Network Architectures
and Services
NET 2012-08-2

SN
SS2012

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

Summer Semester 2012

Munich, Germany, 23./24.08.2012

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 2012-08-2

SN
SS 2012

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

Sommersemester 2012

München, Germany, 23./24.07.2011

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

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

Technische Universität München

SN SS 2012
Proceedings of the Seminar Sensor Nodes – Operation, Network and Application (SN)

Editors:

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

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

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

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

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

Cataloging-in-Publication Data

SN SS 2012
Proceedings of the Seminar Sensor Nodes – Operation, Network and Application (SN)
Munich, Germany, 23./24.07.2012
Georg Carle, Corinna Schmitt, Alexander Klein, Uwe Baumgarten, Christoph Söllner
ISBN: 3-937201-28-9
DOI: 10.2313/NET-2012-08-2
ISSN: 1862-7803 (print)
ISSN: 1862-7811 (electronic)
Lehrstuhl Netzarchitekturen und Netzdienste (I8) NET 2012-08-2
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 2012 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:

- Vergleich von Betriebssystemen: Mantis OS, cocoOS, und .NET Micro Framework
- Vergleich der Betriebssysteme TinyOS und Contiki
- Reprogrammierungstechniken für drahtlose Sensornetze
- Energie-effiziente Kommunikation in drahtlosen Sensornetzen
- Datenverarbeitungstechnik für zeitliche Ordnung der empfangenen Pakete
- TLS Lösung für Sensornetze
- Wireless Body Area Networks (WBAN)
- Umweltmonitoring: SensorScope

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, August 2012


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 2012.

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:

- Comparison of Operating Systems: Mantis OS, cocoOS, and .NET Micro Framework
- Comparison of Operating Systems TinyOS and Contiki
- Reprogramming Techniques for Wireless Sensor Networks
- Energy-efficient Communication in Wireless Sensor Networks
- Packet Reordering on Gateway as a Representative of Data Processing Techniques
- TLS Solutions for WSNs
- Wireless Body Area Networks (WBAN)
- Environmental Monitoring with SensorScope

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, August 2012

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/ss12/seminare/

Inhaltsverzeichnis

Session 1: Grundlagen

Session 2: Kommunikation und Verarbeitung

Session 3: Sicherheit

Session 4: Anwendungen

Ein Vergleich von Betriebssystemen für Sensorknoten: Mantis OS, cocoOS und .NET Micro Framework

Maximilian Weber
Betreuer: Christoph Söllner
Hauptseminar Sensorknoten - Betrieb, Netze & Anwendungen
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitekturen
Fakultät für Informatik, Technische Universität München
Email: webermax@in.tum.de

KURZFASSUNG

In diesem Papier wird ein Vergleich der Betriebssysteme Mantis OS, cocoOS und .NET Micro Framework vorgenommen. Dabei wird der Einsatz auf Sensorknoten hinsichtlich Effizienz, Sicherheit und Benutzungskomfort beleuchtet. Die jeweiligen Vor- und Nachteile werden diskutiert und eine abschließende Bewertung vorgenommen. Für die optimale Wahl eines der behandelten Betriebssysteme ist der konkrete Anwendungsfall ausschlaggebend.

Schlüsselworte

Sensorknoten, Sensornetz, Betriebssystem, Mantis OS, cocoOS, .NET Micro Framework

1. EINLEITUNG

Die systematische Erfassung großer Datenvolumen innerhalb technischer und biologischer Systeme findet zunehmend Verbreitung. So werden nicht nur Fußballspieler während eines gesamten Turniers möglichst ganzheitlich überwacht, auch Smartphones generieren Positions- und Verbindungsdaten in erheblichem Ausmaß an zentraler Stelle. Ebenso werden mittels Sensornetzen Frühwarnsysteme für Erdbeben realisiert. Doch damit sind die Anwendungsmöglichkeiten vernetzter Sensorknoten bei Weitem nicht ausgeschöpft. Beispielsweise kann das bereits 1999 von Kahn [4] beschriebene Smart Dust dank Innovationen aus der Nanotechnologie heute längst realisiert werden. Dabei kommen winzige Sensorknoten zum Einsatz, die nicht größer als ein Kubikmillimeter sind. Lee [7] verwendet hierzu optische Kommunikationswege und Solarzellen zur Energiegewinnung.

Die vom gewöhnlichen Rechnerbetrieb abweichenden, äußeren Bedingungen bei derartigem Einsatz von IT Systemen erfordern besondere Softwarelösungen. Denn die Beschränkungen der Leistungsfähigkeit durch den extrem verkleinerten Formfaktor eines Sensorknotens von meist wenigen Zentimetern erzwingt eine wesentlich sparsamere Art der Ressourcenverwaltung und damit der Programmausführung. Für die Lösung der hieraus resultierenden Problemstellungen wie der geringeren Bandbreite bei der Datenübertragung oder der niedrigen Taktraten von Mikrocontrollern gibt es grundsätzlich verschieden etablierte Ansätze für die Schaffung von Rahmenbedingungen für Anwendungsprogramme.

Die zahlreich vorhandenen Umsetzungen werden unterschiedlichen Sicherheitsanforderungen bei Kommunikation und Programmausführung gerecht und unterstützen den Entwickler mehr oder weniger ausführlich mit Dokumentationsmaterial, was im Folgenden anhand der konkreten Ausprägungen Mantis OS, cocoOS und .NET Micro Framework erläutert wird.

2. BEGRIFFSKLÄRUNGEN

Ein Betriebbsystem für Sensorknoten weist grundlegende Unterschiede zu konventionellen Betriebssystemen, wie sie in Desktopumgebungen verwendet werden, auf. Diesem neuen Kontext wird der Begriff des Betriebssystems entsprechend angepasst. Dieser Definition wird außerdem die Klärung der benötigten Begriffe Sensorknoten und Sensornetz vorweggenommen.

2.1 Definition eines Sensorknotens

Sensorknoten sind auf das Wesentliche reduzierte Rechenmaschinen. Sie verfügen, wie zum Beispiel das Modell TELOSB von Crossbow [12], in der Regel mindestens über einen Mikrocontroller, eine Programmierschnittstelle, eine (Funk-)Netzwerkeinheit und die Möglichkeit, Sensoren für Messungen anzubringen, beispielsweise um die Umgebungstemperatur, die Lichtstärke oder Beschleunigungen zu erfassen. Üblicherweise werden diese mittels einer mobilen Stromquelle betrieben. Dabei kann es sich um eine Batterie handeln oder sogar um eine dauerhafte Energieversorung wie sie beispielsweise mittels photovoltaischer Zellen realisiert werden kann.

2.2 Definition eines Sensornetzes

Die Zusammenschaltung einer Anzahl von Sensorknoten erfolgt nur bedingt analog der Vernetzung konventioneller Rechner. Besonders den Einschränkungen hinsichtlich Reichweite und Bandbreite muss Rechnung getragen werden. Dabei kann zwischen zentral vernetzten Netzwerken, beispielsweise mit Sterntopologie, und vermaschten Sensornetzen ohne feste Infrastruktur unterschieden werden. Die logische Netzwerkstruktur kann abhängig sein von den physikalischen Bedingungen für die Kommunikationswege innerhalb des Netzwerks (siehe Abbildung 1) [9]. So sind Sensornetze mit mobilen Knoten, welche sogar den Kommunikationspartner wechseln können, in ihrer physikalischen Topologie flexibel, was aber nicht zwingend eine Veränderung der logischen Topologie nach sich ziehen muss.

2.3 Definition eines Betriebssystems für Sensorknoten

Im Gegensatz zu Desktopsystemen wird auf einem Sensorknoten in der Regel keine separate Installation für das Be-

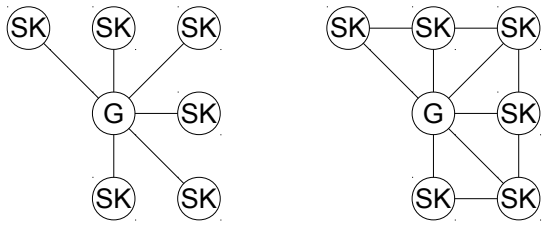


Abbildung 1: Physikalische Sterntopologie (links) und logische Vermaschung (rechts) von Sensorknoten (SK) und Gateway (G)

triebssystem durchgeführt. Stattdessen wird ein Rahmenprogramm eingesetzt, welches Anwendungsprogrammen eine Hardwareabstraktion liefert und die Zuteilung von Resourcen wie Rechenzeit und Sendezeit übernimmt. Eingebettet in diesen Rahmen wird somit das Programm gemeinsam mit dem Betriebssystem auf den Sensorknoten übertragen.

3. SPEZIELLE ANFORDERUNGEN

Um trotz der Einschränkungen auf Seiten der Hardware möglichst flexibel zu bleiben, werden die speziellen Betriebssysteme, beziehungsweise Frameworks, den restriktiven Anforderungen angepasst, um einen komfortableren Umgang mit den nachfolgenden Problemstellungen zu ermöglichen. Damit müssen häufig auftretende Probleme für verschiedene Programme nicht mehrfach gelöst werden.

3.1 Energieverbrauch

Der wichtigste Aspekt für den Produktivbetrieb eines Sensorknotens ist die Leistungsaufnahme. Einerseits sollen Daten möglichst großflächig erfasst werden, was eine relativ hohe Übertragungsleistung der verwendeten Funkeinheiten erfordert. Andererseits sollen Daten üblicherweise über einen möglichst langen Zeitraum hinweg erfasst werden, um die Aussagekraft zu erhöhen. Diesen Widerspruch gilt es durch einen effektiven Bereitschaftszustand aufzulösen, welchem sich das Gesamtkonzept für den Softwarebetrieb unterordnet.

3.2 Rechenleistung

Die Rechenleistung verbreiterter Mikrocontroller für Sensorknoten liegt weit unterhalb derjeniger gängiger Desktop CPUs. Dies ist sowohl der lediglich knapp verfügbaren Energieressource geschuldet als auch der Wirtschaftlichkeit. Damit werden beispielsweise Kontextwechsel beim Übergang zu einem anderen Prozess teuer, was ein geeignetes Scheduling Verfahren erfordert.

3.3 Sicherheit

Nicht nur hinsichtlich des Datenschutzes ist die Anfälligkeit eines Sensornetzes für Manipulationen, Ausspähungen oder Überlastung von Bedeutung. So gilt es ebenso, sicherheitskritische Systeme hinreichend vor Ausfällen zu schützen und diese zu erkennen.

3.4 Multitasking

Ein sinnvoller Ansatz für den Betrieb eines Sensorknotens in einem Netzwerk ist die parallele Ausführung mehrerer Aufgaben. Dies kann auf unterschiedliche Art, zum Beispiel mittels der Nutzung von Semaphoren realisiert werden.
Vor allen Dingen in mobilen Sensornetzen mit einer Vielzahl gleichartiger Knoten kann die Idee des Multitaskings gut ausgenutzt werden. Krüger [5] hat dies mit der funktionalen und geographischen Gruppierung von Sensoren gezeigt. Vor allen Dingen wenn mehrere Sensoren je Knoten verwendet werden ist es notwendig, die Messwerte des einen Sensors zu erfassen, während die Daten des anderen Sensors bereits ausgewertet und übertragen werden.

4. MANTIS OS

Nach Aufgaben abgegrenzte Schichten bilden das Modell für das Mantis OS-Design.
Mantis OS bietet weitreichende Möglichkeiten für Multitasking und ist speziell für den Einsatz in Sensornetzen ausgelegt. Das Betriebssystem ist unter einer GNU General Public License (GPL) kompatiblen Lizenz frei verfügbar. Weiterhin ist die gcc Toolchain für den Übersetzungsvorgang ausreichend.

4.1 Dokumentation

Bhatti [1] legt in einer ausführlichen Veröffentlichung die Grundprinzipien des Frameworks dar und die Projekthomepage [20] sammelt weitere Publikationen über Mantis OS. Eine anschauliche Dokumentation ist dort ebenfalls zu finden. Diese stellt eine gelungene Ergänzung zur ausführlichen Online-Beschreibung der Mantis API dar und beschreibt den Installationsvorgang des Systems auf der verwendeten Workstation sowie den schematischen Aufbau der vorhandenen Schichten.

4.2 Hardwareunterstützung

Durch die Orientierung am POSIX-Standard erzielt Mantis OS eine hohe Hardware-Kompatibilität. Zur weiteren Vereinfachung wird von Bhatti eine Referenzimplementierung, der *MANTIS hardware nymph sensor node* [1], vorgestellt. Diese besteht aus einem single-board Design mit einem Atmel Atmega128(L) Mikrocontroller sowie einer analogen Sensorschnittstelle. Der Nymph Sensorknoten ist als Ausgangspunkt für individuelle Umsetzungen erhältlich.
Weiterhin kann Mantis OS auf verbreiteten Plattformen wie MICA und TELOS Motes eingesetzt werden [20].

4.3 Architektur

Das Mantis System ist geprägt von einer nicht-strikten Schichtenarchitektur (siehe Abbildung 2) [21]. Den Kern bilden die Geräteschicht, die Netzwerk- und Protokollschicht sowie die Kommunikationsschicht.

4.3.1 COM Layer

Kommunikationsaufgaben werden von der entsprechenden Schicht abgewickelt. Sämtliche Kommunikationsgeräte werden mit ihrer Hilfe angesprochen. Dazu zählen sowohl Drahtlosverbindungen als auch die Datenübetragung über weitere, serielle Schnittstellen.
Dabei wird nicht nur das Senden und Empfangen organisiert, sondern auch die konkrete Behandlung der Zwischenspeicher für Lese- und Schreibvorgänge. Damit kann auch das Mantis Terminal angesprochen werden.

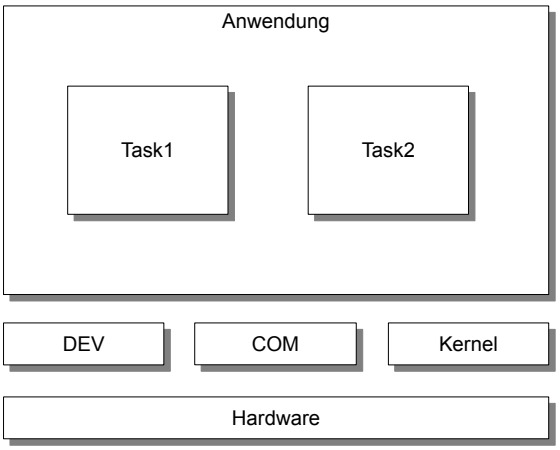


Abbildung 2: Mantis OS Architektur

4.3.2 DEV Layer

Eine weitere Hardwareabstraktion stellt die Geräteschicht dar. Sie ist aber nicht für die Nutzung aller Geräte notwendig, wie die Bezeichnung vermuten lässt, sondern bildet Ein- und Ausgabegeräte wie Sensoren oder den Speicher ab.
Bevor auf ein solches Gerät zugegriffen werden kann, muss es für den entsprechenden Vorgang reserviert werden und anschließend wieder freigegeben werden.
Auf Basis dieser Funktionalität können Gerätetreiber implementiert werden. Dies geschieht durch Programmierung der Schreib- und Lesevorgänge für das jeweilige Gerät innerhalb einer Funktion, deren Bezeichnung einem festgelegten Schema folgt.

4.3.3 NET Layer

Die Protokollschicht wird im NET Layer verankert und greift auf die Kommunikationsschicht zurück, um benutzerdefinierte Protokolle umzusetzen. So können beispielsweise verschiedene Routing Protokolle implementiert werden.

4.3.4 System Management

Auf Systemressourcen kann mittels weiterer Schnittstellen zugegriffen werden. Das Mantis System stellt auf diese Weise die Funktionalität zur Verwaltung von Threads und deren Synchronisation bereit. Für letzteres kann neben einem Mutex auch ein Semaphor verwendet werden.
Es stehen weiterhin Timer zur Festlegung von Standby-Zeiten zur Verfügung. Auch die zeitliche Planung der einmaligen Ausführung kleinerer Aufgaben ist hiermit möglich ohne größere Blöcke wie Threads benutzen zu müssen.

4.4 Programmierung

Das Mantis Betriebssystem kommt der Idee eines konventionellen Betriebssystems sehr nahe indem es dem C Programmierer eine Unix-ähnliche Umgebung bereitstellt.
Die wichtigsten Schnittstellen [8] für die Programmierung von Sensornetzwerken sind Aufrufe an den Task-Scheduler zur Erstellung (*mos_thread_new*) und Pausierung (*mos_thread_sleep*) von Threads, Methoden zum Senden (*com_send*) und Empfangen (*com_recv*) von Nachrichten, die Ansteuerung vorhandener LEDs (*mos_led_toggle*) sowie das Schreiben (*dev_write*) und Lesen (*dev_read*) von Sensorwerten.

4.5 Multitasking

Mantis OS ermöglicht von Haus aus keine ereignisbasierte Programmierung. Das ausschließlich verwendete, präemptive Multitasking stützt sich bei der Wahl für einen aktiven Thread auf das Round Robin verfahren.

4.6 Energiesparen

Mantis OS verwendet einen separaten Scheduler zur Einsparung von Energie. Dieser erkennt den Zustand, in welchem sich alle Threads in Ruhe befinden. Basierend auf den jeweiligen Zeiten, welche die einzelnen Threads in diesem Zustand verweilen werden, wird der Mikrocontroller ebenfalls für einen gewissen Zeitraum abgeschaltet. Das ermöglicht eine zufriedenstellende Energieeffizienz trotz Multitaskings.

4.7 Remote Shell

Die entfernte Verwaltung ist eine weitere Schlüsselfunktion von Mantis OS [1]. Gegenwärtig noch nicht realisiertes Ziel dabei ist die Remote-Programmierung von Sensorknoten, wobei die entfernte Anmeldung zur Manipulation von Variablen bereits implementiert ist.

5. COCOOS

Das wichtigste Merkmal der erst zwei Jahre jungen cocoOS Umgebung sind Koroutinen. Diese werden in Tasks abgebildet, welche in der main Funktion des verwendeten C Programms initalisiert werden.
Die Schlüsselidee der cocoOS Tasks besteht darin, die Programmausführung gezielt zu unterbrechen, um auf Events reagieren zu können. Diese Ereignisse bilden die Schnittstelle zwischen den einzelnen Tasks.
Das System ist unter einer BSD Lizenz frei verfügbar.

5.1 Dokumentation

Die offizielle Projekthomepage [18], liefert lediglich eine rudimentäre Übersicht über das Gesamtkonzept der Plattform. Weitere Quellen wie das offizielle Supportforum [19] bieten kaum eine nennenswerte Ergänzung hierzu.

5.2 Hardwareunterstützung

cocoOS kann auf eingebetteten Mikrocontrollern wie AVR und MSP430 Architekturen eingesetzt werden [18] . Es hat den Anspruch, für den Einsatz in heterogenen Netzwerken prädestiniert zu sein.

5.3 Architektur

Die wesentlichen Elemente des cocoOS Frameworks sind in Abbildung 3 zusammengefasst. Die relativ lose Struktur bietet dabei keine Hardwareabstraktion an. Ein Schichtenmodell, wie es das Mantis OS verwendet, wird nicht angewendet.

5.3.1 Tasks

Als Kernbestandteil von cocoOS verfügen Prozeduren über die wichtigsten Elemente zur Steuerung des Programmablaufs. Zu den Steuerungsmechanismen zählen das Erstellen, Löschen, Pausieren, Fortführen und Warten auf Tasks.

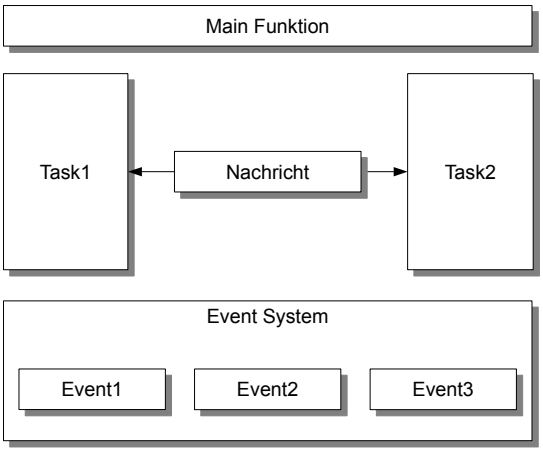


Abbildung 3: cocoOS Architektur

5.3.2 Events

Zur Synchronisation der einzelnen Tasks dienen Events. Sie können innerhalb eines Task-Rumpfes ausgelöst werden und werden unmittelbar danach automatisch wieder deaktiviert. Neben der Möglichkeit, das Auftreten eines einzelnen Events abzuwarten, kann dies ebenso für mehrere Events gleichzeitig geschehen.

5.3.3 Nachrichten

Zur Erweiterung der Events können Nachrichten verwendet werden, um zwischen Prozeduren zu kommunizieren. Solche Nachrichten werden von einem vordefinierten C Struct abgeleitet. Sie verfügen mindestens über einen Identifier und können vom cocoOS Kernel verzögert gesendet werden. Darüber hinaus können sie um beliebige Nutzdaten erweitert werden.

5.4 Programmierung

Die cocoOS Plattform wird in C programmiert. In der Main Funktion des Rahmenprogramms werden einzelne Prozeduren initialisiert. Deren Rumpf wird mittels weiterer C Funktionen implementiert. Weiterhin können eigene Formate für Nachrichten festgelegt werden. Die korrekte Benutzung von Semaphoren, zum Beispiel der Nachrichtenwarteschlange mittels *msg_q_get()* und *msg_q_give()*, bleibt dabei dem Programmierer überlassen.
Listing 1 veranschaulicht dies an einem Beispiel. Das Programm startet zwei Prozeduren welche sich innerhalb von Endlosschleifen im Wechsel Nachrichten und Events zukommen lassen.

Listing 1: Hello cocoOS

```
Evt_t event;

static Msg_t msg[16];

void task1(void) {
    task_open();
    for(;;) {
        /* Setze Semaphor und Empfange
            Nachricht */
        msg_q_get(task2);
        msg_receive(task2, &msg);
        msg_q_give(task2);
        /* Löse Event aus */
        event_signal(event);
    }
    task_close();
}

void task2(void) {
    task_open();
    for(;;) {
        /* Sende Nachricht */
        msg_q_get(task1);
        msg_post_in(task1, &msg, 1000);
        msg_q_give(task1);
        /* Warte auf Event */
        event_wait(event);
    }
    task_close();
}

int main(void)
{
    task_create(task2, 1, msg, 16, sizeof(
        Msg_t));
    task_create(task2, 2, NULL, 0, 0);
    os_start();
    return 0;
}
```

5.5 Semaphore

cocoOS stellt binäre Semaphore zur Verfügung und solche, welche Zählwerte annehmen können. Das verwendete Modell entspricht der Lösung des Consumer-Producer Problems. Bei der Programmierung ist auf die korrekte Verwendung der Semaphore zur Vermeidung von Konflikten zu achten.

5.6 Scheduling

Die cocoOS Tasks werden standardmäßig entsprechend ihrer Priorität ausgeführt. Dabei beeinflussen sich die einzelnen Tasks maßgeblich gegenseitig in ihrer Priorität. So können sie einerseits Events auslösen und andererseits Nachrichten direkt an andere Tasks senden.
Weiterhin besteht die Möglichkeit, diesen Vorgang explizit um das Round Robin Verfahren für die Zuteilung von Prozessorzeit zu erweitern.

5.7 Timer

cocoOS stellt mehrere Timer zur Verfügung. Neben dem Zugriff auf die Systemuhr besteht die Möglichkeit, weitere Timer zu Verwenden. Diese können individuell gesteuert werden, beispielsweise um die Länge einer übertragenen Nachricht zu überwachen.
Der primäre Timer wird per *os_tick()* anhand der benötigten Hardware Uhr aufgerufen und dekrementiert dabei die System Timer. Eigene Timer können mittels per *os_sub_tick(id)* verwendet werden. Dies bietet auch die Möglichkeit, mittels *os_sub_nTick(id, nTicks)* mehr als einen Schritt zu erhöhen.

doi: 10.2313/NET-2012-08-2_01

6. .NET MICRO FRAMEWORK

Das .NET Micro Framework [10] ist Teil der Microsoft Windows Embedded Familie [13] zum Einsatz in Hardwareumgebungen mit begrenzten Ressourcen. Es ist die offizielle Fortführung der Windows CE Familie. Das Framework interpretiert auszuführenden Programmcode anstatt ihn zu kompilieren. Damit handelt es sich nicht um ein Echtzeit–Betriebssystem.
Der Fokus liegt auf MIPS-, ARM- und x86-Prozessoren [3], womit auch die Möglichkeit zur gleichzeitigen Abhandlung mehrerer Ausführungsstränge gegeben ist [6].

6.1 Dokumentation

Der Hersteller ist sichtlich darum bemüht, Entwicklern eine breite Unterstützung für das Micro Framework anzubieten. Der zur Verfügung gestellte Internetauftritt [15] legt starken Fokus auf die Vernetzung der Entwickler untereinander. Er ist mit ausführlichen Materialien ausgestattet und bietet neben Verweisen auf Blogeinträge, Foren und einem obligatorischen Wiki auch Videomaterial an. Dies ermöglicht einen

6.2 Hardwareunterstützung

Das .NET Micro Framework benötigt zur Ausführung mindestens 64KB RAM und 256KB Flash Speicher [14].
Damit die Umgebung direkt auf Hardware eingesetzt werden kann, müssen der Hardwareabstraktionsschicht die entsprechenden Gerätetreiber einzeln beigefügt werden. Aktuell sind keine Treiber für WE Compact 7.0 online aufgeführt [16]. Alternativ sind vorgefertigte Pakete für eine Vielzahl an Zielplattformen vorhanden (siehe Abschnitt 6.3.1).

6.3 Architektur

Der Aufbau des .NET Micro Framework gliedert sich in klar abgegrenzte Schichten. Durch diese Struktur sind Programmierer bei der Umsetzung von Anwendungen im Programmaufbau wenig eingeschränkt. Jedoch ist zur Entwicklung Microsoft's Visual Studio für .NET Anwendungen erforderlich.

6.3.1 Board Support Packages

Der Hersteller bietet sogenannte Board Support Packages [3], welche für ein breites Hardwarespektrum verfügbar sind [17], an. Diese Pakete enthalten bereits, wie in Abbildung 4 gezeigt, die benötigten Gerätetreiber für die jeweilige Zielplattform sowie eine Abstraktion des verwendeten Kernels und eine Hilfseinrichtung für die Fehlersuche während des Entwicklungsprozesses. Für die Erstellung eigener Pakete werden vom Hersteller Vorlagen zur weiteren Anpassung zur Verfügung gestellt.

6.3.2 OEM Adaption Layer

Die hardwareabhängigen Funktionen werden dem Anwendungsprogrammierer mittels dem OEM Adaption Layer bereitgestellt. Dieser bildet eine Schnittstelle zum verwendeten Kernel abhängig vom verwendeten Mikrokontroller. Dabei wird zur vereinfachten Organisation dieser äußerst komplexen Komponente die Verwendung einheitlicher Bibliotheken innerhalb derselben Chipfamilie vorgeschlagen.

6.3.3 Kernel-Independent Transport Layer

Während des Programmiervorgangs kann zu Debugging Zwecken der Kernel-Independent Transport Layer verwendet werden. Diese Schicht wird vom OEM Adaption Layer (OAL)

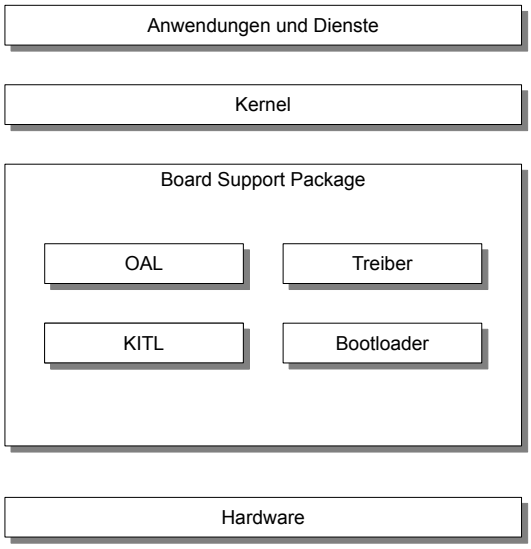


Abbildung 4: .NET Micro Framework Architektur

aufgerufen und ermöglicht das registrieren eigener Kommunikationsdienste. Sie stellt eine Schnittstelle zwischen der Arbeitsumgebung des Entwicklers und der Zielplattform dar.

6.3.4 Bootloader

Für die Zielplattform wird ein Bootloader zur Initialisierung des Betriebssystems benötigt. Der Bootloader stellt einen weiteren Bestandteil der Board Support Packages dar.
Das Speicherlayout kann über Konfigurationsdateien festgelegt werden, welche ebenfalls Bestandteil der Board Support Packages sind und zusätzlich zur Konfiguration der Speicherabbilder von Bootloader und Laufzeitumgebung verwendet werden.

7. ZUSAMMENFASSUNG

Zunächst fällt auf, dass cocoOS ein Nischendasein führt. Im Vergleich zu populäreren Systemen wie TinyOS gibt es lediglich eine geringe Anzahl an Fachpublikationen. Das cocoOS Support Forum [11] ist noch leer und die Dokumentation spärlich. Die ausführlichen Beschreibungen des .NET Micro Frameworks bilden dazu einen starken Kontrast.
Die Sicherheitsmechanismen der betrachteten Systeme sind rudimentärer Natur. Dies ist prinzipbedingt den knappen Ressourcen der Zielplattformen geschuldet.
Hinsichtlich des Arbeitsflusses wärend der Programmierung ist das .NET Micro Framework erste Wahl, sofern neben einer geeignete Zielplattform auch eine geeignete Entwicklungsplattform aus dem Hause Microsoft verfügbar ist. Dabei werden Kenntnisse im Umgang mit .NET vorausgesetzt. Andernfalls steht Mantis OS mit breiter Hardwareunterstützung zur Verfügung. Dieses hinterlässt insgesamt den besten Gesamteindruck im Hinblick auf den Einsatz in Sensornetzwerken. Für kleine Sensornetze empfiehlt sich cocoOS, wobei hier die Auswahl an Zielplattformen eingeschränkt ist.

Die Sicherheit der beiden Betriebssysteme Mantis OS und

Tabelle 1: Vergleichsmatrix

	MantisOS	cocoOS	.NET
Dokumentation	⊕⊕	⊖	⊕
Energiesparen	⊕		
Hardwareunterstützung	⊕⊕	⊕	⊕⊕
Sicherheit			
Programmierung	⊕⊕	⊕	⊖
Multitasking	⊕	⊕⊕	
Footprint	⊕	⊕	
Flache Lernkurve	⊕	⊕⊕	⊖⊖

cocoOS muss vom Anwender protokollseitig bei der Datenübertragung umgesetzt werden. Dabei geht man davon aus, dass dem unberechtigten, physischen Zugriff auf einen Sensorknoten beim aktuellen Stand der Technik kaum etwas entgegenzusetzen ist.

Eine Zusammenstellung der jeweiligen Stärken und Schwächen der betrachteten Betriebssysteme wurde in Tabelle 1 als Vergleichsmatrix erstellt. Dabei dürfen die angegebenen Gewichte als relative Metrik verstanden werden.

8. AUSBLICK

Die voranschreitende Weiterentwicklung von Hardwaremodulen, welche in Sensorknoten eingesetzt werden können, wird langfristig zu komplexeren Lösungen für Bestriebssysteme in Sensornetzen führen.
Damit wird nicht nur die Leistungsfähigkeit hinsichtlich der Erfassung wachsender Datenvolumen zunehmen, sondern auch die Sicherheitsanforderungen. Abhörsicherheit und Vertraulichkeit werden in demselben Maße an Bedeutung gewinnen. Schließlich ist zu erwarten, dass die Handhabung der Software für Sensorknoten weiter vereinfacht werden kann. Sensornetzwerke werden damit Einzug in Privatwohnungen halten und dort den technischen Fortgang, beispielsweise in der Heimautomatisierung, entscheidend prägen. Diese Entwicklung wird zusätzlich befeuert durch die Idee des *Internet of Things* [2] und der Etablierung von IPv6 Netzwerken zur globalen Adressierung kleinster elektrischer Geräte.
Damit ist zu erwarten, dass der Sicherheitsaspekt bei der Entwicklung von Betriebssystemen für Sensorknoten in den Fokus rückt ebenso wie die Leistungsfähigkeit hinsichtlich automatischer Venetzung. Dabei wird die wichtigste Herausforderung weiterhin im Ressourcenmanagement bestehen. Denn einerseits werden zwar die Hardwarekomponenten stetig leistungsfähiger. Doch andererseits werden gleichzeitig auch die Sensorknoten stetig kleiner.

9. LITERATUR

[1] 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. *Mob. Netw. Appl.*, 10(4):563–579, Aug. 2005.

[2] L. Coetzee and J. Eksteen. The internet of things - promise for the future? an introduction. In *IST-Africa Conference Proceedings, 2011*, pages 1 –9, May 2011.

[3] W. Giberson and J. Chan. *The Basics of Bringing up a Hardware Platform*. Microsoft, March 2011.

[4] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for smart dust. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, pages 271–278, New York, NY, USA, 1999. ACM.

[5] M. Krüger, R. Karnapke, and J. Nolte. Controlling sensors and actuators collectively using the cocos-framework. In *Proceedings of the First ACM workshop on Sensor and actor networks*, SANET '07, pages 53–54, New York, NY, USA, 2007. ACM.

[6] G. Langer and J. Chan. *Symmetric Multiprocessing Guide for Windows Embedded Compact 7*. Microsoft, November 2011.

[7] Y. Lee, G. Kim, S. Bang, Y. Kim, I. Lee, P. Dutta, D. Sylvester, and D. Blaauw. A modular 1mm3 die-stacked sensing platform with optical communication and multi-modal energy harvesting. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 402 –404, February 2012.

[8] M. M. R. Mozumdar, L. Lavagno, and L. Vanzago. A comparison of software platforms for wireless sensor networks: Mantis, tinyos, and zigbee. *ACM Trans. Embed. Comput. Syst.*, 8(2):12:1–12:23, Feb. 2009.

[9] X. Wang, F. Silva, and J. Heidemann. Infrastructureless location aware configuration for sensor networks. In *Mobile Computing Systems and Applications, 2004. WMCSA 2004. Sixth IEEE Workshop on*, pages 174 – 183, December 2004.

[10] *.NET Micro Framework Homepage*, `http://www.microsoft.com/en-us/netmf/about/default.aspx`

[11] *cocoOS Support Forum*, `http://www.cocoos.net/forum/`

[12] *TelosB Datasheet*, `http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf`

[13] *Windows Embedded Homepage*, `http://msdn.microsoft.com/de-DE/windowsembedded/`

[14] *.NET Micro Framework - Getting started*, `http://www.microsoft.com/en-us/netmf/about/gettingstarted.aspx`

[15] *Windows Compact*, `http://www.microsoft.com/windowsembedded/en-us/develop/windows-embedded-compact-7-for-developers.aspx`

[16] *Windows Compact Device Drivers*, `http://www.microsoft.com/windowsembedded/en-us/downloads/windows-embedded-driver-chooser.aspx?fsr=1&wince=1&WEItemsPerPage=10&sort=ReleaseDateConverted_desc`

[17] *Windows Compact Board Support Packages*, `http://www.microsoft.com/windowsembedded/en-us/downloads/board-support-packages-for-windows-embedded.aspx`

[18] *cocoOS Homepage*, `http://www.cocoos.net/`

[19] *cocoOS Support Forum*, `http://www.cocoos.net/forum/`

[20] *Mantis OS Homepage*, `http://mantisos.org/`

[21] *MANTIS: Programming Guides*, `http://mantisos.org/index/tiki-index.php%3Fpage=Programming+Guides.html`

Comparison of Operating Systems TinyOS and Contiki

Tobias Reusing
Betreuer: Christoph Söllner
Seminar: Sensorknoten - Betrieb, Netze & Anwendungen SS2012
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitekturen
Fakultät für Informatik, Technische Universität München
Email: reusing@in.tum.de

ABSTRACT

Wireless sensor networks can be used in a lot of different application areas. Since such networks were first proposed, many different node platforms both in regard to hardware and software were introduced. In this paper we present operating systems for wireless sensor nodes in general and two of the best known operating systems, TinyOS and Contiki with their most notable differences and similarities. Both have strengths and weaknesses which are important for various requirements of wireless sensor networks.

Keywords

TinyOS, Contiki, Operating Systems, Sensor Nodes, Wireless Sensor Networks

1. INTRODUCTION

Wireless sensor networks consist of a huge number of single nodes. They can be used for a wide range of applications. For all different application areas different requirements have to be fulfilled. While in one application it may be most important that the nodes can operate unattended for very long periods of time, in another application they may have to be able to process huge amounts of data in short time frames. Therefore it is of high importance to choose the right hard and software components for the particular application. The operating system is one of the most important parts on the software side of this decision process. There are a lot of sensor node operating systems available and at first glance it is not always easy to determine which one is better suited for which applications. Therefore this paper presents two of the best known sensor node operating systems and compares them according to different requirements.

Section 2 of this paper presents operating systems for wireless sensor networks in general. Section 3 and section 4 give an overview of the features of TinyOS and Contiki respectively. In section 5 both operating systems are compared according to requirements for wireless sensor nodes and section 6 presents the conclusions.

2. OPERATING SYSTEMS FOR WIRELESS SENSOR NETWORKS

Operating systems that are designed for wireless sensor networks are very different from operating systems for desktop/laptop computers like Windows or Linux or operating systems for powerful embedded systems like smart phones. The biggest difference is the hardware on which the operating systems are running. The wireless sensor nodes (often called motes) usually have a microcontroller as a CPU that is not very powerful because the main focus of those motes lies in minimal power consumption since they are often designed to run on battery power for very long periods of time. And even though the microcontroller and all other components of motes are designed as low power devices, running them all at full power at all times would still consume way too much energy. So for that matter the main focus of those operating systems is energy conservation optimal usage of limited resources[1].

Operating systems for motes are very simple compared to other operating systems. But they still are often required to handle many different operations at the same time. A mote could for example be required to collect data from a sensor, process the data in some way and send the data to a gateway at the same time. Since the microcontrollers are only able to execute one program at the time, the operating systems have to have a scheduling system that shares the CPU resources between the different tasks, so that all of them can finish in the desired time frame.

Since the requirements for the operating system vary between applications it is in most cases not possible to exchange the client program on a mote without changing the operating system. In fact in most cases the operating system behaves more like a library: It gets integrated into the application and both the application and the operating system are compiled into one single binary that is then deployed onto the sensor node.

In summary the main requirements for an operating system for sensor networks are[1]:

- Limited resources: The hardware platforms offer very limited resources so the operating system should use them efficiently.
- Concurrency: The operating system should be able to handle different tasks at the same time.
- Flexibility: Since the requirements for different applications vary wildly, the operating system should be able to be flexible to handle those.
- Low Power: Energy conservation should be one of the main goals for the operating system.

3. TINYOS

TinyOS was developed at the University of California in Berkeley[2] and is now maintained as an open source project by a community of several thousand developers and users lead by the TinyOS Alliance[12]. The current Version of TinyOS from April 6, 2010 is 2.1.1.

TinyOS uses an event driven programming model and concurrency is achieved with non-preemptive tasks[1]. TinyOS programs are organized in components and are written in the NesC language[3], a dialect of C.

3.1 Programming Model

As already mentioned, TinyOS user applications and the operating system itself are composed of components. Components offer three types of elements: Commands, Events and Tasks [1]. All three are basically normal C functions but they differ significantly in terms of who can call them and when they get called. Commands often are requests to a component to do something, for example to query a sensor or to start a computation. Events are mostly used to signal the completion of such a request.

Tasks, on the other hand, are not executed immediately. When a task is posted, the currently running program will continue its execution and the posted task will later be executed by the scheduler (see section 3.2 for details).

Components expose which commands they can send and which events they can handle through interfaces. An interface consists of a number of commands and events that are specified by their function signatures. A component can either specify to use an interface or to provide it. Components that provide an interface have to implement all of the specified commands and can signal the specified events. Components that, on the other hand, use an interface have to implement all of the specified events and can use all of the commands [4, p. 26] (cf. Figure 1).

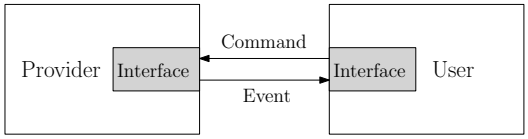


Figure 1: An interface specifies which commands the user of the interface can call and which events the provider can signal

The connections between components are specified in so-called configurations. A configuration defines for each interface of a component which other component uses the provided interfaces and which component provides the used interfaces. These connections are called wirings[4, p. 31]. Configurations are itself components, which means that they can also provide and use interfaces. This makes it possible to build TinyOS applications in a hierarchical manner where components on a higher level are made up of several components of a lower level (cf. Figure 2).

