Ressourcenoptimiertes Netzwerkprotokoll für Miniatur-Sensorknoten

Diplomarbeit am Telecooperation Office (TecO)
Prof. Dr. W. Juling
Fakultät für Informatik
Universität Karlsruhe (TH)

von

cand. inform.
Patrik Spieß

Betreuer:
Prof. Dr. W. Juling
Dr.-Ing. Michael Beigl

Tag der Anmeldung: 1. März 2004
Tag der Abgabe: 31. August 2004

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 31. August 2004

Inhaltsverzeichnis

1. Einleitung

Drahtlose Sensornetzwerke werden eingesetzt, um digitalen informationsverarbeitenden Systemen eine Sicht auf die sie umgebende reale Welt zu ermöglichen. Aus den gewonnenen Sensordaten werden Kontexte (d. h. abstrahierte Aussagen über die Umgebung des Sensors) erzeugt, kombiniert und verarbeitet.

Hierzu zwei Beispiele: Eine Gebäudesteuerung kann mit einer gewissen Wahrscheinlichkeit annehmen, dass gerade keine Personen im Haus sind (weil die Bewegungsmelder keine Aktivität feststellen) und die Umgebung entsprechend anpassen. Ein Türschild an einem Konferenzraum kann wissen und anzeigen, dass ein Meeting stattfindet, weil Mitarbeiter ihre mit Sensoren ausgestatteten Kaffeetassen mitgenommen haben – der Intranet-Server des Unternehmens kann den Raum als belegt ausweisen, die Telefonanlage kann Anrufe der Teilnehmer auf ein Mailboxsystem weiterleiten.

Am TecO[1] wurde die Particle-Plattform entwickelt, die es ermöglicht, schnell und modular Anwendungen, wie die gerade beschriebenen, in den Bereichen mobile und drahtlose Ad-hoc-Sensornetzwerke zu realisieren. Mit Hilfe der Particles können Produkte wie z. B. intelligente und vernetzte Alltagsgegenstände, Computer in Textilien und Sensoren zur Gebäudewahrnehmung entwickelt werden. Sie können u. a. für Systeme zur Umweltüberwachung, für Logistik und Supply-Chain-Management verwendet werden.

Das zu Grunde liegende Netzwerk konfiguriert sich ohne menschliches Eingreifen selbstständig. Ein Knoten baut nach dem Einschalten mit anderen Knoten vor Ort ein so genanntes Ad-hoc-Netzwerk auf, das sofort betriebsbereit ist. Die Knoten sollen möglichst wenig Energie verbrauchen, da sie durch mobile, in manchen Fällen sogar nicht austauschbare Energiequellen versorgt werden. Da sie unter Umständen in großer Menge und Dichte betrieben werden, sollen die Kosten pro Knoten möglichst niedrig sein. Die über Funk realisierbare Bandbreite ist begrenzt, daher muss ihre konkurrierende Nutzung möglichst effizient erfolgen.

[1] Telecooperation Office des Instituts für Telematik an der Universität Karlsruhe, Website: http://www.teco.edu/

1.1 Zielsetzung der Arbeit

Die bisherige Particle-Implementierung des TecO, das PIC-Particle, basiert auf einem Board mit TR1001 Transceiver von RF Monolithics Inc.[2] und einem PIC 18F6720 Mikrocontroller von Microchip Technology Inc.[3] Der Stückpreis des Boards liegt bei ca. 110,- Euro.

Um die Kosten für ein Particle auf etwa ein Drittel dieses Preises zu senken, wurde beschlossen, die Particle-Plattform auf eine neue Hardware, den CC1010-Chip von Chipcon[4], zu portieren. Die Zielsetzung dieser Diplomarbeit ist die softwareseitige Anpassung der Netzwerkarchitektur an die veränderten Hardwarekomponenten der Chipcon Particles. Hierdurch ergeben sich insbesondere folgende Aufgaben:

- Entwurf von neuen unteren Netzwerkschichten (Bitübertragungsschicht und Medienzugriffs-Teil der Sicherungsschicht) unter Berücksichtigung der neuen Hardware.
- Optimierung der neuen Implementierung bezüglich Energieverbrauch, automatische Konfiguration und Skalierbarkeit.

