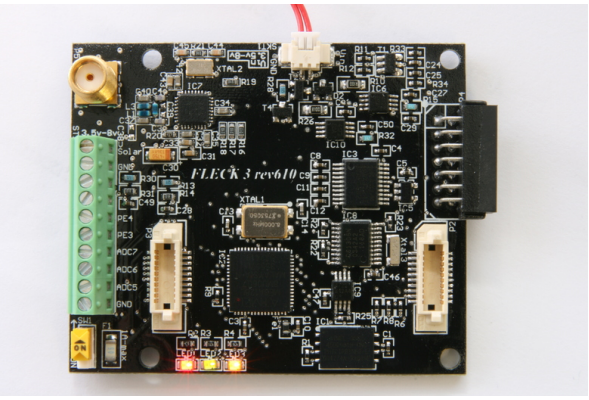


Abbildung 1: Fleck3 Platine mit Atmega128 Microcontroller und nRF905 Multiband Transceiver [27].

Gegenüber klassischen telemetrischen Sensorsystemen bietet Fleck™ zahlreiche Vorteile: Kosteneffizienz, einfache „Plug-and-Play“(PnP)-Installation des Systems sowie eine gesteigerte Zuverlässigkeit durch Selbstheilungsfähigkeiten des Netzwerkes. Der Aspekt mit dem das Fleck™-System jedoch am meisten gegenüber seinen Konkurrenzprodukten punkten kann, ist die geringe Stromaufnahme seiner Sensorknoten. Diese sind so gefertigt, dass sie mit drei AA-Akkus betrieben werden können, welche tagsüber durch ein kleines Solarpanel aufgeladen werden [26].

Damit Anwendungsprogrammierer sich nicht direkt mit der Hardware auseinandersetzen müssen, sind die Fleck™-Sensorknoten mit dem Fleck Operating System (FOS) [5] ausgestattet. Anders als das weitverbreitete TinyOS setzt FOS nicht auf Eventhandler um die Ressourcen auf verschiedene Programme aufzuteilen, sondern auf Threads. Dies ermöglicht eine einfache, intuitive Programmierung mit einem POSIX-ähnlichen Interface. FOS stellt zudem sicher, dass der Energieverbrauch des Systems auf ein Minimum beschränkt bleibt [5].

3. SICHERHEIT IN DRAHTLOSEN SENSORNETZWERKEN

Wie bei den meisten technischen Systemen gilt auch bei WSN-Systemen die Faustregel: Je komfortabler ein System zu benutzen ist, desto unsicherer wird es. Da es sich beim Fleck™-System um ein kommerzielles Produkt handelt, waren seine Entwickler bemüht, ein möglichst komfortabel benutzbares System zu erschaffen. Dies zeigt sich u. a. im Fokus auf die PnP-Funktionalität, welche es sehr einfach macht, ein Fleck™-System aufzubauen bzw. zu erweitern. Diese Einfachheit bedeutet jedoch auch, dass es einem Angreifer sehr einfach möglich ist in das System einzudringen, indem er beispielsweise seinen eigenen Sensorknoten in das System einschleust oder den Datenverkehr mitschneidet [12].

Im Gegensatz dazu haben die meisten Systembetreiber jedoch ein nicht zu vernachlässigendes Sicherheitsbedürfnis. Im Falle eines WSN lässt sich dieses Bedürfnis meist auf folgende Schutzziele zusammenfassen [10]:

- Informationsvertraulichkeit
- Verbindlichkeit von Nachrichten
- Datenintegrität

Klassischerweise wird die Einhaltung dieser Schutzziele durch die Verwendung kryptographischer Methoden sichergestellt. Im Nachfolgenden seinen diese kurz vorgestellt:

3.1 Symmetrische Verschlüsselungsverfahren

Charakterisierend für symmetrische Verschlüsselungsverfahren ist die Tatsache, dass die zur Ver- bzw. zur Entschlüsselung von Nachrichten verwendeten Schlüssel gleich, oder zumindest leicht voneinander ableitbar sind. Um den Inhalt verschlüsselter Nachrichten vor Angreifern zu schützen ist es deshalb nötig, beide Schlüssel geheim zu halten. Daraus ergibt sich jedoch auch gleich das größte Problem symmetrischer Verschlüsselungsverfahren: die Schlüsselverteilung, oder anders formuliert: „Wie ist es dem Sender möglich, dem berechtigten Empfänger (und nur diesem) den Entschlüsselungsschlüssel zukommen zu lassen?“ [10].

Im Falle von eingebetteten Systemen wird das Schlüsselverteilungsproblem oft sehr pragmatisch dadurch gelöst, dass den einzelnen Komponenten noch vor ihrem Einsatz die benötigten Schlüssel eingespeichert werden. Wurde auf diese oder eine andere Weise das Problem gelöst, ist es allen beteiligten Komponenten möglich eine vertrauliche Kommunikation zu führen.

Populäre Vertreter der symmetrischen Verschlüsselungsverfahren sind beispielsweise der „Data Encryption Standard“ (DES) [23], der „Advanced Encryption Standard“ (AES) [4] oder das speziell für eingebettete Systeme geeignete „Extended Tiny Encryption Algorithm“ (XTEA) [20].

3.2 Asymmetrische Verschlüsselungsverfahren und Signaturen

Anders als die symmetrischen Verschlüsselungsverfahren verwenden die asymmetrischen Verschlüsselungsverfahren separate Ver- und Entschlüsselungsschlüssel.

Kernidee der asymmetrischen Verschlüsselung ist es, dass sich jeder Kommunikationspartner ein Schlüsselpaar, bestehend aus einem privaten und einem öffentlichen Schlüssel generiert. Die Schlüsselpaare werden auf eine Weise generiert, die sicherstellt, dass sich Informationen, welche mit dem öffentlichen Schlüssel verschlüsselt wurden, ausschließlich mit dem privaten Schlüssel entschlüsseln lassen und umgekehrt. Der öffentliche Schlüssel wird nach der Erzeugung mit dem Namen des Erstellers versehen und der Allgemeinheit zur Verfügung gestellt. Dies stellt kein Sicherheitsrisiko dar, da es bei sicheren asymmetrischen Verschlüsselungsverfahrens nicht möglich ist, lediglich auf Grund der Kenntnis des öffentlichen Schlüssels den privaten Schlüssel effizient zu berechnen. Vielmehr stellt dieses Vorgehen eine sehr elegante Möglichkeit dar das Schlüsselverteilungsproblem zu lösen [10].

Im direkten Vergleich mit den symmetrischen Verfahren schneiden die asymmetrischen Verfahren (wie z.B. RSA [21]) eher schlecht ab: Sie lassen sich bei gleicher Schlüssellänge leichter brechen, können oft nicht so effizient in Hardware umgesetzt werden und benötigen mehr Zeit für Berechnungen [10]. Im Falle von eingebetteten Systemen sind die beiden letzten Faktoren kritisch, da Rechenzeit mit Stromverbrauch einhergeht und somit möglichst zu minimieren ist.

Die Stärken der asymmetrischen Verfahren liegen auf den Gebieten, auf denen die symmetrischen Verfahren keine Lösung anbieten, wie es z.B. beim der Schlüsselverteilungsproblem der Fall ist. Gerade in WSN-Systemen, in denen dynamisch Knoten hinzukommen oder entfernt werden, kann so ein Grad an Sicherheit erreicht werden, der durch das einmalige Einspeichern von Schlüsseln in die Hardware niemals erreicht werden könnte.

Zudem stellt ein mit einem privaten Schlüssel verschlüsseltes - man sagt auch signiertes - Datum die Information dar, dass der Besitzer dieses Schlüssels das Datum kannte, denn nur er konnte es verschlüsseln. Handelt es sich bei diesem Datum beispielsweise um eine Nachricht, so kann also von

jeder beliebigen Person geprüft werden, ob die Nachricht dem Signierer vorgelegen hat. Durch Signaturen lässt sich also verbindlich nachweisen, aus welcher Quelle eine Nachricht stammt [10].

Um die Vorteile sowohl der symmetrischen als auch der asymmetrischen Kryptographie ausnutzen zu können wurden sogenannte hybride Verfahren entwickelt. Diese Verfahren verwenden zunächst ein asymmetrisches Verfahren, um symmetrische Schlüssel auszutauschen. Die darauf folgende Kommunikation wird dann mit den eben ausgetauschten Schlüsseln symmetrisch verschlüsselt [10].

3.3 Hashfunktionen

Unter einer Hashfunktion versteht man im mathematischen Sinne eine Einwegfunktion, welche ein Eingabedatum beliebiger Größe auf einen Bitstring fester Länge abbildet. Für kryptographisch starke Hashfunktionen wie dem SHA-1 [8] ist es u.a. nicht möglich, in realistischer Zeit

- zu einem gegebenen Hash H ein Datum zu bestimmen, welches gehasht H ergibt [10].
- zwei verschiedene Eingabewerte zu finden, welche gehasht denselben Bitstring erzeugen [10].

Ein beliebtes Einsatzgebiet für Hashfunktionen ist die Sicherstellung der Integrität von über ein Netzwerk übertragenen Nachrichten. Ein mögliches Vorgehen hierfür ist, dass der Sender zunächst zu einer Nachricht M ihren korrespondierenden Hashwert H(M) bestimmt. Anschließend wird dieser Wert unter Benutzung eines asymmetrischen Verfahrens signiert und an die Nachricht angehängt. Wird die Nachricht während der Übertragung verändert, so kann dies vom Empfänger erkannt werden [10].

3.4 TPM und Trusted Computing

Da die in Abschnitt 3 genannten Sicherheitsschutzziele auch in großen, potentiell nicht gänzlich vertrauenswürdigen Systemen gewährleistet werden sollen (Trusted Computing), werden viele Systeme heutzutage mit Trusted Plattform Modulen (TPM) ausgestattet. Im Wesentlichen handelt es sich hierbei um ein Subsystem mit geschütztem Speicher und der Möglichkeit sicherheitskritische Operationen in Hardware durchzuführen. Die für diese Arbeit zentralen Funktionen sind [24]:

- **Tcsip_GetRandom:** Erzeugen einer starken Zufallszahl.
- **Tspi_Data_Bind:** Asymmetrisches Verschlüsseln von Daten z.B. mit RSA (vgl. Abschnitt 3.2).
- **Tspi_Data_Unbind:** Entschlüsseln Asymetrisch verschlüsselter Daten (vgl. Abschnitt 3.2).
- **Tspi_Hash_Sign:** Hasht ein Objekt und signiert das Ergebnis (vgl. Abschnitte 3.2, 3.3).
- **Tspi_Hash_VerifySignature:** Verifiziert, ob der Hash eines Objekts identisch ist mit dem, der in seiner Signatur steht (vgl. Abschnitte 3.2, 3.3).
- **Tcsip_PcrRead:** Auslesen der PCR Register (vgl. Abschnitt 3.4.2).
- **Tcsip_Seal:** Verschlüsseln mit binden an den TPM-Chip. Selbst dieser kann die Daten nur dann wieder entschlüsseln, wenn sich das System im passenden Zustand befindet (korrekter Wert des PCR-Registers).
- **Tcsip_Unseal:** Entschlüsseln versiegelter Daten.

Lediglich die asymmetrische Verschlüsselungsfunktionalität ist Teil derjenigen Spezifikation, die ein TPM-Chip erfüllen muss. Symmetrische Verfahren müssen nicht notwendigerweise unterstützt werden, weshalb sie auch oft nicht angeboten werden [24].

Der TPM-Chip stellt seiner Plattform neben den kryptographischen Funktionen eine Reihe von Features zur Verfügung, welche im Folgenden kurz erläutert werden.

3.4.1 Überprüfbare anonyme Identitäten

Jeder TPM-Chip ist in der Lage beliebig viele asymmetrische Schlüsselpaare zu erzeugen, von denen zwar überprüft werden kann, ob sie von einem dem Standard entsprechenden TPM-Chip generiert wurden, aber nicht von welchem. Indirekt ist somit überprüfbar, ob ein Schlüsselpaar vorliegt, dessen privater Schlüssel nur *einem einzigen* Chip bekannt ist. Diese Schlüsselpaare werden „Attestation Identity Key Pairs“ (AIKs) genannt [19].

Das Prinzip nach dem diese Schlüssel erzeugt werden lässt sich in wenigen Sätzen erklären: Bei der Fertigung wird jedem TPM-Chip sowohl ein sog. „Endorsement Key Pair“ (EK), als auch ein vom Hersteller unterschriebenes Zertifikat eingespeichert, das versichert, dass besagter EK einem standardkonformen TPM-Chip gehört. Da der EK nicht selbst zum Verschlüsseln von Daten verwendet werden darf generiert sich der TPM-Chip ein AIK Paar. Den öffentlichen Schlüssel dieses AIKs signiert der Chip mit seinem EK und schickt das Ergebnis zusammen mit seinem EK-Zertifikat an eine beliebige „Privacy-Certification Authority“ (P-CA). Diese kann dank dem vom Hersteller unterschriebenen EK-Zertifikat ein weiteres Zertifikat ausstellen, das besagt, dass der AIK wirklich einem standardkonformen TPM-Chip gehört. Da die P-CA nur dann signiert, wenn sie dem Hersteller vertraut, kann daraufhin jeder, der der P-CA vertraut, dem AIK vertrauen [19].

Diese Funktionalität kann in Sensornetzwerken von großem Vorteil sein. So ist beispielsweise ein Szenario vorstellbar, in dem Personen mit Positionssensoren ausgestattet werden. Diese Daten werden durch das Netzwerk zu einer Basisstation geschickt, wo sie ausgewertet werden. Theoretisch wäre es einem manipulierten Sensorknoten also möglich Bewegungsprofile von jedem anderen Sensor anzufertigen und einer Person zuzuordnen. Werden jedoch statt einem Knotenbezogenem Schlüssel AIKs verwendet, die von der Basisstation unterschrieben wurden, so ist nur noch diese zu dieser Zuordnung in der Lage.

3.4.2 Beglaubigter Bootvorgang

Jeder TPM-Chip verfügt über eine Anzahl von „Platform Configuration Registers“ (PCRs), in denen eine Verknüpfung von Messergebnissen über den Zustand des Systems gespeichert wird. Diese Register befinden sich in einem geschützten Speicherbereich und können nur durch den Neustart des Systems zurückgesetzt werden. Typischerweise wird in diesen Registern der Bootvorgang mitprotokolliert. Konkret bedeutet dies, dass jede am Bootvorgang beteiligte Komponente zuerst von der Vorherigen gehasht wird, das Ergebnis den PCR-Registern hinzugefügt wird und danach erst die Kontrolle an die Komponente übergeben wird (Abb. 2) [19].

Die Methodik, nach der die PCRs modifiziert werden basiert auf der kryptographisch starken Hashfunktion SHA-1 (vgl. Abschnitt 3.3). Aus diesem Grund ist es einer modifizierten - also pot. schädlichen - Komponente nicht möglich ihre Andersartigkeit gegenüber dem Standard im Nachhinein zu verschleiern. Stimmt hingegen der Wert der PCRs am Ende des Bootvorgangs, so kann von einem integeren System ausgegangen werden [19].

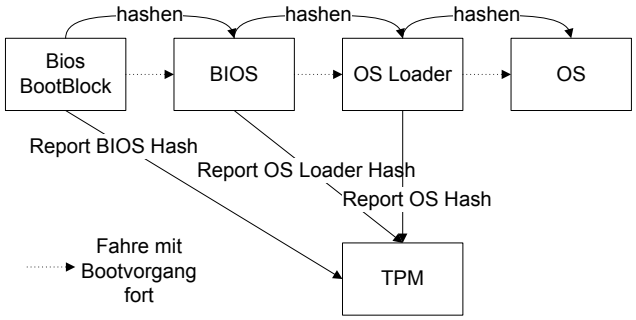


Abbildung 2: Beglaubigter Bootvorgang nach [19]

Das TPM bietet mehrere Möglichkeiten für die Verwendung des Wertes des PCRs. Einerseits kann das TPM dazu verwendet werden, Daten wie z.B. Passwörter nur dann zur Verfügung zu stellen, wenn der Wert des PCR korrekt ist (Seal bzw. Unseal). Andererseits kann es - und das ist gerade in Sensornetzwerken wichtig - auch dazu verwendet werden, die Integrität eines Systems anderen gegenüber zu bescheinigen. Dazu wird der PCR-Wert mit einem AIK unterschrieben und das Ergebnis z.B. der Basisstation zugesendet. Da der AIK nie das TPM verlassen hat kann dieser „Report on integrity“ von niemandem gefälscht worden sein [19].

3.4.3 Sicherer Speicher

TPMs können dazu verwendet werden, Daten sicher zu speichern. Hierzu werden die zu sichernden Daten mit einem Schlüssel verschlüsselt, der nur dem TPM selbst bekannt ist. Zugriff auf diese Daten erhält dann nur, wer sich dem TPM gegenüber als berechtigt ausweisen kann. Zusätzlich können Daten auch so geschützt werden, dass das TPM die Daten nur dann entschlüsselt, wenn nicht nur die Berechtigung korrekt ist, sondern auch der Systemzustand (PCR) [19].

Da sich Sensorknoten potentiell in nicht vertrauenswürdigen Umgebungen befinden, besteht gerade hier ein hohes Bedürfnis nach solchen sicheren Speichermöglichkeiten.

4. SECFLECK

Jeder Fleck^{TM}-Sensorknoten verfügt standardmäßig über einen Erweiterungssteckplatz. Dieser ist dafür gedacht Umgebungssensoren an das System anzuschließen - kann jedoch prinzipiell auch für andere Erweiterungen genutzt werden. Diese Tatsache haben sich Wen Hu et al. in [12] zunutze gemacht und den TPM v1.2 standardkonformen TPM-Chip Atmel AT97SC3203S angeschlossen. Ihr Ziel war es, ein auf der Fleck^{TM}-Platine bzw. dem FOS aufsetzendes sicheres WSN zu schaffen, dessen

- Sicherheitslevel dem des E-Commerce im Internet entspricht [12].
- Geschwindigkeit dem des normalen Fleck^{TM}nicht wesentlich nachsteht [12].
- Energieverbrauch die Möglichkeiten eines drahtlosen Sensorknotens nicht übersteigt [12].
- Kosten im überschaubaren Bereich liegen [12].

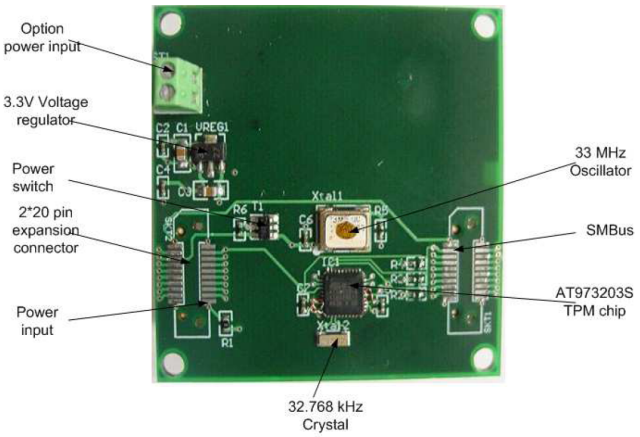


Abbildung 3: secFleck Erweiterungsplatine [12]

Das daraufhin entwickelte System haben die Autoren auf den Namen secFleck getauft. Dieses besteht sowohl aus der Hardware der Erweiterungsplatine (Abb. 3), als auch aus der Erweiterung des FOS um eine Schnittstelle zu Funktionen des TPMs [12].

4.1 Evaluation der Designziele

Im Folgenden wird dargelegt in wie weit und auf welche Weise die in Abschnitt 4 geforderten Designziele erreicht werden konnten. Die Evaluation beginnt mit dem Sicherheitslevel, geht über die Geschwindigkeit des Systems und seinen Energieverbrauch bis hin zu seinen Kosten.

4.1.1 Sicherheitslevel

secFleck stellt an sich selbst den Anspruch, dem Anwendungsprogrammierer sowohl symmetrische als auch asymmetrische Kryptographie zur Verfügung zu stellen. Da TPM-Chips wie bereits in Abschnitt 3.4 erwähnt keine symmetrische Kryptographie anbieten, musste diese in Software implementiert werden.

Die Wahl der Entwickler fiel auf den „eXtended Tiny Encryption Algorithm“ (XTEA) mit einer Schlüssellänge von 128 Bits, da dieser mit nur wenig Hauptspeicher auskommt und somit besonders für Sensorknoten geeignet ist. Überdies gilt der originale XTEA-Algorithmus derzeit als ungebrochen und somit als hinreichend sicher [12].

Die Erzeugung symmetrischer Schlüssel ist eine kritische Aufgabe. Wird hierzu lediglich ein Pseudozufallszahlengenerator eingesetzt, so ist es einem Angreifer, der den Initialwert des Generators bestimmen konnte möglich sämtliche mit diesem Generator erzeugte Schlüssel zu brechen. Um dieser Problematik zu entgehen setzt secFleck auf den TPM-Chip, welcher in der Lage ist starke Zufallszahlen zu erzeugen [12].

Als asymmetrisches Kryptographieverfahren wählten die Entwickler das RSA Verfahren aus. Hierzu ist anzumerken, dass die Sicherheit des RSA-Verfahrens von zwei Faktoren abhängt: Einerseits von der Schlüssellänge k und andererseits vom Exponenenten des öffentlichen Schlüssels, dem sog. „e-Wert“ [21]. Beide beeinflussen die Sicherheit des Verfahrens negativ, wenn sie zu klein gewählt werden. Die derzeitige Empfehlung für die minimale Schlüssellänge liegt bei 1369 Bit. Um darüber hinaus eine gewisse Zukunftssicherheit zu erreichen, wird diese deshalb gerne zur nächst größeren Zweierpotenz von 2048 Bit aufgestockt [17]. Der „e-Wert“ sollte nach Empfehlung der Entwickler nicht kleiner sein als $\log_2$(Schlüssellänge) [21]. SecFleck erfüllt beide Kriterien, indem es eine Schlüssellänge von 2048 Bits verwendet und einen „e-Wert“ des öffentlichen Schlüssels von 65.537 und kann somit ebenfalls als hinreichend sicher angenommen werden [12].

All diese Funktionen stellt secFleck dem Nutzer in einer API zur Verfügung, welche in Abbildung 4 dargestellt wird.

4.1.2 Geschwindigkeit

Den RSA Algorithmus mit den in Abschnitt 4.1.1 geforderten Werten für die Schlüssellänge bzw. für den „e-Wert“ zu betreiben ist für Sensorknoten mit ihrer beschränkten Rechenleistung eine Herausforderung. So würde das einmalige Durchführen einer Verschlüsselung mit 2048 Bit und einem „e-Wert“ von 65.537 auf einem Fleck^{TM}-System mehr als 7 Minuten benötigen und somit unpraktikabel werden (Tabelle 1) [12].

Für dieses Problem gibt es nun theoretisch zwei mögliche Lösungen: Reduzierung der Größe der Verschlüsselungsparameter mit Inkaufnahme von Sicherheitsrisiken oder - wie es von den secFleck-Entwicklern in [12] gemacht wurde - Anbindung eines kryptographischen Koprozessors. Dieser ist in der Lage, die Berechnung in Hardware durchzuführen, und ermöglicht so asymmetrische Kryptographie nahezu „in Echtzeit“ anzuwenden (vgl. Tabellen 1 und 2). Bei dem

```
1 /* Duty cycle TPM chip functions. */
2 uint8_t fos_tpm_startup(void);
3 uint8_t fos_tpm_turnoff(void);
4
5 /* True random number generator. */
6 uint8_t fos_tpm_rand(uint8_t *randNumber,
      uint8_t len);
7
8 /* secFleck public key collector. */
9 uint8_t fos_tpm_getPubKey(uint8_t *pubKey);
10
11 /* Asymmetric key encryption/decryption. */
12 uint8_t fos_tpm_encryption(uint8_t *msg,
      uint16_t len, uint8_t *pubKey, uint8_t *
      cipher);
13 uint8_t fos_tpm_decryption(uint8_t *cipher,
      uint8_t *msg, uint16_t *len);
14
15 /* Digital signature and verification. */
16 uint8_t fos_tpm_sign(uint8_t *digest, uint8_t
      *signature);
17 uint8_t fos_tpm_verifySign(uint8_t *signature
      , uint8_t *pubKey, uint16_t *digest);
18
19 /* Symmetric session key encryption/
      decryption. */
20 uint8_t fos_xtea_encipher(uint8_t *msg,
      uint8_t *key, uint8_t *cipher, uint8_t
      nRounds);
21 uint8_t fos_xtea_decipher(uint8_t *cipher,
      uint8_t *key, uint8_t *msg, uint8_t
      nRounds);
```

Abbildung 4: secFleckAPI [12]

Public Exponent (e)	Software 1024 bit	Software 2048 bit	Hardware 2048 bit
3	0,45s	65s	N/A
65.537	4,185s	450s	0,055s

Tabelle 1: Vergleich von Verschlüsselungszeiten bei RSA [12].

kryptographischen Koprozessor handelt es sich wie oben bereits erwähnt um ein TPM, dessen interne Funktionen durch die secFleck API (vgl. Abb. 4) dem Anwendungsentwickler zugängig gemacht werden.

Um einen vollständig ungestörten Betrieb trotz des Einsatzes von Kryptographie zu gewährleisten ist es notwendig, dass Daten schneller ver-/entschlüsselt werden können als sie über das Netzwerk gesendet werden. Den Autoren in [12] zufolge benötigt der NRF905 Transceiver 23,75μs um ein Bit zu versenden, während die asymmetrische Verschlüsselung eines Bits selbst mit Hardwareunterstützung noch 27μs benötigt. Da jedoch keine Notwendigkeit dafür besteht, Daten asymmetrisch zu verschlüsseln, sobald ein initialer symmetrischer Schlüssel übertragen wurde („Hybride Verschlüsselung“ Abschnitt 3.2), kann alternativ auch der XTEA eingesetzt werden. Dieser benötigt für die Verschlüsselung eines Bits nur noch 18μs, und ist somit signifikant schneller als der Transceiver (vgl. Tabelle 2).

Plattform	Stromstärke (mA)	Zeit (μs)	Energie (μJ)
RSA SW, 2048 bit	8,0	219,730	7.030,0
RSA HW, 2048 bit	50,4	27	5,4
XTEA SW, 128 bit	8,0	18	0,6

Tabelle 2: secFleck Stromaufnahme und Zeitverbrauch für die Verschlüsselung *eines* Bits [12].

Modul	Stromstärke (mA)
Fleck3 (Leerlauf)	8,0
Fleck3 + Empfangen	18,4
Fleck3 + Senden	36,8
Fleck3 + TPM verschlüsseln	50,4
Fleck3 + TPM entschlüsseln	60,8
Fleck3 + TPM signieren	60,8
Fleck3 + TPM Signatur prüfen	50,4

Tabelle 3: secFleck Stromaufnahme [12]

4.1.3 Energieverbrauch

Wie Tabelle 3 entnommen werden kann kostet der Betrieb des secFlecks mit aktiviertem TPM zehnmal mehr Energie als der ohne TPM. Dennoch handelt es sich um eine rationale Entscheidung den TPM-Chip anzubinden.

Der erste Grund dafür ist, dass die Verwendung des TPM-Chips gegenüber der reinen Softwarelösung sogar noch Energie einspart. Tabelle 2 stellt diese Ersparnis sehr eindrücklich dar: 7.030,0μJ wird bei der reinen Softwarelösung für die Verschlüsselung eines Bits benötigt, während bei der Verwendung des TPM-Chips dazu nur 5,4μJ nötig sind. Sofern also die Verwendung von asymmetrischer Kryptographie gewünscht ist, ist die von secFleck verwendete Lösung der Softwarelösung vorzuziehen.

Zudem muss bedacht werden, dass die secFleck API des TPM-Chips die Möglichkeit bietet den TPM-Chip explizit ein- bzw. auszuschalten (vgl. Abb. 4). Dies ist deshalb besonders interessant, da der TPM-Chip bei der Verwendung Hybrider Verschlüsselungsmethoden nur während der initialen Schlüsselaustauschphase benötigt wird. Diese ist im Verhältnis zur gesamten Betriebszeit des Moduls nur relativ kurz, weshalb davon ausgegangen werden kann, dass der durch den TPM-Chip hervorgerufene Mehrverbrauch an Energie die Möglichkeiten des Sensorknotens nicht übersteigt.

4.1.4 Kosten

Die Kosten des von den Autoren in [12] verwendeten TPM-Chips „Atmel AT97SC3203S“ lagen den Angaben der Autoren nach bei $4.5, und machten somit weniger als 5% der Gesamtkosten eines gängigen Sensorknotens aus [12].

Diese Angabe ist nur schwer zu überprüfen. Zwar liegen die Preise gängiger Sensorknoten nach wie vor um die $100, es war zum Zeitpunkt der Anfertigung dieser Arbeit jedoch nicht möglich aktuelle Preisinformationen für den in [12] verwendeten TPM-Chip zu erhalten. Es konnte jedoch der Preis des in seinen Leistungsdaten vergleichbaren „Infinion SLB9635TT 1.2“ TPM-Chips zu 2€ - also in etwa $2,6 - bestimmt werden [9]. Der Einbau von TPM-Chips in Sensorknoten bewegt sich also damals wie heute im vertretbaren Kostenrahmen.

4.2 secFleck Features

SecFleck bietet dem Anwender als Gesamtsystem eine Reihe von Features, die im Folgenden kurz beschrieben werden.

Die secFleck-API erlaubt es, kryptographische Schlüssel nicht nur im EEPROM oder dem RAM des Knotens zu speichern, sondern auch im EEPROM des TPMs. Möglich wird dies dadurch, dass der von den secFleck-Autoren verwendete TPM-Chip Atmel AT 97SC3203S über den TPM v1.2 Standard hinaus über einen internen, sicheren Speicher verfügt. Das Feature der Schlüsselspeicherplatzwahl ist ein außerordentlich wichtiges: Einerseits wurde in [11] gezeigt, wie einfach es ist, den RAM und den EEPROM eines Knotens auszulesen, wodurch die Möglichkeit des Speicherns im TPM EEPROM besonders wichtig wird. Andererseits erfordert das Speichern der Schlüssel im TPM-Chip einen höheren Energieverbrauch als das im Knoten selbst. Weniger kritische Schlüssel sollten deshalb auch im Knoten selbst gespeichert werden können [12].

Als zweites wichtiges Feature preisen die Autoren in [12] secFlecks sichere Sitzungsschlüssel-Verwaltung an. Nicht nur, dass es die secFleck-API ermöglicht komfortabel mit Hilfe asymmetrischer Verfahren symmetrische Schlüssel für die direkte Kommunikation zwischen Basisstation und den Knoten zu verteilen, auch die Etablierung von Gruppenkommunikationsschlüssel wird unterstützt. Auf diese Weise ist es z.B. möglich, allen Knoten des WSNs denselben Schlüssel mitzuteilen, und so das gesamte Netzwerk vertraulich kommunizieren zu lassen [12].

Ein weiteres Feature das secFleck seinen Anwendern zur Verfügung stellt, ist die sichere Reprogrammierung von Sensorknoten. Dieses Feature ist deshalb so wichtig, weil es zwar bereits eine Reihe von „Multihop Over the Air Programming“ (MOAP) Protokollen gibt, der Sicherheitsaspekt bei diesen bisher jedoch vernachlässigt wurde. Ein Beispiel für ein solches System wäre Deluge [14]. Soll ein WSN mithilfe von Deluge umprogrammiert werden, so wird zunächst ein neues Programm-Image erstellt und dieses in kleine Chuncks aufgeteilt. Diese werden dann solange von Sensorknoten zu Sensorknoten verteilt, bis alle Knoten die neueste Programmversion besitzen. SecFleck erweitert dieses System um zwei Sicherheitsmechanismen. Einerseits werden die Chunks nach ihrer Erstellung durch den Programmierer unterschrieben, wodurch es den Knoten möglich wird die Integrität und die Quelle des neuen Programms zu prüfen. Andererseits bietet secFleck wie oben beschrieben die Möglichkeit einen globalen Gruppenschlüssel einzurichten und so die Vertraulichkeit der übertragenen Daten zu gewährleisten [12].

Das letzte von den Autoren in [12] angepriesene secFleck-Feature ist das des sicheren „Remote Procedure Calls“ (RPC). Bereits das normale FOS bietet Anwendungsprogrammen die Möglichkeit per RPC z.B. den Batteriezustand von Sensorknoten abfragen oder ihren RAM bzw. ihren EEPROM auslesen. Auch hier setzt secFleck wieder auf Sicherheit durch einen globalen symmetrischen Gruppenschlüssel. Dies stellt

sicherlich nicht die sicherste Variante dar, da lediglich die Vertraulichkeit - nicht jedoch die Verbindlichkeit oder die Integrität sichergestellt werden kann. Zudem führt die Kompromittierung des Schlüssels zu einem globalen Sicherheitsproblem. Der Vorteil eines einzigen globalen Schlüssels liegt jedoch auf der Hand: Eine einzelne RPC-Anfrage kann an viele Sensorknoten geschickt werden, ohne dass vorher bekannt sein muss, welcher Knoten wirklich die Anfrage bearbeiten wird.

5. VERWANDTE ARBEITEN

Neben dem in dieser Arbeit vorgestellten Ansatz die Kommunikation in drahtlosen Sensornetzwerken sicherer zu gestalten gibt es noch zahlreiche Weitere. Einige davon seien hier kurz vorgestellt:

In [15] wird mit TinySec eine Link-Layer-Sicherheitsarchitektur vorgestellt. Diese ist in das offizielle TinyOS-Release eingearbeitet worden und steht somit einer breiten Community zur Verfügung. Gegenüber dem von secFleck gewählten Weg der Ende-zu-Ende Verschlüsselung bietet der von TinySec gewählte Weg der Link-Layer-Verschlüsselung sowohl deutliche Vor- als auch Nachteile. Beispielsweise gehen die Autoren in [15] davon aus, dass Ereignisse meist von mehr als nur einem Sensorknoten aufgenommen werden. Im Falle einer Ende-zu-Ende Verschlüsselung müssen alle Nachrichten - auch wenn sie einen identischen Inhalt enthalten - dem Empfänger zugestellt werden. Unter Verwendung einer Link-Layer-Verschlüsselung können Duplikate frühzeitig erkannt und entfernt werden. Die aktuelle TinySec Implementierung lässt jedoch einige Fragestellungen wie die der effektiven Einbindung asymmetrische Verfahren unbeantwortet. Weitere Architekturen mit Link-Layer-Sicherheitsmechanismen wären beispielsweise GSM, Bluetooth oder 802.15.4 [15].

Auch Ansätze ein WSN auf der Network-Layer abzusichern existieren. So wird beispielsweise in [13] mit Ariadne eine Sicherheitserweiterung für das „Dynamic Source Routing“ (DSR) Routing-Protokoll vorgestellt, welches unter anderem eine Reihe von Denial-of-Service-Angriffen auf das WSN ausschließt.

Handelt es sich bei den Sensorknoten des Netzwerkes statt um eingebettete um vollwertige Rechensysteme, so können auch die aus dem Internet bekannten Sicherheitsmechanismen wie IPSec [7] oder TLS [6] eingesetzt werden. So wird in [16] beispielsweise IPSec eingesetzt, um eine Sicherheitsarchitektur für globale Raketenabwehrsysteme zu entwickeln.

Eine Liste aktueller Publikationen zum Thema Sicherheit in drahtlosen Sensornetzwerken kann in [3] gefunden werden.

6. ZUSAMMENFASSUNG UND AUSBLICK

Zusammenfassend lässt sich sagen, dass der von secFleck gegangene Weg der Hardwareanbindung eines TPM-Chips zur Verfügbarmachung von asymmetrischen Verschlüsselungsverfahren in drahtlosen Sensornetzwerken Erfolg verspricht. Nicht nur ermöglicht der TPM-Chip eine vollwertige, unbeschnittene RSA-Verschlüsselung, er führt diese zudem auch nahezu „in Echtzeit“ durch. Der Preis, der für diese Funktionalität gezahlt werden muss ist der deutlich höhere Stromverbrauch gegenüber einem Fleck^{TM}-System ohne TPM-Chip. Da jedoch der TPM-Chip abschaltbar ist und bei Verwendung eines hybriden Verschlüsselungsverfahrens zudem nur sporadisch zum Einsatz kommt, stellt dies einen akzeptablen Preis dar.

Kritisch sei an dieser Stelle angemerkt, dass der TPM-Chip über wesentlich mehr Fähigkeiten verfügen würde, als tatsächlich genutzt werden. So ist gerade das Feature des beglaubigten Bootvorgangs in einer typischen WSN-Umgebung, in der keineswegs davon ausgegangen werden kann, dass jeder Knoten unmanipuliert ist, von extrem hohem Wert. Auch für die Features der überprüfbaren anonymen Identitäten und des sicheren, an den Zustand des Systems gebundenen Speicherplatzes sind Szenarien vorstellbar, in denen sie einen erheblichen Nutzen darstellen würden.

Für die Zukunft wäre also durchaus ein secFleck-v2 mit einer besseren Ausnutzung des TPM-Chips vorstellbar. Auf diese Weise würde secFlecks vielversprechendes Potential noch besser ausgenutzt werden.

7. LITERATUR

[1] Australian commonwealth scientific and research organization (csiro). `http://www.csiro.au`.

[2] Datacall telemetry. `http://www.datacall.net.au`.

[3] H. Alzaid. Wireless sensor networks security, 2010 5. `http://www.wsn-security.info/Security_Lounge.htm`.

[4] B. Carter, A. Kassin, and T. Magoc. Advanced Encryption Standard. 2007.

[5] P. Corke and P. Sikka. Fos-a new operating system for sensor networks. In *Proceedings of the 5th European Conference on Wireless Sensor Networks (EWSN08)*, Brisbane, Australia, 2008. CSIRO ICT Centre.

[6] T. Dierks and C. Allen. The TLS protocol version 1.0, 1999. RFC 2246, January 1999.

[7] N. Doraswamy and D. Harkins. *IPSec: the new security standard for the Internet, intranets, and virtual private networks*. Prentice Hall, 2003.

[8] D. Eastlake and P. Jones. US secure hash algorithm 1 (SHA1), 2002. RFC 3174, September 2001.

[9] EBV Elektronik GmbH & Co. KG. `http://www.ebv.com`.

[10] C. Eckert. *IT-Sicherheit: Konzepte-Verfahren-Protokolle*. Oldenbourg, 5 edition, 2008.

[11] C. Hartung, J. Balasalle, and R. Han. Node compromise in sensor networks: The need for secure systems. *Department of Computer Science University of Colorado at Boulder*, 2005.