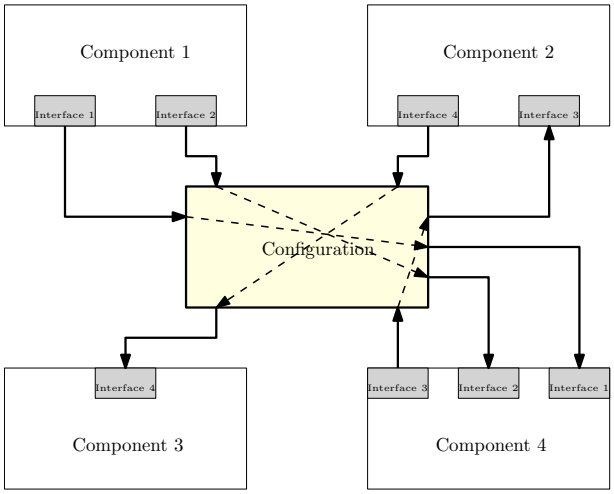


Figure 2: Configurations map the exposed interfaces of components onto each other

TinyOS programs and the operating system itself are written in NesC[3]. NesC is a dialect of C. It incorporates all of the concepts of TinyOS, like components, interfaces, commands, events, tasks, configurations etc. into a C like language. The biggest difference between C and NesC is how the function to be executed is selected. In C the function to be executed at a function call is selected by its name. In NesC when a command or event should be executed the programmer explicitly selects with the wiring in configurations which component's implementation of the command or event should be used[4, p. 13]. But "normal" C functions can still be used in NesC and to differentiate between them and commands, events and tasks special keywords are used for invoking each of the non-C function types. C libraries and C preprocessor directives can be used[4, p. 46-47]. NesC is also designed to allow whole-program analysis which allows the detection of data-race conditions which can improve reliability and it can help with inlining across component border which can reduce resource consumption[3].

3.2 Execution Model

TinyOS uses a split-phase execution model[1]. This means that the request to do an operation and the response after completion are decoupled. This approach resembles how interaction with hardware in a sensor node often works: The micro controller sends a request to a hardware component, which then works on it independently from the controller and later signals the completion of the task through an interrupt. In TinyOS both hardware and software modules follow this split-phase execution model, which is represented in the programming model: Both are components that can handle commands and at a later time signal events after the completion of the operation.

Concurrency in TinyOS is achieved with tasks. Tasks are basically functions that can be posted by other tasks or interrupt handlers[1]. They don't get executed immediately but instead will later be executed by the scheduler. The TinyOS scheduler executes one task after another and so tasks can never preempt each other. Each task runs until completion and the next task is started after that. This

doi: 10.2313/NET-2012-08-2_02

```
// BlinkC.nc - Blink module
module BlinkC {
  uses interface Boot;
  uses interface Timer;
  uses interface Leds;
}
implementation {
  event void Boot.booted() {
    call Timer.startPeriodic(1000);
  }

  event void Timer.fired() {
    call Leds.led0Toggle();
  }
}


// BlinkAppC.nc
configuration BlinkAppC { }
implementation {
  components MainC, LedsC, TimerC, BlinkC;

  BlinkC.Boot -> MainC.Boot;
  BlinkC.Leds -> LedsC.Leds;
  BlinkC.Timer -> TimerC.Timer;
}
```

Listing 1: Minimal example of a TinyOS application that turns a LED on and off every second. With modifications from [4]

makes it possible that all tasks can use the same stack, which saves memory because not every task needs it's own designated stack like in a multi-threaded approach (see section 3.3 for details). Normally tasks are executed in a first-in-first-out (FIFO) order, so tasks run in the same order they're posted[4], but it is possible to implement other (more complex) scheduling strategies[2]. If there is no more task in the queue after completion of the previous task, TinyOS sets the mote into a low-power sleep state until an interrupt wakes the microcontroller[4].

Code in TinyOS can only be executed in the context of either a task or an interrupt. Interrupts can preempt tasks and other interrupt handlers of lower priority. Code that is reachable from an interrupt handler is called asynchronous code and special measures have to be taken to handle concurrency issues that can occur because of that (see Figure 3 or [4, p. 192 ff] for further details).

Since tasks in TinyOS can not be preempted, long running tasks, for example tasks that handle complex computations like cryptographic functions, will block the whole application and tasks that are time critical, like tasks handling sending and receiving of data over a communication medium, may be waiting for too long before the scheduler executes them. In those cases the long running computation should be split up in shorter running tasks that post themselves after completing a part of the computation, which enables other tasks to run in between them. Since posting and executing a task generates a overhead of about 80 clock cycles on a current mote platform, a tradeoff between lots of short tasks and long running tasks that execute the same computation has to be found. Since data can not be kept on the stack between the execution of tasks, all state information has to be kept in the private memory of the tasks component[4, p. 74].

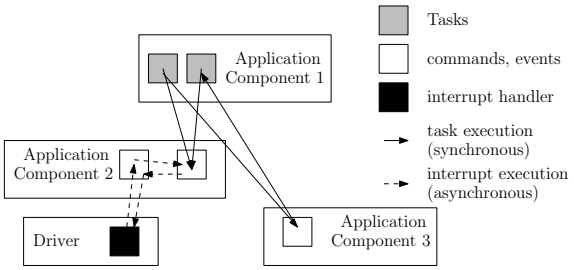


Figure 3: The TinyOS execution model. Component boundaries are crossed between all of the components[4]

3.3 Resource Use

TinyOS was built with the goal of minimal resource consumption since wireless sensor nodes are generally very constrained in regards to processing speed, program memory, RAM and power consumption.

3.3.1 Processing Power

To preserve processing power in TinyOS boundary crossings between different components are optimized. Since all function locations are known at compile time and there is no address-space crossing basic boundary crossing has at most the overhead of a single procedure call. With whole-program analysis many boundary crossings can be entirely removed. In some cases the compiler can even inline a whole component into it's caller[3].

To keep the overhead of task-switching minimal the scheduler in TinyOS is very simple. For example tasks have no return value and take no parameters so the scheduler does not need to take care of them[4, p. 72].

3.3.2 Program Memory

Since it is known at compile time, which components and which parts of those components of the application and the operating system are used, the compiled image of the application includes only the actually used procedures and as good as no dead code. Also the operating system itself has a very low memory footprint: The core TinyOS operating system uses less than 400 bytes of program memory[3].

3.3.3 RAM

Keeping the RAM usage of wireless sensor nodes low is very important, since the used microcontrollers are very restricted in RAM size. For example the Atmel Atmega128L micocontrollers used in the MICA2 sensor node[15] only offers 4 Kbytes of RAM[16] which have to be shared between the operating system and all running user programs. There are no memory management units (MMU) or other memory protection measures available on these microcontrollers so

doi: 10.2313/NET-2012-08-2_02

the risk of stack overflows (when the stack exceeds it's maximum size) should be avoided. The execution model with run to completion tasks is very efficient in terms of RAM usage, since all tasks use the same stack as opposed to a classic multi-threading approached where each thread needs a designated stack for itself. To further decrease stack size deep call hierarchies inside tasks should be avoided. TinyOS programmers are especially discouraged of using recursion and other similar techniques[4, p. 36-38].

3.3.4 Power Consumption

Since many wireless sensor nodes run on battery power, energy consumption should be kept as low as possible. To achieve a low power consumption the microcontroller should be kept in a low power sleep state for as long as possible. For example the Atmel Atmega128L needs a supply current in the magnitude of a few milliamperes when active. In the lowest sleep state a current of only a few microamperes[16] is sufficient, which means the difference between power consumption in active and sleep states can be a factor of 1000 or more. TinyOS copes with this by using the split-phase and event-driven execution model. As long as there are no tasks in the task queue the scheduler puts the CPU in sleep mode. So in combination with the split-phase operation the CPU does not waste energy while waiting for other hardware components[1].

But the CPU is not the only power critical component in a wireless sensor mode, periphery hardware can use a lot of power, too. For example the radio is in most cases the most expensive part of the node in respect to energy consumption[4, p. 7]. To save energy in those components TinyOS has a programming convention that allows subsystems to be put in a low power idle state. Components have an interface, which exposes commands to tell the component to try to minimize it's power consumption, for example by powering down hardware, and to wake up again from those power saving states[1].

3.4 Hardware Platforms

The current release of TinyOS supports quite a few sensor node hardware platforms. This includes different motes by Crossbow Technologies like the MICA family of motes or the iMote2 developed at Intel Research and many more[13]. These hardware platforms run on a range of different microcontroller including the ATMEL AVR family of 8-bit microcontrollers, the Texas Instruments MSP430 family of 16-bit microcontrollers, different generations of ARM cores, for example the Intel XScale PXA family, and more[13].

It is of course possible to use TinyOS for a custom or not yet supported hardware platform. The platforms in TinyOS are designed to be very modular. For each of the supported microcontrollers and other hardware, for example radio chips, components exist in the TinyOS installation. To port TinyOS to a platform it basically suffices to specify which components to use, how they are connected to each other (for example which pins of the microcontroller are connected to the radio chip) and to configure each of the components correctly[14].

If a hardware device of a platform is not supported by TinyOS a driver has to be programmed or a existing driver has to be ported to TinyOS. This is not trivial and can get very complex depending on the hardware[14].

To make software for TinyOS as platform independent as possible but at the same time offer the possibility to push the hardware to its limits with platform specific code, the hardware abstraction architecture (HAA) was introduced. The HAA offers three levels of hardware abstraction for drivers[4, p. 206-207]:

- The hardware interface layer (HIL): This is the most platform independent level of device drivers. It offers only the functionality that is common to all devices that use the common interfaces.
- The hardware adaption layer (HAL): The HAL is a tradeoff between platform independence and the use of platform specific code. It should offer platform independent interfaces when possible and platform specific interfaces in all other cases
- The hardware presentation layer (HPL): The platform specific level of the HAA. It sits directly on top of the hardware and offers all of it's functionality in a NesC friendly fashion.

3.5 Toolchain

The core of the TinyOS toolchain is the NesC compiler. Current implementations of the NesC compiler take all NesC files, including the TinyOS operating system, that belong to a program and generate a single C file. This C file can then be compiled by the native C compiler of choice for the target platform. The resulting binary can then be deployed on the motes in a appropriate way. Many optimizations are already done by the NesC compiler, for example the exclusion of dead code. Furthermore the output C file of the NesC compiler is constructed in a way, that makes it easy for the C compiler to further optimize it. Since it is just one single file the C compiler can freely optimize across call boundaries.

Apart from the actual build toolchain there is also a TinyOS simulator called TOSSIM[5]. It can simulate whole TinyOS programs and the underlying motes without the need of actually deploying it on hardware. With TOSSIM it is possible to simulate sensor node networks of thousands of nodes.

4. CONTIKI

Contiki is a open source operating systems for sensor nodes. It was developed at the Swedish Institute of Computer Science by Dunkels et al. [6]. It's main features are dynamic loading and unloading of code at run time and the possibility of multi-threading atop of an event driven kernel, which are discussed in sec. 4.1 and sec. 4.2 respectively. It's current version is 2.5 released on September 12, 2011.

4.1 Programming Model

A Contiki application consists of the Contiki kernel, libraries, the program loader and processes. Processes are either services or an application program. The difference between services and application programs is, that the functionality of services can be used by more than one other process,

while application programs only use other processes and do not offer functionality for different other processes[6].

Each of the processes must implement an event handler function and can optionally implement a poll handler function. Processes can be executed only through these handlers. Every process has to keep its state information between calls of these functions, since the stack gets restored after the return from these functions[6].

One of the special features of Contiki is the ability to replace all programs dynamically at run-time. To accomplish that Contiki offers a run-time relocating function. This function can relocate a program with the help of relocation information that is present in the program's binary. After the program is loaded the loader executes its initialization function where one or more processes can be launched[6].

A running Contiki system consists of the operating system core and the loaded programs. The core typically consists of the kernel, different libraries and drivers and the program loader (See Figure. 4). The core usually is deployed as one single binary while the loaded programs each can be distributed independently[6].

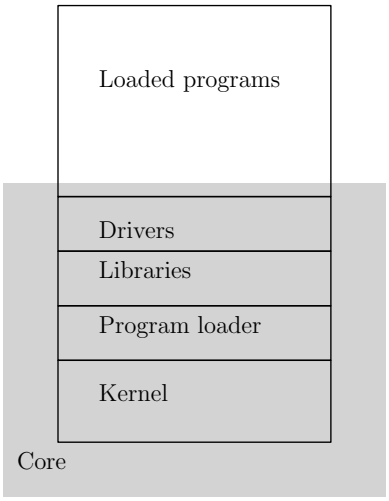


Figure 4: The partitioning of the Contiki core and the loaded programs

The Contiki kernel offers no hardware abstraction. If hardware abstraction is desired libraries and/or drivers have to implement it themselves. All components of a Contiki application have direct access to the underlying hardware.

4.2 Execution Model

The Contiki kernel is event-driven. Processes can only be executed by the scheduler when it either dispatches an event to the event handler of the process or by calling the polling handler. While events always have to be signaled by a process, the scheduler can be configured to call the polling handlers of all processes that implement one in periodic intervals between the dispatching of events. Both the event handlers and the polling handlers are not preempted by the scheduler and therefore always run to completion. Like the tasks in

```
// blink.c
PROCESS(blink_process, "blink example");
AUTOSTART_PROCESSES(&blink_process);

PROCESS_THREAD(blink_process, ev, data)
{
  PROCESS_BEGIN();
   leds_off(LEDS_ALL);
   static struct etimer et;
   while(1) {
     etimer_set(&et, CLOCK_SECOND);

     PROCESS_WAIT_EVENT();
     leds_toggle(LEDS_GREEN);
     }
  PROCESS_END();
}
```

Listing 2: The example from Listing 1 implemented as a protothread for Contiki. With modifications from [17]

TinyOS these handlers all can operate on the same stack and do not need a private stack of their own[6].

There are two kinds of events in Contiki: Asynchronous and synchronous events. Asynchronous events work similar to the posting of tasks in TinyOS. When an asynchronous event is signaled the scheduler enqueues the event and will call the corresponding event handler after all currently enqueued events were processed. Synchronous events on the other hand are more like inter-process procedure calls: The scheduler immediately calls the corresponding event handler and returns the control to the calling process after the event handler has finished running.

While event handler can not preempt each other, interrupts can of course preempt the current running process. To prevent race-conditions from happening, events can not be posted from within interrupt handlers. Instead they can request the kernel to start polling at the next possible point in time.

On top of this basic event-driven kernel other execution models can be used. Instead of the simple event handlers processes can use Protothreads[7]. Protothreads are simple forms of normal threads in a multi-threaded environment. Protothreads are stackless so they have to save their state information in the private memory of the process. Like the event handler Protothreads can not be preempted and run until the Protothread puts itself into a waiting state until it is scheduled again[7].

Contiki also comes with a library that offers preemptive multi-threading on top of the event-driven kernel. The library is only linked with the program if an application explicitly uses it. In contrast to Protothreads this multi-threading approach requires every thread to have it's own designated stack[6].

Both multi-threading approaches were introduced because

modeling application in the event-driven approach can be very complex depending on the specific requirements of the program. The event-driven execution model requires in most cases the implementation of a state machine to achieve the desired behavior, even if the programmer may not be aware of that. Dunkels et al. suggest in [7] that the code size of most programs can be reduced by one third and most of the state machines could entirely removed by using Protothreads. While the overhead in execution time is minimal there is a moderate increase in the program memory required by this approach.

4.3 Resource Use

Since the scheduler and the kernel in general are more complex in Contiki than in TinyOS and the possibility of dynamically load processes, which doesn't allow cross boundary optimization, the required program memory and execution time for Contiki programs is higher than that of TinyOS programs. When using the preemptive multi-threading library the RAM usage will be higher than using only the event-driven kernel for scheduling[6].

4.4 Energy Consumption

The Contiki operating system offers no explicit power saving functions. Instead things like putting the microcontroller or peripheral hardware in sleep modes should be handled by the application. For that matter the scheduler exposes the size of the event queue so that a power saving process could put the CPU in sleep mode when the event queue is empty.

4.5 Hardware Platforms

Contiki has been ported to a number of mote platforms on basis of different microcontrollers. Supported microcontroller include the Atmel AVR, the Texas Instruments MSP430 and the Zilog Z80 microcontrollers[6].

Porting Contiki requires to write the boot up code, device drivers and parts of the program loader. If the multithreading library is used, its stack switching code has to be adapted. The kernel and the service layer are platform independent. According to Dunkels et al. in [6] the port for the Atmel AVR was done by them in a view hours and the Zilog Z80 port was made by a third party in one single day.

4.6 Toolchain

Contiki is written in plain C so a native C compiler for the target platform can be used.

5. DISCUSSION

As presented in sec. 2 operating systems for wireless sensor nodes have to fulfill a few requirements. After the look at both TinyOS and Contiki we now compare both operating systems by means of these requirements:

- Limited resources: Both operating systems can be run on microcontrollers with very limited resources. But due to the higher complexity of the Contiki kernel TinyOS can generally get by with lower resource requirements.
- Concurrency: TinyOS offers only the event-driven kernel as a way of fulfilling the concurrency requirements. While Contiki also uses an event-driven kernel it also has different libraries that offer different levels of multithreading on top of that. But there are efforts to offer libraries similar to those of Contiki, for example by Klues et al. [8] or MacCartney et al. [9].
- Flexibility: Both operating systems are flexible to handle different types of applications. When it comes to updating an application that is already deployed Contiki can dynamically replace only the changed programs of the application, while an application using TinyOS has to be replaced completely, including the operating system. But there are solutions for making dynamic loading op application code possible for TinyOS, for example by introducing a virtual machine[10, 11].
- Low Power: TinyOS has out-of-the-box better energy conservation mechanisms but for Contiki similar power saving mechanisms can be implemented.

6. CONCLUSION

Both operating systems can generally fulfill all of the discussed requirements. In details there are differences, so while TinyOS is better suited when resources are really scarce and every little bit of saved memory or computing power can help, Contiki might be the better choice when flexibility is most important, for example when the node software has to be updated often for a large amount of nodes.

This paper is not an in depth discussion of both operating systems. When choosing the operating system for a specific application many things have to be considered and not all could be presented here. These two operating systems are also not the only operating systems for wireless sensor nodes so others may be considered.

7. REFERENCES

[1] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer *TinyOS: An operating system for sensor networks* In Ambient intelligence, p. 115-148, Springer, Berlin, 2005

[2] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, K. S. J. Pister *System architecture directions for networked sensors* In SIGPLAN Not. 35 (11), p. 93–104, ACM, 2000

[3] D. Gay, P. Levis, R. von Behren, M.Welsh, E. Brewer, and D. Culler *The nesC language: A holistic approach to networked embedded systems* In Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI), ACM, 2003

[4] P. Levis, D. Gay *TinyOS Programming* Camebridge University Press, Camebridge, 2009

[5] P. Levis, N. Lee, M. Welsh, D. Culler *TOSSIM: Accurate and scalable simulation of entire TinyOS applications* In Proceedings of the 1st international conference on Embedded networked sensor systems, p. 126-137, ACM, 2003

[6] A. Dunkels, B. Grönvall, T. Voigt *Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors* In Proceedings of the First IEEE

Workshop on Embedded Networked Sensors, Tampa, Florida, USA, 2004

[7] A. Dunkels, O. Schmidt, T. Voigt, M. Ali *Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems* In Proceedings of the Forth International Conference on Embedded Networked Sensor Systems, p. 29-42, ACM, 2006

[8] K. Klues, C.J.M. Liang, J. Paek, R. Musaloiu-E, P. Levis, A. Terzis, R. Govindan *TOSThreads: thread-safe and non-invasive preemption in TinyOS* In Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, p. 127-140, ACM, 2009

[9] W. P. McCartney, N. Sridhar *Stackless Preemptive Multi-Threading for TinyOS* In Proceedings of the 2011 International Conference on Distributed Computing in Sensor Systems (DCOSS), IEEE, 2011

[10] P. Levis, D. Culler *Maté: a tiny virtual machine for sensor networks* In Proceedings of the 10th international conference on Architectural support for programming languages and operating systems, p. 85-95, ACM, 2002

[11] A. Dunkels, N. Finne, J. Eriksson, T. Voigt *Run-time dynamic linking for reprogramming wireless sensor networks* In Proceedings of the 4th international conference on Embedded networked sensor systems, p. 15-28, ACM, 2006

[12] *TinyOS Open Technology Alliance*, `http://www.cs.berkeley.edu/~culler/tinyos/alliance/overview.pdf`

[13] *TinyOS Wiki - Platform Hardware*, `http://docs.tinyos.net/tinywiki/index.php?title=Platform_Hardware&oldid=5648`

[14] *TinyOS Wiki - Platforms*, `http://docs.tinyos.net/tinywiki/index.php?title=Platforms&oldid=4712`

[15] Crossbow Technology *MICA2 Datasheet* `http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf`

[16] Atmel Corporation *ATmega128/L Datasheet* `http://www.atmel.com/Images/doc2467.pdf`

[17] *Zoleria Dokumentation Wiki*, `http://zolertia.sourceforge.net/wiki/index.php?title=Mainpage:Contiki_Lesson_1&oldid=1138`

Reprogrammierungstechniken für drahtlose Sensornetzwerke

Christian Sternecker
Betreuer: Christoph Söllner
Seminar Sensorknoten - Betrieb, Netze & Anwendungen SS2012
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitekturen
Fakultät für Informatik, Technische Universität München
Email: sterneck@in.tum.de

KURZFASSUNG

Aufgrund ihres Einsatzgebietes müssen drahtlose Sensornetze (WSNs) häufig drahtlose reprogrammiert werden. Dabei treten diverse Probleme auf, die durch Betriebssysteme und Datendisseminationsprotokolle gelöst werden können. Diese Arbeit stellt zunächst die Handhabung der Reprogrammierung bei den Betriebssystemen TinyOS und Contiki dar. Anschließend werden die Datendisseminationsprotokolle, die die neuen Programme im Netz verteilen, XNP, Trickle, Deluge, Zephyr und das Selective Reprogramming Scheme von Krüger, Pfisterer und Buschmann genauer beschrieben und soweit möglich miteinander verglichen. Während die vier erstgenannten in dieser Reihenfolge aufeinander aufbauen und jeweils mehr Funktionen und eine bessere Performanz als der Vorgänger liefern, verfolgt das Selective Reprogramming Scheme einen anderen Ansatz und ist schlecht mit den anderen vergleichbar.

Schlüsselworte

Drahtlose Sensornetzwerke, WSN, XNP, TinyOS, Contiki, Trickle, Deluge, Zephyr, Selective Reprogramming Scheme, data dissemination protocol

1. EINLEITUNG

Drahtlose Sensornetzwerke (wireless sensor networks, WSNs) bestehen aus einer oft grossen Anzahl einzelner Sensorknoten, die sehr weit verteilt sein können. Sie sind dazu gedacht, über einen längeren Zeitraum eingesetzt zu werden. Dadurch kann es notwendig werden, einzelne oder alle Knoten zu reprogrammieren. Durch die reine Anzahl und die Verteilung der Knoten ist eine Reprogrammierung jedes einzelnen Knotens mittels eines angeschlossenen Laptops meist nicht praktikabel. Deshalb ist es erforderlich, die Knoten über den Funkkanal zu reprogrammieren.

Dabei ergeben sich jedoch mehrere Schwierigkeiten. Eine Schwierigkeit sind die geringen Ressourcen, die den einzelnen Sensorknoten zur Verfügung stehen, da ein Knoten deshalb keine zu komplexen Algorithmen in vertretbarer Zeit ausführen kann. So hat ein Mica2-Knoten nur 128 kByte Programmspeicher, 4 kByte RAM, 512 kByte externen Flash-Speicher und einen 7 MHz Prozessor[1]. Ein etwas neuerer TelosB-Knoten verfügt über 48 kByte Programmspeicher, 10 kByte RAM, 1024 kByte externen Flash-Speicher und einen 8 MHz Prozessor[2]. Zwar existieren mittlerweile wesentlich leistungsfähigere Knoten, in etwa die Sun Spots mit 8 MByte Programmspeicher, 1 MByte RAM und 400 MHz Prozessor[3], jedoch befinden sich noch viele Mica2-Knoten im Einsatz. außerdem benötigt die Kommunikation zwischen den Knoten bis zu zehn mal mehr Energie[1, 2] als der Betrieb der Prozessoren. Diese Energie wird dem Knoten in der Regel nur von einzelnen Batterien oder durch Energy Harvesting, etwa mittels Sollarzellen, zur Verfügung gestellt. Jede Benutzung des Funkkanals verkürzt die Restlaufzeit des Knotens.

Eine weitere Schwierigkeit stellt noch der, im Gegensatz zum Kabel, unzuverlässige Funkkanal dar, der die fehlerfreie Verteilung des neuen Programms im Netzwerk behindert. Das Laden des neuen Programms wird zusätzlich zu den begrenzten Ressourcen noch durch die Architektur mancher Betriebssystem der Knoten, etwa TinyOS (siehe Abschnitt 3.1), erschwert.

Die Teilschritte der Reprogrammierung eines WSN sind die Verteilung des neuen Programmcodes auf die Knoten, das Laden des erhaltenen Codes in den Programmspeicher des Knotens und die Ausführung des neuen Codes. Die Verteilung ist Aufgabe von Datendisseminationsprotokollen (data dissemination protocols), Laden und Ausführen sind Aufgaben des Betriebssystems auf dem jeweiligen Knoten.

Diese Arbeit gibt einen Überblick darüber, wie diese Aufgaben in verschiedene Betriebssysteme und Datendisseminationsprotokollen gelösst werden.

Zuerst wird in Abschnitt 2 der grundlegende Aufbau eines drahtlosen Sensornetzwerks beschrieben.

Im Abschnitt 3 wird auf zwei Betriebssysteme, namentlich TinyOS (Abschnitt 3.1) und Contiki (Abschnitt 3.2) eingegangen. Dabei werden die Eigenschaften des Betriebssystems und die sich daraus ergebenden Lösungen zum Laden von neuem Code erörtert.

Der Abschnitt 4 beschreibt die Lösungen der Verteilung des Codes durch eine Auswahl an Datendisseminationsprotokollen und vergleicht deren Performanz miteinander soweit möglich.

Von den zahlreichen Protokollen ([4] beinhaltet eine unvollständige Liste) werden hier XNP (Abschnitt 4.1), Trickle (Abschnitt 4.2), Deluge (Abschnitt 4.3), Zephyr (Abschnitt 4.4) und das Selective Reprogramming Scheme von Krüger, Pfisterer und Buschmann (Abschnitt 4.5) besprochen.

Abschnitt 5 schließlich fasst die Arbeit zusammen.

2. AUFBAU EINES DRAHTLOSEN SENSORNETZWERKS

Ein drahtloses Sensornetz besteht aus einer Anzahl einzelner Sensorknoten. Mindestens einer dieser Knoten fungiert als Basisstation. Die Basisstation dient dazu, die vom gesamten Netwerk gemessenen Daten einzusammeln und auszuwerten. Oft werden die Daten zur Auswertung an fest mit der Basisstation verbundene, leistungfähigere Systeme weitergegeben. Die Basisstation dient auch als Ausgangspunkt, um Daten wie etwa Programmcode zur Reprogrammierung der restlichen Knoten im Netzwerk zu verteilen. Da meist nicht alle restlichen Knoten des Netzwerkes direkten Funkkontakt zur Basisstation haben, ist es nötig, Daten über mehrere Zwischenstationen (Hops) zu übertragen. Daher muss eines von zahlreichen Routingprotokollen[5] verwendet werden. Das verwendete Routingprotokoll bestimmt die Rolle der restlichen Knoten im Netzwerk. Zu beachten ist, dass die in Abschnitt 4 vorgestellten Protokolle nicht das vorhandene Routingprotokoll des Netzes verwenden, sondern ihre Daten auf eigene Weise verteilen.

3. BETRIEBSSYSTEME

In diesem Abschnitt wird die Reprogrammierung in den beiden Betriebssystemen TinyOS und Contiki beschrieben.

3.1 TinyOS

TinyOS[6] wird seit 1999 von der UC Berkeley als Betriebssystem für WSNs entwickelt. Es ist im C-Dialekt NesC (network embedded system C[7]) geschrieben und ist komponentenbasiert und ereignisgesteuert. TinyOS unterstützt seit Version 1.1 die drahtlose Reprogrammierung. Dafür wurde das von Crossbow Technologies Inc. entwickelte XNP (Xbow in-network programming)-Protokoll (siehe Abschnitt 4.1) verwendet. Seit Version 1.1.8 wird das Deluge-Protokoll (siehe Abschnitt 4.3) der UC Berkeley verwendet.
Bei der Reprogrammierung wird das neue Programm an die Knoten des Netzwerks verteilt und vom jeweilien Knoten in den externen Flash-Speicher des Knotens geschrieben. Sobald das neue Programm vollständig im Flash-Speicher ist, wird der Bootloader aufgerufen, der am Anfang des MCU-internen Programmspeichers steht. Er überprüft das neue Programm auf Vollständigkeit und Fehlerfreiheit und lädt es anschließend in den Programmspeicher. Abschließend startet er den Knoten neu. Bei TinyOS wird das Anwendungsprogramm immer zusammen mit dem Betriebssystem kompiliert und zusammen auf den Knoten aufgespielt. außerdem ist das Linking in TinyOS statisch. Aus diesen Gründen ist es nicht möglich, einen Knoten nur teilweise zu reprogrammieren. Bei jeder noch so kleinen Änderung, etwa dem Ändern eines Messintervalls, ist es nötig, das gesamte System des Knotens zu ersetzen, sofern diese Stelle im Programm nicht speziell programmiert wurde um von außen konfigurierbar zu sein. Um diesem Problem zu begegnen, wurde von der UC Berkeley die virtuelle Maschine Maté[8] entwickelt. Jedoch wurde Maté nicht für TinyOS 2.x portiert und ist daher nicht mehr relevant.

3.2 Contiki

Seit 2003 wird Contiki[9, 10] am Swedish Institute of Computer Science als Betriebssystem für vernetzte Systeme mit begrenzten Ressourcen unter anderem Sensornetzen entwickelt. Es ist in C geschrieben und wie TinyOS ereignisgesteuert. Anders als TinyOS ist es in Contiki nicht notwendig, das Anwendungsprogramm zusammen mit dem Betriebssystem zu kompilieren. Das auf einem Knoten laufende System besteht bei Contiki aus zwei Teilen. Der eine ist der Kern, bestehend aus dem Kernel, dem Programloader, einigen Bibliotheken, Teilen des Runtimes und dem Kommunikationsstapel. Der zweite Teil ist das momentan laufende Programm. Abbildung 1 zeigt diesen Aufbau. Das Programm oder Teile davon können zur Laufzeit ausgetauscht werden. Auf diese Weise ist es möglich, die Datenmenge, die bei der Reprogrammierung durch das Netzwerk verbreitet werden muss, zu reduzieren. Die auszutauschenden Programmteile werden beim Empfangen in den Flash-Speicher geschrieben. Sobald das Programm vollständig empfangen ist, lädt der Programloader das Programm in den Programmspeicher des Knotens und ruft seine Initialisierungsfunktion auf. Sollen Teile des Kerns, etwa im Kommunikationsstapel geändert werden, so ist es auch bei Contiki erforderlich, das gesamte System des Knotens auszutauschen. Um die auszutauschenden Programmteile beziehungsweise das gesammte neue System im Netzwerk zu verteilen, benutzte Contiki in Version 1.0 ein XNP-ähnliches Protokoll. Seit Version 2.0 wird Trickle (siehe Abschnitt 4.2) verwendet.

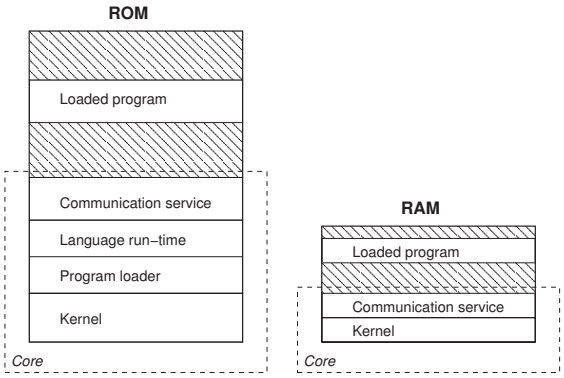


Abbildung 1: Teile eines laufenden Systems in Contiki. Entnommen aus [9].

4. DATENDISSEMINATIONS-PROTOKOLLE

Dieser Abschnitt beschreibt die Datendisseminationsprotokolle XNP, Trickle, Deluge, Zephyr und das Selective Reprogramming Scheme von Krüger, Pfisterer und Buschmann näher und vergleicht ihre Performanz soweit möglich. Da die Kommunikation zwischen den Knoten von allen Aktionen, die die Knoten bei diesen Protokollen ausführen, den Großteil der Energie verbraucht, ist der Energieverbrauch eines Protokolls in etwa proportional zu der Menge insgesamt an gesendeten Paketen.

4.1 XNP

XNP[11, 12] ist ein relativ einfaches Datendisseminationsprotokoll. Es wurde von den TinyOS Versionen 1.1 bis 1.1.7 verwendet. XNP kann nur zur Reprogrammierung von Knoten verwendet werden, die von der Basisstation aus ohne Zwischenstation erreichbar sind. Um das neue Programm im Netzwerk zu verteilen, wird es von einem PC auf die Basis-

doi: 10.2313/NET-2012-08-2_03

station überspielt und von dort aus versendet. Das Protokoll unterscheidet bei der Reprogrammierung drei Phasen:

- die Downloadphase, in der das neue Programm im Netzwerk veteilt wird
- die Nachfragephase, in der das verteilte Programm auf vollständigkeit geprüft wird
- die Reprogrammierungsphase, in der die Knoten reprogrammiert werden

In der Downloadphase sendet die Basisstation eine Download-Start-Nachricht an alle Knoten des Netzwerks. Diese reservieren daraufhin die notwendigen Ressourcen für die Reprogrammierung. Das Senden der Download-Start-Nachricht wird mehrmals wiederholt, um die Wahrscheinlichkeit, dass ein Knoten sie nicht empfangen hat zu verringern. Anschließend werden die Nachrichten mit dem neuen Programm an alle Knoten im Netz gesendet. Jede Nachricht wird einmal versendet und beinhaltet 16 Byte des Programms. Die Knoten schreiben die empfangenen Teile in ihren externen Flash-Speicher. Sobald alle Teile des Programms versendet wurden, beginnt die Nachfragephase. Die Basisstation versendet eine Download-Terminate-Nachricht an alle Knoten. Die Knoten prüfen daraufhin, ob sie alle Teile des Programms erhalten haben. Die Basisstation sendet daraufhin eine Anfrage (Request) an alle Knoten, ob sie alle Programmteile erhalten haben. Vermisst ein Knoten einen Programmteil, so sendet er eine Anfrage für diesen Teil an die Basisstation. Diese versendet daraufhin den angefragten Teil erneut an alle Knoten. Anschließend sendet die Basisstation die Anfrage erneut an alle Knoten. Sobald die Basissation auf diese Anfrage wiederholt keine Antwort erhalten hat, beginnt die Reprogrammierungsphase. In dieser Phase sendet die Basisstation eine Reprogrammierungsanfrage an alle Knoten. Sobald ein Knoten diese Anfrage empfangen hat, ruft er seinen Bootloader auf. Dieser kopiert das neue Programm in den Programmspeicher und startet den Knoten neu. Abbildung 2 zeigt einen beispielhaften Ablauf von XNP bis zum Aufrufen des Bootloaders.
XNP hat mehrere offensichtliche Schwächen. Die erste ist, dass es keine Multi-Hop-Reprogrammierung ünterstützt. Eine weitere ist, dass keine Bestätigungen für den Empfang der Programmteile verwendet wird, sondern nur Anforderungen nicht vorhandener Teile. Falls diese Anforderungen mehrfach verloren gehen, wird der Knoten entweder gar nicht, oder mit dem unvollständigen Programm reprogrammiert. In letzterem Fall läuft kein funktionierendes Programm mehr auf dem Knoten und es ist nicht möglich, ihn erneut zu reprogrammieren ohne ihn an einen PC anzuschließen. Laut[12] ist XNP jedoch sehr zuverlässig. Allerdings wurden nur relativ kleine WSNs mit höchstens 16 Knoten getestet. Bei größeren Netzwerken steigt das Risiko für verlorene Nachrichten jedoch an. Nach Messungen in[12] dauerte das Verteilen eines Programms der Grösse 37000 Byte inklusive dem erneuten Versenden fehlender Teile im Schnitt 159 Sekunden, im schlechtesten Fall betrug die Gesamtdauer 250 Sekunden.

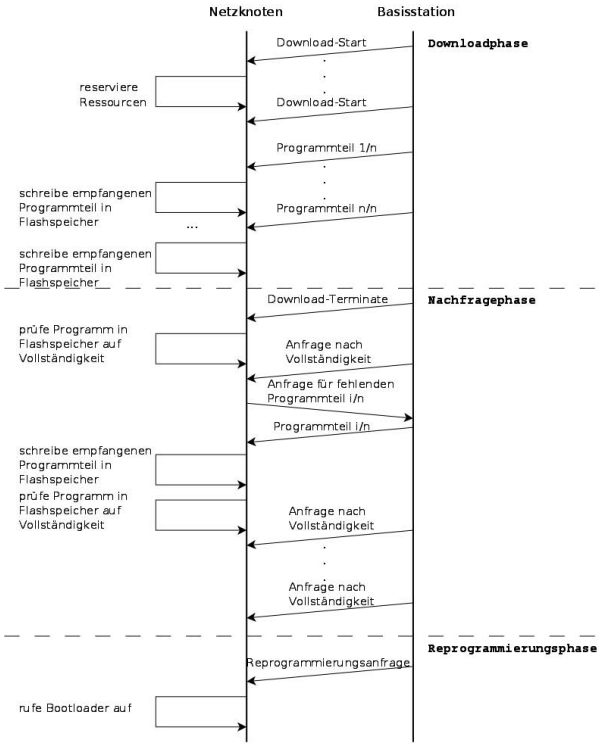


Abbildung 2: Beispielhafter Ablauf von XNP zwischen der Basisstation und einem Netzknoten. Dem Netzknoten fehlt am Ende der Downloadphase nur der Programmteil i. Die Basisstation versendet all ihre Nachrichten per Broadcast an alle Knoten und reagiert auf Anfragen aller Knoten des Netzes.

4.2 Trickle

Trickle[13] wurde von der UC Berkeley entwickelt und wird von Maté und Contiki ab Version 2.0 verwendet. Es benutzt den so genannten "polite gossip"-Ansatz, um neue Programmteile im Netzwerk möglichst schnell und mit möglichst wenig Overhead zu verteilen. Dazu teilen die Knoten dem Netzwerk regelmäßig mit, auf welchem Stand sich ihr Programm befindet. Bekommt ein Knoten mit, dass ein anderer auf dem gleichen Stand ist, so geht er davon aus, dass er selbst auf dem neuesten Stand ist und sendet selbst keine neue Nachricht, da dies keinen Mehrwert an Information darstellen würde. Hört er von einem Knoten mit einer neueren Version des Programms, teilt er dem Netzwerk seinen veralteten Stand mit. Erhält ein Knoten die Nachricht, dass ein anderer auf einem älteren Stand ist, so sendet er seine Version des Prgramms an alle erreichbaren Knoten, die sich daraufhin reprogrammieren. Auf diese Weise verbreitet sich der neueste Stand im gesamten Netz.
Trickle funktioniert im Detail wie folgt: Jedes Programm auf einem Knoten trägt eine Versionsnummer. Version x eines Programms wird als ϕ_x bezeichnet. Jeder Knoten teilt die Zeit in Intervalle der Länge τ ein. außerdem besitzt jeder Knoten noch die Parameter c, k und t. Am Anfang jedes Zeitintervalls wird t zufällig auf einen Wert aus dem Intervall $[0, \tau]$ gesetzt, c wird auf 0 und k auf einen festen Wert (1 oder 2) gesetzt. Sobald der Zeitpunkt t erreicht ist, überprüft der Knoten, ob $c < k$ gilt. Ist das der Fall, sendet der Knoten eine Nachricht mit den Versionsnummern seines Programms per Broadcast an alle erreichbaren Knoten. Der Knoten hört zu jedem Zeitpunkt den Kanal ab. Empfängt er eine Nachricht von einem anderen Knoten mit den gleichen Versionsnummern, so wird c um eins erhöht. Wenn der Knoten eine Nachricht mit einer niedrigeren Versionsnummer empfängt, sendet er sein Programm per Broadcast an das

Netzwerk. Empfängt er hingegen eine Nachricht mit einer höheren Versionsnummer, so sendet er sofort eine Nachricht mit seiner älteren Versionsnummer, um so den anderen Knoten zu veranlassen, ihm die neue Version des Prgramms zu senden. Anschließend empfängt er die neue Version, schreibt sie in seinen externen Flash-Speicher und reprogrammiert sich möglichst sofort selbst.