1.2 Gliederung der Arbeit

Im Kapitel Grundlagen wird die vorliegende Arbeit in das Gebiet der Ubiquitären Informationssysteme eingeordnet, indem vergleichbare Ansätze anderer Forscher dargestellt und relevante Konzepte erläutert werden, die in die Arbeit eingeflossen sind. Im Kaptitel Protokollentwurf wird zunächst eine Motivation für den gewählten Ansatz gegeben und ein Überblick über das im Rahmen dieser Arbeit erstellte Protokoll skizziert. Darauf folgt eine detaillierte Beschreibung der einzelnen Komponenten sowie eine Bewertung des Entwurfs. Das Kapitel Implementierung beschreibt die verwendete Hard- und Software und deren Zusammenspiel bei der Implementierung des Protokolls. Eine Bewertung der Implementierung liefert das Kapitel Evaluation. Der Abschnitt Zusammenfassung und Ausblick schließt die Arbeit ab. Im Anhang werden einige Implementierungsdetails genauer erläutert sowie Glossar, Index und Abbildungsverzeichnis bereitgestellt.

1.3 Konventionen

Zahlen im Hexadezimalsystem werden mit dem Index *hex* und Zahlen im Binärsystem mit dem Index *bin* dargestellt, z. B. $4F3B_{hex}$, 11010110_{bin}.

Die PDF-Online-Version dieses Dokuments benutzt Hyperlinks, das sind Textstellen, die beim Anklicken mit der Maus zu anderen Dokumentteilen oder WWW-Adressen weiterleiten. Diese sind in dunkelblauer Farbe gehalten. Beispiele: Interner Link: Kapitel 1.3, Externer Link: http://www.teco.edu/~spiess/, etc.

[2] http://www.rfm.com/
[3] http://www.microchip.com/
[4] http://www.chipcon.com/

Fachbegriffe werden bei der ersten Verwendung auf Deutsch angegeben. Gibt es für den Begriff einen äquivalenten und geläufigen englischen Begriff so wird dieser in Klammern oder in einer Fußnote angegeben. Nach der ersten Erwähnung wird die jeweils üblichere Version des Fachbegriffs (also entweder die deutsche oder die englische) verwendet.

2. Grundlagen

2.1 Ubiquitäre Informationssysteme und drahtlose Sensornetzwerke

Um diese Arbeit einordnen zu können, muss man den Kontext verstehen, in dem die gegebene Anwendung angesiedelt sind. Hierbei handelt es sich um das Feld der drahtlosen Sensornetzwerke. Die Idee dahinter ist es, alltäglichen Gegenständen, Kleidung, Gebäuden oder sogar ganzen Ökosysteme eine „Sinneswahrnehmung“ zu ermöglichen. Dabei werden die zu erweiternden Systeme mit kompakten, langlebigen Sensoren ausgestattet, die drahtlos – also über Funk – miteinander kommunizieren können.

Meistens werden die von einem Sensornetzwerk erfassten Daten nicht von ihm direkt verarbeitet. Die Knoten sind oft nur mit geringer Rechenleistung und Speicherkapazität ausgerüstet. Daher müssen Schnittstellen zur Interaktion mit einem leistungsfähigen Hintergrundsystem definiert werden.

Die über die Umgebung gesammelte Information kann dann verwendet werden, um dem Menschen zusätzlichen Nutzen zu bieten. Der zusätzliche Nutzen soll sich dem Nutzer intuitiv erschließen, die Anwendungen sollen sich an den Nutzer anpassen, nicht der Nutzer an die Anwendung – wie es z. B. beim PC der Fall ist, der den Benutzer zwingt, zur Interaktion eine auf die Maschine zugeschnittene Schnittstelle zu benutzen (Tastatur/Monitor).

Dieses noch weiter übergeordnete Feld, das sich mit der Aufbereitung und Nutzung der gesammelten Daten befasst, ist der Forschungsbereich der *Ubiquitären Informationssysteme*. Es wurde wesentlich beeinflusst durch die Arbeiten von Mark Weiser et al. am Xerox Parc. [Weis91]

2.2 Kontexte

Ein Kontext ist Information über die reale Welt, die ein technisches Gerät umgibt bzw. deren abstrakte Repräsentation. Kontexte sind als Eingaben für

die laufenden Programme zu verstehen. Neben klassischen, vom Benutzer initiierten Eingaben wie Tastatureingabe können Kontexte von der Software verwendet werden und dem Benutzer einen Mehrwert bieten.

