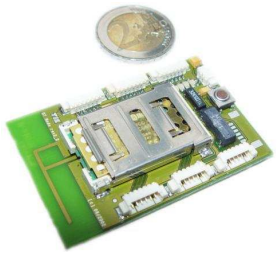
Dokumentation zur Diplomarbeit

Energiesparende BTnodes

Pieric Ferrari, März 2004



Institut für Technische Informatik
und Kommunikationssysteme TIK,
ETH Zürich

Betreuung
Lennart Meier
Jan Beutel

Professor
Prof. L. Thiele

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dieses Dokument beschreibt die im Rahmen der Diplomarbeit 'Energiesparende BTnodes' erzielten Resultate und Erfahrungen. Die Arbeit wurde am Institut für Technische Informatik und Kommunikationsnetze (TIK) am Departement für Informationstechnologie und Elektrotechnik (ITET) der ETH Zürich verfasst. Sie bildet den Abschluss meines Studiums am Departement Informatik.

In der vorliegenden Arbeit geht es um den Einsatz von BTnodes, kleinen bluetoo thfähigen Netzknoten in Sensornetzen. Nebst den Vorteilen wie beispielsweise der geringen grösse und der Kommunikationsbandbreite, weist die eingesetzte Hardware den grossen Nachteil des relativ hohen Energiebedarfs auf. Die Herausforderung besteht darin, Stromsparmethoden einzusetzen, um die begrenzte Batteriekapazität wettzumachen und damit eine verlängerte Laufzeit zu ermöglichen und gleichzeitig die Funktionalität des Sensornetzes möglichst gut zu erhalten.

Die offizielle Aufgabenstellung findet sich im nachfolgenden Kapitel. In Kapitel 3 wird die BTnode-Hardware und -Plattform vorstellt. Kapitel 4 umfasst detaillierte Erläuterungen zur Funktionsweise von Bluetooth. Kapitel 5 wendet sich den Eigenschaften und Problemen von Sensornetzen zu und stellt ein Szenario vor, für welches der Stromverbrauch optimiert werden soll. Die grundsätzlichen Methoden zur Reduktion des Stromverbrauchs und zur Erhöhung der Lebenszeit der Netzknoten werden in Kapitel 6 untersucht. Anschliessend wird die Eignung für die BTnode-Plattform überprüft, ein angepasster Algorithmus implementiert und ein erweiterter Algorithmus skizziert. Kapitel 7 analysiert den Strombedarf der BTnodes und die Effizienz des entwickelten Algorithmus und zieht ein Fazit.

Kapitel 2

Aufgabenstellung

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Prof. Dr. L. Thiele

Wintersemester 2003/2004

Energiesparende BTnodes

DIPLOMARBEIT

für
Pieric Ferrari

Betreuer: Lennart Meier
Jan Beutel

Ausgabe: 24. November 2003
Abgabe: 7. April 2004

Einleitung

BTnodes sind kleine (6 x 4 x 0.5 cm^3), programmierbare Netzknoten; sie bestehen im Wesentlichen aus einem Mikrocontroller und einem Radiomodul, welches drahtlose Kommunikation gemäss dem Bluetooth-Standard [7] ermöglicht [3, 5]. Maximale Mobilität und Wartungsfreiheit sind wünschenswerte Eigenschaften für BTnodes, besonders falls sie in einem Sensornetz [1] Verwendung finden. Die Energieversorgung ist hierbei kritisch; die BTnodes sollten möglichst lange mit einer einzigen Batterie funktionieren können.

Ziel dieser Arbeit ist es, diverse für drahtlose Netze erdachte Energiesparmechanismen [6] auf ihre Eignung für BTnodes zu untersuchen und gegebenenfalls zu verbessern und zu implementieren. Hierbei sollen statt Hardwareoptimierungen (z.B. beim Chip- oder Platinenentwurf oder bei der Auswahl der Energiequelle) Methoden auf Systemebene verwendet werden, welche per Software umgesetzt werden können. So könnte beispielsweise die Sendeleistung des Radiomoduls eines BTnodes reduziert oder das Radiomodul zeitweise komplett abgeschaltet werden.

Aufgabenstellung

1. Erstellen Sie einen Projektplan und legen Sie Meilensteine fest [8]. Erarbeiten Sie in Absprache mit den Betreuern ein Pflichtenheft.
2. Verschaffen Sie sich einen Überblick über Energiesparmechanismen für drahtlose Netze mit Hilfe von [6].
3. Machen Sie sich mit dem Aufbau und der Funktionsweise der BTnodes vertraut [4, 2, 9].
4. Untersuchen Sie die in [6] vorgestellten Methoden, auch unter Zuhilfenahme weiter führender Literatur, auf ihre Eignung für die BTnodes. Passen Sie die Methoden falls möglich an, um Sie auf BTnodes anwendbar zu machen.

5. Identifizieren Sie die aussichtsreichsten Methoden für BTnodes und skizzieren Sie, wie diese konkret in Software umgesetzt werden können.
6. Machen Sie sich mit der BTnode-Entwicklungsumgebung vertraut und implementieren Sie eine oder mehrere ausgewählte Energiesparmethoden.
7. Dokumentieren Sie Ihre Arbeit sorgfältig mit einem abschliessenden Vortrag sowie mit einem Bericht. Die Qualität der Dokumentation fliesst in die Bewertung der Arbeit ein.

Durchführung der Arbeit

Allgemeines

- Wichtig: Dokumentieren Sie Ihre Arbeit vom ersten Tag an. Halten Sie sich hierbei an die Richtlinien in [8]. Vertrauen Sie nicht auf Ihr Gedächtnis.
- Der Verlauf der Arbeit soll laufend anhand des Projektplanes und der Meilensteine evaluiert werden. Unvorhergesehene Probleme können Änderungen am Projektplan erforderlich machen. Auch diese sollen dokumentiert werden.
- Besprechen Sie Ihr Vorgehen regelmässig mit Ihren Betreuern (persönlich, per E-Mail oder telefonisch). Sie können sich jederzeit mit Fragen an Ihre Betreuer wenden. Überlegen Sie sich Ihre Fragen und mögliche Lösungswege bzw. nächste Schritte in Ihrer Arbeit im Voraus, und diskutieren Sie diese dann mit den Betreuern. Zeigen Sie Initiative [8].
- Sie sollten Ihr Projekt zu Beginn der Arbeit in einem Kurzvortrag (5 Minuten, 2-3 Folien) vorstellen. Hier werden natürlich noch keine Resultate erwartet. Am Ende der Arbeit sollen Sie Ihre Resultate im Rahmen eines 15- bis 20-minütigen Vortrags präsentieren.

Abgabe

- Geben Sie vier unterschriebene Exemplare des Berichts spätestens am festgelegten Abgabedatum bei einem der Betreuer oder nötigenfalls im Institutssekretariat (ETZ G88) ab. Diese Aufgabenstellung soll vorne im Bericht eingefügt werden.
- Räumen Sie Ihre Rechnerkonten soweit auf, dass nur noch die relevanten Dateien, Verzeichnisstrukturen usw. bestehen bleiben. Programmcode und Filestruktur sollen ausreichend dokumentiert sein, so dass eine spätere Anschlussarbeit mit geringem Aufwand auf dem hinterlassenen Stand aufbauen kann.

Literatur

[1] I.F. Akyildiz et al. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, Mar. 2002.

[2] J. Beutel, M. Dyer, O. Kasten, M. Ringwald, F. Siegemund, and L. Thiele. Bluetooth smart nodes for ad-hoc networks. Technical report, `ftp://ftp.tik.ee.ethz.ch/pub/publications/TIK-Report167.pdf`.

[3] J. Beutel, O. Kasten, and M. Ringwald. Btnodes - a distributed platform for sensor nodes. In *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003)*. ACM Press, New York, Nov. 2003.

[4] J. Beutel, O. Kasten, and M. Ringwald. Btnodes - applications and architecture compared. Technical report, `ftp://ftp.tik.ee.ethz.ch/pub/people/beutel/BKR2003.pdf`, 2003.

[5] J. Beutel et al. Prototyping wireless sensor network applications with BTnodes.

[6] L. M. Feeney. Energy efficient communication in ad hoc wireless networks. Technical report, Computer and Network Architectures Laboratory, Swedish Institute of Computer Science, 2003.

[7] J.C. Haartsen. The Bluetooth Radio System. *IEEE Personal Communications*, Febr. 2000.

[8] E. Zitzler. Studien- und Diplomarbeiten, Merkblatt für Studenten und Betreuer. ETH Zürich, TIK, Mar. 1998.

[9] BTnodes - A Distributed Environment for Prototyping Ad Hoc Networks. `http://www.btnode.ethz.ch.`

Zürich, den 1. Dezember 2003

Pieric Ferrari	Lennart Meier
Student	Betreuer

Kapitel 3

BTnode

3.1 Überblick

BTnode steht für Bluetooth-node und ist eine Plattform zur Erstellung von drahtlosen Ad-hoc-Netzwerken. Auf der Fläche einer Kreditkarte finden sich μController, Bluetooth-Modul, eine Antenne, Speicher, LEDs und diverse Schnittstellen. Die geringe Anzahl Komponenten reduziert den Herstellungsaufwand und garantiert niedrige Anschaffungskosten. Durch die Kommunikation mittels Bluetooth und die variable Betriebsspannung lassen sich BTnodes flexibel einsetzen. Die Programmierung geschieht mittels C, wobei die System-Software ein High-Level Bluetooth Interface beinhaltet. BTnodes können auch unter Linux emuliert werden. Die Kommunikation wird dann von einem über die serielle Schnittstelle betriebenen Bluetooth-Modul übernommen, dem ROK-Tester. Im Hinblick auf die Entwicklung ist das ein grosser Vorteil, da so zeitraubendes Uploaden von Programmcode auf die BTnodes entfällt.
Drei Versionen des BTnode sind bisher gebaut worden. BTnode rev1, BTnode rev2 und BTnode rev2_2. BTnode rev3 existiert auf dem Papier und wird in nächster Zeit verfügbar sein; Für diese Arbeit wurden ausschliesslich BTnodes rev2_2 eingesetzt.
Ähnliche prototypisierbare Geräte wurden von verschiedensten Hochschulen und Unternehmen entwickelt. Die Berkeley Motes[1] sind die bekanntesten. Unter anderem existieren auch die über Bluetooth kommunizierenden Intel Research Motes [2], BlueWand [3] entwickelt von Beecon oder die TeCo smart-its [4].
Zahlreiche Anwendungen sind erfolgreich für BTnodes entwickelt worden. Ein Beispiel dafür ist die bestückte Eierschachtel, die Erschütterungen oder Hitze registriert und allfällige Beschädigungen über eine Kurznachricht an ein Mobiltelefon weiterleitet [49].

3.2 μController

Als Mikrocontroller wird ein ATmega128L von Atmel eingesetzt. Die erweiterte AVR RISC-Prozessorarchitektur führt pro Taktzyklus eine Operation aus und erreicht so 8

[1] http://webs.cs.berkeley.edu/nest-index.html
[2] http://www.intel.com/research/exploratory/motes.htm
[3] http://www.beecon.de/produkte/BlueWand2/
[4] http://smart-its.teco.edu/

MIPS bei einer Frequenz von 8 MHz. Der Prozessor verfügt über einen 133 Instruktionen umfassenden Befehlssatz, 32 8 Bit breite general purpose Register, einen Zweitakt-Multiplizierer, verschiedene Timer und eine Echtzeituhr mit externem Oszillator. Der ATmega128L ist mit 128KB Flash, 4KB EEPROM und 4KB internem SRAM ausgestattet. Der Controller ist zusätzlich in der Lage, 60 KB externen Datenspeicher zu adressieren und besitzt ein JTAG-, zwei UART- und eine zweidrahtige, byteorientierte serielle Schnittstelle. Ein Serial Programming Interface (SPI) ermöglicht direkte Programmierung, beispielsweise mit Hilfe eines Atmel Low Cost Programmers. Der Controller besitzt überdies mehrere stromsparende Schlafmodi und die Taktfrequenz lässt sich softwaremässig anpassen. Die Betriebsspannung des ATmega128L liegt zwischen 2.7V und 5.5V.
Die folgenden Unterkapitel beziehen sich auf Aspekte des Prozessors, die im Kontext der dokumentierten Arbeit von Interesse sind. Eine ausführliche Beschreibung des ATmega128L-Controllers findet sich in [3].

3.2.1 Clocks

Die Architektur des ATmega128L ist so konzipiert, dass jedes Modul eine separate Clock verwendet. Diese können unabhängig voneinander angehalten werden um Stromsparfunktionen zu ermöglichen.

- clk_{ADC} taktet den Analog Comparator,
- clk_{ASY} kann vom Asynchronen Timer/Zähler verwendet werden,
- clk_{CPU} taktet den Prozessorkern und das RAM,
- clk_{FLASH} ist für die Taktung des Flash und des EEPROM zuständig,
- clk_{IO} kann vom asynchronen Timer/Zähler verwendet werden und ist verantwortlich für alle Arten von I/O.

clk_{ASY} ist abhängig vom externen Timer/Counter Oszillator, die restlichen Clocks von der Main Clock. Diese kann über einen Clock Multiplexer wahlweise mit verschiedenen internen oder externen Oszillatoren betrieben werden.

3.2.2 Stromsparende Zustände

Die Anwendung kann den Controller in Schlafzustände versetzen, um unbenützte Komponenten auszuschalten. Sechs Modi stehen zur Auswahl. Abbildung 3.1 zeigt eine Übersicht aller Zustände, der jeweils deaktivierten Komponenten und der Reaktivierungsbedingungen.

- Idle Modus. Der Idle-Modus stoppt clk_{CPU} und clk_{FLASH}, deaktiviert also den Prozessor und den Flash-Speicher. Dieser Modus wird verlassen, sobald interne oder externe Interrupts auftreten. Dazu gehören Timer-Overflow-Interrupts, UART Transmit Complete Interrupts und Interrupts vom Analog Comparator.
- ADC Noise Reduction Modus. Dieser Modus stoppt zusätzlich zu clk_{CPU} und clk_{FLASH} auch clk_{IO}. Durch das Ausschalten der digitalen Schaltkreise wird das Umgebungsrauschen des Analog Comparators verringert und ermöglicht höher aufgelöste Messungen. Folgende Ereignisse veranlassen den Controller den ADC

Noise Reduction Modus zu verlassen: ADC conversion complete Interrupt, externer Reset, Watchdog-Reset, Two-Wire Serial Interface Address Match Interrupt, Timer Overflow Interrupt, SPM/EEPROM Ready Interrupt und externe Interrupts.

- Power Down. Alle Clocks werden angehalten und nur ein externer Interrupt oder ein Two-wire Serial Interface Address Match Interrupt lassen den Controller wieder aufwachen.

- Power Save. In diesem Zustand läuft nur noch clk_{ASY} und damit verbunden der Timeroszillator. Power Save wird verlassen, sobald ein Two-wire Serial Interface match Interrupt, ein externer Interrupt oder ein Timer Overflow auftritt.

- Standby. Dieser Modus entspricht dem Power Down Modus, wobei die Main Clock nicht angehalten wird.

- Extended Standby. Dieser Modus entspricht dem Power Save Modus, wobei die Main Clock nicht angehalten wird.

Modus			Idle	ADC Noise Reduction	Power-down	Power-save	Standby	Ext. Stdby
Aktive / inaktive Komponenten	Clocks	clk_{CPU}						
		clk_{FLASH}						
		clk_{IO}	x					
		clk_{ADC}	x	x				
		clk_{ASY}	x	x		x		x
	Oszillatoren	Main Clock	x	x			x	x
		Timer	x	x		x		x
Zustandsverlassende Ereignisse	Interrupt 7:0		x	x	x	x	x	x
	TW Serial Addr match		x	x	x	x	x	x
	Timer-Overflow		x	x		x		x
	SPM / EEPROM ready		x	x				
	ADC		x	x				
	I/O		x					

Abbildung 3.1: Schlafzustände des Controllers.

3.2.3 Taktfrequenz

Neben den Schlafmodi bietet der Prozessor auch die Möglichkeit die Taktfrequenz während der Ausführung anzupassen. Alle Taktfrequenzen $f_{CPU}, f_{I/O}, f_{ADC}$ und f_{FLASH} werden dazu optional durch einen Wert zwischen 2 und 129 geteilt. Damit werden Betriebsfrequenzen von 8 MHz, 4 MHz, $2\frac{2}{3}$ MHz, ..., 64 kHz möglich.
Gesteuert wird diese Frequenzdivision durch das XDIV-Register (Abbildung 3.2). Das XDIVEN-Bit gibt an, ob eine Frequenzteilung durchgeführt wird oder nicht. Die Bits

XDIV0 - XDIV7 bestimmen den Divisorwert d, die resultierende Frequenz berechnet sich nach der folgenden Formel:

$$f_{clk} = \frac{Source\,Clock}{129 - d}$$

Alle Registerbits enthalten als Intialwert 0, die Betriebsfrequenz bleibt damit unverändert. XDIV0 - XDIV7 können nur verändert werden solange XDIVEN gelöscht ist, erst durch das Setzen von XDIVEN wird die Frequenz angepasst.

	XDIVEN	XDIV6	XDIV5	XDIV4	XDIV3	XDIV2	XDIV1	XDIV0
Bit	7	6	5	4	3	2	1	0

Abbildung 3.2: XDIV-Register.

3.3 Bluetooth-Modul

Das Bluetooth-Modul ist ein Ericsson ROK 101 007. Es ist scatternetzfähig und verfügt über eine Sendeleistung der Klasse 2, was 1mW oder einer Reichweite von rund 10 Metern entspricht. Die maximale Datenrate liegt bei 460kbit/s. Das Bluetooth-Modul inte griert einen UART-Anschluss für Daten, eine PCM-Schnittstelle für Sprache und ein USB-Interface für Daten und Sprache. Das Modul verfügt sowohl über Baseband-Controller, als auch über eingebauten Flash-Speicher und natürlich einen Radio-Sender/-Empfänger.
Beim BTnode rev2_2 ist das Modul in einem Bluetooth Module Carrier untergebracht, was leichtes Auswechseln ermöglicht.

3.4 Schnittstellen und Anschlüsse

Der BTnode verfügt über 8 Anschlüsse.

- Da der BTnode mit einem Spannungsregulator ausgestattet ist, kann der Stromanschluss (J9) mit Spannungen von 3.3V bis 12V versorgt werden. Zur Stromversorgung kann alternativ Pin1 und Pin6 jeder Schnittstelle (ausser ISP J1) verwendet werden, dabei wird aber der Spannungsregulator umgangen und eine Spannung von 3.3V wird vorausgesetzt.

- Die ISP-Schnittstelle J1 (In System Programming) wird verwendet um den BTnode zu programmieren. Dazu wird ein Atmel STK500 oder ein ATAVRISP (Atmel low cost programmer) eingesetzt. Der Programmer sollte während des Programmiervorgang nicht vom BTnode getrennt werden, da der FLASH-Speicher des Gerätes dadurch gelöscht werden kann.

- Über die Bluetooth PCM Schnittstelle J7 kann Sprache mittels Bluetooth übertragen werden.

- Der Controller verfügt über zwei UART-Schnittstellen (Universal Asynchronous Receiver-Transmitter). Die eine dient der Steuerung und dem Datenaustausch mit dem Bluetooth-Modul, die andere kann von der Anwendung zur seriellen Kommunikation mit 57600 bit/s und ohne flow control eingesetzt werden (J4). Verbunden mit einem Terminal lassen sich so Debug- und Statusinformationen anzeigen. Die höhere Betriebsspannung der seriellen PC-Schnittstelle macht in diesem Fall einen RS232 Level Converter nötig.
- Die Sensorboard Schnittstelle J5 wird verwendet um ein Sensorboard an den BTnode anzuschliessen.
- Die drei generischen Schnittstellen J2, J3 und J6 haben keine vorbestimmte Funktion und können von der Anwendung frei verwendet werden. Sie sind direkt mit den I/O-Pins des Controllers verbunden.

3.5 BTnode Systemsoftware

Die BTnode Systemsoftware ist ein eventgesteuertes Betriebssystem mit einer Funktionsbibliothek, die den Zugriff auf Komponenten des BTnode wie Bluetooth, serielle Schnittstelle, LEDs und ähnliches ermöglicht. Die Systemsoftware gliedert sich in drei Teile, Abbildung 3.3 zeigt eine vereinfachte Darstellung davon.
Der erste Teil besteht aus Low-level-Treibern, die direkt mit der BTnode Hardware interoperieren. Dazu gehören unter anderem die Echtzeituhr (RTC), Treiber für serielle Ports und Treiber für die LEDs.
Der zweite bildet der Dispatcher, welcher kooperatives Multitasking implementiert. Tasks werden mit Events assoziert und sequentiell in der Reihenfolge der auftretenden Ereignisse abgearbeitet. Events werden einerseits von Low-level-Treibern generiert und können durch die Anwendung abgefangen werden, andererseits ist es möglich, eigene Events zu registrieren und auszulösen.
Der dritte Teil umfasst einen reduzierten Bluetooth Protokollstack, welcher in Abbildung 3.4 dargestellt ist. Die Abbildung beinhaltet ebenfalls das Sensornet-Protokoll (siehe Kapitel 7.1).

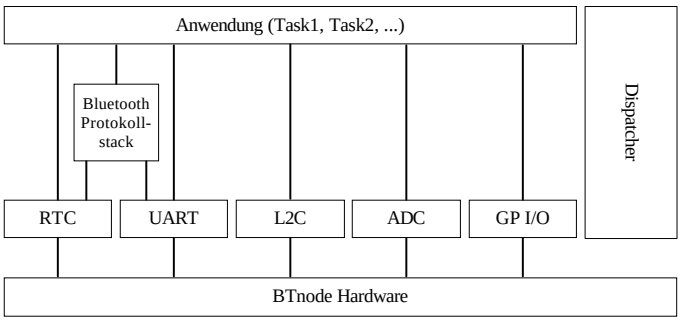


Abbildung 3.3: System Software des BTnode.

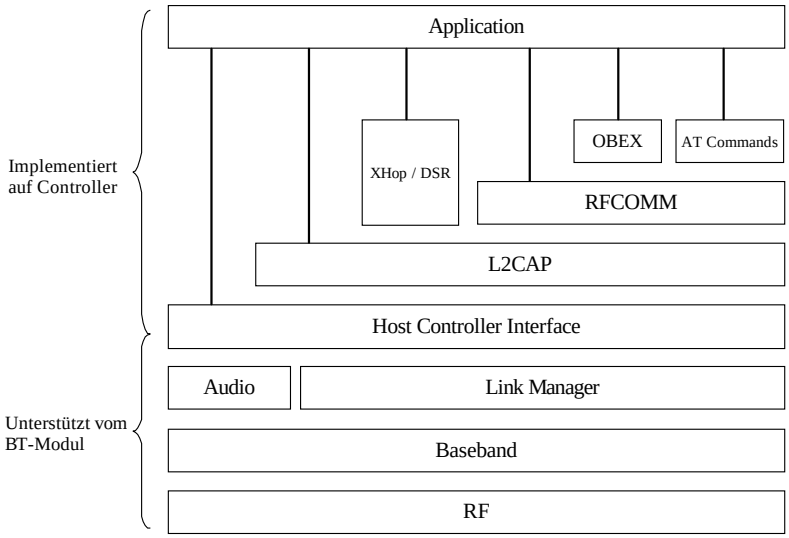


Abbildung 3.4: BTnode Bluetooth Protokollstack.