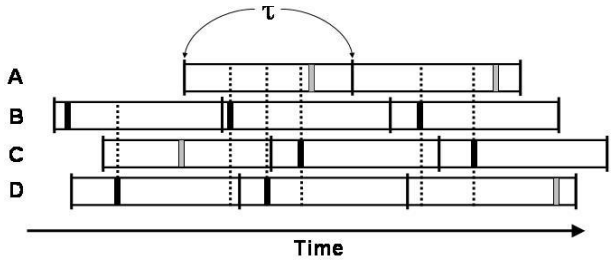
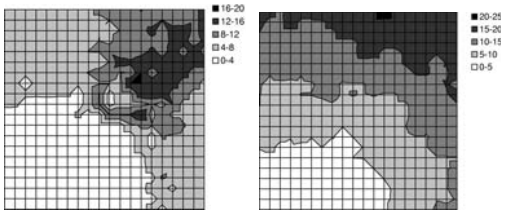


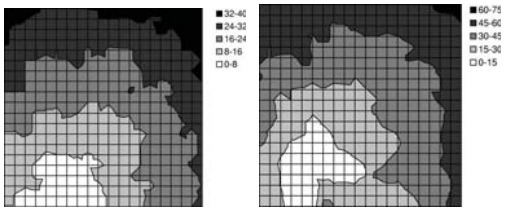
Abbildung 3: Beispiel für das short-listen-Problem, entnommen aus[13]. Die langen Striche sind die Intervallgrenzen. Dicke schwarze Striche sind die gesendeten Nachrichten, dicke helle Striche sind nicht gesendete Nachrichten. Die gestrichelten Linien verdeutlichen die Unterdrückung der Nachrichten der anderen Knoten im momentanen Intervall.

In nicht zeitsynchronisierten Netzen kann es mit Trickle zum so genannten "short-listen"-Problem kommen. Dabei führt die Verschiebung der Intervalle τ zwischen den Knoten dazu, dass mehr Nachrichten mit Metadaten gesendet werden als es in einem synchronen Netz der Fall wäre. Abbildung 3 zeigt ein Beispiel für das Problem. Die redundanten Nachrichten behindern die restliche Kommunikation im Netz und führen so zu einem eventuell höheren Energieverbrauch der Knoten. Um das Problem zu reduzieren, wird t in der Regel auf einen Wert im Intervall $[\frac{\tau}{2}, \tau]$ statt $[0, \tau]$ gesetzt. Dadurch reduziert sich die Zahl der redundanten Nachrichten fast auf den Wert von synchronen Netzen.
Eine weitere Methode, um die Zahl der gesendeten Nachrichten zu reduzieren, ist die dynamische Anpassung des Zeitintervalls τ. Dabei werden für das Intervall eine Untergrenze τ_l und eine Obergrenze τ_h festgelegt. τ wird zu Beginn auf τ_l gesetzt. Nach dem Ablauf eines Intervalls wird τ bis auf maximal τ_h verdoppelt. Beim Empfang einer Nachricht mit einer neueren Versionsnummer oder einer neuen Version des Programms wird τ wieder auf τ_l gesetzt. Auf diese Weise senden Knoten, die keine neuen Informationen haben mit der Zeit immer weniger Nachrichten. Das Verteilen einer neuen Version des Programms wird so nicht beeinträchtigt.
Zur Performanz von Trickle wurden in[13] Messungen sowohl mit dem zu TinyOS gehörenden Simulator TOSSIM[14] als auch in einem echten Netzwerk durchgeführt. Gemessen wurde jeweils die Zeit, die benötigt wurde, um alle Knoten zu reprogrammieren in Abhängigkeit von der Anzahl Hops des Netzwerkes und der Einstellung für τ_h.
Auf den Knoten war Maté im Einsatz, der neue Programmteil bestand aus einem Paket der Größe 30 Byte. Das echte Netzwerk bestand aus 14 Mica2-Knoten und war mehrere Hops breit. Die Simmulierten Netzwerke bestanden aus einem Gitternetz mit 20×20 Knoten mit variablem Abstand und damit variabler Hopzahl. Alle Knoten konnten immer erfolgreich reprogrammiert werden. Die Erhöhung von τ_h von einer Minute auf 20 Minuten führte bei den simulierten Netzen zu keinem nennenswerten Unterschied bei der gemesssenen Zeit. Im realen Netzwerk führte der höhere Wert zu einer starken Erhöhung des Durchschnitts der gemessenen Zeit von 22 auf 32 Sekunden. In den simulierten Netzwerken stieg die Zeit wie erwartet mit der Anzahl der Hops. Das neue Programm breitete sich immer außer im dichtesten Netz immer in der erwarteten Wellenform aus. Abbildung 4 veranschaulicht diese Ausbreitung und die dazu notwendige Zeit.



(a) 5' Spacing, 6 hops

(b) 10' Spacing, 16 hops



(c) 15' Spacing, 32 hops

(d) 20' Spacing, 40 hops

Abbildung 4: Ausbreitung des neuen Programms mit Trickle im simulierten Netzwerk entnommen aus[13]. Die angegebene Anzahl der Hops bezieht sich auf den Abstand zwischen dem Knoten ganz links unten und dem ganz links oben. Die Ausbreitungszeit ist in Sekunden angegeben.

4.3 Deluge

Während Trickle primär dazu gedacht ist, kleinere Programmteile im Netzwerk zu verteilen, ist das darauf basierende Protokoll Deluge[15] auch zur Verteilung größerer Datenmengen geeignet. Deluge wurde ebenfalls an der UC Berkeley entwickelt und ist seit Version 1.1.8 Bestandteil von TinyOS. Es funktioniert ähnlich wie der "polite gossip"-Ansatz von Trickle. Auch hier sendet jeder Knoten den Stand seines Programms periodisch an alle Knoten in Reichweite. Knoten, die auf dem selben Stand sind, halten sich auch hier mit dem Senden zurück. Ein Knoten, der eine Nachricht von einem Knoten auf einem älteren Stand mithört, sendet hier nach einer kurzen, zufälligen Wartezeit, um Kollissionen mit anderen Knoten mit der selben Version zu vermeiden, eine neue Nachricht mit seinem Stand und genaueren Informationen über sein Programm an das gesamte Netz. Ein Knoten, der von einer neueren Version des Programms erfährt, versendet nun eine Nachricht, in der er die ihm fehlenden Programmteile spezifisch anfordert. Ein Knoten, der diese Anforderung hört und erfüllen kann, sendet die angeforderten Teile an das Netz. Knoten, die die gesendeten Teile ebenfalls brauchen, speichern sie genau wie der anfordernde Knoten in ihrem externen Flash-speicher. Durch das stückweise Versenden ist es

doi: 10.2313/NET-2012-08-2_03

in Deluge möglich, die Verbreitung des Programms mittels Pipelining zu beschleunigen, wenn das Netzwerk über mindestens drei Hops verfügt.
Deluge funktioniert im Detail wie folgt: Jedes Programm ist ein Objekt. Jedes Objekt wird in Seiten gleicher Länge eingeteilt, die wiederum aus Paketen gleicher Länge bestehen. Pakete und Seiten verfügen über einen CRC der Länge 16 Bit. Jeder Knoten speichert zu jedem Objekt ein Objektprofil bestehen aus einer Versionsnummer v und einen Altersvektor $\mathbf{a} =< a_0, a_1, .., a_{p-1} >$, der angibt, welche Seite wann zuletzt geändert wurde. Seite i wurde in Version $v - a_i$ geändert. v wird bei jeder Änderung des Programms um eins erhöht. Wie in Trickle wird die Zeit auch hier in Intervalle unterteilt. Jeder Knoten besitzt die Variablen $\tau_{m,i}$, die die Länge des i-ten Intervalls angibt, τ_l und τ_h, die die Ober- und Untergrenze der Intervalllänge angeben, den festen Wert k und eine Zusammenfassung $\phi = \{v, \gamma\}$ seines Programmstandes. Dabei ist v die Versionsnummer seines Programms und γ die Nummer der verfügbaren Seite mit der höchsten Nummer. Eine Seite ist bei Deluge verfügbar, wenn alle Pakete davon vorhanden und fehlerfrei sind und alle Seiten mit niedriegeren Nummern ebenfalls verfügbar sind. außerdem befindet sich jeder Knoten immer in einem von drei Zuständen, Maintain, Receive(RX) oder Transmit(TX). Ein Knoten startet im Zustand Maintain. Nur in diesem Zustand teilt der Knoten die Zeit in Intervalle ein. Der Intervall i beginnt zum Zeitpunkt $t_i = t_{i-1} + \tau_{m,i-1}$. Zu Beginn jedes Intervalls überprüft der Knoten, ob er im letzten Intervall eine Nachricht mit einer Zusammenfassung $\phi' \neq \phi$ oder eine Anfragenachricht oder Seitenpakete empfangen oder mitgehöht hat. Ist nichts davon der Fall, so wird $\tau_{m,i} = min(2\tau_{m,i-1}, \tau_h)$ gesetzt, also die Länge dieses Intervalls bis zur Obergrenze verdoppelt. außerdem wird ein Zufallswert r_i aus dem Intervall $[\frac{\tau_h}{2}, \tau_h]$ erstellt. Dies geschieht aus dem selben Grund wie bei Trickle, um das short-listen-Problem zu reduzieren. Hat der Knoten jedoch eine der oben genannten Nachrichten empfangen oder mitgehöhrt, so setzt er $\tau_{m,i}$ auf τ_l zurück und beginnt sofort ein neues Intervall. Zum Zeitpunkt $t_i + r_i$ im Intervall i überprüft der Knoten, ob und wie viel Zusammenfassungen $\phi' = \phi$ er in diesem Intervall bereits empfangen hat. Hat er k oder weniger empfangen, sendet er selbst seine Zusammenfassung ϕ. Hat der Knoten jedoch vor dem Zeitpunkt $t_i + r_i$ eine Zusammenfassung ϕ' mit $v' < v$, also einer kleineren Versinsnummer als seine eigene, empfangen, so sendet er bei $t_i + r_i$ sein Objektprofil an das Netzwerk, sofern er nicht vorher k oder mehr Objektprofile mit der gleichen Versionsnummer mitgehöhrt hat, die ander Knoten bereits geschickt haben. Empfängt ein Knoten ein Objektprofil, so benutzt er es, um seine Zusammenfassung zu aktualisieren. v wird auf $v\prime$ gesetzt und mittels $\mathbf{a}'$ wird der höchstmögliche Wert für γ gesetzt. Ein Knoten, der im Intervall i eine Zusammenfassung ϕ' mit $v' = v$ und $\gamma' > \gamma$ empfängt, prüft einerseits, ob er innerhalb der Zeit $2\tau_{m,i}$ zuvor eine Anfrage für eine Seite mit Nummer $p \leq \gamma$ empfangen hat und andererseits, ob er ein Paket einer Seite mit Nummer $p \leq \gamma$ erhalten hat. Trifft keiner dieser Fälle zu, so wechselt er in den Zustand RX. Empfängt ein Knoten im Zustand Maintain eine Anfrage für eine Seite mit Versionsnummer v und Seitennummer $p \leq \gamma$, so wechselt er in den Zustand TX. Ist der Knoten im Zustand RX, so wartet er für einen Zeitraum $\omega T_{tx} + r$. ω ist dabei eine festgelegte Anzahl an Paketen, T_{tx} ist die Zeit, die ein Paket zum senden braucht, r ist ein Zufallswert in der Größenordnung des festgelegten Wertes τ_r. Hat er bis dahin kein Paket und keine Anfrage erhalten, so sendet er eine Anfrage nach dem nächsten benötigten Paket. Erhält er ein Paket, so überprüft er den CRC und wartet erneut für den Zeitraum $\omega T_{tx} + r$ bis er die nächste Anfrage schickt. Sobald der Knoten eine feste Anzahl λ an Anfragen verschickt hat, auf die er fehlerfreie Pakete nur mit einer Rate unter einem Schwellwert α empfangen hat, so wechselt er trotz unvollständigem Objekt wieder in den Zustand Maintain. Diese Verhalten dient dazu, asymmetrische Links tolerieren zu können. Ein Knoten im Zustand RX, der alle Pakete für die angeforderten Seiten erhalten und auf Fehlerfreiheit überprüft hat, aktualisiert sein Objektprofil und wechselt in den Zustand Maintain zurück. Ein Knoten im Zustand TX verfügt über eine Menge Π_{tx} an angeforderten und noch zu versendenden Paketen. Sobald er eine Anforderung erhält, die die Menge Π'_{rx} anfordert, wird diese Menge der der schon zu versendenden Pakete zugeschlagen $\Pi_{tx} = \Pi_{tx} \cup \Pi'_{rx}$. Der Knoten versendet alle Pakete aus Π_{tx} im round-robin Verrfahren und entfernt sie dann aus Π_{tx}. Sobald $\Pi_{tx} = \emptyset$ ist, wechselt der Knoten in den Zustand Maintain zurück. Abbildung 5 stellt ein vereinfachtes Zustandsdiagramm des Protokolls dar.

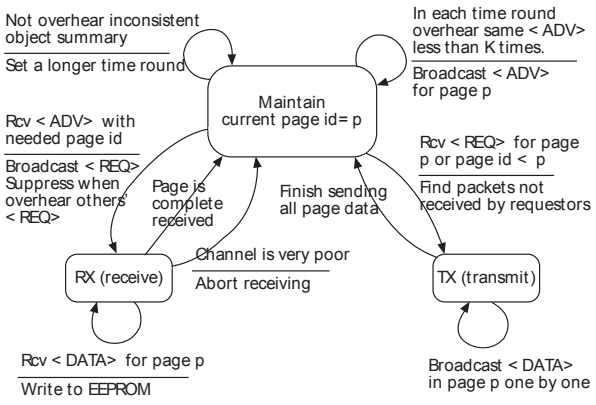


Abbildung 5: Vereinfachtes Zustandsdiagramm für Deluge, entnommen aus [4]. <ADV> (Advertisement) steht für Nachrichten mit Zusammenfassung, <REQ> (Request) für Anfragen.

Wie Trickle wurde auch Deluge sowohl in einem realen Netwerk als auch in der Simulation mit TOSSIM getestet. Das reale Netz bestand aus 75 Mica2-Knoten. τ_l wurde auf 2 Sekunden, τ_h auf 60 Sekunden und k auf 1 gesetzt. Diese Werte entsprechen denen aus den Tests für Trickle in Abschnitt 4.2. τ_r wurde auf 0.5 Sekunden gesetzt, λ auf 2 und ω auf 8. Für den Test wurde das reale Netzwerk mit einer variierenden Anzahl an Seiten reprogrammiert. Eine Seite hatte die Größe 1104 Byte und war in 48 Pakete zu je 23 Byte aufgeteilt. In jedem Durchlauf konnten alle Knoten fehlerfrei reprogrammiert werden. Die Zeit um das Netzwerk zu reprogrammieren stieg nicht linear mit der Anzahl der Seiten, sondern langsamer. Der Grund dafür war das Pipelining von Deluge. Jeder Knoten kann bereits erhaltene Seiten weiter verbreiten, ohne auf die restlichen Seiten des Objekts warten zu müssen, was die Ausbreitung vor allem bei größeren WSNs stark beschleunigt. Deluge erreichte in diesem Test eine durchschnittliche Datenübertragungs-

rate von 88,4 Byte/ Sekunde. Dies ist wesentlich schneller als die etwa 1,5 Byte/Sekunde von Trickle. Der Overhead an Zusammenfassungs- und Anfragenachrichten betrug zusammen 18.18%. Der Energieverbrauch ist bei Trickle und Deluge in etwa gleich, da alle Knoten permanent eingeschaltet bleiben müssen.
Die simulierten Netzwerke bestanden bei Deluge wie bei Trickle aus einem Gitternetz von 20×20 Knoten mit variablem Abstand. Hier wurde eine variable Zahl von Seiten der Größe 552 Byte aus 24 Paketen zu je 23 Bytes zum reprogrammieren des Netzes verwendet. Der Abstand zwischen zwei Knoten betrug 10 beziehungsweise 15 Fuß. Wie in Abbildung 6 zu sehen ist, geschieht die Ausbreitung bei 15 Fuß Abstand im erwarteten Wellenmuster, während das bei 10 Fuß Abstand nicht der Fall ist. Dies ist auf das hidden-terminal-problem zurückzuführen. Dabei senden zwei Knoten, die sich gegenseitig nicht sehen können, gleichzeitig an den gleichen Knoten, was zu Kollissionen und damit zum Paketverlust führt. Die Datenrate, mit der sich das neue Programm ausbreitet ist auch hier im Vergleich zu Trickle wesentlich höher. Wegen der Unterstützung größerer Datenmenge und der höheren Datenrate ist Deluge Trickle wesentlich überlegen und daher vorzuziehen.

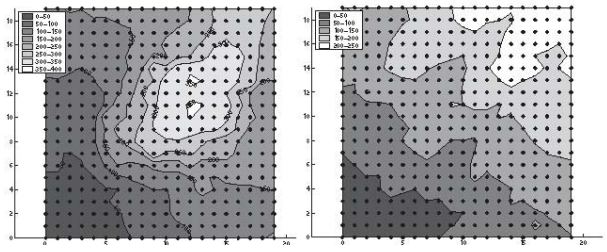


Abbildung 6: Ausbreitung einer neuen Seite bei Deluge. Links mit 10 Fuß Abstand zwischen den Knoten rechts mit 15 Fuß. Zeitangaben in Sekunden. Einzelbilder entnommen aus[15]

Parallel zu Deluge wurde für TinyOS der Bootloader TOSBoot entwickelt. Dieser erlaubt einem Knoten, mehrere Programmabbilder im externen Flash-Speicher zu halten und jeweils eins davon nach eine Neustart in den Programmspeicher zu laden. Deluge unterstützt dies, indem es bei seinen Zusammenfassungsnachrichten immer die Objektprofile aller Programme mitschickt.

4.4 Zephyr

Zephyr[16] wurde 2009 an der School of Electrical and Computer Engeneering der Purdue University entwickelt. Aufbauend auf Deluge beschleunigt Zephyr die Reprogrammierung von WSNs, indem es die Menge der zu versendenden Daten auf ein Minimum zu reduzieren versucht. Dazu bedient es sich einerseits einer Methode aus dem Deluge ähnlichen Protokoll Stream[17] und andererseits einer optimierten Variante des Rsync-Algorithmus[18]. Zephyr wurde zuerst für TinyOS implementiert, kann jedoch ohne Probleme von Contiki verwendet werden.
Stream reduziert die Größe der zu versenden Programmabbilder, indem es die Komponenente, die für das Ausführen der Reprogrammierung notwendig ist, nicht als Bestandteil des Programms sondern als seperate Komponente behandelt. Diese Komponenete wird von den einzelnen Knoten nicht im Programmspeicher sondern im externen Flash-Speicher gespeichert, das Programm enthält nur den Code, der notwendig ist, um Aktuallisierungen zu erkennen. Sobald der Knoten reprogrammiert werden muss, ruft das Programm des Knotens den Bootloader auf. Dieser lädt nun die Reprogrammierkomponente in den Programmspeicher und startet den Knoten neu. Die Reprogrammierkomponente führt nun die Reprogrammierung durch und ruft erneut den Bootloader auf, der daraufhin das neue Programm in den Programmspeicher lädt und den Knoten erneut neu startet. Auf diese Weise kann das neue Programm ohne die Komponente im Netz verteilt werden. Dieser Ansatz ermöglicht es ein einerseits, die Größe des Programmabbildes zu reduzieren und dadurch sowohl die Verteilung des Programms im Netzwerk als auch die Reprogrammierung der einzelnen Knoten zu beschleunigen, andererseits verhindert er aber Änderungen an der Reprogrammierkomponente und belegt permanent einen Platz im externen Flash-Speicher der Knoten.
Zephyr versendet jedoch nicht das kürzere Programmabbild von Stream, sondern nur ein so genanntes delta script, das nur den Unterschied auf zwischen zwei Versionen des Abbildes beinhaltet. Das neue Abbild wird mit Hilfe des delta scripts aus dem alten Abbild erstellt. Um das delta script zu erstellen, verwendet Zephyr eine angepasste Version des Rsync Algorithmus. Dieser wurde 2000 an der Australien National University entwickelt und diente ursprünglich dazu, Daten zwischen zwei Rechnern über ein Netzwerk mit geringer Bandbreite zu synchronisieren. Zephyr modifiziert Rsync so dass teuere Rechenoperationen nicht auf den Knoten, sondern nur auf dem an die Basisstation angeschlossenen Rechner ausgeführt werden müsen. Der modifizierte Algorithmus unterteilt das alte und das neue Abbild in Blöcke fester Länge. Anschließend berechnet er für jeden Block des alten Abbildes zwei Hashwerte. Der erste wird mit einer in[18] spezifizierten schnellen Methode berechnet, der zweite mittels der kryptographischen Hashfunktion MD4[19]. Beide Werte werden für jeden Block in einer Hashtabelle gespeichert. Dann wird für jeden Block im neuen Abbild der schnelle Hashwerte berechnet. Für jeden Block wird mittels der Hashtabelle nach einem gleichen Block im alten Abbild gesucht. Bei einem Treffer wird auch der MD4 Wert des neuen Blocks berechnet. Dies dient dazu, Kollissionen möglichst auszuschließen. MD4 wird nur bei möglichen Treffern berechnet um Rechenzeit zu sparen. Anschließend wird das delta script erstellt. Für jeden Block aus dem neuen Abbild, der eine Entsprechung im alten hat, wird die Codezeile
`COPY <oldOffset> <newOffset> <len>`
erstellt, für jeden Block ohne Entsprechung die Zeile
`INSERT <newOffset> <len>`.
Das delta script wird nun optimiert indem der längste mit dem alten Abbild übereinstimmende Abschnitt im neuen Abbild gesucht wird. Dies erreicht man durch suchen der längsten ununterbrochenen Folgen von `COPY` -Anweisungen. Diese `COPY`-Anweisungen werden nun zu einer verschmolzen.

```
COPY <old> <new> <B>
COPY <old+1> <new+1> <B>
...
COPY <old+k> <new+k> <B>
```

wird zu

```
COPY <old> <new> <(k+1)*B> .
```

So wird das delta script wesentlich verkleinert. Jedoch kann

es trotzdem auch bei kleinen Änderungen im Programm zu großen delta scripts kommen. Werden etwa durch die Veränderungen am Programm die Anfangsadressen von Funktionen verschoben, so ändern sich im binären Abbild alle Aufrufe dieser Funktionen, was das delta script stark aufblähen kann. Um solchen und ähnlichen Problemen entgegenzuwirken nimmt Zephyr auch Änderungen des Programms auf höherer Ebene vor. Diese Änderungen geschehen mit allen Versionen des Programms, auch mit der ersten und immer vor der Erstellung des delta scripts. Zephyr modifiziert dafür das Linking des Programms. Eine dieser Modifikationen sorgt dafür, dass die Unterbrechungsbehandlung des Programms trotz Änderungen immer an der gleichen Stelle im Speicher steht. Die zweite Änderung ist das Erstellen einer Tabelle für indirekte Funktionsaufrufe. Funktionen werden in einem Programm in der Regel mehr als einmal aufgerufen. Um die dadurch nötigen Änderungen am Programm auf ein Minimum zu beschränken, erstellt Zephyr am Ende des Programmspeichers eine Tabelle mit den Startadressen aller Funktionen. Alle Funktionsaufruf werden so geändert, dass sie die entsprechende Stelle in dieser Tabelle aufrufen, von wo aus dann wiederum die Funktionen selbst aufgerufen werden. Abbildung 7 stellt ein Beispiel für eine solche Tabelle dar.

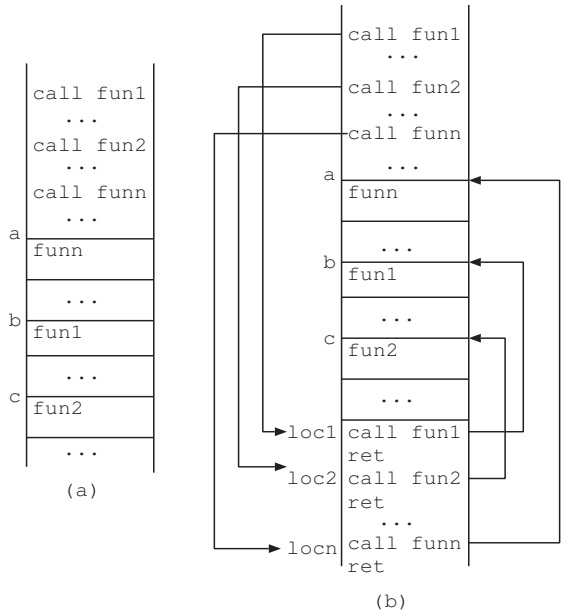


Abbildung 7: Beispielprogramm ohne(a) und mit(b) Tabelle für indirekte Funktionsaufrufe. Entnommen aus[16]

Ändert sich nun die Anfangsadresse einer Funktion, so muss nur das Ziel des Aufrufs in der Tabelle geändert werden, was nur eine Zeile im delta script ausmacht. Nach den Änderungen auf höhere Ebene, dem Erstellen und der ersten Optimierung des delta scripts führt Zephyr noch weitere Optimierungen durch. Die erste Optimierung ist das ersetzen von Anweisungsblocks bestimmter Form durch Metaanweisungen. Blöcke, bei denen sich mehrmals `COPY`-Anweisungen großer Länge mit `INSERT`-Anweisungen kleiner Länge abwechseln, werden durch `CIW(COPY_WITH_INSERTS)`-Anweisungen ersetzt, die aus einem Kopierbefehl und den Adressen und Daten für die Einfügungen bestehen. So wird das delta script um mehrere `COPY`-Anweisungen kürzer. Blöcke, in denen bestimmte Folgen mit leichten Unterschieden hintereinander eingefügt werden, werden durch `REPEAT`-Anweisungen ersetzt, die die unveränderliche und die sich ändernden Folgen mit Adressen beinhalten. So werden mehrere `INSERT`-Anweisungen eingespart. Die letzte Optimierung ist das entfernen aller `<newOffset>` Teile aller Anweisungen. Dies ist möglich, da alle Anweisungen der Reihe nach ausgeführt werden und die Länge ihrer Daten beinhalten.
Um die Reprogrammierung des Netzwerks zu starten, sendet die Basisstation eine Rebootanweisung an das Netz und ruft ihren Bootloader auf. Dieser lädt die Reprogrammierkomponente in den Programmspeicher und startet den Knoten neu. Jeder Knoten, der die Rebootanweisung erhält, sendet sie an das gesamte Netzwerk weiter und verfährt dann ebenso wie die Basisstation. Auf diese Weise wird das Netz mit der Rebootanweisung kontrolliert geflutet. Sobald alle Knoten die Reprogrammierkomponente gestartet haben, erhält die Basissation das delta script und verteilt es per Deluge beziehungsweise Stream im Netz. Dabei wird eine dynamische Seitengröße benutzt um nicht unnötig Bandbreite durch Padding zu verbrauchen. Ein Knoten, der das delta script vollständig erhalten hat, verwendet es, um aus dem Abbild des alten Programms in seinem externen Flash-Speicher das Abbild des neuen Programms zu erstellen. Abschließend wird das Netz mit einer erneuten Rebootanweisung geflutet, woraufhin alle Knoten mit dem neuen Programm rebooten. Abbildung 8 zeigt die Belegung des Speichers eines Knotens während der Reprogrammierung.

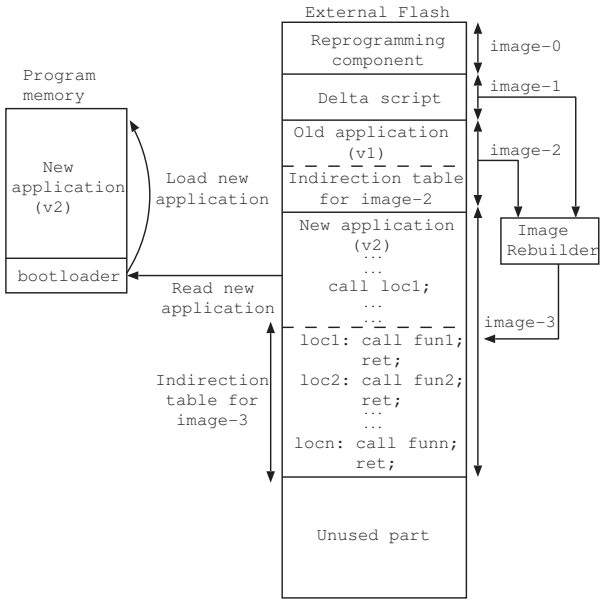


Abbildung 8: Belegung des Speichers eines Knotens während der Reprogrammierung mit Zephyr. Entnommen aus[16]

Um die Performanz von Zephyr zu testen, wurden in [16] reale und simulierte WSNs aus Mica2-Knoten verwendet. Die realen Netze waren einerseits 2×2, 3×3 und 4×4 große Gitter und andererseits Ketten von 2 bis 10 Knoten Länge. Die mit TOSSIM simulierten Netze waren Gitter der Größe 6×6 bis 14×14. Als Testfälle wurden Änderungen an den bei TinyOS mitgelieferten Standardprogrammen

doi: 10.2313/NET-2012-08-2_03

wie CntToLeds, das einen internen Zähler über die Leds des Kontens ausgibt, und an dem real verwendetem Programm eStadium[20] mit unterschiedlicher Tragweite, von einzelnen neuen Parametern bis zum Austausch des kompletten Programms, verwendet. Alle Netze wurden mit allen Testfällen reprogrammiert. Jede Reprogrammierung wurde mit Deluge, Stream, dem einfachen Rsync-Algorithmus, dem optimierten Rsync-Algorithmus ohne Änderungen auf höherer Ebene und Zephyr durchgeführt. Gemessen wurden jeweils die Zeit zum Reprogrammieren des gesamten Netzes und die Menge der insgesamt versendeten Pakete. Zephyr stellt sich als wesentlich schneller als die anderen Ansätze heraus. Um wie viel schneller Zephyr ist, hängt von der Tragweite der Änderungen am Programm ab. Abbildung 9 zeigt die benötigte Zeit zur Reprogrammierung und die Menge der gesendeten Pakete für eine möderate Änderung an eStadium in den simulierten WSNs. Das Verhältniss der Performanz zwischen den Verfahren ist in allen Fällen ähnlich, jedoch unterscheidet sich der maximale Vorteil von Zephyr stark. So ist Zephyr in der Simmulation mit moderaten Änderungen am Programm bis zu 92,9 mal schneller als Deluge, während es in realen Netzen bei sehr großen Änderungen im Durchschnitt nur 2,1 mal so schnell ist. Auffällig ist, das Zephyr gegenüber dem optimierten Rsync-Algorithmus bei größeren Änderungen schneller wird, während bei allen anderen Verfahren das Gegenteil der Fall ist. Die Zahl der versendeten Pakete ist bei Deluge zwischen 2,5 und 146 mal so groß wie bei Zephyr.

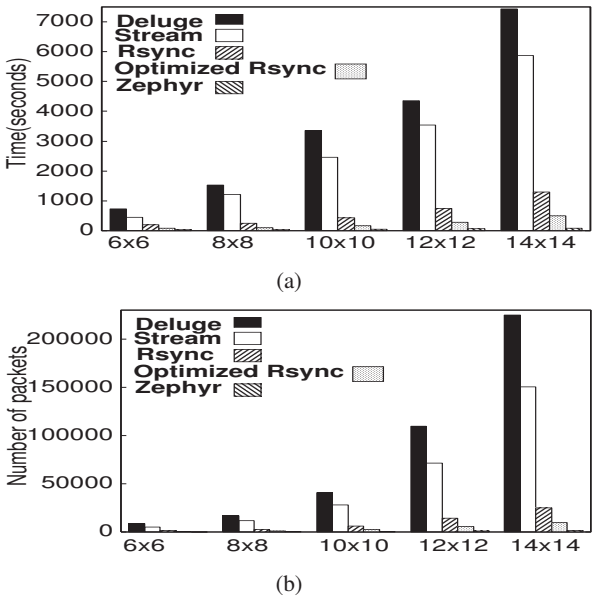


Abbildung 9: Reprogrammierungszeit(a) und Anzahl der gesendeten Pakete(b) bei moderater Änderung an eStadium in den simulierten Netzen. Entnommen aus[16]

4.5 Selective Reprogramming Scheme von Krüger, Pfisterer und Buschmann

In einem WSN läuft nicht zwangsläufig das selbe Programm auf jedem Knoten. Zum Beispiel ist es nicht nötig, Knoten am Rand des Netzes mit Aggregationsfunktionen auszustatten. Neue Versionen dieser Funktionen müssen so auch nur an bestimmte Knoten verteilt werden. Ein neues Protokoll, dass diese Reprogrammierung einer Untergruppe von Knoten eines WSNs unterstützt ist das von Krüger, Pfisterer und Buschmann an der Universität Lübeck entwickelte Selective Reprogramming Scheme[21].
Das Protokoll unterteilt die Reprogrammierung in vier Phase: Die erste ist die Node-Discovery-Phase, in der die zu reprogrammierenden Knoten ausgewählt und ein Routing-Baum zu ihnen eingerichtet wird. Die zweite ist die Code-Desssemination-Phase, in der das neue Programm an die ausgewählten Knoten verteilt wird. In der dritten Phase, Request Missing Parts, fragen die Knoten nach den nicht erhaltenen Programmteilen. Die letzte Phase ist Finalization and Reboot, in der die Knoten mit dem neuen Programm neu gestartet werden.
Die Node-Discovery-Phase läuft wie folgt ab: Die Basisstation flutet das Netzwerk mit einem Weckruf, so dass alle Knoten aktiv werden. Der Weckruf beinhaltet den Zeitpunkt, an dem der nächste Schritt durchgeführt wird. Dieser Schritt besteht darin, dass die Basisstation ein Präsenzerkennungspaket versendet, mit dem ein Routing-Baum im Netz aufgebaut wird. Ein Knoten, der das Präsenzerkennungspaket erhält, sendet ein Präsenzantwortpaket mit seiner ID über den Routing-Baum an die Basisstation. Dabei werden auf der Verbindungsschicht Acknowledgements verwendet, um alle Antwortpakete zuverlässig zu übermitteln. Die IDs in den Antwortpaketen werden an jedem Hop aggregiert. Erhält die Basisstation keine Antwort, so wird das Präsenzerkennungspaket erneut gesendet. Sobald die Basisstation über die IDs aller Knoten verfügt, werden die zu reprogrammierenden Knoten ausgewählt. Die Basisstation flutet das Netz mit einem Selektionsanzeigepaket mit ihren IDs. Die ausgewählten Knoten senden jeder ein Selektionsbestätigungspaket über den Routing-Baum zurück. Jeder dabei beteiligte Knoten wird zu einem Weiterleitungsknoten. Sind alle Selektionsbestätigungspakete an der Basisstation angekommen, so beginnt die Code-Desssemination-Phase.
In der zweiten Phase wird das Netzwerk mit dem neuen, in Segmente unterteilten Programm probabilistisch geflutet. Das bedeutet, dass jeder Knoten ein neu erhaltenes Paket nicht wie beim einfachen Fluten immer per Broadcast im Netzwerk verteilt, sondern nur mit einer bestimmten Wahrscheinlichkeit. Jedes Paket enthält neben seinem Segment auch die Gesamtzahl alle Segmente. Anhand dieser Information und dem Zeitunterschied zwischen der Ankunft der Pakete schätzt jeder Knoten die restliche Dauer dieser Phase bei jedem erhaltenen Paket neu. Nach Ablauf der geschätzten Zeit gehen alle Knoten in die dritte Phase über.
In der Request-Missing-Parts-Phase überprüft jeder Knoten, welche Segmente er erhalten hat und welche nicht. Daraufhin sendet er eine Nachricht mit einem Vektor mit dieser Information an alle erreichbaren Knoten. Nachdem er die entsprechenden Nachrichten der Nachbarknoten erhalten hat, sendet er zufällig bis zu 15 Segmente, die er hat und die mindestens ein Nachbar braucht an das Netz. Dieses abwechselnde Senden des Vektors und der Segmente wird wiederholt, bis der Knoten alle bei den Nachbarn verfügbaren Segmente erhalten hat. Die noch fehlenden Segmente werden über den Routing-Baum bei der Basisstation angefordert. Über den Routing-Baum wird die Basisstation auch über den die in jedem ausgewählten Knoten vorhandenen Segmente informiert. Sobald die Basisstation weiß, dass alle ausgewählten

Knoten das gesamte Programm haben, beginnt Phase vier. In der letzten Phase, Finalization and Reebot, flutet die Basisstation das Netz mehrmals mit Rebootpaketen. Jeder ausgewählte Knoten, der ein solche Paket erhält, lädt das neue Programm und startet sich neu.
Das Selective Reprogramming Scheme wurde in[21] evaluiert. Dabei wurde ein reales WSN aus 22 iSense-Knoten[22] und mehrere mit Shawn[23] simmulierte Netze aus den gleichen Knoten der Größen 15, 20×20 und 30×30 verwendet. Da die iSense-Knoten in verschieden Konfigurationen mit stark unterschiedlichen Leistungswerten vorkommen[24, 25] und die Konfiguration in[21] nicht angegeben ist, sind die Ergebnisse schlecht mit den anderen Datendisseminationsprotokollen in dieser Arbeit vergleichbar. Außerdem wurden kaum mit den Test der anderen Protokolle vergleichbare Werte gemessen. Hier werden nur die für den Vergleich interessanten Messungen aufgeführt. Für den Test wurde jedes Netzwerk mit einem Programm aus 1200 Paketen nicht näher definierter Größe aufgeteilt auf 600 Segmente reprogrammiert. Es wurden verschieden Algorithmen für das Senden der Pakete der Knoten entlang des Routing-Baums verglichen. Auf diese Algorithmen wird hier nicht näher eingegangen.
Bei allen Tests war die Reprogrammierung aller Knoten erfolgreich. Dauer und Anzahl gesendeter Pakete in Phase 1 abhängig von Algorithmus und Anteil der ausgewählten Knoten am Netzwerk wurden im realen WSN gemessen. Die Zeitdauer dieser Phase stieg kaum mit dem Anteil am Netzwerk, sie lag bis auf wenige Ausreisser unter 625 ms. Die Menge an der Basisstation empfangenen Pakete stieg mit dem Anteil der ausgewählten Knoten von 2 auf maximal 10. Eine weitere interessante Vergleichsgröße ist die Dauer von Phase 3 im simulierten 15 Netzwerk und die Anzahl der dabei versendeten Nachrichten, beide abhängig vom verwendeten Algorithmus und dem Anteil der ausgewählten Knoten am Netz. Bei 5% Anteil der ausgewählten Knoten betrug die Dauer beim schnellsten Algorithmus 50 Sekunden, beim langsamsten 175 Sekunden. Bei steigendem Anteil näherten sich die beiden Dauern an bis sie bei 100% beide bei 135 Sekunden lagen. Die Menge der gesendeten Nachrichten lag beim langsamsten Algorithmus bei 5% Anteil bei 50000, während er beim schnellsten bei 150000 lag. Mit steigendem Anteil näherten sich beide Mengen wieder an und stiegen beide bis auf 375000 bei 100% Anteil. Aus diesen Messwerten kann der Overhead wie folgt berechnet werden:
$$\frac{\#gesendete\ Pakete - 225 \times Anteil\ ausgewaehlte\ Knoten \times 1200}{\#gesendete\ Pakete}$$
Der Overhead ist also im ungünstigsten Fall 90%, im günstigsten 18%.