Das *WordNet Dictionary*[1] definiert den Begriff „Context“ wie folgt: The set of facts or circumstances that surround a situation or event.

Englische Synonyme für Context sind: Circumstances, Environment, Milieu, Setting, Situation, Surroundings, Situation.

Kontexte können verschieden stark abstrahiert sein. Ein einfacher Kontext ist z. B. ein gemessener Temperaturwert an einem Sensor. Ein weiterer ist die Position dieses Sensors. Letztere kann entweder aktiv ermittelt werden, weil der Sensor mobil ist, oder ist bekannt, weil der Sensor fest verortet ist. Bereits solche einfachen Kontexte ergeben für eine Anwendung nützliche Information. Wenn ein Handy oder ein PDA z. B. seine Position kennt, kann es seinem Benutzer ortsbezogene Dienste anbieten – im Zusammenspiel mit einer Datenbank oder WAP-Anwendung könnte es z. B. die Adresse des nächsten Supermarktes oder Restaurant ermitteln oder dem Benutzer gleich den Weg dorthin weisen.

Ein anderer, sehr hochwertiger Kontext wäre z. B. die automatisch ermittelte Information, welcher Benutzer gerade vor einem PC sitzt. Der PC kann sich auf den Benutzer einstellen und ihm Zugriff auf nur seine Daten gewähren. Heute wird dieser Kontext mit Hilfe des „Benutzername / Passwort“-Konstrukts ermittelt.

Kontexte können übermittelt, aggregiert und verarbeitet werden. So können z. B. mehrere Sensoren in einem Raum die Temperatur und Lautstärke messen und diese Information drahtlos verbreiten. Ein Knoten, der an einer Glocke angeschlossen ist, könnte diese Kontexte empfangen. Er filtert aus dem drahtlosen Netzwerk nur die für ihn relevanten Kontext-Informationen heraus (Temperatur) und vernachlässigt andere (Lautstärke). Der Knoten an der Glocke kann die Kontexte nun verarbeiten, indem er die Durchschnittstemperatur des Raumes errechnet. Steigt diese über 100 Grad Celsius erstellt er einen neuen, höherwertigen Kontext – Feuergefahr. Er sendet diese Information aus und betätigt über ein Relais die Glocke.

Zahllose Arten von weiteren nützlichen Kontexten sind denkbar. Auch die Anwendungen, die mit Kontexten arbeiten, unterscheiden sich in vielen Punkten. Verbreitung, Verarbeitung, Auswahl (Filterung) und Vorhaltung können jeweils sehr unterschiedlich sein.

Ein einzelner Knoten, registriert oft nur einfache Kontexte, denn er hat in der Regel keine großen Kapazitäten was Speicherplatz und Rechenkapazität betrifft. Kontexte, deren Erstellung die Kombination vieler untergeordneter Kontexte unter Verwendung aufwändiger mathematischer Modelle erfordert, können beispielsweise besser von Knoten ausgeführt werden, die über eine hohe Rechenleistung und viel Speicher verfügen (PCs, PDAs, ...). Da diese leistungsfähigeren Geräte meist nicht mit dem Sensornetzwerk direkt kommunizieren kann, da es ein proprietäres, für mobile Sensornetzwerke opti-

[1] http://www.cogsci.princeton.edu/cgi-bin/webwn

miertes Protokoll verwendet, muss ein Gateway zu gängigen Protokollfamilien (Bluetooth, W-LAN, Ethernet) bereitgestellt werden, der die Kontextdaten in die Dateneinheiten des anderen Netzes kapselt (siehe Abschnitt 2.4.1 auf Seite 9).