[12] W. Hu, P. Corke, W. C. Shih, and L. Overs. secfleck: A public key technology platform for wireless sensor networks. In *Wireless Sensor Networks*, volume 5432 of *Lecture Notes in Computer Science*, pages 296–311. Springer Berlin / Heidelberg, 2009.

[13] Y. Hu, A. Perrig, and D. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. *Wireless Networks*, 11(1):21–38, 2005.

[14] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94, New York, NY, USA,

2004. ACM.

[15] C. Karlof, N. Sastry, and D. Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 162–175, New York, NY, USA, 2004. ACM.

[16] G. Katsis and NAVAL POSTGRADUATE SCHOOL MONTEREY CA. Multistage Security Mechanism For Hybrid, Large-Scale Wireless Sensor Networks. 2007.

[17] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. In *PKC '00: Proceedings of the Third International Workshop on Practice and Theory in Public Key Cryptography*, pages 446–465, London, UK, 2000. Springer-Verlag.

[18] M. Li and Y. Liu. Underground coal mine monitoring with wireless sensor networks. *ACM Trans. Sen. Netw.*, 5(2):1–29, 2009.

[19] C. Mitchell. Trusted computing, August 2006. `http://www.isg.rhul.ac.uk/cjm/060823.zip`.

[20] R. Needham and D. Wheeler. eXtended Tiny Encryption Algorithm, October 1997.

[21] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[22] P. Sikka, P. Corke, and L. Overs. Wireless sensor devices for animal tracking and control. *Local Computer Networks, Annual IEEE Conference on*, 0:446–454, 2004.

[23] D. Standard. Federal Information Processing Standards Publication 46. *National Bureau of Standards, US Department of Commerce*, 1977.

[24] N. Sumrall and M. Novoa. Trusted Computing Group (TCG) and the TPM 1.2 Specification. In *Intel Developer Forum*, volume 32, March 2007.

[25] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Commun. ACM*, 47(6):34–40, 2004.

[26] The Powercom Group. Fleck long range wireless sensing and control, November 2008. `http://www.powercomgroup.com/Latest_News_Stories/Fleck_long_range_wireless_sensing_and_control.shtml`.

[27] TIK WSN Research Group. The sensor network museum™, 2010. `http://www.snm.ethz.ch/Projects/Fleck`.

Lightweight Key Management in Wireless Sensor Networks by Leveraging Initial Trust

Marc Ströbel
Betreuerin: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: stroebem@in.tum.de

KURZFASSUNG

In vielen Anwendungsgebieten von Sensornetzwerken ist die Authentizität, Integrität und Vertraulichkeit erfasster Daten eine wichtige Voraussetzung für einen erfolgreichen Einsatz. Um diesen Anspruch zu gewährleisten, muss in erster Linie die drahtlose Kommunikation innerhalb eines Sensornetzwerkes verschlüsselt werden. Da die Rechenleistung und Speicherkapazität einzelner Knoten in Sensornetzwerken jedoch stark begrenzt ist, kann dies im Allgemeinen nicht mit konventionell benutzten Verfahren zum Aufbau sicherer Verbindungen, wie etwa Diffie-Hellman, realisiert werden.
Das Paper *Lightweight Key Management in Wireless Sensor Networks by Leveraging Initial Trust* stellt ein günstiges Verfahren zum Schlüsselaustausch zwischen Sensorknoten vor, welches nur auf einer geringen Menge von vorweg ausgetauschten Schlüsseln basiert. Zudem begnügt es sich mit dem Gebrauch symmetrischer Kryptographie und benötigt keine vertrauenswürdige Basisstation.

Schlüsselworte

Lightweight Key Management, Wireless Sensor Networks, Initial Trust

1. EINLEITUNG

Ursprünglich entwickelt in der Rüstungsindustrie für den militärischen Kontext, haben Sensornetzwerke heute bereits Einzug in viele Anwendungen in unserem alltäglichen Leben gewonnen. Im Vergleich zu traditionell eingesetzten Sensoren, besitzen Sensoren in drahtlosen Sensornetzwerken aufgrund ihrer Autonomie nur sehr begrenzte Ressourcen. Ihre unabhängige Energiequelle und drahtlose Datenübertragung ermöglicht jedoch den ausdrücklich gewünschten Einsatz in schwer erreichbaren Zielgebieten, oder in mobilen Szenarien [1].
Dass Sicherheitsaspekte in Sensornetzwerken, auch außerhalb des militärischen Bereichs, von kritischer Bedeutung sind, zeigt ein Einsatzszenario von Sensornetzwerken als tragbare Technologie im Gesundheitswesen: Ein Patient wird mit mehreren tragbaren Geräten ausgestattet, welche bestimmte Vitalparameter überwachen und deren Werte per Funk in das Netzwerk des Krankenhauses übermitteln. Diese Anwendung von Sensornetzwerken ermöglichen einem Patienten mehr Freiräume während seiner Genesung, in manchen Fällen sogar einen kürzeren Klinikaufenthalt mit einer anschließenden elektronischen Überwachung zu Hause [9]. Offensichtlich kann eine ungesicherte Kommunikation in dieser Anwendung nicht nur zu einer empfindlichen Verletzung der Privatsphäre des Patienten führen, sondern im schlimmsten Fall auch zu einer falschen Behandlung, aufgrund kompromittierter Datensätze.
Um eine sichere Kommunikation zwischen den Sensorknoten zu gewährleisten, und somit Authentizität, Integrität und Vertraulichkeit erfasster Daten sicherzustellen, wird ein *effizienter* Schlüsselaustauschmechanismus benötigt. *Effizienz* ist erforderlich, da Sensorknoten im Allgemeinen nur sehr limitierte Rechenleistung besitzen und daher aufwändige Verfahren, wie Public-Key-Verschlüsselung oder das Diffie-Hellman Schlüsselaustauschprotokoll, zu teuer sind [4]. Ein möglicher Lösungsansatz umgeht diese Problematik durch den Einsatz vertrauenswürdiger Basisstationen. Eine Basisstation ist ein zentraler Knoten im Sensornetzwerk, welcher sowohl über unterbrechungsfreie Energie, als auch leistungsstarke Hardware verfügt. Neben der Speicherung und Weiterleitung eingehender Daten der Sensorknoten, wird eine Basisstation oftmals auch als verauswürdiger Mittler eingesetzt, der Sensorknoten im Netzwerk gegenseitig authentifiziert und paarweise Schlüssel bereitstellt. So können sich Sensoren gegenüber der Basisstation durch ein langfristiges Geheimnis authentifizieren, welche mit ihrer Rechenleistung, starke Verfahren zur Schlüsselerzeugung einsetzten kann.
Dutertre et al. identifizieren folgende Schwächen in diesen Verfahren:

- Zusätzlicher administrativer Aufwand: Noch vor dem Aufstellen des Sensornetzwerkes im entsprechenden Zielgebiet, muss ein Austausch eines langfristigen, individuellen Geheimnisses zwischen Basisstation und jedem Knoten des Netzwerkes erfolgen.
- Die Eigenschaft der Basisstation als Single-Point-of-Failure: Die Basisstation bietet ein ideales Ziel für einen Denial-of-Service Angriff und im Falle einer Kompromittierung verliert jeglicher Schlüssel im Sensornetzwerk seine Gültigkeit.

Darüber hinaus existieren Netztopologien für Sensornetzwerke, welche bewusst auf eine Basisstation verzichten, und somit auf ein Schlüsselaustauschverfahren für limitierte Rechenleistung angewiesen sind.
In [7] entwerfen Dutertre et al. ein Schlüsselverwaltungsverfahren mit minimalen administrativen Aufwand, welches basierend auf einen minimalen Anteil an bereits vorweg ausgetauschten Schlüsseln, sichere Verbindungen für Untermen-

gen eines Sensornetzwerkes aufbauen kann – ohne aufwändige Verschlüsselungsverfahren oder vertrauenswürdigen Basisstationen.

Im Kapitel 2 werden zunächst einige Zeichen und Abkürzungen als Notation eingeführt. Anschließend behandelt Kapitel 3 den Ablauf und die Funktionsweise Dutertre et al.'s Schlüsselverwaltungsverfahrens. Kapitel 4 beinhaltet eine objektive Bewertung der Sicherheit und Ausfallsicherheit des Verfahrens, und zieht einen Vergleich mit ausgewählten Protokollen zur Schlüsselverwaltung. Nach einer kurzen Vorstellung der Implementation des Lightweight Key Management für das Betriebssystem TinyOS in Kapitel 5, stellt Kapitel 6 eine Zusammenfassung der Arbeit dar.

2. NOTATION

Um das Schlüsselaustauschverfahren von Dutertre et al. [7] im Folgenden genauer beschreiben zu können, werden zunächst einige Abkürzungen für einzelne Schlüssel und kryptographische Funktionen einführt:

A, B, C, ...	einzelne Sensorknoten
N_a	eine Zufallszahl, generiert vom Sensorknoten A
R_a	eine Zufallszahl, vor dem Aufstellen abgelegt im Sensorknoten A
$G_k(m)$	Hashfunktion mit Schlüssel k über Zeichenkette m
$MAC_k(m)$	Message-Authentication-Code für Nachricht m erzeugt mit Schlüssel k
bk_1	Authentifikationschlüssel für das Bootstrapping
bk_2	Schlüssel zur Schlüsselgenerierung für das Bootstrapping
gk_i	Authentifikations Schlüssel für Generation i
K_{ab}	Gemeinsamer Schlüssel zwischen den Sensorknoten A und B
$ENC_{K_{ab}}(m)$	Verschlüsselung der Nachricht m mit dem Schlüssel K_{ab}

Die Hashfunktion G mit Schlüssel k hat die Eigenschaft, dass der mit ihr berechnete Schlüssel r unter der Eingabe von Schlüssel k und der Zeichenkette m ($G_k(m) = r$) effizient berechenbar ist, jedoch ohne die Kenntnis von k keine Aussage über r getroffen werden kann.
Der Message-Authentication-Code $MAC_k(m)$ ist ebenfalls eine Hashfunktion mit Schlüssel k, welche zur Authentifizierung von Nachrichten verwendet wird: Wenn ein Knoten A einen gemeinsamen Schlüssel K_{ab} mit Knoten B teilt, kann A den Hashwert $MAC_{K_{ab}}(m)$ seiner Nachricht m zu B mitschicken. B kann nun überprüfen, ob die Nachricht m wirklich von A versendet wurde, indem es den Hashwert selbst berechnet und mit dem angehängten Hashwert von A vergleicht. Dieses Verfahren sichert sowohl die Authentizität als auch die Integrität einer Nachricht [8, Kap. 7].
Die Abkürzung $ENC_{K_{ab}}(m)$ steht für die Verschlüsselung der Nachricht m mit dem Schlüssel K_{ab} und ermöglicht den Knoten A und B, wenn sie im Besitz dieses Schlüssels sind, vertrauliche Kommunikation.
Des Weiteren betrachten wir im folgenden Kapitel das Sensornetzwerk S mit den Knoten $\{A, B, C, D\}$.

3. DAS PROTOKOLL

Das Schlüsselaustauschverfahren erfolgt in zwei Phasen: Zunächst baut jeder Knoten sichere Kanäle (Secure Local Links) zu allen Nachbarknoten in seiner Reichweite auf. Im zweiten Schritt können mit Hilfe dieser sicheren Kanäle, welche das gesamte Sensornetzwerk verbinden, gemeinsame Schlüssel für Gruppen von Knoten vereinbart werden.

3.1 Secure Local Link

Das Initialisieren des sicheren Kanals – von Dutertre et al. auch als Bootstrapping ihres Protokolls bezeichnet – beruht im Wesentlichen auf den gemeinsamen Schlüsseln bk_1 und bk_2, welche vor dem Aufstellen auf allen Knoten von S abgelegt werden.
Betrachten wir die Initialisierung eines Kanals zwischen den Knoten A und B (siehe Abbildung 1): Nach der Aktivierung des Sensornetzwerkes, beginnt Knoten A mit dem Broadcast der Nachricht $< Hello, A, N_a, MAC_{bk1}(Hello, A, N_a) >$, welche neben seiner Identität A, eine Nonce N_a sowie den Message-Authentication-Code $MAC_{bk1}(Hello, A, N_a)$ über der gesamten Nachricht enthält. Der Message-Authentication-Code erzeugt mit dem gemeinsamen Schlüssel bk_1 authentifiziert A bei dem jeweiligen Empfänger als Mitglied des Sensornetzwerkes S. Wird die Nachricht von B oder einem anderen Mitglied von S empfangen, sendet dieses in einer Empfangsbestätigung die eigene Identität B, eine Nonce N_b und einen Message-Authentication-Code zurück. Da auch B seiner Nachricht einen MAC über m und der Nonce N_a, erzeugt mit bk_1 anhängt, haben sich A und B gegenseitig authentifiziert.
Im nächsten Schritt berechnen beide Knoten den gemeinsamen Schlüssel K_{ab} aus der Funktion $G_k(m)$, den beiden Nonces N_a, N_b und dem gemeinsamen Schlüssel bk_2: $K_{ab} = G_{bk_2}(N_a N_b)$. K_{ab} ermöglicht den Knoten A und B zukünftig, sicher und effizient durch symmetrische Verschlüsselung zu kommunizieren.

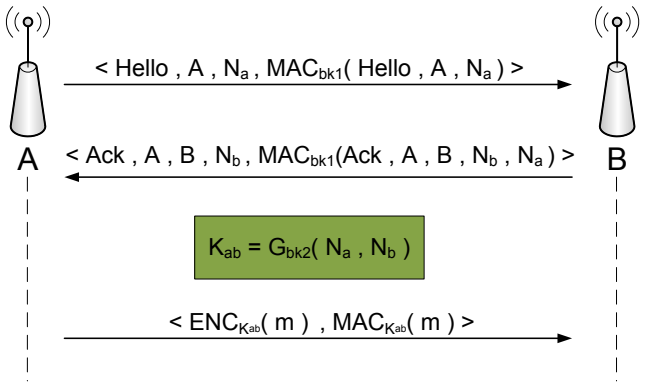


Abbildung 1: Aufbau eines Secure Local Links

Hat ein Knoten sichere Kanäle zu den in seiner Reichweite liegenden Nachbarknoten aufgebaut, werden die gemeinsamen Schlüssel bk_1 und bk_2 gelöscht. Dieses Verhalten stellt sicher, dass eine spätere Kompromittierung des Knotens, nicht jeglichen ausgehandelten Schlüssel gefährdet (siehe auch Kapitel 4.1).

3.2 Group-Key Distribution

Die Secure Local Links können mit sogenanntem *chaining* – dem Weiterleiten einer Nachricht von Knoten zu Knoten, auch zum Kommunizieren zwischen zwei entfernten Knoten verwendet werden:

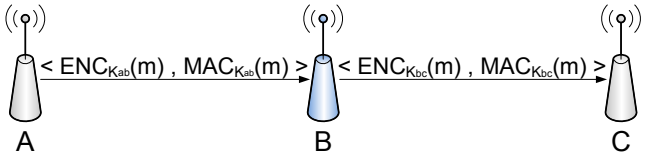


Abbildung 2: Verkettung einer Nachricht vom Sender A zum Empfänger C

Um eine teurere Neuverschlüsselung an jedem Knoten für Multicast- oder Broadcastnachrichten zu verhindern, kann über die aufgebauten lokalen Kanäle ein Gruppenschlüssel ausgehandelt werden, welcher Message Chaining für Multicast durch eine einmalige Verschlüsselung ersetzt. Dazu besitzt Dutertre et al.'s Protokoll den Mechanismus *KeyRefresh*, welcher einen Gruppenschlüssel K_g an alle gewünschten Knoten im Sensornetzwerk versendet. Ein KeyRefresh von A, gesendet an B, hat die Form:

$$< KeyRefresh, B, A, O, N, ENC_{K_{ab}}(K_g), L, \\ MAC_{K_{ab}}(KeyRefresh \dots) >$$

Während der Wert O für den Herkunftsknoten des neuen Schlüssels steht, bezeichnet L eine Liste auszuschließender Knoten. L kann dual genutzt werden, einerseits um einen gemeinsamen Schlüssel nur zwischen einer Untermenge des Sensornetzwerkes zu instantiieren, andererseits um mögliche kompromittierte Knoten aus der Gruppe auszuschließen (siehe auch Kapitel 4.1). Für letzteren Fall existiert auch das Feld N, welches die Sequenznummer des Gruppenschlüssels angibt: mit ihm wird sichergestellt, dass ein bereits ausgeschlossener Knoten keine neuen Schlüssel über eine KeyRefresh-Nachricht verbreitet. Der zu verteilende neue Gruppenschlüssel K_g wird für Dritte unlesbar ($ENC_{K_{ab}}(K_g)$), verschlüsselt mit dem paarweisen Schlüssel von A und B, in die Nachricht eingefügt. Der MAC wird analog zu den anderen Protokollnachrichten wieder über den Inhalt der gesamten Nachricht berechnet.
Wenn Knoten A also einen Schlüssel K_{g1} zwischen ihm und den Knoten B und C instantiieren möchte, versendet er die Nachricht $< KeyRefresh, B, A, A, 0, ENC_{K_{ab}}(K_{g1}), \{D\}, MAC_{K_{ab}}(KeyRef \dots) >$ an seinen einzigen Nachbarn B, welcher die Nachricht an C weiterleitet: $< KeyRefresh, C, B, A, 0, E_{K_{bc}}(K_{g1}), \{D\}, MAC_{K_{bc}}(KeyRef \dots) >$.
Multicast-Nachrichten von A können nun per Broadcast im Sensornetzwerk verteilt werden (siehe Abbildung 3), wobei der Schlüssel K_{g1} sicherstellt, dass nur B und C die Nachrichten lesen können.

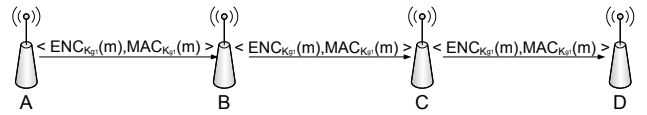


Abbildung 3: Senden einer Multicast-Nachricht von A nach B und C

3.3 Multiphase Deployment

Oftmals werden in Sensornetzwerken, zu einem späteren Zeitpunkt, neue Sensoren hinzugefügt oder defekte Sensoren ausgetauscht. Diese benötigte Flexibilität muss auch vom Schlüsselverwaltungsprotokoll unterstützt werden. Dutertre et al. haben daher das *Multiphase Deployment* in ihr Protokoll implementiert. Das Multiphase Deployment sieht vor, dass verschiedene Generationen von Sensorknoten, zum Beispiel die Generationen j bis n, nacheinander eingesetzt werden können. Untereinander etablieren die Knoten einer Generation paarweise Schlüssel mit den gemeinsamen Geheimnissen bk_1 und bk_2 ihrer Generation (siehe Kapitel 3.1). Weil generationsübergreifend bk_1 und bk_2 verschieden sind, wird jeder Knoten X einer Generation i mit einer Zufallszahl R_x und einer Menge an Schlüsseln $S_{x,(i+1)}, .., S_{x,n}$ initialisiert. $S_{x,i}$ ist zwar mit der Hashfunktion G berechenbar, es gilt jedoch, dass $G_{gk_i}(R_x) = S_{x,i}$ nur mit Hilfe des Generationenschlüssels gk_i effizient berechenbar ist. Der Generationenschlüssel gk_i der Generation i steht jedoch nur Sensoren der Generation i zur Verfügung.
Zurück zum Beispiel des Sensornetzwerkes S: Sei S für den Einsatz von zwei Generationen angelegt, so besitzt jeder der Knoten $\{A, B, C, D\}$ aus der ersten Generation eine Zufallszahl $R_a, .., R_d$, sowie ihren zugehörigen Schlüssel für die zweite Generation $S_{a,2}, .., S_{d,2}$. Besteht nun die zweite Generation aus dem Sensorknoten E, welcher in unmittelbarer Nähe von A platziert wird, dann hat der Schlüsselaustausch zwischen E und A folgende Form (siehe Abbildung 4): E sendet die Nachricht $< Hello, E, 2, N_E >$, welche neben dem Absender, die Kennzahl 2 für E's Generation und eine Zufallszahl enthält. Diese Nachricht besitzt keinen Message-Authentication-Code, da A und E bisher keinen gemeinsamen Schlüssel besitzen. Erhält A die Nachricht und befindet sich, wie in unserem Falle, im Besitz eines Schlüssels für die zweite Generation, schickt A eine Empfangsbestätigung $< Ack, A, E, R_a, MAC_{S_{a,2}}(Ack, A, E, R_a, N_E) >$. Diese Empfangsbestätigung enthält die Zufallszahl R_a und einen MAC erzeugt mit $S_{a,2}$. A authentifiziert sich somit als Teil einer vorausgegangenen Generation im Sensornetzwerk S. E kann den Message-Authentication-Code überprüfen, indem es $S_{a,2}$ aus G, gk_2 und R_a ($G_{gk_2}(R_a) = S_{a,2}$) berechnet. Anschließend authentifiziert E sich gegenüber A in einer weiteren Empfangsbestätigung und schließt somit den Schlüsselaustausch zwischen ihnen ab. Im weiteren Verlauf können die beiden Knoten über den Schlüssel $S_{a,2}$ sicher kommunizieren.

4. DISKUSSION

Grundsätzlich unterliegt ein Schlüsselverwaltungsverfahren gewissen Annahmen und Einschränkungen, welche Sicherheit und Ausfallsicherheit des Verfahrens betreffen, im Besonderen im Verfahren von Dutertre et al. aufgrund der Absicht, ein leichtgewichtiges Protokoll zu entwerfen. Das folgende Kapitel wirft eine Blick auf diese Eigenschaften und vergleicht das Protokoll mit ähnlichen Arbeiten.

4.1 Sicherheit und Ausfallsicherheit

4.1.1 Bootstrapping

Die paarweise Authentifizierung und Erzeugung eines gemeinsamen Schlüssels beruht auf einem Verfahren, das bereits 1993 von Bellare und Rogaway vorgestellt wurde [2]. Bellare und Rogaway haben bewiesen, dass ihr Verfahren

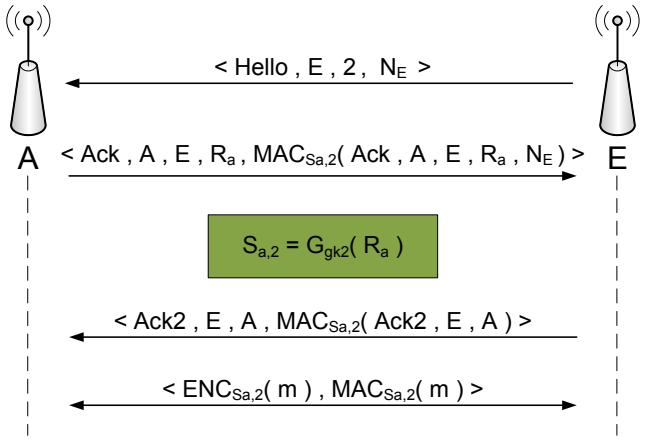


Abbildung 4: Schlüsselaustausch zwischen verschiedenen Generationen von Sensorknoten

ausreichende Sicherheit gegen einen Angreifer, der sowohl den Netzverkehr abhören, als auch Nachrichten in das Netz einspeisen kann, bietet. Die Voraussetzung für diese Sicherheit ist jedoch, dass der Angreifer weder Kenntnis des Schlüssel bk_1 noch des Schlüssels bk_2 besitzt. Eine größere Gefahr geht im Szenario der Sensornetzwerke von der Kompromittierung einzelner Knoten aus: Sollte es einem Angreifer gelingen, einen Knoten zu übernehmen und die Bootstrapping-Schlüssel bk_1 und bk_2 abzurufen, kann er an jegliche Schlüssel durch Abhören der Nonces gelangen. Darüber hinaus kann er sich selbst als Teil des Sensornetzwerkes gegenüber den anderen Knoten authentifizieren. Aus diesem Grund sollte das Zeitfenster des Bootstrappings so kurz wie möglich gehalten werden, um die Schlüssel bk_1 und bk_2 anschließend zu löschen.
Im Bezug auf die Ausfallsicherheit besitzt das Bootstrapping mehrere negative Bedingungen:

- Das Bootstrapping soll auf einen möglichst kurzen Zeitraum beschränkt sein, um die Sicherheit der Schlüssel bk_1 und bk_2 zu gewährleisten.
- Die Funkverbindung der Sensorknoten ist unzuverlässig.
- Das Erreichen von Nachbarknoten basiert auf Broadcastnachrichten, was dazu führen kann, dass Kollisionen mehrerer Nachrichten auftreten.

Um trotzdem eine robuste Ausfallsicherheit zu gewährleisten, initiieren die Sensoren ihre Nachrichten zu zufällig gewählten Zeitpunkten, um die Wahrscheinlichkeit von Kollisionen gering zu halten. Zusätzlich ist es möglich, die Sensoren auf das Versenden mehrerer *Hello*-Nachrichten zu konfigurieren, so dass im Falle von Paket-Kollisionen eine zweite Chance zur Erzeugung eines gemeinsamen Schlüssels besteht.

4.1.2 Key-Refresh

Solange der Initiator und alle Empfänger eines Key-Refresh vertrauenswürdig sind, besteht sicherlich keine Gefahr für eine Kompromittierung des gemeinsamen Schlüssels. Unmittelbar problematisch wird es, wenn es einem Angreifer gelingt, einen Knoten der Gruppe zu übernehmen und eventuell sogar durch das Initiieren einer eigenen Key-Refresh-Nachricht andere Knoten aus der Gruppe auszuschließen. Letztlich stellt sich die Frage wie ein Knoten die Übernahme eines anderen Gruppenknotens entdecken kann, um folglich einen neuen Schlüsselaustausch unter Ausschluss dieses Knotens durchzuführen. Diese Problematik führt in den Bereich der *Intrusion Detection* Systeme für Sensornetzwerke. Als weiterführende Literatur kann zum Beispiel die Arbeit [5] betrachtet werden.

4.1.3 Multiphase Deployment

Analog zum Bootstrapping muss auch im Multiphase Deployment darauf geachtet werden, den Schlüsselaustausch auf einen möglichst kurzen Zeitraum zu beschränken, da der Verlust des Schlüssels gk_i der Generation i, alle paarweisen Schlüssel zwischen Generation i und Knoten aus vorausgegangenen Generationen kompromittiert.
Darüber hinaus muss beachtet werden, dass ein Schlüssel $S_{a,i}$ eines Knotens A zu einem Sensor der Generation i identisch zu weiteren Schlüsseln zwischen A und anderen Sensoren der Generation i sind, da alle auf dem Geheimnis der Nonce R_a beruhen. So ist es einem Angreifer möglich, durch die Übernahme eines Knotens, eine Vielzahl von paarweisen – sicher geglaubten – Verbindungen zu kompromittieren.

4.2 Das Protokoll im Vergleich

Die große Schwachstelle von Dutertre et al.'s Protokoll ist die Tatsache, dass ein Angreifer im Besitz der Schlüssel bk_1, bk_2 und der Generationenschlüssel $gk_j, .., gk_n$ alle Schlüssel in einem Sensornetzwerk kompromittieren könnte.
Andere Verfahren wie zum Beispiel das *q-Composite Random Key Predistribution Scheme*, vorgestellt in [6], oder das *Location-based pairwise Key Establishment* [10], bieten eine höhere Sicherheit, bringen jedoch den, von Dutertre et al. missbilligten, großen administrativen Aufwand mit sich:
Das *q-Composite Random Key Predistribution Scheme* verteilt auf jedem Knoten des Sensornetzwerkes eine zufällige Teilmenge, mit der Kardinalität n, aus einer großen Menge B von Schlüsseln. Haben nun zwei Nachbarknoten mehr als q ($q > 1$) gemeinsame Schlüssel, benutzen sie einen Hash all ihrer gemeinsamen Schlüssel als Schlüssel für ihren sicheren Kanal. Die n Schlüssel sind mit einer bestimmten Wahrscheinlichkeit p aus B gewählt, um eine Mindest-Konnektivität im Sensornetzwerk zu garantieren. Dieses komplexe Verfahren verhindert dass ein Angreifer mit der Übernahme eines Knotens mehr als die paarweisen Verbindungen dieses Knotens kompromittieren kann.
Das *Location-based pairwise Key Establishment* verteilt ähnlich, wie das *q-Composite Random Key Predistribution Scheme*, vorweg Schlüssel auf die einzelnen Knoten. Da es zusätzlich die Annahme trifft, dass die Position der einzelnen Knoten vorweg bekannt ist, ermöglicht es eine effizientere Verteilung der Schlüssel. Diese Effizient führt zu einer bedeutenden Verringerung des Speicherbedarfs.
Andere Schlüsselverwaltungsprotokolle basieren auf dem Konzept von Blundo et al., vorgestellt in [3], welches symmetrische Polynome ($P(x, y) = P(y, x)$) vom Grad λ einsetzt: Jeder Knoten i speichert ein Polynom mit $\lambda + 1$ Koeffizienten, welches einem symmetrischen Polynom $P(x, y)$ an der Stelle $P(i, y)$ entspricht. Ohne zu tief in die Zahlentheorie

abzusteigen (für die genaue Funktionsweise siehe [3]) sei gesagt, dass jeder Knoten i nun mit seinem Polynom $f_i(y)$, welches dem Polynom $P(x, y)$ an der Stelle (i, y) entspricht $(f_i(y) = P(i, y))$, einen gemeinsamen Schlüssel mit einem beliebigen Knoten j berechnen kann. Zur Berechnung des gemeinsamen Schlüssels wertet i sein Polynom $f_i(y)$ an der Stelle j aus. Dementsprechend berechnet j den gemeinsamen Schlüssel aus $f_j(i)$. Somit erzeugen beide Knoten den gemeinsamen Schlüssel $K_{ij} = f_i(j) = P(i, j) = f_j(i)$ [4]. Solange ein Angreifer nicht mehr als λ Knoten kompromittiert, und somit nicht mehr als λ Polynome verschiedener Knoten erlangt hat, sind alle erzeugten Schlüssel sicher. Hat ein Angreifer jedoch $\lambda + 1$ Knoten kompromittiert, ist er in der Lage aus diesen, alle Koeffizienten des ursprünglichen Polynoms $P(x, y)$ zu berechnen und somit jeden beliebigen Schlüssel im Sensornetzwerk zu bestimmen.
Auch diese Verfahren ziehen mehr Verwaltungsaufwand als Dutertre et al.'s Schema auf sich, jedoch können sie als wesentlich sicherer eingeschätzt werden, da ein Angreifer im Zweifelsfall *sehr viele* Sensorknoten in seine Gewalt bringen muss.

5. IMPLEMENTIERUNG

Dutertre et al. stellen in ihrem Paper eine Implementierung ihres Protokolls für das TinyOS Operating System [12] vor, welche neben der Protokolllogik hauptsächlich einen zusätzlichen Netzwerkstack in das TinyOS integriert. Ein neuer Netzwerkstack ist notwendig, da weder der Standard-Stack noch die sicherheitsspezifische Erweiterung des alten Stacks, TinySec [11], die nötige Funktionalität für das *Bootstrapping* und *Key-Refresh* bietet: Der ursprüngliche Stack lässt jegliche Sicherheitsfeatures vermissen, aber auch TinySec bietet nur Verschlüsselung und Signierung auf Basis *eines* Schlüssels an, nicht jedoch die Möglichkeit, verschiedene Schlüssel für verschiedene Empfänger zu benutzen. Des Weiteren wird das Signieren von unverschlüsselten Nachrichten, wie beim Bootstrapping angewandt, nicht von TinySec unterstützt.
Dutertre et al.'s Netzwerkstack verwendet viele Komponenten beider Stacks, erweitert diese aber auch durch ein *raw Messages*-Format, in welchem der Stack die Nachrichten ausschließlich versendet, Verschlüsselung, Signierung und jegliche Überprüfungen aber einem Programm überlässt. Auch beim Empfang werden *raw Messages* ohne Verarbeitung direkt wieder an die zugehörige Applikation hochgegeben. Eine andere Erweiterung ist die flexible Schlüsselwahl für das Ver- und Entschlüsseln ausgehender bzw. eingehender Nachrichten.
Das Bootstrapping implementieren Sie durch einen *SecureLinkManager*, der mit raw Messages das Bootstrappingprotokoll realisiert und eine Tabelle von authentifizierten Nachbarknoten, mit paarweisen Schlüsseln, aufbaut. Für die Verschlüsselung wird der Advanced Encryption Standard verwendet (für detaillierte Information siehe [8, Kap. 6]).
Auch der Mechanismus Key-Refresh wurde bereits als Prototyp realisiert. Zur Verbreitung von Key-Refresh-Nachrichten greift der Prototyp auf die Tabelle des SecureLinkManagers zu und versendet die Nachrichten im *raw Message* Format.

6. ZUSAMMENFASSUNG

Das Verfahren des *Lightweight Key Management in Wireless Sensor Networks by Leveraging Initial Trust* bietet für den Fall, dass weder die Bootstrapping Schlüssel bk_1 und bk_2, noch die Generationenschlüssel gk_i in die Hand eines Angreifers gelangen, gute Sicherheit, bei minimalem Verwaltungsaufwand. Auch die Implementierung für das TinyOS zeigt, dass die Autoren ihre Ziele erreicht haben, und ihre Schlüsselverwaltung mit limitierten Ressourcen einsatzfähig ist. Doch es sollte darauf hingewiesen werden, dass Protokolle, wie das *Random pairwise Key Scheme* deutlich stärkere Sicherheit mit höherem, aber automatisierbarem Verwaltungsaufwand besitzen und daher Dutertre et al.'s Arbeit vorgezogen werden sollten – sofern diese mit ihrem deutlich höheren Speicheraufwand auf den zugrunde liegenden Sensoren realisierbar sind.

7. LITERATUR

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.

[2] M. Bellare and P. Rogaway. Springer-verlag. this is the full version. entity authentication and key distribution, 1993.

[3] C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung. Perfectly-secure key distribution for dynamic conferences. In *CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 471–486, London, UK, 1993. Springer-Verlag.

[4] S. A. Camtepe and B. Yener. Key distribution mechanisms for wireless sensor networks: a survey. Technical report, 2005.

[5] A. P. R. da Silva, M. H. T. Martins, B. P. S. Rocha, A. A. F. Loureiro, L. B. Ruiz, and H. C. Wong. Decentralized intrusion detection in wireless sensor networks. In *Q2SWinet '05: Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks*, pages 16–23, New York, NY, USA, 2005. ACM.

[6] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili. A pairwise key predistribution scheme for wireless sensor networks. *ACM Trans. Inf. Syst. Secur.*, 8(2):228–258, 2005.

[7] B. Dutertre, S. Cheung, and J. Levy. Lightweight key management in wireless sensor networks by leveraging initial trust, sdl. Technical report.

[8] C. Eckert. *IT-Sicherheit : Konzepte - Verfahren - Protokolle*. Oldenbourg, 2001.

[9] J. M. Ferro, L. M. Borges, O. J. Velez, and A. S. Lebres. Applications of wireless sensor networks.

[10] D. Huang, M. Mehta, D. Medhi, and L. Harn. Location-aware key management scheme for wireless sensor networks. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 29–42, New York, NY, USA, 2004. ACM.

[11] C. Karlof, N. Sastry, and D. Wagner. Tinysec: A link layer security architecture for wireless sensor networks.

[12] P. Levis, S. Madden, J. Polastre, R. Szewczyk, 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.

Mechanismen der zufallsbedingten Schlüsselvorverteilung in Sensornetzwerken

Nadine Rieß
Betreuerin: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: riess@in.tum.de

KURZFASSUNG

Diese Arbeit befasst sich mit der zufallsbedingten Vorverteilung von Schlüsseln in Sensornetzwerken. Aufgrund vielseitiger Einschränkungen bei Sensoren, wie zum Beispiel begrenzte Speicherressourcen oder geringe Rechenleistung, ist es nicht möglich, bewährte kryptographische Verfahren einzusetzen. Aus diesem Grund werden hier drei neue Mechanismen vorgestellt und hinsichtlich der Anforderungen an die Sensorknoten und die Sicherheit des Netzwerkes untersucht. Als Erstes wird das q-Verbundschlüssel-Schema näher betrachtet, womit klein angelegte Angriffe mit verhältnismäßig wenigen Knotenübernahmen effektiv reduziert werden können. Daraufhin folgt das Multipfad-Verstärkung-Schema, in dem die Sicherheit einer Kommunikationsverbindung zwischen zwei Knoten bei deren Etablierung erhöht wird. Zum Schluss wird auf das Zufalls-Schlüsselpaar-Schema eingegangen, welches bei kompromittierten Knoten die Sicherheit des restlichen Netzwerkes gewährleistet.

Schlüsselworte

Zufallsbedingte Schlüsselvorverteilung, Sensornetzwerk

1. EINLEITUNG

Sensornetzwerke werden heutzutage in vielen Bereichen eingesetzt, um verschiedenste Umgebungszustände zu erfassen oder zu überwachen. Dabei kann ein Netzwerk aus vielen tausend Sensorknoten bestehen, welche zur Ortung von Großflächenbränden, der Überwachung des Verkehrs in Realzeit, der Nachverfolgung von wild lebenden Tieren oder der Messung der Umweltverschmutzung dienen [1]. Auch der Einsatz im medizinischen Bereich zur Überwachung von lebensbedrohlichen Eigenschaften ist realisierbar. Sensoren erfreuen sich großer Beliebtheit aufgrund ihrer geringen Größe, der niedrigen Kosten und den weitreichenden Einsatzmöglichkeiten. Doch gerade bei sicherheitskritischen Anwendungen, wie Alarm- und Einbrucherkennungssystemen, ist es von enormer Wichtigkeit, dass die Sensoren zuverlässig arbeiten und nicht von außen manipuliert oder sogar außer Betrieb gesetzt werden.

Die Aufgabe besteht nun darin, die Sensoren vor dem Aufbau der Netzwerkverbindungen mit Kommunikationsschlüsseln auszustatten, so dass diese dann aufgrund verschlüsselter Kommunikationswege ein Netzwerk, mit gesicherten Verbindungen zwischen den Knoten, bilden. Es muss allerdings auch dafür gesorgt werden, dass Sensoren, die dem Netzwerk später beitreten möchten, die Möglichkeit haben, eine gesicherte Verbindung zu diesem aufzubauen. Dieser Aufbau des Netzwerkes inklusive der Aufnahmemöglichkeit später beitretender Sensoren wird als Bootstrapping-Problem bezeichnet [1]. Asymmetrische Kryptographieverfahren können aufgrund der begrenzten Hardwareausstattung der Sensoren, wie u. a. der geringen Speicherressourcen, nicht eingesetzt werden. Diese Einschränkungen und wichtige Sicherheitsaspekte, auf die im Hinblick auf sichere Kommunikation zu achten sind, werden in Kapitel 2 näher erläutert.