5. ZUSAMMENFASSUNG

In dieser Arbeit wurde ein Überblick über die Reprogrammierung von drahtlosen Sensornetwerken gegeben. Zuerst wurde der grundlegende Aufbau eines drahtlosen Sensornetzwerks beschrieben. Daraufhin wurden die Handhabung der Reprogrammierung auf den einzelnen Knoten durch die Betriebssysteme TinyOS und Contiki vorgestellt. TinyOS macht es erforderlich, bei jeder Änderung das gesamte Programm des Knotens auszutauschen, während Contiki Teile des Programms ohne Neustart des Knotens austauschen kann. Anschließend wurden die für die Verteilung des neuen Programms im Netzwerk verantwortlichen Datendisseminationsprotokolle XNP, Trickle, Deluge, Zephyr und das Selective Reprogramming Scheme von Krüger, Pfisterer und Buschmann im einzelnen beschrieben und deren Performanz soweit möglich miteinander verglichen.
XNP ist ein relativ einfaches Protokoll, das eine rudimentäre Unterstützung der drahtlosen Reprogrammierung für alte Versionen von TinyOS darstellte. Trickle verwendet den "polite gossip"-Ansatz, um einzelne neue Programmteile zu verteilen. Die Knoten versenden dabei möglichst nur neue Informationen über den Stand des Programms. Wird bei einem Nachbarknoten ein alter Stand festgestellt, so wird er sofort auf den neuseten gebracht. Deluge baut auf Trickle auf und erweitert es um die Unterstützung für die Verteilung größerer Datenmengen. Einzelne Pakete des neuen Programms werden hier gezielt angefordert. Deluge unterstützt auch das Pipelining bei der Verteilung des neuen Programms und kann gegenüber Trickle eine höhere Veteilungsgeschwindigkeit vorweisen. Zephyr baut wiederum auf Deluge auf und versendet keine ganzen Abbilder des neuen Programms sondern nur delta scripts, mit denen das neue Abbild aus dem alten erstellt werden kann. Zephyr verwendet Ansätze aus dem Protokoll Stream, eine optimierte Version des Rsync-Algorithmus und Änderungen des Programmabbildes auf höherer Ebene um die Größe der delta scripts stark zu verringern. Durch die geringe zu sendende Datenmenge hat Zephyr gegenüber Deluge einen starken Geschwindigkeitsvorteil. Das Selective Reprogramming Scheme schließlich dient dazu, nur Teile des Netzwerks zu reprogrammieren. Dazu wählt es zuerst die zu reprogrammierenden Knoten aus und baut einen Routing-Baum mit ihnen auf. Das Netz wird mit den neuen Paketen geflutet, die Rückmeldungen erfolgen über den Routing-Baum. Das Selective Reprogramming Scheme ist wegen seines anderen Ansatzes und der verwendeten Testmethoden schlecht mit den anderen Protokollen vergleichbar. Fast alle Protokolle konnten alle Knoten eines Netzwerks zuverlässig reprogrammieren, Fehler gab es nur in geringem Ausmaß bei XNP.
Insgesamt ist zu sehen, dass sich der Funktionsumfang und die Performanz von Datendisseminationsprotokollen im Laufe der Zeit stetig verbessert haben. Beides ging jedoch stets mit einer Steigerung bei der Komplexität und dem Ressourcenbedarf (Rechenzeit, Speicherplatz auf dem Knoten) einher. Für reale Sensornetze erscheint es daher sinnvoll, ein Datendisseminationsprotokoll zu verwenden, das die Anforderungen an die Reprogrammierung des Netwerks erfüllt und dabei selbst möglichst simpel und Ressourcensparend ist. So genügt es, für Netzwerke mit nur einem Hop und wenigen Knoten, XNP zu verwenden. Bei größeren Netzen ist mindestens Trickle erforderlich, da jedoch nicht nur kleine Teile eines Programms ausgetauscht werden, wird meist Deluge notwendig. Zephyr bietet gegenüber Deluge zwar einen starke Performanzvorteil, benötigt jedoch viel Platz im externen Flash-Speicher des Knotens und kann nur verwendet werden, wenn dieser nicht anderweitig gebraucht wird. Wenn der Speicher zur Verfügung steht, rechtfertigt die höhere Performanz Zephyrs Komplexität. Das Selective Reprogramming Scheme ist komplexer als alle anderen Protokolle in dieser Arbeit und sollte nur verwendet werden, wenn die Reprogrammierung einzelner Knoten notwendig ist.

6. LITERATUR

[1] Crossbow Technology, Inc., *Mica2 Wireless Measurement System Datasheet*, 2003.

[2] Crossbow Technology, Inc., *TelosB Datasheet*, 2004.

[3] Oracle America, Inc., *Sun SPOT Main Board Technical Datasheet*, 2010

[4] Q. Wang, Y. Zhu, L. Chen *Reprogramming Wireless Sensor Networks: Challenges and Approaches*, In IEEE Network, Seiten 48-55, Mai/Juni 2006.

[5] S. Singh, M. Singh, D. Singh *Routing Protocols in Wireless Sensor Networks - A Survey* In International Journal of Computer Science & Engineering Survey, Seiten 63-83, November 2010.

[6] TinyOS Community, *TinyOS FAQ* `http://docs.tinyos.net/tinywiki/index.php/FAQ`.

[7] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler *The nesC Language: A Holistic Approach to Networked Embedded Systems*, In Proceedings of SIGPLAN'03, 2003.

[8] P. Lewis, D. Culler *Maté: A Tiny Virtual Machine for Sensor Networks*, In Proceedings of ASPLOS-X, October 2002.

[9] A. Dunkels, B. Grönvall, T. Voigt *Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors*, IEEE Emnets, Seiten 455-462, 2004.

[10] A. Dunkels *The Contiki OS* `http://www.contiki-os.org/`

[11] Crossbow Technology, Inc., *Mote In-Network Programming User Reference*, 2003.

[12] J. Jeong, S. Kim, A. Broad *Network Reprogramming*, 2003.

[13] P. Levis, N. Patel, S. Shenker, D. Culler *Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks* In Proceedings of 1st Symposium Networked System Design and Implementation, Seiten 15-28, 2004.

[14] P. Lewis, N. Lee, M. Welsh, D. Culler *TOSSIM: Simulating large wireless sensor networks of TinyOS motes* In Proceedings of the First ACM Conference on Embeded Networked Sensor Systems(SenSys 2003), 2003.

[15] J. Hui, D. Culler *The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale*, In J. Stancovic, A. Arora, R.Govindan(Editoren), SenSys, Seiten 81-94, ACM, 2004.

[16] R. Panta, S. Bagchi, S. Midkiff *Zephyr: Efficient Incremental Reprogramming of Sensor Nodes using Function Call Indirections and Difference Computation* In Proceedings of Proceedings of the 2009 conference on USENIX Annual technical conference, Seiten 32-32, 2009

[17] R. Panta, I. Khalil, S. Bagchi *Stream: Low Overhead Wireless Reprogramming for Sensor Networks*, IEE Infocomm, Seiten 928-936, 2007

[18] A. Trygdell *Efficient Algorithms for Sorting and Synchronization*, Dissertation, Australian National University, 2000

[19] R. Rivest, *RFC 1320*, 1992, `http://tools.ietf.org/html/rfc1320`

[20] `http://estadium.perdue.edu`

[21] D. Krüger, D. Pfisterer, C. Buschmann *Selective Reprogramming in Sensor Networks* In Proceedings of 4th IFIP International Conference on New Technologies, Mobility and Security (NTMS),Seiten 1-5, 2011.

[22] C. Buschmann, D. Pfisterer *iSense: A modular hardware and software platform for wireless sensor networks* Technical report, 6.Fachgespräche Drahtlose Sensornetze der GI/ITG-Fachgruppe Kommunikation und Verteilte systeme, 2007.

[23] S. Fekete, A. Kröller, S. Fischer, D. Pfisterer *Shawn: The fast, highly cunstomizable sensor network simulator*, In Proceedings of the Fourth International Conference on Networked Sensing Systems (INSS'07), 2007

[24] `http://www.coalesenses.com/index.php?page=core-module-2`

[25] `http://www.coalesenses.com/index.php?page=core-module-3`

Energy-efficient communication in Wireless Sensor Networks

Martin Enzinger
Betreuer: Dr. Alexander Klein
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2012
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: martin.enzinger@tum.de

ABSTRACT

This paper describes why energy-efficient communication in Wireless Sensor Networks is crucial and identifies the main sources of energy dissipation as well as counter measures to ensure a long network lifetime. Furthermore the communication protocol stack and the different types of protocols for routing, Medium Access Control and transport are presented and the disparities of the categories are clarified by means of examples.

Keywords

Wireless Sensor Networks, WSN, energy conservation, energy-efficient communication, protocols, Medium Access Control, MAC, routing

1. INTRODUCTION

The ongoing miniaturization of electro mechanical parts and the permanent decrease of costs, lead to a growing number of applications for Wireless Sensor Networks (WSNs). In contrast to other sensing methods WSNs facilitate an areal impression of the measured phenomenon and an in all very close to reality measurement. Due to the constrained resources of the sensor nodes, targeted approaches are required to meet the demands for long-running networks and low latency of data. As most of the energy consumption is originated by sensing, data processing and communication, these operations are the basis for identifying and exploiting energy saving potentials.

2. COMPONENTS OF A WSN

Typically a large number of sensor nodes is randomly deployed within a geographic area to measure within or close to a certain phenomenon. The basic functionality of those sensor nodes is data sensing, data processing and communication with other nodes or base stations. To perform these operations the sensor nodes need to have a power supply - usually a battery, which often cannot be changed and the nodes therefore serve as a one-way product. The sensor nodes collaborate to deliver data from source to sink, hence the nodes are both data originator and data router [17]. [15] distinguishes between terminal nodes which only collect data and the in addition to this data forwarding intermediate nodes. Furthermore the sensor nodes can be consolidated in distinct clusters with a certain cluster head where the sensed data of direct neighbors, which is most likely to have

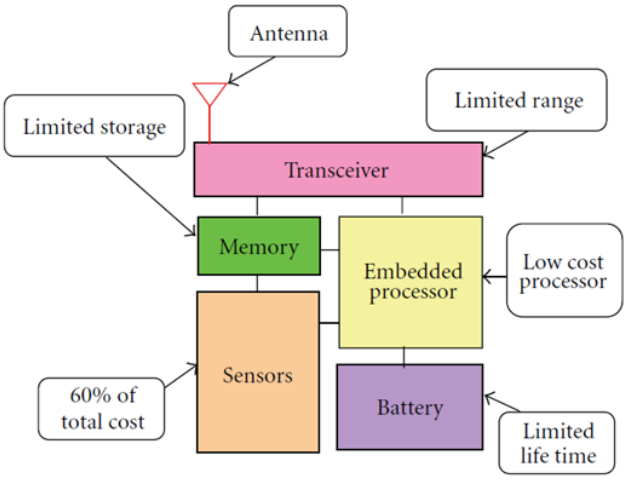


Figure 1: A basic setup of a sensor node [4]

some consonances, is brought together prior to the transmission. All sensing information from the actual sensor nodes is gathered at base stations, whose energy resources are not as limited as the node's energy storage, since they are commonly equipped with a permanent connection to a continual power supply. At those base stations the transmitted data from the sensor nodes is processed and the results are forwarded to the point of delivery where the user can access the collective sensing yield.

3. DESIGN ISSUES OF A WSN

As mentioned before, the resource constraints of the sensor nodes hinder the usage of proven network protocols, inasmuch as resource-conservation is one of the key aspects of routing in WSNs. The commonly huge amount of sensor nodes inhibits the reasonable deployment of a global addressing system, since the expense for ID maintenance would be to high [3]. The number of enclosed sensor nodes can reach up to 10.000 elements dependent on the required granularity of the measurement and robustness requisites [20]. The sensor nodes on their own are not very reliable due to external ascendancies what leads to a varying quantity of sensor nodes. If nodes are added or die, this can have significant impact on communication paths and structure of the

doi: 10.2313/NET-2012-08-2_04

network. Time is a critical factor, not only since the data becomes eventually useless if the latency of the network is to high, but the transmitted data has to cover different numbers of hops respectively depending on the position of the sensor nodes varying distances.

4. SOURCES OF ENERGY DISSIPATION

4.1 Idle listening

Since only being in active mode is a major source of energy consumption and the difference of the expenditures for sending, receiving and waiting for transmission is relatively small, it is crucial to reduce the thus wasted energy resources. Usually the individual sensor node is not at any time involved in a data transmission process and thereby not every component of the node especially the Transceiver does implicitly have to be in an active state. This awaiting ready to transmit data while not receiving or sending packets is called idle listening. There are different approaches to find out when the particular components are not needed or to just reduce the overall active time without further examination. The sleeping sensor nodes either switch back to active mode after a certain time span or after the processing of a wake-up signal.

4.2 Collisions

Collisions occur if nodes receive multiple data packets at the same time. In consequence the received data is useless and the transmission process has to be repeated while energy is dissipated. These re-transmissions can consume quite a lot of energy, since the energy losses are multiplied by the number of hops between source and target [11]. For reran transfers, methods like random delays can impede further collisions [2].

4.3 Overhearing

In high density sensor networks the short distances between sensor nodes lead to interferences with non-participant neighbor nodes during data conveyance. This impact on adjacent nodes is called Overhearing. As the transmitted signals idealised move circular from the sending signal source to its surrounding nodes, sensor nodes within reach, which are at the time in active mode, are a common problem. The not involved sensor nodes within reach burn up energy resources owing to receiving and processing useless information. Connectivity requirements have to be weighed up with the arising disadvantiges regarding energy dissipation and latency caused by generously keeping nodes in active mode.

4.4 Overemitting

As Overhearing data meant for other sensor nodes causes energy dissipation, so does Overemitting, meaning information propagated during inactive phases of the data sink or as the case may be the target node, which subsequently has to be resent [4]. The repeated transmissions increase the energy expenditures of the individual nodes for sending data and the latency of the whole sensor network. No customary but thoroughly imaginable defect is the simultaneous occurance of Overhearing and Overemitting. The ideal situation would be a previously known path through the network with

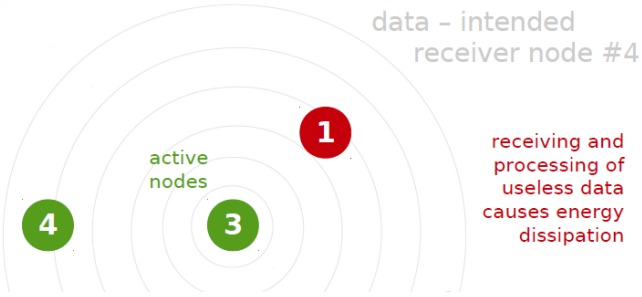


Figure 2: Overhearing

timely activation of the included sensor nodes, which in turn could lead to problems when concurrent transmissions take place and in unison paths are crossing.

4.5 Reduction of protocol overhead

The transmission of e.g. protocol header information and control messages depletes energy resources and since this data in the end is not exploitable, it should be kept a minor share. Techniques for the reduction of the protocol overhead are for instance adaptive transmission periods, cross-layering approaches, where information from the other network layers is used for optimization, and optimized flooding to e.g. enable a less resource-intensive determination of locations [18]. A short transmission period leads to less energy consumption and helps therefore saving resources, at the same time latency to changes is increased [18]. In consequence a favorable value for the transmission period depends on the frequency of change.

4.6 Traffic fluctuation

Traffic peaks caused by the event-based communication in Wireless Sensor Networks can temporarily lead to congestion or high delays [11]. When the network is working on its maximum capacity congestion rises to extremely high levels.

5. MEASURES FOR ENERGY CONSERVATION

5.1 Node activity management

5.1.1 Sleep scheduling

Ready-to-receive mode consumes nearly as much energy of the sensor node's resources as receive mode [17]. Thus a way to set the node to a sleeping mode and determine the right time to wake it again is necessary to effectively save energy in idle time spans. The smaller the network load the greater are the saving potentials, as more nodes remain in active mode when they are not needed to. For sleep scheduling without any previous examination, certain periods are determined in which the individual node is in sleeping mode. The ideal time span for the periodically recurring inactive mode depends on the network traffic.

doi: 10.2313/NET-2012-08-2_04

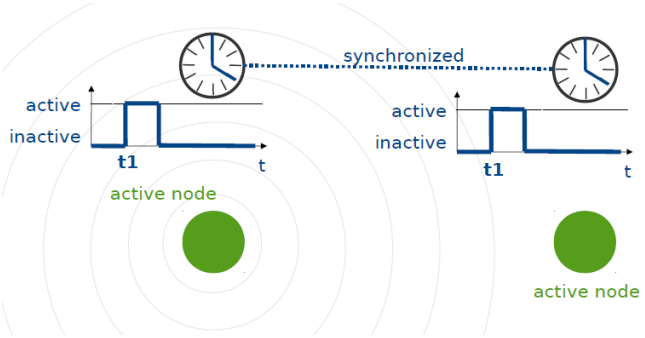


Figure 3: Scheduled rendevouz

5.1.2 *On-demand node activity*

In on-demand activity approaches the sleeping and transmission periods of the sensor nodes are not scheduled, but the nodes are by default permanently in an inactive state with a simple stand-by functionality. If a plain wake-up signal is broadcasted on a reserved channel, the neighboring nodes located within reach switch to active mode. After activation the data transmission can take place. As the start-up signal usually does not have to be decoded, a very energy conserving design for the listening device is feasible, nonetheless during transmission all surrounding nodes are switched on and for the most unnecessarily [4].

5.1.3 *Scheduled rendevouz*

Scheduled time slots for simultaneous active phases of multiple adjacent sensor nodes can lead to a substantial simplification of the communication between the neighboring nodes; however for an effective application this method requires accurately synchronized internal timers [4].

5.2 Data aggregation, data fusion, data preprocessing, data reduction

5.2.1 *Clustering*

Since the energy cost of transmitting data is higher than the effort on data processing it is beneficial to aggregate data within clusters [9]. Appointed sensor nodes which act as cluster heads provide the connection between sensor nodes and the respective base station [3]. Clustering can reduce the amount of data, as the cluster head is in charge of monitoring and processing queries [18] and thereby not as many connections have to be established. At meeting points of data from different sensor nodes redundancies can be recognized and avoided. Data aggregation is, due to the fact that it eases several of the energy dissipating effects, an important feature of energy-efficient Wireless Sensor Networks, but one should way the extend, as buffering could lead to delays [11].

5.2.2 *Adaptive sampling rate*

The overall data can for example be reduced by adapting the sampling rate to the current situation. Circumstances permitting, this can significantly reduce the amount of generated data. Resepective algorithms check for instance density in certain areas (the higher the density the lower the individual frequency), volatility of the sensed data or the traffic level to avoid congestion by lowering the sampling rates.

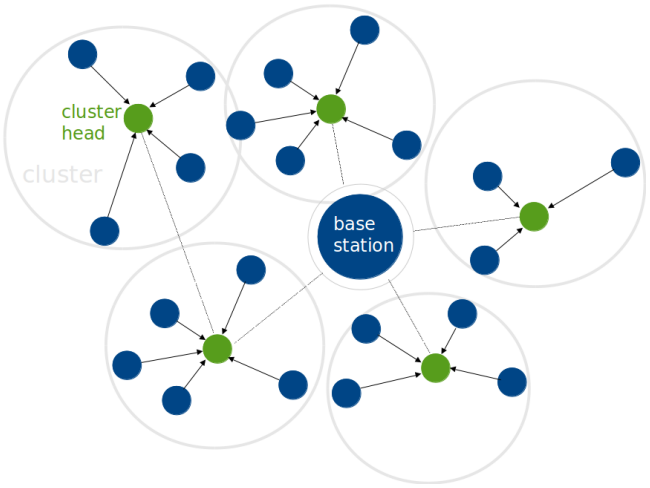


Figure 4: Clustering

5.2.3 *Deviation control*

An alternative approach is to save the latest measured data and only report deviations which exceed a certain threshold. For this type of sensing the sensor nodes need a buffer to save the latest values directly at the node and thereby reduce necessary data transmissions between the nodes. Dependent on the volatility of the sensed data the load of total network communication can significantly be reduced.

5.2.4 *Data compression*

Data compression saves energy by relocating effort from communication to processing. The sensing data is either previously gathered at some point or directly compressed to lower the workload for transmission. This leads to additional computational cost for compressing and decompressing, but at the same time information content is not reduced. Since processing data customarily consumes much less energy than transmitting data in a wireless medium the implementation of data compression algorithms can lead to considerable energy savings [10]. On the other hand the compression process takes time and hence increases latency of the sensor network. According to [10] the following categories of compression algorithms for Wireless Sensor Networks can be distinguished:

- *Coding by ordering*: The permutation of other variables within the sent data can be used to represent a value and therefore the overall sent data can be reduced. A permutation of three distinguishable variables can for example be utilized to save one of six different values.
- *Pipelined in-network compression*: The energy consumption is reduced by combining within a certain time span sensed data. Redundancies are removed and more data is aggregated into one data packet, which

doi: 10.2313/NET-2012-08-2_04

narrows the amount of sent data, but on the other side increases latency.

- *Low-complexity video compression*: Algorithm based on block changing detection and JPEG-compression for wireless video systems and especially designed to suit the limited hardware resources.

- *Distributed compression*: A side-information is used to encode information of two correlated sources and thereby to reduce the transmitted data.

5.3 Load balancing

The actual position of a sensor node within the set of deployed nodes determines which operations have to be conducted and how many established communication paths comprise the particular node. The energy consumption can be balanced among the deployed sensor nodes e.g. by counting performed operations or by consideration of the remaining battery capacities. Due to the additional strain, regular sensor nodes appointed as cluster heads would have a much shorter lifetime as the nodes which are mere in charge of sensing data. If clustering is applied and there is no special more powerful hardware for the cluster heads in use, the usual approach for cluster head selection are stochastic methods which do not consider energy consumption [9]. As energy expenses of the cluster heads for data transmission are dependent on the distance between sensor node and base station, routing protocols like the in [13] proposed modification of LEACH, which consider cluster head energy conservation, have a positive effect on network lifetime, as they can divide the energy consumption into equal shares for all qualified nodes.

5.4 Adaptive transmission range

Since a reduction of the sensor node's broadcasting range lowers energy expenditure, the coverage should be broad enough to ensure proper connectivity, but all at once not unnecessarily extensive. Consideration of the remaining energy in addition to inclusion of the node distances obtains especially in heterogeneous sensor networks good results [18].

5.5 Directional antennas

With directional antennas, as their name implies, all communication efforts are concentrated into a certain direction. This wages on the one hand additional problems, since the nodes for successful transmission not only have to be active but also properly oriented, on the other hand communication can take place over larger distances with less energy consumption and causes immediately less useless data which, on its way off the targeted sink, could lead to e. g. Overhearing and therefore more energy dissipation.

6. COMMUNICATION PROTOCOL STACK

6.1 MAC protocols

Medium Access Control protocols for Wireless Sensor Networks can be categorized into centralized and distributed protocols with for each case the subgroups schedule-based and contention-based protocols, and hybrid protocols which combine the stated approaches [4]. Whereas the MAC protocol manages operations of the transducers and thus of the most energy consuming components of the sensor nodes, it is crucial for the energy-efficient implementation of a Wireless Sensor Network [6].

6.1.1 TDMA-based protocols

TDMA (Time Division Multiple Access)-based approaches schedule the access to the medium by means of time slots. Time is partitioned in periodic frames with a certain number of slots and the channel access is managed on a slot-by-slot basis [5]. Since the delays are controlled and these contention-free protocols reduce or even eliminate collisions, high performance and reliability are consequent characteristics [20],[6]. TDMA is particulary suitable for high-traffic networks. Examples are the TRAMA (Traffic-Adaptive Medium Acccess) protocol, which separates a random-access period with slot reservation and a scheduled-access period whose slots are assigned to individual nodes, and the for periodic monitoring applications optimized FLAMA (Flow-Aware Medium Access) protocol, which is based on TRAMA [5]. TDMA-based protocols allow, provided the respective parameters are adequately appointed, a minimized energy consumption, but this precondition leads to limitations in adaptability to topology changes. These protocols are particularly qualified for high levels of contention and not as popular as the Contention-based type.

6.1.2 Contention-based protocols

Contention-based approaches are often associated with the use of Carrier Sense Multiple Access (CSMA) protocol, which is typically deployed in unsynchronized wireless networks, or a amended version of it [11]. CSMA examines the operating status of the channel before sending data and whenever it is occupied, transmissions are postponed [20]. If the medium is occupied, a random time intervall is bided until the next status check is carried out. Otherwise the data transmission begins. For minor work loads contention-based protocols lead to higher channel utilization and are therefore in this case the means of choice. [19] states CSMA as the predominant class of Medium Access Control protocols. Representatives of these type are for example:

- *B-MAC (Berkeley MAC)*: B-MAC is a very common MAC protocol as it is included in the TinyOS operating system. It offers a optimized carrier sense procedure and sleep interval definition within runtime [12].

- *S-MAC (Sensor-MAC)* [5]: S-MAC is a scheduled rendevouz approach with synchronization via sync packets. In listen periods the synchronization and other control information is exchanged. The nodes establish their own schedule and aggregate in virtual clusters.

Contention-based protocols are robust, scalable, can easily adapt to traffic changes and facilitate low latency [5]. The downside is, that they are less energy efficient than TDMA-based protocols.

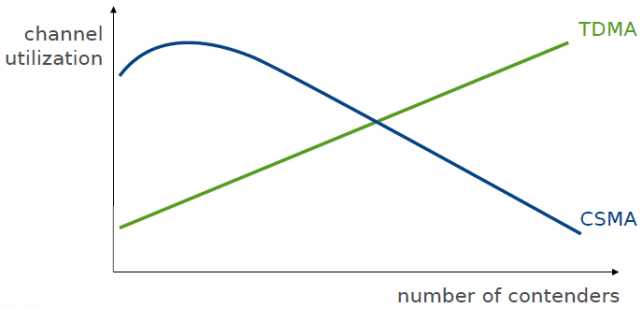


Figure 5: Correlation between number of contenders and channel utilization for TDMA and CSMA [16]

6.1.3 Hybrid protocols

Hybrid Protocols change their behavior dependent on the level of contention, which means they switch from a contention-based behavior at low contention levels to a TDMA approach in case of high levels of contention [5]. A established exponent of this category is Z-MAC (Zebra MAC), which employs CSMA in cases of slight traffic and otherwise avails TDMA [18]. Within a two-hop reach the sensor node, which wants to send data, has to contend for access. If a too many packets are lost, a control message is sent, which tells the neighbouring nodes to switch to high contention mode and after a certain time interval the nodes switch back to normal mode [12]. Even though the combination of the TDMA-based and the Contention-based approach can cancel out or at least alleviate the weaknesses of both protocol categories, it leads to high complexity and is arising thereby not practical for Wireless Sensor Networks with large node counts [5].

6.2 Routing protocols

As consumed energy for transmission rises exponentially with distance [20], the sensing data is forwarded by directly neighboring sensor nodes and the routing protocol has to specify the most suitable stopovers in terms of factors like load balancing and duration of the transfer. Besides classification by network structure (as conducted hereafter) routing protocols can be distinguished by their route establishment [11]. As proactive routing protocols on the one hand try to establish routes before they are actually needed and reactive protocols only react to inquiries, hybrid approaches combine the principles [11].

6.2.1 Data-centric routing

For Wireless Sensor Networks with high numbers of nodes it is often not practical to assign global identifiers to every single node and as a result it is difficult to communicate with a particular node set [1]. As queries within the sensor network would otherwise lead to significant redundancies specific routing protocols to cope with this situation are necessary. In data-centric routing no great store is set by the originator of the transmitted data, so that the data easily can be combined and processed. Special cases of data-centric routing are negotiation-based routing, where meta descriptions of the content are generated for advertising and interested nodes can request the data, and query-based routing, which is based on in-network processing by means of a pilot node aggregating the data and an ancillary query layer enabling inquiries of specific information [8]. Examples for data-centric routing protocols are:

- *Flooding and Gossiping* are antiquated mechanisms for broadcasting within a Wireless Sensor Network. For Flooding each node which receives data passes it on to its neighboring nodes until a maximum number of hops is reached [7]. Gossiping is based on Flooding but the individual node which receives data relays it only to a randomly selected neighbour [7]. Common problems are duplicated messages sent to the same node (implosion) and two nodes sensing the same region sending similar data (overlap) [1].
- *SPIN (Sensor Protocols for Information via Negotiation)*: Is based on negotiation between the nodes by means of data advertisement through meta-data. SPIN is one of the first data-centric routing protocols for Wireless Sensor Networks and it avoids typical flooding problems like overlaps, implosions and resource-blindness [7].
- For *gradient-based routing* the minimal number of intermediate stops on the way to the destination of the data is crucial. This minimum of waypoints is called height of the node [17]. The gap between the sensor node's height and the height of a neighboring node represents the gradient of this particular connection. The largest difference between heights leads the way to the next hop.

6.2.2 Location-based routing

Location-based routing's greatest advantage is the scalability. Due to the availability of location information for the sensor nodes, there is no global knowledge of the network necessary and thus routes can be specified and locations can be found independently from flooding [18]. Since location information is directly available, location-based protocols can help saving time and energy. The route determination is more efficient and apart from that the geographical information at hand can be used to only set the nodes of a specific area, which is of interest, in active mode, while the residual sensor nodes are in sleeping state [8].

- *GEAR (Geographic and Energy Aware Routing)*: The respective neighboring node and next hop to pass the data packet on is selected dependent on remaining energy resources and the distance to the destination [7].
- *MECN (Minimum Energy Communication Network)* employs low power GPS and identifies relay regions for the respective nodes, inclosing the sensor nodes, which would selected as next hop lead to less energy consumption than a direct communication with the data sink [1].

6.2.3 Hierarchical routing

Hierarchical routing protocols either subdivide the sensor nodes into clusters and appoint superior cluster heads which gather data originated by the cluster's nodes, process it and directly communicate with the sink or form other node sets

doi: 10.2313/NET-2012-08-2_04

to aggregate data and narrow the energy expenditures on data transmissions. Hierarchical routing enhances scalability and reduces the comprehensive traffic [18], but concurrently local energy cost for communication is increased due to e.g. the direct connection to the sink and the processing overhead [8]. Examples for hierarchical routing protocols are:

- *LEACH (Low-Energy Adaptive Clustering Hierarchy)* is a very popular hierarchical routing protocols which facilitates random change of cluster-heads to evenly balance the energy consumption and all data processing is carried out within the individual cluster [7].
- *PEGASIS (Power-Efficient GAthering in Sensor Information Systems)* instead of clusters forms node chains, which means that it is based on a multiple-hop strategy, and since there is no dynamic cluster formation necessary and the data aggregation reduces the number of transmissions, PEGASIS generally is more efficient than LEACH [1].
- *TEEN (Threshold sensitive Energy Efficient sensor Network)* is due to its responsiveness especially suitable for time-critical applications and the level of energy consumption is relatively low, as the sensor nodes only transmit values which exceed a certain predefined threshold [1].

6.2.4 Network-flow-based routing / Quality-of-Service-based routing

Network-flow-based routing protocols resemble the already mentioned types but their main focus is to optimize the balancing of traffic and to maximize the network lifetime. Maximum lifetime energy routing for instance defines link costs depending on remaining energy and required transmission energy, which are utilized to even out the energy expenditures of the nodes [1]. Quality-of-Service functions like end-to-end guarantees and hence further examination of e.g. correct transmission are usually an additional feature of routing protocols. An example for a Quality-of-Service approach is the location-based protocol SPEED, which allows the estimation of end-to-end delays by ensuring a certain packet speed [1].

6.3 Transport protocols

Transport protocol functionality is in several cases directly integrated in the routing protocol, though if implemented the transport protocol verifies the arrival of sent data at the sink, regulates the data traffic within the network and splits larger data loads into conveyable fragments [20]. There are three types of transport protocols [21]:

6.3.1 Protocols for congestion control

Congestion is detected for instance via the queue length (Fusion, CODA) or packet service time (CCF, PCCP) and usually either implicitly or explicitly notified [21]. In most cases this notification is followed by a rate adjustment to mitigate the congestion. Exceptions are for example Siphon, which detects mitigation through queue length and application fidelity and reacts by redirecting traffic to virtual sinks, or Trickle, which aims to prevent congestion by utilising a polite-gossip approach and there needs no additional congestion detection [21].

6.3.2 Protocols for reliability

Protocols for reliability usually provide hop-to-hop loss recovery and therewith packet reliability. For loss detection and notification mostly NACK or IACK are used and popular types are for example RMSC, RBC and GARUDA [21]. An exeption to the rule is ESRT, which provides event reliability by means of end-to-end source rate adjustment [21]. Since the connectivity within the Wireless Sensor Network is not predictable and the error rates for transmission are generally high, hop-to-hop loss recovery seems like the more adequate choice for WSNs. On the other hand examinations after every hop increases the latency. Reliability protocols can be distinguished by their reliability direction: Upstream (sensor to sink), downstream (sink to sensor) and bidirectional reliability [14].

6.3.3 Protocols for congestion control and reliability

Transport protocols like STCP combine the two previous types and provide congestion control as well as reliability [21]. Other examples are RCRT, CRRT and TRCCIT [14].

7. CONCLUSION

With decreasing costs and advancing network lifetime of Wireless Sensor Networks the number of potential application areas is on the rise. To achieve both, energy conserving communication techniques become increasingly important and are in fact subject to many present research projects. The three prevalent boundary conditions of sensor nodes are communication consumes more energy than computation, the consumption is not just in transmission states but as well in idle state relatively high and depending on the application the sensing unit can cause a considerable amount of energy expenditure. As TDMA-based MAC protocols are very energy efficient if correctly configured, Contention-based approaches on the other hand are very robust and flexible by means of traffic changes and scalability. Due to this a integration within Hybrid Protocols and thereby a combination of their strengths is desirable but owing to complexity not suitable for Wireless Sensor Networks with large node quantities. Evaluations show, that the best results are obtained with combinations of the termed energy saving measures. However, there is no general recipe, which fits for every situation. The most suitable solution for a certain application depends on various factors like environment, density of nodes, hardware, usage of data, frequency of changes.

8. REFERENCES

[1] K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Ad hoc networks*, 3(3):325–349, 2005.

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

[3] J. Al-Karaki and A. Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE*, 11(6):6 – 28, dec. 2004.

[4] M. A. Ameen, S. M. R. Islam, and K. Kwak. Energy saving mechanisms for mac protocols in wireless sensor networks. *International Journal of Distributed Sensor Networks*, 2010, 2010.

[5] G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7(3):537 – 568, 2009.

[6] C. Cano, B. Bellalta, A. Sfairopoulou, and M. Oliver. Low energy operation in wsns: A survey of preamble sampling mac protocols. *Computer Networks*, 55(15):3351 – 3363, 2011.

[7] S. Dai, X. Jing, and L. Li. Research and analysis on routing protocols for wireless sensor networks. In *Communications, Circuits and Systems, 2005. Proceedings. 2005 International Conference on*, volume 1, pages 407–411. IEEE, 2005.

[8] S. Ehsan and B. Hamdaoui. A survey on energy-efficient routing techniques with qos assurances for wireless multimedia sensor networks. *Communications Surveys Tutorials, IEEE*, 14(2):265 –278, quarter 2012.

[9] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *Wireless Communications, IEEE Transactions on*, 1(4):660 – 670, oct 2002.

[10] N. Kimura and S. Latifi. A survey on data compression in wireless sensor networks. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, volume 2, pages 8–13. Ieee, 2005.

[11] A. Klein. *Performance Issues of MAC and Routing Protocols in Wireless Sensor Networks*. PhD thesis, University of Würzburg, Dec. 2010.

[12] K. Langendoen. Medium access control in wireless sensor networks. *Medium access control in wireless networks*, 2:535–560, 2008.

[13] N. V. Lonbale, S. Gupta, and R. S. Bhute. Energy conservation of wsn using routing protocol. *IJCA Proceedings on National Conference on Innovative Paradigms in Engineering and Technology (NCIPET 2012)*, ncipet(5):18–21, March 2012. Published by Foundation of Computer Science, New York, USA.

[14] A. Rathnayaka, V. Potdar, A. Sharif, S. Sarencheh, and S. Kuruppu. Wireless sensor networks: Challenges ahead. In *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*, pages 824–829. IEEE, 2010.

[15] Z. Ren, S. Shi, Q. Wang, and Y. Yao. A node sleeping algorithm for wsns based on the minimum hop routing protocol. In *Computer and Management (CAMAN), 2011 International Conference on*, pages 1 –4, may 2011.

[16] I. Rhee, A. Warrier, M. Aia, J. Min, and M. Sichitiu. Z-mac: a hybrid mac for wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 16(3):511–524, 2008.

[17] J. Sinha and S. Barman. Energy efficient routing mechanism in wireless sensor network. In *Recent Advances in Information Technology (RAIT), 2012 1st International Conference on*, pages 300 –305, march 2012.

[18] R. Soua and P. Minet. A survey on energy efficient techniques in wireless sensor networks. In *Wireless and Mobile Networking Conference (WMNC), 2011 4th Joint IFIP*, pages 1 –9, oct. 2011.

[19] J. Stankovic and T. He. Energy management in sensor networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1958):52–67, 2012.

[20] J. Suhonen, M. Kohvakka, V. Kaseva, T. D. Hämäläinen, and M. Hännikäinen. *Low-Power Wireless Sensor Networks - Protocols, Services and Applications*. Springer, Berlin, Heidelberg, 2012. aufl. edition, 2012.

[21] C. Wang, K. Sohraby, B. Li, M. Daneshmand, and Y. Hu. A survey of transport protocols for wireless sensor networks. *Network, IEEE*, 20(3):34–40, 2006.

Packet Reordering on Gateway as a Representative of Data Processing Techniques

Martin Johannes Waltl
Supervisor: Corinna Schmitt
Seminar Sensor Nodes: Operation Modes, Networks and Applications SS2012
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitekturen
Fakultät für Informatik, Technische Universität München
Email: martin.waltl@mytum.de

ABSTRACT

Wireless Sensor Networks (WSN) are becoming an important tool for data acquisition in environmental monitoring. Their advantages are cheap deployment, long system life time and little maintenance during operation. In many applications the WSN use a multi-hop communication protocol to transmit their data to the sink node. The arriving data is of poor quality suffering from duplicates, packet loss, device reboots and unsynchronized time stamps caused by local clock drifts. Some of these problems have not been solved by appropriate system designs yet. Therefore, it is necessary to correct these communication artefacts and enhance data quality in a post sequent step. Several algorithms to cope with these challenges and aim higher data accuracy already exist. In this work a model-based approach for the reconstruction of the temporal packet order in a multi-hop WSN is discussed. Furthermore, the challenges for WSN in harsh environments are presented by the PermaSense project, which observes permafrost changes in the alpine region. The multi-hop communication protocol Dozer is described in more detail to highlight the reasons for packet loss and duplicate generation.

Keywords

Environmental monitoring, WSN, Dozer, Data Processing

1. INTRODUCTION

In many technical applications and scientific fields the observation of different phenomena is a necessity for the understanding of processes and their interactions. The analysis of these data delivers new knowledge about the phenomena or is the basis to control technical systems. Wireless Sensor Networks (WSN) are one option to support the collection of these information. Advantages of WSN are their unattended operation over a long period of time, flexible adoption of the sensor nodes, cheap deployment and operation without human interaction.

WSNs consist of sensor nodes, which are equipped with a short-range radio. The sensor nodes acquire the measurements using their sensor equipment and transmit the data to a specified sink. The communication can ether be directly from a sensor to the sink node, or indirectly using multi-hop communication over several nodes. At the sink node all data is collected and stored for further data processing [2]. Some examples of sensor networks are the monitoring of heritage buildings [3], data center monitoring [10] and environmental monitoring of permafrost changes in the Swiss Alps [1].

Environmental monitoring is the continuous observation of natural phenomena, mostly studied over a long period of time ranging from days up to several years. It can support researchers and scientists with reliable information to verify existing models or gather data for future predictions. The collected data serves as basis for statistical analysis and optimization of existing models. Schmitt and Osenberg state that the major aim of environmental monitoring is to support decision making for economic, political and social authorities [13]. The high data quality of these scientific measurements is essential for more quantitative and qualitative conclusions. Furthermore, the analysis of environmental data allows to identify the impacts of natural disasters and human interventions to the environment. A more detailed introduction on environmental monitoring and state-of-the-art developments can be found in the journal 'Environmental Monitoring and Assessment' (Springer). [1]

WSN are one opportunity to support environmental monitoring. An appropriate WSN deployment accomplishes the acquisition of reliable data over its operational period, hence allowing the sensing of long-term evolutions. The field of applications for environmental monitoring is extremely broad and covers many areas of research. This work deals with the permafrost phenomena in the Alps, which is one example of environmental monitoring by a WSN. The PermaSense project is a project that investigates permafrost by the deployment of a WSN in the Swiss Alps [1]. Its architecture, deployment, data acquisition process and challenges in the harsh alpine environment are discussed in the following. Besides that, the problems within a multi-hop communication WSN like packet loss, duplicate generation and disordered arrival of data packets at the sink node are described. To provide valuable measurements for appropriate analysis it is necessary to account for these communication artefacts. One solution is a model-based approach presented by Keller et. al in [7] to reconstruct the temporal order of packets.