Kapitel 4

Bluetooth

4.1 Einleitung

Hinter dem Begriff Bluetooth versteckt sich eine Technologie zur drahtlosen Datenübermittlung. 1998 haben Unternehmen wie IBM, Intel, und Ericsscon begonnen einen solchen, nach dem dänischen König Blatand (Blauzahn) benannten Standard auszuarbeiten. Inzwischen sind bluetoothfähige Geräte überall anzutreffen. Mobiltelefone, Laptops, Headsets und Drucker sind bereits mit dieser Technik ausgestattet, und die Anwendungsmöglichkeiten scheinen unbegrenzt: Health-Monitoring in Spitälern, Gepäckverfolgungssysteme in Flughäfen und Überwachungssysteme in Privathaushalten.
Bluetooth beruht auf Funktechnik im 2.4 GHz Bereich und erlaubt asynchrone Datenübermittlung und synchrone Übermittlung von Sprache oder Daten mit Datentransferraten bis zu 1 Mbit/s. Das sogenannte ISM-Frequenzand (Industrial, Science, Medicine) ist weltweit lizenzfrei verfügbar. Der Bluetooth-Standard sieht drei Leistungsklassen vor, wobei die übliche Sendeleistung 1mW beträgt und Kommunikation über eine Distanz von etwa zehn Metern erlaubt. Die maximale Sendeleistung beträgt 100mW und ermöglicht Reichweiten um 100m.

4.2 Funktionsweise

Um einen paketorientierten physischen Kanal zwischen zwei bluetoothfähigen Geräten einzurichten, wird eine sogenannte Baseband-Verbindung aufgebaut. Der Verbindungsaufbau ist in zwei Operationen unterteilt: Inquiry findet Geräte in Reichweite, Paging leitet die eigentliche Verbindungsaufnahme ein.
Bluetooth verwendet ein Frequenzsprungverfahren (frequency hopping), wobei das Frequenzspektrum in 79 Kanäle à 1 MHz aufgeteilt wird. Die Frequenz wird nach jeweils 625μs nach dem Pseudozufallsprinzip gewechselt. Die Sequenz wird durch die 48 Bit lange MAC Adresse des Piconetz-Mastergerätes (das verbindungsanfragende Gerät, siehe 4.3) bestimmt.
Man unterscheidet sechs Baseband-Pakettypen, welche sich in der Länge der Payload und der Fehlerkorrektur unterscheiden. Lange Pakete ohne Fehlerkorrektur ermöglichen hohe Datenübertragungsraten. Pakete mit Fehlerkorrektur werden dagegen vorzugsweise zur Datenübermittlung auf fehlerbehafteten Verbindungen eingesetzt. Bluetooth unterstützt sowohl Forward Error Correction (Redundanz) als auch Backward Error Correction (erneutes Übertragen). Die Pakete enthalten zusätzlich zum Header einen

Zugriffscode, dem sowohl bei Inquiries als auch bei der Kommunikation in Piconetzen eine wichtige Rolle zukommt. Mittels Paging werden physische Kanäle aufgebaut, worauf verschiedene Kommunikationsprotokolle eingesetzt werden können (siehe Kapitel 4.4). Abbildung 4.1 zeigt das Zustandsdiagramm von Bluetooth.

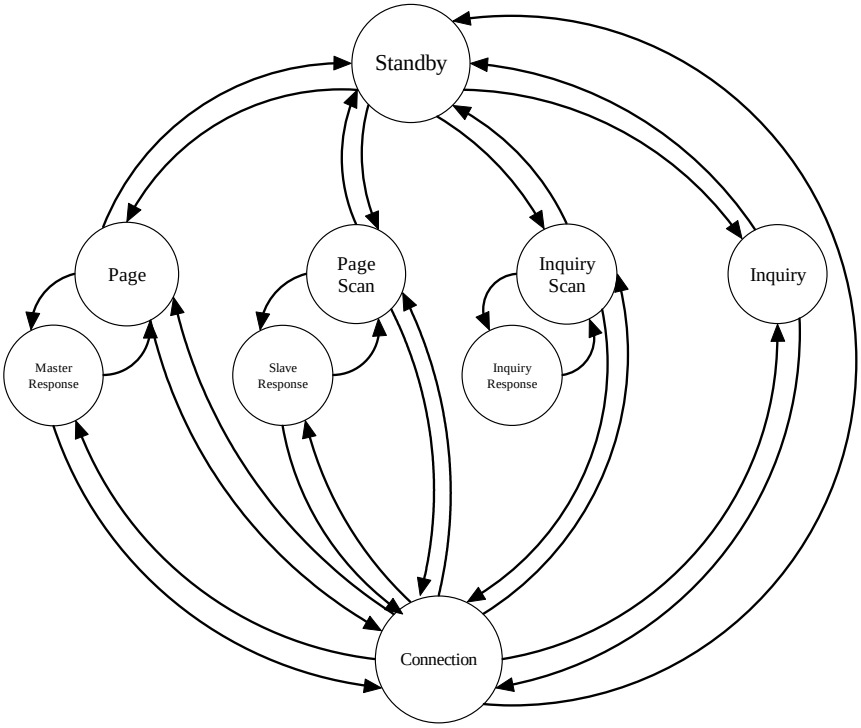


Abbildung 4.1: Bluetooth Zustandsdiagramm

4.2.1 Inquiry

Um Bluetooth-Geräte in Reichweite zu finden wird die Inquiry-Prozedur verwendet. Das Prinzip basiert darauf, Pakete in zeitlich kurzen Abständen auf insgesamt 32 verschiedenen Frequenzen zu senden. Geräte, welche sich im Inquiry-Scan Modus befinden und auf der Frequenz des gesendeten Pakets horchen, können darauf antworten und sich so zu erkennen geben.
Der Kanal des anfragenden Gerätes wird in gerade und ungerade Zeitschlitze konstanter Länge eingeteilt, wobei die geraden für das Senden, die ungeraden für das Empfangen von Daten vorgesehen sind. Während eines geraden Zeitschlitzes wird ein Paket mit einem GIAC (General Inquiry Access Code) im Access Code Feld auf jeweils zwei verschiedenen Frequenzen übermittelt. Die Frequenzabfolge wird durch den GIAC bestimmt und umfasst 32 verschiedene Frequenzen, welche in zwei Gruppen - sogenannte Trains - eingeteilt sind. Gemäss dem Bluetooth Standard muss Train A 256 mal wiederholt werden, bevor Train B verwendet werden darf.

Ein einzelner Train dauert demnach $16 * 625\mu s = 10000\mu s = 10ms$. Nach $256 * 10ms = 2560ms = 2.56s$ kommt Train B zum Einsatz.

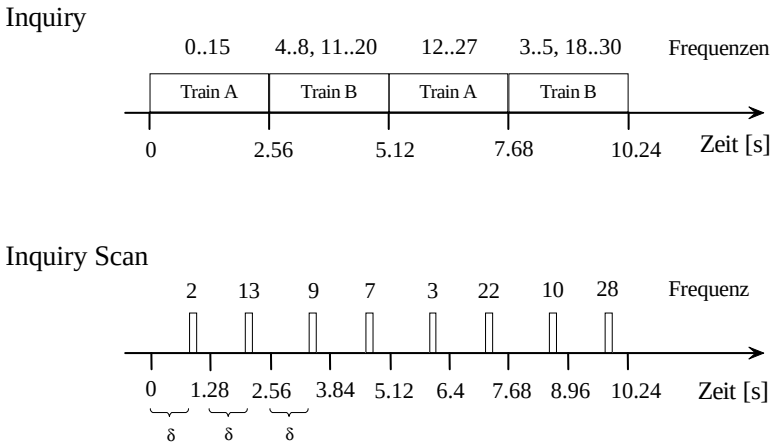


Abbildung 4.2: Zeitdiagramme Inquiry, Inquiry Scan

Um die Wahrscheinlichkeit von Paketkollisionen zwischen gleichzeitig antwortenden Geräten zu verringern wird ein Antwortprotokoll eingesetzt. Empfängt ein Gerät eine Inquiry-Nachricht, so antwortet es nicht unmittelbar sondern lässt zuvor eine zufällige Anzahl Zeitschlitze im Intervall zwischen 0 und 1023 verstreichen. Die Antworten erfolgen mittels Frequency-Hopping-Sequence-Paketen (FHS). In einem FHS-Paket[1] sind die eigene Adresse und das Zeitoffset enthalten. Die durch FHS-Nachrichten empfangenen Informationen über Geräte werden vom Bluetooth-Modul für den späteren Verbindungsaufbau gespeichert.

Wann und wie lange Inquiries ausgeführt werden ist nicht spezifiziert. Es liegt also in der Entscheidung des Programmierers dies regelmässig oder nur vereinzelt zu tun. Im Hinblick auf den Energieverbrauch sind Inquiries die teuersten Operationen des Bluetooth-Moduls. Es muss beachtet werden, dass die Ergebnisse von Inquiries nicht deterministisch sind und daher die Wahrscheinlichkeit besteht, dass Geräte in Reichweite nicht auf Anhieb gefunden werden.
Inquiries benötigen alle verfügbaren Zeitschlitze, daher bleiben Geräte während dieser Zeit unsichtbar und es können auch keine Verbindungen zu ihnen hergestellt werden.

4.2.2 Inquiry Scan

Um Inquiries zu entdecken verlassen Bluetooth-Geräte periodisch den Connection- oder Standby-Zustand um Inquiry-Scans auszuführen. Jedesmal wird auf einer anderen Frequenz auf eintreffende Inquiry-Nachrichten gewartet und gegebenenfalls mit einem FHS-Paket geantwortet.

[1] Die BTnode Systemsoftware bietet Zugriff auf Informationen der FHS-Pakete mit dem Befehl $btn\_bt\_inquire\_verbose$

Das Frequenzband ergibt sich wiederum pseudozufallsgeneriert aus der eigenen Adresse. Die Periode aufeinanderfolgender Scans beträgt 0s (continuous scan), 1.28s (R1-Modus) oder 2.56s (R2-Modus), die Dauer der Scans entspricht der Länge eines einzelnen Trains, also 10ms.
Das Bluetooth-Modul der BTnodes arbeitet im R1-Modus. Dies zeigt die Tatsache, dass die Dauer von Inquiries immer ein Vielfaches von 1.28 s beträgt und wird ausserdem durch eigene Messungen bestätigt (siehe Kapitel 8.4). Inquiry Scans werden vom Bluetooth-Modul selbstständig ausgeführt, lassen sich aber bei Bedarf deaktivieren. Geräte bleiben in diesem Fall für Inquiries unsichtbar, trotzdem können Verbindungen zu ihnen aufgebaut werden. Wie bereits angedeutet können im Inquiry-Zustand keine Inquiry-Scans durchgeführt werden.

4.2.3 Paging

Paging kommt zum Einsatz wenn eine dauerhafte Punkt-zu-Punkt-Verbindung hergestellt werden soll. Informationen aus vorhergehenden Inquiries werden dazu verwendet, deren Vorhandensein ist allerdings keine Voraussetzung.
Beim Paging werden - ähnlich wie beim Inquiry - Pakete auf 32 veschiedenen Frequenzen - aufgeteilt in zwei Trains und in geraden Zeitschlitzen - gesendet, bis eine Antwort eintrifft. Das Wissen über das clock-Offset des Slave-Gerätes und dessen Adresse lassen eine Schätzung zu, wann und auf welcher Frequenz sich das Gegenüber im Page-Scan Zustand befindet. Sei $f(k)$ die Frequenz des geschätzten Hops k des Slave, so besteht Train A der Sendefrequenzen aus $f(k-8), f(k-7), .., f(k), .., f(k+6), f(k+7)$, Train B aus den verbleibenden Frequenzen.
Ein Train dauert $16 * 625\mu s = 10000\mu s = 10ms$ und wird abhängig vom Modus einmal, 128 mal oder 256 mal wiederholt.
Empfängt ein Gerät eine für ihn bestimmte Page-Nachricht, so wird diese auf der gleichen Frequenz erwidert. In den nächsten beiden Zeitnischen werden FHS-Pakete ausgetauscht, womit das angefragte Gerät die eigentliche Slave-Rolle einnimmt und die ersten Datenpakete übermittelt werden können. Abbildung 4.4 zeigt dieses Protokoll auf Paketebene.

4.2.4 Page Scan

Genauso wie Inquiry Scan Inquiries beantwortet, ist Page Scan für die Erkennung von Paging verantwortlich. Periodisch wird auf einer von 32 Frequenzen nach Page-Nachrichten gehorcht, wobei die Dauer zwischen aufeinanderfolgenden Page Scans wiederum 0s, 1.28s oder 2.56s betragen kann. Page Scans werden vom Bluetooth-Modul selbstständig ausgeführt, können aber unterdrückt werden. Die Geräte bleiben dann zwar auffindbar, Verbindungen zu ihnen können aber nicht aufgebaut werden. Auf Abbildung 4.3 ist das Prinzip des Pagings graphisch dargestellt.

4.2.5 Connection

Es existieren vier verschiedene Arten von Connection-Zuständen

- Aktive Verbindung
  Das Master-Gerät teilt die Zeitnischen für den bidirektionalen Datenverkehr mit den Slave-Geräten ein. In diesem Zustand befinden sich die Geräte nach erfolgtem Verbindungsaufbau.

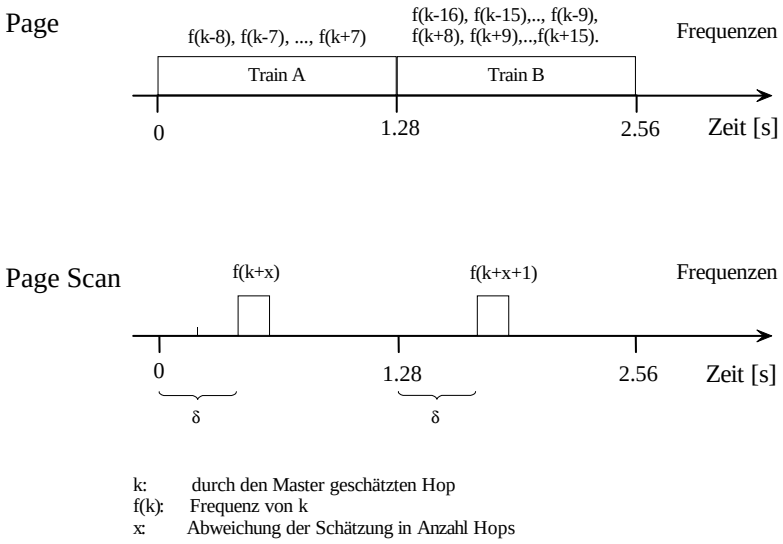


Abbildung 4.3: Zeitdiagramm Page, Pagescan

- Sniff-Modus
  Der Sniff-Modus wird hauptsächlich verwendet um Energie zu sparen. Dabei wird das Listen-Intervall der Slaves vergrössert.

- Hold-Modus
  Im Hold-Modus werden bis zum Ablauf eines Vereinbarten Timeouts ACLs (Asychronous Connectionless Link) nicht unterstützt.

- Park-Modus
  Wenn ein Slave-Gerät nicht an einem Piconetz teilnimmt, aber trotzdem synchronisiert bleiben soll, wird der Park-Modus verwendet. Geparkte Geräte tauschen ihre 3 Bit lange AM-Adresse (Active Member Address) durch eine 8 bit lange PM-Adresse (Parked Member Address) ein. Beim Wiedereintritt in den Active-Modus wird ihnen eine neue AM-Adresse zugewiesen. Ein geparktes Gerät wacht regelmässig auf, um Resynchronisations- und Broadcastnachrichten zu empfangen. Der Bluetooth-Standard unterstützt pro Piconetz bis zu 255 geparkte und 7 aktive Slaves.

4.3 Piconetze und Scatternetze

Wie in den vorangegangenen Abschnitten erklärt, nehmen bei Bluetooth-Verbindungen Geräte entweder eine Master- oder eine Slave Rolle ein. Die Instanz, welche den Verbindungsaufbau einleitet, wird definitionsgemäss zum Master und gibt mit ihrer Adresse die Frequenzabfolge vor. Ein Master kann bis zu sieben Verbindungen offen halten. Ein Master und seine Slaves bilden ein Piconetz.

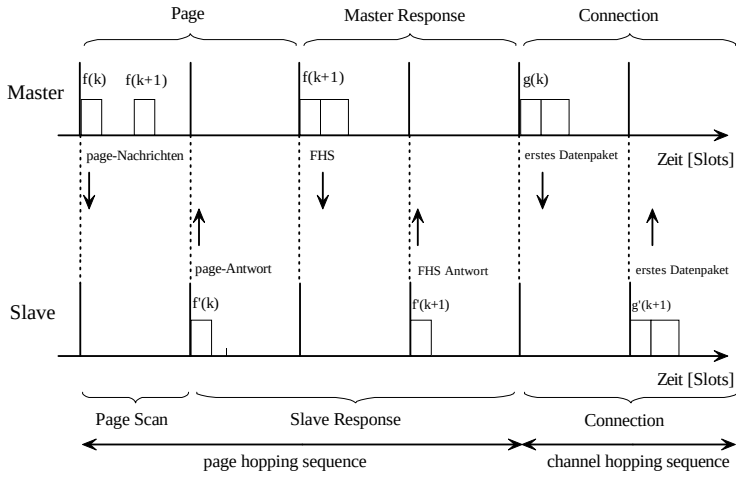


Abbildung 4.4: Aufbau einer Baseband-Verbindung

Piconetze lassen sich zu Scatternetzen zusammenschliessen, dabei ist die Frequenzsequenz nur innerhalb eines Piconetzes gleich. Verbindungsknoten müssen in diesem Fall eine Master-Slave-Doppelrolle einnehmen (Siehe Abbildung 4.5). Sie besitzen die Funktion eines Slaves, agieren aber gleichzeitig als Master eines anderen Piconetzes. Mit dieser Methode lassen sich beliebig viele Piconetze zu einem einzigen Scatternetz zusammenschliessen.
Nicht alle Bluetooth-Module unterstützen die Scatternet-Funktionalität. Mit dem im BTnode rev 2_2 eingesetzten ROK 101 007 lassen sich solche Scatternetze bilden, der Treenet-Algorithmus[2], welcher selbstständig eine Baumtopologie aufbaut zeigt dies eindrücklich[7].

[2]http://www.btnode.ethz.ch/scatternet

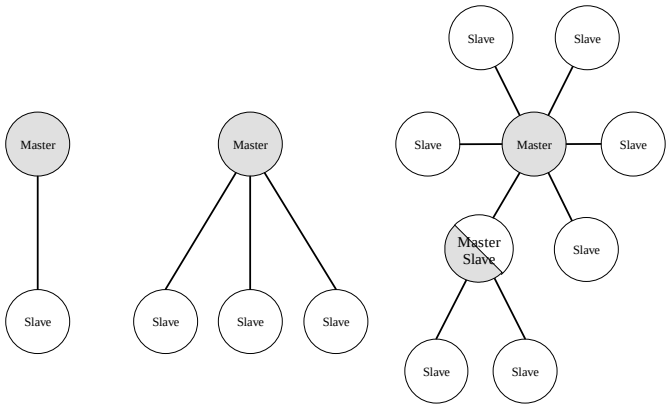


Abbildung 4.5: Einfache Verbindung, Piconetz, Scatternetz (v.l.n.r.)

4.4 Der Bluetooth-Protokollstack

In diesem Kapitel wird auf den kompletten Protokollstack eingegangen [38]. Der Aufbau des vereinfachten Protokollstacks - wie er in den BTnodes Verwendung findet - ist in Abbildung 3.4 dargestellt.

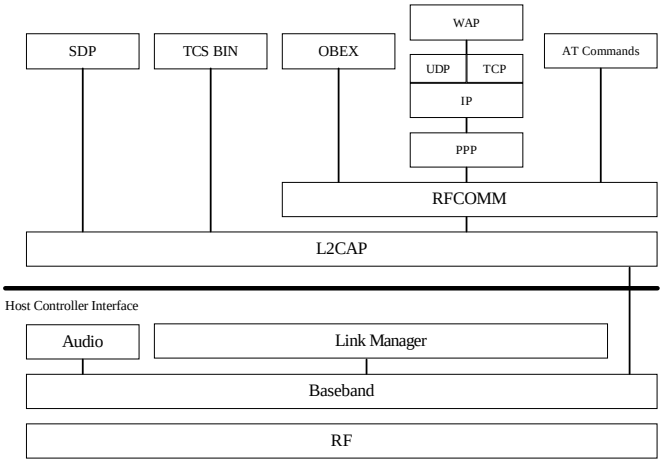


Abbildung 4.6: Bluetooth Protokollstack

- Baseband
  Die Baseband und Link Control Schichten ermöglichen eine physische Verbindung zwischen zwei Bluetoothgeräten innerhalb des gleichen Piconetzes. Diese Verbindung kann sowohl asynchron und verbindungslos als auch synchron und verbindungsorientiert sein.
- Audio

Audiodaten können über einen Audio-Link zwischen Bluetoothgeräten direkt auf der Baseband-Verbindung mit SCO-Paketen ausgetauscht werden.

- Link Manager Protocol
  Das Link Manager Protocol (LMP) ist verantwortlich für Sicherheitsaspekte beim Aufbau einer Verbindung und kümmert sich um Authentisierung und Verschlüsselung mit der dazugehörigen Schlüsselgenerierung und dem Schlüsselaustausch. LMP steuert ausserdem die Park Modes und kontrolliert die Verbindungszustände in Piconetzen.

- Logical Link Control and Adaption Protocol (L2CAP)
  Die L2CAP Protokollschicht teilt Pakete hierarchisch höhergelegener Protokolle zur Übermittlung in Stücke maximaler HCI-Länge auf und kümmert sich um deren Rekonstruktion. L2CAP arbeitet auf asynchronen, verbindungslosen und verbindungsorientierten Links und und leitet im Falle eines Mastergerätes die Pa kete an den richtigen Slave weiter. Jeder logische L2CAP Kanal wird mit einem lokalen Channel Identifier (CID) assoziiert. Übergeordnete Protokolle werden anhand des Protocol Service Multiplexer (PSM) erkannt. Der Verbindungsaufbau geschieht unter Angabe des PSM und der Geräteadresse, worauf L2CAP dem logischen Kanal eine CID zuweist. Jede Schicht oberhalb von L2CAP besitzt eine eindeutigen PSM:

  - SDP 0x0001
  - RFCOMM 0x0003
  - TCS-Bin 0x0005 (Telephony Control Protocol)
  - TCS-Bin Cordless 0x0007

- Service Discovery (SDP)
  Services sind ein wichtiger Bestandteil des Bluetooth-Frameworks, da sie die Basis vieler Anwendungsmodelle bilden. Mit dem Service Discovery Protocol können Geräteinformationen und die Charakteristiken angebotener Geräte abgefragt werden.

- RFCOMM
  RFCOMM emuliert eine serielle RS232-Verbindung über Baseband und dient höheren Protokollschichten zum Transport.

- Telephony Control Protocol - Binary
  Das bitorientierte Telephony Control Protocol - Binary (TCS BIN) kontrolliert Sprach- und Datenverbindungen und kümmert sich um TCS Gerätegruppen.

- Telephony Control Protocol - AT Commands
  AT Commands steuert Mobiltelefone und Modems.

- OBEX
  OBEX oder IrOBEX ist ein von der Infrared Data Association (IrDA) entwickeltes Sessionprotokoll, welches den Austausch von Objekten auf einfache Art und Weise ermöglicht.

- WAP
  Das Wireless Application Protocol umfasst eine ganze Klasse von Protokollen für drahtlose Netzwerke mit dem Ziel, Internetinhalte auf Mobiltelefonen und anderen Geräten verfügbar zu machen.

- PPP
  PPP wird verwendet um Punkt-zu-Punkt IP-Verbindungen auf dem LAN zu realisieren.

- TCP, UDP, IP
  Diese Protokolle wurden entwickelt, um Kommunikation über das Internet zu ermöglichen. Der Einsatz von TCP/IP ist weit verbreitet und wird von verschiedensten Geräteklassen, üblicherweise in Verbindung mit Sockets, unterstützt.