Das *Basisschema der zufallsbedingten Schlüsselvorverteilung* (basic random key predistribution scheme) von Eschenauer und Gligor [2] dient als Grundlage für die in dieser Arbeit diskutierten Mechanismen von H. Chan, A. Perrig und D. Song. Es geht davon aus, dass sich jeder Sensorknoten vor der Initialisierung des Netzwerkes m Schlüssel aus einer Menge von vorgegebenen Schlüsseln zufällig auswählt und diese in seinem Schlüsselring speichert. Mit Hilfe dieser Schlüssel können dann jeweils zwei benachbarte Knoten eine sichere Verbindung aufbauen, sofern sie einen gleichen Schlüssel aufgrund der vorherigen Zufallsverteilung besitzen und ihre Kommunikationsreichweiten ausreichen.
Das *q-Verbundschlüssel-Schema* (q-composite random key predistribution scheme) baut auf diesem Mechanismus auf und stellt die zusätzliche Anforderung, dass eine Kommunikation nur mit q anstatt mit einem einzigen gemeinsamen Schlüssel etabliert werden kann.
Das *Multipfad-Verstärkung-Schema* (multipath key reinforcement scheme) verbessert die Sicherheit beim Aufbau der gemeinsamen Verbindung zweier Knoten, da in diesem Fall mehrere disjunkte Teile eines Schlüssels über mehrere diskunkten Pfade verschickt und beim Endknoten wieder zum Schlüssel zusammengesetzt werden. So muss ein Angreifer mehr Knoten kompromittieren als bisher, damit eine Kommunikation zweier Knoten dechiffriert werden kann.
Zum Schluss wird auf das *Zufalls-Schlüsselpaar-Schema* (random-pairwise keys scheme) eingegangen, welches für jeweils zwei Knoten einen gemeinsamen Schlüssel vorsieht, den kein anderes Knotenpaar verwendet. Auch die gegenseitige Authentifizierung von Sensorknoten und die Implementierung einer Sperrliste der kompromittierten Knoten werden durch diesen Mechanismus ermöglicht.

Die Arbeit ist wie folgt aufgebaut: In Kapitel 2 wird zunächst auf die Restriktionen von Sensorknoten eingegangen und Kriterien zur Bewertung der Sicherheit vorgestellt. Danach wird eine Einführung in das Basisschema der zufallsbe-

dingten Schlüsselvorverteilung von Eschenauer und Gligor [2] in Kapitel 3 gegeben. Darauf aufbauend folgen das q-Verbundschlüssel-Schema in Kapitel 4, das Multipfad-Verstärkung-Schema in Kapitel 5 sowie das Zufalls-Schlüsselpaar-Schema in Kapitel 6, jeweils mit entsprechender Sicherheitsanalyse. In Kapitel 7 werden die einzelnen Mechanismen miteinander verglichen. Abschließend gibt es in Kapitel 8 eine Zusammenfassung der Ergebnisse.

2. RESTRIKTIONEN UND SICHERHEIT

Sensornetze unterliegen einigen Restriktionen. Die Sensorknoten sind hinsichtlich ihrer Rechenleistung und Speicherressourcen sehr beschränkt. Auch die Bandweite und Übertragungsleistung ist stark begrenzt, wodurch die Verlässlichkeit einer Datenübertragung gemindert wird. Bekannte asymmetrische Verschlüsselungsverfahren wie RSA [4] oder Schlüsselaustauschverfahren wie Diffie-Hellman [5] würden eine zu lange Berechnungszeit benötigen und sich wie eine Denial of Service (DOS) Attacke auswirken. Hinzu kommt, dass die Sensoren auch an öffentlichen oder sogar in feindlichen Gebieten platziert werden und sie einem Angreifer somit auch physikalisch ausgesetzt sind. Ein Sensorknoten hat auch vor der Initialisierung des Netzwerkes keinerlei Hinweise darauf, welcher und wieviele Knoten sich in seiner Nachbarschaft befinden werden. Viele Sensornetzwerke werden mit Basisstationen versehen, die als Vertrauensquelle gelten und zentrale Aufgaben übernehmen. Solche Basisstationen ziehen somit verstärkt das Interesse der Angreifer auf sich. Die nachfolgenden Mechanismen sollen ohne Basisstationen auskommen, um die Sicherheit zu erhöhen. All diese Punkte müssen bei der Entwicklung eines Mechanismus für die Schlüsselvorverteilung in Sensornetzwerken beachtet werden.

Da Sensorknoten hinsichtlich ihrer Sicherheit stark gefährdet sind, werden in dieser Arbeit besonders die folgenden Aspekte untersucht.

- Benötigte Speicherressourcen
- Widerstandsfähigkeit des Netzwerkes gegen kompromittierte Knoten
- Replikation von Sensorknoten
- Maximal unterstützbare Netzwerkgröße

Aufgrund des geringen Speicherplatzes der Sensoren muss darauf geachtet werden, die Datenmengen so gering wie möglich zu halten. Es können also nicht beliebig viele Daten abgespeichert werden, um damit bekannte Sicherheitsmechanismen zu realisieren. Wurden ein oder mehrere Knoten kompromittiert, dann soll ein Angreifer von den erhaltenen Daten keine bedeutenden Informationen über das Sensornetz
schließen können und der Rest des Netzwerkes widerstandsfähig gegen die feindliche Übernahme dieser Knoten bleiben. Des Weiteren muss darauf geachtet werden, dass das Sensornetz eine Resistenz gegenüber Duplikaten von Knoten aufweist. Es soll nicht möglich sein, Kopien der Sensorknoten anzufertigen, um diese für Attacken zu verwenden. Dazu können Sensoren Sperrlisten verwalten, welche es ermöglichen, kompromittierte Knoten abzuspeichern, um damit jede weitere Kommunikation mit ihnen zu unterbinden. Da unsichere Knoten gleichzeitig auch einen Teil des Netzwerkes unsicher machen, ist es interessant, die maximale Größe des unterstützbaren Netzwerkes zu kennen, bis zu der effektiver Schutz gewährleistet werden kann.

3. BASISSCHEMA DER ZUFALLSBEDINGTEN SCHLÜSSELVORVERTEILUNG

Die in dieser Arbeit beschriebenen Mechanismen bauen auf dem Basisschema der zufallsbedingten Schlüsselvorverteilung auf, welches nun kurz erläutert wird. Entwickelt wurde es von Eschenauer und Gligor [2] und lässt sich in drei verschiedene Phasen einteilen:

1. Initialisierungsphase
2. Konfigurationsphase
3. Pfadschlüssel-Aushandlungsphase

Zu Beginn der Initialisierungsphase wird ein *Schlüssel-Set S* von zufällig gewählten Schlüsseln vom gesamten möglichen Schlüsselraum extrahiert. Für jeden Sensorknoten werden nun wiederum m verschiedene Schlüssel vom Set S nach dem Zufallsprinzip ausgewählt und diese in dessen eigenen Schlüsselring abgelegt, welches in Abbildung 1 dargestellt ist. Die Anzahl der Schlüssel $|S|$ in der Schlüsselmenge wird dabei so gewählt, dass zwei Sensorknoten mit jeweils m gespeicherten Schlüsseln mindestens einen Schlüssel mit der Wahrscheinlichkeit p gemeinsam haben.

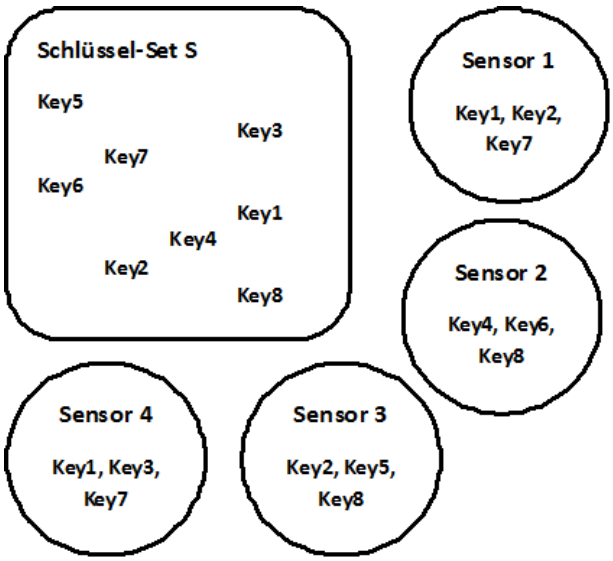


Abbildung 1: Schlüssel-Set S (Schlüsselringgröße m=3)

Ist die Initialisierung abgeschlossen, so werden in der darauf folgenden Konfigurationsphase die Sensorknoten aktiv und es erfolgt die Suche nach gemeinsamen Schlüsseln mit Nachbarknoten. Dazu wurde jedem Schlüssel in S ein kurzer *Identifikator* zugewiesen. Alle Sensorknoten senden nun einen Broadcast mit sämtlichen Identifikatoren. Falls ein Nachbarknoten einen gemeinsamen Identifikator, und damit einen

gemeinsamen Key, in seinem Schlüsselring entdeckt, so kann er diese Gemeinsamkeit durch ein *Challenge-Response-Protokoll* mit dem entsprechenden Nachbarknoten verifizieren. Ist dieser Schritt erfolgreich durchlaufen, so ist fortan der Schlüssel das gemeinsame Geheimnis zwischen den beiden Sensorknoten und sie können somit verschlüsselt kommunizieren. In der Abbildung 2 können beispielsweise Sensorknoten S3 und Sensorknoten S1 durch den Schlüssel K2 geheime Nachrichten austauschen.

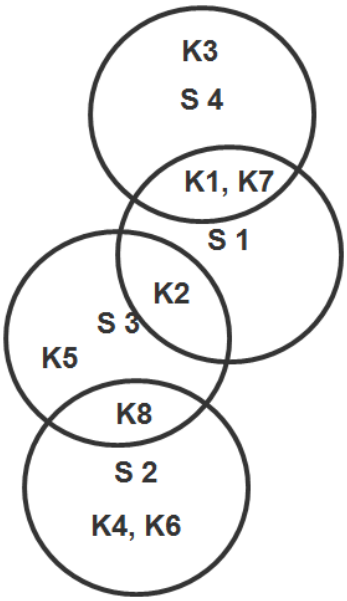


Abbildung 2: Kommunikationsverbindungen der Knoten

Nach der Konfigurationsphase ist ein Netzwerk aus miteinander verbundenen Knoten entstanden, welches als zusammenhängender Graph interpretiert werden kann. In der Aushandlungsphase ist es nun möglich, dass zwei Knoten, die bisher keinen gemeinsamen Schlüssel hatten, über einen entstandenen Pfad einen Schlüssel aushandeln. Falls der Graph also zusammenhängend ist, so kann ein Pfad vom Ausgangsknoten zu einem Nachbarknoten in Kommunikationsreichweite gefunden werden. Der Ausgangsknoten generiert dafür einen Pfadschlüssel und sendet ihn sicher über den Pfad an den Zielknoten [1]. Dadurch kann jeder Knoten zu allen Nachbarknoten direkte Verbindungen aufbauen.

4. Q-VERBUNDSCHLÜSSEL-SCHEMA

Das q-Verbundschlüssel-Schema ist dem Basisverfahren von Eschenauer und Gligor sehr ähnlich. Es unterscheidet sich lediglich darin, dass im q-Verbundschlüssel-Schema die Überlappung des Schlüsselringes der beiden Knoten von einem gemeinsamen Schlüssel auf q gemeinsame Schlüssel angehoben wird. Dies hat zur Folge, dass das Netzwerk widerstandsfähiger gegen Angriffe wird, da mehrere Schlüssel benötigt werden, um Verbindungen abzuhören. Somit wird es schwieriger für einen Angreifer, mit einer bereits erbeuteten Menge an Schlüsseln eine Verbindung zu belauschen. Um jedoch zu gewährleisten, dass zwei Sensorknoten sich q gemeinsame Schlüssel mit einer festgelegten Wahrscheinlichkeit p teilen, muss die Anzahl der Schlüssel in S gesenkt werden. Daraus folgt, dass auch der Angreifer weniger Knoten attackieren muss, um prozentual mehr Schlüssel aus S zu bekommen. Es ist eine Balance zwischen diesen konträren Faktoren zu finden, um die optimale Anzahl q und $|S|$ an Schlüsseln zu finden.

4.1 Funktionsweise des Algorithmus

Im Folgenden wird die genaue Vorgehensweise des Mechanismus dargestellt.

In der Initialisierungsphase wird eine Teilmenge an Schlüsseln aus dem gesamten Schlüsselraum gewählt. Von diesem werden wiederum Schlüssel extrahiert und im Schlüsselbund des Sensorknotens, welcher m Schlüssel umfasst, abgelegt.

In der Konfigurationsphase müssen nun wieder sämtliche Sensorknoten über die vorhandenen Schlüssel ihrer Nachbarknoten informiert werden. Dies könnte naiver Weise mit einem Broadcast der Schlüsselidentifikatoren an alle Knoten realisiert werden. Allerdings kann ein Angreifer die Verbindungen abhören und somit sämtliche ausgesendete Schlüsselsets eines Sensors aufzeichnen und diese Information dazu einsetzen, um gezielt Knoten anzugreifen und damit einen Großteil der Schlüssel in S in Erfahrung zu bringen. Eine sicherere, aber auch langsamere Methode stellen *Client-Puzzles* dar, wie beispielsweise das *Merkle-Puzzle* [6]. Jeder Knoten kann für jeden seiner m Schlüssel ein Client-Puzzle ausgeben. Der Sensor, der sich mit der richtigen Antwort rückmeldet, beweist, dass er den passenden Schlüssel besitzt.

Nachdem nun jeder Sensorknoten seine Nachbarknoten und deren Schlüsselmengen kennt, kann eine Verbindung initialisiert werden. Nur wenn die Anzahl der gemeinsamen Schlüssel zweier Knoten $q' \geq q$ ist, wird ein neuer Kommunikationsschlüssel K aus dem Hash aller gemeinsamen Schlüssel k_i generiert: $K = hash(k_1||k_2||\ldots||k_{q'})$. Gehasht wird nach einer kanonischen Ordnung, zum Beispiel darauf basierend, in welcher Ordnung die Schlüssel in der Schlüsselmenge S vorkommen [1].

Möchte nun ein neuer Sensorknoten dem Netzwerk beitreten, so durchläuft auch dieser die drei Phasen der Initialisierung, Konfiguration und Pfadschlüssel-Aushandlung. Zunächst wählt er zufällig m Schlüssel aus dem gesamten Schlüsselraum. Danach sendet er die entsprechenden Client-Puzzles und erhält von den Knoten mit dem passenden Schlüssel eine Antwort. Zum Schluss können wiederum Pfadschlüssel ausgehandelt werden.

4.2 Sicherheit

Das in diesem Abschnitt vorgestellte Schema ist nicht resistent gegen eine Replikation von Knoten, da die Anzahl der Verbindungen nicht beschränkt ist und es kein Limit gibt, wie oft ein Schlüssel genutzt werden kann. Das Schema kann allerdings eine *Sperrliste* von kompromittierten Knoten unterstützen, falls eine vertrauenswürdige Basisstation eingesetzt wird [1].

Die Widerstandsfähigkeit des Netzwerks gegen kompromittierte Knoten wird anhand der Gefahr, die von den erhaltenen Schlüsseln und den damit verbundenen Informationen des Knotens ausgeht, evaluiert. Man möchte also herausfinden, wie wahrscheinlich es ist, dass eine Verbindung zweier nicht kompromittierter Knoten entschlüsselt werden kann, falls ein Angreifer Sensorknoten des Netzes kompromittiert und versucht, aus den erhaltenen Daten die Schlüs-

selmenge S, und damit gleichzeitig das Set an Schlüsseln dieser Verbindung zu reproduzieren.
In Abbildung 3 wird gezeigt, dass bei einer niedrigeren Anzahl an kompromittierten Knoten die Widerstandsfähigkeit des Netzwerkes durch das q-Verbundschlüssel-Schema verbessert wird. Erhöht man q, so macht man es dem Angreifer schwerer an Teilinformationen mittels weniger Knotenübernahmen zu gelangen [1]. Im Gegensatz dazu weist dieser Mechanismus bei einem großen Netzwerk Schwächen auf. Hier wird es Angreifern erleichtert, an weitere Informationen zu gelangen, falls dieser bereits eine große Anzahl an Knoten kompromittiert hat. Dies ist allerdings ein wünschenswerter Kompromiss, weil Angriffe auf mengenmäßig wenige Knoten ("small-scale") viel schwieriger zu entdecken sind als groß angelegte Angriffe. Eine Attacke auf einen einzelnen Knoten kann im Gegensatz zu einer Attacke auf viele Knoten leicht als natürlicher Verbindungsabbruch getarnt werden.

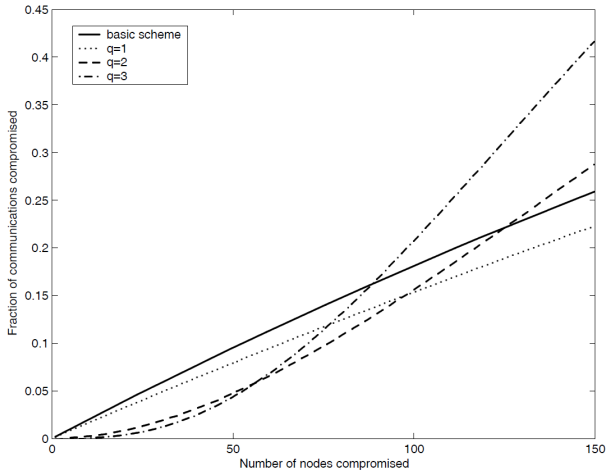


Abbildung 3: Wahrscheinlichkeit, dass eine Verbindung entschlüsselt werden kann, wenn daran nicht beteiligte Knoten kompromittiert wurden. (m=200, p=0,33) [1]

Eine feste Anzahl an kompromittierten Knoten bedeutet, dass ein bestimmter Anteil des Netzwerkes unsicher ist und die Mechanismen nicht für beliebig große Netzwerke verwendet werden können. Die maximal unterstützbare Größe eines Netzwerks ist so zu wählen, dass Angreifer bei einer festgelegten Anzahl von kompromittierten Knoten nur einen begrenzten Erfolg verbuchen können und nicht mehr über den Rest des Netzwerkes lernen, als sie über die Kommunikationschlüssel des kompromittierten Knotens selbst erfahren. Dadurch wird der Anreiz für einen Angriff gemindert, denn eine Attacke muss durch den Wert des einzelnen kompromittierten Knotens gerechtfertigt sein und nicht durch die Information, die die erhaltenen Schlüssel vom Rest des Netzwerkes preisgeben. Es existiert also ein Grenzwert der maximalen Menge an kompromittierten Informationen, ab der das Netzwerk als nicht mehr sicher gilt. Abbildung 4 stellt diesen Sachverhalt dar und berechnet den Grenzwert wie folgt. Sei x_m die Anzahl der kompromittierten Knoten. f_m ist der durch die x_m direkt kompromittierten Knoten und Verbindungen zusätzlich entstandene unsichere Teil des Netzwerkes. Der durchschnittliche Grad d eines Knotens bezeichnet die Anzahl der Verbindungen zu anderen Knoten. Der Angreifer hat somit $x_m d$ erwartete Verbindungen, in welche die kompromittierten Knoten verwickelt sind. Da es insgesamt $\frac{nd}{2}$ Verbindungen im Netzwerk gibt, ist die Forderung $(\frac{nd}{2} - x_m d) f_m \leq x_m d$.. Umgeformt ergibt sich die obere Schranke für die maximale Netzwerkgröße: $n \leq 2x_m \left(1 + \frac{1}{f_m}\right)$.

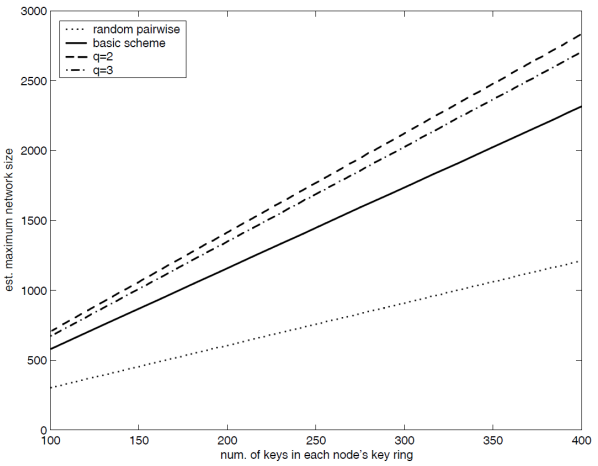


Abbildung 4: Maximale Netzwerkgröße (p=0,33, f_m=0,1) [1]

5. MULTIPFAD-VERSTÄRKUNG-SCHEMA

Dieses Kapitel behandelt den Mechanismus des Multipfad-Verstärkung-Schemas, dessen Grundidee von Anderson und Perrig stammt [3]. Das Ziel dieses Mechanismus ist es, die Sicherheit des Kommunikationsschlüssels zweier Knoten zu verbessern, indem dieser Schlüssel mittels mehrerer Pfade etabliert wird. Diese Herangehensweise führt allerdings dazu, dass der Netzwerkverkehr zunimmt.

5.1 Funktionsweise des Algorithmus

Beim Multipfad-Verstärkung-Schema findet die Initialisierungs- und Konfigurationsphase entsprechend dem Basisschema der zufallsbedingten Schlüsselvorverteilung statt und es wird angenommen, dass sichere Kommunikationsverbindungen mittels der vorhandenen Schlüssel aufgebaut wurden. Ein Schlüssel, der von zwei Sensorknoten genutzt wird, könnte damit noch in weiteren Schlüsselringen von anderen Knoten vorkommen und zu deren Kommunikation verwendet werden. Falls diese Knoten kompromittiert werden, so ist ebenfalls die Kommunikationssicherheit der beiden anderen Knoten, welche den gleichen Schlüssel nutzen, in Gefahr. Daher ist es sinnvoll, den gemeinsamen Schlüssel zweier Knoten nach der Konfigurationsphase durch einen neuen, zufällig gewählten auszutauschen, im Folgenden als *Schlüsselupdate* bezeichnet. Jedoch wäre es fahrlässig, den neuen Schlüssel über die direkte Verbindung, mit dem alten Schlüssel chiffriert, zu senden. Ist der alte Schlüssel bekannt, so kann der neue damit ebenfalls entschlüsselt werden.
Deshalb nutzt das Schlüsselupdate mehrere disjunkte Pfade von einem Knoten A zu einem Zielknoten B. Es werden nach der Konfigurationsphase genug Informationen bzgl. der bestehenden Routen im Netzwerk gesammelt. Somit sind alle

disjunkten Pfade von A zu B bekannt, die aus einer maximalen Anzahl an Hops bestehen. Die Abbildung 5 erklärt diesen Sachverhalt. Die Verbindung zwischen A und dem Sensorknoten S6 darf wegen der Disjunktheit lediglich einmal genutzt werden und der zweite Weg, markiert durch X, ist somit nicht zulässig. Angenommen, es wurden j disjunkte Pfade vom Sensorknoten A zu B während der Konfigurationsphase gefunden. Knoten A erzeugt nun j zufällige Werte, die der Länge eines gewöhnlichen Schlüssels entsprechen und sendet über jeden der j Pfade einen dieser Werte. B kann sich, sofern es alle j Werte erhalten hat, genauso wie A, aus diesen einen neuen Schlüssel $K = (k \otimes v_1 \otimes v_2 \otimes \ldots \otimes v_j)$ generieren, welcher von nun an als Kommunikationsschlüssel zwischen A und B genutzt wird [1].

Wenn lediglich Pfade über 2 Verbindungen untersucht werden, genannt *2-Hop-Multipfad-Verstärkung-Schema*, dann wird der Aufwand, alle Pfade zu finden, wesentlich reduziert. Zwei Knoten A und B, welche einen geheimen Schlüssel vereinbaren wollen, müssen lediglich eine Liste ihrer gemeinsamen Nachbarn erstellen. Desweiteren sind alle diese Pfade disjunkt, weshalb kein weiterer Aufwand betrieben werden muss, um die Disjunktheit zu garantieren.

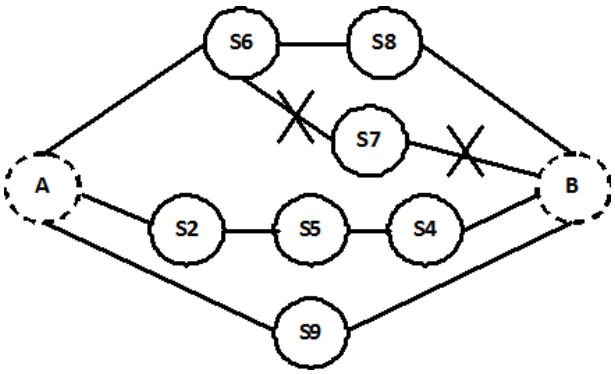


Abbildung 5: Disjunkte Pfade, max. Hoplänge=4

Auch in diesem Schema durchläuft ein neu hinzukommender Sensor die drei Phasen. Nach der zufälligen Auswahl der Schlüssel und dem Broadcast findet das Schlüsselupdate wie oben beschrieben statt.

5.2 Sicherheit

Die Sicherheit des neuen Kommunikationsschlüssels ist durch die j zufälligen Werte gegeben. Der Angreifer muss also sämtliche j Pfade abhören, um alle Werte zu bekommen und somit den neuen Schlüssel rekonstruieren zu können. Daraus folgt, je mehr Pfade zwischen zwei Knoten existieren, desto sicherer ist der neue Schlüssel. Allerdings steigt mit der Länge der Pfade die Wahrscheinlichkeit, dass ein Angreifer einen Knoten dieses Pfades kompromittiert und der Datenfluss mitgelesen wird. Der Pfad ist also unsicher, sobald ein Knoten dieses Pfades unsicher ist. Desweiteren benötigen lange Pfade einen enorm großen Kommunikationsaufwand aufgrund der Überprüfung auf Disjunktheit.

In Abbildung 6 kann man das Verhältnis von der Anzahl der kompromittierten Knoten zum Anteil der unsicheren Verbindungen ablesen. Das Basisschema und das q-Verbundschlüssel-Schema werden durch die Kombination mit dem 2-Hop-Multipfad-Verstärkung-Schema jeweils verbessert, wobei das Basisschema besser als das q-Verbundschlüssel-Schema für $q \geq 2$ ist. Dies liegt daran, dass das q-Verbundschlüssel-Schema nach dem gleichen Prinzip arbeitet wie das Multipfad-Schema, da es mehrere Schlüssel bzw. Pfade benötigt, um den neuen Schlüssel zu berechnen. Die Kompromisse die man beim Verbundschlüssel-Schema ($|S|$ wurde minimiert) und beim Multipfad (erhöhter Kommunikationsaufwand) eingegangen ist, wirken nun gleichzeitig gegen die Vorteile des Multipfad-Schemas und minimieren diese.

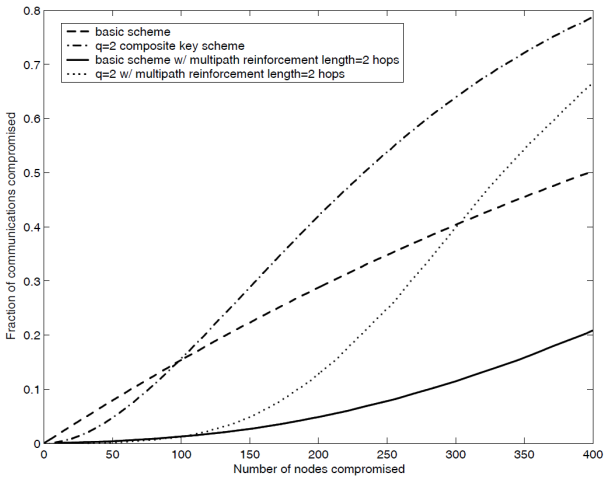


Abbildung 6: Resistenz gegen kompromittierte Knoten (m=200, p=0,33) [1]

In Abbildung 7 lässt sich ablesen, welche Anzahl an gespeicherten Schlüsseln im Knoten welche maximale Netzwerkgröße zulässt. Auch hier werden beide Mechanismen mit dem 2-Hop-Verfahren kombiniert und verglichen. Das Multipfad-Verfahren, angewandt auf das Basis-Verfahren, ermöglicht eine wesentlich größere Netzwerkgesamtgröße, hat aber beim q-Verbundschlüssel-Schema nur einen geringen Effekt.

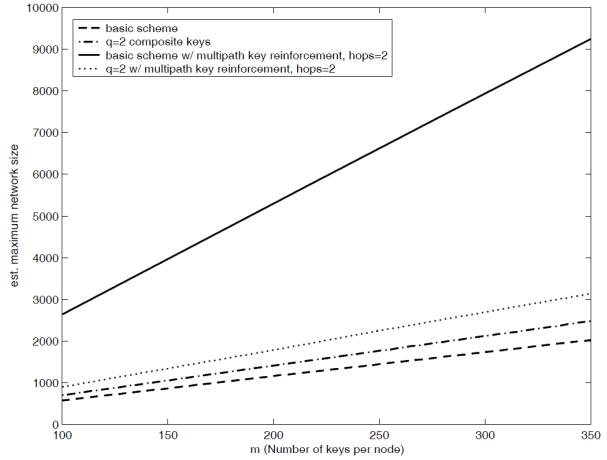


Abbildung 7: Maximale Netzwerkgröße (p=0,33) [1]

Der zusätzlichen Sicherheit durch den Multipfad-Mechanismus steht ein größerer Kommunikationsaufwand aufgrund der Suche nach disjunkten Pfaden und der Etablierung des neuen Schlüssels gegenüber. Ob dies ein guter Kompromiss ist, hängt von der spezifischen Anwendung und der Dichte des Sensornetzwerks selbst ab [1]. Mit dem Multipfad-Verfahren können aber auch Schlüssel für Knoten erzeugt werden, die nach der Konfigurationsphase keine Schlüssel miteinander teilen.

6. ZUFALLS-SCHLÜSSELPAAR-SCHEMA

In den bereits behandelten Mechanismen kann jeder Knoten bestätigen, dass sein Nachbarknoten bestimmte geheime Schlüssel besitzt und sich damit für eine Kommunikation qualifiziert. Allerdings ist kein Sensorknoten in der Lage den jeweils anderen zu authentifizieren, ihm ist also die Identität seines Kommunikationspartners nicht bekannt.
Beispielsweise teilt A mit B eine Menge K von geheimen Schlüsseln, womit sie ihre Kommunikation verschlüsseln. Da die Schlüssel auch mehrmals von S extrahiert und zugeteilt werden können, ist es möglich, dass ein Knoten C diese Menge K ebenfalls in seinem Schlüsselbund verwaltet. Knoten A kann also nicht sicher gehen, ob er mit B oder C kommuniziert. Die gegenseitige Verifikation von Knoten wird als *Knoten-zu-Knoten-Authentifizierung* (Node-to-Node-Authentification) bezeichnet und unterstützt viele Sicherheitsfunktionen.

6.1 Funktionsweise des Algorithmus

Angenommen ein Netzwerk besteht aus n Sensorknoten. Eine einfache Lösung des Schlüsselpaar-Schemas ist es, dass jeder Knoten *n-1* Kommunikationsschlüssel speichert, welche er mit jeweils einem anderen Knoten des Netzwerks teilt. Das Zufalls-Schlüsselpaar-Schema ist eine Modifikation davon, wobei aber nicht alle *n-1* Keys im Schlüsselbund abgespei- chert werden müssen, um einen zusammenhängenden Graph mit einer hohen Wahrscheinlichkeit zu erhalten. Wie oben bereits erwähnt wird in einem Sensornetzwerk mit der Wahrscheinlichkeit p eine Verbindung zwischen zwei Knoten aufgebaut. Somit muss ein Knoten nur np Schlüssel anstatt *n-1* Schlüssel in seinem Schlüsselring der Größe m abspei- chern. Dies ergibt die Formel $m = np$, was sich zu $n = \frac{m}{p}$ umformen lässt.
Die Nutzung von paarweise vergebenen Schlüsseln anstatt von zufällig gewählten Schlüsseln aus einer Menge S erlaubt Knoten-zu-Knoten Authentifizierung, falls jeder Sensor zum Schlüssel k die *Identität* (ID) des anderen Knotens abspeichert. Somit weiß jeder Sensorknoten anhand des genutzten Kommunikationsschlüssel mit welchem Knoten er kommuniziert.

In der Initialisierungsphase des Zufalls-Schlüsselpaar-Schemas werden zunächst n einzigartige Identitäten generiert. Falls das Netzwerk aktuell weniger als n Knoten besitzt, so können die verbleibenden Identitäten für später hinzukommende Knoten verwendet werden. Jede Knotenidentität wird nun mit m anderen IDs verbunden und die Schlüsselpaare für jeweils zwei Knoten werden generiert. Ein Schlüssel wird dann in beiden Knoten zusammen mit der ID des jeweils anderen Knotens gespeichert.
In der Konfigurationsphase findet nun ein Broadcast der eigenen Knoten-ID statt. Findet ein anderer Knoten diese ID in seinem eigenen Schlüsselbund, so initiiert er einen *kryptographischen Handshake* zwischen ihm und dem anderen Knoten, wodurch eine gegenseitige Authentifizierung stattfindet.

6.2 Erweiterung des Algorithmus

Da sich die Sensorknoten im Zufalls-Schlüsselpaar-Schema gegenseitig authentifizieren, kann der Mechanismus durch *Sperrlisten* für Knoten (node revocation list) erweitert werden. In diesen Sperrlisten werden Sensorknoten erfasst, die sich nicht konform verhalten und somit der Verdacht auf Manipulation naheliegt. Ist ein Knoten einmal notiert, so werden die anderen Knoten jegliche Kommunikationsanfragen von diesem zurückweisen. Normalerweise geschieht dies anhand von Basisstationen, welche die Listen abspeichern, auf Anfrage der Knoten ändern oder Informationen weitergeben. Aufgrund der hohen Latenzzeit zwischen den Knoten und der Basisstation ist dieses Verfahren extrem langsam, manipulierte Knoten müssen allerdings schnellstmöglich vom Netzwerk verbannt werden. Um diesem Nachteil zu entgehen, werden die Sperrlisten auf die Knoten verteilt.
Die Funktionsweise ist einer Wahl sehr ähnlich. Sensorknoten erhalten dabei eine Wahlstimme gegen einen Nachbarknoten, wenn sie mit ihm Kontakt aufgenommen haben. Erkennt nun ein Sensor eine verdächtige Handlung eines Nachbarn X, so wird der Sensor von seinem Stimmrecht Gebrauch machen und mittels Broadcast eine öffentliche Stimmabgabe gegen X versenden. Falls nun die Anzahl der Stimmen gegen X einen bestimmten Grenzwert übersteigt, so ist dieser Knoten als unsicher einzustufen und in die eigene Sperrliste aufzunehmen. Eine detailiertere Beschreibung kann in [1] nachgelesen werden.

6.3 Sicherheit

In diesem Verfahren authentifizieren sich die Sensorknoten gegenseitig und bestätigen somit ihre Identitäten. Es besteht die Möglichkeit, Knoten zu entlarven, die sich nicht konform verhalten und diese in eine Sperrliste aufzunehmen. Nicht konformes Verhalten können beispielsweise außergewöhnlich viele Broadcasts sein, die auf einen Denial of Service Angriff schließen lassen. Anstatt eine solche Sperrliste bei einer Basisstation zu verwalten, können die Knoten diese Aufgabe selbst übernehmen, was eine wesentliche Verbesserung der Reaktionszeit mit sich bringt. Somit lässt sich eine Kommunikation mit kompromittierten Knoten noch effektiver verhindern.
Des Weiteren können Kopien von Sensorknoten enttarnt werden, falls alle bereits initialisierten Knoten des Netzwerkes verwaltet werden und somit eine nochmalige Initialisierung der selben Identität auffallen würde. Andere Knoten werden dann eine Kontaktaufnahme zu diesem kopierten Sensor zurückweisen. Ein Knoten kann sich also nicht als ein anderer Knoten ausgeben und somit ist eine Replikation von Sensoren nicht möglich.
Die Widerstandsfähigkeit gegen kompromittierte Knoten ist optimal, denn durch die paarweise verteilten einzigartigen Keys kann ein Angreifer jeweils nur die Verbindungen entschlüsseln, in die der kompromittierte Knoten selbst verwickelt ist und bekommt keine weiteren Informationen über das Netzwerk.
Die maximal unterstützbare Größe des Netzwerks kann nicht nach dem Verfahren des q-Verbundschlüssel-Schema berechnet werden, da in diesem Fall ein kompromittierter Knoten keine weiteren Informationen über das Netzwerk preisgibt

[1]. Die maximale Netzwerkgröße kann also, wie in der Funktionsweise des Mechanismus bereits erwähnt, mit $n = \frac{m}{p}$ angegeben werden.

7. GEGENÜBERSTELLUNG DER ALGORITHMEN

Nun werden anhand der in Kapitel 2 genannten Bewertungskriterien die drei vorgestellten Mechanismen verglichen. Die Tabelle in Abbildung 8 zeigt hierfür das Basisschema von Eschenauer und Gligor (Basis) und das q-Verbundschlüssel-Schema für $q = 2$, jeweils mit und ohne der Anwendung des Multipfad-Verstärkung-Schemas, sowie das Zufalls-Schlüsselpaar-Schema. In der linken Spalte befinden sich die Bewertungskriterien. Die Zeichen + und - zeigen die Güte des Mechanismus im Gegensatz zu den anderen Verfahren an. Bei dem Kriterium der maximalen Netzwerkgröße wurde eine Rangfolge aufgestellt, wobei das Basisschema in Kombination mit dem Multipfad-Verstärkung-Schema deutlich größere Netzwerkkapazitäten zulässt als alle anderen Verfahren.