The outline of this seminar paper is as follows: The next section contains some background information on multi-hop communication and introduces the Dozer protocol. Section 3 gives an overview of the PermaSense project and Section 4 presents a model-based approach to enhance data quality by correcting for the artefacts of multi-hop communication. The last two sections present related work on data processing techniques and a conclusion.

[1] http://www.springerlink.com/content/0167-6369/open/

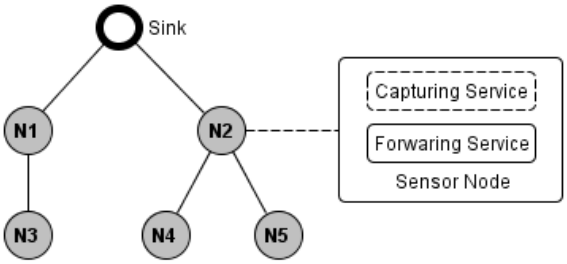


Figure 1: Example of a tree architecture for a WSN. The sink is the root of the communication tree and receives data from each node. Sensor nodes act as forwarding node for their children and can optionally perform data capturing [2].

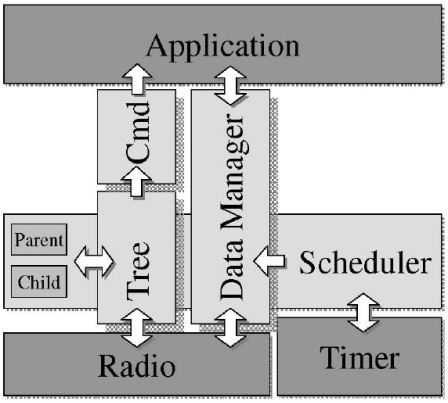


Figure 2: Architecture of the Dozer protocol. The four main components are represented by the light gray boxes. The arrows show the command flow between the different components [2].

2. BACKGROUND INFORMATION

Sensor networks for environmental monitoring are often distributed over a large area. One possible WSN architecture is a tree as shown in Figure 1. The network consists of sensor nodes that collect data and transmit it to one sink node, which is the root of the tree. Wireless sensor nodes are equipped with a short-ranged radio to send their data over the network. In many applications only a few nodes have a direct connection to the sink node, due to the large network expansion and physical obstacles that prohibit a direct communication. The other nodes are indirectly connected over other nodes, which forward the data to the sink. The latter is described as multi-hop communication. Consequently, the sensor nodes provide a sensing and forwarding functionality.

2.1 Dozer communication protocol

Dozer is a data gathering protocol developed for WSN with a multi-hop scenario [2]. It is designed to fulfil the demands of environmental monitoring applications which includes self-stabilizing properties, enable reliable data transfer and is optimized to maximize the system life time. Particularly it provides the process of periodic data collection, can account for topology changes and is optimized for ultra-low power consumption of the sensor nodes. The topic of minimizing power consumption is essential for WSN, because in many scenarios the nodes are battery powered and must endure several years without maintenance.

The Dozer system constructs a data gathering tree, with one sink node as its root. All other sensor nodes perform data acquisition and forwarding of network packets towards the sink. If a network node fails or is temporally unavailable, Dozer handles local recording of the data and rebuilds the gathering tree. When the node reconnects to the network, it is integrated again and the buffered data is transmitted to the sink node [2]. The Dozer protocol is described in more detail, because it is used by the PermaDAQ architecture in [1] and fits the assumptions in [7] to perform the introduced post processing technique.

2.2 Dozer system architecture

In the Dozer system architecture each sensor node is a child and parent node simultaneously. As parent node, it receives packets from its children and forwards them to its parent, until the packets arrive at the sink node. As child, the node sends data packets to its parent. Each node has a packet queue that is shared by its own data packets and the packets from its children. The sending process is designed after the Time Division Multiple Access (TDMA) mechanism, where each child is assigned to a time slot to transmit its data to the parent. In this design, the nodes only control the communication to their adjacent nodes and act locally in the network. The reason for this decision is that a global TDMA mechanism for the whole network would be too expensive concerning energy consumption and system life time. As a consequence, each node handles two schedules: *a*) a schedule for the transmission to the parent node *b*) a schedule to administer its children and assign their time slots.
Figure 2 shows the architecture of the Dozer system with the four main components represented by the light gray boxes. The *Radio* and *Timer* component are hardware elements of the sensor node. All data packets are transmitted using the *Radio* component. The *Timer* module is an essential part, because it delivers the interrupts for the precise execution of the parent and child schedules. The *Application* component represents higher layer applications of the sensor node. Dozer operates using a periodic TDMA frame transmission between all adjacent nodes. This frame starts with a beacon, that is used as synchronization for the children and allows them to compute their transmission time slots. The four main components are explained in more detail.

The **Tree Maintenance** component is responsible for the building of the gathering tree (see Figure 1). When a sensor node is powered up, the first action is to integrate itself into the network. In case of a connection loss (i.e. node is covered with snow) it iteratively tries to reconnect to the network. The reconnection is power consuming and according to strict power requirements this is only done spontaneously. Consequently, the reintegration of a sensor node can take a significant amount of time. In this state, the sensor nodes performs local storage of the acquired data to counteract data loss.

The management of the two time schedules *parent* and *child* is handled by the **Scheduler** component. As parent it accepts requests from child nodes and assigns them to a time slot. Each TDMA frame has a limited amount of time slots,

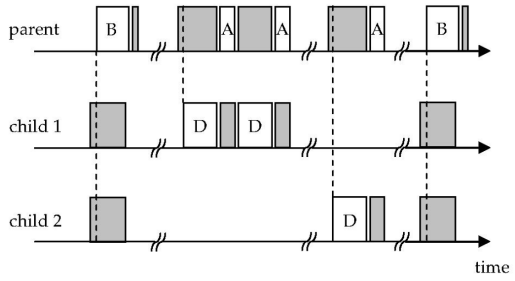


Figure 3: Example for message reception at a parent with two children. The upload slots for the current frame are determined with the beacon (B) as reference. All data messages (D) are acknowledged (A) to ensure data integrity [2].

which is equal to the maximum number of children a node can serve. On the other side as child, the node listens for the parents' beacon and computes the time for its assigned time slot when to send its data. The beacon is also used to synchronize to the parent. Due to the periodic event of the TDMA frame, the children can compute the transmission time slot and stay in idle mode at very long time to reduce power consumption and extend system life time.

Another goal of Dozer is to ensure data integrity. Therefore, each data packet needs to be acknowledged by the parent node. The component **Data Manager** controls and supervises the transmission process of the packets. Figure 3 shows an example communication process for one parent and two children. The parent node starts the TDMA frame with the transmission of a beacon, which signals the start of the frame. The beacon is time stamped at each child node and used as synchronization to compensate for local clock drift. Based on this time stamp each child computes its transmission time slot, which was negotiated at connection establishment. When the scheduler signals the beginning of the upload slot, the Data Manager tries to transmit queued messages. The packets are acknowledged by the parent to ensure data integrity and account for transmission failures. In case of a transmission failure, the node stops sending and makes a new sending attempt in the next time slot. Although, the node could retry in the same slot, this behaviour tries to overcome temporal interruptions of the radio channel. Nonetheless, data loss can occur, but these situations are explained in more detail in Section 4.

The main data flow is specified from the sensor nodes to the sink. In many applications it is desirable to convey control information from one to several nodes. Therefore, the **Command Manager (Cmd)** component provides a backward channel via the beacon messages. Control information are transmitted within the beacon signals downwards the gathering tree. Every node forwards the control information of a parent beacon in its own beacon, thus the information reaches every node in the network. This design allows application to specify custom messages in the Dozer protocol system.

3. PERMASENSE

The PermaSense project has the goal to gather real-time environmental data at high quality on high-mountain permafrost in the Swiss Alps. Scientists want to investigate the permafrost situation in the alpine region and its response to the climate change. A more detailed introduction to this topic can be found in [5]. Besides that, the PermaSense project acts as a prototype for the deployment of WSNs in harsh environments. The alpine region is a favoured terrain as a benchmark for system robustness due to its difficult environmental conditions. PermaDAQ is the WSN system architecture within the PermaSense project to monitor permafrost changes at the Matterhorn (Switzerland, 3450m) [1]. Prior to this project there has been one deployment at the Jungfraujoch (Switzerland 3500m) in 2006/07 [15], which served as basis for the improvement of the PermaDAQ architecture. The aim of the PermaDAQ architecture was to develop a robust sensor network, that delivers high quality data in the mountain terrain for a three year period with unattended operation.

3.1 System requirements

The alpine region and the requirement of several years continuous operation demanded a new system design for the necessary WSN. Major challenges are *a*) the aimed unattended operation time of several years without physical repair, *b*) wide temperature range from -40°C to +60°C, *c*) survival in the harsh high-alpine environment (rock falls, avalanches, snow, ice, rime, lightning, storm) and *d*) the desired data yield of 99%. The sensor nodes are exposed to the harsh environment and can loose their connection to the gathering network, due to coverage with snow or other circumstances. In this situation, the sensor nodes can't deliver data to the sink node. Therefore, another requirement is to provide an autonomous storage capability of at least 6 months to compensate for sensor unavailability. The experiences from the prior project at the Jungfraujoch revealed three main design challenges, which are coped in the PermaDAQ system architecture [1]:

Precision Sensing – For precise measurements it is important to reduce the influences during the sensing and support the creation of exact time stamps. Accurate timing is important, because it delivers higher quality for data analysis and enables data recovery if the sensor needed to store the data locally. One finding from the first deployment was that the simultaneous operation of sensor equipment and radio transmission resulted in corrupted data. The new design needed to be more robust and must consider these effect to allow precision sensing.

Reliability in Harsh Environment – The severe conditions of the mountain region require a robust mechanical design and well durability of the components to ensure high data quality and permit long life time.

Energy Constraints – The long life time requirement can only be achieved by an efficient energy management of the sensor unit. Beutel et. al use the Dozer protocol to minimize energy consumption and a Li-$SOCL_2$ battery. This type of battery is optimized for slow discharge at low temperatures.

3.2 WSN architecture

Figure 4 displays the tiered architecture of the PermaSense project with sensors, wireless sensor network, base station and backend. All components are designed to operate independently to prevent system failure in case of a malfunction of one tier.

doi: 10.2313/NET-2012-08-2_05

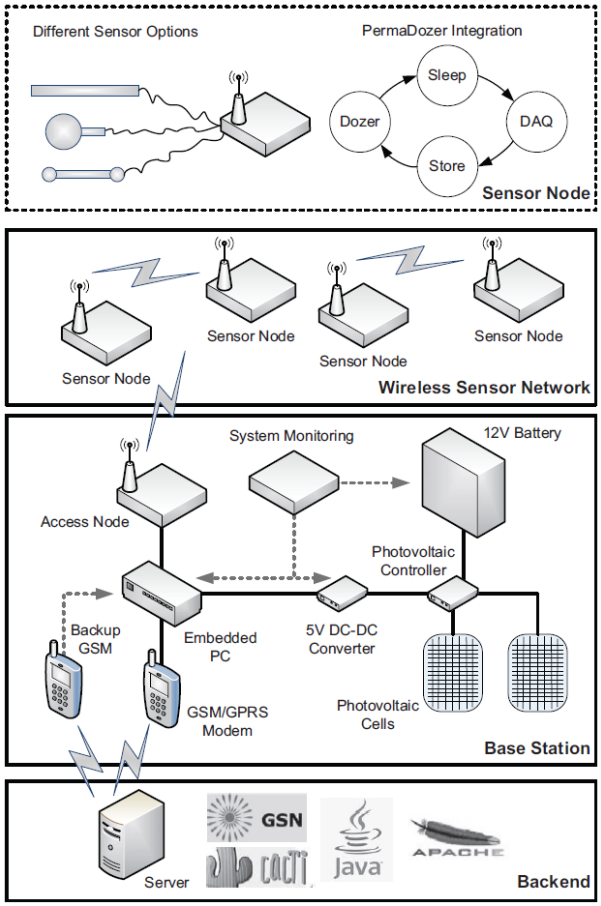


Figure 4: PermaSense architecture [1].

WSN – The PermaDAQ WSN is build of sensor nodes that have been developed for the measurements of permafrost changes. Each sensor node consists of a Shockfisch TinyNode[2], a self-developed sensor interface board (SIB) which runes the Dozer protocol and includes an SD card with 1GB external storage. The SIB is connected to the different sensor units, that perform the measurements for the permafrost observation. In the current implementation there are six different sensors [1]: *1*) a sensor rod for profiling of temperature and electrical conductivity in solid rock, *2*) crack meters consisting of a linear potentiometer for measuring movements, *3*) thermistor chains for profiling of temperatures in rocks, *4*) digital water pressure sensors to assess water flow in cracks, *5*) analog earth pressure cells for assessing ice stress inside lager crack and *6*) self potential sensors using analog differential conductivity measurements with electrodes mounted on the rock surface. The sensor node is powered by an Li-$SOCL_2$ battery. In the current deployment, 25 nodes are installed with a node spacing between 10m to 150m.

Base Station – The base station is an embedded PC platform, that is the sink of the WSN. It collects the system data and transmits it to the backend via an GPRS/EDGE module. The base station is powered by a solar cell system, because it requires more energy and can be constructed in a protective area compared to the sensor nodes.

[2]`http://www.tinynode.com`

Backend – The backend consists of a server which stores the retrieved data in a data base, running a global sensor network (GSN) application. The GSN is a flexible network management software for WSN data and provides a graphical overview.

3.3 Operation and data acquisition

The sensor nodes run the Dozer system, which drives the data acquisition and transmission processes. For the PermaSense project Keller et. al developed an own implementation named PermaDozer that is optimized for their application. Figure 4 illustrates the periodic operation of the PermaDozer integration within the sensor node at the top. Dozer is based on a periodic duty cycle separated into data acquisition, storing, transmission and returning the system into sleep mode to minimize power consumption. The process in the PermaDozer is as follows: The node is in *sleep* mode until the timer initializes the data acquisition process (*DAQ*). When the data is obtained it is *stored* and the system returns to the normal dozer operation, where the data transmission is handled (*Dozer*). Afterwards the system switches back into *sleep* mode.

In the current implementation of the PermaSense project the sensing is performed every 2 minutes and the Dozer beacon is sent every 30 seconds. This activity schedule is optimized for little de-synchronization and energy constraints. Consequently, the beacon rate limits the sensing time to less than 30 seconds within two consecutive beacons. The DAQ is triggered every two minutes and must complete within 30 seconds, before the next beacon of the Dozer protocol arrives. Within the DAQ each sensor measures sequentially, because the simultaneous operation of two sensors would interfere and result in a lower data accuracy [1].

When the data acquisition is finished, the data is appended to the message queue and uploaded along the gathering tree by the Dozer protocol. As mentioned in the beginning, it is possible that nodes become disconnected from the WSN. In this case, the acquired data is locally stored on a SD card and uploaded when the connection is re-established. A deeper insight into the PermaDozer operation exceeds the scope of this work, but can be found in [8].

4. RESTORING TEMPORAL ORDER

The data quality of a WSN suffers from local clock drift, packet duplicates, node reboots and packet loss. Another problem of multi-hop routing protocols (like Dozer) is that the packets don't arrive in correct temporal order, because the packets can be routed along different paths or are locally buffered and delayed in time. Therefore, it is necessary to correct the received data by detecting duplicates and reconstruct the temporal order. This post processing step can be performed on powerful systems, which reduces the complexity of sensor nodes and their energy consumption. Therefore, a formal model describing the data acquisition and transmission process is required. Keller et. al developed in [7] a model-based approach to reconstruct the temporal packet order for a WSN with a tree architecture.

Another reason for data inaccuracy is that the data acquisition equipment suffers from noise, outliers and inaccuracy due to faulty calibration. This effects can be minimized by an appropriate design of the sensor unit, sensing process and detailed analysis on the sensor behaviour in the targeted region. In this work, the reconstruction algorithm doesn't rely

doi: 10.2313/NET-2012-08-2_05

on the sensor data, why the topic of calibration is not discussed further. The next subsection introduces the underlying system model, necessary for the model-based approach in Subsection 4.2. Section 4.3 presents a case study that evaluates the performance of the model-based approach on data of the PermaSense project.

4.1 System model

The specification of a formal model is a crucial part, because the temporal order is reconstructed on these assumptions. In general a WSN consists of sensor nodes that periodically send packets via a tree structure to the sink node (see Section 2). A packet is a tupel of five elements (o, s, p, $\tilde{t_s}$, t_b), where o is the node address, s the sequence number (bounded by s_{max}), p the payload, $\tilde{t_s}$ the estimated sojourn time and t_b the absolute arrival time at the sink. The sojourn time is the packet residence time within the network until it reaches the sink node. As a system assumption, the sink node has an absolute clock that does not suffer from any clock drift and restarts. On the contrary, the sensor nodes are committed to restarts and have a free running clock that suffer from local clock drift.

Nodes are exposed to two different kinds of restarts: *a*) a warm restart is caused by a malfunction of the operation system, initiated by the watchdog timer and resets the system clock and *b*) a cold restart is equal to a hardware restart, where the system clock and sequence number are reset and the message queue is emptied (data loss). After a restart the sensor node initiates data capturing immediately, hence the time difference between two consecutive packets can be lower than T. Cold restarts are one reason for packet loss. There is no global synchronization within the network. Every node provides two different services [7]:

Packet Capturing Model – The sensor nodes perform data capturing at a constant interval T, that is relative to the local clock. In each interval a new packet is generated with the data as payload and a sequence number, which is incremented afterwards. One issue of a sensor node is that it suffers from a local clock drift due to physical properties of the oscillator caused by i.e. temperature variations or fluctuating supply voltage.

Forwarding Model – In a multi-hop network, packets are conveyed along the parent nodes to the sink. Each packet is acknowledged by the parent node to prevent data loss. The system model assumes that the packets are stored in a message queue and forwarded after a sojourn time, which equals the residence time of the packet within the sensor node. When a packet i is send from a node e, the sojourn time of the packet is updated according to Equation 1.

$$\tilde{t_s}(i) = \tilde{t_s}(i) + t_{residence}(i, e) \tag{1}$$

The sojourn time received at the sink, represents the sum of the sojourn times of the passed nodes. It is only an estimation of the real sojourn time, because the time measurements are taken correspondingly to the nodes' local clocks. The actual transmission time of a packet between nodes is neglected, because it is significantly smaller than the residence time. At the sink, each packet is time stamped at receipt with t_b to estimate its generation time. Duplicates can occur if the acknowledgement for a packet does not arrive in time and the child node retransmits the packet.

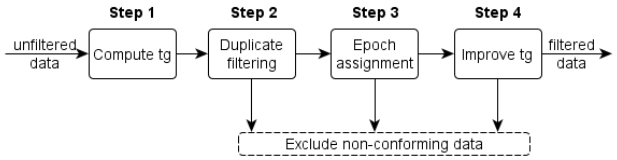


Figure 5: Steps of the model-based approach [7]. (t_g - generation time)

4.2 Model-based approach

The model-based approach to reconstruct the temporal order is based upon the previous system model and subdivided into four steps (see Figure 5). The result is a list of temporal ordered packets, excluding non-conforming packets [7].

Step 1: Compute packet generation time interval – The packet generation time is computed by subtracting the estimated sojourn time of a packet i from the packet arrival time at the sink node:

$$\tilde{t_g}(i) = t_b(i) - \tilde{t_s}(i) \tag{2}$$

This is only an approximation of the generation time, because the sensor nodes suffer from clock drifts and have a time measurement uncertainty of $(-\hat{t_u},0]$. The local clock drift is within the range $[-\hat{p},\hat{p}]$ and $h * \hat{t_u}$ accounts for the worst-case error of timing uncertainties over h hops. Accounting for these effects we get an estimation interval for the generation time:

$$t_g(i) \in [t_g^l(i), t_g^u(i)] \tag{3}$$

where

$$t_g^u(i) = t_b(i) + \frac{\tilde{t_s}(i)}{1+\hat{p}} \tag{4}$$

$$t_g^l(i) = t_b(i) - \frac{\tilde{t_s}(i) + \hat{h} * \hat{t_u}}{1-\hat{p}} \tag{5}$$

Step 2: Duplicate filtering – The next step is to get rid of duplicates within the full data set D. Duplicate packets are determined by the same source address (o), same sequence number (s), equal payload (p) and overlapping generation times intervals. Duplicate packets can have different $\tilde{t_s}$, hence packets can take different paths to the sink. The problem is transformed to find the maximum independent set in a graph. A graph $G = (V, E)$ consists of vertices V that are marked as duplicates and their interconnections E. Two vertices v and w are connected by an edge $(v, w) \in E$, if they have overlapping generation time intervals:

$$(v, w) \in E \Leftrightarrow (t_g^u(v) \geq t_g^l(w)) \wedge (t_g^u(w) \geq t_g^l(v)) \tag{6}$$

A standard algorithm is used to find the maximum independent subset $I \subseteq V$. Therefore, the full data set D (unfiltered data) is separated into a subset of duplicate packets and the corresponding graph is constructed. In this graph the maximum independent set I is computed. All packets that correspond to a vertex $z \in V \setminus I$ are marked as duplicates and removed. The resulting duplicate free data set is used for further processing. This simple graph based selection algorithm can also reject non duplicate packets incidentally.

doi: 10.2313/NET-2012-08-2_05

But the superior goal to obtain a duplicate free data set is weighted more important than incidentally removed packets.

Step 3: Epoch assignment – The sequence number s is bounded by the maximum value s_{max} with $0 \leq s < s_{max}$. An epoch contains maximum s_{max} subsequent packets. The reasons for epochs are an overrun of the sequence counter ($(s+1) \bmod s_{max}$) and cold restarts, which reset the sequence counter to zero. Epochs are labelled with incremental index $e \in N$ and each packet i is assigned to exactly one epoch $e(i)$. The following statement determines the epoch assignments of packets with different sequence numbers. If the packets have similar sequence numbers, they belong to different epochs, because the data set is duplicate free.

$$t_g(k) < t_g(l) \Leftrightarrow (e(l) < e(k)) \vee ((e(k) \equiv e(l)) \wedge (s(k) < s(l))) \quad (7)$$

Packets from the same epoch share the same epoch centre T_c. The epoch centre is computed by subtracting the sensing interval T multiplied by the sequence number of the packet i from its generation time:

$$T_c(i) = \tilde{t_g}(i) - s(i) * T \quad (8)$$

$T_c(i)$ is the estimated epoch centre, because $\tilde{t_g}$ is only an estimation. Epoch centres for packets k and l of the same epoch are therefore not equal, but close to the real epoch centre T_c. ΔT is the allowed window of the accepted epoch centre for packets within the same epoch.

$$e(k) = e(l) \Rightarrow |T_c(k) - T_c(l)| \leq \Delta T \quad (9)$$

If nodes are not exposed to restarts, two epoch centres are always $s_{max} * T$ apart. On account of the local clock drift and node restarts, the difference between two epoch centres is $< s_{max} * T$. Therefore, Keller et. al specify the epoch assignment as follows [7, 5.3]:

> *All packets whose "epoch centers" are close enough are assigned to the same epoch, whereas packets whose "epoch centers" have a large distance are assigned to different epochs.*

This definition is determined by the equations (9) and

$$|T_c(l) - T_c(k)| > \Delta T_c \quad (10)$$

where the virtual epoch centres of the two packets k and l must be at least ΔT apart. All packets that violate (9) or (10) are marked as non-conforming, which is illustrated by Figure 6. The choice of the parameter ΔT is crucial, because it determines the tolerance of the epoch assignment.

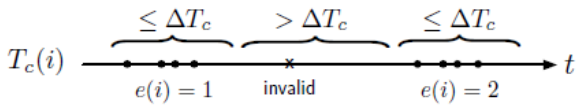


Figure 6: Two epoch centres must be at least ΔT apart [7].

Another problem is caused by two cold restarts happening shortly one after another. In this case, packets from two disjoint epochs can be within the same epoch centre interval. Therefore, the minimum distance between two epoch centres must be at least $2 * \Delta T$ apart, to guarantee a sufficient separation of epochs. Based on these definitions the algorithm for the epoch assignment comprises the following steps:

1. Packets exceeding a maximum $t_{s,max}$ are removed.
2. Compute the virtual epoch centres for all packets (8).
3. Remove packets that violate the definitions (9) or (10).
4. Assign a unique id to the remaining packets, reflecting the temporal order of the packets by equation (11). Afterwards remove all packets that have the same id, which is caused by two consecutive restarts that are separated less than $2 * \Delta T$.

$$id(i) = e(i) * s_{max} + s(i) \quad (11)$$

The result of this algorithm is that each packet has a unique id and the filtered data set satisfies:

$$t_g(k) < t_g(l) \Leftrightarrow id(k) < id(l) \quad (12)$$

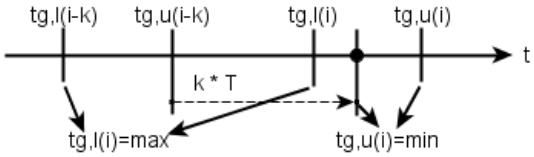


Figure 7: Illustration of forward reasoning [7].

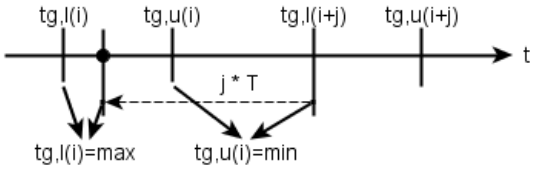


Figure 8: Illustration of backward reasoning [7].

Step 4: Forward/Backward reasoning – The generation time is limited by an upper and lower bound after step 1. The additional sequence information obtained in step 3 and the generation time interval T can be exploited to tighten this interval. Therefore, Keller et. al use forward and backward reasoning, where k is the number of generation time intervals between two consecutive packets in the ordered data set. The forward and backward reasoning are illustrated in the Figures 7 and 8. It is essential that the data set is ordered accordingly $id(i) < id(i+1)$. For each packet i the tightening mechanism comprises the following steps. First the backward and forwarding reasoning is executed, which return a lower and upper bound for $t_g(i)$. Next the upper and lower bounds are combined accordingly:

$$t_g^l(i) = max(t_g^l(i-k), t_g^l(i), t_g^l(i)_{backward}) \quad (13)$$

$$t_g^u(i) = min(t_g^u(i)_{forward}, t_g^u(i), t_g^u(i+j)) \quad (14)$$

If the packet violates $t_g^u(i) < t_g^l(i)$ it is non-conforming and removed from the list. This process is executed for each packet. The result of this step is that the packets feature tightened generation time intervals within the ordered set [7].

doi: 10.2313/NET-2012-08-2_05

4.3 Evaluation

Keller et. al compare their model-based approach to an simple heuristic to evaluate its performance [7]. As data basis they use data sets from the PermaSense project that was gathered by the WSN described in Section 3. The data sets comprise measurements from July 2008 until May 2010, which represent three deployment phases. The three phases have different system behaviour: (Phase A) highly non-conforming system behaviour, (Phase B) sensor nodes subject to a high frequency of unplanned warm restarts and (Phase C) high accumulation of transmission delays of several hours to days by one third of the sensor nodes. They used three metrics to evaluate their model-based approach in comparison to the simple heuristic: *1*) Packet acceptance rate, *2*) correctness of retrieved packet sequence and *3*) improvement of generation time intervals.

In their first analysis they applied the model-based approach and the simple heuristics to retrieve the temporal order by each one. Afterwards they compared the found sequence violations for both approaches and against ground truth data, which was obtained from the SD cards of the sensor nodes. Both the model-based approach and the simple heuristics deliver very good results with little sequence violations and a high packet acceptance rate, for the last two phases B and C that represent normal system behaviour. In the evaluation of phase A the model-based approach resulted in a much better reconstruction of the temporal order and a higher packet acceptance rate than the simple heuristics.

The second analysis focused on the improvement of the generation time intervals. Since the system behaviour in phase A is not according to the assumed system model, only phase B and C were used for this evaluation. The model-based approach was again applied on the two last phases and the improvement of the generation time was evaluated. The results are that forward and backward reasoning lead to a reduction of the interval in 90% of the packets. In 75% of the cases, the generation time interval could be reduced by at least half, compared to step 1. On the absolute scale, the initial generation time was decreased by up to 100s and the mean was reduced from 8.1s to 2.8s. Concluding, the authors state that forward and backward reasoning deliver a considerable improvement in general and compensate for introduced worst-case uncertainties based on the clock drift and the missing global synchronization.

5. RELATED WORK

WSN are distributed event-based systems that differ from conventional communication systems by the requirement of energy constraints, a favoured communication flow from many-to-one (sensor node to sink) and low bandwidth demand. The whole data communication and data aggregation process is designed to optimize energy consumption and ensure a long sensor node life time without maintenance [9].
Sensor networks suffer from several drawbacks like clock drift, temporary drop outs of sensor nodes, restarts and a dynamic network topology [1]. One resulting problem is that the temporal order of the arriving packets does not agree with the logical order of reception at the sink node. Therefore, counter measures are necessary to achieve the correct temporal packet order. There are two basic approaches to cope this problem [7].
The first approach is to establish a global time base over the whole network. If all network nodes are synchronized the sensor nodes can time stamp each measurement correctly. This time stamp allows a simple reordering at the data sink, although if packets arrive unsorted. Nonetheless, this approach requires a synchronization protocol for the WSN. Keller et. al state in [7] that introducing a global synchronization scheme is cost intensive, because it increases the node complexity and reduces the battery life time. Another problem is, that the sensor nodes can loose the network connectivity for a longer period and would not receive timing updates, hence suffer from local clock inaccuracies again.
The Flooding Time Synchronization Protocol (FTSP) is a synchronization protocol designed for WSN. It operates by periodically flooding the network with synchronization messages and should be robust against link and node failures [12]. Werner-Allen et. al implemented the FTSP in their sensor network to perform volcano monitoring. They reported an inaccurate synchronization by FTSP, caused mainly by the unstable network topology, which resulted in time offsets from several hours [16]. This additional contextual information is transmitted with the sensor data and enables to reconstruct a global time stamp for this measurement at the sink. However, Elson and Römer state that a global time synchronization performed by Network Time Protocols (NTP) is inappropriate for WSN. In their opinion the sensor nodes should maintain their own timescale. They propose an application depended network design to minimize these negative influences and state to use domain knowledge to improve data quality [4]. Another possibility to establish a global time scale is to exploit application domain properties and obtain timing information i.e. by microseismics [11] or measurements of the sunlight [6].
The second approach is to perform packet filtering at the sink node. Post processing can be performed on powerful machines without energy constraints and reducing sensor node complexity in return. On the other site, an accurate model of the data aggregation process is necessary to allow an appropriate reconstruction of the temporal order. In general, a system model includes many simplifications, because modelling the exact behaviour would result in too complex models. Traditional post processing is performed on application domain knowledge for packet filtering and outlier detection. In these cases, data cleaning and filtering relies only on the measured sensor data [14].

Keller et. al propose a two-staged post processing alternative to enhance data accuracy [7]. In the first stage, filtering is only performed on packet header information to reconstruct the temporal order using packet sojourn time, sequence counter or various time stamps. This ordered data set serves as input for the second stage, which performs filtering and outlier detection on the gathered sensor data in combination with application domain knowledge. The major difference from [7] to previous methods is, that they merely consider application header information of the packets instead of pure sensor data. Therefore, Keller et. al developed a formal model of the data acquisition and transmission process in a multi-hop WSN. The formal model is applied on the unfiltered data to reconstruct the temporal packet order. Their evaluations reveal promising results compared to a simple heuristics and real packet order. Moreover, their model-based approach delivers satisfactory results if the system behaviour is violated [7].

6. CONCLUSION

Wireless sensor networks are an interesting technology to perform monitoring of areas and processes. Environmental monitoring is one possible field of application, where WSN can support data acquisition. In this work, the PermaSense project was introduced as an example of a WSN for environmental monitoring. The focus was on the challenges of WSN in harsh environments and extreme weather conditions to ensure reliable data acquisition. Its system design and the deployment showed that the PermaDAQ architecture can endure the challenges in the alpine region. Furthermore, the multi-hop protocol Dozer was briefly introduced to highlight the reasons for duplicates and packet loss. The second part of this work covered the reconstruction of the correct temporal packet order using a model-based approach from [7]. This approach relies on a system model that specifies the properties of the WSN, as well as the data acquisition and transmission process. The model was applied on the unfiltered data sets of the PermaSense project and compared to a simple heuristics. The evaluation showed a good packet acceptance rate and that the model-based approach is a suitable method to reconstruct the temporal order of packets. Additionally, the model-based approach delivered more accurate results in contrast to the simple heuristics, though the data was obtained under non-conforming system behaviour.

The focus of this work was to give a short overview on the communication artefacts of WSN and the necessity of post data processing techniques to improve data quality. The model-based approach developed in [7] was presented as one example for post processing to correct for these effects and reconstruct the temporal packet order. In addition to that, environmental monitoring in harsh environments and its challenges for WSN have been introduced by the PermaSense project, which relies on multi-hop communication.

7. REFERENCES

[1] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuecel. PermaDAQ: A scientific instrument for precision sensing and data recovery in environmental extremes. In *Processing of the International Conference on Sensor Networks (IPSN)*, pages 265 – 276, April 2009. IEEE Computer Society.

[2] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-Low Power Data Gathering in Sensor Networks. In *Processing of the 6th International Symposium on Sensor Networks (IPSN)*, pages 450 – 459, April 2007. IEEE Computer Society.

[3] M. Ceriotti, L. Mottola, G. P. Picco, A. L. Murphy, S. Guna, M. Corra, M. Pozzi, D. Zonta, and P. Zanon. Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, pages 277 – 288, Washington, DC, USA, 2009. IEEE Computer Society.

[4] J. Elson and K. Römer. Wireless sensor networks: a new regime for time synchronization. In *SIGCOMM Comput. Commun. Rev.*, 33(1), pages 149 – 154, January 2003.

[5] S. Gruber and W. Haeberli. Mountain Permafrost. In *Permafrost Soils*, Vol. 16 of *Soil Biology*, pages 33 – 44, 2009. Springer Berlin/Heidelberg.

[6] J. Gupchup, R. Musăloiu-E., A. Szalay, and A. Terzis. Sundial: Using sunlight to reconstruct global timestamps. In *Wireless Sensor Networks*, Vol. 5432 of *Lecture Notes in Computer Science*, pages 183 – 198, 2009. Springer Berlin/Heidelberg.

[7] M. Keller, L. Thiele, and J. Beutel. Reconstruction of the correct temporal order of sensor network data. In *10th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 282 – 293, April 2011. IEEE Computer Society.

[8] M. Keller, M. Woehrle, R. Lim, J. Beutel, and L. Thiele. Comparative performance analysis of the PermaDozer protocol in diverse deployments. In *36th Conference on Local Computer Networks (LCN)*, pages 957 – 965, October 2011. IEEE Computer Society.

[9] L. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops*, pages 575 – 578, 2002. IEEE Computer Society.

[10] C.-J. M. Liang, J. Liu, L. Luo, A. Terzis, and F. Zhao. RACNet: a high-fidelity data center sensing network. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, (SenSys)*, pages 15 – 28, New York, NY, USA, 2009. ACM.

[11] M. Lukac, P. Davis, R. Clayton, and D. Estrin. Recovering temporal integrity with data driven time synchronization. In *Proceedings of the International Conference on Information Processing in Sensor Networks, (IPSN)*, pages 61–72, Washington, DC, USA, 2009. IEEE Computer Society.

[12] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 39–49, New York, NY, USA, 2004. ACM.

[13] R. Schmitt and C. Osenberg. Detecting Ecological Impacts (1st Edition) - Concepts and Applications in Coastal Habitats. Academic Pr Inc, 1996.

[14] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Wireless Sensor Networks*, Vol. 2920 of *Lecture Notes in Computer Science*, pages 307–322, 2004. Springer Berlin/Heidelberg.

[15] I. Talzi, A. Hasler, S. Gruber, and C. Tschudin. Permasense: Investigating Permafrost with a WSN in the Swiss Alps. In *Proceedings of the 4th workshop on Embedded Networked Sensors (EmNets)*, pages 8 – 12, New York, NY, USA, 2007. ACM.

[16] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI)*, pages 381–396, Berkeley, CA, USA, 2006. USENIX Association.

TLS solutions for WSNs

Philipp Lowack
Betreuer: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2012
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitekturen
Fakultät für Informatik, Technische Universität München
Email: lowack@in.tum.de

ABSTRACT

As most nodes of wireless sensor networks are not very powerful in regard to computational operations and additionally very limited in power supply, it is not a trivial task to deploy security features. As more applications for wireless sensor networks are developed, it becomes more and more important and necessary to also provide secure communication between nodes and to external terminals. In this paper, three already existing approaches for using the Transport Layer Security protocol in combination with wireless sensor networks are presented. The first uses identity based cryptography and bilinear pairing based on elliptic curve cryptography instead of RSA. The second solution modifies the TLS handshake in order to establish a single TLS session between multiple entities. In the third solution expensive cryptographic operations are outsourced to a more powerful gateway node. For each solution the principles of operation are explained and afterwards the presented approaches are compared in regard to applicability and energy consumption.

Keywords

Wireless Sensor Networks, secure communication, Transport Layer Security, bilinear pairing, Elliptic Curve Cryptography, Tiny 3-TLS

1. INTRODUCTION

The `Transport Layer Security (TLS)` protocol is a widespread and well accepted cryptographic protocol for encrypting and integrity-protecting the payload of network data streams. Because of its transparency in regard to the outer and inner protocols, it is nowadays widely used in many different applications. The perhaps most well-known protocol which uses TLS is the `Secure HyperText Transport Protocol (HTTPS)` which is used by everyone while surfing in the Internet. But TLS can be used with almost any kind of protocol and offers sophisticated security for simple private websites as well as for large commercial banking systems.

The question for researchers is: is this protocol also suitable for embedded applications with very limited computing power and where the power consumption of algorithms is an essential criterion?

In this paper three separate already existing approaches are analyzed and compared in regard to their suitability for embedded platforms. To accomplish this task, at first in Section 1.1 a short motivation is given why it is not trivial to find a solution. Then an introduction to TLS is given in Section 1.2, explaining the basic working principles. Then the three papers are presented in Section 2 and each one is explained. The solution presented in Section 2.1 uses identity based cryptography and bilinear pairing to reduce the number of packets for the TLS handshake. The second solution, presented in Section 2.2, adapts the TLS handshake protocol to be able to establish a session between multiple entities. The last solution, presented in Section 2.3, makes the use of TLS feasible on embedded platforms by offloading as much cryptographic operations as possible to a more powerful gateway node without interrupting the security model. The last Section 3 of this paper compares the previously presented approaches in regard to applicability and energy consumption, followed by a conclusion.

1.1 Motivation

`Wireless Sensor Networks (WSN)` are networks of embedded systems which communicate via a wireless network. These embedded systems, called `nodes`, are usually limited in computing power as well as in power supply because only microprocessors with battery supply are used. The nodes are deployed in an area to collect sensor measurements which are then transfered to a central entity. The data is routed through the network via multiple nodes and finally arrives at the central entity, called gateway node, which translates between the wireless network and an external network, for example the Internet, in order to send the data to its destination or allow an operator to manage the WSN. The gateway node is usually not as limited as the sensor nodes and may even have a static power supply.

If sensitive sensor data is collected, for example health-data of patients in a hospital or security information of an alarm system, it is desirable to protect this data against unwanted disclosure and unauthorized modifications. In order to achieve this goal encryption algorithms and message authentication codes can be used.

Of course, there are many protocols available for this purpose. Renowned protocols are `IPsec`[9] and `Transport Layer Security (TLS)`[5]. The disadvantage with `IPsec` is that it encrypts the network layer payload and therefore also non-critical data like for example TCP handshakes and adds more extra size to the original packet than TLS [1]. As the computing power of embedded devices is strictly limited, this additional overhead is undesirable. TLS however works between the transport and the application layer and

therefore reduces said overhead. It offers privacy and data integrity and, by using asymmetric key cryptography, also can authenticate the sender of a message. Additionally, TLS is transparent to the operating system and the network and can be used to encapsulate any other protocol.