2.3 ISO-OSI Schichtenmodell

Um digitale Netzwerke zwischen Geräten zu beschreiben, verwendet man oft das so genannte ISO-OSI[2]-Schichtenmodell. Dabei hat man folgende Schichten festgelegt:

Nummer	Name	engl. Bezeichnung
7	Anwendungsschicht	Application Layer
6	Darstellungsschicht	Presentation Layer
5	Kommunikationsschicht	Session Layer
4	Transportschicht	Transport Layer
3	Vermittlungsschicht	Network Layer
2	Sicherungsschicht	Data Link Layer
1	Bitübertragungsschicht	Physical Layer

Jede Schicht greift auf die jeweils darunterliegende zu und stellt der darüber liegenden Dienste zur Verfügung. Je höher eine Schicht in dem Modell angeordnet ist, desto abstrakter werden die erbrachten Dienste und der damit zur Verfügung gestellte Mehrwert.

Schicht 2 kann bei konkurrierendem Medienzugriff weiter unterteilt werden in Logische Verbindungssteuerung (engl. Logical Link Control, Schicht 2a) und Medienzugriffs-Steuerung (engl. Media Access Control, Schicht 2b).

In der vorliegenden Arbeit geht es darum, ein neues Konzept für die Schicht 2b zu erstellen und dieses auf der CC1010-Hardware zu implementieren. Der Schwerpunkt dabei liegt auf dem Medienzugriff und niedrigem Energieverbrauch. Weiterhin muss Schicht 2a auf die neue Hardware angepasst werden.

2.4 TecO Particle-Protokoll

Grundlage dieser Arbeit ist ein bereits vorliegendes Protokoll für Particles, dass erprobt und im Einsatz ist. Es unterscheidet 3 Schichten: RF-, LL- und ACL-Layer. Abbildung 2.1 auf der nächsten Seite zeigt, wie die Schichten des Particle-Protokolls mit denen des OSI-Modells zusammenhängen. Die Schichten, die im Rahmen dieser Arbeit neu entworfen wurden, sind grau unterlegt.

Die oberste Schicht (ACL[3]) ist zuständig für das Senden und Empfangen von Kontexten und anderen Arten von Information. Die medienunabhängige Sicherungsschicht ermöglicht das Erkennen von Fehlern bei der Übertragung eines Paketes. Die Medienzugriffsschicht implementiert den Medienzugriff mit Zeitmultiplex durch feste Slots und enge Synchronisation zwischen allen

[2] Abk. für International Standards Organization – Open Systems Interconnection
[3] Abk. für engl. Abstract Communication Layer, abstrakte Kommunikationsschicht

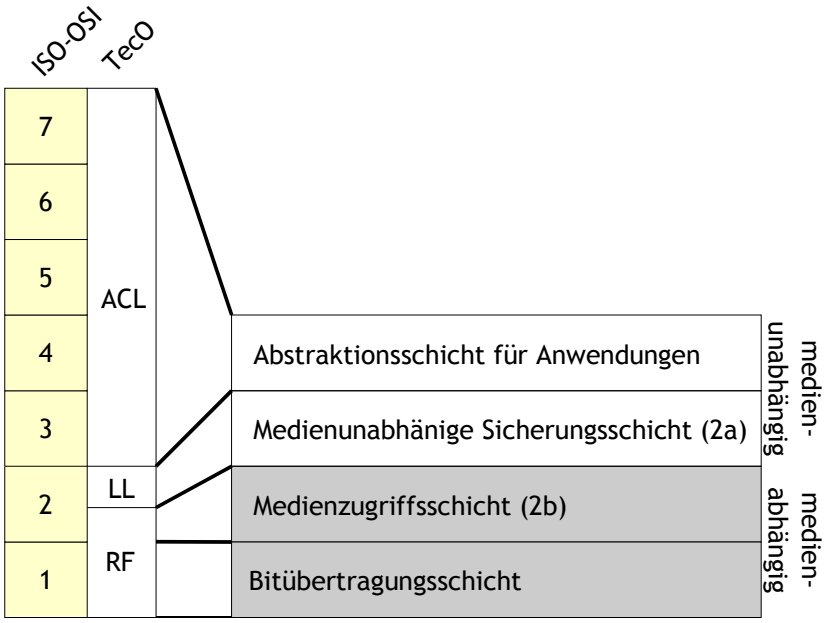

Abbildung 2.1: *Zuordnung der Schichten des TecO Particle-Protokolls zu den Schichten des ISO-OSI-Modells*