		Basis		2-Verbundschlüssel		Zufalls-Schlüsselpaar
		ohne Multipfad	mit Multipfad	ohne Multipfad	mit Multipfad	
Speicherbedarf		m Schlüssel m Identifikatoren		m Schlüssel m Identifikatoren		eigene KnotenID np Schlüssel np KnotenID's
Widerstands-fähigkeit bei Knotenübernahme (Anzahl Knoten)	wenige	--	++	-	++	+++
	viele	-	++	--	+	+++
Resistenz gegen Knoten-Replikation		Nein		Nein		Ja
Sperrliste		Ja (Basisstation)		Ja (Basisstation)		Ja
Max. Netzwerkgröße		4	1 ++	3	2	5

Abbildung 8: Vergleich aller Algorithmen

Aus der Tabelle ist ersichtlich, dass das Multipfad-Schema sowohl beim Basisschema als auch beim 2-Verbundschlüssel-Schema eine wesentliche Verbesserung der Wiederstandsfähigkeit bei Knotenübernahme erwirkt. Dennoch kann das Zufallsschlüsselpaar-Schema diesen Erfolg übertreffen. Letzteres bietet auch einen Schutz gegen Knoten-Replikation, da sich jeder Sensor gegenüber den anderen Sensoren authentifizieren muss. Eine Kopie eines Knotens, welche die Identität eines anderen für sich nutzt, kann somit identifiziert werden. Bei dem Kriterium der maximalen Netzwerkgröße wurde eine Rangfolge aufgestellt, wobei das Basis-Verfahren in Kombination mit dem Multipfad-Verfahren deutlich größere Netzwerkkapazitäten zulässt als alle anderen Verfahren. Das Zufalls-Schlüsselpaar-Schema ist nach [1] hinter dem Basisschema ohne Multipfad-Verstärkung-Schema einzuordnen.

8. ZUSAMMENFASSUNG

Sicherheit spielt in Sensornetzwerken eine große Rolle. Dabei wird nicht nur auf eine verschlüsselte Kommunikation zwischen den Sensoren Wert gelegt, sondern auch auf einen sicheren Eintritt von Sensoren, welche dem Netzwerk später beitreten. In dieser Arbeit wurden drei Mechanismen vorgestellt, die das Bootstrapping-Problem und damit verbundene Sicherheitsaspekte behandeln.
Das q-Verbundschlüssel-Schema ging dabei besonders auf den Verbindungsaufbau einer Kommunikation ein, was nur mit mehreren gemeinsamen Schlüsseln möglich ist. Dieses Schema verbesserte die Erkennung von "small-scale"-Attacken erheblich, führte allerdings auch zu einer größeren Verwundbarkeit bei großflächigen Angriffen.
Beim Multipfad-Verstärkung-Schema wurde für den Kommunikationsaufbau ein neuer Schlüssel generiert, welcher in Teilen über mehrere disjunkte Pfade zum Kommunikationspartner gesendet wird. Dies macht es für einen Angreifer schwerer, an den eigentlichen Schlüssel zu kommen, um die Kommunikation zwischen Knoten entschlüsseln zu können, jedoch ist dieses Verfahren mit einem erheblich größerem Kommunikationsaufwand im Netzwerk verbunden.
Zum Schluss ist das Zufalls-Schlüsselpaar-Schema, mit disjunkten Schlüsselpaaren für jeweils zwei Sensorknoten, untersucht worden. Dieser Mechanismus ist widerstandsfähig gegen kompromittierte Knoten, erkennt Replikationen von Knoten und ermöglicht eine Sperrliste.
Das Zufalls-Schlüsselpaar-Schema kann meiner Meinung nach aufgrund den Erkenntnissen aus Kapitel 7 als das beste Schema angesehen werden, sofern die Größe des auszubringenden Netzwerkes gering ist. Wird jedoch ein großes Netzwerk benötigt, so sollte das Basisverfahren in Kombination mit dem Multipfad-Verstärkung-Schema die erste Wahl sein.

9. LITERATUR

[1] H. Chan, A. Perrig, D. Song: *Random Key Predistribution Schemes for Sensor Networks*, IEEE Symposium on Security and Privacy, 2003.

[2] L. Eschenauer, V. D. Gligor: *A Key-Management Scheme for Distributed Sensor Networks*, Proceedings of the 9th ACM Conference on Computer and Communication Security, Seite 41-47, November 2002.

[3] R. Anderson, H. Chan, A. Perrig: *Key Infection: Smart Trust for Smart Dust*, Unpublished Manuscript, November 2001.

[4] R. L. Rivest, A. Shamir, L. Adleman: *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, 1978.

[5] W. Diffie, M. Hellman: *New directions in cryptography*, IEEE Transactions on information Theory, 1976.

[6] R. C. Merkle: *Secure communications over insecure channels*, Communications of the ACM, 1978.

Key management in wireless sensor networks focusing on predistribution by group deployment

Alexander Regler
Betreuerin: Corinna Schmitt
Seminar Innovative Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: regleral@in.tum.de

KURZFASSUNG

Die Sicherheit von Sensornetzen effizient zu gewährleisten, stellt aufgrund der limitierten Speicher- und Leistungsfähigkeit im Vergleich zu anderen Netzwerken eine ungleich komplexere Aufgabe dar. In dieser Arbeit wird ein gruppenbasiertes Verfahren präsentiert, das durch eine Vorinitialisierung der Sensoren sichere Kommunikationsverbindungen ermöglicht. Die Konnektivität, der Grad an Sicherheit und der Speicherbedarf können mittels Parameter flexibel der jeweiligen Anwendung angepasst werden.

Schlüsselworte

Sensornetzwerk, Schlüsselvorverteilung, Gruppenausbringung, Transversal Design

1. EINLEITUNG

Sensornetze erfreuen sich dank ihrer sinkenden Kosten, aber auch aufgrund technischer Fortschritte immer größerer Beliebtheit. Sie bestehen aus mehreren Sensorknoten – Geräten, welche physische Größen, wie Temperatur oder Feuchtigkeit, messen und drahtlos miteinander kommunizieren können [2]. Sensornetze werden mittlerweile in einer Vielzahl von Bereichen eingesetzt, beispielsweise zur Überwachung von Tieren, zum Lokalisieren von Lawinenverschütteten, in der Medizin zur Kontrolle der Lebenszeichen eines Patienten oder im militärischen Bereich zur Verfolgung der eigenen Militärfahrzeuge [1].

Wie in jedem anderen Netzwerk werden auch in Sensornetzwerken geheime und vertrauliche Daten ausgetauscht, die geschützt werden müssen. Allerdings besitzen Sensorknoten je nach Anwendung eine im Vergleich zu gewöhnlichen Netzwerken stark eingeschränkte Energie- und Rechenleistung sowie beschränkte Speicherfähigkeit [3]. Bewährte Verfahren aus der Public-Key-Kryptographie, wie der Diffie-Hellman-Schlüsselaustausch oder eine RSA-Signatur, stellen sehr hohe Anforderungen an die Rechenleistung und den Speicher eines Sensorknotens. Deshalb sollten diese in Sensornetzen ausschließlich bei Applikationen, deren benötigter Sicherheitsgrad diesen zusätzlichen Aufwand unbedingt erforderlich macht, eingesetzt werden.

Eine Alternative zur aufwändigen Public-Key-Kryptographie ist das Verteilen von Schlüsseln auf die Sensorknoten, bevor das Sensornetzwerk aktiv wird. Mit Hilfe der verteilten Schlüssel soll ein Sensorknoten dann eine gesicherte Verbindung zu einem anderen Sensorknoten des aktiven Netzwerkes aufbauen können. Dieses Schema wird im Folgenden als *Schlüsselvorverteilung* (key predistribution) bezeichnet [2]. Eine leicht modifizierte Variante der Schlüsselvorverteilung wird mit Blom's Schema ab Kapitel 4.1 betrachtet. An Stelle der direkten Speicherung der Schlüssel werden hierbei die Sensorknoten mit Polynomen zur Schlüsselgenerierung initialisiert.

Die Ausbringung der Sensorknoten nach der Schlüsselvorverteilung kann hierbei auf unterschiedliche Weise vollzogen werden. Eine Möglichkeit stellt die so genannte *Gruppenausbringung* dar [2]. Diese Ausbringung kann zum Beispiel durch die Benutzung mehrerer Fahrzeuge realisiert werden, bei der jedes Fahrzeug eine Menge von Sensorknoten, eine so genannte *Gruppe*, ausbringt. Sämtliche ausgebrachte Sensorknotengruppen sollen nach der Ausbringung das gemeinsame Sensornetz bilden.

Diese Arbeit konzentriert sich auf ein 2008 von Keith M. Martin, Maura B. Paterson und Douglas R. Stinson entwickeltes Verfahren der Schlüsselvorverteilung, das optimiert ist für eine Ausbringung der Sensorknoten eines Netzwerkes in Gruppen [2]. Hierbei wird gezeigt, dass dieses Verfahren gegenüber anderen vor allem aufgrund seiner Flexibilität Vorteile hinsichtlich der Konnektivität, des Schutzes vor unbefugtem Zugriff und des Speicherbedarfs besitzt.

In Kapitel 2 wird zunächst das Umfeld, in dem das Verfahren von Martin, Paterson und Stinson eingesetzt werden kann, charakterisiert. Anschließend werden in Kapitel 3 Kriterien zur Bewertung von Schlüsselvorverteilungsverfahren eingeführt. Blom's Schema (Kapitel 4.1), das Verfahren von Liu, Ning und Du und dessen Beschränkungen (Kapitel 4.2) sowie die mathematische Grundlage Transversal Design (Kapitel 4.3) dienen dem Verständnis des in Kapitel 4.4 vorgestellten Verfahrens von Martin, Paterson und Stinson. Dieses Verfahren wird hinsichtlich seiner Konnektivität (Kapitel 5.1), seines Schutzes vor unbefugtem Zugriff (Kapitel 5.2) und seiner Anforderungen an die Sensorknoten (Kapitel 5.3) bewertet. Die Ergebnisse werden in Kapitel 6 zusammengefasst.

2. ABGRENZUNG DES UMFELDS

Wie in der Einleitung bereits angesprochen, besitzen Sensorknoten Eigenschaften, die sich direkt auf die Gestaltung eines Verfahrens auswirken [3]:

- beschränkte Rechenleistung
- wenig Speicherplatz
- begrenzte Energieressource

- eingeschränkte Kommunikationsweite
- geringe Bandbreite der Kommunikation.

Daher wird eine Schlüsselvorverteilung für die Sensorknoten vorgenommen, d.h. jeder Sensorknoten wird mit bestimmten Schlüsseln initialisiert [2]. Die Knoten besitzen noch keinen Kontakt zueinander. Jedoch steht bereits bei der Schlüsselvorverteilung fest, dass sich die Sensorknoten in Gruppen gleicher Größe befinden werden und in welchen Gruppen die Knoten ausgebracht werden. Somit ist im Vorfeld für jedes beliebige Paar von Sensorknoten bekannt, ob sie sich in derselben Gruppe befinden werden oder nicht. Dementsprechend werden Schlüssel auf die Sensorknoten verteilt.

Anschließend werden die Sensorknoten in den festgelegten Gruppen ausgebracht. Hierbei werden sowohl die Gruppen als auch die Sensorknoten innerhalb einer Gruppe zufällig verteilt, d.h. es ist nicht bekannt, an welchem Ort sich die einzelnen Sensorknoten nach der Verteilung befinden werden.

Die Gruppenausbringung hat hierbei gegenüber einer zufälligen Ausbringung eines jeden einzelnen Sensorknotens den entscheidenden Vorteil, dass die Konnektivität des Sensornetzes durch die partielle Ortsgebundenheit der einzelnen ausgebrachten Gruppen erhöht werden kann.

Bei den Gruppen des Sensornetzes unterscheidet man grundsätzlich zwei Gestaltungsalternativen [2]:

- heterogene Gruppen
- homogene Gruppen.

Heterogene Gruppen bestehen aus mindestens einem Sensorknoten, der eine bessere technische Ausstattung im Vergleich zu den restlichen Sensorknoten aufweist und z.B. Kommunikationsaufgaben zwischen Gruppen übernimmt. Dahingegen wird eine homogene Gruppe aus einer Menge gleichartiger Sensorknoten gebildet.

Das in dieser Arbeit vorgestellte Verfahren geht von homogenen Sensorgruppen aus.

3. GÜTEKRITERIEN EINES VERFAHRENS

Zur Bewertung von Verfahren müssen Kriterien festgelegt werden.

Eine wesentliche Rolle bei der Bewertung eines Verfahrens zur Schlüsselvorverteilung spielen drei Aspekte, die Konnektivität des Sensornetzes, der Speicheraufwand eines jeden Sensorknotens und die Sicherheit des Sensornetzes vor einer unbefugten Übernahme.

3.1 Konnektivität

Nach der Ausbringung der Sensorknoten versuchen diese, gemeinsame Verbindungen herzustellen, so dass jeder Sensorknoten mit jedem beliebigen Knoten (auch über andere Knoten) kommunizieren kann.

Eine *direkte Verbindung* zwischen zwei Sensorknoten wird genau dann aufgebaut, wenn beide Knoten den gleichen Schlüssel besitzen und der Abstand der beiden Knoten kleiner oder gleich der drahtlosen Kommunikationsreichweite ist [2].

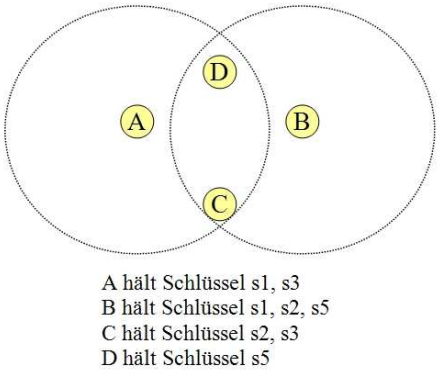


Abbildung 1: Sensornetz der Sensorknoten A, B, C und D nach der Initialisierung und Ausbringung

Zum Beispiel halten die Sensorknoten A und B des Sensornetzes in Abbildung 1 mit *s1* denselben Schlüssel, können aber keine direkte Verbindung aufbauen, da B nicht innerhalb des Kommunikationsradius von A liegt und umgekehrt. Stattdessen können {A,C}, {B,C} und {B,D} eine gemeinsame Verbindung, wie in Abbildung 2 dargestellt, aufbauen, wenn man davon ausgeht, dass die aufgrund der Übersichtlichkeit nicht eingezeichneten Kommunikationsradien von C und D denen von A und B entsprechen.

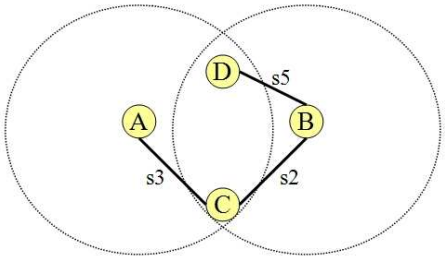


Abbildung 2: Sensornetz der Sensorknoten A, B, C und D nach dem Aufbau der Verbindungen

Das Ziel des Verbindungsaufbaus ist es, dass ein zusammenhängender Graph entsteht, damit je zwei Sensorknoten innerhalb des Netzwerkes miteinander kommunizieren können und somit eine optimale *globale Konnektivität* (global connectivity) entsteht [7]. Kann kein zusammenhängender Graph gebildet werden, so ist mindestens ein Sensorknoten vom Sensornetz abgeschnitten. Je mehr Sensorknoten vom Sensornetz getrennt sind, desto geringer ist die globale Konnektivität.

Die *lokale Konnektivität* (local connectivity) hingegen ist definiert als die Wahrscheinlichkeit, dass lokal benachbarte Knoten miteinander sicher und effizient kommunizieren können [7]. Sie ist als Bewertungskriterium im Vergleich zur globalen Konnektivität besser geeignet, da von den Abhängigkeiten bei der physischen Gruppenausbringung abstrahiert wird und zudem eine hohe lokale Konnektivität auch ein Indiz für eine hohe globale Konnektivität ist [2].

Zur Messung und Bewertung der lokalen Konnektivität werden die so genannten *2-Hop-Wege* (2-hop pathes) eingeführt [2]. Dies sind Kommunikationspfade zwischen zwei Knoten X und Y, die mit maximal einem weiteren Knoten Z gebildet werden können, wobei Z entweder Mitglied der Gruppe von X oder von Y ist. 2-Hop-Wege umfassen somit exakt die direkte Verbindung

zwischen X und Y sowie Verbindungen über einen Knoten Z, der nicht zu einer dritten Gruppe gehört.

Angenommen, die Knoten A, B, C und D in Abbildung 2 bilden eine Gruppe. Dann besteht zwischen A und B über den Knoten C genau ein 2-Hop-Weg. Zwischen A und D besteht kein 2-Hop-Weg.

3.2 Sicherheit vor unbefugtem Zugriff

Es wird davon ausgegangen, dass einzelne Sensorknoten von Unbefugten übernommen und ausgelesen werden können. Diese erhalten somit die Schlüssel, mit denen der ausgelesene Sensorknoten kommuniziert hat. Alle weiteren Verbindungen, die diesen Schlüssel nutzen, werden dadurch unsicher.

In den folgenden Abschnitten sei *fail(s)* die Wahrscheinlichkeit, dass eine sichere Verbindung zwischen zwei noch nicht übernommenen Knoten unsicher wird, nachdem Unbefugte Zugriff auf *s* zufällige Knoten erlangt haben [2]. Je kleiner *fail(s)*, desto größer ist die Sicherheit vor einer unbefugten Übernahme.

3.3 Speicheraufwand

Ein Verfahren muss so konstruiert sein, dass es möglichst wenig Speicherplatz auf den Sensorknoten belegt, da Sensorknoten in der Regel nur eine geringe Speicherkapazität zur Verfügung stellen und diese auch von anderen Aufgaben in Anspruch genommen wird [2].

Angenommen, „unterschiedliche paarweise Schlüssel“ (distinct pairwise keys) wird als Schlüsselvorverteilungsverfahren verwendet, d.h. jedem Paar von Sensorknoten wird je ein Schlüssel zur Kommunikation zugewiesen und keine zwei Paare von Sensorknoten besitzen den gleichen Schlüssel [2]. Dann muss jeder Sensorknoten in einem Sensornetz bestehend aus *n* Knoten *n-1* Schlüssel speichern. Dies bedeutet bei größeren Sensornetzen einen sehr hohen Speicheraufwand, wohingegen die Konnektivität und die Sicherheit vor einem unbefugten Zugriff optimal sind.

Der Speicheraufwand ist hingegen minimal, wenn zum Beispiel ein Sensornetz lediglich einen Schlüssel besitzt und jeder Sensorknoten genau diesen Schlüssel hält. Ebenso ist die Konnektivität optimal, jedoch bietet dies nur eine minimale Sicherheit vor einem unbefugten Zugriff. Ist ein Sensorknoten übernommen, so ist das gesamte Sensornetz unsicher.

4. VERFAHREN ZUR SCHLÜSSELVORVERTEILUNG

4.1 Blom’s Schema

1984 entwickelte Rolf Blom ein symmetrisches Schlüsselgenerierungssystem (symmetric key generation system). Dieses wird in dem nachfolgend vorgestellten, flexiblen Schlüsselvorverteilungsverfahren von Martin, Paterson und Stinson verwendet, um in einer Menge von Sensorknoten eine gesicherte direkte Verbindung zwischen jedem Paar von Sensorknoten herstellen zu können. Die für diese Arbeit relevanten Grundzüge des Schemas werden im Folgenden erklärt, für weitere Details sei auf [5] verwiesen.

4.1.1 *Verfahren*

Blom’s Schema verwendet ein symmetrisches, zweidimensionales Polynom $P(x,y)$ über einem endlichen Feld $GF(q)$, d.h. $P(i,j) = P(j,i)$ für alle $i,j \in GF(q)$ [2]. Jeder Sensorknoten mit einer ID *i* wird mit einem so genannten *Anteil* (share) von P, einem eindimensionalem Polynom $h_i(y)$ vom Grad *t*, initialisiert. Hierbei wird der Anteil $h_i(y)$ aus dem Polynom P berechnet, es gilt: $h_i(y) = P(i,y)$ für eine ID *i*. Damit eine Verbindung zwischen zwei Sensorknoten mit IDs *i* und *j* aufgebaut werden kann, muss der gemeinsame Schlüssel $K_{ij} = h_i(j) = h_j(i)$ von jedem Sensorknoten aus seinem zugeordneten Anteil berechnet werden. Somit kann zwischen jedem Paar von Sensorknoten eine sichere Verbindung aufgebaut werden.

4.1.2 *Eigenschaften*

Dieses Schema besitzt aufgrund je eines verwendeten Schlüssels bei der Kommunikation zwischen je zwei Sensorknoten dieselbe Konnektivität im Vergleich zu dem in Kapitel 3.3 erwähnten Verfahren „unterschiedliche paarweise Schlüssel“.

Im Speicher eines jeden Sensorknotens werden genau *t+1* Koeffizienten eines Polynoms vom Grad *t*, dem Anteil, gespeichert, um eine Verbindung zu allen anderen *n* Sensorknoten des Sensornetzes herstellen zu können.

Ein unbefugter Zugriff des Sensornetzes ist erst dann möglich, wenn *t+1* oder mehr Sensorknoten des Netzes übernommen wurden, denn dann kann das Polynom P interpoliert und somit alle Schlüssel berechnet werden.

4.2 Liu, Ning und Du’s Verfahren

Liu, Ning und Du haben 2005 ein Verfahren zur Schlüsselvorverteilung unter Verwendung von Gruppenausbringung veröffentlicht [4], welches einen Ausgangspunkt für das von Martin, Paterson und Stinson entwickelte Verfahren darstellt und zu dessen Verständnis beiträgt. Es wird im Folgenden zunächst kurz vorgestellt, bevor anschließend aufgezeigt wird, unter welchen Bedingungen dessen Einsatz an Grenzen stößt.

4.2.1 *Verfahren*

In Liu, Ning und Du’s Verfahren werden die Sensorknoten auf Gruppen verteilt. Zudem werden zur Kommunikation zwischen den Gruppen so genannte *Kreuzgruppen* (cross-groups) gebildet. Hierbei enthält jede Kreuzgruppe genau einen Knoten aus jeder Gruppe und jeder Sensorknoten einer gewöhnlichen Gruppe ist in genau einer Kreuzgruppe enthalten.

Innerhalb einer Gruppe oder Kreuzgruppe soll jeder Knoten mit jedem anderen Knoten der gleichen Gruppe über jeweils einen gemeinsamen Schlüssel der Kommunikationspartner kommunizieren können. Hierzu werden Schlüssel innerhalb der Gruppen nach dem Prinzip der „unterschiedlichen paarweisen Schlüsseln“ (siehe Kapitel 3.3) oder nach Blom’s Schema (siehe Kapitel 4.1) vergeben. Somit kann durch die festgelegten Gruppen und Kreuzgruppen sowie einer entsprechenden Schlüsselvorverteilung ein zusammenhängendes Sensornetz nach der Sensorenausbringung gebildet werden.

Im Folgenden wird dieses Verfahren an einem Beispiel gezeigt. Eine Menge von 9 Sensorknoten a bis i soll in Gruppen zu je drei Knoten aufgeteilt werden. Abbildung 3 zeigt eine mögliche Verteilung. Hierbei wurden die Gruppen {a,b,c}, {d,e,f} und {g,h,i} gewählt. Drei mögliche Kreuzgruppen, die Liu, Ning und Du’s Schema genügen, sind {a,d,g}, {b,e,h} und {c,f,i}. Nun kann beispielsweise Knoten a mit Knoten b, c, d und g direkt kommunizieren, sofern die physischen Gegebenheiten dazu erfüllt werden.

Abbildung 3 veranschaulicht das resultierende Sensornetz graphisch. Eine gewöhnliche Gruppe ist durch ein Oval dargestellt, eine Kreuzgruppe entsteht durch je zwei Verbindungen zwischen 3 Knoten.

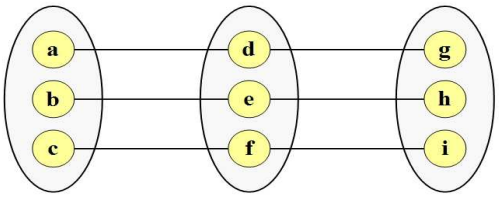


Abbildung 3: Beispiel zu Liu, Ning und Du's Verfahren

4.2.2 *Beschränkungen*

Hinsichtlich der in Kapitel 3 vorgestellten Gütekriterien wird im Folgenden gezeigt, dass das von Liu, Ning und Du entwickelte Verfahren im Bereich der Konnektivität Schwächen aufweist.

Zunächst lässt sich aber feststellen, dass bei einer Schlüsselvorverteilung innerhalb der Gruppen und Kreuzgruppen unter Verwendung von Blom's Schema eine individuelle Abstimmung zwischen Speicheraufwand der einzelnen Sensorknoten und der Sicherheit vor einem unbefugten Zugriff getroffen werden kann [2]. Je größer der Grad t gewählt wird, desto geschützter ist das Sensornetz vor einer unbefugten Übernahme. Je kleiner der Grad t gewählt wird, desto geringer ist der Speicherbedarf der einzelnen Sensorknoten.

Die Konnektivität dieses Verfahrens weist allerdings Mängel auf [2].

Angenommen, es werden n Sensorknoten in λ Gruppen ausgebracht, dann beträgt die Wahrscheinlichkeit, dass sich Knoten verschiedener Gruppen auch innerhalb einer Kreuzgruppe befinden und miteinander kommunizieren können, λ / n .

Bei einer Ausbringung in wenigen Gruppen, also bei einem kleinen Wert von λ, ist diese Wahrscheinlichkeit sehr gering. Dies birgt einerseits das Risiko, dass ganze Gruppen nicht mit dem Sensornetz verbunden werden können und andererseits kann es zu Engpässen zwischen zwei Gruppen kommen, zwischen denen nur wenige Sensorknoten für die Verbindung zuständig sind.

Das in Kapitel 4.4 vorgestellte Verfahren zur Schlüsselvorverteilung mit Transversal Designs löst diese Beschränkung der Konnektivität. Hierzu wird ein zusätzlicher Parameter eingeführt, wodurch die Konnektivität zwischen Gruppen auch bei einer Ausbringung in wenigen Gruppen beliebig verändert werden kann.

4.3 Transversal Design

Bevor das Verfahren von Martin, Paterson und Stinson erläutert wird, wird zu dessen Verständnis zunächst mit einem Transversal Design eine dazu benötigte mathematische Grundlage eingeführt.

Definition 1 [2]: Ein *Transversal Design TD(k, n)* besteht aus

- einer Menge V mit genau $k \bullet n$ Elementen. Die Elemente werden als *Punkte* bezeichnet.
- einer Partition G der Punkte von V in k Mengen mit je n Elementen. Die Elemente von G werden als *Design-Gruppen* bezeichnet.
- und einer Menge B, bestehend aus k-Teilmengen von V mit der Eigenschaft, dass jedes Paar unterschiedlicher Punkte in genau einem Element von G ∪ B enthalten ist. Die Elemente von B werden als *Blöcke* bezeichnet.

Beispielsweise sei k = 2 und n = 3. Dann besitzt die Menge V des Transversal Designs TD(2,3) 6 Elemente. Diese Elemente seien mit A bis F bezeichnet:

$$V = \{A,B,C,D,E,F\}$$

Die Menge V lässt sich unter anderem wie folgt in Elemente der Menge G zerlegen:

Design-Gruppe 1: {A,B,C}

Design-Gruppe 2: {D,E,F}

Eine Menge, bestehend aus allen 2er-Teilmengen der Menge V, enthält $\binom{k \bullet n}{2} = \binom{2 \bullet 3}{2} = 15$ Elemente. Mit der Eigenschaft, dass jedes Paar unterschiedlicher Punkte nur in genau einem Element von G ∪ B enthalten ist, ergibt sich B wie folgt:

$$B = \{\{A,D\},\{A,E\},\{A,F\},$$
$$\{B,D\},\{B,E\},\{B,F\},$$
$$\{C,D\},\{C,E\},\{C,F\}\}$$

Definition 2 [2]: Ein Transversal Design TD(k, n) ist *auflösbar*, falls die Blöcke der Menge B in Mengen B_1, B_2, …, B_S partitioniert werden können, so dass jeder Punkt der Menge V in genau einem Block in jeder dieser Mengen vorkommt. Diese Mengen werden als *Parallelklassen* bezeichnet.

Zur Erläuterung dieser Definition wird das obige Beispiel aufgegriffen. Die dort ermittelte Menge B kann in folgende drei parallele Klassen partitioniert werden:

Parallelklasse 1 = {{A,D},{B,E},{C,F}}

Parallelklasse 2 = {{A,E},{B,F},{C,D}}

Parallelklasse 3 = {{A,F},{B,D},{C,E}}

Die durch ein auflösbares Transversal Design erhaltenen Parallelklassen werden im Verfahren von Martin, Paterson und Stinson benötigt.

4.4 Schlüsselvorverteilung mit einem Transversal Design

Im Folgenden wird das von Keith M. Martin, Maura B. Paterson und Douglas R. Stinson entwickelte Verfahren vorgestellt [2]. Die Definition bezieht sich auf ein Sensornetz mit n^2 Sensorknoten, welche in λ Gruppen G_1, G_2, …, G_λ der Größe n^2 / λ ausgebracht werden sollen.

Definition 3: Gegeben sei eine Primzahlpotenz n und zwei positive Zahlen k und t, für die gilt: $1 \le k \le n$, $\lambda \mid n$ (λ ist ein Teiler von n) und $0 \le t < (n^2 / \lambda) - 1$.

1. Zerlege die n Parallelklassen P_1, P_2, …, P_n des Transversal Designs TD(k, n) in λ Mengen mit je n / λ Parallelklassen. Diese Mengen werden im Folgenden als S_1, S_2 bis S_λ bezeichnet und jede Menge S_i enthält genau n^2 / λ Blöcke.

2. Ordne jedem Block einen Sensorknoten zu. Alle zu den Blöcken einer Menge S_i zugeordneten Sensorknoten bilden die Gruppe G_i.
3. Jedem Punkt der Menge V des TD(k, n) wird ein Polynom von Blom's Schema zugeordnet und jedem Sensorknoten, dessen Block den entsprechenden Punkt aus V enthält, wird ein Anteil vom Grad t zugeordnet.
4. Jeder Parallelklasse P_i wird ein Polynom gemäß Blom's Schema zugeordnet und jedem Sensorknoten, dessen Block in der Parallelklasse P_i liegt, wird ein Anteil vom Grad t zugeordnet.

Durch Anwendung dieser Definition können alle n^2 Sensorknoten mit Anteilen nach Blom's Schema initialisiert werden.

Als erläuterndes Beispiel sei $k = 2$, $n = 3$, $\lambda = 3$ und $t = 1$. Dann wird das in Definition 3 beschriebene Verfahren ausgeführt:

1. Die im vorangegangenen Kapitel 4.3 ermittelten drei Parallelklassen zu einem auflösbaren TD(2,3) können wegen $k = 2$ und $n = 3$ wieder aufgegriffen werden. Da $\lambda = n$ sind dies die Mengen S_1, S_2 und S_3.
2. Nun werden den Blöcken aus S_i die Sensorknoten a bis i zugeordnet und es entstehen folgende Gruppen:

$$G_1 = \{a,b,c\}$$
$$G_2 = \{d,e,f\}$$
$$G_3 = \{g,h,i\}$$

3. Sei $f_x(X)$ die Notation für den Anteil nach Blom's Schema, der dem Sensorknoten x, dessen zugeordneter Block den Punkt X enthält, zugeordnet ist. Sensorknoten a wurde dem Block {A,D} zugeordnet und somit bekommt a die Anteile $f_a(A)$ sowie $f_a(D)$. Analoges gilt für die weiteren Sensorknoten. Somit wurden Anteile zur Kommunikation zwischen den Gruppen G_1, G_2 und G_3 verteilt und implizit Kreuzgruppen gebildet.
4. Sei $f_x(i)$ die Notation für den Anteil nach Blom's Schema, der dem Sensorknoten x, dessen zugeordneter Block in der i-ten Parallelklasse P_i liegt, zugeordnet ist. Der Sensorknoten a liegt in der Parallelklasse 1 und somit bekommt a den Anteil $f_a(1)$. Analoges gilt für die weiteren Sensorknoten. Durch die verteilten Anteile kann eine sichere Kommunikation innerhalb einer Gruppe stattfinden.

Damit ist das Verfahren vollständig ausgeführt. Die Sensorknoten benötigen keine weiteren Informationen zum sicheren Kommunikationsaufbau innerhalb des Sensornetzes. Es ergibt sich die folgende Verteilung der Anteile auf die Sensorknoten:

a: $\{f_a(A), f_a(D), f_a(1)\}$ d: $\{f_d(A), f_d(E), f_d(2)\}$

b: $\{f_b(B), f_b(E), f_b(1)\}$ e: $\{f_e(B), f_e(F), f_e(2)\}$

c: $\{f_c(C), f_c(F), f_c(1)\}$ f: $\{f_f(C), f_f(D), f_f(2)\}$

g: $\{f_g(A), f_g(F), f_g(3)\}$

h: $\{f_h(B), f_h(D), f_h(3)\}$

i: $\{f_i(C), f_i(E), f_i(3)\}$

Nun können die Sensorknoten, physische Gegebenheiten vorausgesetzt, entsprechend ihrer gehaltenen Anteile miteinander direkte Verbindungen aufbauen. Sensorknoten a kann mit den Sensorknoten b und c über die Anteile $f_x(1)$ kommunizieren, mit den Sensorknoten d und g über $f_x(A)$ und mit den Sensorknoten f und h über $f_x(D)$. D.h. Sensorknoten a kann innerhalb der Gruppe {a,b,c} sowie den Kreuzgruppen {a,d,g} und {a,f,h} mit allen anderen Knoten der Gruppen direkt kommunizieren. Analoges gilt für alle weiteren Sensorknoten.

Eine graphische Veranschaulichung der Gruppen, innerhalb derer direkte Kommunikation zwischen jedem beliebigen Knotenpaar stattfinden kann, findet sich in Abbildung 4. Hierbei wird eine Kreuzgruppe jeweils durch zwei gestrichelte (bzw. durchgezogene) Verbindungen zwischen drei Knoten dargestellt, eine gewöhnliche Gruppe durch ein umschließendes Oval. Insgesamt werden somit die entstandenen sechs Kreuzgruppen und die drei gewöhnlichen Gruppen des Beispiels repräsentiert.

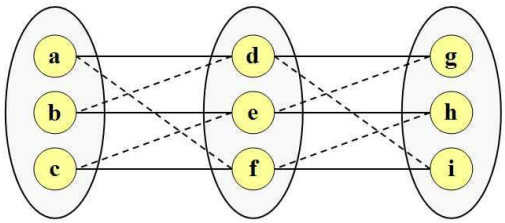


Abbildung 4: Beispiel zur Schlüsselvorverteilung mit einem Transversal Design

5. BEWERTUNG DES VERFAHRENS

5.1 Konnektivität

5.1.1 *Flexibilität der Konnektivität*

Die Konnektivität des vorgestellten Schlüsselvorverteilungsverfahrens mit einem Transversal Design kann flexibel in Abhängigkeit der Parameter k und λ eingestellt werden [2]. Dies ist ein Vorteil gegenüber dem Verfahren von Liu, Ning und Du, da dieses keine Parameter zur Einstellung einer gewünschten Konnektivität bietet. In dem Verfahren mit einem Transversal Design hingegen bewirkt eine Erhöhung von k oder λ stets eine Konnektivitätssteigerung.

Allerdings hat diese Konnektivitätssteigerung auch ihre Grenzen. Für eine Erhöhung von k wird mehr Speicherplatz auf jedem Sensorknoten benötigt. Eine Erhöhung von λ, und somit eine Erhöhung der Gruppenanzahl, kann bei einer sehr hohen Anzahl der Gruppen zu Schwierigkeiten in der praktischen Umsetzung führen.

5.1.2 *Vergleich mit anderen Verfahren mittels Simulationen*

Im Folgenden werden Simulationsumgebungen eingerichtet und durchgeführt [2]. Hierbei wird gezeigt, dass das Verfahren von Martin, Paterson und Stinson (MPS) im Vergleich zu dem Verfahren von Liu, Ning und Du (LND) und einem der bekanntesten zur Schlüsselvorverteilung genutzten Verfahren, das von Eschenauer und Gligor (EG) (für Details siehe [6]), auch bei einem vordefinierten Speicherplatz Vorteile bezüglich der lokalen und globalen Konnektivität bietet.

Die Simulationsumgebung besteht aus einem Sensornetz mit 16384 Sensorknoten, welche in $\lambda = 16$ Gruppen zu je 1024 Knoten ausgebracht werden. In jedem Sensorknoten können $m = 32$ Schlüssel bzw. Koeffizienten gespeichert werden. Zudem wird mit Pr_1 der erwartete Anteil der Knotenpaare, die eine Verbindung mit einem gemeinsamen Schlüssel aufbauen können, bei jeder Simulationsdurchführung vorab festgelegt. Diese Bindungen

ermöglichen insgesamt einen gerechten Vergleich zwischen den einzelnen Verfahren.

Der Speicherplatz der Verfahren LND und MPS wird durch feste Parameterwerte für *k* und *t* fixiert. Das ohne Gruppenausbringung realisierte Verfahren EG wird durch den Parameter *K* beschränkt, der die Anzahl aller verwendeter Schlüssel repräsentiert.

Anhand des festgelegten Umfelds können die die lokale Konnektivität beschreibenden Größen $2Hop_g$ und $2Hop_k$ berechnet werden. $2Hop_g$ (bzw. $2Hop_k$) entspricht der Wahrscheinlichkeit, dass zwei Knoten einer Gruppe (bzw. einer Kreuzgruppe) eine gemeinsame Verbindung über einen 2-Hop-Weg aufbauen können.

Außerdem wird mit *E(K)* eine die globale Konnektivität beschreibende Größe ermittelt, welche die durchschnittliche Anzahl an Sensorknoten der größten zusammenhängenden Komponente in dem entstandenen Sensornetzwerk, basierend auf 100 Wegen, abschätzt. Hierbei wird eine Ausbringung der Knoten auf ein Gebiet von 500 m x 500 m simuliert, wobei alle Knoten einer Gruppe gleichmäßig in einem Kreis mit Radius 100 m verteilt werden und jeder Knoten eine Kommunikationsreichweite von 6 m besitzt.

Die Simulationsergebnisse finden sich in Tabelle 1.

Tabelle 1: Konnektivitätsmessung eines Sensornetzes bestehend aus 16 Gruppen mit je 1024 Knoten [2]

Schema	Parameter	Pr_1	$2Hop_g$	$2Hop_k$	E(K)
LND	t = 17	0,0634	1,00	0,0145	1510
MPS	k = 7, t = 3	0,0620	0,329	0,172	2140
EG	K = 15760	0,0630	0,0888	0,114	1800
MPS	k = 15, t = 1	0,124	0,466	0,395	8640
EG	K = 7750	0,124	0,215	0,296	7600
MPS	k = 31, t = 0	0,248	0,713	0,769	13200
EG	K = 3620	0,248	0,518	0,691	12600

Aus der Tabelle ist ersichtlich, dass das Schlüsselvorverteilungsverfahren mit einem Transversal Design (MPS) in jeder Simulation stets eine höhere lokale und globale Konnektivität als das Verfahren EG aufweist – repräsentiert durch die Werte $2Hop_g$, $2Hop_k$ und *E(K)*.