When implementing and deploying encryption in a WSN the two key points to consider are the computational impact and the implicated power consumption of encryption algorithms. Therefore not all available algorithms are applicable. But this does not only concern the encryption of the payload data, but most importantly the costly operations to do public-key cryptography when initializing a session. Another very important aspect to consider is the number of packets to send. Transmitting data via a radio link consumes more energy the stronger the signal is. If the distribution of the sensor nodes is very sparse and therefore the signal has to be rather strong, transmitting more data strongly affects the power consumption. But also if the nodes are distributed rather dense and therefore the signal does not have to be that strong, it is still desirable to limit the amount of data to send.

1.2 Introduction to TLS

The goal of TLS is to provide a secure communication channel between two applications. The established channel ensures confidentiality and data integrity and also offers replay protection.

The `Secure Socket Layer (SSL)` was originally developed by Netscape and the first draft was released to the public in 1995 [8]. In 1999, the `Transport Layer Security (TLS)` protocol was released as an upgrade to SSL 3.0, but because of "significant changes" was not interoperable with SSL [4]. The latest version of TLS is 1.2 and has been released in 2008 [5].

TLS itself is two-layered and consists of multiple sub-protocols. The `TLS Record Protocol` builds the lower layer of TLS and is responsible for transporting any data, may it be of other TLS sub-protocols or of higher level protocols. This layer is also responsible for encrypting and integrity protecting the data. As asymmetric cryptographic operations are very expensive, the TLS Record Protocol uses a symmetric algorithm for the bulk encryption of the data.

The most important sub-protocol of the upper TLS layer is the `TLS Handshake Protocol`. It is responsible for establishing a session by negotiating security parameters and can also be used to authenticate the two endpoints. Other protocols of the upper layer are the `Change Cipher Spec Protocol` (switches the currently used algorithms and keys), the `Alert Protocol` (communicates alerts to the remote endpoint) and the `Application Data Protocol` (encapsulates higher level data).

Because of the huge number of available encryption and hash algorithms, TLS uses so called `cipher suites` to organize them. A cipher suite is a set of four algorithms, namely a key-exchange algorithm, a bulk-encryption algorithm, a message authentication code and a pseudorandom function. The key-exchange algorithm determines how the server and the client authenticate each other during the session initialization. The bulk-encryption algorithm is used by the `TLS Record Protocol` to encrypt the data. The message authentication code algorithm is also used by the `TLS Record Protocol` to protect the data's integrity. The pseudorandom function is a cryptographically secure pseudorandom number generator used for generating keying material.

TLS is mostly used along with asymmetric cryptography and certificates. When a `Public Key Infrastructure (PKI)` is not available or authentication is not necessary the Diffie-Hellman key exchange algorithm (or one of its variants) can be used instead of public and private keys [5]. But as this technique is vulnerable to man-in-the-middle attacks it is barely ever used.

The following explanation of the Handshake Protocol is based on the use of public key cryptography with certificates. In order to prevent the explanation from getting to complex, it is limited to essential elements which are necessary to understand TLS itself and the improvements that will be presented. Therefore some messages are omitted. In order to provide a consistent naming scheme, Table 1 defines the used symbols and their meaning.

Symbol	Description
{x}_K	encryption of message x with key K
a|b	concatenation of messages a and b
Pub-x, Pub_x	the public key of entity x
Cert-x, $Cert_x$	the certificate of entity x (includes Pub_x)
N-x, N_x	nonce choosen by entity x
ECDH-x, $ECDH_x$	the public ECDH values of entity x
E-q, E_q	an elliptic Curve
P	a point on E_q

Table 1: Variables used in this paper

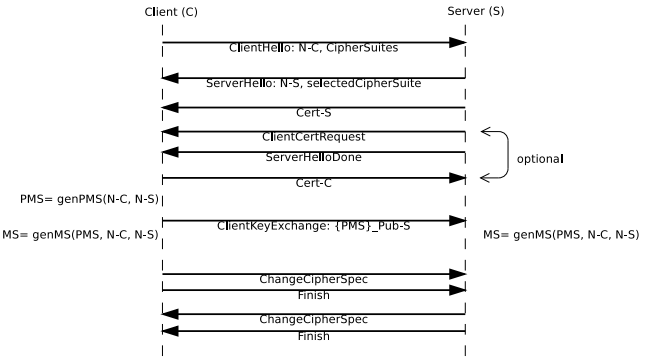


Figure 1: Standard TLS handshake using asymmetric cryptography. [5]

The basic message flow for establishing a TLS session is depicted in Figure 1. As TLS works with exactly two endpoints, the one establishing the connection is named `client`, the other one is named `server`. The client initiates the hello phase of the `TLS Handshake Protocol` by sending an (unencrypted) `ClientHello` message to the server. This message contains mainly a random number (named `nonce`, number only used once) and all supported cipher suites of the client.

doi: 10.2313/NET-2012-08-2_06

The server also generates a nonce, chooses an acceptable cipher suite out of the received ones and sends these values in a `ServerHello` message to the client. The server also sends his certificate, which contains his private key, to the client. If the client needs to be authenticated, the server also sends a requests for the client's certificate. The `ServerHelloDone` message indicates the end of the hello phase. If the client's certificate has been requested by the server, the client will send it to the server upon reception of the `ServerHelloDone` message.

Now begins the authentication phase. In order to ensure authenticity, the received certificates are now verified. If the certificates can be verified, the client generates a so called `pre-master secret` from both random numbers, encrypts it with the servers public key and sends it in a `ClientKeyExchange` message to the server. The server can decrypt the pre-master secret with its private-key. As both parties now are in the possession of the pre-master secret and both random numbers, they both can generate the `master secret` from these values. The master secret is used for generating the session keys for the encryption of all further data.

In the finish phase, the protocol switches to the new cipher suite and session keys. The client sends a `change cipher spec` message to signal that the negotiated parameters are valid and should be used now. The server also answers with a `change cipher spec` message and now expects encrypted messages. In order to verify that the encryption works correctly, both endpoints send a `finish` message to each other and verify that they are able to decrypt the received message.

In regard to WSN, there are also some disadvantages with TLS. One problem is that TLS oftentimes uses RSA for asymmetric cryptography as this algorithm is considered very secure. But the computational impact of the RSA algorithm is very high, resulting in high energy consumption when encrypting or decrypting data. Though with the recent progress on `Elliptic Curve Cryptography (ECC)` this problem has become less severe. A 224-bit key used with ECC is considered as secure as a 2048-bit key used with RSA and at the same time ECC is as fast as the RSA public-key operation and even outperforms the RSA private-key operation by one order of magnitude [7]. With longer keys this results in an even greater advantage for ECC. Another problem with TLS is the number of packets in the handshake. The standard handshake with mutual authentication needs 13 packets. As the radio module needs a relatively large amount of energy when transmitting a signal, even saving only a few packets per handshake results in a significant energy saving, as the number of performed handshakes can be very large. Designing power-aware protocols is therefore very important for WSNs [2]. All these problems combined make it non-trivial to implement TLS-based security on such limited hardware as it is used in WSNs.

2. TLS SOLUTIONS FOR WSN

In the following part three already existing solutions for securing communication in a WSN using TLS are presented. The first one uses identity based cryptography and bilinear pairing in order to reduce the number of sent messages. It is followed by an approach which enables TLS to establish a shared session between multiple parties. The third solution, called Tiny 3-TLS, offloads expensive cryptographic operations from the sensor node to the gateway node in order to allow for very performance limited sensor nodes. All solutions will be compared in Section 3.

2.1 Adapting TLS for Identity Based Cryptography

When using public-key cryptography while also allowing peer to peer communication, a certificate for each node is necessary. Additionally the whole certificate path to the root certificate authority needs to be known in order to verify a certificate. All these certificates need to be stored on each node and occupy precious storage.

Mzid et al. describe in [12] how to modify the TLS handshake protocol to support identity based cryptography in an IP-based WSN in order to save this space.

`Identity Based Cryptography (IBC)` is based on ECC and eliminates the need for certificates [13]. In the context of WSNs the (public) IP address of a node can be used as its public key. Instead of a PKI only a `Private Key Generator (PKG)` is needed, which takes the role of the Trusted Third Party. For each node, its private key is generated by the PKG and then, bundled with some further parameters like the elliptic curve itself as well as a point P on that curve, transferred to and stored on the node before deployment. The key must only be known to the node and the PKG. All other parameters may be publicly known and the elliptic curve even has to be the same for all nodes. Using IBC together with elliptic curve based Diffie-Hellman key exchange (`ECDH` [11]) as shown in Figure 2, it is now possible to reduce the number of necessary handshake protocol packets from 13 to 10.

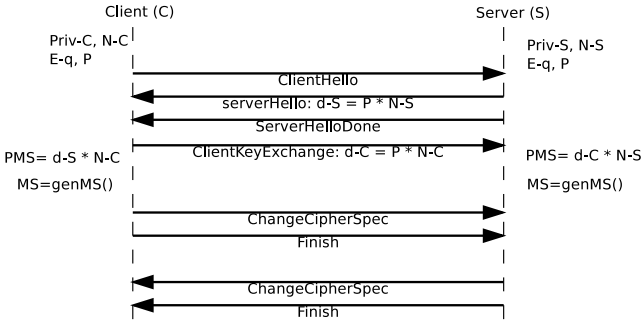


Figure 2: TLS handshake using IBC and ECDH. [12]

The second proposed improvement to the handshake protocol is the use of bilinear pairing, which is the basis for a different key exchange algorithm. Let G_1 denote a cyclic additive group of some large prime order $q = 2^m$ and G_2 denotes a cyclic multiplicative group of the same prime order. Then a map $e : G_1 \times G_1 \mapsto G_1$ is called a pairing. This map has the very special property of bilinearity: if $P, Q \in G_1$ and $a \in Z_q^*$ then $e(aP, Q) = e(P, aQ) = e(P, Q)^a$. More details can be found in reference [10]. This mathematical construct enables two parties to generate a shared secret without any interaction between them.

doi: 10.2313/NET-2012-08-2_06

In order to use bilinear pairing, two separated phases are considered. In the pre-deployment phase, the PKG chooses several parameters:

- an elliptic curve E_q,
- $G_1 \subseteq E_q$ and G_2,
- a bilinear map e as described above,
- a hash function H which returns points of the elliptic curve, and
- a random number $S \in Z_q^*$.

Except S all values can be publicly known. The PKG generates the private key of a node calculating $Priv_i = S * H(IP_i)$ where IP_i is the IP address of node i. To finish the pre-deployment phase, the tuple $< G_1, G_2, e, q, Priv_i, H >$ is transfered to and stored on the nodes.

In the post-deployment phase, two nodes can generate the pre-master secret as soon as they know the address of the remote end by calculating:

- on the server: $e(H(IP_{client}), Priv_{server})$
- on the client: $e(Priv_{client}, H(IP_{server}))$

As both parties are in possession the remote IP address after the ServerHello message[1], the number of necessary packets can be reduced from 13 to only 7. Figure 3 illustrates the improved handshake.

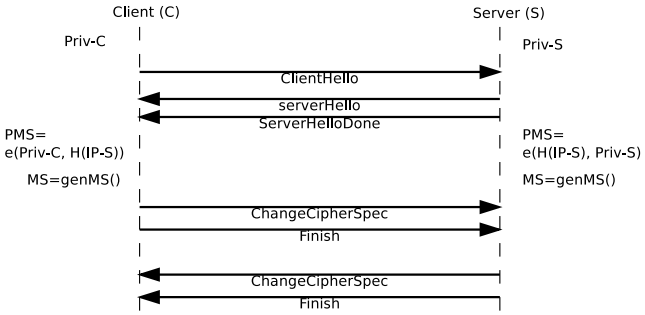


Figure 3: TLS handshake using bilinear pairing. [12]

Mzid et al. not only propose the improvements but also evaluates them in regard of storage costs, handshake duration and energy consumption [12]. The storage costs are significantly reduced by the use of IBC as no certificates need to be stored. Each node only has to store its private key as the public key can be derived from an identity. The duration of the handshake is especially important for real-time applications but it is not only then desirable to reduce this delay. Both improvements reduce the duration of the handshake. And when using both server and client authentication, the improvement is almost by an order of magnitude as two certificate verification operations are needless. Also both improvements reduce the energy consumption of the handshake by an order of magnitude.

[1]Client and server have first to agree on a cipher suite that uses bilinear pairing.

2.2 Using a single TLS session for multiple entities

Badra proposes a solution to establish a TLS session between more than two entities [3]. The new behavior is implemented by a new set of cipher suites which reuse existing RSA based cipher suites. Solely the TLS handshake protocol is modified to establish a session between multiple entities. As all entities then have the session keys, they can freely communicate with each other.

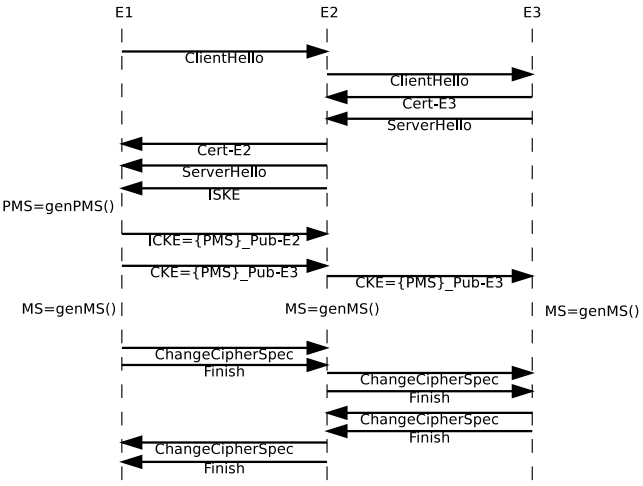


Figure 4: Handshake between three entities. [3]

For the following part, three entities are considered, but the protocol can be easily adapted to any number of entities. The basic message flow is depicted in Figure 4. The modified handshake happens as follows: The first entity (E_1) connects to the second entity (E_2) and sends a ClientHello message which includes one or more of the new cipher suites in the list of supported cipher suites and a random number. If E_2 supports the new cipher suites, then one of them is selected. E_2 now sends a ClientHello message to the third entity (E_3) which includes the selected cipher suite and the same random number as received by E_1.

If the cipher suite sent by E_2 is accepted by E_3, it sends a ServerHello message containing a session identifier, a new random value and the selected cipher suite to E_2. E_2 and E_3 now finish the hello phase and E_3's ServerHello message as well as its certificate are forwarded by E_2 to E_1. In addition, the new `IntermediateServerKeyExchange (ISKE)` message is also sent to E_1. It contains a list of all host names involved in the session and E_2's certificate.

In the authentication phase, E_1 generates a pre-master secret. The pre-master secret is then encrypted once with E_3's public key to get the `EncryptedPreMasterSecret` and once with E_2's public key to get the `InterEncryptedPreMasterSecret`. The InterEncryptedPreMasterSecret is sent via the new `IntermediateClientKeyExchange (ICKE)` message to E_2 and the EncryptedPreMasterSecret is sent to E_2 via the `ClientKeyExchange (CKE)` message, which also includes the previously received list of involved host names. Also, E_1 immediately sends its finish message.

doi: 10.2313/NET-2012-08-2_06

E_2 can now compute the master-key from the pre-master secret included in the IntermediateClientKeyExchange message and therefore verify E_1's certificate, signature and finish message. All messages except the IntermediateClientKeyExchange message are forwarded to E_3. E_3 can now decrypt the ClientKeyExchange message, compute the master key and also verify E_1's certificate, signature and finish message. If these operations are successful, a finish message is sent to E_2 which forwards it to E_1. After E_1, and optionally also E_2, verified E_3's finish message, the TLS handshake is complete and all involved entities are in possession of the master key. If one of the entities now wants to communicate with any other involved entity, it just sends a ClientHello message containing the previously stored session identifier to the other entity[2] and is able to securely communicate with the other entity without performing a full handshake.

This design is fully backwards compatible with standard TLS and even allows to communicate with non-modified clients. Furthermore this approach theoretically performs better than establishing separate TLS sessions, as for N entities at best $N-2$ signature operations, $N-2$ certificate verification operations and $N-2$ signature verification operations are saved.

2.3 Tiny 3-TLS

The goal of "Tiny 3-TLS" as presented in [6] is to offload expensive cryptographic operations to the more powerful gateway node in order to relieve the sensor nodes without putting the security at risk. The network model of the paper is as follows. A WSN consisting of multiple sensor nodes is connected to an external network via a gateway acting as reverse proxy. A so called "remote monitor" R wants to establish a secure connection to one of the sensor nodes S. Each sensor node has a pre-shared key K which is known to the gateway G. In order to ensure confidentiality, each sensor node should have a different pre-shared key.

Two distinct modes of operation are presented which differ in the trust-level of the gateway. If the gateway is fully trusted, it can handle all of the cryptographic operations of the TLS handshake and then provide the sensor node with the actual session keys. That way, the gateway is also in possession of the secrets. If the gateway is only partially trusted, then the gateway does handle the asymmetric encryption and decryption of the TLS-handshake packets but the generation of the session keys happens at the remote monitor and the sensor node via ECDH. This way, the gateway does not know the session keys. In both cases, all three parties are mutually authenticated. The gateway and the sensor nodes of the WSN authenticate each other by proving the knowledge of the pre-shared key K. The gateway and the remote endpoint authenticate each other via certificates. The trust between the remote endpoint and the sensor node is then transitively achieved. If on the other hand the gateway is not trusted at all, for example because it is either unknown or publicly available, this solution is not applicable. An untrusted or even malicious gateway could easily perform a man-in-the-middle (MITM) attack and successfully capture or manipulate any data.

[2]TLS calls this a resumed handshake.

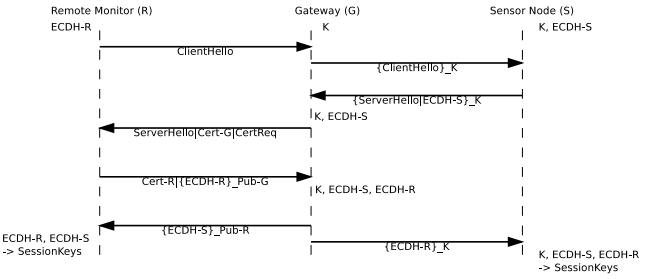


Figure 5: Tiny3-TLS with a partially trusted gateway. [6]

The message flow in case of a partially trusted gateway is depicted in Figure 5. When the remote monitor wants to communicate with a sensor node it sends a ClientHello message, which is encrypted with the appropriate pre-shared key by the gateway and relayed to the sensor node. The node answers with a ServerHello message encrypted by K and also its public ECDH values. The gateway decrypts the message, keeps the ECDH values $ECDH_S$ and forwards the ServerHello message to the remote monitor. It also adds its own certificate and a request for the certificate of the remote monitor. The remote monitor validates the gateway's certificate and, if valid, answers with his own certificate and his public ECDH values $ECDH_R$, both encrypted with the gateway's public key. If the gateway can validate the remote monitor's certificate it sends $ECDH_R$, encrypted with K, to the sensor node and $ECDH_S$, encrypted by the remote monitor's public key, to the remote monitor. Now the sensor node and the remote monitor can generate the session keys from $ECDH_R$ and $ECDH_S$ and can encrypt all further direct communication.

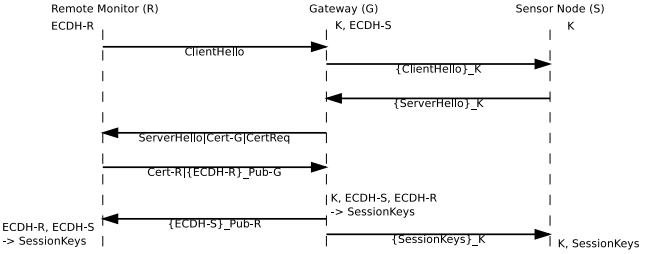


Figure 6: Tiny 3-TLS with a fully trusted gateway. [6]

In case of a fully trusted gateway, the handshake happens as before, but this time $ECDH_S$ is not selected by the sensor node but directly by the gateway. The message flow is depicted in Figure 6. That way the gateway itself can generate the session keys without any interaction of the sensor node. Once the session keys are generated, the gateway encrypts them with the pre-shared key and sends them to the sensor node.

This approach can, if applicable, reduce the load on the sensor node drastically. Instead of expensive asymmetric cryptographic operations like signature verifications only rather inexpensive symmetric encryption and decryption operations are necessary. But this comes at the cost that the gateway

doi: 10.2313/NET-2012-08-2_06

	Protocols			
	TLS	Single-TLS-Session	Tiny-3TLS	IBC
Uses certificates	x	x	x	-
Based on	RSA	RSA	ECC	ECC
usable for P2P	x	x	-	x
Ext. → Node	x	x	x	x
Performance	13 messages	saves: $N-2$ signature operations $N-2$ signature verifications $N-2$ certificate verifications	GW: 7 message Node: 3 messages	7 messages

Table 2: Comparison of the presented solutions

node has a much higher load.

3. CONCLUSION

All three solutions presented in this paper help to secure communication in a WSN environment. The improvements presented in Section 2.1 reduce the number of necessary packets for a TLS handshake and also eliminate the need for certificates and a PKI. This also significantly improves the energy consumption of the nodes, enabling them to operate for a longer period of time, or even making it possible to lower the hardware requirements. Establishing a single TLS session for multiple nodes as described 2.2 also saves packets and therefore energy. But as that paper did not have embedded systems in mind, power consumption was not a key point of the improvement. This approach is also only applicable when it is desirable to share a single set of session keys for multiple entities. But in an environment where data streams of different nodes must be protected from each other, for example sensor nodes collecting health-data from patients in a hospital, it is not desirable that all nodes share the same keys. Finally, Tiny 3-TLS, presented in Section 2.3, only secures the communication with an external remote terminal, but does not provide secure point to point communication within the WSN.

Tiny 3-TLS and the paper about establishing a single TLS session for multiple entities have each a special use-case in mind for which they try to find an optimal solution. Therefore they do not directly compete with each other as different environments and use-cases require different solutions. As for IBC and bilinear pairing, these approaches can help the former solutions to save more energy and other resources and therefore further boost them.

References

[1] A. Alshamsi and T. Saito. A technical comparison of ipsec and ssl. In *19th International Conference on Advanced Information Networking and Applications, 2005. AINA 2005.*, volume 2, pages 395–398. IEEE, 2005.

[2] S. Aslam, F. Farooq, and S. Sarwar. Power consumption in wireless sensor networks. In *Proceedings of the 7th International Conference on Frontiers of Information Technology*, page 14. ACM, 2009.

[3] M. Badra. Securing communications between multiple entities using a single TLS session. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1–4. IEEE, 2011.

[4] T. Dierks and C. Allen. The TLS protocol. IETF RFC 2246, January 1999.

[5] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol, version 1.2. IETF RFC 5246, August 2008.

[6] S. Fouladgar, B. Mainaud, K. Masmoudi, and H. Afifi. Tiny 3-TLS: A trust delegation protocol for wireless sensor networks. In L. Buttyán, V. Gligor, and D. Westhoff, editors, *Security and Privacy in Ad-Hoc and Sensor Networks*, volume 4357 of *Lecture Notes in Computer Science*, pages 32–42. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-69172-3.

[7] N. Gura, A. Patel, A. W, H. Eberle, and S. C. Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. 2004.

[8] K. E. Hickman. The SSL protocol. Netscape Communications Corp., Feb 1995.

[9] S. Kent and K. Seo. Security architecture for the internet protocol. IETF RFC 4301, December 2005.

[10] D. Meffert. Bilinear pairings in cryptography. Master's thesis, Radboud Universiteit Nijmegen, 2009.

[11] V. Miller. Use of elliptic curves in cryptography. In H. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin / Heidelberg, 1986. ISBN 978-3-540-16463-0.

[12] R. Mzid, M. Boujelben, H. Youssef, and M. Abid. Adapting TLS handshake protocol for heterogenous IP-based WSN using identity based cryptography. In *International Conference on Communication in Wireless Environments and Ubiquitous Systems: New Challenges (ICWUS), 2010*, pages 1–8. IEEE, 2010.

[13] A. Shamir. Identity-based cryptosystems and signature schemes. In *Advances in cryptology*, pages 47–53. Springer, 1985.

Wireless Body Area Networks (WBAN)

Michael Ederer
Betreuer: Christoph Söllner, Corinna Schmitt
Seminar: Sensorknoten - Betrieb, Netze und Anwendungen
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitekturen
Fakultät für Informatik, Technische Universität München
Email: michael.ederer@in.tum.de

KURZFASSUNG

Der technologische Fortschritt auf dem Gebiet der kabellosen Kommunikation sowie in der gesamten Halbleiterbranche hat einen signifikanten Einfluss auf den Bereich der Sensornetzwerke, die in einer Fülle von unterschiedlichen Anwendungen verbreitet sind. Thema dieses Papers sind sogenannte Wireless Body Area Networks (WBAN), welche vorrangig zur Erfassung von Vitalfunktionen eingesetzt werden können. Hierbei werden die Daten von unterschiedlichen Sensoren erfasst und an einen zentralen, intelligenten Sensorknoten zur weiteren Verarbeitung kabellos übermittelt. Dieses Konzept beschränkt sich auf die Anbindung der am Körper getragenen Sensorik, wobei das jeweilige drahtlose Netzwerk nur genau die Sensoren eines bestimmten Trägers umfasst. Im weiteren Verlauf dieses Papers werden neben der zugrundeliegenden Architektur die typischen Einsatzszenarien sowie die Bestandteile eines WBANs dargestellt.

Schlüsselworte

Wireless Body Area Networks, WBAN, Sensornetz, Activity Monitoring, Medical Monitoring

1. EINLEITUNG

Im Profisport werden die Athleten mittels neuester Computertechnik ständig beobachtet, meistens unter Verwendung von hochwertigen Tracking-Systemen. Aber auch abseits des Profisports lassen sich neuerdings Leistungswerte relativ einfach ermitteln, durch den Einsatz von WBAN basierenden Anwendungen [17]. Der Bereich der Wireless Body Area Networks profitiert wie alle Sensornetzwerke vom rasanten Fortschritt im Bereich der drahtlosen Kommunikation sowie der Miniaturisierung elektronischer Schaltkreise [3]. Beflügelt durch den technischen Fortschritt bieten WBANs eine breite Palette an möglichen Anwendungsgebieten. Ein wichtiges Anwendungszenario eines WBAN Systems könnte im Bereich der intelligenten medizinischen Patientenüberwachung [4], sowie im Sportsegement zur Erfassung von Körperfunktionen, liegen. Durch die große Vielfalt an existierenden Sensoren können die verschiedensten Applikationen realisiert werden. Traditionelle Überwachungssysteme wie etwa ein Elektrokardiogramm basierten bisher auf kabelgebundenen Sensoren, welche nur im offline Betrieb verwendet wurden. D.h. die erfassten Daten werden nicht weiterkommuniziert sondern direkt verarbeitet. Ein kabelloses Monitoring System könnte unter Verwendung eines Body Area Networks (BAN) am Körper des Nutzers angebracht oder in dessen Kleidung integriert werden [7]. Mit Sensorik ausgestattete Kleidung ist kein neues Thema. So wurde zum Beispiel bereits um die Jahrtausendwende an der Universität von Rochester an intelligenten Socken und Bandagen geforscht [12]. Allgemein betrachtet stellen sogenannte „smart clothes“ eine Alternative zu WBAN basierenden Systemen dar [4]. Bei diesem Ansatz wird auf drahtlose Kommunikation verzichtet, was den Preis der Sensorik verringert, allerdings mit dem Nachteil der eingeschränkten Erweiterbarkeit.

Im 2. Abschnitt dieses Papers wird neben dem Begiff des WBANs die Architektur sowie typische Herausforderungen an ein solches System definiert. Kapitel 3 gibt einen Überblick über typische Einsatzszenarien, welche anhand von zwei Anwendungsbeispielen im medizinischen und kommerziellen Segment illustriert werden. Abschließend wird in Abschnitt 5 eine kurze Zusammenfassung des Papers präsentiert.

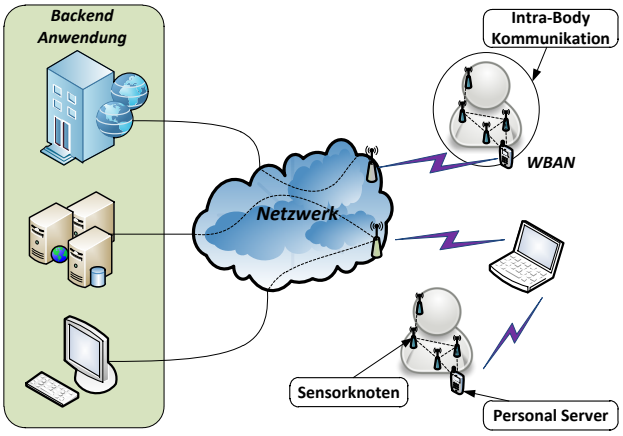


Abbildung 1: Illustration eines WBANs

2. DEFINITION EINES WBANS

Ein drahtloses körpernahes Netzwerk (Wireles Body Area Network - WBAN) stellt ein spezialisiertes Netzwerkkonzept dar, das sich im IEEE 802.15 Standard der Wireless Personal Area Network - WPAN einordnen lässt (vgl. Abbildung 2). Der prinzipielle Unteschied zwischen WBAN, das in der IEEE Working Group sechs und sieben [8] bearbeitet wird und WPAN liegt in den verschiedenen Reichweiten. Man spricht von einem WPAN typischerweise bei einer

doi: 10.2313/NET-2012-08-2_07

Entfernung von 0,2 - 50 Meter. Bei einem WBAN liegt die

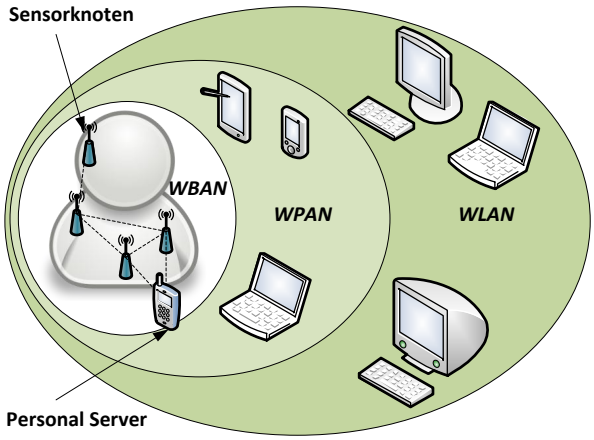


Abbildung 2: Einordnung eines WBANs

maximale Ausbreitung bei etwa zwei Metern [8]. Funktionell entspricht ein WBAN den sogenannten Body Area Networks (BAN), welche hauptsächlich zur Erfassung von Vitalfunktionen eingesetzt werden. Die wesentliche Abgrenzung liegt in der drahtlosen Kommunikation der Sensorik. Bei diesem Konzept werden am Körper getragene oder implantierte Sensoren via Funktechnologie verbunden, wobei jeweils nur die Geräte genau einer Person zu einem WBAN zusammengefasst werden. Die heterogenen Sensorknoten kommunizieren kabellos mit einem zentralen, intelligenten Knoten, welcher für die weitere Verarbeitung bzw. Übertragung der erfassten Sensordaten zuständig ist. Ein Wireless Body Area Network besteht im wesentlichen aus drei Segmenten:

1. Erfassung der Vitalfunktionen
2. Zentrale Datensammlung
3. Datenauswertung/ Analyse

Abbildung 1 stellt die mögliche Struktur eines WBANs mit den zuvor aufgeführten Teilbereichen dar. Die am Körper angebrachte Sensorik erfasst Vitalfunktionen wie Blutdruck, Herzfrequenz oder Körpertemperatur und übermittelt diese an einen zentralen Knoten, welcher zum Beispiel durch ein Smartphone oder einen PDA repräsentiert werden kann. Häufig wird der zentrale Knoten als CCU (Central Control Unit) [1] oder als Personal Server [7] bezeichnet, wobei es sich beim Personal Server in der Regel um eine Anwendung handelt, welche auf dem jeweiligen Endgerät installiert wird. Hingegen entspricht die CCU einem eigenständigen Gerät. Diese Tatsache ermöglicht bereits Rückschlüsse auf typische Einsatzgebiete dieser beiden Varianten. So kann der Personal Server im privaten Sektor und die Variante mit einer CCU eher im Krankenhausbereich angesiedelt werden [1]. Der zentrale Knoten agiert, abhängig vom Einsatzgebiet als Benutzerschnittstelle oder eine Art Gatewayknoten, welcher die Daten über ein beliebiges Kommunikationsnetzwerk an eine Remote-Anwendung übermittelt. Gerade bei der Smartphone Variante ist oftmals eine Kombination der beiden Features die Regel. In den nachfolgenden Teilabschnitten dieses Kapitels werden die einzelnen Sektoren eines WBANs genauer beschrieben.

2.1 Architektur

Wie im einleitenden Abschnitt diese Kapitels bereits erwähnt, besteht ein Wireless Body Area Network im wesentlichen aus drei Segmenten, wobei jeder Bereich durch eigenständige Netzwerkkomponenten repräsentiert wird. Abbildung 1 zeigt eine Multi-Tier-Architektur eines typischen WBANs, welche sich in drei Segmente mit zwei möglicherweise unterschiedlichen Kommunikationsverbindungen einteilen lässt. Dabei stellt der erste Bereich, bestehend aus der einfachen Sensorik und dem zentralen Knoten die Basisfunktion eines WBANs dar. Der Kommunikationskanal dieser beiden Netzsegmente wird in Abbildung 3 durch die kabellose Verbindung, betitelt als 1. Hop, dargestellt. Im Prinzip entspricht dieses Subsegment dem eigentlichen Body Area Network. Das zweite Teilsegment bildet sich aus dem zentralen Knoten und der möglichen Remote-Anwendung, zum Beispiel in Form einer Patientendatenbank. Die Kommunikation dieser beiden Bereiche wird durch den 2. Hop aus Abbildung 3 visualisiert.

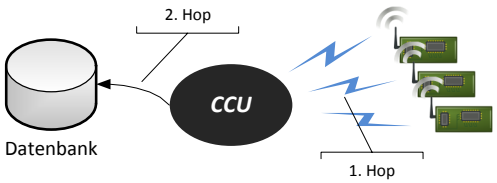


Abbildung 3: Multi-Hop Architektur

2.1.1 Sensorebene

In einem Wireless Body Area Network kann eine Vielzahl von unterschiedlichen Sensoren, aber auch Aktuatoren eingesetzt werden [6]. Hauptaufgabe der Sensor Ebenen besteht in der Erfassung biometrischer Daten und der direkten Interaktion mit dem Träger eines solchen Systems. Somit stellt dieser Bereich eines WBANs, welcher zwingend immer unmittelbar am Körper getragen werden muss, die Basisfunktionalität dar. Hauptbestandteil dieser Ebenen sind kabellose Sensorknoten, welche verschiedene Sensoren und Aktuatoren beinhalten können. Hierbei sind unterschiedliche Konfigurationen möglich. So kann ein WBAN dahingehend aufgebaut sein, dass jeweils pro Sensorknoten nur ein Sensor oder Aktuator eingesetzt wird. Im Falle eines einzelnen Aktuators spricht man auch von einem Aktuator Knoten. Ein solcher könnte zum Beispiel ein Reservoir für Medikamente beinhalten und diese, getriggert durch ein externes Ereignis, in den Organismus injezieren. Eine weitere Möglichkeit des Sensorknoten Designs liegt darin, dass pro Knoten mehrere Sensoren oder Aktuatoren angebracht bzw. angebunden sind.

Ein klassischer Sensorknoten ist dahingehend aufgebaut, eines der nachfolgenden physiologischen Signale zu erfassen

doi: 10.2313/NET-2012-08-2_07

und zu verarbeiten [13][1].

- Elektrokardiogramm (ECG)
- Photoplethysmogram (PPG)
- Elektroenzephalografie(EEG)
- Puls
- Blutfluss
- Blutdruck
- Sauerstoffsättigung des Blutes
- Temperatur
- Atemfrequenz
- Bewegung

Die vom eigentlichen Sensor erfassten analogen Signale werden im Sensorknoten digitalisiert, kodiert und für die drahtlose Übertragung aufbereitet. In welchem Intervall erfasste Vitaldaten vom Sensorknoten aus weiter propagiert werden, ist wieder stark anwendungsspezifisch. Auch bestehen für die verschiedenen Signale, aufgrund der unterschiedlichen physiologischen Eigenschaften, verschiedenartige Anforderungen zum Beipsiel bezüglich der Abtastrate eines bestimmten Signales [7].

Zur drahtlosen Kommunikation werden meistens Übertragungsverfahren mit geringer Sendeleistung und kurzer Reichweite im Frequenzbereich von 2,40 GHz eingesetzt [8]. Ein typisches Funkprotokoll im Bereich der WBAN Systeme, welches hier kurz vorgestellt werden soll, stellt das sogenannte ANT+ Protokoll dar. Dieses Protokoll basiert auf dem ANT Protokoll Stack, welcher die ersten vier Schichten des ISO-OSI Modells implementiert. ANT+ arbeitet im Frequenzband zwischen 2,403 GHz - 2,480 GHz auf insgesamt 78 Kanälen. Ein ANT+ Sensornetzwerk besteht aus eigenständigen ANT+ Teilnehmern welche in verschiedenen Topologien zusammengefasst werden können. Vor allem die geringe Leistungsaufnahme, welche im Bereich von 25 - 30 Mikroampere liegt, qualifizieren diese Technologie für den Einsatz in WBAN Systemen [16].

2.1.2 Zentraler Sensorknoten

Die zentrale Datensammlung der einzelnen Sensorwerte geschieht in einem zentralen Sensorknoten. Dieser stellt eine Art Schnittstelle zwischen der Sensorik und der weiteren Datenverarbeitungsanwendung dar. Hierbei sind wieder unterschiedliche Variationen denkbar. So könnte der zentrale Sensorknoten direkt am Körper getragen werden, repräsentiert durch eine Art Master-Sensornode, welcher die Daten der einfachen Sensoren sammelt und diese anschließend weiter kommuniziert. Eine andere Möglichkeit, die vor allem für Patientenüberwachung in Krankenhäusern von Interesse sein könnte, besteht darin, dass jeweils pro Zimmer ein zentraler Sensorknoten vorhanden ist. Dieser sammelt die Sensordaten aller Patienten innerhalb eines Raumes und kümmert sich um die Datenübertragung. Ein hybrider Ansatz dieser beiden Möglichkeiten könnte durch einen zusätzlichen leichtgewichtigen zentralen Sensorknoten pro Patient bzw. Nutzer realisiert werden, welcher wiederum die biometrischen Daten pro Person sammelt und diese anschließend an die nächste Hierachieebene in Form eines zentralen Sensorknotens pro Raum weitergibt [1]. Diese Möglichkeit wird an dieser Stelle aus Gründen der Vollständigkeit erwähnt aber nicht weiter vorgestellt, da es noch kein konkretes Forschungsszenario zu diesem Aufbau gibt.

Wiederum sind also, offensichtlich abhängig vom Einsatzgebiet, unterschiedliche Varianten eines WBANs möglich. Die grundlegenden Aufgaben auf dieser Ebene lassen sich aber immer wieder antreffen und stellen sich wie folgt dar [7]:

- Konfiguration, Initialisierung der Sensorknoten
- Synchronisation der Kommunikation
- Datensammlung der physiologischen Sensordaten
- Kommunikation zur dezentralen Zielanwendung

Wie eingehend bereits erwähnt, wird der zentrale Sensorknoten des öfteren als Personal Server [7], Personal Device [3] oder als CCU [1] bezeichnet. Im Rahmen dieses Papers wird der Begriff des Personal Servers als Assoziation für eine Anwendung verwendet, welche auf einem handelsüblichen Gerät wie einem Smartphone, PDA oder einem PC implementiert werden kann. Die CCU stellt hingegen ein vollständig eigenständiges Gerät dar, welches exklusiv für die Aufgaben innerhalb des WBANs verwendet wird. Unabhängig von der Konfiguration und der Tatsache, dass es sich bei dem zentralen Sensorknoten um einen Personal Server oder eine CCU handelt werden die Daten der Sensoren zentral gesammelt und anschließend die Gatewayfunktionalität realisert.

D.h. der zentrale Knoten bedient sich eines beliebigen Kommunikationsnetzwerkes für die eigentliche Datenübertragung hin zur Zielanwendung (vgl. Abbidung 3 2. Hop). Im Prinzip sind alle gängigen Kommmunikationstechnologien denkbar, angefangen bei Festnetztelefonie, Mobilfunk, LAN- oder WLAN-Technik bis hin zu mobilen 3G/4G Datennetzwerken [1]. Für die drahtlose Kommunikation des 1. Hops zwischen den einzelnen Sensorknoten und dem zentralen Knoten werden in der Regel sparsamere Technologien benötigt. So werden für die Kommunikation zwischen CCU bzw. Personal Server und den einfachen Sensorknoten in der Regel drahlose Technologien wie ZigBee, WLAN, ANT+, Bluetooth oder gerade im medizinischen Umfeld spezielle ISM (Industrial, Scientific and Medical) Frequenzbänder verwendet [1].