Kapitel 5

Sensornetzwerke

5.1 Einleitung

Ein Sensornetzwerk bildet ein Medium zur Datenübertragung zwischen einem Beob achter und einem Phänomen, das es zu überwachen gilt. Ein Beobachter ist diejenige Endinstanz, welche gewonnene Daten auswertet und verfügbar macht oder selbstständig Massnahmen ergreift. Die Kommunikation erfolgt üblicherweise unidirektional in Richtung des Beobachters. In typischen Sensornetzwerken nehmen die Sensoren nur ihre unmittelbare Umgebung wahr und übermitteln die Messwerte anderen Sensoren oder direkt an den Beobachter. Hierarchisch können Informationen aggregiert und komplementiert werden und vermitteln dem Beobachter so den Zustand des gesamten Phänomens. An einem Sensornetzwerk kann ein Beobachter oder auch mehrere beteiligt sein.
Die wichtigsten Eigenschaften von Sensornetzwerken werden in [57] zusammengefasst.

5.2 Performance

Die Performance von Sensornetzwerken kann anhand verschiedener Aspekte bewertet werden.

- Enegieeffizienz / Lebensdauer des Netzes
  Die Lebensdauer des Netzes bedingt durch den begrenzten Energievorrat der Knoten kann auf verschiedene Arten definiert werden. Die Definition kann beispielsweise über die Zeitspanne bis zum ersten Knotenausfall oder über die Zeitspanne bis zum Ausfall der Hälfte aller Knoten erfolgen.

- Latenz
  Die Verzögerung mit welcher der Beobachter über die Sensorwerte informiert wird.

- Präzision
  Der Begriff der Genauigkeit ist in verschiedenen Anwendungsgebieten anders definiert. Um die Präzision eines Sensornetzwerkes zu erhöhen kann beispielsweise die Abfragefrequenz der Sensoren oder die Anzahl involvierter Senso-

ren erhöht werden. Zwischen Präzision, Latenz und Enegieeffizienz existiert ein Trade-off.

- Fehlertoleranz
  Sensorausfälle dürfen nicht die Funktionstüchtigkeit des gesamten Netzes gefährden und können daher vor der Anwendung verborgen werden. Fehlertoleranz kann beispielsweise durch Replikation (SPIN-Protokoll [24]) erreicht werden.

- Skalierbarkeit
  Ein wichtiger Faktor in Sensornetzwerken ist die Skalierbarkeit. Die Funktionsweise darf durch die Menge der beteiligten Knoten nicht eingeschränkt werden.

5.3 Architektur

Ein Sensornetzwerk lässt sich auf drei Ebenen betrachten. Die Infrastruktur entspricht der untersten und hängt von den Charakteristiken der Sensorknoten (Sensorreichweite, Speichergrösse, Batterietyp, Übertragungsreichweite) und der Einsatzstrategie (Sensordichte, Lage der Sensoren, Mobilität der Sensoren) ab. Das Netzwerkprotokoll befindet sich auf der zweiten Stufe und gewährleistet Kommunikation zwischen Sensoren durch Routingalgorithmen. Die dritte Ebene entspricht der Anwendung, welche die Sensoren steuert und mit Hilfe derselben ein bestimmtes Phänomen überwacht.

5.4 Kommunikationsmodell

Die Kommunikation in Sensornetzwerken setzt sich aus zwei Teilen zusammen. Die Anwendungskommunikation umfasst Sensordaten im weiteren Sinne, welche zum Beobachter übermittelt werden sollen. Die vom Netzwerkprotokoll generierte Infrastruk turkommunikation dagegen wird benötigt, um das Netzwerk aufzubauen und zu unterhalten.
Infrastrukturkommunikation spielt demnach vor allem in der Anfangsphase eine wichtige Rolle. Später wird diese Art der Kommunikation benötigt um das Netzwerk bei Topologieränderungen durch mobile Knoten oder Knotenausfall zu rekonfigurieren. Schliesslich wird Infrastrukturkommunikation auch für Optimierungszwecke eingesetzt. Es kann sich lohnen zusätzliche Zeit und Energie in Infrastrukturkommunikation zu investieren um späteren Datenverkehr zu minimieren und die Netzwerkeffizienz zu steigern.

5.5 Data Delivery

Sensornetzwerke lassen sich durch die Ansprüche des Beobachters an den Datenaustausch innerhalb der Anwendungskommunikation klassifizieren.

- Continuous data delivery
  Die Sensoren übermitteln Messwerte mit einer festgelegten Frequenz.

- Event-driven data delivery
  Die Sensoren übermitteln Messwerte als Reaktion auf bestimmte Ereignisse.

- Observer-instantiated data delivery
  Die Sensoren übermitteln Messwerte auf Anfrage des Beobachters.

- Hybrid data delivery
  Die drei genannten Modelle lassen sich konbinieren, wobei man dann von einem hybriden System spricht.

Für jedes Modell sind drei Formen des Routings möglich. Flooding (an alle), Multicast (an mehrere) oder Unicast (an einen).

5.6 Netzwerkdynamik

Ein Sensornetzwerk bildet einen Pfad zwischen Phänomen und Beobachter. Das Problem dieses Pfadaufbaus ist vergleichbar mit demjenigen des Routings in Ad-hoc-Netzwerken [10]. Ein Unterschied besteht in der individuellen Adressierung einzelner Knoten, welche in Sensornetzwerken nicht zwingend notwendig ist. Im Gegensatz zu Knoten eines Sensornetzwerkes nehmen diejenigen innerhalb des Pfades eines Ad-hoc-Netzwerkes eine passive Rolle ein; Ihre Aufgabe besteht lediglich im Weiterleiten von Informationen.
Man unterscheidet zwischen statischen und dynamischen Sensornetzwerken.

- statische Sensornetze
  In statischen Sensornetzwerken gibt es keine Bewegung zwischen Sensoren, Beobachter(n) und dem Phänomen. Der Einsatz von lokalisierten Algorithmen eignet sich hier besonders gut: Dabei fassen ausgewählte Sensorknoten die Beobachtungen ihrer Umgebungssensoren in einer oder mehreren Hierarchiestufen zusammen. Solche Algorithmen können die Lebenszeit des Sensornetzwerkes beträchtlich verlängern, da das Netzwerk nur in der initialen Phase Infrastrukturkommunikation benötigt und sich der weitere Datenverkehr auf Anwendungskommunikation beschränkt.

- dynamische Sensornetze
  Bei dynamischen Netzen sind entweder die Sensoren, der Beobachter oder das Phänomen mobil. Dynamische Sensornetzwerke lassen sich in drei Gruppen klassifizieren:

  – mobiler Beobachter
    Der Beobacher bewegt sich relativ zu den Sensoren und dem Phänomen. Ein vorstellbares Szenario würde darin bestehen, dass ein Beobachterflugzeug verwendet wird, um in unzugänglichen Gebieten eingesetzte Sensoren zu kontaktieren.

  – mobile Sensoren
    Die Sensoren sind in Bewegung, während Beoachter und Phänomen am selben Ort verweilen. Eine Beispielsituation könnte sich ergeben, wenn an Fahrzeugen angebrachte Sensoren verwendet werden um die Verkehrsdichte zu messen.
    Bewegen sich Sensorknoten zwischen Phänomen und Beobachter, müssen möglicherweise neue Pfade gesucht werden. Entweder der betroffene Knoten oder der Beobachter muss diesen Prozess einleiten. Man unterscheidet zwischen reaktiven, proaktiven oder hybriden Routingalgorithmen. Im Gegensatz zu reaktiven Algorithmen (beispielsweise Dynamic Source Routing [26, 27]), die den Pfad beim Versenden von Daten aufbauen, tauschen

proaktive Algorithmen ständig Informationen aus und erkennen entweder die aktuelle Netztopologie (Link State Routing) oder führen eine Entfernungstabelle (Distance Vector Routing[41, 42]). Hybride Systeme verbinden beide Methoden, indem sie lokal proaktive Algoritmen einsetzen, bei Anfragen zu weiter entfernten Knoten jedoch reaktiv vorgehen.

– mobiles Phänomen
Ein typischen Beispiel für den Einsatz von Sensornetzwerken bei mobilen Phänomenen sind Tierdetektoren. Es ist ausreichend Sensoren in unmittelbarer Umgebung des Phänomens, abhängig von der gewünschten Genauigkeit, Latenz oder Energieeffizienz, zu aktivieren.

An dieser Stelle sei nochmals betont, dass die Mobilität in traditionellen drahtlosen Ad-hoc-Netzwerken sich von solcher in Sensornetzwerken unterscheidet. In Sensornetzwerken gilt das Interesse des Beobachters nicht unbedingt einzelnen Sensoren und damit auch nicht deren Bewegung. Ein Sensornetzwerk muss sich demnach unter Berücksichtigung des Phänomens anpassen; Sensoren die sich davon entfernen, könnten beispielsweise ihre Verantwortlichkeit selbstständig anderen Sensorknoten übertragen.

5.7 Anwendungszenario: Temperaturüberwachung

In diesem Kapitel wird ein Beispielszenario vorgestellt, für welches ein Sensornetzwerk konzipiert und implementiert werden soll.
Europäische Hygieneverordnungen schreiben im Bezug auf den Transport von tiefgefrorenen Lebensmitteln in Transportbehältern über $2m^3$ Lufttemperaturmessgeräte vor. Neben fehleranfälligen mechanischen Lösungen werden heute meist elektronische Temperaturschreiber eingesetzt. Temperaturdaten werden nach erfolgtem Transport über eine Infrarot-Schnittstelle auf einen Computer übertragen oder ausgedruckt. Dieses System besitzt den Nachteil, dass Temperaturüberschreitungen im schlimmsten Fall erst nach dem Transport erkannt werden und für Grosstransporte auf Wasser oder Schiene unpraktikabel ist. Über den Einsatz von Satellitenkommunikation und Solarpanels zur Energieversorgung wurde schon vor Jahren diskutiert, doch trotz der grossen Vorteile ist eine solche Anschaffung für kleine Transportunternehmen zu kostspielig.
BTnodes könnten hier Abhilfe schaffen: Mit Temperaturfühlern ausgestattet werden sie als Sensoren eingesetzt und übermitteln ihre Messungen periodisch an einen mit GSM ausgerüsteten Beobachterknoten. Im Folgenden werden Eigenschaften und Anforderungen an ein solchen Systems aufgezählt:

- In Güterwaggons und anderen Transporteinrichtungen steht oftmals keine zentrale Energieversorgung zur Verfügung und der Aufwand, Akkus regelmässig nachzuladen ist vor allem bei grossen Wagenparks enorm. Die Sensoren sollen also über eine unabhängige Enerieversorgung verfügen und möglichst lange wartungsfrei arbeiten können.

- Geringe Abmessungen der Sensorknoten ist eine Voraussetzung für den Einsatz in kleindimensionierten Transportbehältern.

- Der Beobachterknoten befindet sich an gut zugänglicher Stelle mit genügend Platz für einen grösser dimensionierten und leistungsfähigeren Akku. Beispiels-

weise wäre es möglich einzelne Güterwaggons oder Lastkraftwagen damit auzurüsten.

- Warnungen und Messwerte sollen über grosse Distanzen an eine zentrale Stelle übermittelt werden können.
- Das System muss verlässlich und fehlertolerant arbeiten, ein aufgebrauchter Sensorakku darf nicht das gesamte Netz funktionsuntüchtig werden lassen.
- Die Skalierbarkeit muss gewährleistet sein, da beispielsweise in Lagerhäusern Kühlboxen zu Hunderten auf engstem Raum zwischengelagert werden.
- Temperaturveränderungen geschehen in der Regel sehr langsam. Es ist daher ausreichend, Temperaturmessungen in relativ grossen Intervallen vorzunehmen, die Latenz der Nachrichtenübermittlung ist unerheblich.
- Die Datenmenge ist bei Temperaturmessungen vergleichsweise gering.
- Transportbehälter verweilen während den langen Transportwegen und in Lagern am selben Ort. Das vorliegende Sensornetzwerk ist also eher von statischem Typ, obwohl Topologieänderungen auftreten können. Das Phänomen ist ebenfalls statisch, da Sensoren ihre Umgebungstemperatur messen.

Kapitel 6

Reduktion des Energieverbrauchs in drahtlosen Netzwerken

6.1 Einleitung

Es existiert eine Vielzahl von Ansätzen, die Betriebsdauer von Netzen mit batteriebetriebenen, drahtlos kommunizierenden Geräten zu verlängern. Grundsätzlich kann davon ausgegangen werden, dass der Energieverbrauch von sendenden und empfangenden Knoten beträchtlich höher ist als jener von nichtkommunizierenden.
Darauf zugeschnittene Methoden verteilen den Energieverbrauch in Netzwerken durch geschicktes Routing so, dass der Ausfall einzelner Knoten herausgezögert wird. Man spricht dabei von Maximum-Lifetime-Routing (MLR). MLR kann insbesondere in Verbindung mit variablen Sendeleistungen sehr effizient sein.
Ein anderer Ansatz besteht darin, Netzknoten zeitweise auszuschalten, womit auch deren Kommunikationsfähigkeit erlischt. Dies verkleinert den Gesamtenergieverbrauch und kann die Betriebsdauer des Netzes erheblich verlängern.
Ein dritter Ansatz beruht darauf, die Funktionalität der Knoten einzuschränken, die Kommunikationsbereitschaft aber zu erhalten. Beispielsweise kann der Strombedarf durch erniedrigen der Taktrate oder der Sendeleistung verringert werden.
In diesem Kapitel wird auf die Methoden, welche in [19] Erwähnung finden, und weitere eingegangen. Ihre Funktionsweise wird erläutert und ihre Eignung bezüglich der BTnode-Plattform resp. dem gewählten Szenario (Kapitel 5.7) analysiert.

6.2 Maximum Lifetime Routing

Beim Maximum-Lifetime-Routing [11, 12] sind zu jedem Zeitpunkt alle Knoten aktiv. Da durch den Ausfall von wenigen Knoten die Funktionsweise des gesamten Netzwerks in Mitleidenschaft gezogen werden kann, sollen solche Ausfälle möglichst lange hinausgezögert werden. Es geht also darum, mit Hilfe von Routingprotokollen Knoten mit geringeren Energiereserven weniger zu beanspruchen als solche mit grösseren oder Pfade zu finden, auf welchen weniger Knoten involviert sind. Es existieren drei Metriken für das maximum-lifetime-Routing.

- Minimum-Energy-Routing
  Die totale Energie, die benötigt wird, um ein Paket entlang einer Route zu übertragen, soll minimiert werden. Minimum-Energy-Routing maximiert nicht die Laufzeit eines Netzwerkes, da die verfügbare Energie der einzelnen Knoten vernachlässigt wird. Im Falle von konstanten Übertragungsleistungen entspricht minimum-energy-Routing einem Routing über den kürzesten Pfad.

- Max-min Routing
  Max-min routing wählt eine Route, welche die minimale verbleibende Energie der involvierten Knoten maximiert. Solche Pfade können länger sein und daher einen grösseren Energieverbrauch aufweisen als minimum-energy-Routen. Es ist auch möglich, an Stelle des Wertes der verbleibenden Energie eine andere Kostenfunktion einzusetzen.

- Minimum Cost Routing
  Minimum-cost-Routing Wählt eine Route, welche die Summe der Kosten minimiert. Die Form der Kostenfunktion ist dabei entscheidend. Generell ist sie monoton zunehmend mit sinkender Restenergie. Es werden also tendenziell Routen gewählt, die a) kurz sind und b) Knoten mit grösserer Energiereserve beanspruchen. Abbildung 1.9. in [19] gibt Aufschluss darüber, welche Funktionen geeignet sind und welche weniger.

- Battery Efficient Routing
  Eine weitere Metrik ist das sogenannte battery efficient routing [58]. Dieses Verfahren baut auf der Tatsache auf, dass Batterien eine längere Laufzeit aufweisen, wenn sie über kurze Zeiträume stark belastet werden (recovery-Effekt). Das elektrochemische Verhalten kann aber zwischen unterschiedlichen Batterietypen stark variieren, was die Effizienz des Algorithmus beeinträchtigt. Auf diese Art des Routings wird daher nicht weiter eingegangen.

Simulationsergebnisse in einem Netzwerk mit 20 Knoten zeigten, dass mit minimum-cost-Routing und einer geeigneten Kostenfunktion durchschnittlich 84-96% der optimalen Laufzeit erreicht werden. Max-min routing schnitt annähernd gleich gut ab. Mit minimum-energy-Routing wurde nur etwa 30-50% der optimalen Laufzeit erreicht [11, 12].
Bei Conditional Min Max Battery Capacity Routing (CMMBCR) werden minimum-energy-Routing und max-min-Routing über einen Schwellenwert für die minimale verfügbare Energie aller Knoten entlang der Route kombiniert, was jedoch keine erheblichen Verbesserungen brachte [18].
Max-min-Routing, minimum-cost-Routing und battery-efficient-Routing gehen davon aus, dass die verbleibende Batterieladung bekannt ist. Ist keine entsprechende Hardware zur Messung auf den Knoten vorhanden, könnte die Akkumulation der Sende- , Empfangs- und Standby-Dauer in Verbindung mit einem Energiemodell, wie es beispielsweise in [17] vorgestellt wird, Abhilfe schaffen.
Beim max-min-Routing und dem minimum-cost-Routing macht der Einsatz von regelmässig aktualisierten Tabellen mit Energieinformationen in allen Knoten und Berechnungen für Distance Vector Routing wenig Sinn. Ein besserer Ansatz besteht darin, diese Metriken in der route discovery-Phase eines on-demand Routingprotokolls (Bsp. AODV [42]) zu verwenden.

6.3 Energiesparende Protokolle

Das Prinzip der nachfolgenden Energiesparprotokolle beruht auf der temporären Deaktivierung bestimmter Netzknoten. Für das Funktionieren der Protokolle wird vorausgesetzt, dass sich die Knoten selbstständig aus- und über einen externen Timer getriggert wieder einschalten können. Eine Alternative ist das zeitweise Ausschalten des Kommunikationsmoduls. Im inaktiven Zustand sind Netzknoten weder sichtbar noch ist es möglich, sie durch Kommunikationsanfragen anderer Knoten aufzuwecken.

6.3.1 Klassifizierung

Drei Typen von Energiesparprotokollen unterscheidet man:

- Synchrone Energiesparprotokolle
  Die Netzknoten befinden in einem stromsparenden sleep-Zustand und wachen gemeinsam von Zeit zu Zeit auf um allenfalls anstehende Kommunikation auszuführen. Das Hauptproblem dieses Mechanismus ist die Synchronisation aller beteiligten Knoten.

- Topologiebasierte Energiesparprotokolle
  Für das Netzwerk wird ein covering set (siehe Kapitel 6.3.2) gewählt. Darin enthaltene Knoten bleiben dauernd aktiv während die restlichen im sleep-Modus verweilen und diesen gelegentlich zur Kommunikation verlassen - dies kann sowohl synchron als auch asynchron geschehen. Um die Lebenszeit des Netzwerkes zu maximieren, muss das covering set regelmässig rotiert werden.

- Asynchrone Energiesparprotokolle
  Die Knoten agieren unabhängig und besitzen unterschiedliche sleep-wake-Schedules. Regeln innerhalb der Protokolle garantieren, dass sich Schedules benachbarter Knoten überlappen und gewährleisten somit gemeinsame Kommunikation.

6.3.2 Funktionsweise

Im Folgenden wird die Funktionsweise der wichtigsten Energiesparprotokolle genauer erläutert [19].

IEEE 802.11, Synchrones Energiesparprotokoll

Dieses Protokoll [25] setzt die Phasensynchronisation aller Teilnehmer voraus. Es muss also ein globaler sleep-wake-Schedule existieren, damit alle Parteien gleichzeitig aufwachen resp. gleichzeitig in einen energiesparenden Zustand übergehen. Ein ausgewählter Knoten initiiert die Synchronisation, welche dezentralisiert aufrechterhalten wird.
Diese Methode ist weniger für dynamische Ad-hoc-Netzwerke geeignet, da das Zusammenschliessen von Teilnetzen eine gemeinsame Synchronisation voraussetzt oder einen Vermittlerknoten erfordert. Ein weiteres Problem ist die Latenz, welche eine Synchronisation erschwert. Die Lösungen dazu sind sehr komplex und mit viel Overhead verbunden. Obwohl Experimente zeigen, dass IEEE 802.11 keine erheblichen Energieeinsparungen mit sich bringt, sei das Prinzip im Folgenden kurz erläutert.
Alle beteiligten Knoten besitzen ein gemeinsames, synchronisiertes Intervall, das sogenannte beacon-Intervall. Zu Beginn dieses Intervalles wachen alle Teilnehmer auf,

verschicken eine beacon-Nachricht und synchronisieren sich mit der ersten erhaltenen beacon-Nachricht. Danach werden Kommunikationsanfragen mit Hilfe von ATIM (Ad Hoc Traffic Indication Message) und zugehörigen Bestätigungsnachrichten ausgetauscht. Die Zeitspanne dafür ist begrenzt auf das ATIM-Intervall und alle Knoten, welche keine Kommunikationsanfrage oder Bestätigungsnachricht erhalten haben, fallen in den sleep-Zustand zurück. Die verbleibenden Kommunizieren gemäss ATIM.
Eine Schwierigkeit in der Umsetzung besteht unter anderem darin, geeignete Werte für das beacon- und das ATIM-Intervall zu wählen.

SPAN

SPAN [13] gehört in die Klasse der topologiebasierten Energiesparprotokolle. In einem ersten Schritt geht es darum, ein connected covering set (Abbildung 6.1) zu finden, welches eine Art Backbone des Netzwerkes bilden wird.

> Ein covering set (CS) ist eine Teilmenge der Knotenmenge, so dass jeder Knoten entweder darin enthalten ist oder mindestens einen Nachbarn besitzt, der es ist.
>
> Ein connected covering set (CCS) ist ein covering set, wobei zwischen jedem Knotenpaar ein Pfad existiert, welcher nur über im covering set enthaltene Knoten führt.

Das Problem, das kleinste CCS zu ermitteln ist NP-hart, der vorgestellte Algorithmus sucht daher nur nach einer approximativen Lösung.
Der Algorithmus unterscheidet zwischen sogenannten coordinators - also jenen Knoten, die zum ccs gehören - und normalen Knoten. In einem ersten Schritt wird ermittelt, welche Knoten potentiell eine coordinator-Funktion übernehmen können. In einem zweiten Schritt scheiden überflüssige Knoten aus der Koordinatormenge aus.
Knoten verschicken HELLO-Nachrichten, um ihre 2-Hop-Nachbarschaft zu entdecken. Bemerkt ein Knoten, dass zwischen zwei seiner direkten Nachbarn keine Verbindung besteht, oder die zwei Nachbarn nicht bereits über einen Koordinator verbunden sind, so markiert er sich als potentieller Koordinator und wartet auf eintreffende coordinator-Ankündigungen. Möglicherweise wird seine Aufgabe bereits von einem anderen Knoten übernommen und seine Rolle als Koordinator verfällt.
Besteht die Markierung nach Ablauf eines backoff-Intervalles noch immer, wird er sich mit einer entsprechenden Nachricht definitiv als coordinator ankündigen. Der Länge dieses backoff-Intervalles kommt eine besondere Bedeutung zu. Das Intervall wird für jeden Knoten individuell bestimmt und ist nicht nur zufällig, sondern auch abhängig von der Konnektivität sowie den verfügbaren Energiereserven. (Je grösser die Konnektivität und je grösser die Energiereserve, desto kleiner das Intervall). Die Wahrscheinlichkeit, dass ein Knoten definitiv zu einem coordinator wird, steigt mit abnehmender Intervalllänge.
Die Knoten durchlaufen diesen Algorithmus regelmässig, wodurch die coordinator-Rolle rotiert und die Energiereserven gleichmässig geschont werden.
Die Koordinatoren bilden ein aktives Backbone und kommunizieren mit den übrigen, im Schlafzustand verweilenden Knoten gemäss IEEE 802.11. Die Synchronisation

zwischen coordinator- und nicht-coordinator-Knoten ist dabei trivial, da die Koordinatoren immer wach sind. Für die Kommunikation zwischen Koordinatoren wird aus dem selben Grund kein spezielles Protokoll benötigt.