Außerdem zeigt sich in der ersten durchgeführten Simulationsumgebung ein deutlicher Unterschied der globalen Konnektivität zwischen LND und MPS. Hierbei weist die Gruppenausbringung nach LND sogar im Vergleich zur zufälligen Ausbringung nach EG eine schlechtere globale Konnektivität auf (aufgrund der geringen lokalen Konnektivität zwischen den einzelnen Gruppen, ausgedrückt durch $2Hop_k$), so dass von einer Anwendung des LND-Verfahrens in dieser Simulationsumgebung abzuraten ist. Dagegen erzielt das Verfahren MPS aufgrund seiner Flexibilität die besten Simulationsergebnisse.

5.2 Sicherheitskriterien

Sichere Verbindungen zwischen einzelnen Sensorknoten in dem Sensornetz werden mittels Schlüsseln gebildet. Besitzt ein Unbefugter den Verbindungsschlüssel zwischen zwei Sensorknoten, so ist die Verbindung unsicher.

5.2.1 *Sicherheit gegen unbefugten Zugriff*

Ein Maß für die Sicherheit gegen unbefugten Zugriff wurde in Kapitel 3.2. mit *fail(s)* eingeführt, wobei *s* die Anzahl der durch Unbefugte übernommenen Knoten darstellt. Für das zur Schlüsselvergabe verwendete Schema von Blom gilt, dass eine gesicherte Verbindung zwischen zwei Knoten genau dann unsicher wird, wenn mehr als *t* Knoten mit einem entsprechenden Anteil übernommen wurden. Deshalb gilt für das Verfahren der Schlüsselvorverteilung mit einem Transversal Design [2]:

$$fail(s) = \begin{cases} 0 & \text{für} \quad s \le t \\ 1 - \sum_{i=0}^{t} \frac{\binom{n-2}{i}\binom{n^2-n}{s-i}}{\binom{n^2-2}{s}} & \text{für} \quad s > t \end{cases}$$

Die Sicherheit von Blom's Schema und damit die Sicherheit des Verfahrens von Martin, Paterson und Stinson kann flexibel, aber zu Lasten des Speicherbedarfs, erhöht werden, indem Polynome höheren Grades verwendet werden. D.h. je größer *t* gewählt wird, desto sicherer ist das Verfahren.

5.2.2 *Vergleich mit anderen Verfahren mittels Simulationen*

Im Folgenden wird gezeigt, dass das Verfahren von Martin, Paterson und Stinson (MPS) auch bezüglich der Sicherheit vor einem unbefugten Zugriff im Vergleich zu dem Verfahren von Liu, Ning und Du (LND) und dem Verfahren von Eschenauer und Gligor (EG) Vorteile besitzt [2].

Betrachtet wird ein Sensornetz mit 16384 Knoten, das durch das Transversal Design TD(39,128) gebildet wurde und aus 16 Gruppen zu je 1024 Knoten besteht. Zur Vergleichbarkeit der Verfahren werden wie in Kapitel 5.1.2 die Parameter *m* und Pr_1 verwendet. Ebenso werden für die Verfahren LND und EG die Formeln für *fail(s)* nach [2] aufgestellt.

Nun können die drei Verfahren in Abhängigkeit der Werte m und Pr_1 miteinander verglichen werden.

Sei $m = 32$ und $Pr_1 \approx 0{,}063$. Die zugehörigen Funktionsverläufe der Funktion *fail(s)* zu den drei Verfahren sind für $s \in [0{,}1000]$ in Abbildung 5 dargestellt.

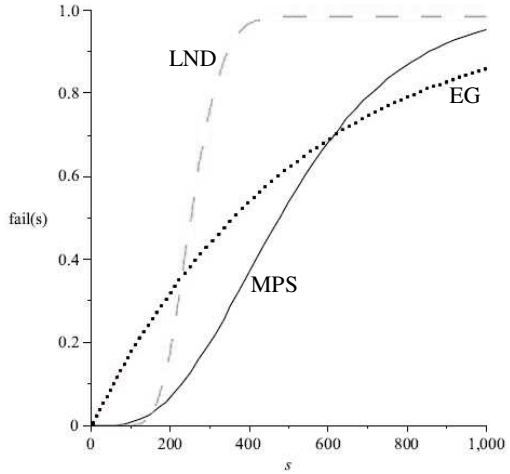


Abbildung 5: Vergleich des Verfahrens MPS mit den Verfahren LND und EG für $m = 32$ und $Pr_1 \approx 0{,}063$ [2]

Hierbei erkennt man, dass LND für kleine Werte von s die größte Sicherheit vor einem unbefugten Zugriff bietet, aber bereits ab ca. 300 übernommenen Knoten ist das gesamte Sensornetz unsicher. Dahingegen liefert MPS für Werte von s von ca. 200 bis ca. 600 den besten Widerstand. Erst ab $s \approx 600$ ist das Verfahren von Eschenauer und Grigor am günstigsten, allerdings erweist es sich bis $s \approx 600$ als deutlich schlechter im Vergleich zu MPS. Das Verfahren von Martin, Paterson und Stinson weist somit im Vergleich zu LND und EG für $Pr_1 \approx 0{,}063$ günstige Sicherheitswerte auf.

5.3 Anforderungen an die Knoten

Das Verfahren stellt hinsichtlich Speicherbedarf und Rechenleistung Anforderungen an einen Sensorknoten.

Auf einem Sensorknoten müssen bei Anwendung des Verfahrens $(k+1)(t+1)$ Schlüssel bzw. Koeffizienten der Polynome von Blom's Schema gespeichert werden und eine dementsprechende Speicherkapazität ist auf jedem Sensorknoten erforderlich [2].

Der Speicherbedarf kann aber durch die beiden Parameter k und t angepasst werden. Je kleiner k oder t, desto weniger Speicherplatz wird auf dem Sensorknoten benötigt. Allerdings bedeutet ein kleinerer Wert für k nach [2] auch Konnektivitätseinbußen und ein geringerer Wert für t eine reduzierte Sicherheit vor unbefugter Übernahme.

Hinsichtlich der Rechenleistung erfordert die Verwendung von Blom's Schema zusätzlichen Rechenaufwand im Vergleich zu herkömmlichen Schlüsselvorverteilungsverfahren, in denen der Schlüssel direkt auf dem Sensorknoten gespeichert wird. Ein zur Kommunikation mit einem anderen Sensorknoten benötigter Schlüssel muss erst berechnet werden, indem das gespeicherte Polynom (der Anteil nach Blom's Schema) mit der Kennung des Kommunikationspartners ausgewertet wird. Dies muss bei jedem Verbindungsaufbau durchgeführt werden.

6. ZUSAMMENFASSUNG

Das vorgestellte Verfahren von Martin, Paterson und Stinson zur Schlüsselvorverteilung bietet zahlreiche Vorteile, es weist aber auch Nachteile gegenüber bewährten Verfahren auf.

Zur Initialisierung der Sensorknoten müssen zunächst mehrere mathematische Schritte und Operationen durchgeführt werden. Andere Verfahren zur Schlüsselvorverteilung, wie beispielsweise das Verfahren von Liu, Ning und Du oder das Verfahren von Eschenauer und Gligor, besitzen einen niedrigeren Komplexitätsgrad und damit eine einfachere Sensorknoteninitialisierung.

Zudem verwendet das Verfahren von Martin, Paterson und Stinson zur Schlüsselgenerierung Blom's Schema. Hierdurch muss ein Sensorknoten bei jedem Aufbau einer sicheren Verbindung zu einem anderen Sensorknoten den gemeinsamen Schlüssel aus seinem gespeichertem Polynom berechnen. Dies wäre bei einer direkten Speicherung der Schlüssel auf den Sensorknoten nicht notwendig.

Dennoch kann trotz der genannten Nachteile der Einsatz des Verfahrens äußerst sinnvoll sein.

Durchgeführte Simulationen demonstrieren, dass das Verfahren von Martin, Paterson und Stinson gegenüber dem Verfahren von Liu, Ning und Du sowie dem Verfahren von Eschenauer und Gligor in bestimmten Umgebungen die besten Ergebnisse hinsichtlich Konnektivität und Sicherheit liefert.

Außerdem kann die Konnektivität und der Schutz vor unbefugtem Zugriff durch zusätzliche Speicherbenutzung mittels Parametereinstellungen gesteigert und somit gezielt, der Anwendung entsprechend, angepasst werden.

Daneben führt auch die Ausweitung der Gruppenanzahl bei der Ausbringung der Sensorknoten zu Konnektivitätssteigerungen.

Insgesamt sollte man aufgrund der dargelegten Verbesserungen und der flexiblen Anpassungsmöglichkeiten den Einsatz dieses Verfahrens in Erwägung ziehen.

Zur Vereinfachung des Verfahrens sowie zur Reduzierung der Rechenoperationen der Sensorknoten könnte anstelle von Blom's Schema ein anderes Verfahren, zum Beispiel das von Eschenauer und Gligor, verwendet werden. Es ist aber fraglich, wie sich dies auf die Konnektivität und die Sicherheit des entstehenden Sensornetzes auswirkt.

7. LITERATURQUELLEN

[1] Römer, K., and Mattern, F. 2004. The design space of wireless sensor networks. IEEE Wireless Communications, Vol. 11, No. 6, pp. 54-61.

[2] Martin, K. M., Paterson, M. B., and Stinson, D. R. 2008. Key predistribution for homogeneous wireless sensor networks with group deployment of nodes. Cryptology ePrint Archive, Report 2008/412.

[3] Chan, H., Perrig, A., and Song, D. 2003. Random key predistribution schemes for sensor networks. SP '03: Proceedings of the 2003 IEEE symposium on security and privacy.

[4] Liu, D., Ning, P., and Du, W., 2008. Group-based key predistribution in wireless sensor networks. ACM transactions on sensor networks, Vol. 4, No. 2, pp. 1-30. DOI = 1340771.1340777

[5] Blom, R., 1985. An optimal class of symmetric key generation systems. Proceedings of the EUROCRYPT 84 workshop on advances in cryptology: theory and application of cryptographic techniques, pp. 335-338.

[6] Eschenauer, L., and Gligor, V. D. 2002. A key management scheme for distributed sensor networks. Proceedings of the 9th ACM conference on computer and communication security, pp. 41-47. DOI = 586110.586117

[7] Lee, J., and Stinson, D. R. 2008. On the construction of practical key predistribution schemes for distributed sensor networks using combinatorial designs. ACM Transactions on Information and System Security, Vol. 11, Issue 2, pp. 1-35. DOI = 1330332.1330333

Routing in Sensornetzen - Angriffe auf ausgewählte Protokolle und Lösungsansätze

Nadine Herold
Betreuer: Alexander Klein
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: nadine.herold@onlinehome.de

KURZFASSUNG

Die Nutzung von Sensornetzwerken und deren Bedeutung nimmt immer weiter zu. Dies führt unter anderem zu vermehrten Angriffen auf diese Netzwerke. In diesem Paper sollen verschiedene Angriffe vorgestellt und deren Auswirkungen auf bestimmte Protokolle untersucht werden. Wichtige Fragestellungen, wie die Klassifizierung des Angreifers, die Möglichkeiten des Angreifers im Netz und die Erkennung des Angriffs werden betrachtet. Zudem soll ein Überblick über Gegenmaßnahmen aufzeigen, wie ein Schutz gegen diese Angriffe möglich ist und welche Protokolle anfällig für Attacken sind.

Schlüsselworte

AODV, SBR, MCFA, Wurmloch, Sybil-Attacke, Sinkhole-Attacke, Sensornetze, Routing, Angriffe, Gegenmaßnahmen

1. EINLEITUNG

Sensornetze, d.h. räumlich verteilte Netze von programmierbaren und mit Sensoren ausgestattete Recheneinheiten, gewinnen durch ihre Vielseitigkeit zunehmend an Bedeutung. Als kostengünstige Einheit lassen sie sich in vielen Bereichen leicht einsetzen. Die große Masse der Sensorknoten ist neben der Mobilität der einzelnen Knoten eine wichtige Eigenschaft, daher nimmt das Routing in einem solchen System einen hohen Stellenwert ein [5].

Die Abgrenzung zu den klassischen Ad-hoc Netzen ist fließend und daher sehr schwierig. Innerhalb eines Sensornetzes werden mehr Knoten eingesetzt als ein einem Ad-hoc Netz. Auf Grund der Beweglichkeit der Sensoren ändert sich auch die Topologie des Netzes schneller als ein einem klassischen Ad-hoc Netz. Zudem kann ein Sensorknoten nur auf beschränkte Ressourcen zurückgreifen, sowohl an Rechenleistung als auch an Speicherplatz [5].

Diese Einschränkungen haben dazu geführt, dass die herkömmlichen MANET-(mobile ad-hoc networks) Protokolle für Ad-hoc Netzwerke durch speziell für Sensornetze entwickelte Protokolle ersetzt werden sollten. Einige Studien zeigen jedoch, dass sich die MANET Protokolle im praktischen Einsatz besser eignen als spezielle Sensornetzprotokolle [5, 29]. Hier hat sich vor allem das Ad-hoc On-Demand Distance Vektor Routing Protokoll (AODV) bewährt, daher wird es in die Betrachtungen einbezogen.

In der folgenden Arbeit werden nun zunächst die einzelnen Protokolle AODV, SBR (Statistic-Based Routing) und MCFA (Minimum Cost Forwarding Algorithmus) betrachtet. Im folgenden Kapitel wird der Angreifer beschrieben und seine möglichen Attacken betrachtet. Es folgt eine Auswertung, wie sich mögliche Attacken auf die vorgestellten Protokolle auswirken. Das letzte Kapitel beschäftigt sich mit Gegenmaßnahmen zum Schutz des Sensornetzes. Eine Zusammenfassung und Auswertung der Ergebnisse bilden den Schluss der Arbeit.

2. ROUTINGPROTOKOLLE

In diesem Kapitel soll ein kurzer Überblick über die Routing-Protokolle AODV [22], SBR [12] und MCFA [28] geschaffen werden. Es werden nur die grundlegenden Mechanismen erklärt. Details werden behandelt, wenn diese für einen späteren Angriff wichtig sind.

2.1 AODV

Das AODV Routing Protokoll folgt dem reaktiven on-demand Ansatz, d.h. Routen werden erst bei Bedarf etabliert. Es gibt daher keine periodischen Updates der Routingtabellen. Eine vollständige Sicht auf die Netztopologie steht den Knoten nicht zur Verfügung [15]. Wird nun eine Route benötigt, sendet der Quellknoten einen sogenannten Route Request (RREQ) in das Netz (broadcast). Dieser enthält u.a. die Quelladresse, eine Broadcast ID und die Zieladresse. Die Quelladresse und die Broadcast ID identifizieren jeden RREQ eindeutig und verhindern Replay-Angriffe sowie unnötige Netzlast [22].

Empfängt ein Zwischenknoten, der keine Route zum Ziel kennt, einen RREQ, dann wird der RREQ und zusätzlich die eigene ID des Zwischenknotens weiter geleitet. Der Zwischenknoten merkt sich neben dem RREQ auch den Knoten, von welchem er den RREQ erhalten hat. Dieser Vorgang nennt sich Reverse Path Setup [22]. Kennt der Zwischenknoten eine Route zum Ziel oder handelt es sich um den Zielknoten selbst, wird mit einem Route Reply (RREP) geantwortet. Dieser enthält u.a. die Ziel- und Quelladresse. Der RREP wird an den Knoten gesandt, von welchem der letzte RREQ kam. Nun beginnt das Forward Path Setup. Der RREP wird an den Knoten gesandt, von welchem der ursprüngliche RREQ weitergeleitet wurde. Die Zwischenknoten konfigurieren entsprechend ihre Routingtabellen.

Der Protokollverlauf ist in Abbildung 1 dargestellt. Die erste Abbildung zeigt das Netz, der Knoten A möchte mit G

kommunizieren. A sendet einen RREQ aus, den B und C empfangen. B leitet den RREQ weiter an D und F, C leitet den RREQ weiter an E. E erhält zusätzliche RREQ von D und F, die verworfen werden und leitet den RREQ weiter an F. G sendet einen RREP über den Pfad E-C-A. Die Kommunikation zwischen A und G ist nun möglich.

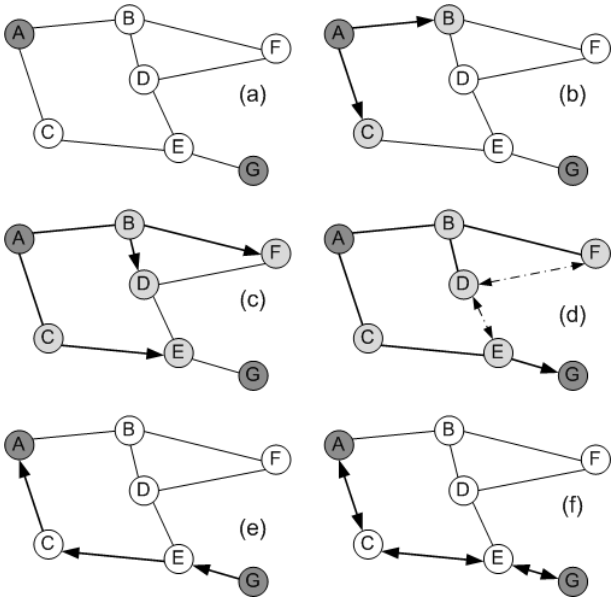


Abbildung 1: Kommunikation von A nach G

Die etablierten Routen bleiben in den Routingtabellen, solange sie aktiv sind oder bis ein Zwischenknoten ausfällt. Die Prüfung der Aktivität kann über einen Timeout geregelt werden, sodass eine Route nach einer festen Zeitspanne gelöscht wird [22]. Fällt ein Zwischenknoten aus wird eine Route Error Nachricht (RERR) zum Quellknoten gesendet [24].

Eine Erweiterung des Protokolls wird in [15] vorgestellt. AODV-BR legt während des Routenfindungsprozesses alternative Routen an, sodass im Falle eines Knotenausfalls der Datenverkehr fortgesetzt werden kann. Fällt ein Knoten aus, werden die Daten über die Alternativrouten umgeleitet. Gleichzeitig wird eine neue Route gesucht, um suboptimale Pfade zu verhindern [15].

2.2 SBR

Im Gegensatz zum AODV Protokoll wurde das SBR Protokoll speziell für den Einsatz in Sensornetzwerken entwickelt. Ziel war es hier, die Energiebeschränkungen einzuhalten und wenig Overhead und Rechenaufwand zu generieren [1].

Das SBR-Protokoll basiert auf der Verwendung von HELLO-Nachrichten. Jeder Knoten sendet periodisch HELLO--Nachrichten mit folgenden Informationen aus: Quellknoten, d.h. die eigene Adresse, eine Sequenznummer zur Identifizierung der HELLO-Nachricht, die von der Quelle bei neuen HELLO-Nachrichten inkrementiert wird, und ein Time-To-Live Feld, um zu steuern, wie lange eine HELLO-Nachricht weitergeleitet wird. In einem Intermediate-Feld wird der Knoten gespeichert, welcher die HELLO-Nachricht weitergeleitet hat [13].

Tabelle 1: Routingtabelle des Knoten A

Knoten A	empfangene HELLO's				
Quelle	B	C	D	E	F
B	50	-	-	-	-
C	-	60	-	-	-
D	9	-	-	-	-
E	-	36	-	-	-
F	40	-	-	-	-

Jeder Knoten besitzt eine Routing-Tabelle mit den ihn bekannten Sensorknoten (Quellen der HELLO-Nachrichten). Zusätzlich wird vermerkt, über welche Knoten diese Quellen zu erreichen sind. Ein generischer Wert gibt Aussage über die Linkqualität zur Quelle über den Zwischenknoten [12].

Erhält der Knoten einen HELLO-Request von einer Quelle, die noch nicht in der Routing-Tabelle auftaucht, wird eine neue Zeile und Spalte für die Quelle angelegt. Ist die Quelle der HELLO-Nachricht bekannt, werden die Sequenznummern verglichen. Falls der empfangene Wert größer ist als der gespeicherte, wird der generische Wert erneuert und das HELLO-Packet weitergeleitet. Sind beide Werte gleich, wird die TTL betrachtet. Ist die TTL des empfangenen Pakets kleiner als die gespeicherte, dann wird es weitergeleitet. In allen anderen Fällen, wird das Paket verworfen. Es ist zu beachten, dass HELLO-Nachrichten weitergeleitet werden, wenn sie über den besten Nachbarn empfangen wurden [13].

Mit Hilfe einer Decrease Routing Value Function (DRVF) und einer Increase Routing Value Function (IRVF) kann zusätzlich auf Topologieänderungen und andere Störungen reagiert werden. Die DRVF wird in regelmäßigen Intervallen, den Descrease Routing Value Interval (DRVI), angewandt und senkt die Linkqualität, sodass es nicht zu einem stetigen steigen der Linkqualität kommt. Die IRVF kommt beispielsweise bei dem Erhalt neuer Informationen über die Linkqualität zum Einsatz und steigert die Linkqualität. Im Gegensatz zur DRVF wird diese Funktion ereignisgesteuert und nicht in regelmäßigen Abständen ausgeführt. Eine Routingtabelle ist in Tabelle 1 dargestellt. Das zugehörige Netz ist das selbe wie in Abbildung 1.

2.3 MCFA

Der MCFA ist speziell für die Bedürfnisse in Sensornetzwerken entwickelt worden. Er basiert auf dem Kostenfeldkonzept, d.h. jeder Sensorknoten besitzt Informationen darüber, wie „teuer“ es ist, eine Nachricht zur Basisstation zu senden [28]. Es handelt sich um ein proaktives Protokoll, d.h. Routen sind vorhanden, bevor ein Paket gesendet werden soll [9].

Das Vorgehen teilt sich in zwei Phasen: die Initialisierung und die operative Phase [6]. Die Initialisierung beschreibt das Anlegen des Kostenfelds, d.h. alle Sensorknoten legen zunächst ein Feld mit dem Wert ∞ an, welches die Kosten zur Basis beschreibt. Die Basis sendet eine sogenannte Advertisement-Nachricht (ADV) mit dem Wert 0 aus. Unter der Verwendung des Backoff-Based Cost Field Establishment Algorithmus errechnen die Sensoren die Kosten, z.B. als Anzahl Hops, zur Basis dann wie folgt [28]:

Zuerst erhält ein Sensor eine ADV direkt von der Basis. Dann speichert er sich die Kosten C, die beim Senden dieser Nachricht entstanden sind, in seinem Kostenfeld und wartet die Zeitspanne $\alpha * C$. α ist eine Konstante die je nach Anforderungen und Infrastruktur des zu Grunde liegenden Netzes gewählt werden muss. Empfängt er in dieser Zeit keine relevanten Pakete, sendet er den ADV weiter und ändert aber den Wert von 0 auf C. Falls der Sensorknoten in der Wartezeit ein Paket von einem anderen Sensorknoten mit den Kosten C' empfängt, wird folgendes gepüft: Gilt C' zuzüglich der Kosten zwischen den Sensorknoten ist echt kleiner als das gespeicherte C, dann wird der Wert auf die angegebene Summe verändert. Eine weitere Wartezeit wird errechnet, in der auf neue Pakete gewartet wird [28].

Ist das Feld etabliert, kann nun das Routing beginnen. Ein Beispiel für ein etabliertes Kostenfeld ist in Abbildung 2 zu finden. Die Linkqualität der einzelnen Verbindungen findet sich auf den Pfeilen. Der Wert des Kostenfeldes für jeden Knoten ist neben dem Knoten angezeigt. Je höher der Wert auf den Pfeilen, desto größer sind die Kosten für die Übertragung. Niedrige Werte in den Kostenfeldern bedeuten geringere Übertragungskosten.

Beim Senden, werden zwei Felder an die Nachricht zugefügt: die minimalen und die bisher verbrauchten Kosten. Erhält ein Zwischenknoten eine solche Nachricht, prüft er, ob die bisher verbrauchten Kosten abzüglich der entstandenen Kosten für den letzten Hop immer noch größer oder gleich seiner minimalen Kosten zur Basis sind. Trifft dies zu, broadcastet der Knoten das Paket. Sonst wird es verworfen [28].

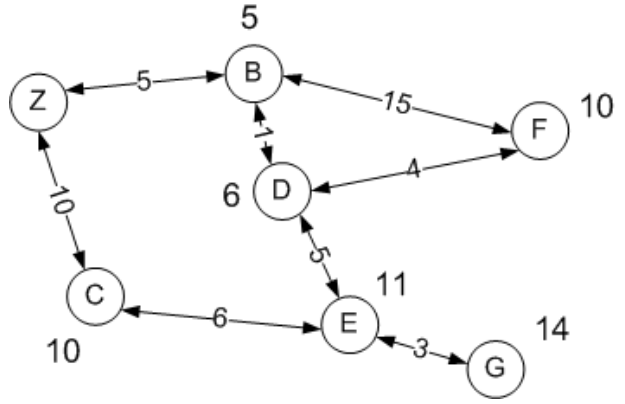


Abbildung 2: Ein etabliertes Kostenfeld für Z

Vorteilhaft bei dieser Methode ist, dass keine Routingtabellen verwendet werden müssen, die viel Speicher benötigen. Zudem sind keine ID's für die einzelnen Knoten notwendig. Durch die Verwendung der minimalen Kosten, ist auch ein optimaler Pfad zur Basis garantiert [21].

Dies zeigt jedoch einen Nachteil des Protokolls. Es kann nur für Wege von den einzelnen Sensorknoten zur Basisstation genutzt werden. Problematisch gestalten sich auch Knotenausfälle und Topologieänderungen, sowie Pfade mit gleichen Kosten [6].

3. ANGREIFER UND ANGRIFFE

Im nun folgenden Kapitel sollen zunächst unterschiedliche Angreifermodelle vorgestellt werden. Im Anschluss werden verschiedene Angriffe betrachtet.

3.1 Angreifermodelle

Man muss zunächst zwischen Insider- und Outsider-Attacken unterscheiden. Ein Outsider ist nicht Teil des Sensornetzes. Er kann aber als passiver Angreifer die Kommunikation der Knoten abhören (eavesdropping). Aktiv kann er Pakete abfangen, verändern oder neue Pakete in das Netz einschleusen und Sensorknoten physisch beschädigen [25].

Gelingt es dem Angreifer einen Knoten im Sensornetz unter seine Kontrolle zu bringen, oder einen neuen Knoten einzuschleusen, spricht man von einer Insider-Attacke. Neben den Möglichkeiten des Outsiders, kann der Angreifer nun auch an der Kommunikation im Netz teilnehmen und alle Funktionen eines normalen Knotens ausführen [25].

Eine weitere Unterscheidung ist nach den Ressourcen zu treffen. Karlof et. al. unterscheiden in [10] in mote-class und laptop-class Angreifer. Die erste Klasse beschreibt jene Angreifer, welche die gleichen Ressourcen (Rechenleistung, Speicher, etc.) zur Verfügung haben, wie auch die Sensorknoten. Ein Angreifer der laptop-class hingegen hat mehr Rechenleistung und kann somit auch mehr Angriffe auf das Sensornetz starten [10].

3.2 Direkte Angriffe auf das Routing

Die meisten Angriffe auf Sensornetze lassen sich nach [10] in eine der folgenden Kategorien einteilen:

- Manipulation der Routinginformationen
- Selektives Weiterleiten
- Manipulation mit HELLO-Paketen
- Manipulation von Infrastruktur- und Link-Layer Wissen
- Sinkhole-Attacken
- Sybil-Attacken
- Wurmlöcher

Die ersten vier Angriffsarten führen direkt zu Schäden innerhalb der Netzes. Die letzen drei wirken sich nicht direkt negativ auf das Netz aus, sondern müssen mit anderen schädigenden Aktionen verbunden werden. Ihre bloße Existenz stellt aber keine Einschränkung für das Netz dar.

Die Manipulation von Routinginformationen umfasst ändern, löschen, wiedereinspielen oder spoofen von Routingpaketen. Der Angreifer kann durch geschicktes Manipulieren Routingschleifen konstruieren, direkt den Paketfluss steuern, Routen festlegen, die Netzlast steigern oder das Netzwerk partitionieren [10].

Das selektive Weiterleiten unterscheidet sich in Grey- und Black-Holes, d.h. Nachrichten werden zufällig oder nach bestimmten Kriterien weitergeleitet oder alle Nachrichten werden gelöscht. Diese Attacke wird meist mit Angriffen wie Sinkholes, Sybil-Attacken und Wurmlöchern kombiniert [10].

HELLO-Pakete werden verwendet, um festzustellen, welche Knoten in Reichweite und somit Nachbarn sind. Dies stellt allerdings einen Angriffspunkt dar, da der Angreifer solche Pakete in das Netz fluten kann. Dies kann durch Wiedereinspielen (Replay) an gleicher oder anderer Stelle geschehen oder durch das komplette Fälschen der Pakete. Solche Angriffe bringen die gesamte Topologie durcheinander und stören zudem die Datenflusskontrolle [10].

Basieren Routingstrategien auf dem Wissen der darunterliegenden Ebenen kann dies ebenfalls von einem Angreifer ausgenutzt werden. Informationen wie Adressen der Knoten können vom Angreifer gefälscht werden und somit das Routing stören. Es kann auch versucht werden Knoten auszuschließen, indem sie als unerreichbar erscheinen oder nur eine schlechte Verbindung zu ihnen vorliegt. Solche Manipulationen können auch zur Realisierung anderer Angriffe wie selektivem Weiterleiten verwendet werden [10].

Eine Sinkhole-Attacke wird dazu verwendet den gesamten Netzverkehr über einen Knoten (Senke) lenken. Die Senke erscheint für die Sensorknoten besonders attraktiv, z.B. durch das Bereitstellen einer guten Verbindung zur Basisstation. Je nach Protokoll gibt die Senke verschiedene Eigenschaften vor, nach denen von den Sensorknoten der nächste Hop ausgewählt wird [10]. Dieser Angriff wird besonders gefährlich, wenn die Attacke mit anderen schadhaften Aktionen, wie Abhören oder selektiven Weiterleiten verbunden wird [14].

Bei einer Sybil-Attacke täuscht der Angreifer mehrere Identitäten mit nur einem physischen Knoten vor [3]. Der Angreifer kann entweder Identitäten generieren oder stehlen. Er hat auch die Möglichkeit seine Identitäten simultan oder nicht-simultan einzusetzen. Simultan bedeutet, dass es mehrere falsche Identitäten gleichzeitig im Netz gibt. Im nichtsimultanen Angriff wechseln die verschienden Identitäten. Eine Abwandlung der Sybil-Attacke ist die Replikation der gleichen Identität, der Angreifer gibt vor, an mehreren Stellen im Netz gleichzeitig zu sein. Eine solche Attacke kann sich bei Missbrauch nicht nur auf das Routing, sondern unter anderem auch auf verteilte Speichersysteme, Datenaggregation und Ressourcenverteilung auswirken [18].

Für eine Wurmloch-Attacke braucht der Angreifer zwei Knoten. Mit Hilfe des einen Knotens empfängt er Daten im Netz. Anschließend tunnelt er die gesammelten Daten an den zweiten Knoten. Besonders effektiv wird dieser Angriff, wenn der zweite Knoten in der Nähe der Basisstation liegt. Wie bereits erwähnt führt dieser Angriff nicht zu direkten Schäden, bringt aber den Angreifer in eine strategisch überlegene Position [8].

3.3 Indirekte Angriffe

Neben den bereits vorgestellten direkten Angriffe auf das Routing, die sich auf Schwachstellen in den einzelnen Routingprotokollen stützen, gibt es auch Angriffe auf Sensornetze, die nicht den Routingmechanismus an sich angreifen, ein Routing aber unmöglich machen, wie z.B. Jamming oder Denial-of-Service (DoS) Angriffe.

Beim Jamming sendet der Angreifer Radiosignale aus, welche die ausgesandeten Daten der Sensorknoten überlagern. Der Empfänger ist dann nicht mehr in der Lage, die übertragenen Informationen zu entnehmen. Der Angreifer benötigt hierfür ein stärkeres Signal als der Sender. Bei Sensornetzen stellt dies allerdings kein großes Hindernis dar [17].

Es wurden bereits verschiedene Ansätze wie das Frequency Hopping Spread Spectrum oder das Direct-Sequence--Spread-Spectrum entwickelt, die es dem Jammer erschweren das Signal zu stören und auch für Sensornetze einsetzbar sind. Zusätzlich gibt es auch für Sensornetze zugeschnittene Lösungen wie JAM oder JAID [17].

Mit Hilfe des Jammings kann eine weitere Bedrohung für das Routing in Sensornetzen realisiert werden: DoS-Angriffe. Bei einem solchen Angriff, ist das Netz nicht mehr in der Lage seine normale Aufgabe zu erledigen, sondern kommt zum Erliegen. Dies kann durch Angriffe wie Jamming, Fluten des Netzes mit Paketen oder auch Ausnutzen von Schwachstellen in Protokollen auf dem gesamten Protokollstack realisiert werden [27].

Um Protokolle resistent gegen DoS-Angriffe zu machen, werden oft kryptografische Verfahren und Authentifizierung verwendet. In Sensornetzen ist dieses Vorgehen jedoch nicht praktikabel [27].

4. ANGRIFFSSZENARIEN AUF AODV, SBR UND MCFA

Nachdem nun sowohl die Protokolle als auch die einzelnen Angriffe vorgestellt wurden, wird nun behandelt, welche Angriffe überhaupt in den Protokollen möglich sind und wie das Netz und Routingverhalten darauf reagieren.

4.1 Angriffe auf AODV

Ein wichtiger Angriffspunkt im AODV Protokoll sind die RREQ und RREP sowie RERR. Durch geschickte **Manipulation dieser Routinginformationen** lassen sich existierende Routen stören, einen manipulierten Knoten in eine Route einfügen oder auch zusätzliche Ressourcen verbrauchen [20]. Durch falsche RERR's lassen sich gezielt Ressourcen verbrauchen oder vorhandene Routen unterbrechen, da der Knoten mit einer scheinbar zerstörten Verbindung wieder eine neue Route suchen muss. Um einen Knoten in eine bestehende Route einzufügen, kann beispielsweise das Vorgehen in Abbildung 3 verwendet werden. Durch geschicktes Fälschen der Nachrichten wird die Route über den Angreifer umgeleitet. Das selektive Weiterleiten ist nun für den Angreifer möglich.

Weitere Möglichkeiten der Fälschung innerhalb der Nachrichten bietet die Broadcast-ID. Fälscht der Angreifer ein Paket eines Knotens und verwendet dabei eine sehr hohe Broadcast-ID, werden alle Pakete, die von dem betroffenen Knoten stammen ignoriert, bis dieser die gefälschte Sequenznummer erreicht hat. Auf diese Weise können Knoten isoliert, Ressourcen verbraucht und die Netzlast gesteigert werden. Erkennt ein Knoten einen zu großen Sprung in den ID's, so könnte er das Paket abweisen und eine entsprechende RERR an den betroffenen Knoten senden. Dies erhöht jedoch die Netzlast.

Durch das vorzeitige Senden eines RREP durch den An-

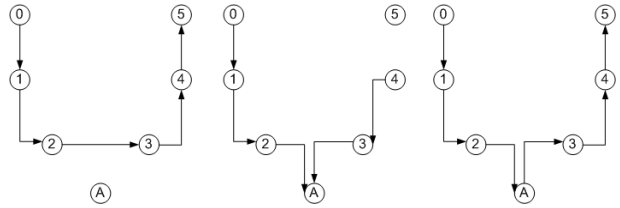


Abbildung 3: Ein Angriff auf AODV mittels Manipulation der Routinginformationen. Bild nachgezeichnet aus [20]

greifer kann zudem ein schwarzes Loch, ein Sonderfall des selektiven Weiterleitens, entstehen [16]. Dabei tarnt sich der Angreifer als Zielknoten des RREQ oder als Zwischenknoten mit einer aktiven Verbindung zum Ziel. Weitere Möglichkeiten für das **selektive Weiterleiten** innerhalb des AODV Protokolls bestehen durch die Anwendung der Wurmloch- oder Sinkhole Attacke.

Um ein **Wurmloch** zu erzeugen, wird der originale RREQ direkt zum Ziel getunnelt und dann dort an das Ziel weitergeleitet. Das Ziel antwortet nach Protokollvorschrift mit einem RREP, der dann über das Wurmloch zurück an den Initiator des RREQ getunnelt wird. Alle später eintreffenden RREP's, welche nicht über das Wurmloch laufen würden, werden dann vom Zielknoten verworfen, da dieser bereits eine Verbindung über das Wurmloch etabliert hat [8].

Da ein **Sinkhole-Angriff** auch über ein Wurmloch realisiert werden kann, ist AODV folglich auch anfällig für diesen Angriffstyp [19]. Darüber hinaus kann durch das Fälschen der Hop Counts eine Sinkhole Attacke durchgeführt werden. Hier erscheint ein Sensorknoten als besonders attraktiv für eine Route, wenn er einen geringen Hop Zähler aufweisen kann [4].

Das AODV Protokoll greift auf Identitäten zurück. Daher ist es prinzipiell möglich eine **Sybil-Attacke** zu starten. Durch das Vorspielen mehrerer Identitäten werden die Routingtabellen in den einzelnen Knoten vergrößert. Dies führt künstlich zu einem erhöhten Speicherverbrauch. Das Bilden von Alternativrouten z.B. bei der Erweiterung AODV-BR kann gestört werden, da eine scheinbare Alternativroute dann der Originalroute entspricht. Darüber hinaus wird bei Ausfall des Sybil-Knotens das Netz mit RERR-Nachrichten geflutet.

In [20] weisen Ning und Sun darauf hin, dass eine Verwendung von HELLO-Paketen bei der Nutzung von AODV nicht zwingend nötig ist. Daher ist auch ein **Fluten mit HELLO-Pakten vermeidbar**. Auch auf Infrastrukturwissen wird nicht zurückgegriffen und kann somit auch nicht als Grundlage für einen Angriff dienen.

4.2 Angriffe auf SBR

Das SBR Protokoll basiert auf dem Austausch von **HELLO-Paketen**, welche zum Austausch für **Routinginformationen** verwendet werden. Diese Informationen können allerdings vom Angreifer manipuliert werden. Durch das Abfangen und Ändern ausgewählter Nachrichten kann der Angreifer den Netzverkehr lenken. Indem er eine hohe Linkqualität vorgibt, werden Informationen an bestimmte Knoten gesandt. Eine niedrige Linkqualität führt dazu, dass ein Knoten nicht angesprochen wird.