2.1.3 Backend Anwendung

Auf der höchsten Hierachiestufe gibt es eine sehr große Vielfalt an unterschiedlichen Zielanwendungen. Das Herzstück einer jeden Backend Anwendung wird in den meisten Fällen durch einen Datenbankserver dargestellt, welcher die Gesamtheit der gesammelten Daten enhält. Basierend auf diesem Datenschatz folgen nun weitere anwendungsspezifische Analyseschritte. So können im Gesundheitssektor die gesammelten Daten zielgerichtet ausgewertet, und nach bestimmten Mustern automatisiert vorverarbeitet werden [1]. Außerdem ergeben sich gerade im Bereich des Patienten-Monitoring neue umfassende Möglichkeiten (siehe 3.1). Im

Tabelle 1: Physiologische Signal Eigenschaften [3][1]

Signal	Datenrate	Frequenz
EEG	43 kbps	0,5 - 60 Hz
ECG	288 kbps	0,01 - 250 Hz
EMG	320 kbps	10 - 5000 Hz
Blutdruck	k.A.	0 - 50 Hz
Atemrate	k.A.	0,1 - 10 Hz
Temperatur	120 bps	0 - 0,1 Hz

privaten bzw. kommerziellen Segment muss nicht zwingendermaßen eine dezentrale Datensammlung stattfinden. Es wäre druchaus denkbar, dass die biometrischen Daten einer Zielperson lokal am Personal Server analysiert und über eine Benutzerschnittstelle visualisiert werden. In den meisten aktuell verfügbaren Produkten geht allerdings die Tendez in Richtung zentrale Datenhaltung in der Cloud. Mehr Details zu den unterschiedlichen Anwendungen können dem Abschnitt 3 entnommen werden.

2.2 Herausfoderungen

In einem Wireless Body Area Network existieren verschiedene Herausforderungen, welche vor allem durch den direkten Kontakt zum menschlichen Körper sowie die begrenzte räumliche Ausdehnung geprägt werden. Beim menschlichen Körper handelt es sich um einen sehr sensitiven Bereich. D.h. es müssen bestimmte Richtlinien zum Beispiel bezüglich der maximal zulässigen Sendeleistung eingehalten werden. In diesem Abschnitt werden nun allgemeingültige Anforderungen an ein WBAN diskutiert.

2.2.1 Heterogene Datenbübertragungsraten

Aufgrund der heterogenen biometrischen Signale ergeben sich unterschiedliche Daten- und Abtastraten. Eine kurze Übersicht bezüglich der Datenraten kann Tabelle 1 entnommen werden. Obwohl das Datenaufkommen eines einzelnen Sensors ziemlich gering ist, ergeben sich aus der Summe der Sensoren relativ schnell einige Megabyte an Sensordaten. Außerdem muss das Abfrageintervall bzw. der Einsammelvorgang der Sensordaten vom einfachen Sensorknoten hin zur CCU bzw. Personal Server an das jeweilige Signal angepasst werden [3].

2.2.2 Energiebedarf

Der Energieverbrauch eines WBAN Knotens lässt sich im wesentlichen in drei Bereiche aufteilen:

1. Datenerfassung
2. Datenverarbeitung
3. Kommunikation

Die Datenkommunikation verbraucht hierbei die meiste Energie. Die Menge an verfügbarer Energie ist gerade im Bereich der am Körper getragenen Sensorknoten sehr stark begrenzt. Meistens basiert die Energieversorgung auf Batterien, die im Idealfall relativ klein gehalten werden um einen optimalen Tragekomfort zu gewährleisten. Aus diesem Grund ist es essentiell wichtig zum einen energieeffiziente Hardware einzusetzen und zum anderen Energiesparfunktionen zu implementieren [3].

2.2.3 Benutzerfreundlichkeit

Dieser Punkt addressiert neben den Trägern auch die Betreuer eines WBAN basierenden Systems. Der Tragekomfort muss sichergestellt werden um die Akzeptanz beim Patienten bzw. Nutzer zu gewährleisten. Die Erweiterung des WBANs mit zusätzlicher Sensorik muss ohne das Zutun externer Fachkräfte vom medizinischen Personal durchgeführt werden können. Dies verlangt ein gewisses Maß an Selbstorganisation und automatisierter Rekonfiguration des Systems [3].

2.2.4 Ausfallsicherheit/Verfügbarkeit

Die Verlässlichkeit bzw. ständige Verfügbarkeit eines WBAN Systems ist eines der wichtigsten Ziele beim Design solcher Anwendungen. Gerade im medizinischen Umfeld spielt dieser Punkt eine sehr wichtige Rolle. Es muss garantiert werden, dass die zu überwachenden Vitalfunktionen im geforderten zeitlichen Intervall korrekt erfasst und übermittelt werden. Daher werden oftmals „Quality of Service“ Verfahren benötigt um kritische Signalen oder bedürftigeren Patienten eine höhre Priorität einzuräumen [3].

2.2.5 Sicherheit

Da es sich bei dem anfallenden Datenaufkommen um hochgradig persönliche Daten handelt, müssen diese mit äußerster Vorsicht behandelt werden. Vor allem die Datenübertragung zur Remoteanwendung muss daher zwingend verschlüsselt erfolgen. Eine verschlüsselte Kommunikation des 1. Hops muss aufgrund der begrenzten Ressourcen gut ausbalanciert werden. So existieren für Sensornetzwerke spezifische Anforderungen bzüglich der Verschlüsselungsverfahren und einsetzbaren Schlüsselungsstärke[4]. Wegen der geringen Reichweite der Körpersensorik muss eine verschlüsselte Datenübetragung im 1. Hop nicht immer zwingendermaßen realisiert werden [3].

2.2.6 Sensorknoten Merkmale

Folgende Merkmale sind für Sensorknoten eines Wireless Body Area Networks und bei Sensornetzwerken im allgemeinen gültig [1]:

- Geringe Rechenleistung
- Begrenzte Kommunikationsbandbreite
- Niedrige Sendeleistung
- Energieeffizienz
- Lange Laufzeit
- Skalierbarkeit

3. EINSATZSZENARIEN

Es bietet sich eine Fülle von Anwendungsmöglichkeiten für WBAN basierende Systeme immer mit dem Fokus auf der Erfassung von Körperfunktionen und der anschließenden Verarbeitung der erhobenen Daten. Im wesentlichen konzentrieren sich die Einsatzgebiete dabei momentan hauptsächlich auf den medizinischen und den kommerziellen Sektor. Aber auch für diverse Forschungbereiche bieten diese Art von Systeme neue Möglichkeiten und Methoden. So können WBAN Systeme zur Verhaltens- und Bewegungsanlyse an

Menschen, aber auch bei Tieren eingesetzt werden [2]. Ein WBAN System bietet im Vergleich zu herkömmlichen BAN Systemen zwei wesentliche Vorteile, welche diese besonders für die bereits erwähnten Sektoren qualifizieren. Zum einen wird die Mobilität des Trägers durch den Einsatz portabler Sensoren deutlichen weniger eingeschränkt und zweitens kann der Monitoring-Prozess ortsunabhängig druchgeführt werden [1]. In diesem Abschnitt werden vor allem der medizinische sowie der kommerzielle Sektor genauer vorgestellt und durch sektorspezifische Beispiele illustriert.

3.1 Health-Care Sektor

Im Gesundheitssektor werden vermehrt innovative ICT (Information- and Communicationtechnology) Systeme eingesetzt um bestehende Health-Care Anwendungen zu optimieren und deren erhobene Daten effizient zu verarbeiten [1]. Ein ICT System wie etwa ein WBAN bietet Gesundheitsanwendungen, wie zum Beispiel Patienten-Monitoring, nicht nur für Patienten innerhalb eines Krankenhauses, sondern ermöglicht es auch bestimmte Dienste von Zuhause oder vom Arbeitsplatz aus in Anspruch zu nehmen [7]. Auf diese Weise wird die Lebensqualität des Patienten deutlich verbessert und es können zusätlich Kosten eingespart werden. Außerdem ergbit sich durch den Einsatz der WBAN Technik die Möglichkeit der Langzeitüberwachung eines Patienten [5]. So könnte zum Beispiel ein Früherkennungssystem für epileptische Anfälle auf der Basis eines langzeit EEG bei Epilepsie Patienten eingesetzte werden um den Nutzer frühzeitig über einen bevorstehenden Anfall zu informieren. Desweitern stellen die erfassten Sensorwerte vor, während und nach einem Anfall für die Erforschung bzw. Bekämpfung dieser Krankheit einen großen Wert dar. ICT Systeme sind bereits seit

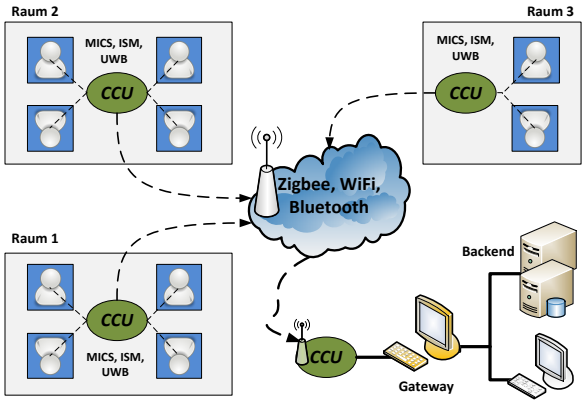


Abbildung 4: Muli-Patienten Monitoring WBAN System [1]

längerem in verschiedenen Anwendungen im Einsatz. Allerdings zumeist in Form der traditionellen kabelgebundenen Variante mit den Nachteilen der Ortsabhängigkeit und der fehlenden Mobilität für den Patienten. Daher besteht gerade im Bereich der Patientenüberwachung großes Potenzial für WBAN basierende Systeme. Aktuelle, sich im Einsatz befindende Monitoring Systeme basieren auf kabelgebundenen Lösungen mit sperriger Elektronik, welche die Mobilität eines Patienten sehr stark einschränken [1]. Daher werden WBANs benötigt welche individuelle, schlanke Sensoren kabellos miteinander verbinden, die Mobilität und Ortsunabhängigkeit des Patienten erhalten und auf diese Weise seine Lebensqualität verbessern.

Anwendung: Multi-Patienten Monitoring
Eines der potenziellen Hauptanwendungsgebiete eines WBAN basierenden Systems liegt im Bereich der Kranken- und Gesundheitspflege[1]. Ein möglicher Aufbau einer solchen Anwendung im Krankenhausumfeld wird in Abbildung 4 dargestellt. Dabei handelt es sich um ein Multi-Patienten Monitoring System, in welchem ein große Anzahl von Patienten gleichzeitg und kontinuierlich überwacht werden soll. In Abbildung 4 wird eine Ebene eines Krankenhauses bestehend aus mehreren Krankenzimmern dargestellt. Für eine großflächige WBAN Anwendung wie diese, werden mehrere unterschiedliche zentrale Sensorknoten benötigt um eine dauerhafte und vor allem zuverlässige Patientenüberwachung zu ermöglichen. So existiert pro Zimmer eine CCU, welche jeweils die Sensordaten der sich in diesem Raum befindlichen Patienten erfasst. Die Anwesenheit von zusätzlichen CCUs, zum Beispiel auch auf den Fluren, ist ein weiterer wichtiger Bestandteil um den Personen ein gewisses Maß an Bewegungsfreiheit innerhalb des Krankenhauses zu ermöglichen.

Abbildung 4 zeigt ein komplettes WBAN, das im Rahmen eines Forschungsprojektes an der Universität von Newcastle, Australien entwickelt wurde[1]. Dieses Netzwerk basiert auf unterschiedlichen Kommunikationsverbindungen und Frequenzen um Interferenzen so weit wie möglich zu eliminieren. Für den 1. Hop zwischen Sensorknoten und CCU wird ein ISM (Industial, Scientific and MeciaD) Band wie MICS (Medical Implant Communication Service), WMTS (Wireless Medical Telemetry Systems) oder ein 433 ISM Frequenzband eingesetzt. Die einzelnen CCUs kommunizieren über existierende drahtlose Standards wie ZigBee, Bluetooth oder WiFi mit einer CCU Basis Station, welcher via USB/RS232 an einem lokalen PC angeschlossen ist. Der PC in Kombination mit der Basis Station stellt den Gateway Knoten dar, der sich um die Anbindung der Zielanwendung über eine beliebige TCP/IP Verbindung kümmert.

3.2 Kommerzielle Anwendungen

Ein weiteres Anwendungsszenario für WBANs, das sich in den letzten Jahren immer größere Beliebtheit erfreut, liegt im Bereich von sportlichen Aktivitäten, bei welchen die Nutzer eines solchen Systems ihre persönlichen Vitalfunktionen protokollieren. Athleten unterschiedlichster Sportarten oder Hobbysportler können mit Hilfe solcher Geräte Leistungsdaten erfassen, analysieren und ihr Training zielführend optimieren. Vor allem im Bereich der Lauf- und Ausdauersportarten erfreuen sich solche Systeme großer Beliebtheit. Außerdem ergeben sich mit zusätzlicher Intelligenz bzw. steigender Rechenleistung in der zentralen Kontrolleinheit weitere Möglichkeiten. So existieren bereits Anwendungen, welche dem Sportler ein interaktives, virtuelles Coaching während einer Trainingseinheit bieten.

Anwendung: Adidas micoach
Adidas bietet unter der Marke „micoach“ eine breite Palette an Aufzeichnungsgeräten und Software, welche die Erfassung und Analyse von Trainingsergebnissen ermöglichen.

doi: 10.2313/NET-2012-08-2_07

Der weitere Verlauf diese Abschnitts basiert, falls nicht anders angegeben, auf Informationen der Adidas Homepage [10]. Das erste Gerät dieser Serie stellt das Fitnesshandy Samsung SGH-F110 micoach [11] dar, das bereits 2008 erschienen ist. Dieses Gerät wurde in Kooperation von Samsung und Adidas entwickelt und bietet neben der klassischen Handyfunktionalität einen Personal Trainer. Das SGH-F110 lässt sich per Bluetooth mit einem Herzfrequenzmesser sowie einem Schrittzähler vernetzen und stellt somit den zentralen Sensorknoten eines WBANs dar. Die erfassten Vitalfunktionen werden vom Handy überwacht, mit nützlichen Trainingsinformationen aufbereitet und mittels Kopfhörer an den Sportler weitergegeben. Mittlerweile hat sich das Sortiment der „micoach“ Produktfamilie vergrößert. So existieren, neben einer micoach Smartphone Anwendung für alle gängigen Betriebssysteme auch komplexere Produkte. Ein Ausschnitt aus dem Produktsortiment kann Abbildung 5 entnommen werden. Aktuell hat Adidas mit SPEEDCELL,

Abbildung 5: Adidas micoach Übersicht

dem Herzfrequenz-Messer sowie Schrittzähler insgesamt drei Sensoren im Sortiment. Die sogenannte SPEEDCELL repräsentiert einen Trackingsensor, welcher die Aufzeichnung von Leistungsdaten wie Maximalgeschwindigkeit, zurückgelegte Distanz und Anzahl der Sprints ermöglicht. Die SPEEDCELL wird direkt am Schuh, oder in einer Art Zwischensohle im Schuh angebracht. Dies stellt an sich noch keine Neuerung dar, da bereits seit längerem Laufschuhe mit Schrittzählern ausgestattet werden. Allerdings bietet Adidas mit der micoach SPEEDCELL erstmals einen Sensor, welcher sich für Sportarten mit dynamischeren Bewegungsabläufen, wie etwa Fußball oder Basketball eignet. Dafür besitzen die Sportschuhe eine Ausbuchtung in der Sohle für den Einsatz des Trackingsensors. Die SPEEDCELL bietet Speicherkapazität für acht Stunden Trainingsdaten aus welchen folgende Werte extrahiert werden [9]:

- Zeit
- Durchschnittsgeschwindigkeit
- Höchstgeschwindigkeit
- Zurückgelegte Distanz
- Anzahl Sprints
- Intensität

Wird der Trackingsensor zur einfachen Protokollierung einer Sporteinheit verwendet, so erfolgt die Verbindung mit einem zentralen Sensorknoten bzw. Personal Server erst im Anschluss an eine Einheit. Prinzipiell besteht die Möglichkeit auch während des Sports die Daten direkt zu verarbeiten, allerdings birgt dies gerade für Mannschaftssportarten zusätliche Schwierigkeiten da ein zentraler Knoten immer mitgeführt werden müsste. Dies liegt vor allem an der Beschaffenheit der verfügbaren Personal Server. Die Rolle des zentralen Senorknotens kann je nach Sensortyp und sportlicher Aktivität von einem anderen Gerät ausgeführt werden. Prinzipiell besteht die Nutzungsmöglichkeit eines iPhones oder iPods in Verbindung mit einem sogenannten micoach Connector. Dieses Modul wird an der Ladeschnittstelle des jeweiligen Endgerätes angebracht und realisiert die drahtlose Kommunikation mit den Sensorknoten. Desweiteren existiert ein USB basierender Connector für PC oder Mac, welcher für die Auswertung einer Trainingseinheit böntigt wird, um die Daten via ANT+ aus den Sensoren auszulesen. Eine Alternative zu den bisher genannten zentralen Sensorknoten stellt der sogenannte micoach PACER dar, welcher die Daten mehrer Sensoren in Echtzeit kombiniert, auswertet und an den Sportler via Köpfhörer weitergibt. Ein typisches Anwendungsszenario für Ausdauersportler stellt die Kombination aus Herzfrequenzmesser, Schrittzähler und PACER dar. Der Nutzer wird während einer Sporteinheit von einem virtuellen Trainer begleitet, welcher seine Vitalfunktionen überwacht und diese interaktiv an den Sportler weitergibt.

Sind die Sensorinformationen auf einem der Personal Server angelangt, so bieten sich verschiedene Möglichkeiten. Die Daten können, falls der zentrale Knoten im Sinne eines WBANs immer mitgeführt wird in Echtzeit verarbeitet und dem Nutzer während des Trainings über die Coaching Funktion mitgeteilt werden. Außerdem können die Daten, wie im Falle der SpeedCell, während des Sports erfasst und im Anschluss analysiert werden. Für die Auswertung der Daten wird ein micoach Konto auf der bereits genannten Webseite benötigt. Mit diesem Konto werden die Sensordaten mittels der Synchronisationssoftware micoach Manager abgeglichen. Eine lokale Datenauswertung ist zum jetzigen Zeitpunkt nicht verfügbar. Die Tendenz geht auch in diesem Bereich in Richtung zentrale Datenhaltung. So werden neben diversen Funktionen im Bereich der Trainingsplanung, Datenauswertung und Datenvisualisierung auch Funktionen geboten, welche man aus dem Bereich der sozialen Netze kennt. So können zum Beispiel Trainingsdaten direkt bei Facebook veröffentlicht oder intern mit anderen micoach Nutzern verglichen werden.

Eine ähnliche Produktserie wird vom Sportartikel Hersteller Nike unter dem Markenname „Nike+“ angeboten, wobei der Schwerpunkt im Laufsegement liegt [14]. Mittlerweile bietet aber auch Nike Systeme für dynamischere Sportarten wie zum Beispiel Basketball an. Hierbei besteht das Produkt, ähnlich wie bei Adidas, aus einem speziellen Sportschuh, in dessen Sohle ein SPEEDCELL ähnlicher Sensor integriert ist. Die Aufgabe des zentralen Sensorkonten kann auch hier wieder von iPhone oder iPod übernommen wer-

doi: 10.2313/NET-2012-08-2_07

den. Die abschließende Datenauswertung und Aufbereitung geschieht über die Nike+ Webseite.

Abschließend muss an dieser Stelle mit dem Navigationssysteme Hersteller Garmin noch ein weiterer Vertreter WBAN basierender Analyse Systeme aufgeführt werden. Unter dem Namen „GARMIN connect“ bietet das amerikanische Unternehmen eine Webplattform zur zentralen Datenanalyse der Garmin eigenen Sensor-Geräte an. Hierbei wurde dieses System für die Trainings- und Outdoorgeräte in Form von Fahrradcomputern, Läuferuhren oder ähnlichen Produkten entwickelt. Wie bei den bereits genannten Produkten von Nike und Adidas könnnen die gesammelten Daten der Geräte via Funkschnittstelle oder USB an einen lokalen PC übertragen und von dort aus mit der Webseite von Garmin synchronisiert und analysiert werden [15].

3.3 Gegenüberstellung

Im Verlauf dieses Kapitels wurden die beiden Hauptanwendungsgebiete von WBANs vorgestellt und anhand eines Beispiels veranschaulicht. Stellt man einen Vergleich zwischen medizinischen und kommerziellen Anwendungen an, so ergeben sich signifikante Unterschiede in den Anforderungen an ein WBAN System. Bedenkt man die Herausforderungen aus Abschnitt 2.2 so ergeben sich für die folgenden Punkte anwendungsspezifische Unterschiede [4][3].

- Sicherheit
- Verlässlichkeit
- Benutzerfreundlichkeit

Diese lassen sich wie folgt gruppieren. Die Anforderungen Sicherheit, Privatsphäre und vor allem Verlässlichkeit stellen die wohl wichtigsten Kriterien im medizinischen Umfeld dar. Auch existieren bei bestimmten Healthcare Anwendugen wie der Multi-Patienten Monitoring Anwendung aus Abschnitt 3.1 besondere Anforderungen an die Datenübertragungsraten aufgrund der Vielzahl aktiver Netzwerkteilnehmer. Was wiederum unter dem wohl wichtigsten Punkt der Verlässlichkeit eingeordnet werden kann. Gerade im medizinischen Bereich muss eine hohe Ausfallsicherheit gewährleistet werden [1]. Man bedenke ein Monitoring System, welches kritische Vitalfunktionen mittels EKG überwacht. Eine Störung oder Fehlfunktion könnte drastische Folgen nach sich ziehen, indem zum Beispiel aufgrund eines Ausfalls des Systems ein Herzstillstand eines Patienten nicht oder verspätet erkannt wird. Auch im Hinblick auf implantierte Sensoren oder Aktuatoren ist eine höchstmaß an Verfügbarkeit bzw. Verlässlichkeit unabdingbar, da jeder Austausch der Sensorik eine Operation erfordert.

Dem gegenüber steht die Nutzbarkeit bzw. Benutzerfreundlichkeit im Sektor der kommerziellen WBAN Systeme [4]. Der Tragekomfort stellt einen wichtigen Unterpunkt der Benutzerfreundlichkeit dar, welcher in beiden Anwendungsgebieten von großer Bedeutung ist. Vor allem bei Langzeitanwendungen bestehen besondere Anforderungen an die eingesetzten Sensoren [5]. So muss zum Beispiel eine sehr gute Hautverträglichkeit gewährleistet sein. Die Benutzbarkeit einer WBAN Anwendung betrifft in diesem Sinne die Interaktion mit dem Endanwender. In diesem Fall liegen größere Anforderungen auf Seiten der kommerziellen Anwendungen. Die Usability und nicht zu vergessen das Design einer Applikation entscheiden über den Erfolg des Produktes.

Sicherheitstechnische Probleme betreffen im wesentlichen beide Systeme, wobei im medizinischen Bereich der Anteil an sensitiven Vitalwerten deutlich höher liegt [4]. Aus diesem Grund müssen die vertraulichen Daten durch anwendungsspezifisch Maßnahmen entsprechend geschützt werden.

4. ZUSAMMENFASSUNG

Dieses Paper gibt einen kurzen Überblick über den Aufbau, die Strukur und mögliche Anwendungsgebiete von Wireless Body Area Networks. Neben der Architektur wurden die Anforderungen, Herausforderungen und Merkmale eines WBAN Systems diskutiert.

WBAN Systeme mit physiologischen Sensoren stellen eine sehr nützliche Technologie mit großem Potenzial im Bereich der Monitoring Anwendungen dar. Im kommerziellen Sektor existieren bereits seit längerem WBAN basierende Anwendungen, welche den Weg zum Endanwender gefunden haben. Repräsentanten solcher Produkte sind die Systeme „Nike+“, „micoach“ und „GARMIN connect“ von den Sportartikelherstellern Nike und Adidas sowie dem in der Navigationsbranche angesiedelten Unternehmen Garmin. Gerade aber für den Einsatz im medizinischen Umfeld bieten WBAN Systeme ideale Voraussetzungen um sich in naher Zukunft dauerhaft zu etablieren. Durch den technologischen Fortschritt werden schon bald drahtlose Sensoren in Form von einfachen Pflastern bzw. Patches entwickelt werden, welche in ein WBAN integriert werden können[3]. Auf diese Weise entwickeln sich WBAN basierende Systeme, welche den Einzug in den medizinischen Alltag schaffen können und somit die Lebensqualtität von Patienten signifikant verbessern.

5. LITERATUR

[1] Jamil. Y. Khan and Mehmet R. Yuce (2010): *Wireless Body Area Network (WBAN) for Medical Applications*, New Developments in Biomedical Engineering, Domenico Campolo (Ed.), ISBN: 978-953-7619-57-2, InTech, `http://www.intechopen.com/books/new-developments-in-biomedical-engineering/wireless-body-area-network-wban-for-medical-applications`

[2] Johannes Thiele, Jo Agila Bitsch Link, Okuary Osechas, Hanspeter Mallot and Klaus Wehrle (2010): *Dynamic Wireless Sensor Networks for Animal Behavior Research*, New Developments in Biomedical Engineering, Domenico Campolo (Ed.), ISBN: 978-953-7619-57-2, InTech, `http://www.intechopen.com/books/new-developments-in-biomedical-engineering/dynamic-wireless-sensor-networks-for-animal-behavior-research`

[3] Benoît Latré, Bart Braem, Ingrid Moerman, Chris Blondia, and Piet Demeester (2011): *A survey on wireless body area networks*, Wirel. Netw. 17, 1 (January 2011), 1-18. DOI=10.1007/s11276-010-0252-4, `http://dx.doi.org/10.1007/s11276-010-0252-4`

[4] Dejan Raskovic, Thomas Martin and Emil Jovanov

(2003): *Medical Monitoring Applications for Wearable Computing*, The Computer Journal (2004) 47 (4): 495-504. doi: 10.1093/comjnl/47.4.495

[5] Hao Qu, Jean Gotman (1997): *A Patient-Specific Algorithm for the Detection of Seizure Onset in Long-Term EEG Monitoring: Possible Use as a Warning Device*, Biomedical Engineering 44, 2, 115-122.

[6] Jovanov E, Milenkovic A, Otto C, De Groen P, Johnson B, Warren S, Taibi G (2005): *A WBAN System for Ambulatory Monitoring of Physical Activity and Health Status: Applications and Challenges.*, Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005, 3810-3813.

[7] Emil Jovanov, Aleksandar Milenkovic, Chris Otto and Piet C de Groen (2005): *A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation*, Journal of NeuroEngineering and Rehabilitation 2005, 2:6 doi:10.1186/1743-0003-2-6, `http://www.biomedcentral.com/1743-0003/2/6`

[8] IEEE Working Group 802.15.6 (2012): *802.15.6 IEEE Standard for Local and metropolitan area networks - Part 15.6: Wireless Body Area Networks*, ISBN 978-0-7381-7206-4

[9] *SPEEDCELL User Manual* aufgerufen am 20. Juni 2012, `http://www.adidas.com/com/micoach/Multimedia/com/PDF/miCoach%20SPEED_CELL%20User%20Manual_en.pdf`

[10] *Adidas micoach*, aufgerufen am 20. Juni 2012, `http://www.adidas.com/de/micoach/`

[11] *Samsung SGH-F110 micoach*, aufgerufen am 22. Juni 2012, `http://www.samsung.com/de/consumer/mobile-device/mobilephones/feature-mobile-phones/SGH-F110DAAXEG-spec`

[12] *University of Rochester, Center for Future Health*, aufgerufen am 30. Juni 2012 `http://www.rochester.edu/pr/Review/V62N2/feature2.html`

[13] T. Falck, J. Espina, J. P. Ebert, and D. Dietterle(2006): *BASUMA - the sixth sense for chronically ill patients*, Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on, Cambridge, MA, USA, 3-5 April 2006, 57-60.

[14] *Nike+*, aufgerufen am 20. Juni 2012, `http://nikeplus.nike.com/plus/`

[15] *GARMIN connect* , aufgerufen am 10. August 2012, `http://connect.garmin.com/`

[16] *ANT Message Protocol and Usage*, aufgerufen am 10. August 2012, `http://www.thisisant.com/images/Resources/PDF/1204662412_ant%20message%20protocol%20and%20usage.pdf`

[17] *Netzwelt*, aufgerufen am 29. Juni 2012, `http://www.netzwelt.de/news/92661-adizero-f50-micoach-speed-cell-sensor-test.html`

Umweltmonitoring mit SensorScope

Stefan Stöckl
Betreuerin: Corinna Schmitt
Seminar: Sensorknoten - Betrieb, Netze & Anwendungen SS 2012
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitekturen
Fakultät für Informatik, Technische Universität München
Email: stefan-stoeckl@mytum.de

KURZFASSUNG

Gerade in Zeiten der Klimaerwärmung versuchen immer mehr Wissenschaftler mit Hilfe von technischen Hilfsmitteln Umweltphänomene zu erklären. Dazu werden Messwerte an verschiedenen Orten und über einen längeren Zeitraum eingeholt. Dieses vorgehen bezeichnet man als Umweltmonitoring. Mit den gesammelten Daten wird dann anhand von Berechnungen versucht, Rückschlüsse auf die untersuchten Umweltphänomene zu ziehen. Ein System das zum Umweltmonitoring eingesetzt wird ist SensorScope. Es dient der Untersuchung von Klimabedingungen mit deren Hilfe Ereignisse, wie beispielsweise Schlammlawinen, vorhergesagt werden können. Seine Funktionsweise wird in dieser Arbeit ausführlich beschrieben. Anschließend werden noch zwei weitere Systeme kurz erklärt und es wird ein Vergleich hergestellt.

Schlüsselworte

Umweltmonitoring, Umweltbeobachtung, Environmental Monitoring, SensorScope, Wireless Sensor Networks (WSN)

1. EINLEITUNG

Unter Umweltmonitoring versteht man die Erhebung und Auswertung von Daten über Umweltbedingungen. Zuerst werden hierbei Daten in Form von Messwerten erhoben und gesammelt. Anschließend wird versucht mit wissenschaftlichen Verfahren einen Zusammenhang zwischen Messwerten und Umweltphänomenen herzustellen, damit man sich diese erklären und vorhersagen kann. Die erhobenen Daten kommen vor allem aus den Bereichen Chemie, Physik und Biologie [8]. Bei der Auswertung der Daten kommen vor Allem Verfahren aus der Mathematik, insbesondere aus der Statistik, zum Einsatz. Hierbei versucht man herauszufinden, welche Umweltfaktoren einen Einfluss auf die untersuchten Phänomene haben. Typische Anwendungsgebiete von Umweltmonitoring sind die Beobachtung von Klima, Luftqualität, Wasserqualität, Natur-Phänomenen (wie z.B. Vulkane), Bakterien und Radioaktiver Strahlung.

Der Fokus dieser Arbeit liegt auf der Verwendung von SensorScope, einem Sensornetz (Wireless Sensor Network, WSN) das zur Erfassung von Daten im Bereich der Klimabeobachtung eingesetzt wird. Der Einsatz des WSNs bringt viele Vorteile im Vergleich zu herkömmlichen Verfahren. Zu erwähnen wäre hier, dass die Messwerte nicht manuell eingeholt werden müssen, sondern automatisch erfasst und an eine zentrale Datenbank gesendet werden, in der sie dann ausgewertet werden. Weiterhin ist es durch verteilte Knoten möglich, im Vergleich zu einer zentralen Messstation, eine gute räumliche Abdeckung des beobachteten Areals zu erreichen. Außerdem können durch den Verzicht auf teure Messstationen noch Kosten gespart werden.

SensorScope kommt hauptsächlich in Gebieten mit schwierigen Umweltbedingungen, wie zum Beispiel auf einem Gletscher, zum Einsatz. Daraus ergaben sich für die Entwickler des Systems besondere Herausforderungen. So muss ein reibungsloser Einsatz ohne Anschluss an ein Stromnetz sichergestellt werden. Außerdem muss gewährleistet werden, dass der Sensor die Umwelteinflüsse, wie Feuchtigkeit und extreme Temperaturen, ohne Schaden übersteht und weiterhin fehlerfrei funktioniert.

In Kapitel 2 wird SensorScope genau beschrieben. Hier wird in Abschnitt 2.1 der Aufbau einer Sensorstation, in Abschnitt 2.2 die Netzwerkfunktionalität und in Abschnitt 2.3 Anwendungen von SensorScope genau erläutert. Anschließend werden in Kapitel 3 weitere Umweltmonitoring-Systeme wie Macroscope (Abschnitt 3.1) und PermaSense (Abschnitt 3.2) kurz erklärt. Zuletzt werden diese Systeme noch in Kapitel 4 miteinander verglichen und Unterschiede und Gemeinsamkeiten hervorgehoben.

2. SENSORSCOPE

In diesem Kapitel wird SensorScope, ein WSN, das zum Umweltmonitoring verwendet wird, beschrieben. Es basiert größtenteils auf [7].

Sensorscope ist ein WSN, das für Umweltmonitoring verwendet wird. Es wurde in Zusammenarbeit von Umweltforschern, Hardware- und Software-Entwicklern der EPFL (École polytechnique fédérale de Lausanne; dt.: Eidgenössische Technische Hochschule Lausanne) entwickelt [3].

Abschnitt 2.1 geht auf die einzelnen Sensorstationen ein. Anschließend beschreibt Abschnitt 2.2 wie diese Stationen zu einem Netzwerk zusammengefügt werden und miteinander kommunizieren. Hierbei sollen auch mögliche Schwierigkeiten aufgezeigt werden. In Abschnitt 2.3 werden dann Anwendungen, die mit Hilfe von SensorScope umgesetzt wurden, erläutert und deren Ergebnisse präsentiert.

2.1 Sensorstation

Eine SensorScope-Messstation besteht aus einer Solarzelle, Sensoren und einer hermetisch abgeriegelten Box. In dieser Box befindet sich die Batterien und das Herzstück, der Sensorknoten. All dies ist auf eine 2 Meter hohe Stange montiert, da durch zu geringen Abstand zum Boden die Klimabedingungen verfälscht werden. Abbildung 1 zeigt eine Sensorstation. Solch eine Station kostet rund €1500.

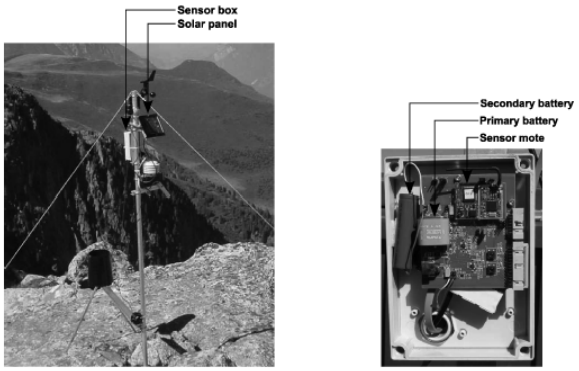


Abbildung 1: Eine SensorScope Station (links) und die Box die den Microcontroller und die Batterien enthält (rechts) [7]

2.1.1 Stromversorgung

Die Stromversorgung von SensorScope besteht aus drei Komponenten und basiert auf [9].
Solarzelle: Zur Energiegewinnung wird eine MSX-01F Solarzelle von BP Solar verwendet. Laut Datenblatt ist sie 162x140 mm groß, hat eine zu erwartende Haltbarkeit von 20 Jahren und hat bei direkter Sonneneinstrahlung eine Ausgangsspannung von mindestens 1W [5].
Primäre Batterie: Als primäre Batterie wird, eine wiederaufladbare 150mAh NiMH Batterie verwendet, welche gegenüber dem vorgeschlagenen Superkondensators den Vorteil hat, dass sie billiger ist und auch noch mehr Energie speichern kann. Sie hält bis zu 5 Tage und befindet sich in der hermetisch abgeriegelten Box. Sie wird mit Hilfe der Solarzelle wiederaufgeladen und dient als Hauptstromquelle für den Sensorknoten.
Sekundäre Batterie: Als zweite Batterie wird eine LiIo-Baterie mit einer Kapazität von 2200mAh verwendet. Sie dient als Backup, falls die erste Batterie leer wird.
Mit Hilfe dieser drei Komponenten kann eine autonome Stromversorgung sichergestellt werden. In Abbildung 1 sieht man, wie die beiden Batterien in der Sensorstation untergebracht sind.
In zukünftigen Generationen von SensorScope wird vermutlich nur noch eine Batterie verbaut sein [7]. Diese Überlegung wird in Abschnitt 2.3.2 genauer beschrieben.

2.1.2 Sensoren

Jeder Knoten verfügen über sieben Sensoren, mit denen es möglich ist, bis zu neun umweltbezogene Größen zu messen. Tabelle 1 fasst die hardwarespezifischen Informationen zu den Sensoren zuammen. Durch eine durchgeführten Erprobungen wurde jedoch festgestellt, dass es sehr nützlich wäre, noch eine Kamera zur Verfügung zu haben, um Bilder der Umgebung aufnehmen zu können (siehe Abschnitt 2.3.2).

Tabelle 1: Von SensorScope verwendete Sensoren [7]

Sensor	Messgröße	Range	Präzision
Sensirion SHT75	Luftfeuchtigkeit	0-100%	± 2%
Sensirion SHT75	Lufttemperatur	-20-60°C	±0,3°C
Davis Rain Collector	Niederschlagsintensität	0-∞mm	± 1 mm
Decagon EC-5	Bodenfeuchtigkeit	0-100%	±0,1%
Davis Solar Radiation	Sonneneinstrahlung	0 - 1800 W/m^2	± 90 W/m^2
Zytemp TN901	Oberflächentemperatur	-33-220°C	± 0,6°C
Irometer Watermark	Bodenfeuchte	-200-0kPa	unbekannt
Davis Anemomenter	Windrichtung	0-360°	± 7°
Davis Anemomenter	Windgeschwindigkeit	1,5-79m/s	±1,5m/s

2.1.3 Kerneinheit

Als Kern von SensorScope wird eine TinyNode-Einheit verwendet. Sie besteht aus einem TI MSP430 Microcontroller und einer Xemics XE1205 Funkeinheit [1]. Es handelt sich um einen 16-Bit Microcontroller der aufgrund seindes geringen Energieverbrauchs für batteriebetriebene Geräte ausgelegt ist. Weitere Informationen kann man dem Datenblatt unter [12] entnehmen. Die Kerneinheit ist in Abbildung 1 zu sehen.

2.2 Netzwerk

In diesem Abschnitt wird die Netzwerkfunktionalität von SensorScope beschrieben. Abbildung 2 zeigt den Typischen Aufbau eines SensorScope Netzwerks. Die Sensorknoten sind über Funk miteinander verbunden. Einer von ihnen verfügt über GPRS, so dass er mit einem Datenbank-Server kommunizieren kann um diesem Daten aus dem Netzwerk zu senden. Dieser wiederum ist mit einem Webserver verbunden, der die Daten aufbereitet und den Benutzern zur Verfügung stellt.

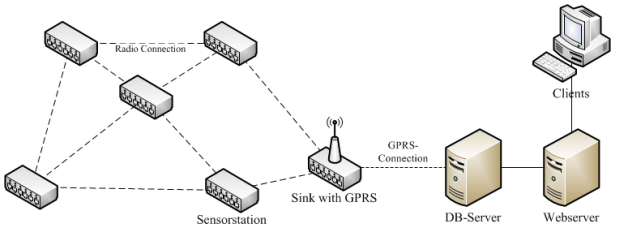


Abbildung 2: Typischer Aufbau eines SensorScope Netzwerks

Im weiteren Verlauf wird erklärt, wie das Netzwerkprotokoll aufgebaut ist, welche Herausforderungen beim Design des Netzwerkes es gab und wie diese gelöst wurden. Im Einzelnen wird Nachbarschaftsmanagement (2.2.2), Synchronisation
(2.2.3), Energiemanagement (2.2.4) und Routing (2.2.5) genauer erläutert.

2.2.1 Protokollimplementierung

Bei der Entwicklung der Netzwerkarchitektur von SensorScope wurde darauf geachtet, dass alles möglichst einfach gestaltet wird, um das ganze robust zu gestalten. Hierbei wurde ein eigener Protokoll-Stack entwickelt, der in eine TinyOS-Nachricht eingebettet ist (vgl. [4]). Dass der SensorScope-Header in den Payload der TinyOS-Nachricht und nicht in deren Header geschrieben wird hat den Vorteil, dass man an SensorScope nichts ändern muss, falls sich in neuen TinyOS-Distributionen etwas ändert.