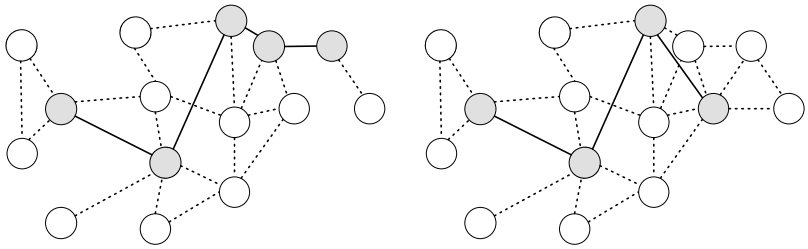


Abbildung 6.1: Die Grösse des covering set (grau) ist abhängig von der Dichte des Netzwerkes

GAF

GAF [63, 64] steht für Geographic Adaptive Fidelity und gehört ebenfalls zu den topologiebasierten Energiesparprotokollen. Im Gegensatz zu SPAN werden jedoch nicht Konnektivitäten sondern Positionsdaten verwendet; Diese können beispielsweise von einem GPS-Empfänger stammen.
Der Algorithmus teilt das Netzwerk gitterförmig auf. Die Gittergrösse ist so gewählt, dass jeder Knoten mit jedem Knoten der benachbarten Zellen über eine direkte Verbindung kommunizieren kann. Somit beträgt die Gittergrösse $R/\sqrt{5}$, wobei R der Kommunikationsreichweite entspricht. Es reicht so aus, wenn nur ein Knoten pro Gitterfeld aktiv bleibt.
Jeder Knoten gelangt periodisch in den discovery-Modus, in welchem er Informationen über Energiereserven und Position broadcastet. Falls er von keinem Knoten mit höheren Energiereserven Antwort erhält, bleibt er aktiv, ansonsten fällt er zurück in den sleep-Modus.
Trotz der Einfachheit dieses Algorithmus ist Vorsicht geboten, da Entfernungen nicht immer mit bestehenden Verbindungen gleichgesetzt werden können. Hindernisse können leicht Verbindungen stören. Eine Lösung kann darin bestehen, mehrere Aktivknoten pro Gitterfeld zuzulassen.

BECA

Zu den asynchronen Energiesparprotokollen gehört BECA (Basic Energy Conservation Algorithm). Die Knoten agieren völlig unabhängig voneinander, somit kann nicht à priori garantiert werden, dass ein aktiver Pfad zum routing-Ziel existiert. Regeln stellen aber eine Kommunikation - wenn auch mit grosser Latenz - sicher. BECA und AFECA sind dem AODV-Routing (Ad-hoc On-demand Distance Vector Routing) [42] ähnlich. Im folgenden Abschnitt wird daher das Funktionsprinzip von AODV kurz erläutert.

> AODV ist ein reaktiver multihop-Routingalgorithmus. Vier Pakettypen werden unterschieden:
>
> - HELLO-Nachricht, um Nachbarn zu erkennen

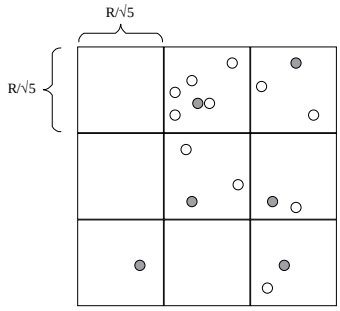


Abbildung 6.2: Bei GAF genügt ein aktiver Knoten pro Gitterfeld

- Routing-request-Nachricht (RREQ) um das Routing einzuleiten
- Routing-reply-Nachricht (RREP) um eine Route zu bestätigen
- Routing-error-Nachricht (RERR) um Fehler zu markieren

Angenommen, ein Knoten A möchte mit einem Knoten D kommunizieren und eine Route ist noch nicht bekannt, so schickt der Ausgangsknoten A eine Nachricht RREQ an alle seine Nachbarn. RREQ enthält sowohl die Kennung des Anfangsknotens ('A') als auch diejenige des Zielknoten ('D'). Indem jeder Knoten die Nachricht weiterleitet, wird das Netz geflutet. Eine Paket-Sequenznummer verhindert mehrfaches Weiterleiten. Gelangt RREQ ans Ziel D, so Antwortet D mit einer RREP-Nachricht, welche auf der umgekehrten Route zu A zurückpropagiert wird. Knoten, welche eine RREP-Nachricht weiterleiten generieren zusätzlich Einträge in der Routingabelle, um zukünftig erhaltene Pakete richtig zu routen. Trifft RREP bei A ein, so kann dieser mit der Kommunikation beginnen. Der Algorithmus optimiert zusätzlich den request-Prozess indem RREQ-Nachrichten von Knoten mit Topologiekenntnissen beantwortet weden können, bevor sie beim Zielknoten angelangt sind.

Beim BEC-Algorithmus besitzt jeder Knoten drei interne Zustände: sleep, listening und active. Im listening-Zustand kann ein Knoten nur Pakete empfangen, im active-Zustand zusätzlich senden. (Knoten können vom sleep-Zustand in den listening-Zustand gelangen und umgekehrt. In den active-Zustand gelangen sie von listening aus und der Folgezustand von active ist immer sleep.) Listening- und sleep-Intervalle wechseln sich ab, wobei die Länge s des sleep-Intervalles ein ganzzahliges positives Vielfaches des listening-Intervalles l ist: $s = k * l, k \in N$.
Der Verbindungsaufbau zu einem entfernten Knoten geschieht analog zu AODV. Um einen routing request zu senden werden jedoch pro Hop maximal $k + 1$ Broadcasts benötigt. Jeder Knoten, welcher einen routing request erhalten hat, wechselt vom listening in den aktiven Zustand. Ist das Timeout eines Knotens grösser als das RREQ retry interval, und ist er nicht an der Verbindung beteiligt (d.h. hat er keine RREP-Nachricht erhalten), so geht er zurück in den sleep-Zustand. Dieser Prozess des Verbindungsaufbaus benötigt maximal $d * (k + 1)$ Zeiteinheiten, wobei d der Pfadlänge in

Hops entspricht. Auf Abbildung 6.3 ist dieser Vorgang dargestellt. Die restlichen Knoten bleiben aktiv und übertragen die Nutzdaten, danach begeben sich alle beteiligten Knoten in den energiesparenden sleep-Zustand.

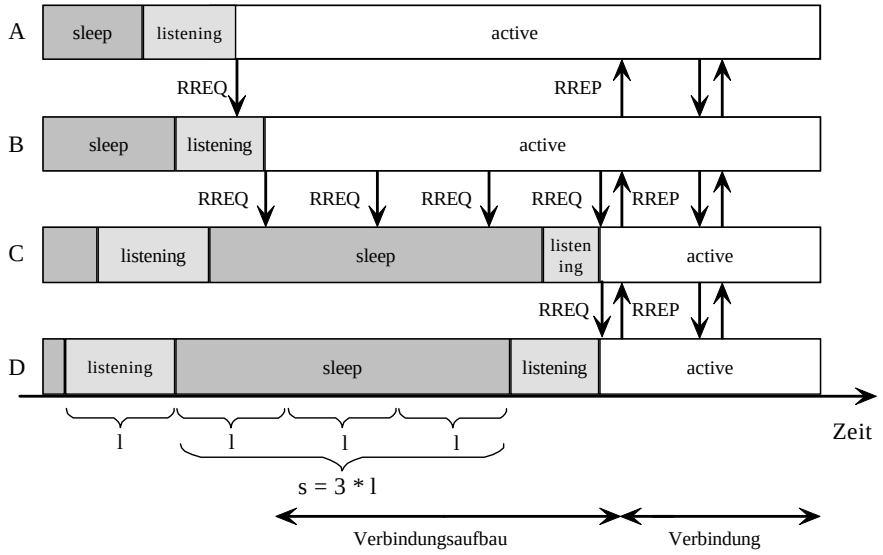


Abbildung 6.3: Verbindungsaufbau beim BECA Algorithmus

AFECA

AFECA (Adaptive Fidelity Energy Conserving Algorithm) ist eine Erweiterung von BECA, wobei Knoten die Länge des sleep-Intervalles an die Netzwerkdichte anpassen. Je dichter das Netz ist, desto länger dürfen die Knoten im inaktiven Zustand verweilen. Der Grund dafür ist, dass mit der Netzwerkdichte die Anzahl Wege zum Zielknoten zunimmt und damit auch die Wahrscheinlichkeit, dass schneller ein solcher gefunden wird. Eine einfache Methode ist, die Anzahl N von Nachbarn zu zählen und die Länge des sleep-Zyklus zufällig im Intervall zwischen 1 und N zu wählen. Simulationsergebnisse zeigen, dass die Verlustrate von Paketen bei AFECA leicht höher ist als bei BECA und der Energieverbrauch um wenige Prozent sinkt.

6.4 Anpassen der Sendeleistung

Die Anpassung der Übertragungsleistung in drahtlosen Netzwerken verfolgt zwei Ziele. Auf der einen Seite sollen unnötige Interferenzen vermieden und so die Netzwerkkapazität erhöht werden, auf der anderen Seite wird durch verminderte Sendeleistung Energie gespart.
Wie Abbildung 6.4 zeigt, ist die Netztopologie anhängig von der verwendeten Leistung. Steht keine Information über die Topologie zur Verfügung, kommen Heuristiken

zum Einsatz. In [44] wird bis zum Erreichen des erwarteten Grades die Sendeleistung sukszessive erhöht. Der in [59] vorgestellte Algorithmus erhöht die Leistung solange, bis Verbindungen zu Nachbarn unter verschiedenen Winkeln möglich werden.

Mit dem Verringern der Sendeleistung kann Minimum Energy Routing eingesetzt werden (siehe dazu auch 6.2). Diese Methode beruht auf dem Prinzip, dass die aufgewendete Gesamtenergie zur Überbrückung mehrerer kurzer Distanzen kleiner ist als eine entsprechende über längere Strecken. In Beispiel 6.4 werden Daten anstatt über eine direkte Verbindung von A nach E bevorzugt von A über B, C, D zu E mit niedrigeren Leistungen gesendet.

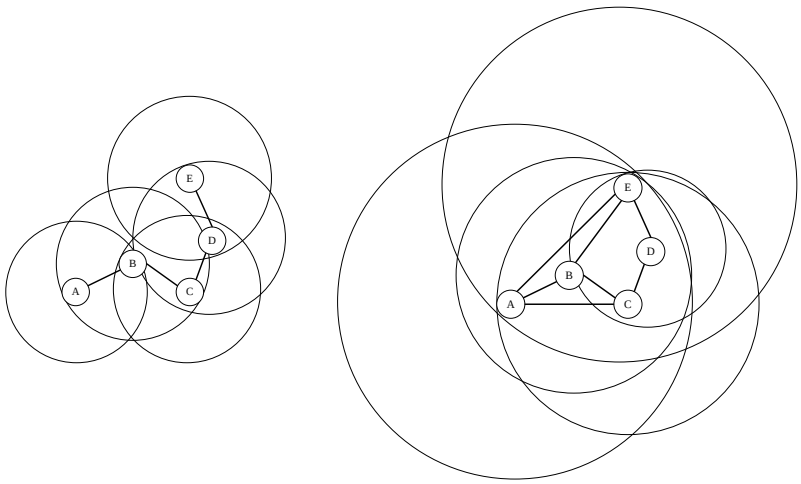


Abbildung 6.4: Die Topologie hängt von der Sendeleistung ab

6.5 Eignung der Stromsparprotokolle und Routingalgorithmen für die BTnode-Plattform

6.5.1 Maximum Lifetime Routing

Maximum-lifetime-Routing ist grundsätzlich auf der BTnode-Plattform realisierbar. Mit Treenet und Xhop existiert ein multihop-Routing, auf welchem aufbauend ein maximum-lifetime-Routing implementiert werden könnte. Der hohe Energieverbrauch im standby-Modus, ein Hauptproblem beim Einsatz von BTnodes, wird damit aber nicht angegangen. Daher kann der alleinige Nutzen von Routing-Algorithmen in Sensornetzen des Beispielszenarios eher als gering eingestuft werden.
In Kombination mit topologiebasierten Energiesparprotokollen wie beispielsweise SPAN macht MLR keinen Sinn, da diese Algorithmen durch Rotation des covering set bereits eine Überbeanspruchung von Knoten mit niedriger Batterieladung vermeiden.
Die Implementation von maximum-lifetime-Routing auf proaktiv gerouteten Netzen mit asynchronen Energiesparprotokollen ist ebenfalls wenig sinnvoll. Der durch das Energiesparprotokoll reduzierte Energieverbrauch wird durch den zusätzlichen Daten-

austausch zur Erstellung von aktualisierten Tabellen mit Energieinformationen wettgemacht.
Der Einsatz von MLR kann nur in Verbindung mit synchronen Protokollen oder mit reaktivem Routing empfohlen werden. Die BTnodes besitzen eine Möglichkeit zur Messung der Restladung, die für eine Kostenfunktion eingesetzt werden könnte.

6.5.2 Energiesparprotokolle

IEEE 802.11, Synchrones Energiesparprotokoll

Die erwartete Effizienz von synchronen Energiesparprotokollen in Verwendung mit BTnodes kann als hoch eingestuft werden. Alle Sensorknoten verbringen die meiste Zeit in einem inaktiven Zustand. Synchronisation lohnt sich überdies besonders gut statische Netzwerke, wie das des Beispielszenarios. Im Gegensatz zu dynamischen Netzwerken entfällt so die Resynchronisation von Knoten.
Die komplette Phasensynchronisation von Sensornetzen ist unter Umständen schwierig zu erreichen, die Synchronisation einzelner Bereiche kann für unsere Zwecke allerdings bereits genügen.

SPAN

Eine Implementation von SPAN auf der BTnode-Plattform wäre - eingeschränkt durch die Eigenschaften von Bluetooth-Scatternetzen und der Voraussetzung eines flooding-Algorithmus für den coordinator election Prozess - sehr aufwändig zu realisieren. Für den Paketaustausch innerhalb des Backbone kann jedoch auf das bereits vorhandene Xhop in Verbindung mit TreeNet zurückgegeriffen werden.
Ein einfacherer Ansatz könnte darin bestehen, den TreeNet-Algorithmus dahingehend zu modifizieren, dass die Blattknoten des Baumes vom restlichen Netzwerk getrennt werden und der verbleibende Teilbaum das connected covering set bildet. Diese Methode lässt bei der Bildung des Baumnetzes die verbleibende Batteriekapazität der Sensorknoten ausser Acht, was schnell zu Knotenausfällen führen würde. Eine Anpassung des Treenet-Algorithmus wäre also auch in dieser Hinsicht nötig.
Fraglich ist, ob ein SPAN-Algorithmus in BTnode-Sensornetzwerken überhaupt wünschenswert ist. Je nach Netzwerkdichte umfasst das covering set von SPAN mehr oder weniger Sensorknoten. Diese verweilen die gesamte Zeit in einem aktiven Zustand, was besonders im Falle der BTnodes mit hohem standby-Stromverbrauch ungünstig ist.

GAF

Geographic Adaptive Fidelity setzt eine Möglichkeit voraus, die absolute Position der Knoten zu bestimmen. BTnodes verfügen über keine Funktionalität, die sich für eine solche Positionsbestimmung eignen würde. GAF ist auf der BTnode-Plattform daher nicht realisierbar.

BECA

BECA ist grundsätzlich auf der BTnode-Plattform umsetzbar. Es kann sowohl ein reaktiver Routingansatz (der Sender sucht eine Route) als auch ein proaktiver (die Knoten

aktualisieren Routingtabellen) angewendet werden, wobei für multihop-Pakete wiederum XHOP zum Einsatz kommen kann. Im Hinblick auf unser Ausgangsszenario, das grosse Nachrichtenlatenzen erlaubt, wäre die Verwendung des Basic Energy Conserving Algorithm gut denkbar und einfach zu implementieren. Trotzdem nützt der Algorithmus die Statik des Sensornetzwerkes nicht aus, selbst regelmässig benützte Verbindungen müssen jedesmal zeitaufwändig und energieintensiv erstellt werden. Berechtigte Zweifel an der Effizienz des Algorithmus sind angebracht.

AFECA

Für AFECA gelten analoge Überlegungen wie für BECA.

6.5.3 Reduktion der Sendeleistung

Das Bluetooth-Modul der BTnodes bietet keine Möglichkeit zu Anpassung der Sendeleistung, diesbezügliche Methoden sind nicht realisierbar.

6.5.4 Weitere stromsparende Massnahmen

Die Taktrate des ATmega128L-Controllers kann während der Laufzeit verändert werden, überdies besitzt er diverse Stromspar-Modi. Diese Möglichkeiten werden in Ka pitel 3.2.2 vorgestellt, und in Kapitel 8.2.2 eingehend untersucht.

6.5.5 Überblick

Die folgende Tabelle zeigt alle erklärten Stromsparmethoden nochmals im Überblick. Wertung: $--$ für sehr schlecht/sehr grosser Aufwand, $-$ für schlecht/aufwändig, 0 für normal/normaler Aufwand, $+$ für gut/geringer Aufwand, $++$ für sehr gut/äusserst geringer Aufwand.

	MLR	Synchr.	SPAN	GAF	BECA, AFECA	Reduktion. d. Sendeleistung	Gerätespez. Methoden
Erwartete Effizienz allgemeine ad hoc-N.	$+$	$+$	$++$	$++$	$+$	$+$	$+$
Erwartete Effizienz Sensornetzszenario	$--$	$++$	$-$	$-$	0	0	$+$
Eignung Plattform BTnode, Bluetooth	0	$+$	$--$	n/a	$+$	n/a	$+$
Aufwand Realisierung auf BTnode Plattform	$-$	0	$--$	n/a	$+$	n/a	$+$
Fazit	$-$	$++$	0	n/a	$+$	n/a	$+$

Kapitel 7

Implementation

7.1 Algorithmus I: Synchronized SensorNet

In diesem Unterkapitel wird ein auf das vorgestellte Szenario (Kapitel 5.7) zugeschnittener Algorithmus vorgestellt, der auf der BTnode-Plattform implementiert und getestet worden ist.
Das Szenario geht davon aus, dass an Transportbehältern angebrachte Sensorknoten laufend Temperaturmessungen durchführen und diese an einen mit GSM ausgerüsteten Knoten senden. Die Sensorknoten sollten mit möglichst wenig Energie auskommen und das Sensornetzwerk sollte skalierbar und fehlertolerant sein. Die Netzwerklatenz isch unkritisch und die Datenmenge eher klein.

7.1.1 Idee

In unserem Sensornetz existieren zwei Arten von Knoten: Sensorknoten und Beobachterknoten (Observerknoten). Die Funktion jedes Knoten - Sensor oder Beobachter - ist von Anfang an festgelegt und unveränderlich. Sensorknoten werden in grossen Mengen eingesetzt, wogegen Beobachterknoten nur vereinzelt vorkommen. Sensorknoten haben die Aufgabe Temperaturmessungen zu sammeln und diese gelegentlich einem ausgewählten Beobachterknoten zu senden. Die Netztopologie garantiert, dass sich jeder Sensor in Reichweite mindestens eines Beobachtes befindet.
Zwischen den Übertragungen sollen die BTnodes möglichst Energiesparend arbeiten, dazu wird das BTmodul, welches selbst im standby-Betrieb einen relativ hohen Strombedarf aufweist aus- und und vor anstehender Kommunikation rechtzeitig eingeschaltet.
Obwohl die Energieversorgung bei Observern als nicht ganz so kritisch betrachtet werden kann, wird hier der gleiche Energiesparmechanismus eingesetzt. Schlaf- und Wachintervalle wechseln sich sowohl beim Sensor als auch beim Observer ab, Phasensynchronisation ist also nötig.
Die Sensoren suchen selbstständig nach einem Observer und tauschen mit diesem Nachrichten zur Phasensynchronisation aus. Sensoren reagieren auf Observerausfälle oder Kommunikationsschwierigkeiten, indem sie nach neuen Beobachtern suchen.
Der Observerknoten besitzt im aktiven Zustand zwei weitere Zustände: im Request-Zustand werden Sensordaten empfangen, im ready_to_sync-Zustand ist er bereit, Synchronisationsanfragen zu beantworten. Der Sensor besitzt ebenfalls zwei Untergeor-

gnete Zustände: Synchronizing, wo ein Beobachter gesucht wird und Synchronisationsanfragen gestellt werden und ready_to_send welcher die Bereitschaft signalisiert, als Reaktion auf DataReq-Nachrichten Sensordaten zu senden.

7.1.2 Algorithmus

Kommunikationsmodell

Als Kommunikationsmodell wird continuous data delivery eingesetzt. Temperaturveränderungen geschehen üblicherweise über längere Zeiträume, daher ist ein Alarmmechanismus im Sensor mittels event-driven data delivery überflüssig.
Die Sensorknoten sind in festgelegten Zeitabständen $t_{request}$ empfangsbereit: Nutzdaten werden aber nur gesendet, wenn eine Verbindung zum Observer besteht und eine entsprechende Anfrage empfangen worden ist. Bleiben Verbindung und Anfrage vom Observer aus, kann davon ausgegangen werden, dass ein Kommunikationsproblem vorliegt oder die Synchronisation ungültig ist. In diesem Fall versucht der Sensor sich mit dem Observer erneut zu synchronisieren, schlägt dies fehl wird ein neuer Observer gesucht.
Es existieren vier verschiedene Pakettypen:

- SynReq-Nachricht, die eine Synchronisation anfordert,
- SynResp-Nachricht, die eine Synchronisationsanfrage beantwortet,
- DataReq-Nachricht, die Sensordaten anfordert,
- DataResp-Nachricht, die Sensordaten enthält.

Zum Austausch von Synchronisationsnachrichten nehmen die Sensoren die Masterrolle ein, Abfragen der Sensordaten werden durch den Observer initiiert. Dies hat den Vorteil, dass Verbindungen zu mehreren Sensoren gleichzeitig geöffnet sein dürfen und Abfragen parallel stattfinden können. Die Wach-Phase des Beobachters wird dadurch verkürzt.

Suchen eines Observers

Drei Hauptprobleme spielen bei der Observersuche mit Inquiries eine Rolle:

1. Durch die Schlaf- und Wachintervalle der Observer sind diese nicht immer sichtbar.
2. Die Resultate von Inquiries sind nicht deterministisch. Es kann also keine Garantie abgegeben werden, dass selbst aktive Beobachterknoten gefunden werden.
3. Bei Inquiries werden auch Knoten gefunden, die keine Observerknoten sind.