Flutet ein Angreifer das Netz mit HELLO-Paketen, kann er den Energieverbrauch der Knoten erhöhen, da sie bei Erhalt von HELLO-Paketen ihre Routingtabellen anpassen müssen. Durch die Anwendung der DRVF und IRVF bei Erhalt der HELLO's ist der Knoten gezwungen unnötig Rechenleistung und Energie zu verbrauchen. Dieser Mechanismus, der eigentlich für eine hohe Adaption des Netzes genutzt werden soll, kann in diesem Fall das Netz erheblich schwächen.

Durch das Vorgeben einer hohen Linkqualität kann der Angreifer ein **Sinkhole** erzeugen. Er hat die Möglichkeit, seine eigenen Knoten mit hoher Linkqualität anzupreisen, sodass diese zu einer Senke werden. Er kann auch Nachrichten spoofen und somit andere Knoten im Netz zur Senke machen. Diese Knoten werden dann besonders mit Netzverkehr belastet. Dies kann zu einem Ausfall der betroffenen Knoten führen oder den Netzverkehr erheblich verlangsamen.

Eine hohe Linkqualität kann auch durch ein **Wurmloch** verursacht werden. Wird die Linkqualität beispielsweise durch die Nähe zur Basis bestimmt, werden die beiden Angreiferknoten im normalen Protokolleinsatz zur bevorzugten Route.

Durch Verwendung vom Senken oder Wurmlöchern bringt sich der Angreifer in eine sehr starke Position. Er ist nach erfolgreichem Angriff in der Lage, **selektiv Nachrichten weiterzuleiten** oder auch ein schwarzes Loch zu erzeugen. Er muss aber darauf achten, dass trotz der Verwendung der IRVF und DRVF seine Position bestehen bleibt. Daher muss er im gesamten laufenden Betrieb Maßnahmen zum Erhalt seiner Position durchführen.

Das Verändern der Routinginformationen ist vor allem bei der Sequenznummer von Bedeutung. Verwendet der Angreifer eine hohe Sequenznummer, muss der betroffene Knoten auch hier sehr viele Nachrichten versenden, bis die anderen Knoten wieder auf seine Nachrichten reagieren. Die Ressourcen des Knotens werden so verbraucht und der Angreifer ist in der Lage, bestimmte Knoten für eine gewisse Zeit aus dem Netz zu isolieren. Eine Möglichkeit wäre der Vergleich der aktuellen Sequenznummer mit der letzten erhaltenen. Ist der Unterschied zu groß, wird das Paket nicht angenommen. Eine Möglichkeit, dies dem Sender mitzuteilen besteht allerdings nicht.

Wenn bei der Berechnung der Linkqualität auf Informationen auf den unteren Schichten zurückgegriffen wird, stellt auch dieses Wissen einen Angriffspunkt dar. Eine Ausnutzung ist dann abhängig von der Nutzung der Informationen sowie der DRVF und IRVF.

Auch ein **Sybil-Attacke** ist potentiell möglich. Die Routingtabellen der Knoten beziehen sich auf die ID's anderer Knoten, d.h. die Identitätsinformation ist essentiell wichtig für das Funktionieren des Protokolls. Zusätzliche Identitäten verlängern die Routingtabellen und erhöhen so den Speicherbedarf und Rechenaufwand für die einzelnen Knoten, um diese Tabellen zu warten. Insgesamt wird so auch

der Energieverbrauch erhöht. Zudem würden real existierende Knoten aus der Tabelle verdrängt werden und somit für den Knoten nicht mehr erreichbar sein.

4.3 Angriffe auf MCFA

Die Möglichkeit **Routinginformationen zu manipulieren** ist dem Angreifer nur in der Initialisierungsphase möglich, indem er ADV-Nachrichten löscht, ändert, spooft oder wiedereinspielt. Das Wiedereinspielen beeinflusst das Routing allerdings nicht, da die Kosten nur geändert werden, wenn eine Verbesserung vorliegt. Durch das Löschen, Ändern oder Spoofen von ADV's ist es möglich, dass die minimalen Kosten für den Nachrichtenversand zu hoch kalkuliert werden und im späteren Produktivbetrieb auch Sensorknoten die Nachrichten weiterleiten, die nicht auf dem optimalen Pfad liegen. Neben einer erhöhten Netzlast hat dies auch Routingschleifen zur Folge.

Karkof et. al. beschreiben in [10] das auch eine Attacke über das Fluten des Netzes mittels **HELLO-Paketen** möglich ist. Mit einem Laptop werden Pakete in der Initialisierungsphase versandt, die Kosten von 0 zur Basis versprechen. Nun müssen die Kosten der Nachricht für jeden Knoten nur niedriger gehalten werden als dessen bisherige Kosten. Werden dann Nachrichten versandt, wird kein anderer Knoten als der Angreifer selbst, die Nachrichten weiterleiten können [10].

Befindet sich der Angreifer innerhalb des Netzes auf einem minimalen Pfad, von der Quelle zur Basis kann er auch **Nachrichten selektiv weiterleiten** oder die Kommunikation blockieren. Um dies zu erreichen könnte der Angreifer vorher eine **Sinkhole Attacke** benutzen. Hierfür ist ein Knoten im Netz ausreichend, welcher Kosten von 0 zur Basisstation propagiert. Der zugrundeliegende Algorithmus für die Routenerstellung versucht die Kosten zu minimieren und wird sich für den Knoten mit Kosten von 0 zur Basis entscheiden. Die Sinkhole Attacke muss aber schon in der Initialisierungsphase durchgeführt werden um im Produktivbetrieb genutzt werden zu können [10].

Eine weitere Möglichkeit das Netz zu stören besteht in der Anwendung eines **Wurmlochs**. Der Angreifer muss in diesem Fall einen leistungsstärkeren Knoten aufweisen, wie beispielsweise einen Laptop [10]. Auch in diesem Szenario muss der Angriff in der Initialisierungsphase geschehen. Er wird ebenfalls durch das Propagieren von Kosten von 0 zur Basis umgesetzt.

Für die einzelnen Knoten sind keine ID's nötig sind, folglich ist eine **Sybil-Attacke nicht anwendbar**. Da nicht auf Link Layer- oder Infrastrukturwissen zurückgegriffen wird, führt auch hier eine Manipulation nicht zu einer erfolgreichen Störung des Netzes.

5. GEGENMAßNAHMEN

Die Nutzung von Kryptografie und global verteilten Schlüsseln ist eine Möglichkeit Gegenmaßnahmen zu ergreifen [10]. Durch die geringe Leistungsfähigkeit sind diese Ansätze im Bereich Sensornetze jedoch nicht vorteilhaft. Im Folgenden sollen nun einige weiterführende Ansätze besprochen werden. Zu den vorgestellten Angriffen existieren einige Frameworks zur Behandlung der Probleme. Hinzu kommen Ansätze für Protokolle, die speziell für Sensornetzwerke mit Hinblick auf Sicherheit entwickelt wurden. Auf Grund der Menge an vorhandenen Lösungsansätzen wird hier nur eine kleine Auswahl der wichtigsten Ansätze vorgestellt.

5.1 Frameworks für spezielle Angriffe

Eine Möglichkeit um Wurmlöcher zu verhindern sind sog. Packet leashes, d.h. zusätzliche Informationen, die die Sendedistanz einschränken [8]. Man unterscheidet zwischen temporalen, hier wird die Sendezeit angefügt, und geografischen, zusätzlich wird die eigene Position übertragen, Leashes. Die emfangenen Werte werden mit den eigenen Daten verglichen und bestimmt, ob sich das Paket zu weit bewegt hat [7]. Einen weiteren Ansatz bietet das WOMEROS Framework von Vu et. al. in [26]. In der Verdachtsphase misst jeder Knoten die Round Trip Time (RTT) zu allen seinen Nachbarn. Neben einer erhöhten RTT bringt auch die Auswertung der Nachbarn Hinweise auf ein Wurmloch. In der Bestätigungsphase wird versucht durch bestimmte Tests den Verdacht auf ein Wurmloch zu bestätigen [26]. Leider werden keine Maßnahmen genannt, das Wurmloch aus dem Netz zu entfernen.

Neben Wurmlöchern wurden auch Sinkhole-Angriffe besonders untersucht. In [19] wird ein solcher Ansatz vorgestellt. Es wird davon ausgegangen, dass eine häufige Anwendung des Sinkholes das selektive Weiterleiten von Daten ist. Daher kann das Fehlen von Daten aus immer wieder den gleichen Bereichen auf ein Sinkhole hindeuten. Mit statistischen Methoden können solche Inkonsistenzen aufgedeckt werden. Durch die Analyse der Routing-Mustern kann der Angreifer identifiziert und isoliert werden. Das genaue Verfahren zur Musteranalyse und das Adaptieren des Vorgehensmusters sind in [19] eingehend beschrieben.

Auch Maßnahmen zur Eindämmung der Sybil-Attacke wurden eingehend in [18] untersucht. Das Radio Resource Testing geht davon aus, dass jeder Knoten nur eine Antenne hat, mit der er auf einem Kanal entweder senden oder empfangen kann. Spricht man einen benachbarten Knoten auf einer bestimmten Frequenz an und dieser antwortet nicht, deutet dies auf einen Sybil-Angriff hin. Das genaue Verfahren wird in [18] erläutert. Ein weiteres Verfahren ist das Random Key Predistribution. Hierbei werden Schlüssel verwendet, die mit der Knoten-Identität verbunden werden. Jeder Knoten kann dann die Schlüssel validieren. Ein Generieren von gültigen Identitäten ist nicht mehr möglich. [18]

5.2 Sicherheitsprotokolle

Ein speziell für Sensornetzwerke entwickeltes Sicherheitssystem ist SPINS (Security Protocols for Sensor Networks) [23]. Es besteht aus zwei unabhängigen Protokollen. Das Sensor Network Encryption Protocol (SNEP) bietet Vertraulichkeit, wechselseitige Datenauthentifikation, Integrität und Datenaktualität (freshness). μTESLA wird für die Authentifikation von Datenbroadcasts verwendet.

Nötig ist hier allerdings ein vorab festgelegter Schlüssel zwischen jedem Knoten und der Basis [23]. Bezüglich Angriffe auf das Routing bietet SNEP einen Schutz gegen Replay Angriffe und Manipulation von Routinginformationen. Problematisch ist jedoch die durch kryptografische Funktionen benötigte zusätzliche Rechenleistung [23]. Der zweite Sicherheitsblock besteht aus einer Adaptierung des TESLA Pro-

tokolls μTESLA. Die vorgenommenen Anpassungen sollen Einschränkungen im Sensorbereich verglichen mit Ad-hoc Netzwerken ausgleichen [23].

Dieses Verfahren kann nun auf Ad-hoc Routing Strategien angewandt werden. Die Routingnachrichten können so authentifiziert werden, was eine Manipulation erschwert [23]. Wurmlöcher, Sinkholes und Sybil-Attacken sind weiterhin möglich, und somit auch das selektive Weiterleiten. Das Fluten mit HELLO-Paketen wird erschwert, wenn der Angreifer Pakete abfangen und verändern muss. Neue, gültige Pakete zu generieren ist nicht möglich, solange er die Schlüsselkette nicht kennt.

5.3 Intrusion Detection Lösungen

Der Ansatz von Eik et. al. in [4] basiert auf der Einbruchserkennung, d.h. es werden keine Maßnahmen zur Vermeidung von Angriffen getroffen, sondern lediglich Angriffe erkannt. Ziel ist es hier, nicht nur einen bestimmten Angriffstyp zu erkennen, sondern alle Angriffsmuster zu erfassen.

Jeder einzelne Knoten benötigt sein eigenes Intrusion Detection System (IDS). Es werden unabhängig voneinander und lokal traffic und nicht-traffic bezogene Daten gesammelt, die einen Eigenschaftsvektor bilen. Da jeder Knoten zusätzliche Daten verwalten muss, steigt der Speicherbedarf und durch die zusätzlich benötigte Rechenleistung wird auch mehr Energie benötigt. Alternativ können sog. monitoring nodes verwendet werden, deren einizige Aufgabe die Netzüberwachung ist [4]. Ein Einbruch wird dann erkannt, wenn das aktuelle Netzverhalten nicht mit dem normalen, vorher erlernten Netzverhalten übereinstimmt [4].

Ein weiterer ganzheitlicher Ansatz ist das DICAS (Detection, Diagnosis and Isolation of Control Attacks in Sensor Networks) Framework. Es soll u.a. Wurmlöcher, Sinkholes, Sybil-Attacken und das Fluten mit HELLO-Paketen erkennen und betroffene Knoten isolieren [11]. Ein Knoten sucht zunächst alle seine Nachbarn. Dann beobachten die Knoten die Kommunikation der anderen Knoten, wenn sie beide Kommunikationspartner als Nachbarn identifiziert haben. Werden verdächtige Pakete ausgetauscht, wird ein Zähler für die jeweilige Kommunikation erhöht. Beim Überschreiten eines Schwellwertes, wird von einem Angriff ausgegangen und ein Alarm ausgelöst [11].

6. VERWANDTE ARBEITEN

Karlof et. al. lieferten bereits in [10] einen Überblick über Routingangriffe. In dieser Arbeit wurden allerdings drei andere Protokolle (AODV, SBR und MCFA) fokussiert, wobei AODV und SBR nicht in den Betrachtungen von [10] liegen. Zudem wurden in diesem Paper auch die möglichen Gegenmaßnahmen genauer betrachtet.

In [1] wird einen sehr genauer Überblick über bestehende Protokolle für den Einsatz in Sensornetzen geboten. Es werden allerdings keine möglichen Angriffen und zugehörige Gegenmaßnahmen besprochen. Auch in [9] legen die Autoren den Fokus auf die Betrachtung und den Vergleich der Routingprotokolle für Sensornetzwerke.

In den Quellen [2] und [16] wird versucht speziell das AODV Protokoll gegen Angriffe abzusichern. Diese Methoden sind allerdings nicht zwangsläufig auf andere Protokolle übertragbar und bieten so keine umfassende Lösung. Umfassende Lösungen werden in [4] und [11] beschrieben. Diese Lösungsschemata sollen einen allgemeinen Schutz gegen Angriffe bieten. Es wird jedoch keine genaue Implementierung dargestellt.

Die Lösungen aus [7] und [26] bieten wiederum nur einen Schutz vor einem bestimmten Angriffstyp und keine umfassende Betrachtung der Probleme. [23] bezieht sich nicht direkt auf mögliche Probleme durch Angriffe auf das Routing, sondern stellt allgemein ein Sicherheitsprotokoll vor.

7. ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit wurde dargelegt, welche Angriffe auf ausgewählte Protokolle möglich sind. Dabei hat sich gezeigt, dass kein Protokoll so konzipiert wurde, dass es allen vorgestellten Angriffsszenarien standhalten kann.

Zu AODV gibt es Verbesserungsvorschläge um konkrete Angriffe zu unterbinden. Auf SBR oder MCFA zugeschnittene Lösungen gibt es aber nicht. Allgemeine Sicherheitsprotokolle für Sensornetze können auch nur teilweise Angriffe unterbinden. In Kombination mit Routingprotokollen wirken sie vor allem der Manipulation von Routinginformationen entgegen. Dies ist zwar schon ein Schritt in die richtige Richtung, löst aber das Problem nicht vollständig.

Umfassendere Lösungen wurden ebenfalls betrachtet, doch sind mit solchen universellen Lösungen neue Probleme verbunden. Erhöhter Speicherbedarf und zusätzliche Rechenleistung, die den Energiebedarf erhöht, stellen neue Forderungen an die Hardware von Sensorknoten und das Energiemanagement.

Diese Ansätze sind zwar vielversprechend, müssen aber noch auf die Bedürfnisse von Sensorknoten angepasst werden. Eine weitere Möglichkeit ist die Entwicklung spezieller Knoten, die im Netz nur die Aufgabe übernehmen, nach Angreifern zu suchen, zu finden und dann aus dem Netz zu entfernen. Solche zusätzlichen Komponenten treiben jedoch auch die Kosten des Netzes in die Höhe, was nicht bei allen Einsatzgebieten akzeptabel ist.

Es konnte keine Lösung gefunden werden, die alle Probleme abdeckt ohne dadurch neue Herausforderungen zu schaffen. Dies zeigt, dass auch im Bereich der Sensornetzwerke noch ein erhöhter Forschungsbedarf besteht.

8. LITERATUR

[1] J. N. Al-karaki, T. H. University, A. E. Kamal, and I. S. University. Routing techniques in wireless sensor networks: A survey. *IEEE Wireless Communications Magazine*, 11(6):6 – 28, December 2004.

[2] S. Bhargava and D. P. Agrawal. Security enhancements in AODV protocol for wireless ad hoc networks. In *Vehicular Technology Conference*, pages 7–11, Atlantic City, New York, October 2001.

[3] J. Douceur. The sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, pages 251–260, 2002.

[4] C. Eik, L. Mun, Y. Ng, C. Leckie, and

M. Palaniswami. Intrusion detection for routing attacks in sensor networks. *International Journal of Distributed Sensor Networks*, 2(4):313 – 332, December 2006.

[5] Y. Gadallah. A comparative study of routing strategies for wireless sensor networks: Are MANET protocols good fit? In *Ad-Hoc, Mobile, and Wireless Networks*, pages 5–18, 2006.

[6] W. D. Henderson and S. Tron. Verification of the minimum cost forwarding protocol for wireless sensor networks. *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*, pages 194 – 201, May 2007.

[7] Y. Hu, A. Perrig, and D. Johnson. Packet leashes: A defense against wormhole attacks in wireless networks. In *INFOCOM*, 2003.

[8] Y. C. Hu, A. Perrig, and D. B. Johnson. Wormhole attacks in wireless networks. *IEEE Journal on Selected Areas in Communications (JSAC), pp*, 24(2):370 – 380, February 2006.

[9] Q. Jiang and D. Manivannan. Routing protocols for sensor networks. *Consumer Communications and Networking Conference, 2004. CCNC 2004. First IEEE*, pages 93 – 98, April 2004.

[10] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. In *First IEEE International Workshop on Sensor Network Protocols and Applications, SPNA*, 2003.

[11] I. Khalil, S. Bagchi, and C. Nita-Rotaru. DICAS: Detection, diagnosis and isolation of control attacks in sensor networks. In *IEEE Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm)*, 2005.

[12] A. Klein. Statistic-based routing SBR. Experimental 453, University of Wuerzburg, October 2008.

[13] A. Klein and P. Tran-Gia. Handover in sensor networks using statistic-based routing. *6th ITG Wireless Sensor Networks*, July 2007.

[14] I. Krontiris, T. Giannetsos, and T. Dimitriou. Launching a sinkhole attack in wireless sensor networks; the intruder side. In *WIMOB '08: Proceedings of the 2008 IEEE International Conference on Wireless & Mobile Computing, Networking & Communication*, pages 526–531, Washington, DC, USA, 2008. IEEE Computer Society.

[15] S. J. Lee and M. Gerla. AODV-BR: Backup routing in ad hoc networks. In *Proceedings of IEEE WCNC 2000, Chicago IL*, 2000.

[16] N. Mistry, D. C. Jinwala, and M. Zaveri. Improving AODV protocol against blackhole attacks. In *Proccedings of International MultiConference of Engineers and Computer Scientist, IMECS*, volume 2, Hong Kong, March 2010.

[17] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos, and G. Pantziou. A survey on jamming attacks and countermeasures in WSN.

[18] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: analysis & defenses. In *Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on*, pages 259–268, 2004.

[19] E. C. H. Ngai, J. Liu, and M. R. Lyu. On the intruder detection for sinkhole attack in wireless sensor networks. In *Communications, 2006 IEEE International Conference on*, volume 8, pages 3383–3389, 2006.

[20] P. Ning and S. K. How. How to misuse AODV: a case study of insider attacks against mobile ad-hoc routing protocols. Technical report, Computer Science Departement, North Carolina State University, March 2003.

[21] T. Padmavthy, G. Divya, and T. Jayashree. Extending network lifetime in wireless sensor networks using modified minimum cost forwarding protocol - MMCFP. *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on*, pages 1 – 4, October 2009.

[22] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computer Systems and Applications*, pages 90–100, 1999.

[23] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of the Annual International Conference on Mobile Computing and Networks (MOBICOM) 2001*, pages 189–199, 2001.

[24] E. M. Royer and C. E. Perkins. An implementation study of the AODV routing protocol. In *IEEE WCNC*, 2000.

[25] E. Shi and A. Perrig. Designing secure sensor networks. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 11(6):38–43, 2004.

[26] H. Vu, A. Kulkarni, K. Sarac, and N. Mittal. WORMEROS: A new framework for defending against wormhole attacks on wireless ad hoc networks.

[27] A. Wood and J. Stankovic. Denial of service in sensor networks. *IEEE Comp*, 35(10):54–62, 2002 2002.

[28] F. Ye, A. Chen, S. Lu, and L. Zhang. A scalable solution to minimum cost forwarding in large sensor networks. In *Proc. of the IEEE Tenth International Conference on Computer Communications and Networks*, pages 304–309, 2001.

[29] Z. Zhang, H. Zhou, and J. Gao. Scrutinizing performance of ad hoc routing protocols on wireless sensor networks. *Intelligent Information and Database Systems, Asian Conference on*, 0:459–464, 2009.

Routing in Sensornetzen II

Philipp Tölke
Betreuer: Alexander Klein
Seminar Sensorknoten: Betrieb, Netze und Anwendungen Sommersemester 2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: toelke@in.tum.de

KURZFASSUNG

Das Routing in drahtlosen Sensornetzen, die zumeist aus vielen kleinen Knoten bestehen, stellt große Anforderungen. Die Anwendung von klassischen Routingprotokollen ist auf Grund der Ressourcenknappheit nicht möglich. Diese Ausarbeitung untersucht einige Routingkonzepte anhand der Betrachtung von konkreten Routingprotokollen.

Schlüsselworte

Routing, Survey, drahtlose Sensornetze

1. EINLEITUNG

In Netzwerken aus kleinen Sensorknoten, die zunehmend in der Automatisierung, in Industrie, Haushalten und zur Umweltüberwachung eingesetzt werden, werden Daten von den Sensoren per Funk zur Auswertung transportiert.

Speicher, Rechenleistung und zur Verfügung stehende Energie ist in diesen Sensorknoten sehr beschränkt. Deshalb sind die aus dem Internet bekannten Routingprotokolle, wie zum Beispiel das Border Gateway Protocol (Bgp), dass im Internet das Inter-Autonomous-System-Routing verwaltet, nicht einsetzbar. Auch eine Eintragung der Routen von Hand verbietet sich bei Sensornetzen von 100 oder mehr Knoten. In manchen Anwendungen wird zusätzlich gefordert, dass sich die Knoten frei bewegen können, Routinginformationen müssen daher ständig erneuert werden.

Diese Seminararbeit soll einen Überblick darüber geben, wie die Anforderungen des Routings erfüllt werden können.

Zunächst wird ein Überblick über verschiedene Konzepte des Routings in Sensornetzen gegeben. Zu diesen Konzepten werden dann exemplarisch Protokolle vorgestellt, die zum Abschluss gegenübergestellt werden um eine Auswahl für einen gegebenen Anwendungsfall treffen zu können.

2. PROAKTIVE UND REAKTIVE ROUTING-PROTOKOLLE

Ein Routingprotokoll, das Routen bestimmt bevor sie wirklich gebraucht werden, nennt man „proaktiv“[6]. Ein Beispiel für ein proaktives Protokoll ist das Collection Tree Protocol (Ctp).

Erstellt ein Protokoll erst dann Routen, wenn ein Paket gesendet werden soll, nennt man es „reaktiv“. Ad-Hoc On-Demand Distance Vector (Aodv) ist ein reaktives Protokoll.

Reaktive Protokolle haben eine größere Latenz für das erste Paket einer „Verbindung“, verbrauchen aber dafür in der Regel weniger Energie Routinginformationen aufzubauen.

3. METRIKEN

Um Routingentscheidungen treffen zu können, müssen Knoten in einem Sensornetz „wissen“, wie weit sie vom Empfänger einer Nachricht entfernt sind, und welcher direkt erreichbare Knoten näher am Empfänger ist.

Die einfachste solcher Abstandsmetriken ist der Hop-Count. Zwei Knoten, die direkt kommunizieren können, haben den Abstand 1. Können zwei Knoten über genau einen Mittelsmann kommunizieren, haben sie den Abstand 2.

In die Routingmetrik können auch Umwelteinflüsse, wie beispielsweise die zur Verfügung stehende Energie einfließen: Knoten, die nur noch über wenig Energie verfügen, geben eine schlechtere Verbindung zu allen Empfängern an, als sie eigentlich haben. Das macht es unwahrscheinlicher, dass sie Pakete weiterleiten müssen und noch mehr Energie verbrauchen.

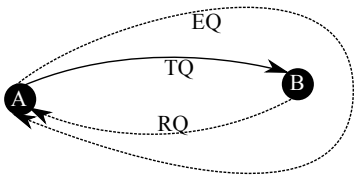


Abbildung 1: Batman: Die drei Werte TQ, RQ und EQ. EQ und RQ (gestrichelt) können direkt gemessen werden.

Im Protokoll Batman („Better Aproach To Mobile Ad-Hoch Networking“) Version IV wird eine Metrik vorgeschlagen, bei dem jeder Knoten ausrechnet, wie viele seiner Pakete auf dem Weg zu direkten Nachbarn verloren gehen. Dazu sendet jeder Knoten regelmäßig einen Broadcast um seine Position bekannt zu geben. Diese Broadcasts werden weitergeleitet und so im Netz verteilt. Weiterhin sind sie durchnummeriert, so dass ein Knoten erkennen kann, wenn er Broadcasts seiner Nachbarn nicht bekommen hat. Er verfügt damit über zwei Messwerte: die Qualität der Verbindung von seinen Nachbarn RQ („Receiving Quality“) und er kann messen wie viele seiner eigenen Broadcasts der Nachbar seinerseits weitergeleitet hat, EQ („Echo Quality“). Die gesuchte Metrik ist die TQ („Transmit Quality“). Sie beschreibt wie viele Pakete beim Nachbarn ankommen. Es gilt offensichtlich:

$$EQ = TQ \cdot RQ, \tag{1}$$

der Anteil der empfangenen wiederholten Broadcasts ist gleich der Anzahl der vom Nachbarn empfangenen Broadcasts multipliziert mit der Wahrscheinlichkeit überhaupt ein vom Nachbarn gesendetes Paket zu empfangen.

Durch Umformung lässt sich die gesuchte Sendequalität berechnen:

$$TQ = \frac{EQ}{RQ} \tag{2}$$

TQ kann als Metrik dafür eingesetzt werden, wie zuverlässig die Verbindung zum Nachbarn ist.[5]

4. PROTOKOLLE MIT WENIGEN EMPFÄNGERN

In vielen Anwendungen ist es ausreichend, wenn nicht jeder Knoten mit jedem anderen Knoten beliebig kommunizieren kann, sondern wenn nur bestimmte Knoten Daten empfangen können. So ist es beispielsweise zur Temperaturüberwachung eines Raumes nicht nötig, dass die Knoten untereinander Daten austauschen, sondern nur, dass die Temperaturdaten an bestimmte Steuerknoten gesendet werden. Diese Knoten nennt man „Senken“.

4.1 Collection Tree Protocol

Das „Collection Tree Protocoll“ (CTP) ermöglicht es einen logischen Routing-Baum zu erstellen, so dass jeder Knoten über einen möglichst kurzen Pfad Daten zu einer Senke schicken kann[2]. Es wurde als TinyOs Enhancement Proposal (TEP) entwickelt.

Mit welcher Metrik die Pfadlänge gemessen wird, hängt vom Anwendungsfall ab. Im ursprünglichen Vorschlag (TEP-123[1]) wird die erwartete Anzahl der Sendewiederholungen auf Schicht 2 herangezogen.

Das Protokoll stellt an die darunter liegende Schicht 2 die Anforderung, dass sich effizient Broadcasts an alle Knoten in der Nähe schicken lassen, also an alle Knoten, die eine Nachricht direkt empfangen können. Weiterhin fordert das Protokoll, dass Unicast-Nachrichten auf Schicht 2 bestätigt oder nötigenfalls wiederholt werden.

Jeder Knoten im Netzwerk sendet regelmäßig eine Nachricht („Beacon“) aus, in der er seinen Abstand zur nächsten Senke bekannt gibt. Erhält ein Knoten einen Beacon mit dem ein geringerer Abstand zur Senke bekannt gegeben wird, als dem Knoten bisher bekannt ist, wird er von jetzt an Nachrichten an die Senke über den Absender des Beacons senden.

Diese Beacons werden nach dem Trickle-Algorithmus[4] versandt. Dieser sorgt dafür, dass im stabilen Betrieb nur wenig Beacons versandt werden müssen, indem das Sendeintervall schrittweise vergrößert wird. Es wird in folgenden Fällen auf den kleinsten Wert zurückgesetzt:

Der Knoten bekommt ein Paket von einem Knoten, dessen Pfadlänge zur Senke nicht größer ist als die eigene zur Weiterleitung.

Dies kann passieren, wenn Nachbarn falsche oder veraltete Routinginformationen haben.

Die Pfadlänge verringert sich durch einen empfangenen Beacon erheblich.

Diese Änderung wird sofort allen Nachbarn weitergegeben, da es eine Verbesserung der Routen sein kann.

Ein Paket mit gesetztem P-Flag wird empfangen.

Ein Knoten, der aus irgendeinem Grund keine Routinginformationen besitzt wird zunächst Pakete mit gesetztem „Pull“-Flag senden, damit Nachbarn ihm ihre Pfadkosten mitteilen können.

4.2 Minimal Cost Forwarding Algorithm

Der MCFA ermöglicht es, wie das CTP Daten an einige Senken zu schicken. Anders als das CTP werden Datenpakete allerdings nicht direkt zum nächsten Hop gesendet, sondern per Broadcast an alle Knoten in der Nähe weitergegeben. Jeder Hop trifft dann lokal die Entscheidung, ob dieses Paket weitergeleitet werden muss.

Jeder Knoten kennt seine Entfernung zur nächsten Senke. Der Absender eines Paketes trägt seine Entfernung als „Kontigent“ in das Paket ein und sendet es per Broadcast ab.

Empfängt ein Knoten ein Datenpaket, überprüft er, ob das verbleibende Kontingent des Pakets größer oder gleich seiner Entfernung zur Senke ist. Ist es kleiner, verwirft er das Paket, er liegt nicht auf dem kürzesten Pfad vom Absender zur Senke. Andernfalls sendet er das Paket mit um die Weiterleitungskosten verringertem Kontigent weiter. Wie hoch die Weiterleitungskosten sind, hängt von der Metrik ab. Wird die Hopzahl als Metrik verwendet, verringert jeder Knoten das Kontingent eines Paketes jeweils um eins.

Anschaulich hat jeder Knoten eine Höhe, die den Abstand zur Senke angibt. Pakete laufen immer von hohen Knoten bergab in Richtung der Senke[7].

4.3 Kostenfindung im MCFA

Um die Höhe der einzelnen Knoten zu ermitteln, werden zu Beginn Broadcasts versendet. Die Senken schicken ein Broadcast-Paket in dem sie die Höhe 0 angeben. Knoten, die diese Pakete empfangen setzen ihre eigenen Kosten auf einen, je nach Metrik, höheren Wert und schicken ihrerseits einen Broadcast ab. Damit nicht alle Knoten kurz hintereinander einen Broadcast schicken müssen, der unter Umständen noch nicht die minimalen Kosten des Knotens enthält (siehe Abbildung 2), wartet der Algorithmus vor Versenden des Broadcasts für eine Zeit, die proportional zur Höhe des Knotens ist. Dadurch ist gewährleistet, dass Knoten, über die eine kürzere Verbindung zur Senke möglich ist, früher ihre Broadcasts abschicken. Es schickt also jeder Knoten nur sehr wenige (im Idealfall einen) Broadcasts ab, bis alle Knoten ihre Höhe kennen. [7]

Da nach einmal erfolgtem Broadcast ein Knoten kein weiteres Mal seine Position bekannt gibt, kann der MCFA nicht

auf Veränderungen des Netzes zu reagieren. Für Anwendungen bei denen Mobilität der Knoten wichtig sind, muss also ein modifizierter Algorithmus verwendet werden, und beispielsweise regelmäßig ein Broadcast geschickt werden.

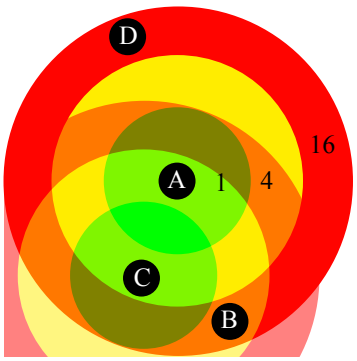


Abbildung 2: MCFA: der Knoten A sendet einen Broadcast aus. Als Metrik wird die Empfangsstärke angenommen, die quadratisch mit der Entfernung abnimmt. Knoten C nimmt also Entfernung 4 an, Knoten B und D die Entfernung 16. Würde Knoten B jetzt ohne zu warten einen Broadcast senden, müsste er nachdem Knoten C seinen Broadcast sendet erneut einen senden: Er hätte dann die Kosten 8.

Unverändert ist dieses Protokoll nur für die Konfiguration von statischen Netzen, zum Beispiel für die Umweltüberwachung, nutzbar.

5. PROTOKOLLE MIT BELIEBIGEN EMPFÄNGERN

Soll es den Knoten möglich sein, beliebig untereinander zu kommunizieren, muss jeder Knoten einen Eintrag in der Routingtabelle jedes andern Knoten haben.

5.1 AODV

AODV ist ein reaktives Protokoll. Die Routinginformationen werden erst dann erstellt, wenn ein Datenpaket gesendet werden soll.

Ein Knoten, der ein Paket zu einem andern Knoten schicken möchte, sendet zunächste eine RREQ-Nachricht („Route Request“) per Broadcast ab. Andere Knoten geben diese Nachricht weiter, bis sie einen Knoten erreicht, der einen gültigen Eintrag für das Ziel hat. Dieser Knoten kann natürlich auch das Ziel selbst sein. Er sendet jetzt eine RREP-Nachricht („Route Reply“), die in der Regel dengleichen Weg durch das Netzwerk zurück nimmt, den die RREQ-Nachricht genommen hat. Dazu schickt jeder Knoten die RREQ-Nachricht mit seiner Adresse als Absender ab und speichert für einige Zeit, dass er diese Nachricht gesehen hat. Bekommt er eine RREP, der zu einem RREQ passt, den er weitergeleitet hat, speichert er den Absender des RREP als nächsten Hop zum angefragten Knoten und sendet seinerseits einen RREP ab.

Damit ein RREQ nicht im kompletten Netz verteilt wird, setzt der anfragende Knoten im erstem Broadcast die TTL (Time To Live, wird bei jedem Hop dekrementiert. Ist sie 0 wird das Paket nicht mehr weitergeleitet) sehr niedrig, so dass das Paket früh verworfen wird. Trifft nach einiger Zeit kein RREP ein, sendet der Knoten erneut einen Broadcast mit erhöhter TTL. Die TTL wird so lange erhöht, bis innerhalb eines Timeouts eine RREP eintrifft. In Abbildung 3 ist beispielhaft der Weg von RREQ-Nachrichten von Knoten A zu Knoten B mit TTL 3 gezeigt.

Der ursprüngliche Absender des RREQ erhält irgendwann einen RREP von dem Knoten, der auf dem Pfad zum Zielknoten am nächsten liegt. Das Datenpaket kann jetzt über diesen Pfad losgeschickt werden.

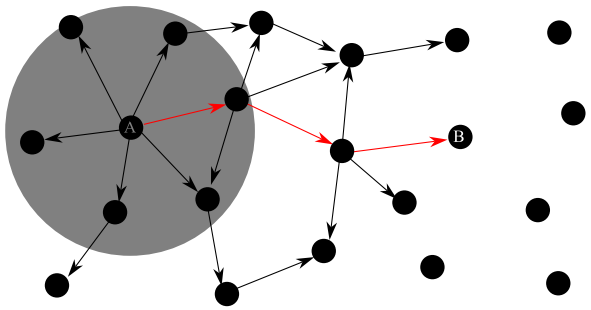


Abbildung 3: RREQ-Nachrichten von A zu B mit TTL 3. Der spätere Kommunikationskanal zwischen A und B ist rot gezeichnet. In grau ist der Broadcastradius von A angezeigt, der der anderen Knoten ist genauso groß.

Erhält ein Knoten ein Datenpaket zur Weiterleitung, kennt aber keinen nächsten Hop für das Ziel des Pakets, sendet er ein RERR-Paket („Route Error“) ab. Dieses läuft den Weg des Datenpakets zurück und sorgt dafür, dass alle Knoten auf dem Weg ihren Routeneintrag für das ursprüngliche Ziel löschen.

5.2 SBR – Statistics Based Routing

SBR ist ein Protokoll, das beliebigen Knoten ermöglicht untereinander zuverlässig zu kommunizieren. Es lässt sich einerseits als proaktives Protokoll einsetzen, jeder Knoten schickt dann regelmäßig so genannte „hello“-Nachrichten um seine Position bekannt zu geben. Diese Pakete werden weitergeleitet und die Routinginformationen darüber aufgebaut.

Andererseits kann es auch in einem hybriden Modus benutzt werden, bei dem, ähnlich wie im AODV eine Routenanfrage gestellt wird, die dann beantwortet wird. Im Unterschied zum AODV sendet der designierte Empfänger aber seine „hello“-Nachricht nicht nur einmal, sondern wiederholt sie regelmäßig, so dass eine Routenänderung sofort erkannt wird.[3]

Der Protokollablauf ist sehr ähnlich zum CTP bzw. zum AODV und wird deswegen hier nicht weiter dargestellt.

6. GEGENÜBERSTELLUNG DER PROTOKOLLE

Protokoll	re-/proaktiv	Struktur	Empfänger
CTP	proaktiv	Baumstruktur	wenige
MCFA	proaktiv	Unstrukturiert	wenige
SBR	einstellbar	kürzeste Wege	beliebig
AODV	reaktiv	kürzeste Wege	beliebig

Bis auf den MCFA erlauben alle Protokolle, dass sich die Knoten bewegen. SBR kann durch das periodische Absenden von „hello“-Nachrichten bei bestehender „Verbindung“

am schnellsten auf Änderungen reagieren, da ohne Fehlerbehandlung (RERR bei AODV bzw. erneutes Absenden eines Beacons bei CTP) die neue Route ermittelt wird.

7. ZUSAMMENFASSUNG

Es wurden viele Routingverfahren für drahtlose Sensornetze entwickelt, die unterschiedliche Anforderungen erfüllen. Bei der Planung eines Sensornetzes muss entschieden werden, welches dieser Verfahren für den Einsatzzweck optimal ist.