Der Payload der TinyOS-Nachrichten beträgt 28 Bytes. Davon werden 4 Bytes als Header für Sensor-Scope und 24 Bytes als Daten genutzt. Der Header besteht aus folgenden vier Feldern (je 1 Byte)[4]:
Die **Sender ID** gibt an, von wem eine Nachricht ursprünglich gesendet wurde. In dem Feld **Cost to Sink** (dt. Kosten

zur Senke) steht die Hop-Distanz zur Senke.
Der **Hop-Count** wird von der Transportschicht geschrieben. Für neue Pakete wird er auf 0 gesetzt, für Pakete die weitergeleitet werden wird er um 1 erhöht. Der Hop-Count ist nicht obligatorisch und wird nur für statistische Auswertungen benutzt.
Die **Sequenznummer** wird ebenfalls von der Transportschicht geschrieben. Sie wird jedes mal erhöht, wenn das Senden eines Pakets erfolgreich war. Anderenfalls wird das Paket nochmal mit der gleichen Sequenznummer geschickt. Dies dient der Auswertung der Link-Qualität.
In Abbildung 3 wird der Netzwerkstack von SensorScope gezeigt. Er besteht aus vier Schichten, die im Folgenden kurz charakterisiert werden.

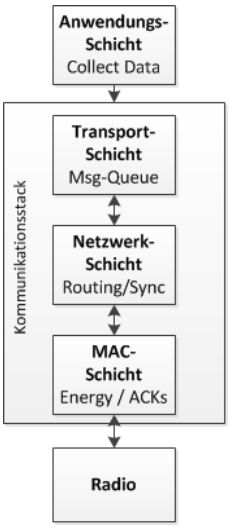


Abbildung 3: Netzwerkstack von SensorScope

Die **Anwendungsschicht** ist für das Sammeln von Daten zuständig. Sie fragt sowohl Sensordaten, als auch den Ladestand der Batterien ab.
Die **Transportschicht** abstrahiert den Rest des Kommunikationsstacks für die Anwendungsschicht und lässt dem Benutzer nur zwei Möglichkeiten, welche Befehle er versenden kann:
Datenpakete enthalten Daten, die zur Senke des Netzwerks geschickt werden sollen. Die Daten die gesendet werden sind gemessene Sensorwerte und Batteriestände.
Kontrollpakete richten sich an einen oder alle Nachbarn des Knotens. Sie enthalten Beacons (siehe Abschnitt 2.2.2) oder Synchronisationspakete (siehe Abschnitt 2.2.3). Die Synchronisation erfolgt nur lokal. Dies hat den Vorteil, dass das Netzwerk dezentralisiert bleibt.
In der Transportschicht werden die vom Benutzer übergenen Daten in Pakete gepackt und in eine Warteschlange (engl. Queue) eingereiht. In diese Warteschlange kommen auch empfangene Nachrichten, die weitergeleitet werden müssen. Kontrollpakete erhalten in der Warteschlange höhere Priorität gegenüber Datenpaketen. Das wird gemacht, da sie höhere zeitliche Anforderungen haben. Ausserdem werden hier die beiden Felder Hop-Count und Sequenznummer des Pakets geschrieben. Eine Überlastungskontrolle (engl. Congestion Control) ist aufgrund des geringen Datenaufkommens nicht nötig.
Die **Netzwerkschicht** ist für das Routing der Pakete zuständig. Sie routet Datenpakete in Richtung Senke. Kontrollpakete werden nicht weitergeleitet, da es nur lokale Broadcasts zu den direkten Nachbarn gibt und keine globalen. Wie genau bei SensorScope der nächste Hop ausgewählt wird, kann Abschnitt 2.2.5 entnommen werden. Die Netzwerkschicht schreibt die anderen beiden Felder des Paket-Headers: Sender ID und Kosten zur Senke. Wenn man einen neuen Routing-Algorithmus einführen will reicht es aufgund der Layer-Architektur, die Netzwerkschicht neu zu implementieren. Die übrigen Schichten können unverändert bleiben.
Die **MAC-Schicht** steuert den Funk und hat folgende Funktionalitäten: Ein- und Ausschalten des Funks, Nachrichten senden und empfangen und Zurücksenden eines Acknowledgements beim Empfang einer Datennachicht.
Wie bei Ethernet wurde ein Backoff-Mechanismus implementiert. Hierbei wird nach jedem Senden, das vom Empfänger nicht mit einem ACK bestätigt wurde, eine zufällige Zeit zwischen 0 und Max-Delay gewartet, bevor das Paket erneut gesendet wird. Bei jedem weiteren Senden, bei dem ein ACK ausbleibt, wird das Max-Delay exponentiell erhöht. Nachdem die Nachricht erfolgreich gesendet wurde wird das Max-Delay wieder zurückgesetzt. Der Einfluss dieses Verfahrens auf den Energieverbrauch der Sensor-Knoten wird in Abschnitt 2.2.4 genauer erläutert.
Der eigentliche Kommunikationsstack besteht aus der Transport-, Netzwerk- und MAC-Schicht.

2.2.2 Nachbarschaftsmanagement

Jeder Sensorknoten pflegt eine eigene, lokale Nachbarschaftstabelle. Sie enthält einen Eintrag für jeden Knoten, von dem Nachrichten direkt empfangen werden können. Zusätzlich enthält die Tabelle auch noch für jeden Nachbarn dessen Entfernung zur Senke (Cost to Sink), seine Verbindungsqualität (Quality of Service, QoS) und ein Timestamp des letzten empfangenen Pakets das von dem Nachbarn gesendet wurde. Diese Informationen werden in der Netzwerkschicht durch Mithören und Auswerten der empfangenen Pakete erstellt.
Timestamp des letzten empfangenen Pakets: Dieser wird verwendet um Knoten, von denen man schon lange keine Pakete mehr empfangen hat (sog. Dead Neighbors) aus der Tabelle zu löschen. Dadurch wird sichergestellt, dass man keine Nachrichten mehr an Knoten schickt, die z.B. durch Hardware-Fehler, schon lange ausgefallen sind. In den in Abschnitt 2.3.2 bschriebenem Deployment wurde das Zeitinterval, nach dem inaktive Nachbarn gelöscht werden, auf 480 Sekunden festgelegt.
Cost to Sink: Als einfache Kosten-Metrik wird bei SensorScope die Hop-Distanz eines Knotens zur Senke gewählt. Am Anfang kennt außschließlich die Senke ihren eigenen Wert. Dieser ist nämlich per Definition null. Durch das Senden sogenannter Beacons an ihre direkten Nachbarn initiert sie die Berechnung der Kosten für alle Knoten des Netzwerks. Durch das Empfangen eines Beacons wissen die Nachbarn, dass Sie nur einen Hop von der Senke entfernt sind und beginnen mit dem Senden von Daten an die Senke. Da die Costs to Sink, wie in Abschnitt 2.2.1 beschrieben, im Header der gesendeten Daten-Pakete stehen, sind keine weiteren Beacons mehr nötig. Alle Nachbarn, die jetzt ein Datenpa-

ket mit Kosten 1 mithören wissen nun ebenfalls ihre Kosten (nämlich 2) und beginnen ebenfalls mit dem Senden von Daten-Paketen. So geht es weiter, bis alle Knoten ihre Hop-Distanz zur Senke kennen. Da Knoten normalerweise mehrere Nachbarn haben nehmen sie für sich immer den Wert der Kosten ihres kostengünstigsten Nachbarn und zählen 1 dazu. Zusammen mit der Service-Qualität (Quality of Service, QoS) sind die Costs to Sink ausschlaggebend für Routing-Entscheidungen (siehe Abschnitt 2.2.5).

Service-Qualität: Die QoS ist ein Maß sowohl für die Übertragungsqualität zu einem Nachbarn, als auch für die Fähigkeit des Nachbarn, Nachrichten weiterzuleiten. Zum berechnen dieser Metrik werden die Sequenznummern der letzten 16 von einem Nachbarn empfangenen Pakete betrachtet. Im Idealfall sind diese Sequenznummern 16 aufeinanderfolgende Zahlen. Wenn die Übertragungsqualität zwischen dem Knoten und dem Nachbarn aber schlecht ist wird es vorkommen, dass der Knoten nicht alle Pakete des Nachbarn empfangen kann und somit Sequenznummern fehlen. Wenn ein Nachbar seinerseits wieder schlechte Nachbarn hat und darum beim Senden von Nachrichten kein ACK zurückbekommt sendet er ein Paket erneut (Siehe Abschnitt 2.2.5) mit der gleichen Sequenznummer. Darum kann man aus dem Empfangen doppelter Seuquenznummern auf die schlechte Fähigkeit zum Weiterleiten von Paketen des Nachbarn schließen. Aufgrund dieser beiden Feststellungen wird die QoS mit Hilfe der Formel $QoS = \frac{16}{16+x+y}$ berechnet, wobei x für die Anzahl der fehlenden und y für die Anzahl der doppelten Sequenznummern steht [7]. Ein idealer Nachbar hat einen QoS-Wert von 1.

Anstatt über die Sequenznummern wurde auch noch in Betracht gezogen, die Übertragungsqualität mit Hilfe von RSSI (Received Signal Strength Indication) zu bestimmen. Jedoch wurde dieses Verfahren als zu ungenau bewertet. Außerdem ist es, wie man aus [2] entnimmt, viel komplexer als das simple Auswerten der Sequenznummern.

Es gibt aber auch noch Verbesserungspotential um Fehler zu vermeiden: Da alle Nachbarn, die ein Knoten hören kann, in dessen Nachbarschaftstabelle eingetragen werden, kann es auch vorkommen, dass Nachbarn, die man zwar hört, denen man aber keine Nachrichten schicken kann (asymmetrische Nachbarn) in die Nachbarschaftstabelle eingetragen werden. Wenn eine Nachricht an solche Nachbarn gesendet wird erhält man von ihnen kein ACK und die Nachricht wird erneut gesendet.

In regelmäßigen Abständen senden die Sensorknoten eine Liste ihrer Nachbarn und deren QoS zum Server. Dies ermöglicht es, Rückschlüsse über die Netzwerktopologie zu gewinnen und mögliche Schwachstellen im Netzwerk aufzuspüren [4].

2.2.3 Synchronisation

Da die Zeit, die es dauert, bis gesendete Nachrichten zum Server gelangen nicht genau berechnet werden kann, ist es nötig, dass Messwerte in Datenpaketen mit Zeitstempeln versehen werden. Außerdem ist für das in Abschnitt 2.2.4 beschriebene synchrone Duty Cycling notwendig, dass die Knoten im Netzwerk untereinander die gleiche Zeitbasis haben. Aus diesen Gründen ist es nötig, dass sich die Knoten untereinander und mit dem Server synchronisieren, was in den folgenden Abschnitten beschrieben wird.

2.2.3.1 Synchronisation zwischen den Stationen.

Die Kristalle der in SensorScope verwendeten TinyNodes haben einen theoretischen Uhrenfehler (Clock Drift) von ± 20 ppm [4], was 72 Milisekunden pro Stunde entspricht. Die Versuche aus [7] haben aber gezeigt, dass der Drift deutlich höher sein kann und auch noch von der Umgebungstemperatur abhängig ist. Dabei war der Fehler bei Zimmertemperatur bei etwa 120 ms/h und in einem Gefrierschrank sogar bei bis zu 375 ms/h. Um diese Driftraten zu kompensieren ist die Synchronisation notwendig.

Bei der Synchronisation zwischen einzelnen Sensorknoten dient die Zeit der Senke als Referenz-Zeit und es wird davon ausgegangen, dass Knoten die näher an der Senke sind (geringere Hop Distanz), eine genauere Zeit haben als welche, die weiter entfernt sind. Der Ablauf bei der Synchronisation ist in Abbildung 4 skizziert und wird im Folgenden erklärt:

Knoten **g** will die Zeit wissen und sendet einen *Sync Request* zu einem seiner Nachbarn, der näher an der Senke ist als er (**d**), da angenommen wird, dass dieser eine genauere Zeit hat. Dieser sendet einen *Sync Reply* mit seiner aktuellen Zeit als lokalen Broadcast an alle seine Nachbarn. Jeder Knoten, der den Reply empfängt setzt seine Zeit auf die aus dem Reply, falls er weiter von der Senke entfernt ist als der Sender des Reply. In der Abbildung ist dies bei den Knoten **b** und **g** der Fall. Die übrigen Empfänger (**c**, **f** und **h**) ignorieren den Reply, da sie näher oder gleich weit von der Senke entfernt sind wie **d**. Das senden eines lokalen Broadcasts hat den Vorteil, dass weniger *Sync Requests* gesendet werden, da alle Empfänger des Replys ihren Nächsten Request, je nach Synchronisationsmodus, aufschieben.

Es gibt zwei verschiedene Synchronisationsmodi. Der erste ist der *High-Frequency Mode*. Dieser wird verwendet, wenn ein Knoten die aktuelle Zeit nicht kennt, was beispielsweise der Fall ist, wenn er neu bootet. Der andere, der *Low-Frequency-Mode* ist dafür da, um die Driftrate der Kristalle der Sensorknoten zu kompensieren. Der Mode ist entscheidend dafür, wie oft sich ein Knoten mit dem Netzwrk Synchronisiert. Für die in dieser Arbeit beschriebenen Outdoor-Experiment (siehe Abschnitt 2.3.2) wurde ein Interval von 5 Sekunden für den *High-* und 1 Stunde für den *Low-Frequenzy Modus*, zwischen den Synchronisationen eines Knotens, verwendet.

2.2.3.2 Synchronisation zwischen Senke und Server.

Da der Server wissen muss, zu welcher Zeit die Messungen der Daten, die in einer Nachricht stehen, durchgeführt wurden, muss er den Offset zwischen der Zeitbasis des Sensor-Knoten-Netzwerks und seiner Zeit (tatsächliche Zeit) kennen. Als Zeitbasis für das Netzwerk wird die lokale Zeit der Senke verwendet. Damit der Server den Offset zu dieser Zeit berechnen kann wird die lokale Zeit in regelmäßigen Abständen an den Server geschickt. Wenn die Senke neu gestartet wird versucht sie zuerst sich mit ihren Nachbarn zu synchronisieren (im *High-Frequency Mode*). Schlägt dies fehl geht sie davon aus, dass das Netzwerk neu gestartet wurde und ihre lokale Zeit wird als Basis für das Netzwerk verwendet.

Dieser Offset wäre nicht nötig, wenn sich Netzwerk und Server ebenfalls synchronisieren würden. Aufgrund der schlechten Erreichbarkeit der Senke über GPRS hat sich dies aber als zu schwierig herausgestellt [4].

In Zukunft ist geplant, die tatsächliche Zeit als Basis für

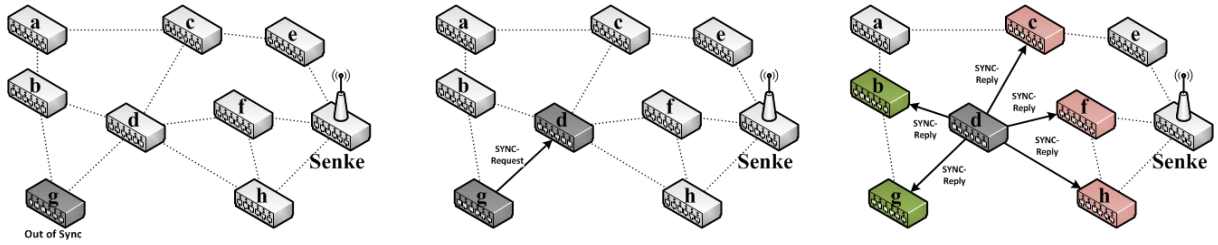


Abbildung 4: Ablauf bei der Synchronisation

das Netzwerk zu verwenden. Diese könnte man beispielsweise durch die Verwendung eines GPS Chips beziehen.

2.2.4 *Energiemanagement*

Ein Großteil des Energieverbrauchs des TinyNodes wird durch den Funk verbraucht. Bei abgeschalteten Funk verbraucht er gerade einmal 2 mA. Wenn man ihn anschaltet braucht er schon 15 mA und beim Senden verbraucht er je nach Stärke 25 - 58 mA (bei 0dBm bzw. 15 dBm). Darum ist es aufgrund der begrenzten Menge der verfügbaren Energie (siehe Abschnitt 2.1.1) in der Sensorstation nicht möglich, den Funk ununterbrochen laufen zu lassen. Eine Lösung hierfür bietet *Duty-Cycling*. Hierbei ist es entscheidend, dass alle Stationen ihren Funk gleichzeitig einschalten, um miteinander zu kommunizieren. Nach so einer Sende-Phase machen die Knoten dann eine Pause und schalten dabei ihren Funk ab, bis nach einem festen Intervall die nächste Sendephase eintritt. Für die in Abschnitt 2.3 beschriebenen Deployments wurden für die Sende-Phase 12 und für die Pause-Phase 108 Sekunden gewählt. Da sich die Stationen nur einmal in der Stunde synchronisieren laufen ihre Uhren in der Regel nicht exakt synchron. Darum passiert es, dass sie sich nicht zum genau gleichen Zeitpunkt einschalten. Darum warten sie am Anfang einer Sende-Phase 500 ms, bevor sie mit dem Senden von Nachrichten beginnen. Durch diese Wartezeit kann, auch unter Einbeziehung des Clock Drifts, sichergestellt werden, dass auch die Knoten, die etwas hinter der Referenz-Zeit sind, ihren Funk aktiviert haben.

2.2.5 *Routing*

Das Routing bei Sensorscope beruht auf Zufall. Wenn ein Knoten ein Datenpaket zur Senke senden will, wählt er zufällig einen geeigneten Nachbarn aus, an den er das Paket weiterleitet. Welche Knoten als geeignet angesehen werden richtet sich nach zwei Kriterien:

- Die Costs to Sink des Empfänger-Knotens müssen kleiner sein als die des Senders. So wird sichergestellt, dass das Paket der Senke mit jedem Hop näher kommt.
- Die Quality of Service des Empfängers muss gut sein. Um dies zu bestimmen werden zwei Thresholds festgelegt. Nachbarn, deren QoS über dem oberen Threshold liegt, werden als High Quality Neighbors bezeichnet. Falls es solche gibt, werden nur diese als geeignet angesehen. Solche, die zwischen den beiden Thresholds liegen, werden als Low Quality Neighbors bezeichnet. Diese werden als geeignet betrachtet, wenn es keine High Quality Neighbors gibt. Die restlichen, also diejenigen, die unter dem unteren Threshold liegen, werden gar nicht beachtet.

Eine weitere Möglichkeit wäre ein fester Backbone, der jeden Knoten mit der Senke verbindet. Dies hätte jedoch Nachteile gegenüber dem gewählten Verfahren. Einerseits wäre ein Overhead nötig, um Unterbrechungen (z.B. durch defekte Knoten) festzustellen. Zum anderen bestünde die Gefahr, dass es Knoten gibt, durch die sehr viele Routen gehen und es so zu einem Flaschenhals (engl. Bottleneck) kommen würde.
Ein Nachteil des verwendeten Verfahrens ist, dass das Netzwerk nur lokal betrachtet wird und so nicht zwingend die beste Route durch das gesamte Netzwerk genommen wird. Eine einfache Verbesserungsmöglichkeit wäre es, verstärkt an Knoten mit wenigen Nachbarn zu senden.
Genau wie bei der Synchronisation wurde auch beim Routing darauf geachtet, dass ein möglichst einfaches Verfahren ausgewählt wurde, um Fehler, die durch zu hohe Komplexität entstehen könnten, zu vermeiden.

2.3 Anwendungen und Ergebnisse

In diesem Abschnitt werden die bisherigen Einsätze von SensorScope erläutert und deren Ergebnisse aufgezeigt.

2.3.1 *Indoor Erprobung*

Das erste Experiment mit SensorScope führten die Entwickler in einem Bürogebäude der EPFL durch unter Verwendung von 17 Sensorstationen. Dabei ging es darum, das implementierte Netzwerkprotokoll zu testen, weshalb an die TinyNodes der Sensorstationen auch keine Sensoren angeschlossen wurden. Jeder Knoten sendete 5054 Nachrichten. Von allen bis auf einen Knoten kamen alle Nachrichten bei der Senke an. Von dem einen Knoten gingen etwa 200 verloren. Das lag wahrscheinlich daran, dass die Verbindung zu diesem Knoten temporär getrennt wurde und dadurch seine Nachrichten-Warteschlange überlief. Außerdem kam es insgesamt zu 6,5% doppelt gesendeter Pakete. Diese kommen zustande, wenn ein Acknowledgement für eine Nachricht verloren geht und diese darum erneut gesendet wird. Die Ergebnisse werden in [7] noch genauer erläutert.

2.3.2 *Einsatz im Freien: Das Projekt auf dem Genepi-Gletscher*

Nach diesem Schritt wurde damit begonnen, SensorScope auch für Outdoor-Projekte einzusetzen. Zuerst wurden Single-Hop Netzwerke getestet, um das Ganze möglichst einfach zu gestalten. Es wurden verschiedene Projekte durchgeführt. Eines davon auch auf dem 3000m hohen Genepi-Gletscher, um die Knoten unter rauhen Bedingungen zu testen.

doi: 10.2313/NET-2012-08-2_08

2.3.2.1 *Projektbeschreibung.*

Als diese Tests nach beseitigung kleinerer Probleme, wie dem fehlerhaften Zusammenspiel von Software und den Hardware-Treibern (z.B. des Solarpanels) erfolgreich beendet waren, wurde damit begonnen, Sensorscope für Multi-Hop Deployments zu verwenden. Aus Platzgründen wird hier nur auf das wichtigste Projekt eingegangen[1]. Es fand auf dem 2500m hohen Genepi-Gletscher in der Schweiz statt und dauerte 60 Tage. Dieser Ort wurde ausgewählt, weil hier nach starken Regenfällen gefährliche Schlammlawinen auftreten. Sensorscope half dabei, die Klimabedingungen auf dem Gletscher besser zu verstehen und so bessere Vorhersagen treffen zu können, wann es zu Erdrutschen kommen könnte [10]. Es gelang, das Mikroklima auf dem Gletscher zu errechnen und so Schlammlawinen besser vorhersagen zu können.

Auf einer Fläche von 500 m x 500 m wurden 16 Sensorstationen und eine mit GPRS ausgestattete Senke aufgestellt. Obwohl GPRS aufgrund der schlechten Konnektivität auf dem Gletscher nur sehr eingeschränkt verfügbar war, war es für das Projekt ausreichend. Aufgrund der Bedeutung des Projekts wurde besonders darauf geachtet, dass möglichst viele Knoten nur einen Hop Abstand zur Senke hatten, um so die Anzahl an doppelten Nachrichten möglichst gering zu halten.

2.3.2.2 *Gewonnene Erkenntnisse.*

In diesem Abschnitt wird genauer erläutert, welche Erkenntnisse durch das Genepi-Projekt für SensorScope gewonnen werden konnten.

Stromversorgung

Aus den Status-Nachrichten der Sensorknoten ging hervor, dass während des gesamten Projekts die sekundäre Batterie kein einziges mal verwendet wurde. Darum wurde geplant, in späteren Versionen von SensorScope nur noch eine Batterie einzubauen. Diese soll dafür größer sein als bisher.

Datenauswertung

Es ist wichtig, die gesammelten Messwerte zeitnah auszuwerten um Hardwarefehler schnell diagnostizieren zu können. Bei dem Genepi-Projekt kam es aufgrund eines durch Korrosion verursachten, Kurzschlusses der Interrupt-Leitung eines Regensensors zu fehlerhaften Messwerten. In einem solchen Fall ist es wichtig, den Fehler früh zu erkennen und zu beheben. Die Erkennung könnte zum Beispiel über den Vergleich mit Messwerten von anderen Sensoren geschehen.

Testbedingungen

Ein weiterer wichtiger Punkt ist es, unter möglichst realistischen Bedingungen zu testen. So gab es auf dem Genepi-Gletscher Phasen, in denen das GPRS-Signal so schwach war, dass die Senke zeitweise keine Nachrichten an den Server schicken konnte. Nachdem die Verbindung wieder hergestellt war, war die Senke ausschließlich damit beschäftigt, Nachrichten aus ihrem Speicher wegzuschicken, so dass sie momentan keine neuen Pakete mehr empfangen konnte, was zu deren Verlust führte. Dieses Fehlverhalten wurde während der Tests auf dem EPFL-Gelände nicht beobachtet, da hier eine stabile GPRS-Verbindung vorhanden war.

Kameraeinsatz

Die Sensoren von SensorScope können zwar die Niederschlagsmenge messen, aber nicht, um welche Art von Niederschlag es sich handelt. Sie können nicht zwischen Regen und geschmolzenem Schnee unterscheiden. Eine mögliche Lösung für dieses Problem wäre der Einsatz von Kameras, die Fotos von den Stationen machen. So könnte man sehen, ob Schnee liegt oder ob es regnet. Tatsächlich wurde auf dem Genepi-Gletscher bereits eine Kamera verwendet. Aufgrund des hohen Stromverbrauchs und Datenaufkommens war diese aber nicht in das SensorScope-Netzwerk integriert, sondern wurde autonom mit einer eigenen Autobatterie und einem eigenen GPRS-Gateway betrieben.

[1]Weitere Projekte sind unter [10] und [1] zu finden.

3. WEITERE UMWELTMONITORING SYSTEME

Natürlich gibt es neben SensorScope noch viele weitere Umweltmonitoring-Systeme. In diesem Kapitel werden zwei davon mit SensorScope verglichen. Zum einen Macroscope (Abschnitt 3.1), das benutzt wird um die Umwelt eines Baumes zu monitoren. Weiterhin wird noch PermaSense behandelt (Abschnitt 3.2). Es dient dem Monitoring von Temperatur und Feuchtigkeit in Felsen zur Vorhersage von Steinschlägen.

3.1 Macroscope

Die Inhalte dieses Abschnitts basieren auf den Angaben aus [13]. Macroscope wurde an der University of California in Berkeley entwickelt, um den Verlauf Mikroklimas im Umfeld eines 70m hohen Redwood-Bames zu beobachten und um Forschern zu ermöglichen, diesen Verlauf genauer zu untersuchen. Diese Daten sind relevant für Biologen, da man mit ihnen unter anderem Rückschlüsse auf das Wachstum von Bäumen ziehen kann. Mit den gewonnen Daten können sie nachvollziehen, ob ihre theoretischen Modelle auch in der Praxis zutreffen. Das Projekt hatte eine Laufzeit von 44 Tagen.

3.1.1 *Aufbau und Funktionsweise von Macroscope*

Das Netzwerk von Macroscope besteht aus 33 Sensorknoten, die auf der Funktionsweise des in [6] vorgestellten Systems TASK basieren. Sie wurden in einer Höhe von 15 bis 70 m an dem Baum angebracht und haben zueinander einen Abstand von circa 2 m. Der Knotenaufbau ist schematisch in Abbildung 5 dargestellt.

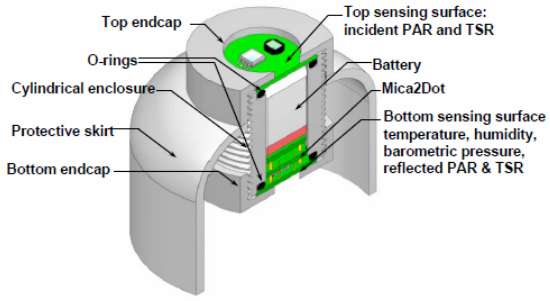


Abbildung 5: Aufbau einer MacroScope-Station[13]

Als Plattform wurde ein Mica2Dot von Crossbow verwendet, der einen Durchmesser von einem Inch hat. Außerdem verfügt die Plattform über einen Atmega128 Microprozessor

doi: 10.2313/NET-2012-08-2_08

von Atmega mit 4Mhz und einem 433Mhz-Funk von Chipcon. Weiterhin wurde ein Flash-Speicher mit einer Größe von 512 KB verbaut.
Vier verschiedene Sensoren dienen dazu, die Umgebungsbedingungen zu messen. Ein Sensor misst die Luft-Temperatur und -feuchtigkeit[2], ein weiterer den Luftdruck[3]. Weiterhin wird noch die photosynthetisch aktive Strahlung (engl. Photosynthetically Active Radiation; PAR) gemessen. Das ist der Anteil des Sonnenlichts, der für Photosynthese genutzt werden kann. Zudem wird noch das gesamte einfallende Sonnenlicht (engl. Total Solar Radiation; TSR) gemessen. Die Sensoren Für PAR[4] und TSR[5] sind je zweimal vorhanden. Einmal für das direkt einfallende Licht und einmal für das ambiente Licht, das von der Umgebung reflektiert wird. Dafür sind sie jeweils an der Oberseite des Gehäues und an der Unterseite angebracht. Zusätzlich zu Mikrokontroller und Sensoren verfügt eine Sensorstation auch noch über eine Batterie für die Energieversorgung.
Eine Solar-Vorrichtung wie bei SensorScope zum Laden der Batterie wurde nicht vorgesehen, da genung Energiesparmaßnahmen getroffen wurden um die gewünschte Laufzeit von 44 Tagen zu gewährleisten. So wurde genau wie bei SensorScope Duty-Cycling angewendet. Ein Zyklus dauerte 5 min, wovon nur 4 sec zum Austausch von Nachrichten genutzt wurden. Die hierfür nötige Synchronisation wurde von TASK erledigt. Wenn ein Knoten eine Nachricht von seinem Eltern-Knoten mithört übernimmt er dessen Zeit aufgrund eines Timestamps, der in allen Nachrichten mitgeschickt wird. So wird die Zeit des Root-Knotens Schritt für Schritt über das gesamte Netzwerk propagiert. Wenn ein Knoten über mehrere Duty-Cycles nichts von seinem Eltern-Knoten hört, bleibt er eine ganze Periode lang eingeschaltet, um sich wieder mit dem Eltern-Knoten zu synchronisieren. Mit diesem Verfahren wird eine Synchronität von 1 - 2 ms erreicht [6]. Im Gegensatz zu SensorScope sind hier keine expliziten Synchronisationsnachrichten notwendig.
Auch beim Routing gibt es Unterschiede zu Sensorscope. Bei Macroscope basiert das Routing auf MINTRoute [14]. Es werden Beacons verwendet, um für jeden Knoten den Nachbarn zu bestimmen, über den er die minimale Anzahl an Übertragungen benötigt, um eine Nachricht an die Senke zu schicken. Daraus wird ein Routing-Tree berechnet. Anders als bei SensorScope wird hier also ein Knoten immer den gleichen Weg zur Senke wählen, falls sich nichts an der Netzwerkstruktur ändert.
Genau wie Sensorscope benutzt Macroscope eine Basisstation (Senke), die die Daten per GPRS an einen Server weiterschickt.

3.1.2 Ergebnisse

Aufgrund der räumlich dicht verteilten Messpunkte war es erstmals möglich die Dynamik des Mikroklimas in der Umgebung der Redwood-Bäume festzuhalten.
So konnten nicht nur qualitative, sondern auch quantitative Aussagen, über die biologischen Prozesse des Baumes, in Abhängigkeit seiner Umwelt, zu treffen [6].

[2]Sensiron SHT11 (Temperatur: ± 0.5 °C; Luftfeuchtigkeit: ± 3.5%)
[3]Intersema MS5534A
[4]Hamamatsu S1087
[5]TAOS TSL2550

3.2 PermaSense

Dieser Abschnitt beschreibt das Umweltmonitoring-System PermaSense und basiert größtenteils auf [11]. PermaSense dient der Erfassung von Daten über Permafrost in den schweizer Alpen. Es soll Geowissenschaftlern dabei helfen, ihr Wärmefluss-Modell von steilen Felshängen zu verbessern um deren Stabilität besser vorhersagen zu können. Das hier beschriebene Projekt fand am Jungfraujoch auf einer höhe von 3500 m statt. Es wurden 10 Sensorknotenverwendet. In Abbildung 6 wird der schematische Aufbau einer Station gezeigt. Genau wie bei SensorScope wurde GPRS verwendet

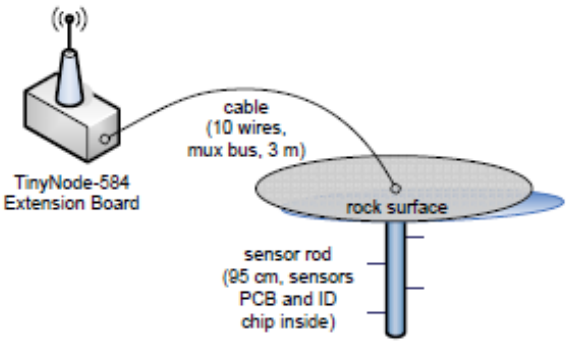


Abbildung 6: Aufbau einer PermaSense-Station [11]

um mit dem Server zu kommunizieren. Außerdem haben die Knoten auch einen TinyNode[6] und ein Semech Funk-Chip[7] verbaut, da diese einen geringen Energieverbrauch bei großer Funk-Reichweite bietet. Der geringe Energieverbrauch ist nötig, da die Knoten mit einer Batterie ohne sich wiederaufzuladen 4 bis 5 Jahre laufen sollen. Die hohe Funk-Reichweite ist nötig, weil die Knoten in Abständen von 250 bis 300 m aufgestellt wurden. Die Knoten sind mit einer 1 m langen Stange verbunden, an der Sensoren zum Messen der Temperatur und der Stromleitfähigkeit[8], angebracht sind. Diese Stange wird in ein 1 m tiefes Loch, das in den Fels gebohrt wurde, gesteckt. Das Loch hat einen Durchmesser von 14 mm. Die Sensoren sind in verschiedenen Tiefen an der Stange montiert. Die Stange ist über einen Bus mit dem TinyNode verbunden. Um robuster gegen Störungen zu sein werden immer 16 Messungn durchgeführt und dann gemittelt.
Da es vorkommen kann, dass die Knoten über 7 bis 8 Monate nicht für Menschen zugänglich sind gibt es einen Mechanismus, mit dem man die Abtastrate der Sensoren über das Netzwerk umstellen kann. Genau wie SensorScope verwendet PermaSense Duty-Cycling um Energie zu sparen. Die Länge eines Zyklus beträgt 30 Minuten und der Duty-Cycle beträgt 0,003%. Zur Synchronisation tauschen die Stationen bei jedem Zyklus ihre Zeiten aus und berechnen daraus ihre lokale Abweichung. Sie stellen jedoch daraufhin nicht ihre Uhr um, sondern passen nur das Zeitintervall bis zum nächsten Wake-Up an. Außerdem senden sie Nachrichten, in denen ihre lokale Abweichung steht. Um die Zeiten auf die tatsächliche Zeit umzurechnen bezieht die Senke die Zeit per

[6]TinyNode 585 von Stockfish
[7]Semtech XE1205
[8]Es wid die Leitfähigkeit zwischen zwei Messing-Ringen gemessen

doi: 10.2313/NET-2012-08-2_08

NTP (Network Time Protocol) über GPRS.
Genau wie bei SensorScope wird Multi-Hop-Routing verwendet, aber der Routing-Mechanismus ist ein anderer. So gibt es einen sogenannten Transmission Corridor, der besagt, wie viele Nachrichten jeder Knoten in einem Zyklus senden darf. Dies sorgt dafür, dass Knoten, die über lange Zeit nich erreichbar waren (z.B. wegen Schnee), nicht das Netzwerk mit Nachrichten überfluten. Außerdem basiert das Routing auf einem Spanning Tree.
Gemessene Daten werden so lange in den Sensor-Knoten zwischengespeichert, bis sie ein Acknowledge vom Server erhalten, das besagt, dass die Daten erfolgreich übertragen wurden.

4. ZUSAMMENFASSUNG

Zum Schluss werden in diesem Kapitel noch die drei Vorgestellten Systeme miteinander verglichen. Eine Übersicht über die Unterschiede und Gemeinsamkeiten gibt Tabelle 2.

Tabelle 2: Vergleich der drei in dieser Arbeit vorgestellten Systeme

	SensorScope	Macroscope	Permasense
Microcontroller	TinyNode	Mica2Dot	TinyNode
Stromversorgung	Solar + Batterie	Batterie	Batterie
Routing	randomisiert	fester Routing-Tree	fester Routing-Tree
Duty-Cycle	120 s / 12 s	5 min / 4 s	30 min / 50 ms
Sync untereinander	einmal pro h	einmal pro 5min	einmal pro 30 min
Sample-Rate	2 min	5 min	30 min

Während SensorScope und Permasense TinyNode Microcontroller verwenden, verwendet Macroscope einen Mica2Dot. Bei der Stromversogung hat nur SensorScope die Möglichkeit siene Batterie mit Hilfe einer Solarzelle wieder aufzuladen. Bei den anderen beiden Systemen müsste die Batterie ausgetauscht werden, wenn sie leer ist. Was aber nicht vorkommen sollte, da die Lebensdauer der Batterie so ausgelegt ist, dass sie über die gesamte Dauer der Projekte hält. Wie in Abschnitt 2.2.5 beschrieben erfolgt bei SensorScope das Routing randomisiert, während Macroscope und Permasense zu beginn einen festen Routing-Tree festlegen. Alle drei Systeme verwenden Duty-Cycling um Energie zu sparen. Jedoch variieren die Sendeintervalle von 12 s alle 2 min bis zu 50 ms alle 30 min. Die Synchronisation erfolgt bei SensorScope einaml pro Stunde. Bei Macroscope wird mit jeder gesendeten Nachricht synchronisiert, was im Normalfall alle 5 Minuten der Fall ist. Bei Permasense erfolgt die Synchronisation zu Beginn jedes Sendeintervalls, was alle 30 min der Fall ist. Bei alle drei Systemen entspricht die Abtast-Rate der Sensoren der Länge des Duty-Cycles.

5. LITERATUR

[1] Sensorscope project homepage. Webseite. Online verfügbar unter http://sensorscope.epfl.ch/; besucht am 30.6.2012.

[2] C. Alippi and G. Vanini. A rssi-based and calibrated centralized localization technique for wireless sensor networks. In *Proceeding of Fourth annual IEEE International Conference on Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006.*, pages 5–pp. IEEE, 2006.

[3] G. Barrenetxea, M. Bystranowski, O. Couach, H. Dubois-Ferriere, S. Dufey, M. Krichane, J. Mezzo, S. Mortier, M. Parlange, G. Schaefer, et al. Demoabstract: Sensorscope, an urban environmental monitoring network. In *Proceeding of 4th European conference on wireless sensor networks (EWSN), Delft, Netherlands*, 2007.

[4] G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange. Sensorscope: Out-of-the-box environmental monitoring. In *Proceeding of International Conference on Information Processing in Sensor Networks. IPSN.*, pages 332–343. IEEE, 2008.

[5] BECOsolar, http://www.becosolar.com/DataSheets/MSX-01F & MSX-005F.pdf. *Datashet: Polycrystalline OEM Modules.*

[6] P. Buonadonna, D. Gay, J. Hellerstein, W. Hong, and S. Madden. Task: Sensor network in a box. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks*, pages 133–144. IEEE, 2005.

[7] F. Ingelrest, G. Barrenetxea, G. Schaefer, M. Vetterli, O. Couach, and M. Parlange. Sensorscope: Application-specific sensor network for environmental monitoring. *ACM Transactions on Sensor Networks (TOSN)*, 6(2):17, 2010.

[8] I. L. P. Janick F. Artiola and M. L. Brusseau. *Environmental monitoring and characterization.* Elsevier Academic Press, Amsterdam Boston, 2004.

[9] X. Jiang, J. Polastre, and D. Culler. Perpetual environmentally powered sensor networks. In *Proceeding of Fourth International Symposium on Information Processing in Sensor Networks. IPSN.*, pages 463–468. IEEE, 2005.

[10] SensorScope. Portfolio. Webseite. Online verfügbar unter http://www.sensorscope.ch; besucht am 30.6.2012.

[11] I. Talzi, A. Hasler, S. Gruber, and C. Tschudin. Permasense: investigating permafrost with a wsn in the swiss alps. In *Proceedings of the 4th workshop on Embedded networked sensors*, pages 8–12. ACM, 2007.

[12] Texas Instruments, http://www.ti.com/lit/sg/slab034v/slab034v.pdf. *Datashet: TI MSP430.*

[13] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, et al. A macroscope in the redwoods. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 51–63. ACM, 2005.

[14] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27. ACM, 2003.

ISBN 3-937201-28-9
DOI 10.2313/NET-2012-08-2

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)