Das erste Problem wird gelöst, indem garatiert wird, dass ein Observer in jedem Intervall $t_{request}$ während einer Mindestdauer t_{ready} sichtbar und bereit ist, Synchronisationsanfragen zu empfangen. Suchen Sensoren in regelmässigen Zeitabständen $t_{search\_observer} \leq t_{ready}$ so ist gewährleistet, dass nach $\frac{t_{request}}{t_{search\_observer}}$ Inquiries mindestens eine Inquiry ausgeführt wurde während ein Observer aktiv war.
Dem zweiten Problem wird begegnet, indem Inquiries solange wiederholt werden, bis

ein Observer gefunden ist. Die Wahrscheinlichkeit, dass aktive Geräte gefunden werden steigt mit der Länge der ausgeführten Inquiries und deren Häufigkeit. Versuche haben gezeigt, dass kein linearer Zusammenhang zwischen Anzahl erkannter Geräte und der Länge des Inquiry erwartet werden kann [50]. Dazu kommt, dass diese Bluetooth-Operation zu den energieintensivsten gehört, was kurze Inquiries im Bereich von 3-5 Sekunden nahelegt.
Die Lösung des dritten Problems besteht in einer Identifikation der Observerknoten. Jedes Bluetooth-Gerät gehört einer Geräteklasse an, einer sogenannten Class of Device (CoD). Einer Inquiry kann eine solche CoD mitgegeben werden, worauf nur Geräte dieser Klasse antworten und andere Gerätetypen verborgen bleiben. Allgemeinen Inquiries wird ein GIAC (General Inquiry Access Code) mitgegeben, alle erreichbaren Bluetooth-Geräte können damit erkannt werden. Das Bluetooth-API der BTnodes bietet keine Möglichkeit zur Definition des CoD innerhalb von Inquiries. Wird eine verbose-Inquiry[1] durchgeführt, ist die jedem Gerät zugehörige CoD zugänglich, was das Öffnen von Verbindungen zu nicht-Observerknoten vermeiden lässt.
Jede gefundene Geräteadresse wird in eine Liste aufgenommen, wo auch der aktuelle Verbindungszustand, Gerätetyp (Sensor, Observer, unbekanntes Gerät), Handles und weitere Informationen gespeichert werden können. Ist bereits ein Observer erkannt worden, wird ein L2CAP-Kanal zu ihm geöffnet und damit die Synchronisation eingeleitet.

Synchronisation

Es bietet sich an, die Echtzeituhr der BTnodes zur Synchronisation von Beobachter und Sensorknoten zu verwenden. Die Echtzeituhr zählt die Millisekunden seit dem Einschalten des Gerätes und wird durch einen Reset auf null zurückgestellt. Das rtc-API beeinhaltet Methoden um die Echtzeituhr zu setzen und auszulesen. Ein Verstellen der Uhr wird vermieden, da dies Einfluss auf gesetzte Timeouts und programminterne Zeitreferenzen nähme. Fortan kommt einer Shift-Variable rtc_{shift} die Aufgabe zu, die Differenz zwischen synchronisierter und lokaler Echtzeituhr zu speichern. Synchronisation erfolgt über Zuweisung von rtc_{shift} und indem sich beide Parteien auf ein gemeinsames Abfrageintervall $t_{request}$ einigen.
Da mehrere Sensoren mit dem Beobachter synchronisiert werden, kommt nur eine sensorseitige Anpassung der Echtzeituhr des Intervalles $t_{request}$ in Frage. Die Aufwachbedingung des Sensors ist nun einfach formuliert: [2]

$$(RTC_S + rtc_{shift})\ mod\ t_{request} = 0 \qquad (1)$$

und diejenige des Observers:

$$RTC_O\ mod\ t_{request} = 0 \qquad (2)$$

RTC_S	Wert der Echtzeituhr des Sensors
RTC_O	Wert der Echtzeituhr des Observers
rtc_{shift}	Verschiebung der Echtzeituhr
$t_{request}$	Abfrageintervall für Sensordaten

[1] mittels btn_bt_inquiry_verbose()

[2] Die Überprüfung wird mittels Polling implementiert. Ein Überlauf von $(RTC_S + rtc_{shift})\ mod t_{request}$ resp. $RTC_O\ mod\ t_{request}$ zeigt die erfüllte Bedinung an.

Das Synchronisationsprotokoll ist ziemlich einfach: der Observer ist nicht gezwungen auf eine SynReq-Nachricht zu Antworten; der Sensor sendet in diesem Fall dem nächsten Beobachter eine Synchronisationsanfrage. Dieses Verhalten kann in Szenarien mit mehreren Beobachtern Sinn machen und könnte beispielsweise verwendet werden um Beobachter mit niedriger Batteriekapazität zu schonen. Observer mit schwacher Akkuladung würden in diesem Fall mit kleinerer Wahrscheinlichkeit antworten als solche mit hoher. In der Implementation des SensorNet wurde darauf verzichtet, der Observer antwortet unverzüglich auf jede empfangene Synchronisationsanfrage:

- Der Sensor öffnet eine L2CAP-Verbindung zu einem Observer, der sich im ready-Zustand befindet,
- Eine synreq-Nachricht signalisiert das Interesse an einer Synchronisation,
- Der Beobachter antwortet mit einer synresp-Nachricht,
- Der Sensor schliesst die Verbindung und passt rtc_{shift} an.

Der Beobachter nimmt nach erfolgreichem Senden der synresp-Nachricht den Sensorknoten in seinen Schedule auf. Die synresponse-Nachricht enthält den relativen Zeitpunkt t''_{rel} des Observers innerhalb $t_{request}$. Der Sensor passt darauf rtc_{shift} gemäss t''_{rel} an:

$$rtc_{shift} = t''_{rel} - (RTC_S mod t_{request}) \qquad (3)$$

Bei der Berechnung von t''_{rel} durch den Observer müssen zwei Dinge beachtet werden:

1. Damit der Beobachter eine Verbindung zum Sensor öffnen kann, um eine DataReq-Nachricht zu übermitteln, muss der Sensor bereits wach sein. Wie in Unterkapitel 7.1.2 besprochen wird, kann der Zeitpunkt des Aufwachens sowohl des Sensors, wie auch des Beobachter nie ganz präzise eingehalten werden.
2. Zum Senden von DataReq-Nachrichten öffnet der Beobachter Verbindungen zu verschiedenen Sensorknoten. Es kann dabei nur eine Baseband-Verbindung gleichzeitig geöffnet werden, was logisch erscheint, wenn man sich die Funktionsweise des Paging vergegenwärtigt. Daher teilt der Observer die Sensoren in Zeitschlitze ein, die nacheinander abgearbeitet werden. Es macht natürlich wenig Sinn, dass ein Sensor, der in einem späten Slot kontaktiert wird, bereits zum Zeitpunkt des ersten Slots aufwacht.

Es gilt grundsätzlich für die relative Zeit t_{rel} innerhalb des Request-Intervalles

$$t_{rel} = RTC_O \; mod \; t_{request}$$

Unter Berücksichtigung von Punkt 1, wird diese Zeit etwas nach vorne geschoben, was einem früheren Aufwachen des Sensors entspricht (vgl. Formel (1), (3)):

$$t'_{rel} = (RTC_O + t_{wakeup\_duration}) \; mod \; t_{request}$$

Nun soll der Sensor, wie unter Punkt 2 beschrieben erst in seinem Slot aufwachen. Dies erreicht man, indem die Zeit etwas zurückgedreht wird, was den Sensor später aufwachen lässt (vgl. Formel (1), (3)):

$$t''_{rel} = (RTC_O + t_{wakeup\_duration} - (s * t_{slot\_duration})) \; mod \; t_{request} \qquad (4)$$

t_{rel}	Relativer Zeitpunkt innerhalb $t_{request}$
$t_{request}$	Abfrageintervall für Sensordaten
RTC_S	Wert der Echtzeituhr des Sensors
RTC_O	Wert der Echtzeituhr des Observers
$t_{wakeup\_duration}$	Zeitverschiebung um unpräzises Aufwachen zu kompensieren
s	Nummer des Slots des Sensorknotens, $s = 0, 1, 2, ..$
$t_{slot\_duration}$	Die Dauer eines Slots

Neben der eigentlichen Synchronisation durch Anpassen von rtc_{shift} und $t_{request}$, informiert der Beobachter den Sensor mit Hilfe der Synchronisationsnachricht über das verwendete Timeout von baseband-Verbindungen.
Nach erfolgter Synchronisation geht der Sensorknoten in die Schlafphase über.

Datenübertragung

Das Protokoll zur Datenübertragung ist einfach.

- Observer öffnet L2CAP Verbindung,
- Eine DataReq-Nachricht signalisiert dem Sensor die Datenübertragung,
- Der Sensor antwortet mit DataResp-Nachricht,
- Der Observer schliesst die Verbindung,

Wie bereits angesprochen, können Baseband-Verbindungen nicht gleichzeitig aufgebaut werden, die Sensoren werden in aufeinanderfolgende Zeitnischen eingeteilt und gemäss diesen kontaktiert. Ein Verbindungsaufbau darf auf keinen Fall über seine Zeitnische hinaus andauern, ansonsten kann mit nachfolgenden Sensoren nicht kommuniziert werden. Das Timeout der Baseband-Verbindungen wird dazu auf einen Wert $\leq t_{slot\_duration}$ festgelegt.
Die Wahl dieses Timeouts ist kritisch. Ein zu kleiner Wert bricht den Aufbau von Verbindungen vorschnell ab, ein zu grosser Wert hingegen lässt den Obverver untätig im aktiven Zustand verweilen.
Beim Paging - der verbindungsherstellenden Bluetooth-Operation - ist das Wissen über das clock-Offset des Slave-Gerätes von Vorteil: Eine Schätzung der aktuellen Hop-Frequenz beschleunigt den Vorgang (Kapitel 4.2.3). Das Bluetooth-Modul merkt sich dazu nach Inquiries den Offset jedes Gerätes. Nach einer beobachterseitigen Schlafphase und dem damit verbundenen Ausschalten des Bluetooth-Moduls ist der flüchtige Speicher gelöscht und damit auch die entsprechenden Informationen. Aber selbst das Erhalten der gespeicherten Offsets ist nutzlos, durch den Neustart des Sensor-ROK wird dessen Clock initialisiert.
Ideen zur Verbesserung des Paging hinsichtlich der Geschwindigkeit exsitieren. Das in [15] vorgeschlagene Verbindungsschema ist jedoch nicht praxiserprobt, die vielversprechenden Ergebnisse daher mit Vorsicht zu geniessen.
Die Praxis hat gezeigt, dass es nicht empfehlenswert ist, während der Phase, in welcher der Beobachter Verbindungen zur Datenübertragung öffnet, passive Verbindungen und Synchronisationsanfragen von Sensoren zu erlauben. Dies kann verhindert werden, indem Page-Scans zeitweise deaktiviert werden. Der Beobachter ist zwar weiterhin sichtbar, reagiert aber nicht auf Page-Nachrichten was passive Verbindungen verhindert.

Die Übertragung der Messwerte geschieht mit L2CAP Paketen. Dazu werden die Daten mit einer entsprechenden Funktion serialisiert übertragen und deserialisiert. Eine explizite Serialisierungsfunktion garantiert eine problemlose Kommunikation mit verschiedenen Plattformen durch Datenkonvertierung. Ein Beispiel ist der Long-Integer-Datentyp, der auf der i386-Plattform 4 Bytes lang ist, auf der BTnode-Plattform aber nur eine Länge von 2 Bytes besitzt.
Auf L2CAP-Ebene werden die Daten in 17 Bytes grosse Pakete gesplittet, was dem DM1-Datentyp entspricht (siehe Anhang E). Leider bietet das auf den BTnodes implementierte L2CAP keine Möglichkeit zur Wahl des Pakettyps. Kommunikation in 17-Bytes-Paketen kann nie die Effizienz von 339 Bytes langen DH5 Paketen erreichen. Für unsere Anwendung macht das keinen erheblichen Unterschied, da die Datenmenge sowieso eher klein ist und in wenigen DM1 Baseband-Paketen Platz findet.
Nach erfolgreicher Übertragung der Sensorwerte, werden diese aus dem Buffer gelöscht.

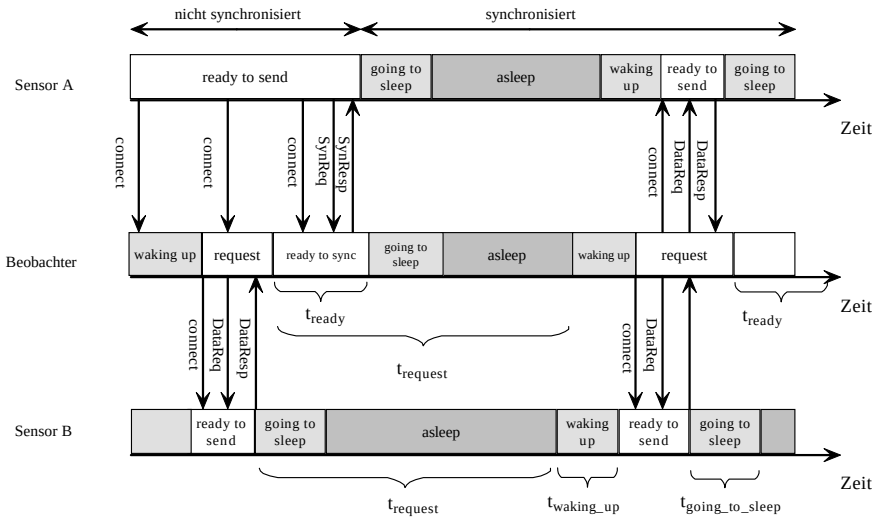


Abbildung 7.1: SensorNet mit zwei Sensoren und einem Beobachter

Aufwachen, Einschlafen

In der Schlaf-Phase der Sensoren und des Beobachters wird das Bluetooth-Modul vom Strom getrennt [3]. Frühere Versuche haben gezeigt, dass trotz Ausschalten des Radiomoduls der Stromverbrauch nicht auf null sinkt.
Das plötzliche Aus- und Wiedereinschalten des Bluetooth-Moduls ist in den diversen Protokollschichten nicht vorgesehen. Einige Bedingungen müssen für einen reibungslosen Ablauf vor dem Ausschalten erfüllt sein:

1. Es dürfen weder aktive noch passive Baseband-Verbindungen bestehen
2. Weder aktive noch passive Baseband-Verbindungen düfen sich im Aufbau befinden
3. Weder aktive noch passive Baseband-Verbindungen dürfen sich im Abbruch befinden

Bezogen auf den Status einer Verbindung heisst dass, dass die Zustände 2 (bb_w4_connection_complete_intiator), 3 und 3' (bb_w4_connection_complete_acceptor), 4 (bb_connection_complete) und 5 (bb_w4_complete_initiator) der bb-connection state machine verlassen sein müssen (siehe Abbildung ¡referenz auf bb statemachine¿).
Leider bietet das API nur Events an, welche den erfolgreichen Aufbau oder Abbruch einer Baseband-Verbindung anzeigen. Ausserdem bleibt dem Programmierer der Zugriff auf die Connection-Table und damit auf den internen Zustand einer Verbindung verwehrt.

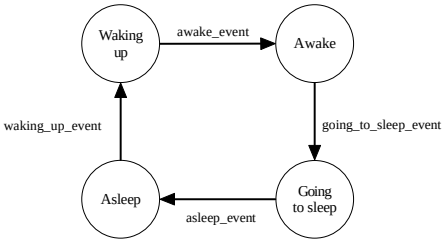


Abbildung 7.2: Zustandsdiagramm Sensor

Die hier vorgestellte Lösung kommt ohne das Wissen dieser Verbindungszustände aus. Jeder Knoten befindet sich in einem von vier Zuständen (Abbildung 7.2). Knoten im Awake-Zustand besitzen ein einsatzbereites Bluetooth-Modul. Beabsichtigen Sie diesen Zustand zu verlassen und das Bluetooth-Modul von der Stromversorgung zu trennen, signalisieren sie das mit einem going-to-sleep Event. Daraufhin wird das Paging ausgeschaltet, um passive Neuverbindungen zu vermeiden und nach einer kurzen Zeit, in der anstehende Daten vom L2CAP noch verschickt werden können, werden alle bestehenden Verbindungen abgebrochen. Es ist wichtig, dass zum Zeitpunkt dieses Events kein aktiver Verbindungsaufbau hängig ist. Nun ist das Bluetooth-Modul bereit, definitiv abgeschaltet zu werden, was durch einen asleep-Event geschieht.
Soll das Bluetooth-Modul eingeschaltet werden, kann ein waking-up Event ausgelöst

[3] void btn_bt_power_off(bool off)

werden. Das Bluetooth-Modul wird unverzüglich mit Strom versorgt, kurz danach zeigt ein awake-Event Kommunikationsbereitschaft an.
Auf keinen Fall dürfen Bluetooth-Befehle im asleep-Zustand ausgeführt werden. Ein CMD_Buffer_Overflow Fehler ist die Folge und ein Reset des BTnode ist unumgänglich. Die Verletzung einer der drei am Anfang dieses Unterkapitels aufgeführten Bedingungen führt zum selben Verhalten.

7.1.3 Implementation

Die Implementation der Sensor- und Beobachterknoten ist getrennt voneinander vorgenommen worden. Anfangs schwebte mir ein Programm vor, welches die Funktionalitäten beider Knoten enthält und in welchem ein Flag den Betriebsmodus festlegt. Mit einem Signal an einer Schnittstelle könnte ein Sensor zu einem Beobachter werden und umgekehrt. Der Vorteil liegt auf der Hand: In der Praxis wird nur eine Knotenart benötigt, ein Sensor könnte im Handumdrehen mit einem GSM-Modul versehen werden und per Knopfdruck die Funktion eines Beobachterknotens übernehmen. Schon im Anfangsstadium der Entwicklung zeigte sich aber, dass eine solche Entwicklung zu viele Nachteile mit sich bringt. Der Code wird unnötig verkompliziert und die Codelänge steigt beträchtlich.
Der Programmcode gliedert sich in sechs Teile, eine detaillierte Auflistung aller Dateien findet sich in Anhang C, Codefragmente in Anhang D.

- Der Programmcode für Sensorfunktionalitäten ist zuständig für das Suchen von Observern, die Synchronisation, das Scheduling und die Übermittlung von Sensordaten.
- Die Sensoranwendung wird auf den konkreten Einsatz zugeschnitten. Sie hat die Möglichkeit Einstellungen am Sensornetz vorzunehmen und kümmert sich um die Erfassung von Sensordaten.
- Der Programmcode für Beobachterfunktionalitäten beantwortet Synchronisationsanfragen, ist verantwortlich für das Scheduling der Sensoren und deren rechtzeitige Abfrage und informiert die Beobachteranwendung über eintreffende Sensordaten.
- Die Beobachteranwendung ist auf das Anwendungsszenario zugeschnitten indem Einstellungen am Sensornetz vorgenommen werden und die Weiterverwendung der empfangenen Sensordaten geregelt wird.
- Funktionen zum Speichern und Finden von Informationen zu Bluetooth-Geräten in Reichweite.
- Funktionen zur Verwaltung, Serialisierung und Deserialisierung und Speicherung von Sensordaten.

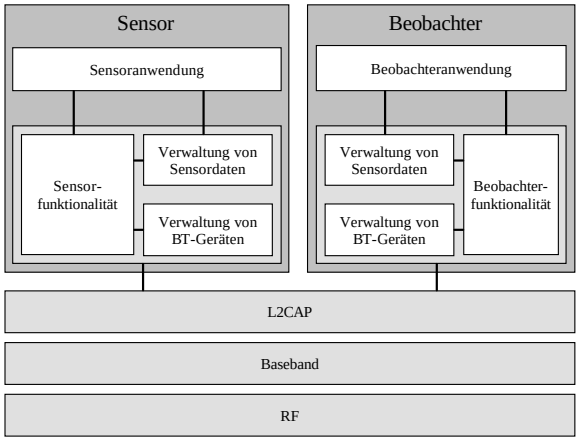


Abbildung 7.3: Die Implementation der Sensor- und Beobachterknoten.

7.2 Algorithmus II: Multihop Synchronized SensorNet

7.2.1 Idee

Der Multihop SensorNet Algorithmus ist eine Erweiterung von SensorNet. Eine Implementation wurde bis zum Ende dieser Arbeit nicht fertiggestellt, darum sei das Prinzip an dieser Stelle nur skizziert.
Die Sensorknoten des bisherigen SensorNet-Algorithmus können Messwerte sammeln und Daten nur senden, der Observer war dahingehend konzipiert diese anzufordern. Die Knoten des Multihop SensorNet-Algorithmus vereinigen beide Funktionalitäten. Sie können einerseits als Sensor agieren, sich mit einem Beobachter synchronisieren und Messwerte sammeln und übermitteln. Auf der anderen Seite haben sie auch Möglichkeiten SynReq-Nachrichten zu beantworten und DataReq-Nachrichten an synchronisierte Sensoren zu richten. Der Observer unterscheidet sich von den Sensoren nur in der Möglichkeit, die Daten über GSM weiter zu leiten. Im Multihop SensorNet sind Sensorknoten paarweise miteinander synchronisiert. Knoten B in Abbildung 7.4 ist mit dem Observer A synchronisiert. Gleichzeitig ist Knoten C mit Knoten B synchronisiert, D mit C usw.

Der Aufbau des Netzes erfolgt von der Wurzel her, welche durch den Beobachter gebildet wird. Jeder Knoten merkt sich die Distanz vom Beobachterknoten in Anzahl Hops und speichert diese in der $counter_{hop}$-Variable. In der initialen Phase des Netzwerkes beträgt dieser Wert in jedem Sensorknoten ∞, $counter_{hop}$ des Observers ist 0. Die Sensoren suchen nun nach Knoten - Observer und Sensoren unterscheiden sich nicht mehr - mit möglichst niedrigem Wert $counter_{hop}$, wobei $counter_{hop} < \infty$ sein muss. Sobald ein passender Kandidat gefunden ist, wird eine Synchronisation nach bekanntem Schema ausgeführt. $counter_{hop}$ des Knotens ist nun um eins grösser als derjenige seines Vorgängers. Der Aufbau des Netzwerkes ist auf Abbildung 7.6 dargestellt. Bei der Synchronisation muss beachtet werden, dass die Sensoren rückwärts in Slots eingeteilt werden, später synchronisierte Knoten werden also früher aufwachen.

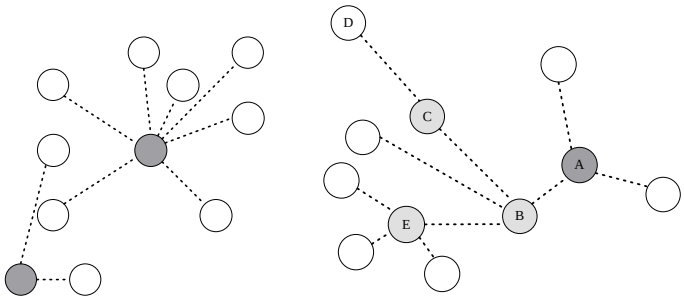


Abbildung 7.4: Im Gegensatz zu SensorNet (links), wo nur Beobachterknoten (dunkelgrau) und Sensorknoten (weiss) existieren, haben in Multihop SensorNet (rechts) Sensorknoten eine Doppelfunktion (hellgrau)

Verliert ein Knoten den Kontakt zu seinem synchronisierten Gegenüber - eine Verbindung wird nicht rechtzeitig aufgebaut und er erhält keine DataReq-Nachricht - setzt er seinen $counter_{hop}$ auf ∞ und sucht nach einem neuen Observer. Dieses Verhalten wird automatisch bis zu den Blattknoten zurückpropagiert, das Netz formiert sich neu. Abbildung 7.5 zeigt den Observer A und die Sensorknoten B, C, D und E in einem Zeitdiagramm auf Kommunikationsebene. Wie leicht zu erkennen ist, besitzen die Sensoren wie bisher eine awake-Phase um Kommunikationsanfragen mit den gesammelten Messwerten zu beantworten. Daneben besitzen sie aber auch eine request-Phase um Messwerte anderer Sensoren zu empfangen und ein Intervall, in welchem sie wach bleiben um Synchronisationsanfragen zu beantworten.

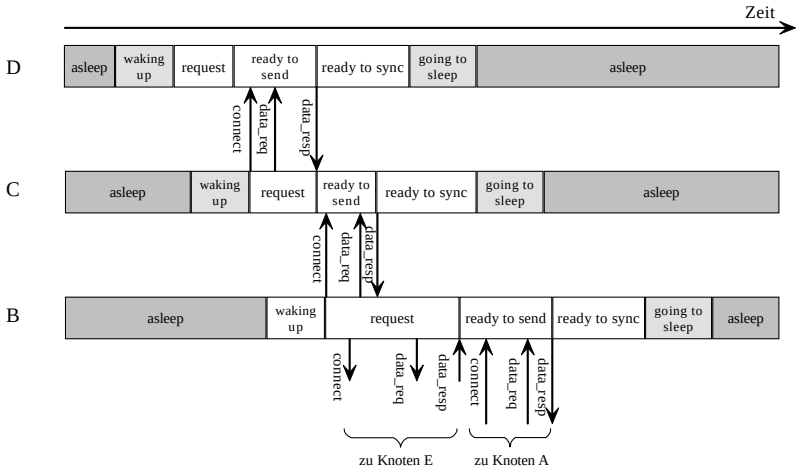


Abbildung 7.5: Drei synchronisierte Sensoren im Multihop SensorNet.