Die wohl wichtigste Fragestellung ist, ob die Knoten untereinander kommunizieren sollen oder ob es ausreicht, wenn nur einige spezielle Knoten Daten von allen anderen erhalten können. Sollen die Knoten untereinander kommunizieren, ist es wichtig abzuwägen, ob die Routinginformationen zu Beginn aufgebaut werden sollen, oder ob es reicht damit zu warten, bis sie benötigt wird.

8. LITERATUR

[1] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo. Tep123: The collection tree protocol. 2006.

[2] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*, pages 1–14, November 2009.

[3] A. Klein. Statistics based routing (sbr), technical report no. 453. `http://www.routingprotokolle.de/Routing/Publications/ResearchReport/SBR_TR453_paged.pdf`, January 2009.

[4] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28, 2004.

[5] A. Neumann, C. Aichele, M. Lindner, and S. Wunderlich. Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N.). Internet-Draft, pp. 1-24, April 2008. Network Working Group.

[6] G. Pei, M. Gerla, and T.-W. Chen. Fisheye state routing: a routing scheme for ad hoc wireless networks. In *Proc. IEEE International Conference on Communications ICC 2000*, volume 1, pages 70–74, June 2000.

[7] F. Ye, A. Chen, S. Lu, and L. Zhang. A scalable solution to minimum cost forwarding in large sensor networks. In *Proc. Tenth International Conference on Computer Communications and Networks*, pages 304–309, October 2001.

Medium Access Control (MAC) in Wireless Sensor Networks

Karl-F. Leiss
Betreuer: Dipl. Inf. Alexander Klein
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: leiss@in.tum.de

ABSTRACT
This paper gives an overview of the common Medium Access Controls (MAC) with respect to performance, latency, power consumption and security. Wireless MAC has to take care about harsh environment, limited power and often a big number of nodes to connect. This paper clearly emphasizes the properties of dynamic and static protocols with a short look into hybrid protocols. After explaining the fundamental aspects of wireless protocols, the main cause of energy waste as well as the limitations through the hardware will be reviewed. A survey on representative protocols is presented and the methods to lower collision probability are explained. Attacking scenarios and possible solutions are discussed at the end.

Keywords
Wireless, sensor, mac, contention, delay, latency, security

1. INTRODUCTION AND OBJECTIVES
Development of MAC in wireless sensor networks (WSN) became more and more important during the last years, since applications can benefit from wireless and energy efficient data exchange. The autonomous activity over long periods without any service access requires well designed sensor nodes. Nodes acting in an Ad-hoc (latin: for the moment) network are normally equipped with a small micro controller (uC), a radio transceiver, sensors and a battery pack. Some of the most common nodes are ScatterWeb [4], Mica 2 [9], Tmote Sky and Imote 2 [10]. The nodes preprocess and finally transmit the data. The knowledge of the hardware is especially needed in understanding the bottleneck of wireless communication. The number of nodes within a WSN and the generated traffic is variable. Therefore different solutions are necessary to cope with the data transmission. Energy consumption plays the most important role, if a node would not go to sleep, the lifetime decreases from a runtime of years to only a few days[3]. Herein comes the role of the MAC protocol which should give the best compromise in terms of throughput, latency, scalability and energy consumption. Section 3 gives a comparison of representative MAC protocols. After discussing the basics of MAC protocol design, a detailed look is taken on a particular protocol in section 4. WSN should not only be reliable but also be protected against external, unauthorized access. The term security in the contex of WSN means more than simple protection on the data. The Denial of Sleep attack will be explained in section 5.

2. MAC
First of all the term MAC has to be explained in detail. After that the basic communication issues of WSNs are reviewed.

2.1 Term definition and fundamentals
MAC is defined as the data link layer within the IEEE specified OSI model[6]. It defines the access and the arbitration of multiple nodes on a shared medium. Standards for wireless networks exist but they cannot be directly used for WSN as they are optimized for data throughput. High data throughput is accompanied by high energy consumption and less reliability of the link. The IEEE specifies the Carrier Sense Multiple Access (CSMA) approach[6]. CSMA is a basic scheme to handle the attempt of multiple nodes to send. Before data can be sent by a node the medium must be sensed. If there is no other transmission in process the node can start its own transmission. Therefore the node's transceiver has to switch from receive into transmit mode which takes between 2 and 6 milliseconds. The switching from one mode into another is called *turnaround*. The transmission delay on the wireless medium is bigger than on the wired medium. The reason is the propagation delay on the wireless medium. After the medium is sensed as free, transmission can start. If the medium is occupied a node has to wait until it is free. It is possible that two or more nodes may start their transmission in parallel on a free medium. Collisions will occur and the MAC protocol has to handle this state. In CSMA a scheduled resend is intended. Every node which detects a collision on a transmission attempt will resend the data later after a specific time interval. So the MAC protocol is occupied most of the time listening to the channel which consumes energy.

Hidden node is the naming for the 1 hop problem shown in the Figure 1. A hop is a station, the packet has to pass under its way to the receiver. Node C does not recognize a packet being sent from node A to node B. As the channel is clear for node C, it may start a transmission, too. Only node B will detect a collision but not the other ones. Exposed node problem occurs if another node D is a neighbor. Node C does not have a clear channel as it gets persistent traffic from the other nodes which results in waiting for the idle of the channel hence lowering the throughput. Listening consumes even more energy than the transmission of data. Internal setup of clock and oscillation circuit takes extra time[10].

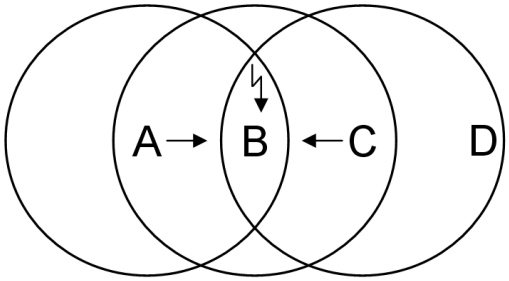


Figure 1: Hidden node problem caused by limited transmission and sensing range in WSN.

2.2 Communication issues

As discussed in the previous section, energy consumption is caused by the time duration spent on sending and receiving. Thus, the goal is to minimize this time. Nearly all WSN MAC protocols address this issue which can be divided into following sources:

- **Idle listening:** Only a small segment of the available bandwidth is really used for the raw data from attached sensors. Most of the time, the radio monitors the medium for being busy[10].
- **Overhearing:** A side effect of listening to incoming bits all the time is, that every message is processed whether the node is the intended receiver or not. Especially in dense networks where many nodes are in the transmission range of each other, the overhearing leads to a great minus in the energy asset[8].
- **Collisions:** In CSMA networks exist always the possibility of collisions even if a random backoff mechanism is used. More explanation on the backoff mechanism is given in Section 4.2. The problem is caused by the turnaround time during which no activity on the medium can be detected. Approaches for handshake like RTS/CTS [1] introduce additional overhead compared to the relatively small payload of the WSN packets [10]. Retransmitting packets can completely shut down the network's transport bandwidth in worst case.
- **Protocol overhead:** Headers for the MAC and control message data should be minimized to improve efficiency. A strategy to minimize the overhead is the aggregation of data. Buffering introduces some delay which can be compensated by reduced protocol overhead.

The first idea to improve the communication, was to listen to the channel only periodically. The problem with this idea is that a short data packet might not be recognized. Therefore a preamble at least as long as the sleep period must be put in front of the data packet. The Low Power Listening (LPL) protocol implements this approach.

[1]ready to send / clear to send

2.3 Performance limiting factors

A performance limiting factor is the Clear Channel Assessment (CCA) delay. This delay is the time a node in receiving mode needs, to clearly detect the channel state. At least eight bits have to be sampled to decide on the medium state. The delay period is 128us wide on a tranceiver with 76kbps[14]. The key parameters of three common transceivers are shown in a compressed form in the Table 1 [10]. Noticeable is the fact, that the modern Chipcon CC2420 transceiver from Texas Instruments draws more current in receiving path[14]. The specifications of the latest device, the CC2520, outperforms the CC2420 in terms of efficiency. The benefit of those newer transceivers is highlighted in Section 4.

Type	RFM TR 1000	CC1000	CC2420
Speed	10kbs	76kbs	250kbs
Sleep	2uW	100uW	60uW
Receive	12mW	36mW	63mW
Transmit	36mW	75mW	57mW
Setup	0.5ms	2ms	1ms

Table 1: WSN transceivers with their reference values

3. COMPARISON OF REPRESENTATIVE PROTOCOLS

Figure 2 classifies the most representative scheduled and random access protocols. In terms of complexity random access

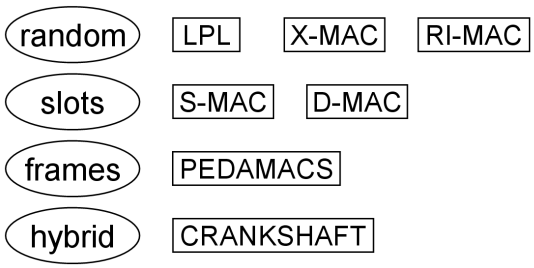


Figure 2: WSNs ordered by class

is the most simple approach. A technique called preamble sampling helps to save energy by sleeping most of the time. A wake up is periodically done to check for a preamble being sent by other nodes. Thus, the effort for carrier sense is shafted towards the sender. Slot based protocols divide their schedule into small segments. Inside these slots the message exchange can take place. Slots are the basis for frame based protocols. A frame is split into multiple slots. The slots inside a frame can have different functionality.

S-MAC

Sensor-MAC or S-MAC belongs to the slot based protocols. A fixed schedule accommodated by a sync packet, synchronizes the nodes to the slot structure. The sync packet contains the time stamp as broadcast permitting the other nodes to adjust their offset. Furthermore, S-MAC implements carrier sense and a RTS/CTS handshake to avoid collisions. Figure 3 illustrates the messaging scenario. Sender and receiver are only active during the data exchange period. It is also possible to determine slots for broadcast data. In

broadcast slots no RTS/CTS handshake exists. The message data can also be used to deploy a new schedule to the WSN which allows dynamical reconfiguration. The Carrier Sense in front of the SYNC and the RTS symbol is necessary to minimize collisions. After the Sender S has received the RTS symbol the data exchange can happen. White boxes are sent data, grey ones are received data packets[8].

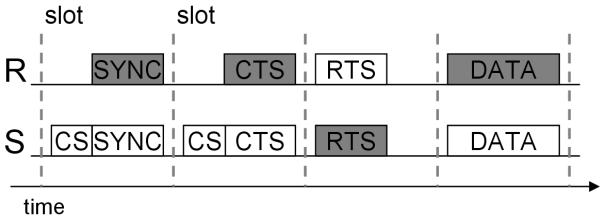


Figure 3: S-MAC slot based packet handling with carrier sense (CS)

D-MAC
Known as Datagathering MAC and a Time Division Multiple Access (TDMA) style MAC variant, D-MAC increases the usage of pure TDMA based protocols. TDMA is the opposite of CSMA. TDMA is based upon a static and predefined send/receive schedule. Slots are used for data exchange and synchronization of the nodes. In most sensor networks, the data packets from many sources are transmitted under involvement of neighbors to a sink. This topological appendage is known as convergecast traffic as the predominant part of the communication is unidirectional. The aim is to be energy efficient while achieving a low latency level. The slots are planned such that subsequent transmissions from hop to hop are appended slot by slot. Figure 4 demonstrates this scheme. During a receive period neighbors can have a

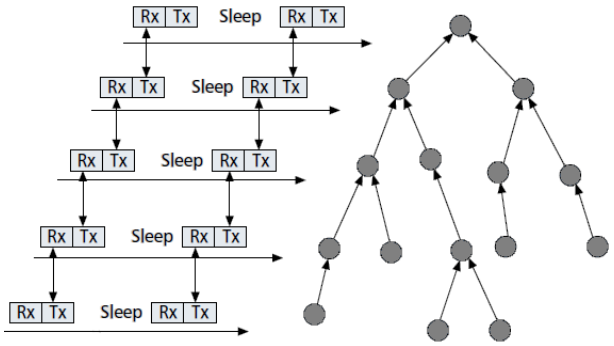


Figure 4: DMAC subsequent slot schedule beside data gathering tree

transmit period and therefore reduce the latency to the data sink. No unwanted sleep/wakeup periods are necessary for optimal latency. Every higher layer in the data gathering tree buffers data from lower nodes until next transmission slot. This is very efficient but not flexible. The drawback of D-MAC is the absence of a collision check. A collision may occur on a mobile network where nodes can move. It cannot be ensured that two or more nodes are sending within the same slot to the same receiver after they changed their place. This effect is likely to break up if the exact transmission path is not known in advance[8].

PEDAMACS
addresses a similar use case as D-MAC. The Power Effcient and Delay Aware Medium Access synchronizes the sink to all nodes implicating the deployment of a high power radio transceiver on the sink. As the sensor nodes are less powerful, a spanning tree has to be set up on initialization and nodes report back hop by hop. Once all information is gathered by the sink, it can compute the schedule based on the topology. The intended traffic pattern is convergecast. It is possible to repeat the initialization to compensate node movement or problems on the wireless link. With all the planing and setup the network is efficient but there are disadvantages. Introduced as TDMA based protocol, PEDAMACS uses CSMA during the initialization. In the initialization period are typically many broadcast frames being sent out which can lead to long initialization time. More over, PEDAMACS cannot guarante the sink will reach all nodes. Environmental circumstances may disrupt the service [8, 13].

Crankshaft
is a hybrid WSN MAC protocol, implementing both CSMA and TDMA in a new way. It is specifically designed for dense networks where the number of neighbor nodes is larger than ten. Instead of scheduling the slots of the sending node, the ones of the receiver are timed. Thus, a wake up is only necessary within the wanted receive slot. A simple algorithm allocates the receivers to the slots by node identifier modulo frame length. Crankshaft schedules the data exchange into frames which are divided into smaller slots. Two types of slots exist, broadcast and unicast slots. Broadcast slots are utilized for messages designated to all receivers, therefore all nodes have to wake up. On the other hand, unicast slots are only addressed to the receiver. Figure 5 shows a unicast slot. During the contention period in a such a slot, CSMA arbitration takes place. Sender S1 polls the channel (grey box) and starts to transmit his preamble P. Sender S2 also wants to transmit but before he has to poll the medium. Since preamble transmission from Sender S1 is already in progress, Sender S2 goes to sleep until his next scheduled slot. The intended Receiver R polls the slot, too. After the preamble is sent, the actual payload is transmitted and finally confirmed with an acknowledgment (ACK). So during the contention window all possible senders have to figure out whether they are allowed to send by using CSMA inside this TDMA scheduled unicast slot. Crankshaft reduces overhead especially for dense networks. It outperforms S-MAC [5].

X-MAC
This protocol belongs to the random access protocols shown in Figure 2. Previously talked about hybrid MAC, a step is taken to the asynchronous duty cycle protocol X-MAC. This protocol organizes the WSN such it operates in completely decoupled schedules for receiver and sender. Again, the low power listening idea together with preamble sampling is applied. The difference is that the preamble embeds the target address to save overhead and the preamble sequence is shortened. Compared to LPL in Figure 6, X-MAC uses multiple and short preambles. There are several advantages. First, the reduced energy amount spent on sending/receiving the preamble. Second, the latency drops as the receiver can answer as soon as he wakes up and does not have to wait until the end of the preamble sequence. Third, unmeant

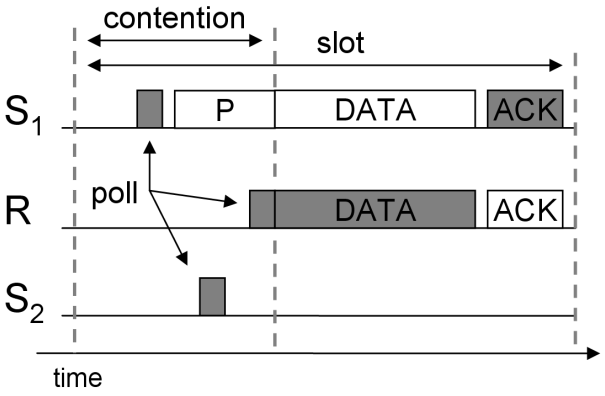


Figure 5: Crankshaft contention and message reception

receivers can go back to sleep after they got the preamble with the address information. Optionally, X-MAC offers an adaptive algorithm to adjust the duty cycle for better energy versus packet balance. With X-MAC being one of the latest protocol members in the evolution of WSN MAC, it takes into account the technological developments on the transceivers. CCA and turnaround time have great influence on the preamble gaps, therefore X-MAC suffers from older hardware [12]. X-MAC is optimized for light traffic, the scaling ratio under higher load is less optimal[15]. A nice side effect of the early ACK is the fact, that it acts as a CTS. This reduces the collision probability furthermore. A early ACK can also be sent in one of the gaps between the strobed preambles. The dotted line marks the activity period of a node. LPL has to wakeup early to get the complete long preamble. X-MAC stays active for a short time after a data transmission to listen for further preambles.

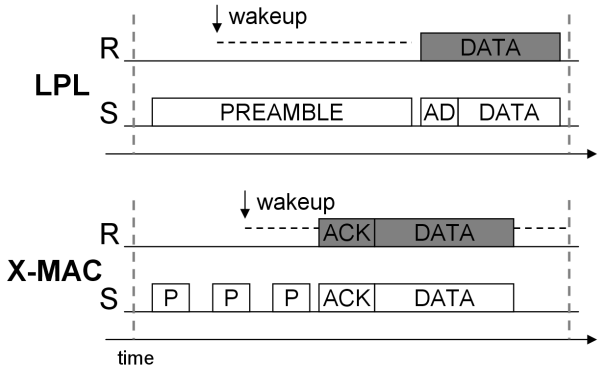


Figure 6: Difference between standard LPL and X-MAC protocol

4. PRESENTATION OF A SPECIFIC WIRELESS MAC PROTOCOL

A newer protocol has been selected to be discussed in detail. It not only implements new or different ideas, it also focuses on issues of other modern protocols like X-MAC. Most designers aim to reduce idle listening with more or less practical analysis. As shortly raised in the MAC protocol comparison before, the underlaying hardware plays a big role. A deep and critical look into **RI-MAC** will be done in the upcoming sections. RI-MAC is the abbreviation of Receiver Initiated MAC.

4.1 RI-MAC

RI-MAC belongs to the random or asynchronous protocols. The name suspects that the transmission sequence is initiated by the receiver (the sink) and not by the sender. The idea behind comes basically from infrastructure driven networks but remembering the multi-hop nature of an Ad-hoc network.

4.2 Design considerations of RI-MAC

A node, intended for receiving data wakes up based on its schedule and checks the channel for being idle. If the channel is in idle state a beacon B is transmitted by the Receiver R. Figure 7 illustrates the basic operation. Next the assignation of the beacon has to be reviewed. The beacon serves as request and as ACK for a data transmission. The Sender S wakes up to wait for a incoming beacon. The dotted line in the Figure 7 marks the active period of the transceiver. The receiver signalizes the sender the start of the data transmission. After the transmission a beacon is sent by the receiver to acknowledge the transmission. In other words, the receiver controls the medium and announces duty cycle changes by the help of the beacon.

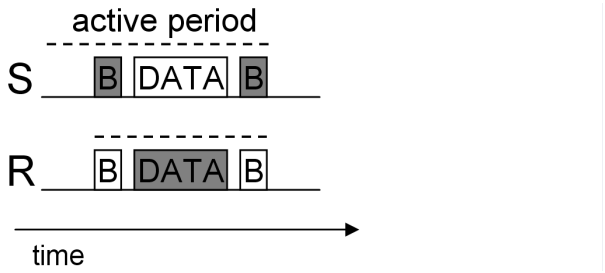


Figure 7: RI-MAC: Initiation of data exchange by the receiver R

The designers of RI-MAC have used the CC2420 radio for the implementation. This transceiver is mainly designed for IEEE 802.15.4 networks[7]. Within 802.15.4 there this a beacon format already defined. RI-MAC reuses the hardware preamble, frame length, frame control field and the frame check sequence. RI-MAC adds specific fields, the source/destination address and the backoff window (BW). Two beacon types exist in RI-MAC, a base beacon including only the source field and the extended beacon with BW and destination field. The two types can be easily divided by the receiving node due to its value inside the length field. A beacon without BW information will request the sender to start data transfer immediately.

The **backoff window** incorporates a value telling the sending nodes when they should start the transmission of their data frame. This value is computed into a time interval which the node should wait until the transmission of the next data frame. The purpose is to keep the risk of collisions on the channel low. A commonly used method to calculate the backoff time is the binary potential backoff strategy as

utilized in IEEE 802.3 e.g. Ethernet. The binary potential backoff window value is doubled on each collision. The problem on the binary potential backoff results in large waiting time on big values when the number of nodes is small. Another method is the geometrical-increasing probability distribution used in the Sift protocol [11]. A node chooses its contention slot from a geometrical distribution. Due to the nature of this distribution many nodes will pick a high slot number and a few nodes a small slot number. The smaller the chosen slot number the less the collision probability.

The difference in BW is RI-MAC being receiver orientated. After reaching the maximum BW size due to multiple collisions the receiver goes to sleep. On the sender side a missing ACK beacon within a dedicated timespan will be recognized and a counter for retransmission will be increased. The sender cancels the transmission if the predefined retry limit is reached. The detection of collisions is based on the hardware preamble bit sequence inside the beacon. If this fixed sequence is corrupted the beacon is not recognized as correct and must be sent again. Collisions on the data frames are possible but with much lower probability. Based on the backoff window value the receiver knows when to get the next data frame from a specific sender. If the received frame is outside the intended time span or the checksum is wrong no ACK will be sent back. Figure 8 shows two Senders S1 and S2 contending for transmission. As soon as the Receiver R sends a beacon B both senders will start immediately with the data transmission. The consequence is a collision. To solve the collision on the next attempt, a new beacon is sent out by the receiver. This time the beacon is populated with a backoff window value, marked with a dotted circle in the Figure 8. This value is processed by the nodes on reception and used to set the next transmission attempt for the pending data. The conflict is solved by Sender S1 doing its cycle before Sender S2 does[15].

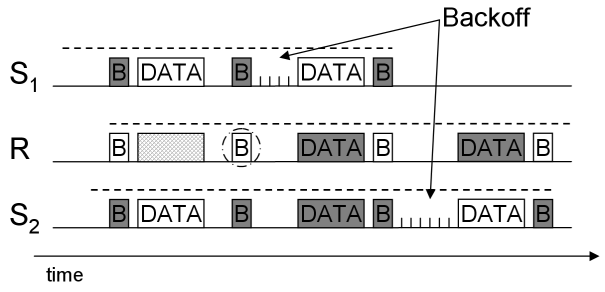


Figure 8: DATA frame transmission from contending senders in RI-MAC.

To let the beacon act as an ACK, the destination field of the beacon is set to the address of the last received data frame. Thereby the sender recognizes the beacon.

RI-MAC lacks the option to get the data directly, but there is the so called beacon on request. In networks with high traffic load or many nodes, the receivers might be active anyway. This can be adapted to process a beacon sent by sending node to tell the receiver the wish to initiate a data transfer. A beacon from a sender requests a beacon from the receiver which starts the normal frame sequence. The benefit is to use the standby activity of the transceiver on the receive side for faster data processing which helps to lower latency and energy consumption.

4.3 Evaluation of RI-MAC

The standard approach for evaluating such networks designs is to build a model for the network simulator ns-2[1]. In this paper, it was decided to concentrate on practical benchmark results as they take the CCA delay into account. Furthermore, it is not clear how an undistinguishable signal affects the protocol state if this is above the CCA threshold. The authors of X-MAC and many other protocol designers do not tackle this possible problem emerging in larger networks.

A preface to the evaluations, most comparisons reflect LPL as basis. In this context, X-MAC was chosen as representative for random access MAC protocols. The practical implementation was done on MICAz motes hardware with TinyOS. The number of nodes is twice the number of data flows. A flow is the individual traffic a receiving node has to satisfy. On increasing number of flows from every sender from one to four, the duty cycle spent on the medium increases on X-MAC while RI-MAC stays below 60%. It is clear that a higher duty cycle affects the energy consumption in proportional way.

Another interesting aspect is the classical hidden node problem. If two senders are not in the reception range of each other, the probability for packet loss at the receiving node is high and accompanied by higher duty cycle due to retransmission. The average ratio of unsuccessful transmissions with RI-MAC is about five percent lower compared to X-MAC. In contrast the results for not hidden nodes differs not much. The results were achieved by the average of ten runs [15].

As RI-MAC was designed especially for dense networks a comparison was made by using ns-2. The simulated scenario consists of a 7x7 nodes network with a sensing range of 100 to 500 meters. The 30 simulation runs trigger a series of 100 events per cycle. RI-MAC outperforms X-MAC in terms of delivery ratio. Under heavy load RI-MAC scales better than X-MAC. RI-MAC can successfully handle more concurrent flows within one transmission cycle. The wider the sensing range the less is the decrease of delivery ration of RI-MAC. Starting with the 100m range, X-MAC and RI-MAC are on the same level. At the 300m range X-MAC looses about 21% and at the 500m range about 42% [15].

5. SECURITY ASPECTS ON WIRELESS SENSOR NETWORKS

Security in general got more and more important in the last years. Many systems suffer from the beginning of their design in these questions. Likewise simple attacks can have serious impact on the operation of a system.

The weakness of WSNs is their dependency of the battery capacity. As discussed above, all protocols try to reduce energy consumption while maintaining latency and data throughput on a low level. A Tmote Sky node using two AA batteries with 3000 mAh each has got a run time of hundreds of days in sleep but only a week in receive mode. A additional problem is the self discharging of the batteries. Even if the

energy consumption of a node is very low, in the range of 0.1 to 70mW, self discharging grows to the end of the battery life cycle. Attacks to increase the energy consumption of the nodes are called Denial of Sleep attacks (DoS). Attacks on WSNs in general can be categorized into three classes:

- **Class 1: unknown protocol attack**
  This categorizes attacks, sending out high power pulses or jamming to disrupt the communication among the nodes. Jamming can also be used to generate collisions every time traffic is detected by the aggressor. More intelligent is a record of traffic and later replay. Replaying data can lead to miss detections.

- **Class 2: known protocol, untrusted traffic**
  Identifying the protocol used in a WSN by traffic analysis helps the attacker to save energy on his own node(s). The more accurate an aggressor emulates frames, the more difficult it will be for the network to detect this. Even if frames are encrypted, a receiving node has to decrypt and afterwards to dump it which wastes energy. Broadcast messages are not further proceeded by the nodes and are discarded.

- **Class 3: full access including authorization**
  Knowing the MAC protocol and it's authorization mechanism on the data link layer offers the aggressor maximum possibilities. He can send trusted traffic which cannot be sorted out by the nodes. It will be difficult to isolate such aggressor node(s) since this attack can be done random. Also wrong or even multiple source identities can be used to disturb the whole network. In this scenario not only energy is wasted, data can be manipulated. Solving the consequences of those attacks can be very cost intensive. A well timed SYNC frame in the S-MAC protocol for example, stops the nodes from entering the sleep mode. After a week the affected nodes would not respond anymore because the battery would be empty.

To prevent those attacks some techniques are listed below.

- **Strong link-layer authentication**
  Authentication at the data link layer is needed to ensure the service availability of a WSN. Using authentication on layers above the data link layer will only ensure data integrity. Broadcast frames in many protocols have got a simple structure. This frame type is well suited for an attack since all nodes receive it and the aggressor does not have to take care about specific data inside this frame. Therefor the authentication technique is important to defense the DoS and broadcast attacks[3]. TinyOS offers the TinySec component for this purpose[2].

- **Replay protection**
  To prevent recorded and replayed traffic from disturbing nodes, a table of neighbor nodes can be established in combination with sequence numbers attached to the packets. But this is not very safe at the data link layer and requires additional memory.

- **Jamming recognition**
  WSNs are resource orientated and have only a single channel radio with limited capabilities. Without a spectrum analysis it is hard to detect a jammer. Generally, jamming blocks the whole traffic, so nodes have to check the medium periodically for a free channel. A real protection cannot be done against this type of attack. It is only possible to recognized it and to go to sleep mode for longer intervals.

- **Isolation of compromised nodes**
  Detecting compromised nodes in alliance with blending the nodes out from the network is a desirable option. Therefore an asymmetric encryption approach is very effective. The negative side, asymmetric key mechanisms overload the sensor nodes processing capabilities.

Especially the early developed protocols are accessible for DoS attacks. Newer ones or ones taking security into account help to reduce the DoS issue. It widely depends on the application and its requirements which protocol and which security features to choose[3].

6. CONCLUSION

The need for dedicated MAC protocols in WSNs has been discussed under the given limitations. These limitations come from the underlaying hardware, limited battery capacity, harsh environment with a wide temperature range and a short transmission range. A wide field of applications makes it impossible to have a single solution. Many issues can be eliminated in the design phase. The decision, which protocol to choose, should not depend on a single parameter like performance or battery runtime. Especially the network size, the node allocation, possible other radio transmitters in the neighborhood and the estimated traffic pattern have to be considered.

Very popular are the dynamic protocols as they are easy to integrate. They are flexible in terms of network movement and network extension. Various implementations exist for different number of nodes under high or low traffic load. The latest protocols improve the energy efficiency again by addressing mainly the schedule and the balance between sender and receiver to lower the duty cycle. Features offered by the hardware are also taken into account.

Finally an important subject, the security, was considered. Popular and easy to adapt attack mechanisms accompanied with possible defense methods were presented. At the moment, most of the protocols are susceptible for comparatively lightweight attacks which have however great impact on the battery lifetime of a node.

7. REFERENCES

[1] The network simulator ns-2, May 2010. http://www.isi.edu/nsnam/ns/.

[2] TinyOS, May 2010. http://www.tinyos.net/.

[3] David Raymond, Randy Marchany, Michael Brownfield, and Scott Midkiff. Effects of Denial of Sleep Attacks on Wireless Sensor Network MAC Protocols. In *Proceedings of the 2006 IEEE Workshop*

on Information Assurance. United States Military Academy, 2006.

[4] Freie Universität Berlin. ScatterWeb. June 2010. http://cst.mi.fu-berlin.de/projects/ScatterWeb/.

[5] G.P. Halkes and K. Langendoen. Crankshaft: An energy-efficient mac-protocol for dense wireless sensor networks. *Faculty of Electrical Engineering, Mathematics and Computer Science Delft University of Technology, The Netherlands*, 2007.

[6] IEEE Computer Society. *802 IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture*. The Institute of Electrical and Electronics Engineers, Inc., 2002.

[7] IEEE Computer Society. *802 Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*. The Institute of Electrical and Electronics Engineers, Inc., 2006.

[8] Ilker Demirkol, Cem Ersoy, and Fatih Alagöz. MAC Protocols for Wireless Sensor Networks: A Survey. *IEEE Communications Magazine*, pages 115–121, April 2006.

[9] J. Hill and D. Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22:22–24, 2002.

[10] K. Langendoen. Medium Access Control in Wireless Networks. 2:535–560, 2007.

[11] Kyle Jamieson, Hari Balakrishnan, Y.C. Tay . Sift: A MAC Protocol for Event-Driven Wireless Sensor networks. May 2003.

[12] Michael Buettner, Gary V. Yee, Eric Anderson, Richard Han. X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks. *SenSys, Boulder*, November 2006.

[13] S. Coleri-Ergen and P. Varaiya. Pedamacs: Power effcient and delay aware medium access protocol for sensor networks. *IEEE Trans. on Mobile Computing*, pages 920–930, May 2006.

[14] Texas Instruments. CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver, June 2004. http://www.ti.com/lit/gpn/cc2420/.

[15] Yanjun Sun, Omer Gurewitz, David B. Johnson. RI-MAC: A Receiver-Initiated Asynchronous Duty Cycle MAC Protocol for Dynamic Traffic Loads in Wireless Sensor Networks. *Department of Computer Science, Rice University, Houston, TX, USA / Department of Communication Systems Engineering, Ben Gurion University, Israel*, November 2008.

Vergleich des Energieverbrauches der Protokolle Z-Wave und Zigbee

David Brodski
Betreuer: Christoph Soellner
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2010
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: brodski@in.tum.de

KURZFASSUNG

Sensornetzwerke sind bei der Überwachung und Steuerung von Anlagen und Gebäuden eine große Hilfe. Jedoch ist es nicht immer möglich die dafür notwendige Stromversorgung bereitzustellen, sodass die Knoten mit Batterie betrieben werden müssen. In diesem Text geht es daher um die Energiesparmaßnahmen von Zigbee und Z-Wave. Angesprochen wird der Protokollaufbau, das Routing und die Gerätetypen mit Fokus auf die Möglichkeiten zum Stromsparen. Im zweiten Teil werden die beiden Funktechniken verglichen und die Batterielaufzeiten mit verschiedenen Batterietypen abgeschätzt. Eine Auswahl der richtigen Technologie für ein Sensornetz kann anschließend anhand der beigelegten Tabellen gemacht werden.

Schlüsselworte

ZigBee, Z-Wave, Energie

1. EINLEITUNG

Gebäudeautomatisierung wird in Zukunft eine immer wichtigere Rolle spielen. Dennoch sind bei späterem Einbau in ein Gebäude meist nicht die nötigen Kabel vorhanden. Dort sind dann verschiedene Funktechniken die einzige Möglichkeit, die anfallenden Daten zu transportieren. Da Strom nicht immer vorhanden ist, ist es oft notwendig, die Sensor- und Aktor-Knoten über Batterie zu versorgen. Ein Vergleich der verschiedenen Techniken ist daher wichtig, um die richtige Technik für das gewünschte Einsatzgebiet zu finden. Diese Arbeit beseht aus drei Teilen, im ersten Teil wird Zigbee vorgestellt, im zweiten Z-Wave. Der dritte Teil beinhalten einen direkten Vergleich der beiden Funktechniken in verschiedenen Szenarien und mit verschiedenen Batterien.

2. FUNKTECHNIKEN IM VERGLEICH

In den zwei folgenden Abschnitten möchte werden zwei Funktechniken vorstellen, welche bei Sensornetzen Verwendung finden. Besonderes Augenmerk soll dabei auf die Energiesparmöglichkeiten dieser Funktechniken liegen. Spezielle Protokolldetails sind im folgenden also nur in Energiesparangelegenheiten zu finden.

2.1 Zigbee

Zigbee ist ein Kurzstreckenfunkverfahren, welches ein Wireless Personal Area Network (WPAN) implementiert. Es ist ausgelegt für eine Energie effiziente Datenübertragung von kleinen Datenmengen mit maximal 250 Kbit/s. Es baut dabei auf IEEE Standard 802.15.4 auf und kann in den 868/915 MHz und im 2,4 GHz Band betrieben werden. Im 868/915 MHz Band kann dabei entweder mittels „Direct Sequence Spread Spectrum“ (DSSS) und dem „Binary Phase Shift Keying“ (BPSK), mittels DSSS und „Offset Quadrature Phase Shift Keying“ (O-QPSK) oder mittels „Parallel Sequence Spread Spectrum“ (PSSS) und „Amplitude Shift Keying“ (ASK) gearbeitet werden. Im 2,4 GHz Band wird mit DSSS und O-QPSK gearbeitet. Eine Implementierung des ebenfalls möglichen Frequenzsprungverfahrens (FHSS), wie sie bei Bluetooth verwendet wird, ist im IEEE 802.15.4 Standard nicht vorgesehen, da dafür eine längere Synchronisation notwendig wird.[6, Seite 57 bis 60]

2.1.1 Rahmen- und Datenformat bei der Übertragung

Auf den Ebene eins bis vier des OSI-Referenzmodells wird bestimmt, wie Daten einer Anwendung über das physikalische Medium versendet werden. Bei jedem Übergang von einer höheren Schicht auf eine niedrigere, werden dabei die Daten in einen Rahmen verpackt welcher zusätzliche Steuerinformationen enthält. Jeder Rahmen vergrößert das Paket und trägt somit einen wesentlichen Anteil an der Sendezeit und somit am Stromverbrauch bei. Ein Paket kann bei Zigbee maximal 133 Bytes groß sein, minimal 16 Bytes. Der Aufbau eines Paketes sieht folgendermaßen aus:

Physikalische Schicht (Schicht eins):

- 4 Byte Präambel
- 1 Byte Start Of Frame (SOF)
- 1 Byte Paketlänge (letztes Bit für Erweiterungen reserviert)

MAC Schicht (Schicht zwei)

- 2 Byte Frame Control - Steuerfeld für Art des Paketes
- 1 Byte Sequenznummer
- 4 bis 20 Byte Adressinformationen / PAN Nummer
- n Byte Nutzdaten
- 2 Byte Frame Check Sequenz (FCS) enthält eine CRC

Der minimale Overhead beträgt somit auf Ebene eins und zwei 12% bei kurzem Header (118 Byte Nutzdaten, 15 Byte Header) und 24% bei langem Header (31 Byte). Errechnen lässt es sich aus der Summe der Header Byte geteilt durch die maximal Länge eines Paketes.

Um auf einem Knoten mehrere Applikationen gleichzeitig getrennten Netzwerkzugriff zu ermöglichen, steht der so genannte „Applikation Support Sublayer“ (APS) bereit. Er befindet sich innerhalb der Nutzdaten der MAC Schicht und bietet eine Infrastrukturkonzept ähnlich der Ports bei UDP und TCP.

Netzwerk Schicht (Schicht drei)

- 1 Byte Frame Control
- 0-1 Byte Destination Endpoint
- 0-2 Byte Group Addresse
- 2 Byte Cluster Identifier
- 2 Byte Profile Identifier
- 1 Byte Source Endpoint
- 1 Byte APS Counter
- 0-n Byte Extended Header
- n Byte Nutzdaten

Nach Schicht drei beträgt der Overhead somit bereits mindestens 22 Byte und macht somit 17% eines Paketes mit 133 Byte aus oder mindestens 20% mehr muss gesendet werden. Bei einem Paket mit langem Header, d.h. alle zusätzlichen Bytes im Paket werden benützt (41 Byte Header) steigt der Overhead auf 31% an oder 45% an zusätzlichen Daten muss mitgesendet werden.

Zusammenfassend bedeutet dies, dass bei Ausnutzung der vollen Paketgröße, eine Anwendung mindestens 20% länger übertragen muss, als es die Datenrate vermuten lässt. Natürlich kann es durch Störungen oder Versenden von kleineren Paketen zu noch größeren Einbrüchen in der Datenrate kommen.[6, Seite 66 / 67]

2.1.2 Gerätetypen bei Zigbee

Im Zigbee Standard sind zwei Gerätetypen definiert. Da sie sich in der Hardware unterscheiden muss die Wahl des richtigen Typen bereits beim Entwurf der Anwendung berücksichtigt werden.[6, Seite 67]

Full Function Device. Ein Full Function Device (FFD) kann jede Rolle in einem Zigbee Netzwerk erfüllen und ist somit in jede Topologie einsetzbar. Als Koordinator kann es ein neues Netzwerk erstellen.