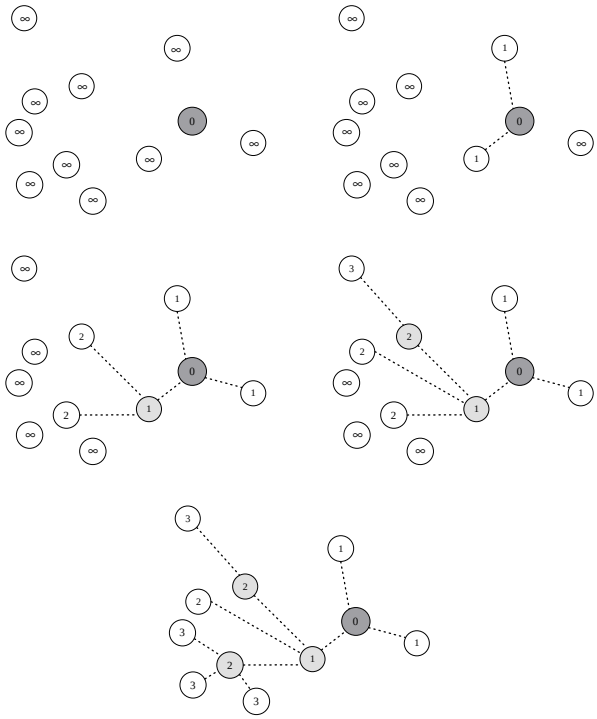


Abbildung 7.6: Der Aufbau des Multihop-Netzwerkes beginnt bei der Wurzel, dem Beobachter

7.2.2 Erweiterung

Die erklärte Form des Multihop-SensorNet-Algorithmus bildet eine Netztopologie, welche die Länge der Pfade und damit den gesamten Energieverbrauch minimiert. Wie bereits das Maximum Lifetime Routing zeigt, maximiert die Verwendung von Pfaden minimaler Energie nicht die Laufzeit des Netzes, dem Ausfall von Knoten mit schwacher Batterieladung wird nicht vorgebeugt.
Wird der bisherige Wert von $counter_{hop}$ nicht als Distanzangabe aufgefasst, sondern als Eignung des Knotens verstanden, können analog Topologien gebildet werden, die der minimum cost-Eigenschaft entsprechen (auch Kapitel 6.2). Der Wert $counter_{cost}$ wird in diesem Fall pro Hop nicht um eins erhöht, sondern um den Wert einer Kostenfunktion, welche die verfügbaren Resenergie berücksichtig.

7.3 Reduktion des Energieverbrauchs

In diesem Kapitel wird auf die Implementation der Energiesparfunktionen des ATmega128L Controllers eingegangen. Eine Beschreibung derselben findet sich in Abschnitt 3.2.
Es bietet sich an die AVR Libc zu verwenden [4]. Diese C-Bibliothek für die 8-bit RISC Microcontrollerfamilie von Atmel stellt nützliche Funktionen zur Verfügung, unter anderem einen vereinfachten Zugriff auf Register und die Aktivierung von Schlafmodi.

7.3.1 Anpassen der Taktrate

Das Anpassen der Taktrate geschieht über das XDIV-Register (siehe Kapitel 3.2.3). Bei einer Division der system clock wird die Geschwindigkeit aller Komponenten reduziert. Es wird also nicht nur die Taktung der CPU und des RAM verändert, sondern insbesondere auch diejenige der Schnittstellen. Eine Ausgabe von Debug-Informationen über UART ist daher nicht mehr ohne weiteres möglich.
Timer/Counter0 kann bei einer Reduktion der Taktrate nur noch über einen externen Oszillator betrieben werden. Die Frequenz des externen Oszillators muss niedriger als 1/4 der Frequenz der skalierten system clock sein, ansonsten gehen Interrrupts verloren und der Zugriff auf die Register des Timer/Counter0 kann nicht gewährleistet werden. Der Betriebsmodus wird durch das ASSR (Asynchronous Status Register) festgelegt. Ist Bit 0 des ASSR auf 1 gesetzt, wird Timer/Counter0 aus einem mit TOSC1 und TOSC2 verbundenen 32'768 Hz-Oszillator generiert, ansonsten aus clk_{IO}. Für die Frequenz der angepassten system clock muss demnach gelten:

$$f_{osz\_ext} \leq 1/4 * f_{clk}$$

$$f_{clk} \geq 4 * f_{osz\_ext} = 4 * 32'768Hz = 131072Hz = 128kHz.$$

damit ist ist der Wert d des XDIV-Registers (Kapitel 3.2.3):

$$129 - d \leq \frac{f_{source\_clock}}{f_{clk}}$$

$$d \geq 129 - \frac{f_{source\_clock}}{f_{clk}} = \frac{8Mhz}{128kHz} = 66.$$

f_{clk}	Frequenz der system clock nach Frequenzteilung
f_{osz_ext}	Frequenz des externen Oszillators für Timer/Counter0
f_{source_clock}	Frequenz der system clock vor Frequenzteilung
d	Wert in XDIV zur Teilung der Taktfrequenz

7.3.2 Schlafmodi des Prozessors

Eine Übersicht der Schlafmodi, deren Auswirkungen und deren Aufwachbedingungen sind in Abschnit 3.2.2 erklärt und abgebildet.
Die RTC-Fähigkeiten der BTnodes sind über den 8-Bit Timer/Counter0 implementiert.

[4] http://savannah.nongnu.org/projects/avr-libc/
http://jubal.westnet.com/AVR/doc/avr-libc-user-manual/

Dieser Timer arbeitet mit einer Frequenz von 1024 Hz und läuft daher vier mal in der Sekunde über, wobei jedesmal ein Interrupt generiert wird. Dieser Interrupt wird von der BTnode System Software abgefangen und die Echtzeituhr um 250 ms erhöht. Gleichzeitig werden durch Timeouts ausgelöste Ereignisse in die Queue des Dispatchers gelegt. Wird die Echtzeituhr-Funktionalität verwendet (was üblicherweise in Anwendungen des BTnode der Fall ist), darf kein sleep-Modus zum Einsatz kommen, der den Timer ausschaltet. Damit fallen *standby* und *power down* weg.
Die restlichen Schlafmodi werden spätestens $1/4$ Sekunde nach Aktivierung durch das Auftreten des Overflow-Interruptes verlassen. Daraufhin wird die Echtzeituhr berechnet, registrierte Events ausgelöst und die Kontrolle dem Dispatcher übergeben. Soll der Prozessor für längere Zeit in einem energiesparenden Zustand gehalten werden, muss der Schlafmodus nach dem Erwachen wieder aktiviert werden. Dazu ist eine Modifikation der Echtzeituhr unumgänglich. Nachdem die Timeouts überprüft worden sind, kommt ein weiteres Event in die Queue des Dispatchers. Dieses Event wird durch die Eigenschaft der verwendeten FIFO-Queue zuletzt abgearbeitet und übergibt die Kontrolle einer Funktion, welche den Controller wieder in den stromsparenden sleep- oder idle-Modus zurückversetzt.

7.3.3 Ausschalten der LEDs

Die LEDs sollten bei der Ausführung von Sensornet ausgeschaltet sein. Das durch die Echtzeituhr bedingte Blinken von LED 0 kann durch eine Anpassung von avr128_rtc.c deaktiviert werden.

Kapitel 8

Analyse

8.1 Versuchsaufbau

Dieses Kapitel wendet sich Messungen zu, die Aufschluss über die Effizienz des implementierten Sensornetzes geben.
Ein modifizierter BTnode wird eingesetzt, um die Stromaufnahme von verschiedenen Komponenten getrennt zu messen. Durch Entfernen der drei 0 Ohm-Widerstände erhält man Messpunkte für den μController, den SRAM und das Bluetooth-Modul:

- Messpunkt 1 umfasst μControlelr, LEDs, externer Oszillator und andere periphere Komponenten des Controllers.
- Messpunkt 2 misst die Stromaufnahme des AMIC SRAM und des Latch.
- Messpunkt 3 gibt Aufschluss über die Stromaufnahme des ROK 101 007 Bluetooth-Moduls.

Zur Strommessung wird ein HP 34401A Multimeter eingesetzt, das über eine serielle Schnittstelle mit einem Terminal verbunden ist. Die Messungen erfolgen mit einer Genauigkeit von $5\,1/2$ Stellen und einer Rate von 50 Messungen/s.
Für die Messungen sind alle Geräte von den Schnittstellen des BTnode getrennt worden und die LEDs wurden (wenn nicht ausdrücklich erwähnt) deaktiviert.

8.2 Komponentenweise Messungen

In diesem Kapitel werden Messergebnisse der Hauptkomponenten in verschiedenen Modi vorgestellt.

8.2.1 Bluetooth-Modul

Die Messwerte sind in Abbildung 8.1 grafisch dargestellt, eine Auflistung der Werte findet sich in Anhang A
Alle Messungen am Bluetooth-Modul sind für zwei Typen des ROK 101 007 ausgeführt worden. Typ 1 mit der Bezeichnung 01W46 auf der Vorder- resp. B141 auf der Rückseite schneidet schlechter ab als der neuere Typ mit den Bezeichnungen 02W21 und B214.

Wie erwartet geht der grösste Teil des Stromverbrauchs auf Rechnung des Bluetooth-Moduls. Die Inquiry und Connect-Operationen sind am Energieintensivsten, aber auch das Offenhalten einer Verbindung - unabhängig davon ob in Master- oder Slaverolle - und das Senden und Empfangen benötigen überraschenderweise annähernd gleich viel Strom. Messungen haben gezeigt, dass der Stromverbrauch bei mehreren offenen Verbindungen nicht erheblich steigt.
Interessant ist das Verhalten beider Module im Idle-Zustand. Wird zuvor eine Inquiry ausgeführt steigt die aufgenommene Leistung von knapp 20mW auf 70mW an. Erst nach einem Verbindungsaufbau und -abbruch sinkt die Leistung wieder auf den Ausgangszustand. Eine Erklärung für dieses Verhalten gibt es im Bezug auf die Funktionsweise von Bluetooth nicht und muss mit dem vom Controller implemetierten Teil des HCI zusammenhängen. Ein starkes Indiz dafür ist, dass selbst das Trennen und Wiederversorgen des Bluetooth-Moduls mit Strom keine Veränderung bewirkt.
Wird das Radiomodul ausgeschaltet, fliesst noch immer ein Strom, selbst wenn dieser deutlich kleiner ist als im Idle-Modus.

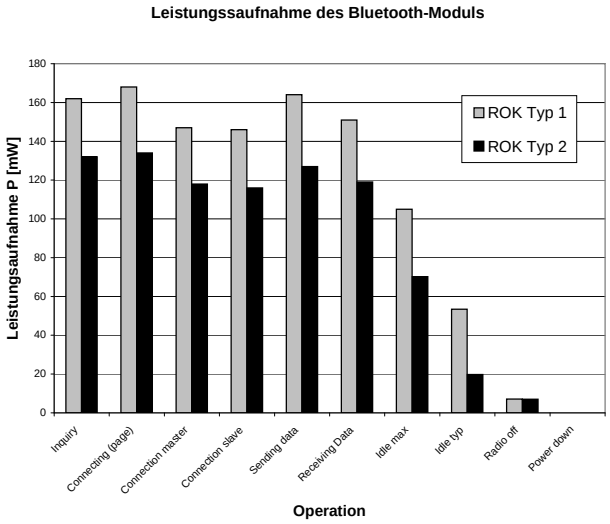


Abbildung 8.1: Die Leistungsaufnahme verschiedender Operationen des Bluetooth-Moduls

8.2.2 μController und SRAM

Grafik 8.2 gibt Aufschluss über die Leistungsaufnahme des μControllers bei Aktivität und in verschiedenen Energiesparmodi. Die zugehörigen Messwerte sind in A zu finden. P_{max} im aktiven Zustand wurde bei belasteter MCU gemessen, P_{typ} hingegen bei Ausführung des Dispatchers ohne auftretende Events, wobei der μController zeitweise im Idle-Modus verweilt. Befindet sich der Prozessor exklusiv im Idle-Modus können gegenüber dem aktiven Zustand 2/3 der Energie gespart werden. Im Power Save Zustand beträgt der Stromverbrauch nur noch $17\mu A$. Vom Einsatz solcher Modi sollte daher wenn möglich Gebrauch gemacht werden.

Über den μController werden ebenfalls die vier Leuchtdioden gespiesen. Direkte Messungen an den Dioden waren nicht möglich, der über die LEDs fliessende Strom kann aber als Differenz $I_{cpu\_led\_on} - I_{cpu\_led\_off}$ berechnet werden. Erstaunlicherweise besitzen die LEDs einen relativ hohen Stromverbrauch, leuchten alle vier benötigen diese mehr Strom als der μController. Auf den Betrieb von Leuchtdioden sollte daher wenn immer möglich verzichtet werden.
Die Stromaufnahme des AMIC SRAM und des Octal Latch werden zusammen erfasst. P_{max} wird beim Schreiben auf das SRAM erreicht, P_{typ} beim Betrieb ohne Scheib- und Lesezyklen.

Taktreduktion des μControllers

Die Strommessungen des μController sind unter Last (P_{max}) und unbelastet (P_{typ}) vorgenommen worden. Die Taktrate wurde sukszessive reduziert. Die Ergebnisse für P_{max} (siehe auch A) decken sich ziemlich genau mit den Herstellerangaben in [3] (Figur 160 auf Seite 333). Wird nicht die komplette Prozessorleistung benötigt, kann es sich also lohnen die Frequenz zu verringern. Es muss beachtet werden, dass eine Frequenzdivision den gesamten μController betrifft. Beispielsweise werden Schnittstellen ebenfalls mit verminderter Geschwindigkeit arbeiten.

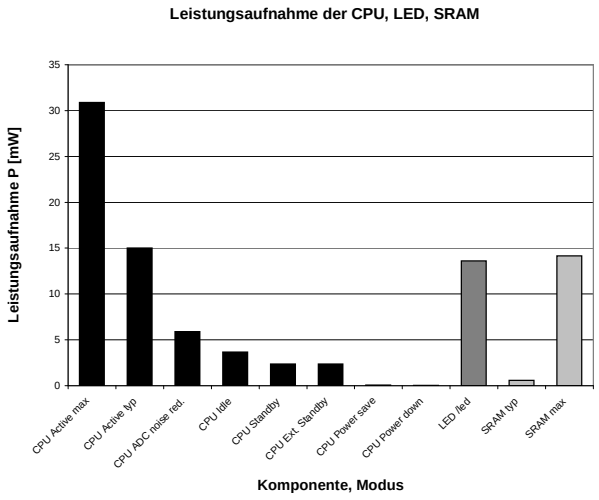


Abbildung 8.2: Leistungsaufnahme des Controllers, LED und SRAM.

8.3 Vergleichsmessungen

In diesem Kapitel werden die erhaltenen Messwerte für die Stromaufnahme mit Werten aus der Literatur [9] verglichen und validiert. Zum Vergleich wird die Summe der Komponenten in den jeweiligen Zuständen herangezogen.
Die in der nachfolgenden Tabelle 8.3 aufgeführten Werte stimmen zwar nicht alle

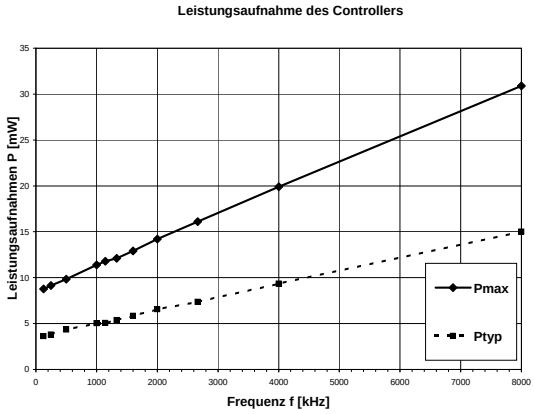


Abbildung 8.3: Die Leistungsaufnahme des Controllers ist abhängig von der Taktfrequenz.

überein, bewegen sich aber in den selben Grössenordnungen. Die in [32] erzielten Messwerte dagegen liegen weit darüber. Über die Gründe kann nur spekuliert werden. Eine Erklärung könnte die Verwendung unterschiedlicher Versionen des ROK101007 Bluetooth-Moduls sein.

Zustand	Messungen aus [9]	Eigene Messung
Bluetooth Connected, CPU on	160 mW	150 mW
Bluetooth Idle, CPU on	95 mW	101 mW
Bluetooth Off, CPU Idle	15 mW	15 mW
Bluetooth Off, CPU Sleep	6 mW	6 mW

8.4 Analyse einer Bluetooth-Verbindung

Bei der Analyse des Bluetooth-Moduls hat sich gezeigt, dass der Stromverbrauch sehr genaue Rückschlüsse über dessen Aktivität zulässt. Die Messauflösung von 50 Hz erlaubt es, kürzeste Peaks zu erfassen und einer Operation zuzuordnen. Die beiden Grafiken 8.4 entsprechen der Stromaufnahme des Bluetooth-Moduls während etwa 25 Sekunden. Die Kurven zeigen den exemplarischen Aufbau einer Bluetooth-Verbindung aus der Sicht eines Master- und eines Slavegerätes. Zeitliche Diskrepanzen zwischen beiden Darstellungen sind auf die nicht-gleichzeitige Ausführung der Messungen zurückzuführen.
Auffallend sind die regelmässigen Peaks der Inquiry- und Pagescans im Abstand von 1.28 Sekunden. Während der Inquiry werden auf dem Mastergerät keine Scans ausgeführt, ebensowenig während dem Aufbau der baseband-Verbindung.
Sehr gut zu erkennen ist die Reaktion auf ein Inquiry: Mit dem ersten inquiry-Scan gelangt der Slave in den inquiry-response-Zustand, was sich schön am leicht erhöhten Stromverbrauch ablesen lässt. Die Antwort erfolgt - wiederum deutlich erkennbar - als FHS-Paket eine zufällige Anzahl Zeitschlitze später.

Bei Zeitpunkt 600 initiiert der Master einen Verbindungsaufbau, das Senden von page-Nachrichten in jedem Zeitschlitz zeigt sich im hohen Strombedarf. Die Antwort mittels page-Nachricht und der Austausch der FHS-Paketen entspricht dem darauffolgenden Peak. Die zeitliche Verschiebung ist auf die anfangs erwähnte Messweise zurück zu führen.
Das Übermitteln von Datenpaketen an den Slave zum Zeitpunkt 900 zeigt sich einem erhöhten Strombedarf des sendenden Gerätes. Das empfangende Gerät weist dabei keinen signifikant höheren Energiebedarf auf.

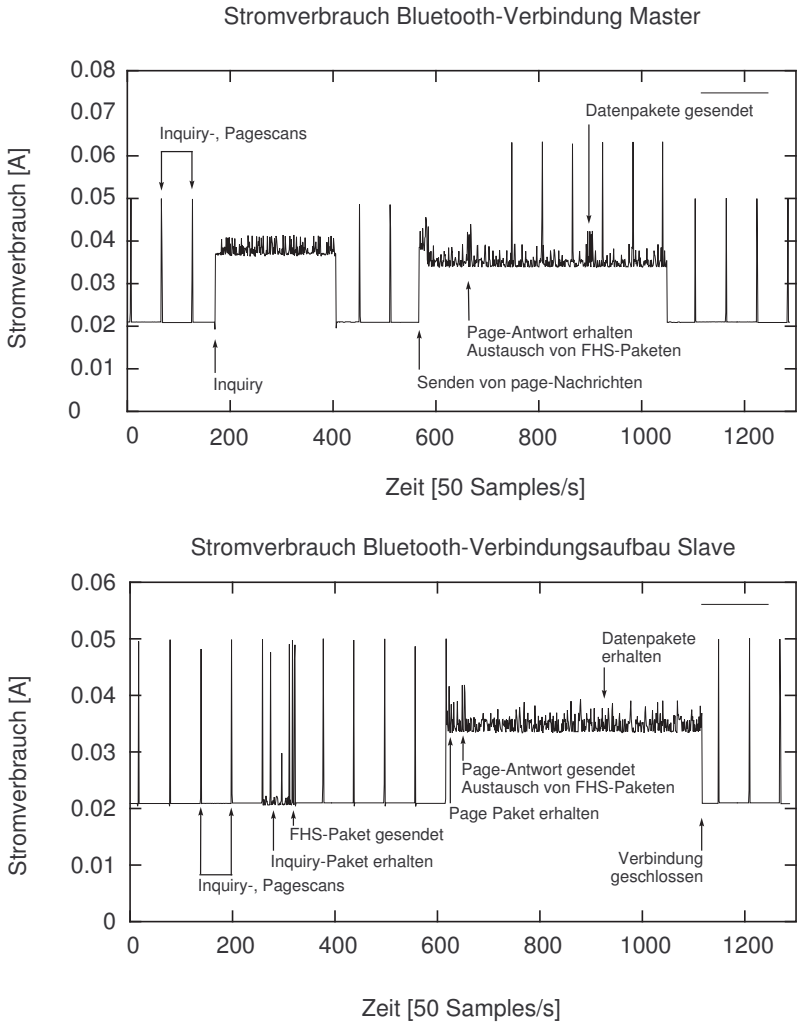


Abbildung 8.4: Analyse einer Bluetooth-Verbindung anhand des Stromverbrauchs

8.5 Analyse SensorNet

Dieses Kapitel widmet sich der Analyse des Stromverbrauchs unter Einsatz des SensorNet-Algorithmus. Die Stromaufnahme des Prozessors und des Bluetooth-Moduls eines Sensors sind getrennt über einen Zeitraum von etwa 90 Sekunden gemessen worden. Sensordaten werden in Intervallen von 45 Sekunden übertragen. Abbildung 8.5 zeigt beide Kurven in einem Diagramm.
Der hohe Stromfluss des Bluetooth-Moduls in den ersten Sekunden ist auf das Suchen eines Beobachter und das synchronisieren zurückzuführen; Im vorliegenden Fall wird auf Anhieb ein Beobachter gefunden. Die Kurvenpeaks werden wie bei vorhergehenden Messungen durch Inuiry- und Pagescans verursacht.
Die Wachphase des Bluetooth-Moduls dauert ungefähr fünf Sekunden. Zwei Sekunden nachdem das Kommunikationsmodul mit Strom versorgt wird, baut der Beobachter die Verbindung auf, was in der Regel nochmals 1 bis 2 Sekunden in Anspruch nimmt. Die darauffolgende Übermittlung von Sensordaten ist vergleichsweise schnell erledigt. Nach dem sicheren Beenden aller Verbindungen, was nochmals eine Sekunde dauert, wird das Bluetooth-Modul vom Strom getrennt.
Der Stromverbrauch des μControllers weist durch das Überprüfen des Schedule regelmässige Peaks auf. Im vorliegenden Beispiel geschieht das jede Sekunde, wobei dieses Intervall auch erhöht werden kann (siehe Kapitel B.1).
Erstaunlich ist, dass der Stromverbrauch des μControllers sprunghaft von etwa 5mA auf 8mA ansteigt, sobald das Bluetooth-Modul ausgeschaltet wird. Die Ursache dafür muss in der Systemsoftware des BTnode gesucht werden. Normalerweise versetzt der Dispatcher den μController nach Abarbeitung aller Events in den Idle-Modus. Das beobachtete Verhalten legt nahe, dass eine Funktionen, welche für die Kommunikation mit dem Bluetooth-Modul zuständig ist, nicht ordnungsgemäss terminiert sobald dieses ausgeschaltet wird. Der μController verweilt in diesem Fall bis zum nächsten auftretenden Timer-Overflow-Interrupt (siehe Kapitel 7.3.2) im aktiven Zustand.
Drei Lösungen sind denkbar:

1. Die Taktrate des Controllers wird angepasst. Eine Halbierung reduziert den Stromverbrauch um über 30%. Das Grundsätzliche Problem der blockierenden Funktionen wird dabei nicht angegangen.

2. Der Mikrocontroller wird unabhängig vom Dispatcher in einen stromsparenden Zustand zu versetzt. Dieser Zustand darf I/O grundsätzlich deaktivieren, da während dieser Zeit keine Kommunikation mit dem Bluetooth-Modul nötig ist.

3. Die sinnvollste Lösung besteht darin, die blockierende Funktion ausfindig zu machen und anzupassen.

Der zweite Lösungsansatz wurde versuchsweise implementiert und bewirkte eine Reduktion des Stromverbrauchs auf das Ausgangsniveau von 5 mA.

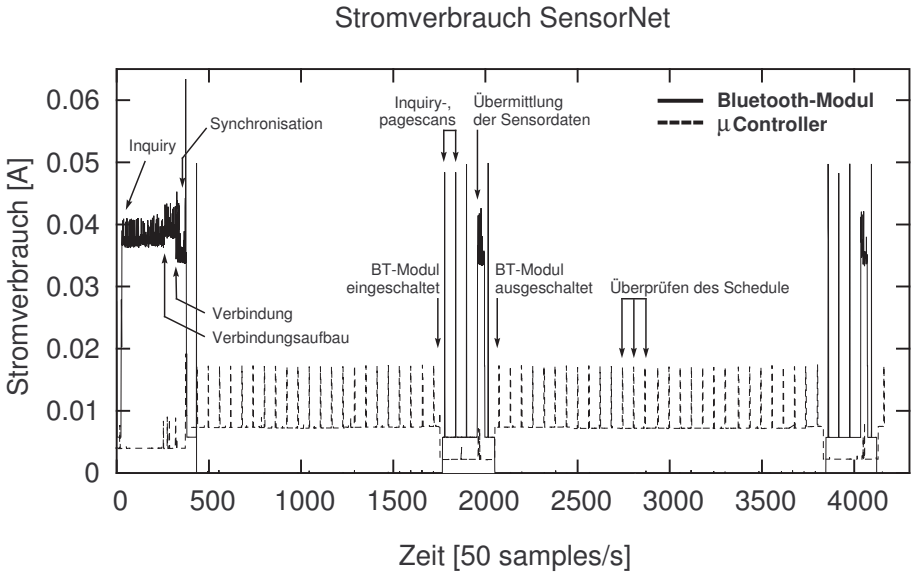


Abbildung 8.5: Der Stromverbrauch eines Sensors in SensorNet

8.6 Deutung der Ergebnisse, Fazit

Wie schon anfangs vermutet ist hauptsächlich das Bluetooth-Modul für den hohen Stromverbrauch der BTnodes verantwortlich. Es war also die richtige Entscheidung, dieses für den Einsatz in Sensornetzen für möglichst lange Zeit auszuschalten; Unser Ausgangsszenario erlaubt dies in besonderem Masse. Auf keinen Fall sollten Verbindungen unnötig lange aufrecht erhalten werden. Das Trennen des Bluetooth-Moduls von der Stromversorgung bringt zusätzliche Einsparungen, abhängig von dessen stark variierenden Grundverbrauch (siehe Abschnitt 8.2.1). Die Verwendung der Stromsparmodi des μControllers ist ebenfalls ein guter Weg Energie zu sparen. Der Energieverbrauch des SRAM ist in unserem Szenario vernachlässigbar und bedarf darum keiner Optimierung.
Auf die Verwendung der LEDs für Statusanzeigen sollte verzichtet werden, der Heartbeat der Echtzeituhr-Implementation kann ebenfalls unterdrückt werden.
Im Folgenden wird die durchschnittliche Leistungsaufnahme eines Sensorknoten unter Einsatz verschiedener stromsparender Massnahmen berechnet. Es wird angenommen, dass der Knoten nach jeweils 60 Sekunden 100 Bytes Daten aufgeteilt in 6 DM1-Paketen an einen Beobachter sendet. Dazu verweilt er während maximal 5s im wachen Zustand.
Im betrachteten Szenario wird der durchschnittliche Stromverbrauch des Bluetooth-Moduls durch das Abbrechen und Wiederaufbauen von Verbindungen bei Bedarf auf weniger als die Hälfte gesenkt. Das Trennen des Kommunikationsmodul von der Energieversorgung reduziert diesen nochmals um den Faktor 15.
Eine Reduktion der Taktrate wurde nicht berücksichtigt, ein Einsatz wäre nur in Verbindung mit einem optimierten Einsatz des Bluetooth-Moduls (Fälle E und F) ratsam. Eine Halbierung der Frequenz könnte den Stromverbrauch des μControllers zusätzlich um einen Drittel senken, wobei jedoch eine Anpassung der Übertragungsrate aller pe ripheren Geräte in Kauf genommen werden müsste (Abschnitt 7.3.1).

SensorNet implementiert die Fälle E und F. Die Kapazität einer 840mAh-Batterie reicht aus um einen Sensor während 80 resp. 150 Stunden zu betreiben, während die Lebensdauer eines Sensors ohne stromsparende Massnahmen auf 5.5 Stunden beschränkt ist. Fall G betachtet die Möglichkeit, den μController während den Schlafphasen des Sensors in den Sleep-Modus zu versetzen; Der Stromverbrauch wird nochmals um den Faktor 4 reduziert und würde damit eine optimale Betriebsdauer von 600 Stunden ergeben.
Die folgende Tabelle gibt Aufschluss über die verwendeten Methoden, Abbildung 8.6 zeigt eine graphische Darstellung der durchschnittlichen Leistungsaufnahme. Eine Wer tetabelle findet sich in Anhang A.

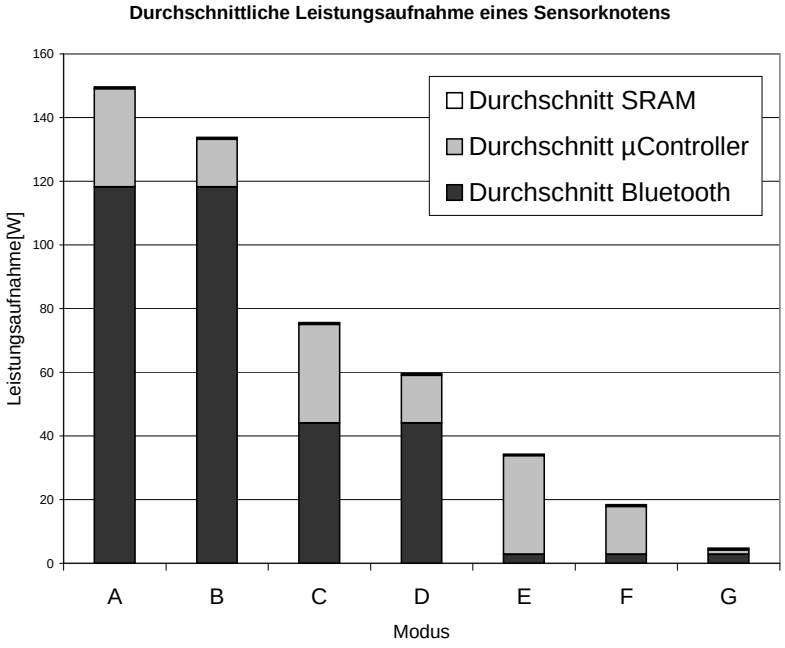


Abbildung 8.6: Durchschnittliche Leistungsaufnahme eines Sensorknotens unter Verwendung verschiedener stromsparender Massnahmen

Massnahme	A	B	C	D	E	F	G
Verbindungen werden nur zur Kommunikation geöffnet			x	x	x	x	x
Trennen des Bluetooth-Moduls von der Stromversorgung					x	x	x
μController verwendet den Idle-Modus		x		x		x	
μ Controller verwendet den Sleep-Modus							x

8.6.1 Anknüpfende Arbeiten

- SensorNet ist darauf ausgerichtet, dass jeder Sensorknoten in Reichweite eines Beobachters liegt. In der Praxis kann dies nur bedingt garantiert werden. Eine Implementation des Multihop-SensorNet-Algorithmus, wie er in Unterkapitel 7.2.1 beschrieben wird, wäre darum wünschenswert. Alle benötigten Funktionalitäten sind entweder im Sensor oder im Beobachter vorhanden. Für eine Realisierung müssten diese kombiniert und unter Umständen leicht angepasst werden.

- Die Verwendung des Sleep-Modus des μController konnte bis zum Abschluss dieser Arbeit erst zu Testzwecken implementiert werden. Eine weitere Arbeit könnte dies vervollständigen.

- Wird das Bluetooth-Modul ausgeschaltet blockiert eine Funktion der Systemsoftware, was dem Dispatcher verunmöglicht den μController in den Idle-Zustand zu versetzen. Die bisherige Lösung besteht darin, dies mit Hilfe eines von der Echtzeituhr ausgelösten Events vorzunehmen. Obwohl der gewünschte Effekt erreicht wird, ist diese Methode nicht sehr schön und eher als work-around zu verstehen. Wie gross der Aufwand ist, die Ursache zu suchen und eine saubere Lösung zu präsentieren kann nicht präzise abgeschätzt werden.

- Der Energieverbrauch für Bluetooth-Verbindungen in verschiedenen Modi (Park, Sniff etc.) wurde nicht gemessen. Der Einsatz dieser zusätzlichen Verbindungszustände wird zwar die Effizienz des SensorNet-Algorithmus in keiner Weise beeinflussen, doch wäre es interessant solche Messungen für Vergleichszwecke hinzuziehen zu können.

Kapitel 9

Zusammenfassung

Das Ziel dieser Arbeit bestand darin, den Einsatz von BTnodes in einem Sensornetz zu optimieren; Die Lösung hat auf Systemebene zu erfolgen
BTnodes sind kleine Netzknoten bestückt mit einem 8MHz μController und ausgerüstet mit einem Bluetooth-Modul. Diese Hardwarekonfiguration weist einen hohen Stromverbrauch auf, was die Betriebszeit der batteriebetriebenen Sensoren stark beeinträchtigt. Da das Kommunikationsmodul um ein vielfaches Energieintensiver arbeitet als die restlichen Komponenten zusammen, wird eine Lösung im Bereich der Kommunikation angestrebt: das Modul lässt sich von der Stromzufuhr Trennen.
Ein Einsatzszenario sieht die Verwendung der BTnodes in einem statischen Sensornetz zur Messung der Lufttemperatur in Behältern für den Lebensmitteltransport vor. Verschiedene Routingalgorithmen, welche die Lebenszeit von Sensornetzen erhöhen und Stromsparprotokolle, welche den Gesamtenergieverbrauch des Netzes reduzieren, wurden diesbezüglich untersucht. Als geeignetste und effizienteste Lösung bietet sich Synchronisation an.
Der entworfene und implementierte SensorNet-Algorithmus unterscheidet zwischen Sensor- und Beobachterknoten, wobei den Sensoren die Aufgabe der Temperaturmessung und der regelmässigen Datenübertragung, den Beobachtern diejenige der Datenspeicherung und Alarmierung von Personal über GSM zukommt. Sensoren und Beobachter arbeiten synchron, das Bluetooth-Modul kann zwischen Kommunikationsintervallen ausgeschaltet werden.
Es in einem ersten Schritt wird angenommen, dass sich jeder Sensor in Reichweite eines Beobachters befindet. Ein erweiterter Algorithmus sieht Kommunikation über mehrere Hops vor.
Die Resultate zeigen, dass die durchschnittliche Stromaufnahme des Bluetooth-Moduls mit SensorNet gegenüber der Kommunikation über ständig offene Verbindungen um ein vierzigfaches reduziert wird. Der Gesamtenergieverbrauch eines Sensors sinkt damit um Faktor 7 und ermöglicht mit einem Akku als Energiequelle Betriebszeiten von 150 Stunden. Lösungen zur weiteren Reduktion des Energieverbrauchs werden vorgeschlagen und sind zu Testzwecken implementiert worden, Betriebszeiten von 600 Stunden werden damit möglich.

Im Ganzen kann gesagt werden, dass das Ziel der Stromreduktion mit gutem Ergebnis errreicht worden ist. Es hat sich gezeigt, dass sich die Kommunikation über Bluetooth und insbesondere auch die BTnode-Plattform sich für Anwendungen in spezifischen Sensornetzen hervorragend eignet.

Literaturverzeichnis

[1] S. Agarwal, R.H. Katz, S.V. Krishnamurthy and S.K. Dao, 2001. Distributed Power Control in Ad-Hoc Wireless Networks. CS Division, University of California, Berkeley.

[2] G. Asada, M.Dong, T.S. Lin, F. Newberg, G. Pottie, W.J. Kaiser and H.O. Marcy, 1998. Wireless Integrated Network Sensors: Low Power System on a Chip. University of California, Los Angeles. Rockwell Science Center, California.

[3] Atmel, 2003. Datasheet Preliminary des 8bit AVR Controller, ATmega128L. http://www.atmel.com.

[4] S.Banerjee, A. Misra, 2001. Minimum Energy Paths for reliable Communication in Multihop Wireless Networks. Dept. of Computer Science, University of Maryland. IBM, T.J.Watson Research Center, Hawthorne, New York.

[5] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund and L. Thiele, 2004. Prototyping Wireless Sensor Network Applications with BTnodes. Computer Engineering and Networks Laboratory, Dept. of IT and EE, ETH Zurich, Switzerland. Institute for Pervasive Computing, Dept. of Computer Science, ETH Zuerich, Switzerland.

[6] J. Beutel, 2003. BTnodes: Topology Discovery and Multihop Networking. Computer Engineering and Networks Laboratory, TIK, ETH Zurich, Switzerland.

[7] J. Beutel, O. Kasten, M. Ringwald, 2003. BTnodes, A Distributed Environment for Prototyping Ad Hoc Networks. Research Group for Distributed Systems, Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland.

[8] J. Beutel, M. Hinz, M. Ringwald, 2003. Ad Hoc Networking of Bluetooth Devices. Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland. Research Group for Distributed Systems, ETH Zurich, Switzerland.

[9] J. Beutel, O. Kasten, M. Ringwald, 2003. BTnodes, Architecture and Applications. Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland. http://www.tik.ee.ethz.ch/~beutel/pub/talks/20030613_btnodes_platform_-wearable.pdf.

[10] J.Broch, D.A. Maltz, D.B. Johnson, Y. Hu J. Jetcheva, 1998. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. Dept. of Computer Science, Carnegie Mellon University, Pittsburgh.

[11] J.H. Chang, L. Tassiulas, 2000. Energy Conserving Routing in Wireless Ad-hoc Networks. Department of Electrical and Computer Engineering and Institute for Systems Research, University of Maryland.

[12] J.H. Chang, L. Tassiulas, 2000. Maximum Lifetime Routing in Wireless Sensor Networks. Department of Electrical and Computer Engineering and Institute for Systems Research. University of Maryland.

[13] B. Chen, K. Jamieson, H.Balakrishnan and R. Morris, 2002. Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad hoc Wireless Networks. Laboratory for Computer Science, Massachusette Institure of Technology, Cambride.

[14] C.F. Chiasserini, 2000. Routing Protocols To Maximize Battery Efficiency. Center for Wireless Communications, University of California, San Diego

[15] Y.Gelzayd, D.J.Goodman, 2002. An alternate Connection Establishment Scheme in the Bluetooth System. Dept. of Computer Engineering, Polytechnic University, Brooklyn, New York.

[16] J.P. Ebert, B. Burns and A. Wolisz, 2002. A trace-based approach for determining the energy consumption of a WLAN network interface. Technical University Berlin, Telecommunications Networks Group, Germany.

[17] L.M. Feeney, 2000. Energy-consuption Model for Performance Analysis of Routing Protocols for Mobile Ad Hoc Networks. Computer and Network Architecture Laboratory, Swedish Institute of Computer Science, Kista, Sweden.

[18] L.M. Feeney, M. Nilsson, 2001. Investigating the Energy Consuption of a Wireless Network Interface in an Ad Hoc Networking Environment. Computer and Network Architecture Laboratory, Swedish Institute of Computer Science, Kista, Sweden.

[19] L.M. Feeney, 2002. Energy Efficient Communication in Ad Hoc Wireless Networks. Computer and Network Architecture Laboratory, Swedish Institute of Computer Science, Kista, Sweden.

[20] J. Gomez, A.T. Campbell, M. Naghshineh, C. Bisdikian, 2001. Conserving Transmission Power in Wireless Ad Hoc Networks. Dept. of Electrical Engineering and Center for Telecommunikations Research, Columbia University, New York. IBM , T.J.Watson Research Center, Hawthorne, New York.

[21] P. Gupta, P.R. Kume, 1999. The Capacity of Wireless Networks. Department of Electrical and Computer Engineering, and Coordinated Science Laborytory, University of Illinois.

[22] Z.J. Haas, M.R. Pearlman, 1998. Providing Ad Hic Connectivity with Reconfigurable Wireless Networks. School of Electrical Engineering, Cornell University, Ithaca, New York.

[23] P.J.M. Havinga, G.J.M. Smit, M. Bos, 1999. Energy efficient eireless ATM design. Dept. of Computer Science, University of Twente, Netherlands.

[24] W. Heinzelmann, J.Kulik, H. Balakrishnan, 1999. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99), 174-185.

[25] LAN MAN Standards Commitee of the IEEE COmputer Society, 1999. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. ANSI/IEEE Std 802.11, 1999 Edition.

[26] D.B Johnson, D.A. Maltz, J.Broch, 2001. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. Computer Science Dept., Carnegie Mellon University.

[27] D.B. Johnson, D.A. Maltz 1996. Dynamic Source Routing in Ad Hoc Wireless Networks. Computer Science Dept., Carnegie Mellon University.

[28] C. Jones, K.M. Sivalingam, P.Agrawal and J.C. Chen 2001. A Survey of Energy Efficient Protocols for Wireless Networks. BBN Technologies, Cambridge. School of EECS Washington State University. Telcordia Technologies, Morristown. Wireless Networks 7, 242-358, Kluver Academic Publishers.

[29] O.Kasten, M. Langheinrich, 2001. First Experiences with Bluetooth in the Smart-Its Distributed Sensor Network. Swiss Federal Institute of Technology, Zurich, Switzerland.

[30] O. Kasten, 2002. Hackfest, hands-on BTnode experience. Slides for BTnode hackfest. Research Group for Distributed Systems, Swiss Federal Institute for Technology, Zurich, Swizerland.

[31] R. Kravets, K. Schwan, Ken Calvert, 1999. Power-Aware Communication for Mobile Computers. College of Computing, Georgia Institute of Technology, Atlanta, Georgia.

[32] M. Leopold, M.B. Dydensborg, P. Bonnet, 2003. Bluetooth and Sensor Networks: A Reality Check. Dept. of Computer Science, University of Copenhagen. Sensys '03, ACM 1-58113-707-9/03/0011.

[33] P. Levis, D. Culler, 2002. Maté: A Tiny Virtual Machine for Sensor Networks. Computer Science Division, University of California, Berkeley, California. Intel Research, Berkeley Intel Corporation, Bekeley, California.

[34] L. Li, J.Y. Halpern, 2001. Minimum-Energy Mobile Wireless Networks Revisited. Cept. of Computer Science, Sornell University, Ithaca, New York.

[35] Q. Li, J. Aslam, D. Rus, 2001. Online Power-aware Routing in Wireless Ad Hoc Networks. Dept. of COmputer Science, Darmouth College, Hanover, NH.

[36] C. Lu, B.M. Blum, T.F. Abdelzaher, J.A. Stankovic, T. He, 2002. RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor Networks. Dept. of Computer Science, University of Virginia.

[37] H. Lundgren, D. Lundgren, J. Nielsen, E. Nordström, C. Tschudin, 2001. A Large-scale Testbed for Reproducible Ad Hoc Protocol Evaluations. Uppsala University, Sweden. Ericsson Research, Switchlab, Sweden.

[38] R. Mettala, 1999. Bluetooth Protocol Architecture, Bluetooth White Paper 1.C.120/1.0, SIG, http://www.bluetooth.org.

[39] A. Misra, S. Banerjee, 2002. MPRC: Maximizing Network Lifetime for Reliable Routing in Wireless Environments. IBM, T.J.Watson Research Center, Hawthorne, New York. Dept. of Computer Science, University of Maryland.

[40] S. Narayanaswamy, V. Kawadia, R.S. Screeniva, P.R. Kumar, 2002. Power Control in Ad Hoc Networks: Theory, Architecture, Algorithm and Implementation of the COMPOW Protocol. Dept. of Electrical and Computer Engineering, and Coordinated Science Laboratory, University of Illinois, USA.

[41] C.E. Perkins, P. Bhagwat, 1994. Highly Dynamic Destination-Sequences Distance vector Routing (DSDV) for mobile Computers. IBM, T.J.Watson Research Center, Hawthorne, New York. Computer Science Dept., University of Maryland.

[42] C.E. Perkins, E.M. Royer, 1997. Ad Hoc On-Demand Distance Vector Routing. Sun Microsystems Laboratories, Advanced Development Group, California. Dept. of Electrical and Computer Engineering, University of California, Santa Barbara, California.

[43] R. Ramanatham, R.Rosales-Hain, 2000. Topology Control of Multihop Wireless Networks using Transmit Power Adjustment. Internetwork Research Dept., BBN Technologies, Cambridge, Massachusetes.

[44] V. Rodoplu, T.H. Meng, 1998. Minimum Energy Mobile Wireless Networks. ??

[45] E.M. Royer, C-K. Toh, 1999. A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks. University of California, Santa Barbara, California. Georgia Institute of Technology, Atlanta, Georgia.

[46] M. Sanchez, P. manzoni, Z.J. Haas, 1999. Determination of Critical Transmission Range in Ad Hoc Networks. Depatamento de Informática de Sistemas Computadores, Universidad Politécnica de Valencia, Spain. School of Electrical Engineering, Cornell University, Ithaca, New York.

[47] C. Schurgers, V. Tsiatsis, S. Ganeriwal, M. Srivastava, 2002. Optimiying Sensor Networks in the Energy-Latency-Density Design Space. IEEE Transactions on mobile computing, Vol. 1, No. 1, Jan-Mar 02.

[48] E. Shih, P. Bahl, M.J. Sinclair, 2002, Wake on Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. Massachusetts Institute of Technology, Cambridge. Microsoft Research, Redmond, WA.

[49] F. Siegemund, C. Flörkemeier, 2003. Interaction in Pervasive Computing Settings using Bluetooth-enabled Active Tags and Passive RFID Technology together with Mobile Phones. Institure for Pervasive Computing, Dept. of Computer Science, ETH Zurich, Switzerland. Proceedings of PerCom 2003 (IEEE International Conference on Pervasive Computing and Communications), pp. 378-387, March 2003.

[50] F. Siegemund, M. Rohs, 2002. Rendezvous Layer Protocols for Bluetooth Enabled Smart Devices. Institute of Information Systems, ETH Zurich.

[51] S.Singh, M. Woo, C.S. Raghavendra, 1998. Power-Aware Routing in Mobile Ad-Hoc Networks. Dept. of ECE, Oregon State University, Corvallis, Oregon. Aerospace Corporation, El Segundo, California.

[52] S. Singh, C.S. Raghavendra, 1999. PAMAS - Power Aware Multi-Access protocol with Signalling for Ad Hoc Networks. Dept. of ECE, Oregon State University, Corvallis, Oregon. Aerospace Corporation, El Segundo, California.