Reduced Function Device. Ein Reduced Function Device (RDF) besitzt weniger Rechenleistung und weniger Speicher und ist somit billiger herzustellen. Jedoch ist ein RDF nur in einer Stern-Topologie einsetzbar, da es einen Router (siehe 2.1.3) zum Versenden von Daten an andere Knoten braucht.

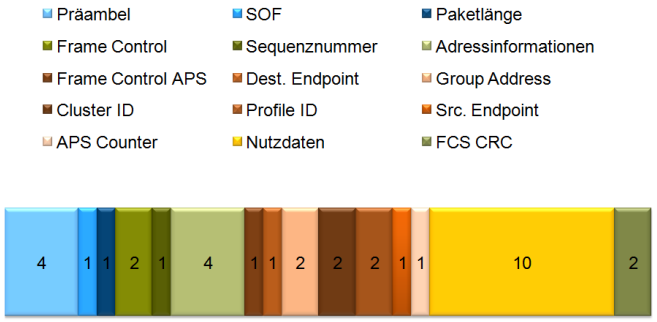


Abbildung 1: Zigbee-Paket mit 10 Byte Nutzdaten

2.1.3 Geräterollen bei Zigbee

In einem Zigbeenetzwerk kann ein Knoten einer der drei folgenden Rollen übernehmen, welche durch die Programmierung festgelegt wird:[6, Seite 84 - 87]

Der ZigBee Koordinator. In jedem ZigBee Netzwerk muss es genau einen Koordinator (ZigBee Coordinator - ZC) geben, welcher auf einem FFD Knoten läuft. Er hat folgende Aufgaben:

- Er startet ein neues ZigBee Netzwerk, indem er das Frequenzband scannt und den Kanal wählt, auf dem am wenigsten Störungen anzutreffen sind. Anschließend legt er die Identifikationsnummer des Netzwerkes fest (PAN ID) und startet damit das Netzwerk.
- Er erlaubt anderen Knoten dem Netzwerk beizutreten. Dazu teilt der Koordinator dem beitretenden Knoten eine temporäre 16 Bit Adresse zu. Die Kommunikation geschieht hierbei direkt mit dem Koordinator oder einem zwischengeschalteten Router (siehe 2.1.4).
- Er kann ein so genanntes „Trust Center“ beinhalten, welches Sicherheitseinstellungen und Schlüssellisten der Knoten verwaltet.
- Er kann auch als Router arbeiten (siehe 2.1.3)
- Er kann die selben Funktionen anbieten wie jeder Endpunkt, z.B. Steuerung einer Lampe

Da der Koordinator immer vorhanden sein muss, wenn ein Knoten dem Netzwerk beitreten will, ist der Koordinator in der Zigbee Spezifikation als nicht batteriebetriebener Knoten definiert. Natürlich muss der Koordinator das erste Gerät sein, welches in einem Netzwerk angeschaltet wird. Sobald alle Knoten dem Netzwerk beigetreten sind und sich der Zustand des Netzwerkes von der Aufbauphase in die normale Arbeitsphase gekommen ist, spielt der Koordinator keine wichtige Rolle mehr. Theoretisch könnte er somit ausfallen, da es jedoch noch keine standardisierte Methode zum Wiedereingliedern eines Koordinators gibt, würde der Koordinator nach dem Wiedereinschalten ein eigenes Netzwerk bilden.

Der Zigbee Router. Der Zigbee Router (ZR) oder auch „PAN Koordinator“ hat hauptsächlich die Aufgabe, Pakete im Netzwerk weiterzuleiten, um die Reichweite des Zigbee Netzwerkes zu erhöhen. Dazu meldet er sich zunächst am Netzwerk an und kann anschließend anderen Knoten die Möglichkeit bieten Daten über ihn zu senden. Falls diese sich noch nicht am Netzwerk angemeldet haben, können diese sich über ihn anmelden. Anschließend bekommen sie eine Adresse zugewiesen, welche diese Knoten eindeutig als seine Subnodes identifiziert. Dies ist wichtig, damit das Routing richtig funktioniert. Natürlich kann der Router auch eigene Anwendungen, wie z.B. Messsensoren, bereitstellen. Da der Router Pakete weiterleiten muss, muss er immer empfangsbereit sein. Daher ist er in der ZigBee Spezifikation als netzbetrieben ausgeführt. Batteriebetriebene Router werden zwar immer wieder angesprochen, sind aber mangels des noch nicht unterstützten Beacon-Enabled-Modus (siehe 2.1.5) in der Zigbee Spezifikation nicht vorgesehen.

Der Zigbee Endknoten. Ein Endknoten, genannt Zigbee End Device (ZED), im ZigBee Netzwerk trägt die eigentliche Funktionalität des Zigbeenetzes. Sie müssen keine Funktionalität des Routings und Managements bereitstellen und können damit auf RFD - Reduced Function Devices laufen. Daher müssen sie keinerlei netzwerkbetreffenden Informationen speichern, welches zu weniger Speicher- und Rechenbedarf führt. Daher sind sie wesentlich billiger herzustellen. Es gibt von ihnen typischerweise wesentlich mehr als von Routern bzw. Koordinatoren.

Zigbee-Endknoten können entweder immer empfangsbereit oder nur periodisch aktiv sein. Dies wird über den Hardwareparameter „RxOnWhenIdle“ festgelegt, welcher somit für einen Großteil der Energieeinsparungen zuständig ist. Falls „RxOnWhenIdle = false“ gesetzt wurde, kann über einen weiteren Parameter die Schlafphasen-Zeit in einem großen Bereich eingestellt werden. Wenn ein Knoten sich in einer Schlafphase befindet, werden Nachrichten, welche an ihn gesendet werden, beim nächsten näheren Router zwischengespeichert. Dieser muss somit eine Liste der Stromspar- und Pollingeinstellungen und einen Puffer für Nachrichten besitzen.

2.1.4 Topologien

Durch Ausnutzung der Gerätetypen kann es zwei verschiedene Topologien auf der MAC-Ebene geben:

Stern. In einer sternförmigen Netzwerktopologie kommunizieren Endknoten über einen PAN-Koordinator um andere Endknoten zu erreichen. Dieser muss ein FFD sein, da RFDs nicht die nötige Hardware besitzt. Als Endknoten können sowohl FFDs als auch RFDs dienen.

Peer-to-Peer. In einer Peer-to-Peer (P2P) Topologie kommuniziert jeder Knoten direkt mit dem Empfänger, solange dieser sich in Funkreichweite befindet. Prinzipbedingt können somit an einer P2P Topologie nur FFDs teilnehmen. Diese Topologie entspricht einer voll vermaschten Topologie, wird aber bei ZigBee nicht so genannt. Die Aufgabe eines PAN Koordinators übernimmt eine der Stationen.[6, Seite 68]

Zudem können beide Topologien gemischt werden. Hierbei ist zu beachten, dass RFDs nur über einen PAN Koordinator Zugang zu einer P2P Topologie erhalten können. Eine beliebte Mischtopologie ist die Baumstruktur welche auch im Routingprotokoll eine Rolle spielt.

2.1.5 Energiesparmöglichkeit Beacon-Enabled Modus

Speziell zum Betrieb von batteriebetriebenen Routern wurde der Beacon-Enabled Modus entwickelt. Er sorgt dafür, dass sich die Endknoten, welche nach der Schlafphase nicht mehr synchron arbeiten, anschließen wieder synchron sind. Zusätzlich werden mit dem Beacon-Enabled Modus zwei neue Hardware - Konfigurationsparameter eingeführt, die „macBeaconOrder“ (BO) welche die Schlafzeit zwischen zwei Beacons regelt, und die „Superframe Order“ (SO), welche die aktive Zeit innerhalb zwei aufeinander folgender Beacons regelt. Die Zeiten können jeweils zwischen 16ms und 252s geregelt werden. Er wurde nach [6] noch nicht in den Standard aufgenommen. In den Berechnungen in 2.3 wird mit dem Beacon-Mode gerechnet, da ohne ihn die Batterien in den Routern weniger als eine Woche halten.[6, Seite 78]

2.2 Z-Wave

Z-Wave wurde speziell für die Heimautomatisierung entwickelt. Beispiele hierfür sind die Steuerung von Lampen, Türschlösser und Hifi Equipment. Nicht vorgesehen sind dagegen Streaming Applications. Da Z-Wave ein proprietärer Standard ist, ist die Beschaffung der Protokollsspezifikationen im Internet nur bedingt möglich. Die originalen Spezifikationen sind über die Z-Wave Alliance nur nach dem Unterschreiben einer NDA und dem Beitritt zu eben dieser, zu bekommen. Der Mitgliedsbeitrag betragt 2500$ im Jahr. Z-Wave sendet im 900MHz ISM Band auf den Frequenzen 908.42MHz (USA), 868.42MHz (Europa), 919.82MHz (Hong Kong), 921.42MHz (Australien/Neuseeland). Die Daten werden mittels Gaussian Frequency Shift Keying (GFSK) übertragen welches durch die Glättung der Frequenzübergänge die Oberwellenanteile verringert und somit die Spektralbreite begrenzt. Die Übertragungsrate liegt bei 9.600 bit/s oder 40 kbit/s [11], wegen Regulierungsauflagen ist der Betrieb in Europa auf 1% Aktivität (duty cycle) bei 25mW beschränkt [6, Seite 51], in der USA ist dagegen der Betrieb nur mit 1mW erlaubt, dort jedoch ohne „duty cycle“ Beschränkung.[10]

2.2.1 Übertragung der Daten

Die Daten werden bei Z-Wave Manchester kodiert gesendet, dabei werden diese in 8 Bit Blocks aufgeteilt und das „Most Significant Bit“ als erstes gesendet. Der Manchester Code sorgt dabei für eine gleichstromfreie Übertragung und die Möglichkeit der Taktrückgewinnung beim Empfänger.

Auf der MAC Ebene sieht ein Paket folgendermaßen aus:

- 1 Byte Präambel
- 1 Byte Start of Frame (SOF)

- bis zu 64 Byte Daten
- 1 Byte End of Frame (EOF)

Da in dem mir vorliegenden Dokumenten leider keine genauen Informationen zu der Anzahl der Bytes vorliegen, sind die Byteangaben nur eine Interpretation der Grafik in [8, Kapitel 4 - Seite 7].
Der Transfer Layer beinhaltet die Informationen welche der Routing Layer zum Routen benötigt. Ein Paket des Transfer Layers besitzt folgende Informationen:

- 4 Byte Home ID
- 1 Byte Absender ID
- 1 Byte Frame Header
- 1 Byte Länge (bezieht sich sehr wahrscheinlich nur auf den Datenteil des Paketes)
- 1 Byte Ziel ID bei Singlecast; 1 - 232 Byte Ziel IDs für 1 - 232 Ziele bei Multicast; FFh bei Broadcast
- n Byte Daten
- 1 Byte Checksumme

Singlecast Pakete können hierbei ein Transfer Ack anfordern, welches einen erfolgreichen Empfang des Paketes bei dem nächsten Knoten signalisiert. Dieses besitzt hierbei die Nutzdatenlänge null. Zusätzlich zu einem Transfer Ack gibt es ein Routing Ack welches ein erfolgreiches Ankommen des Paketes beim Ziel signalisiert. Wie bereits auf der MAC Ebene sind die Bytelängen nur Schätzungen, die aus Zeichnungen in [7, Seite 9] und [5, Grafik 2] entnommen sind.
Die Anwendungsschicht des Netzwerkes ist zuständig für die Dekodierung und Ausführung von Kommandos. Der Anwendungsteil sieht folgendermaßen aus:

- 1 Byte Application command class; 0 - 1Fh Z-Wave Protokoll, 20h - FFh Anwendung
- 1 Byte Application command
- n Byte Kommando Parameter / Daten

Insgesamt ergibt sich somit ein Header von 14 Byte, in welchem aber schon ein Kommando gesendet werden kann, daraus folgt dass der minimale Overhead eine Paketes bei 21% liegt (67 Byte maximale Paketlänge).

2.2.2 Gerätetypen

Z-Wave kennt zwei grundsätzliche Arten von Geräten, Controller und Slaves. Controller organisieren das Netzwerk und initialisieren auszuführende Kommandos. Nur Controller besitzen eine Routingtabelle. Slaves hingegen führen Kommandos aus und antworten auf diese. Beide Gerätearten können Daten weiterleiten falls es keine direkte Verbindung zwischen den kommunizierenden Knoten gibt, Slaves müssen dies aber speziell unterstützen (siehe 2.2.4).

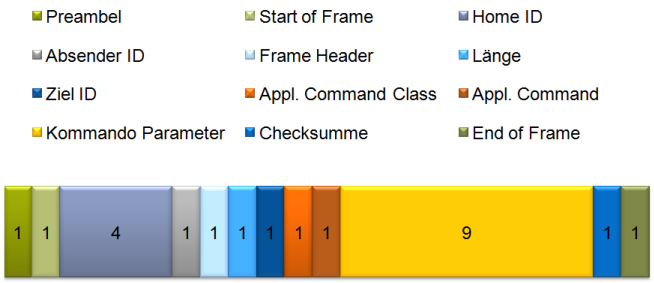


Abbildung 2: Z-Wave-Paket mit 10 Byte Nutzdaten

2.2.3 Geräterollen eines Controllers

Ein Z-Wave Controller kann im Netzwerk verschiedene Aufgaben erledigen. Der erste Controller im Netzwerk ist immer automatisch der „Master“ Controller. Er kann Knoten ins Netzwerk bringen oder diese vom Netzwerk ausschließen, dadurch hat er immer die aktuellsten Informationen über die Netzwerktopologie. Controller welche später hinzugefügt werden, werden „secondary“ Controller genannt. Sie können jedoch nicht wie der Master Controller neue Knoten ins Netzwerk einbinden. Des Weiteren gibt es noch sechs spezielle Arten von Controllern. [7, Seite 3 bis 5]

Portable Controller. Ein „Portable Controller“ ist ein Controller welcher seine Position im Z-Wave Netzwerk ändern kann. Er besitzt spezielle Mechanismen, um seine eigene Position im Netzwerk zu bestimmen und dadurch die kürzeste Route für die Daten zu wählen. Ein Beispiel hierfür wäre eine Fernbedienung.

Static Controller. Ein „Static Controller“ ist das Gegenteil eines portablen Controllers, er darf seine Position nicht ändern. Vorteil hiervon ist, dass routende Slaves immer sicher sein könne, dass dieser in Empfangsreichweite ist, um Statusinformationen zu empfangen. Typischerweise ist ein statischer Controller ein secondary Controller. Beispiel hierfür wäre ein fest eingebautes Steuerpult.

Static Update Controller. In einem Z-Wave Netzwerk gibt es optional einen „Static Update Controller“ (SUC). Dieser bekommt Updates von Routinginformationen vom primären Controller und kann diese anschließend an andere Knoten weitergeben. Ein „Static Update Controller“ wird vom primären Controller bestimmt, es darf in einem Netzwerk immer nur einen geben.

SUC ID Server. Der „SUC ID Server“ (SIS) ist eine Erweiterung des SUC (siehe 2.2.3). Der SIS ermöglicht es anderen Controllern weitere Knoten ins Netzwerk aufzunehmen oder auszuschließen. Diese werden dann Inclusion Controllers genannt und können im Namen des SIS Routingupdates erstellen. Um Eineindeutigkeit der Routinginformationen sicherzustellen, werden diese entweder mittels Timestamp

der Hinzufüge-/Entfernaktion oder des Updates vom SIS gekennzeichnet.

Installer Controller. Ein „Installer Controller“ ist ein portabler Controller, welcher weiterführende Diagnosefunktionen bereitstellt. Meist ist dies ein Controller, welcher von Installateuren zur Einrichtung des Netzwerkes genutzt wird.

Bridge Controller. Ein „Bridge Controller“ ist ein erweiterter statischer Controller. Er stellt Funktionen bereit, welche eine Kopplung des Z-Wave Netzes mit anderen Netzen erlaubt. Dafür speichert er Informationen zu den Knoten und kann 128 virtuelle Slaves kontrollieren. Ein virtueller Slave dient zur Repräsentation eines Knoten von einem anderen Netzwerk in das Z-Wave Netzwerk. Beispiel für einen „Bridge Controller“ ist ein Z-Wave - ZigBee Router.

2.2.4 Geräterollen eines Slaves

Slaves sind die Knoten in einem Z-Wave Netzwerk, welche die Anweisungen entgegennehmen und ausführen. Sie können nicht von selbst einen Datentransfer initialisieren. Ein Beispiel für einen Slave ist ein Dimmer. Zusätzlich gibt es noch vier spezielle Slave Arten:[7, Seite 5/6]

Routing Slave. Ein „Routing Slave“ besitzt die selben Funktionen wie ein normaler Slave, kann zusätzlich aber auch unaufgefordert Nachrichten an andere Knoten im Netzwerk schicken. Dazu besitzt er einen Speicher für statische Routen zu anderen Knoten welche die Nachrichten empfangen sollen. Wenn ein „Routing Slave“ mit einer Batterie betrieben wird, ist dieser dabei selber nicht in Routen anderer Knoten enthalten, andernfalls kann er Pakete bei Bedarf weiterleiten. Beispiele hierfür sind Temperaturfühler oder Bewegungsmelder, da diese selbständig Nachrichten initialisieren müssen.

Frequently Listening Routing Slave. Ein „Frequently Listening Routing Slave“ (FLiRS) ist ein „Routing Slave“ der mit Batterie betrieben wird. Um Strom zu sparen, wacht der Slave periodisch auf und versucht ein „wakeup beam“ zu empfangen, welches anzeigt, dass Daten für ihn vorhanden sind. Beispiel hierfür wäre eine drahtlose Klingel.

Enhanced Slave. Ein „Enhanced Slave“ ist ein um ein EEPROM und eine Echtzeituhr erweiterter „Routing Slave“. Eine Anwendung hierfür könnte zum Beispiel eine Wetterstation sein.

Zensor Net Routing Slave. Ein „Zensor Net Routing Slave“ ist ein FLiRS Knoten mit zwei speziellen zusätzlichen Funktionen. Zum einen kann er sich in ein „Zensor Net“ integrieren, zum anderen besitzt er die nötigen Funktionen, um Nachrichten im „Zensor Net“ zu verbreiten. Dies ist in 2.2.6 genauer beschrieben.

2.2.5 Topologien

In Z-Wave wird nicht wie bei ZigBee eine Unterscheidung zwischen verschiedenen Topologien gemacht. Während der Einrichtung des Netzwerkes werden die möglichen Netzwerkpfade zwischen den Geräten ermittelt und anschließend an die Kontroller und „Routing Slave“ verteilt, welche anschließend mittels Source Routing Pakete versenden können. Die genaue Topologie des Netzwerkes spielt hierbei keine Rolle, da nur versucht wird, ein vermaschtes Netzwerk so zu erstellen, dass jeder Knoten erreichbar ist. Weiterhin wird versucht, Netzwerkpfade möglichst nicht über batteriebetriebene Knoten zu legen, sodass diese eine länger Laufzeit besitzen.

2.2.6 Energiesparmöglichkeit ZENSOR NET

Das „Zensor Net“ ist eine Erweiterung von Z-Wave, welche speziell auf die Bedürfnisse von batteriebetriebenen Sensornetzwerken zurechtgeschnitten ist. Zensor Net bietet zum Stromsparen drei Techniken an, den „Zensor Net Beam“, das „Zensor Net Binding“ und das „Zensor Net Flooding“. [8, Seite 15 bis 21]

Zensor Net Beam. Die „Zensor Net Beam“-Technik ist dafür zuständig, schlafenden Knoten anzuzeigen, dass Nachrichten für sie vorhanden sind. Dafür wird vom Sender ein „Beam“ über einen längeren Zeitraum gesendet. Ein Knoten mit FLiRS oder Zensor Net Technik wacht je nach Einstellung alle 250ms oder 1000ms für 4,5ms auf. Falls in dieser Zeit solch ein Beam erkannt wird, bleibt der Knoten im aktiven Zustand und kann die Nachricht, welche am Ende des Beams gesendet wird, empfangen.Falls der Knoten einen genauen Timer zum Aufwachen (WUT) besitzt, kann zusätzlich Energie gespart werden, indem der Empfänger die im Beam enthaltenden Zeitinformationen auswertet. Er kann sich so erneut schlafen legen, um zum Zeitpunkt der Daten wieder aufzuwachen und diese dann zu empfangen. Die Aufwachzeiten sind 250ms und 1000ms und decken die meisten Einsatzszenarien ab. Gründe für eine 250ms Aufwachzeit sind Anwendungen, bei denen der Benutzer eine direkte Rückmeldung erwartet, Beispiel hiefür ist eine Türklingel. Nachteil ist, dass eine 250ms Aufwachzeit 4 mal mehr Energie verbraucht als ein 1000ms Aufwachzeit. Anwendung für eine 1000ms Aufwachzeit sind zum Beispiel Langzeitmessungen mit Sensoren, bei denen die Verzögerung keine Auswirkungen hat, wie es bei Rauchmeldern der Fall ist.[8, Seite 15 bis 20]

Zensor Net Binding. Das „Zensor Net Binding“ sorgt dafür das alle Knoten, welche „Zensor Net“ unterstützen, sich gegenseitig kennen. Dies ist wichtig um sicherzustellen, dass beim Routing von Paketen der letzte Router „Beaming“ unterstützt. Weiterhin wird mit „Zensor Net Binding“ eine weitere Home ID an teilnehmende Knoten verteilt, sodass diese dann beim „Zensor Net Flooding“ wissen, dass sie das Paket weiterleiten sollen.[8, Seite 20]

Zensor Net Flooding. „Zensor Net Flooding“ ist ein Mechanismus um wichtige Nachrichten mit hoher Sicherheit an ihr Ziel zu bekommen. Wichtig ist dies bei sicherheitskritischen Aufgaben wie Rauchmeldern. „Zensor Net Floo-

ding“ stellt hierbei sicher, dass der Ausfall von Knoten, sei es durch Strommangel oder das diese entfernt wurden, sich nicht negativ auswirkt. Ein spezieller Routing-Algorithmus, eine „Frame Flooding ID“ und ein „Hop Count“ stellen dabei sicher, dass es zu keiner Verstopfung im Netz kommt und normale Z-Wave Knoten noch Daten senden können.[8, Seite 20/21]

2.3 Vergleich der Funktechniken

Im Folgenden sollen nun verschiedene Szenarien durchgespielt werden, um aufzuzeigen, welche Technik in welcher Situation am Besten geeignet ist. Im Voraus ist dazu zu sagen, dass hier Abschätzungen gemacht werden, bei denen versucht wurde, einen möglichst breiten Bereich abzustecken. Alle Werte sind Werte, welche bei der Übertragung der Daten mit der Besten zur Verfügung stehenden Methode der jeweiligen Funktechnik und bei vollständig eingerichtetem Netzwerk zustande kommen können. In allen Szenarien senden die Knoten von sich aus die neuen Daten, es ist keine Steuerkomunikation notwendig. Als Repräsentanten wurden XBee® 802.15.4 (Series 1)[3] für ZigBee und das ZM2102[11] Hardware-Modul für Z-Wave gewählt

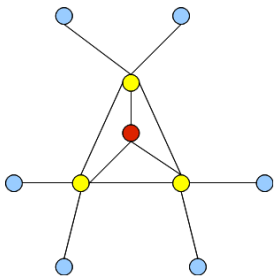


Abbildung 3: Anordnung der Knoten im Testnetz

Der Netzwerkaufbau ist dabei folgendermaßen (siehe auch Abbildung 3):

- alle Aufbauten besitzen 10 Knoten
- ein Knoten in der Mitte (rot)
- 3 Knoten jeweils in Reichweite des mittleren und der anderen Beiden (gelb)
- 6 Knoten paarweise erreichbar über die 3 Knoten (blau)

Die Konfigurationen sind:

1. Feste Installation in einem Gebäude
   - 4 Konten mit Stromversorgung; in der Mitte (rot, gelb)
   - 6 Knoten batteriebetrieben (blau)

   Dieser Aufbau entspricht z.b. einer Fabrik mit Temperatursensoren außen und Maschinen in der Mitte.
2. Feste Installation in einem Gebäude 2
   - 1 Knoten mit Stromversorgung; in der Mitte (rot)
   - 9 Knoten batteriebetrieben (gelb, blau)

   Entspricht z.B. einer Steueranlage für eine Heizung in einem Gebäude mit Bedieneinheit in der Mitte. Die Routerknoten müssen nun die Daten von 3 Knoten senden und von 2 Knoten empfangen.
3. Feste Installation ohne externen Strom. Entspricht z.B. einer Messanlage für Temperatur in einem Wald.

Der Unterschied zwischen den Konfigurationen ist also, dass Router in 2 und 3 auch mit Batterie betrieben werden. In den Tabellen werden nur die Änderungen zu den vorherigen Konfigurationen gezeigt, die Slaves aus Konfiguration 1 verbrauchen in Konfiguration 2 weiterhin den in Konfiguration 1 berechnetet.

Die Szenarien sind:

1. Szenario hohe Datenrate:
   Durchgehend werden viele Daten gesendet, z.B. Audioüberwachung oder größere Messanlage mit Echtzeiterfassung - Pro Knoten 3 kbit/s.
2. Szenario mittlere Datenrate:
   Regelmäßig werden größere Datenpakete versendet, z.B. mehrere Sensoren an jedem Knoten - Pro Knoten 1 Paket pro Sekunde mit 40 Bit.
3. Szenario niedrige Datenrate:
   Selten werden kleine Datenpakete gesendet, z.B. ein Sensor für Temperatur / Schalter pro Knoten.- Pro Knoten 1 Paket pro Stunde.

Als Stromversorgung kommen folgende Batterien vor (siehe Tabelle 1 und 2), diese sind Mittelwerte der möglichen Batterietypen um eine ungefähre Einordnung der Batterien zu ermöglichen. Bei wiederaufladbaren Batterien wurde eine Energie von 1800mAh bei 3V = 5,4Wh angesetzt. Da unterschiedliche Batterien bei unterschiedlichen Spannungen arbeiten, müssen im konkreten Fall noch Wandlerverluste eingerechnet werden. Da sich diese je nach Aufwand der Schaltung stark unterscheiden, werden hier keine Wandlerverluste eingerechnet. Ein weiteres Problem von Batterien ist, dass sie bei großer Belastung weniger Energie abgeben können. Man kennt dies von den Taschenlampen, die nach ein bisschen Abwarten wieder funktionieren. Da die Funkanwendungen aber höchstens Strom im Bereich von 0.1C, d.h. einem Zehntel der Nennkapazität, brauchen, spielt dies hier keine Rolle. Hersteller von Batterien geben die Kapazität bei 0,1C Entladungsstrom an. Der Wirkungsgrad gibt an wie gut aufgenommene Energie wieder abgegeben werden kann, dies ist interessant, falls als Stromversorgung erneuerbare Energiequellen genutzt werden sollen, dort ist auch der Innenwiderstand mit inbegriffen. Quelle für die Daten sind [1], [2], [4] und auf Wikipedia die englischen Seiten der jeweiligen Batterien. Zum Berechnen der Laufzeiten wurde ein vereinfachter Ansatz gewählt, welcher auf der Formel für die Annuitätendarlehen mit Zinseszins beruht (siehe [9]). Noch mal als Warnung, diese Berechnungen hier können nur als grobe Abschätzung zur Wahl eines Batterietypen angesehen werden, bei einem Einsatzgebiet im Freien im Bereich von 5 bis $20C^{\circ}$ schwanken die Selbstentladungsraten bereits um bis zu dem Fakor 10.

2.3.1 Erklärungen zu den Tabellen

Jede der Tabellen 3, 4 und 5 beinhaltet die Auswertung zu den drei Szenarien in der entsprechenden Konfiguration. Jedes Szenario besitzt eine zweigeteilte Tabelle. Der obere Teil

Typ	Kapazität	Gewicht	Selbstentladung
2 * AAA	2,6 W	22g	1%
2 * AA	5,6 W	44g	1%

Tabelle 1: Nicht wiederaufladbare Batterien

Typ	Blei Akku	NiCd	NiMh
Größe in cm^3	$80cm^3$	$54cm^3$	$24cm^3$
Gewicht in g	135g	77g	60g
Selbstentladung im Monat	8%	15%	30%
Wirkungsgrad	70%	80%	65%
Energiedichte Wh/kg	40	70	90
Energiedichte Wh/l	67	100	220

Typ	Lithium-Ion	Li-Polymer
Größe in cm^3	$17cm^3$	$18cm^3$
Gewicht in g	31g	30g
Selbstentladung im Monat	8%	5%
Wirkungsgrad	90%	90%
Energiedichte Wh/kg	170	180
Energiedichte Wh/l	305	300

Tabelle 2: wiederaufladbare Batterien mit 1,8Ah bei 3V [1][2][4]

beinhaltet allgemeine Informationen wie Paketlänge, Paketzahl in einem Sendvorgang, die Zeit zwischen zwei Sendevorgängen, Overhead, die Sendezeit eines Sendevorgangs und den gemittelten Stromverbrauch. Mit Hilfe dieser Informationen ist es möglich die Laufzeit nicht aufgelisteter Batterien abzuschätzen. Zudem kann man über die Paketlänge das Szenario auswählen, welches dem gewünschten Einsatzgebiet am besten entspricht. Im unteren Teil sind die jeweiligen Laufzeiten der Batterietypen in Monaten aufgelistet. Anzumerken zu Tabelle 4 und 5 ist noch, dass die Reaktionszeit bei ZigBee beim 3 Szenario bei 5 Minuten liegt, sodass die neuen Messwerte erst nach 5 Minuten beim Kontroller ankommen. Gut zu sehen ist, dass Z-Wave durch den kleineren Overhead Bandbreite spart, im Besonderen zu sehen in Szenario 3, Konfiguration 1. Durch die niedrigere Datenrate und die kürzeren Schlafphasen der Router im Gegensatz zu Zigbee, schneidet es aber dann schlechter in den anderen beiden Konfigurationen ab. Natürlich setzt dies eine funktionierenden Beacon-Mode beim Zigbee voraus.

3. FAZIT UND AUSBLICK

Z-Wave und Zigbee sind zwei brauchbare Funktechniken, Zigbee sticht durch die größere Übertragungsrate, Z-Wave durch den weniger genutzten Frequenzbereich heraus. Zudem scheint Z-Wave bei den Routingmechanismen besser aufgestellt zu sein, ZigBee bei dem Beacon-Mode, der durch Schlafphasen von mehr als 2 Minuten große Einsparungen möglich macht. Der Unterschied der beiden Funktechniken zeigt sich vor allem bei Langzeitanwendungen, bei denen die richtige Stromversorgung aber auch wichtigere Rolle spielt, da Selbstentladung dort einen großen Anteil besitzt. Die Auswahl der richtigen Funktechnologie muss mit Blick auf die Anwendung ausgewählt werden, keine der beiden Standards ist in allen Situationen die optimale Wahl.

4. LITERATUR

[1] batteryholders.org. alkaline batteries. `http://www.batteryholders.org/alkaline_batteries.shtml`, Nov. 2007.

[2] I. Buchmann. Welches ist die beste Batterie? `http://www.batteryuniversity.com/partone-3-german.htm`, July 2003.

[3] DIGI SERVICE AND SUPPORT. Product Datasheet. `http://www.digi.com/pdf/ds_xbeemultipointmodules.pdf`, 2008.

[4] M. Frehner. Akkulexikon. `http://www.funkcom.ch/akkuinfos.htm`, Nov. 2007.

[5] M. T. Galeev. Catching the Z-Wave. `http://www.embedded.com/columns/technicalinsights/193101000?_requestid=119705`, Oct. 2006.

[6] A. S. Gerald Kupris. *ZigBee: Datenfunk mit IEEE 802.15.4 und ZigBee*. Franzis Verlag GmbH, Gruber Str. 46a; D - 85586 Poing, 2007.

[7] JFR. Z-Wave Protocol Overview (SDS10243). `http://www.eilhk.com/en/product/Datasheet/Zensys/SDS10243-2-Z-WaveProtocolOverview.pdf`, Apr. 2006.

[8] A. J. F. SML. Z-Wave Node Type Overview and Network Installation Guide (INS10244). `http://support.zen-sys.com/modules/iaCM-DocMan/download.php?get=2747d12409c4070fe4e8e8f715030afe`, Dec. 2008.

[9] Stargamer - Wikipedia. Annuitätendarlehen. `http://de.wikipedia.org/w/index.php?title=Annuit%C3%A4tendarlehen&stableid=74840084#Bestimmung_der_Laufzeit`, May 2010.

[10] Wikipedia. Z-Wave. `http://en.wikipedia.org/w/index.php?title=Z-Wave&oldid=374626805`, July 2010.

[11] Zensys A/S. ZM2102 Datasheet; Integrated Z-Wave RF Module. `http://www.eilhk.com/en/product/Datasheet/Zensys/DSH10229-11%20-%20Datasheet,%20ZM2102%20Z-Wave%20Module.pdf`, 2008.

Szenario 1	ZigBee	Z-Wave
Paketlänge in Bit	3000	3000
Paketzahl	28	57
Zeit zw. Sendungen in s	1	1
Gesamtbit	7928	9384
Overhead in %	62%	68%
Übertragungszeit in s	3,17E-02	2,35E-01
Stromverbrauch in mA	1,4367	8,4475
2*AAA in Monaten	0,8	0,3
2*AA	1,8	0,3
Blei Akku	1,6	0,3
NiCd	1,4	0,3
NiMh	1,2	0,2
Litium-Ion	1,6	0,3
Litiumpolymer	1,6	0,3

Szenario 2	ZigBee	Z-Wave
Paketlänge in Bit	40	40
Paketzahl	1	1
Zeit zw. Sendungen in s	1	1
Gesamtbit	216	152
Overhead in %	81%	74%
Übertragungszeit in s	8,64E-04	3,80E-03
Stromverbrauch in mA	0,0489	0,1393
2*AAA in Monaten	21,9	8,2
2*AA	42,3	17,0
Blei Akku	19,5	10,7
NiCd	13,3	8,0
NiMh	7,8	5,2
Litium-Ion	19,5	10,7
Litiumpolymer	24,7	12,5

Szenario 3	ZigBee	Z-Wave
Paketlänge in Bit	40	40
Paketzahl	1	1
Zeit zw. Sendungen in s	3600	3600
Gesamtbit	216	152
Overhead in %	81%	74%
Übertragungszeit in s	8,64E-04	3,80E-03
Stromverbrauch in mA	0,0100	0,0029
2*AAA in Monaten	78,5	163,6
2*AA	127,1	229,1
Blei Akku	36,5	51,0
NiCd	22,5	30,0
NiMh	12,1	15,6
Litium-Ion	36,5	51,0
Litiumpolymer	50,7	74,0

Tabelle 3: Konfiguration 1, Laufzeit der Knoten

Szenario 1	ZigBee	Z-Wave
Paketlänge in Bit	15000	15000
Paketzahl	136	284
Zeit zw. Sendungen in s	1	1
Gesamtbit	38936	46808
Overhead in %	61%	68%
Übertragungszeit in s	1,56E-01	1,17E+00
Stromverbrauch in mA	7,0169	42,1268
2*AAA in Monaten	0,2	0,0
2*AA	0,4	0,1
Blei Akku	0,3	0,1
NiCd	0,3	0,1
NiMh	0,3	0,0
Litium-Ion	0,3	0,1
Litiumpolymer	0,3	0,1

Szenario 2	ZigBee	Z-Wave
Paketlänge in Bit	200	200
Paketzahl	4	5
Zeit zw. Sendungen in s	1	1
Gesamtbit	904	760
Overhead in %	78%	74%
Übertragungszeit in s	3,62E-03	1,90E-02
Stromverbrauch in mA	0,3354	0,6865
2*AAA in Monaten	3,5	1,7
2*AA	7,4	3,7
Blei Akku	5,6	3,1
NiCd	4,6	2,7
NiMh	3,3	2,1
Litium-Ion	5,6	3,1
Litiumpolymer	6,2	3,3

Szenario 3	ZigBee	Z-Wave
Paketlänge in Bit	200	200
Paketzahl	4	5
Zeit zw. Sendungen in s	3600	3600
Gesamtbit	904	760
Overhead in %	78%	74%
Übertragungszeit in s	3,62E-03	1,90E-02
Stromverbrauch in mA	0,0141	0,1062
2*AAA in Monaten	61,2	10,7
2*AA	103,6	21,7
Blei Akku	32,6	12,7
NiCd	20,4	9,3
NiMh	11,2	5,9
Litium-Ion	32,6	12,7
Litiumpolymer	44,6	15,2

Tabelle 4: Konfiguration 2, Stromverbrauch der Router

Szenario 1	ZigBee	Z-Wave
Paketlänge in Bit	27000	27000
Paketzahl	254	516
Zeit zw. Sendungen in s	1	1
Gesamtbit	71704	84792
Overhead in %	62%	68%
Übertragungszeit in s	2,87E-01	2,12E+00
Stromverbrauch in mA	12,9139	76,3100
2*AAA in Monaten	0,1	0,0
2*AA	0,2	0,0
Blei Akku	0,2	0,0
NiCd	0,2	0,0
NiMh	0,2	0,0
Litium-Ion	0,2	0,0
Litiumpolymer	0,2	0,0

Szenario 2	ZigBee	Z-Wave
Paketlänge in Bit	360	360
Paketzahl	11	12
Zeit zw. Sendungen in s	1	1
Gesamtbit	2296	1704
Overhead in %	84%	79%
Übertragungszeit in s	9,18E-03	4,26E-02
Stromverbrauch in mA	0,8365	1,5360
2*AAA in Monaten	1,4	0,8
2*AA	3,0	1,7
Blei Akku	2,6	1,5
NiCd	2,3	1,3
NiMh	1,8	1,1
Litium-Ion	2,6	1,5
Litiumpolymer	2,7	1,5

Szenario 3	ZigBee	Z-Wave
Paketlänge in Bit	360	360
Paketzahl	11	12
Zeit zw. Sendungen in s	3600	3600
Gesamtbit	2296	1704
Overhead in %	84%	79%
Übertragungszeit in s	9,18E-03	4,26E-02
Stromverbrauch in mA	0,0144	0,1064
2*AAA in Monaten	60,4	10,7
2*AA	102,3	21,7
Blei Akku	32,4	12,7
NiCd	20,3	9,3
NiMh	11,1	5,8
Litium-Ion	32,4	12,7
Litiumpolymer	44,2	15,1

Tabelle 5: Konfiguration 3, Stromverbrauch Kontroller

ISBN 3-937201-16-5

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)