[53] M. Stemm, R.H. Katz, 1997. Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices. Computer Science Division, Dept. of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, Canada.

[54] I. Strojmenovitc, X. Lin, 2000. Power-aware localized routing in wireless networks. Computer Science, University of Ottawa, Ontario, Canada.

[55] D. Tian, N.D. Georganas, 2002. A Coverage-Preserving Node Scheduling Scheme for Large Wireless Sensor Networks. Multimedia Communications Research Laboratory University of Ottawa, Canada.

[56] J.Tian, I. Stepanov, K. Rothermel, 2002. Spatial Aware Geographic Forwarding for mobile Ad Hoc Networks. Universität Stuttgart, Germany.

[57] S. Tilak, N.B. Abu-Ghazaleh, W. Heinzelman, 2002. A Taxonomy of Wireless Micro-Sensor Network Models. Computer System Research Laboratory, Dept. of CS Binghamption, Binghamption University, New York. Electrical and Computer Engineering, University of Rochester, Rochester, New York.

[58] C.K. Toh, 2001. Maximum Battey Life Routing to Support Ubiquitous Mobile Computing in Wireless Ad Hoc Networks. Georgia Institute of Technology, Atlanta, Georgia.

[59] R. Wattenhofer, L. Li, Paramvir Bahl, Y. Wang, 2001. Distributed Topology Control for Power Efficient Operation in Multihop Wireless Ad Hoc Networks. Microsoft Research, Redmond, WA. Computer Science Dept., Cornell University, Ithaca, New York.

[60] J.E. Wieselthier, G.D. Nguyen, A. Ephremides, 2000. On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks. Naval Research Laboratory, Washington DC. EE Dept. and Institute of Systems Research, University of Maryland.

[61] H. Woesner, J.-P. Ebert, Adam Wolisz, 2001. Power Saving Mechanisms in Emerging Standards for Wireless LANs: the MAC Level Perspective. Telecommunication Networks Group, Technical University Berlin, Berlin, Germany.

[62] J. Wu, M. Gao, 2000. On Calculating Power-Aware Connected Dominating Sets for Efficeint Routing in Ad Hoc Wireless Networks. Dept. of Computer Science, Florida Atlantic University. University of Ottawa, Ottawa, Ontario.

[63] Y. Xu, 2000. Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks. Information Science Institute, Marina del Rey, California.

[64] Y. Xu, J. Heidermann, D. Estrin, 2001- Geography-informed Energy Conservation for Ad Hoc Routing. Information Schience Institute, Marina del Rey, California. Computer Science Dept., University of California, Los Angeles, California.

Anhang A

Messwerte

A.1 Messwerte Bluetooth-Modul

Komponente, Modus	P Typ 01	P Typ 02
Bluetooth, Inquiry	162mW (48.4mA)	132mW (39.21mA)
Bluetooth, Connecting (page)	168mW (50.1mA)	134mW (39.4mA)
Bluetooth, Connection master	147mW (43.9mA)	118mW (35.1mA)
Bluetooth, Connection slave	146mW (43.5mA)	116mW (34.6mA)
Bluetooth, Sending Data	164mW (48.8mA)	127mW (37.8mA)
Bluetooth, Receiving Data	151mW (44.9mA)	119mW (35.6mA)
Bluetooth, Idle typ	53.4mW (15.9mA)	19.8mW (5.90mA)
max	105mW (31.4mA)	70.2mW (20.9mA)
Bluetooth, Radio Off	7.08mW (2.11mA)	7.06mW (2.10mA)
Bluetooth, Power Down	0mW	0mW

A.2 Messwerte μController, LED, SRAM

Komponente, Modus	P_{max}	P_{typ}
CPU, Active	30.9 mW (9.2mA)	15.0 mW (4.45mA)
CPU, ADC Noise Reduction		5.88 mW(1.75mA)
CPU, Idle		3.66 mW (1.09mA)
CPU, Standby		2.35 mW (0.699mA)
CPU, Extended Standby		2.38 mW (0.708mA)
CPU, Power Save		0.057 mW (0.017mA)
CPU, Power Down		0.037 mW (0.011mA)
LEDs (4 LED, 1 LED)	51.9mW (15.45mA)	13.6mW (4.05mA)
SRAM	14.15mW(4.21mA)	0.591mW (0.176mA)

A.3 Messwerte μController, reduzierte Taktrate

Taktfrequenz	P_{max}	P_{typ}
8 MHz	30.9 mW (9.2mA)	15.0 mW(4.45mA)
4 MHz (d=127)	19.9 mW (5.92mA)	9.34 mW(2.78mA)
2.7 MHz (d=126)	16.1 mW (4.79mA)	7.36 mW(2.19mA)
2 MHz (d=125)	14.2 mW (4.23mA)	6.55 mW(1.95mA)
1.6 MHz (d=124)	12.9 mW (3.86mA)	5.85 mW(1.74mA)
1.3 MHz (d=123)	12.1 mW (3.59mA)	5.38 mW(1.60mA
1.1 MHz (d=122)	11.8 mW (3.51mA)	5.07 mW(1.51mA)
1 Mhz (d=121)	11.4 mW (3.40mA)	5.04 mW(1.50mA)
500 kHz (d=113)	9.84 mW (2.93mA)	4.33 mW(1.29mA)
250 kHz (d=97)	9.14 mW (2.72mA)	3.80 mW(1.13mA)
125 kHz (d=65)	8.77 mW (2.61mA)	3.63 mW(1.08mA)

A.4 Durchschnittliche Leistungsaufnahme eines Sensorknotens

Komponente	A [mW]	B [mW]
Bluetooth-Modul	$(\frac{1}{60} * 127) + (\frac{59}{60} * 118) = 118.15$	118.15
μController	30.9	15.0
SRAM	0.6	0.6
Summe	149.7	133.75

Komponente	C [mW]	D [mW]
Bluetooth-Modul [1]	$\frac{1}{12} * 33.9 + \frac{11}{12} * \frac{(19.8+70.2)}{2} = 44.1$	44.1
μController	30.9	15.0
SRAM	0.6	0.6
Summe	52.8	36.9

Komponente	E [mW]	F [mW]
Bluetooth-Modul [1]	$\frac{1}{12} * 33.9 + \frac{11}{12} * 0 = 2.83$	2.82
μController	30.9	15.0
SRAM	0.6	0.6
Summe	34.3	18.4

Komponente	G [mW]
Bluetooth-Modul	2.83
μController	$\frac{1}{12} * 15.0 + \frac{11}{12} * 0.06 = 1.3$
SRAM	0.6
Summe	4.73

Anmerkung:

Alle in A.4 verwendeten Messwerte können, sofern nicht anders vermerkt, vorausgegangenen Tabellen entnommen werden.
[1]Der Messwert von 33.9 mW entspricht der durchschnittlichen Leistungsaufnahme während dem Wachzustand der Sensorknoten in SensorNet. Als Grundlage dienten die in Abbildung 8.5 dargestellten Werte.

Anhang B

SensorNet API

In diesem Kapitel sind die Befehle und Events aufgeführt, welche durch das SensorNet Observer- und Sensor-API zur Verfügung gestellt werden.

B.1 Sensor API

B.1.1 Funktionen

u8 sensor_get_state()

Diese Funktion gibt den Zustand des Sensors zurück. Mögliche Werte sind:

- STATE_AWAKE: Der Sensor befindet sich im Wachzustand, das Bluetooth-Modul ist eingeschaltet und einsatzbereit.
- STATE_ASLEEP: Der Sensor befindet sich im Schlafzustand, das Bluetooth-Modul ist ausgeschaltet.
- STATE_GOING_TO_SLEEP: Das Bluetooth-Modul wird in Kürze vom Strom getrennt und der Sensor geht in einen energiesparenden Zustand über, Verbindungen sind nicht mehr erlaubt.
- STATE_WAKING_UP: Das Bluetooth-Modul des Sensors ist mit Strom versorgt und wird in den nächsten Sekunden einsatzbereit sein.

u8 sensor_get_substate()

Der Rückgabewert zeigt den Substate an, falls sich der Sensor im AWAKE-Zustand befindet:

- SUBSTATE_IDLE: Der Sensor befindet sich in keinem Substate. Dieser Fall nur auf, falls der Sensor nicht wach ist.
- SUBSTATE_SYNC: Der Sensor sucht einen Beobachter oder synchronisiert sich mit einem solchen.
- SUBSTATE_SEND: Der Sensor ist sendebereit und wartet auf eine entsprechende Aufforderung oder er überträgt Sensordaten.

sensor_get_observer_addr(bt_addr_t observer_addr)

Diese Funktion gibt die Adresse des aktuellen Observers zurück. Besteht keine Synchronisation, so wird die Adresse des letzten gefundenen Beobachters übergeben.

u32 sensor_get_request_interval_ms()

Der Rückgabewert dieser Funktion entspricht dem mit dem Observer vereinbarten request-Intervall, also dem Zeitraum zwischen zwei Datenübetragungen in Millisekunden.

u32 sensor_get_inquiry_interval_ms()

Diese Funktion ermittelt die Länge des inquiry-Intervalles, jenem Zeitraum zwischen zwei aufeinanderfolgenden inquiry-Operationen bei der Suche nach einem Observer.

void sensor_set_inquiry_interval_ms(u32 inq_int)

Mit Hilfe dieser Funktion kann das inquiry-Intervall festgelegt werden. Das Inquiry-Intervall sollte auf jeden Fall kleiner sein als das ready-to-sync-Intervall des Beobachters.

u32 sensor_get_observer_lost_timeout_ms()

Hat der Sensor nach Ablauf diese Intervalles noch keine Sensordaten übertragen, sucht er einen neuen Beobachterknoten. Die Länge dieses Intervalles ist abhängig vom request-Intervall und wird bei bei der Synchronisation automatisch angepasst.

void sensor_init(u16 argc, char *argv[])

Diese Funktion initialisiert den Sensor, unter anderem werden das System und Treiber eingerichtet und benötigte Events registriert.

void sensor_run()

Mit dem Aufruf dieser Funktion wird die Ausführung des Sensors resp. der Dispatchers gestartet.

void sensor_set_verbosity(u8 v)

Diese Funktion legt fest, wie viele Ausgaben über UART zu erfolgen haben erfolgen.

- v = 0: Keine Ausgabe von Debug-Informationen
- v = 1: Nur Informationen zur Synchronisation und zur Datenübertragung werden ausgegeben.

- v = 2: Zusätzlich wird über die Zustände des Sensors informiert.
- v = 3: Alle Informationen werden ausgegeben. Wie v = 2, ausserdem werden Verbindungszustände und Timerwerte angezeigt.

B.1.2 Events

Event	Funktion
ASLEEP_EVENT	Signalisiert dass der Sensor eingeschlafen, das Bluetooth-Modul ausgeschaltet ist.
AWAKE_EVENT	Signalisiert dass der Sensor aufgewacht und das Bluetooth-Modul einsatzbereit ist.
GOING_TO_SLEEP_EVENT	Signalisiert das Einschlafen des Sensors, das Bluetooth-Modul wird in den nächsten Sekunden ausgeschaltet.
WAKING_UP_EVENT	Signalisiert das Aufwachen des Sensors, das Bluetooth-Modul wird in Kürze einsatzbereit sein.
SEARCH_OBSERVER_EVENT	Dieses Event zeigt an, dass ein Observer gesucht wird.
OBSERVER_FOUND_EVENT	Signalisiert das Auffinden eines geeigneten Beobachterknotens.
OBSERVER_LOST_EVENT	Signalisiert, dass der Kontakt zum Beobachter abgerissen ist. Dieses Ereignis tritt nach observer-lost-Intervall auf, falls dazwischen keine Sensordaten gesendet werden konnten.
CHECK_OBSERVER_LOST_ EVENT	Signalisiert ein Überpüfen der observer-lost-Bedingung.
SYNCHRONISATION_ ESTABLISHED_EVENT	Signalisiert das erfolgreiche Synchronisieren mit einem Beobachter.
CHECK_SCHEDULE_EVENT	Dieses Event zeigt an, dass der Schedule, resp. die Aufwachbedingungen überprüft wurden.

B.2 Observer API

B.2.1 Funktionen

u8 observer_get_state()

Der Rückgabewert dieser Funktion entspricht dem Zustand des Beobachter. Mögliche Werte sind:

- STATE_AWAKE: Der Beobachter befindet sich im Wachzustand, das Bluetooth-Modul ist eingeschaltet und einsatzbereit.
- STATE_ASLEEP: Der Beobachter befindet sich im Schlafzustand, das Bluetooth-Modul ist ausgeschaltet.
- STATE_GOING_TO_SLEEP: Das Bluetooth-Modul wird in Kürze vom Strom getrennt und der Beobachter geht in einen energiesparenden Zustand über, Verbindungen sind nicht mehr erlaubt.
- STATE_WAKING_UP: Das Bluetooth-Modul des Beobachters ist mit Strom versorgt und wird in den nächsten Sekunden einsatzbereit sein.

u8 sensor_get_substate()

Diese Funktion liefert den Substate des Beobachter zurück:

- SUBSTATE_IDLE: Der Beobachter befindet sich in keinem Substate. Dieser Fall nur auf, falls der Beobachter nicht wach ist.
- SUBSTATE_SYNC: Der redy-to-synchronize-Zustand zeigt an, dass der Beobachter bereit ist eintreffende Synchronisationsanfragen zu beantworten.
- SUBSTATE_REQUEST: Der Beobachter ist dabei Verbindungen mit Sensoren herzustellen, um Daten anzufordern.

void observer_set_request_interval_ms(u32 req_int)

Diese Funktion legt die Länge des request-Intervall, also den Zeitraum zwischen dem Anfordern von Sensordaten in Millisekunde, fest.

u32 observer_get_request_interval_ms()

Diese Funktion liefert als Rückgabewert die Länge des request-Intervalles in Millisekunden.

void observer_set_ready_to_sync_interval_ms(u32 sync_int)

Diese Funktion setzt die Länge des ready-to-synchronize-Intervalles in Millisekunden. Während diesem Zeitraum ist der Observer wach und kann Synchronisationsanfragen beantworten. Die Länge dieses Intervalles muss auf jeden Fall kleiner der Länge des request-Intervalles sein.

u32 observer_get_ready_to_sync_interval_ms()

Der Rückgabewert dieser Funktion entspricht der Länge des ready-to-synchronize-Intervalles in ms.

void observer_set_sensor_data_request_interval_ms(u32 data_req_int)

Mit dieser Funktion lässt sich der zeitliche Abstand zweier aufeinanderfolgender aktiver Verbindungsaufbauten festlegen. Die Länge dieses Intervalles muss grösser sein als das verbindungs-Timeout.

u32 observer_get_sensor_data_request_interval_ms()

Diese Funktion liefert die Länge des Zeitraumes zwischen dem Herstellen zweier Verbindungen zurück.

void observer_set_connection_timeout_ms(u32 conn_timeout)

Das connection-Timeout lässt sich mit dieser Funktion anpassen. Dieses Timeout ist einerseits zuständig für das Abbrechen erfolgloser Verbindungsversuche. Auf der anderen Seite werden nach Ablauf des Timeouts, falls während dieser Zeit keine explizite Kommunikation stattfindet, Verbindungen geschlossen.

u32 observer_get_connection_timeout_ms()

Der Rückgabewert ist entspricht der Länge des connection-Timeouts in Millisekunden.

void observer_set_max_nof_sensors(u16 max_nof_sensors)

Diese Funktion legt fest, wie viele Sensoren maximal durch den Beobachter verwaltet werden können.

void observer_get_max_nof_sensors(u16 max_nof_sensors)

Diese Funktion legt die maximale Anzahl verwalteter Sensoren durch den Beobachter fest.

u16 observer_get_nof_of_sensors()

Diese Funktion gibt die Anzahl synchronisierter Sensoren zurück. Der Wert muss als obere Grenze verstanden werden (der letzte belegte Slot); geht der Kontakt zu Sensoren zwischenzeitlich verloren, so muss wirkt sich das nicht direkt auf den Rückgabewert dieser Funktion aus.

void sensor_set_verbosity(u8 v)

Diese Funktion legt fest, wie viele Ausgaben über UART zu erfolgen haben erfolgen.

- v = 0: Keine Ausgabe von Debug-Informationen
- v = 1: Nur Informationen zur Synchronisation und zur Datenübertragung werden ausgegeben.

- v = 2: Zusätzlich wird über die Zustände des Sensors informiert.
- v = 3: Alle Informationen werden ausgegeben. Wie v = 2, ausserdem werden Verbindungszustände und Timerwerte angezeigt.

B.2.2 Events

Event	Funktion
ASLEEP_EVENT	Signalisiert, dass der Beobachter eingeschlafen und das Bluetooth-Modul ausgeschaltet ist.
AWAKE_EVENT	Signalisiert, dass der Beobachter aufgewacht und das Bluetooth-Modul einsatzbereit ist.
GOING_TO_SLEEP_EVENT	Signalisiert das Einschlafen des Beobachters, das Bluetooth-Modul wird in den nächsten Sekunden ausgeschaltet.
WAKING_UP_EVENT	Signalisiert das Aufwachen des Sensors, das Bluetooth-Modul wird in Kürze einsatzbereit sein.
SENSOR_DATA_RECEIVED_EVENT	Zeigt den Empfang von Sensordaten an.
Check_SCHEDULE_EVENT	Zeigt an, dass der Beobachter-Schedule geprüft worden ist.
GET_SENSOR_DATA_EVENT	Signalisiert, dass der Beobachter mit dem Anfordern der Sensordaten beginnt. In den nächsten Sekunden werden wahrscheinlich Events des Typs SENSOR_DATA_RECEIVED_EVENT auftreten.

Anhang C

Quelldateien

Im folgenden sind alle Quelldateien aufgeführt, die im Rahmen dieser Arbeit erstellt wurden.

Datei	Beschreibung
$sensornet\backslash sensor\backslash sensornet\_sensor\_impl.c$	SensorNet Sensoranwendung
$sensornet\backslash sensor\backslash sensornet\_sensor.c$	SensorNet Implementation des Sensors
$sensornet\backslash sensor\backslash sensornet\_sensor.h$	SensorNet Header für Sensorimplementation
$sensornet\backslash sensor\backslash Makefile$	Makefile zur Kopilierung des Sensorcodes
$sensornet\backslash observer\backslash sensornet\_observer\_impl.c$	SensorNet Observeranwendung
$sensornet\backslash observer\backslash sensornet\_observer.c$	SensorNet Implementation des Observers
$sensornet\backslash sensor\backslash sensornet\_observer.h$	SensorNet Header für Observerimplementation
$sensornet\backslash bt\_devices\backslash bt\_devices.c$	Verwaltung von Bluetooth-Geräten
$sensornet\backslash bt\_devices\backslash bt\_devices.h$	Header zur Verwaltung von Bluetooth-Geräten
$sensornet\backslash bt\_devices\backslash Makefile$	Makefile zur Kompilierung der Geräteverwaltung
$sensornet\backslash sensor\_data\backslash sensor\_data.c$	Funktionen zur Verwaltung und Übertragung von Sensordaten
$sensornet\backslash sensor\_data\backslash sensor\_data.h$	Header für Sensordaten
$sensornet\backslash sensor\_data\backslash Makefile$	Makefile zur Kompilierung der Implementation für Sensordaten

Anhang D

Quellcode

D.1 Sensor-Anwendung

```
/* Dieses Programm implementiert exemplarisch einen einfachen
 * SensorNet-Sensor, welcher alle 10 Sekunden einen zufälligen
 * Messwert erzeugt.*/

#include "sensorNet_sensor.h"
#include <random.h>

#define MY_EVENT 100
u16 my_data_tag = 0;
u16 my_sensor_id = 1;

void my_function( call_data_t call_data, cb_data_t cb_data){

      printf("my_function called, sensor value generated" NL);

      sensor_data_element_t s;
      s.source = my_sensor_id;
      my_data_tag++;
      s.tag = my_data_tag;
      s.value = (u16) (btn_rand() / 65000);
      sensor_data_add_sensor_data_element(s);

      btn_timeout_reg(MY_EVENT, 10000, 0);
}


int main(int argc, char *argv[]){

      sensor_init(argc, argv);
      btn_disp_ev_reg(MY_EVENT, my_function, MY_EVENT);
      btn_timeout_reg(MY_EVENT, 10000, 0);
```

```
      sensor_run();
      return(0);
}
```

D.2 Beobachter-Anwendung

```
/* Dieses Programm implementiert exemplarisch einen einfachen
 * SensorNet-Beobachter, der alle 45 Sekunden Sensordaten
 * empfängt und diese auf der Konsole ausgibt. */


#include "sensorNet_observer.h"

void my_function( call_data_t call_data, cb_data_t cb_data){
      printf("sensor data received" NL);
      sensor_data_print_all();

}

int main(int argc, char *argv[]){

      observer_set_request_interval_ms(45000);
      observer_set_ready_to_sync_interval_ms(25000);

      observer_init(argc, argv);
      btn_disp_ev_reg(SENSOR_DATA_RECEIVED_EVENT, my_function, \
                                    SENSOR_DATA_RECEIVED_EVENT);

      observer_run();
      return(0);
}
```

Anhang E

Pakettypen BaseBand

Pakettyp	Headerlänge [bytes]	Payload [bytes]	Forward Error Correction	Datenübertragungsrate [kbs]		
				synchron	asynchron	
					hin	zurück
DM 1	1	0-17	2/3	108.8	108.8	108.8
DH 1	1	0-27	nein	172.8	172.8	172.8
DM 3	2	0-121	2/3	258.1	387.2	54.4
DH 3	2	0-183	nein	390.4	585.6	86.4
DM 5	2	0-224	2/3	286.7	477.8	36.3
DH 5	2	339	nein	433.9	723.2	57.6

Abbildung E.1: Baseband-Datenpakettypen

Anhang F

Glossar

Akronym	Bedeutung
ACL	Asynchronous Connectionless Link
ADC	Analog Digital Converter
AFECA	Adaptive Fidelity Energy Conserving Algorithm
AODV	Ad Hoc On Demand Vector Routing
ATIM	Ad Hoc Traffic Indication Message
BB	Baseband
BEC	Backward Error Correction
BECA	Basic Energy COnserving Algorithm
BER	Battery Efficient Routing
CMMBCR	Conditional Min Max Battery Capacity Routing
CoD	Class of Device
CCS	Complete Connecting Set
CS	Connecting Set
DS	Dominating Set
DSP	Dynamic Source Routing
DVR	Distance Vector Routing
EEPROM	Electric Erasable ..
FEC	Forward Error Correction
FHS	Frequency Hopping Sequence
GAF	Geographic Adaptive Fidelity
GP	General Purpose
GIAC	GEneral Inquiry Access Code
HCI	Host Controller Interface
I/O	Input/Output
IP	INternet Protocol
IrDA	Infrared Data Association
ISM	Industry Science Medicine
ISP	In System Programming
JTAG	Joint Test Action Group (IEEE 1149.1)
L2C	
L2CAP	Logical Link Layer and Adaption Protocol
LED	Light Emitting Diode
LMP	Link Manager Protocol

Akronym	Bedeutung
LSR	Link State Routing
MAC	Medium Access Control
MCU	Microcontroller Unit
MCR	Minimum Cost Routing
MER	Maximum Energy Routing
MLR	Maximum Lifetime Routing
OBEX	Object Exchange Protocol
PCM	Pulse Code Modulation (Audio)
RERR	Route Error
RREP	Route Reply
RREQ	Route Request
RF	
RFCOMM	
ROK	
RTC	Real Time Clock
SCO	Synchronous Connection Orientated
SDP	Service Discovery Protocol
SPI	System Programming Interface
SRAM	
TCS	Telephony Control System
TCP	Transport Control Protocol
UART	
UDP	
USB	Universal Serial Bus
WAP	Wireless Applocation Protocol
XDIV	X-Division Register