Particles in Kommunikationsreichweite. Die Knoten in einen Schlafmodus zu versetzen und bei Datenverkehr aufzuwecken wäre weniger effizient, da die Kommunikation auf einem Frequenzband abläuft, dass von vielen Produkten benutzt wird und so die Knoten vielmals unnötig aufgeweckt werden würden. Außerdem müsste der Empfangsteil des Transceivers ständig aktiviert sein, was viel Energie verbrauchen würde. [RASJP+00]

Das Particle-Protokoll hat seinen Schwerpunkt nicht im Bereich Routing[4] oder Verbreitung von Sensordaten über weite Strecken. Vielmehr soll es Particles ermöglichen, sich schnell mit einem bestehenden Netzwerk zu verbinden und Kontexte auszutauschen. [BeGe]

Die ACL-Schicht übernimmt die Codierung von Kontexten für die Übertragung. ACL-Daten werden im Nutzdatenteil der LL-Schicht übertragen. Diese erfolgt mit ACL-Tupeln, die aus einem 2-Byte Indentifikator für den Kontext und bis zu 61 Byte Daten bestehen.

Um beispielsweise den Kontext „Trend der Temperatur“ zu übertragen, könnte man einen Identifikator für diese Information definieren und als Daten ein Byte mit den Werten 0 (stark fallend), 1 (fallend), 2 (annähernd gleich bleibend), 3 (steigend), 4 (stark steigend) definieren. Man könnte ebenfalls ein oder zwei Datenbytes für den zugrunde liegenden Zeitraum in Minuten definierten. Ist ein solches Paket mit den zur Verfügung gestellten ACL-Funktionen zusammengestellt, kann es mit einem Funktionsaufruf verschickt werden. In einem Paket können aber auch mehrere Kontexte gleichzeitig übermittelt werden. [BKZD+03]

[4] als Routing bezeichnet man ein Verfahren zur Wegewahl in einem nicht komplett vermaschten Netzwerk, d. h. ein Netzwerk in dem es Knoten gibt, die nicht direkt miteinander sondern nur über Zwischenknoten kommunizieren können

Jeder Knoten, der ein Paket empfängt, kann mit einem Funktionsaufruf feststellen, ob es einen bestimmten Kontext enthält. Er kann das Protokoll anweisen, nur Pakete zu empfangen, die einen bestimmten Kontext enthalten und alle anderen Pakete zu ignorieren.

Senden und Empfangen erfolgen asynchron, d. h. die Anwendung beauftragt das Senden mit Hilfe einer Funktion und kann den Status der Sendung (noch nicht verschickt, erfolgreich verschickt, Sendefehler) abfragen. Ebenso kann ermittelt werden, ob ein Paket empfangen wurde.

Die Schicht unterhalb der ACL-Schicht ist die LL-Schicht, die die Adressierung und Fehlerkorrektur übernimmt. Die LL-Schicht fügt vor den ACL-Nutzdaten eines Paketes die 8-Byte-Adresse des Knoten ein. Hinter den Nutzdaten wird eine Prüfsumme eingefügt, mit der die Integrität der versendeten Daten überprüft werden kann.

Die darunterliegende RF-Schicht, die im Rahmen dieser Arbeit neu entworfen wurde, wird detailliert im Kapitel3 auf Seite 15 beschrieben. Ihre Aufgaben sind der Medienzugriff und die Kanalcodierung.

Grundlage für die Particles und ihre Netzwerksoftware sind die im SmartIts-Projekt am TecO gesammelten Erfahrungen. [BeGe]

2.4.1 Verbindung mit Infrastrukturnetzen

Um die von den Knoten des Particle-Netzwerks übertragenen Kontexte über weitere Strecken zu übertragen bzw. um sie herkömmlichen Computern in herkömmlichen Computernetzwerken (LANs, Internet, . . .) zur Verfügung zu stellen, ist eine Kapselung des Particle-Protokolls in UDP-Pakete vorgesehen. Abbildung 2.2 zeigt ein Beispiel für ein Sensornetz, das mit einem Infrastrukturnetz verbunden ist.

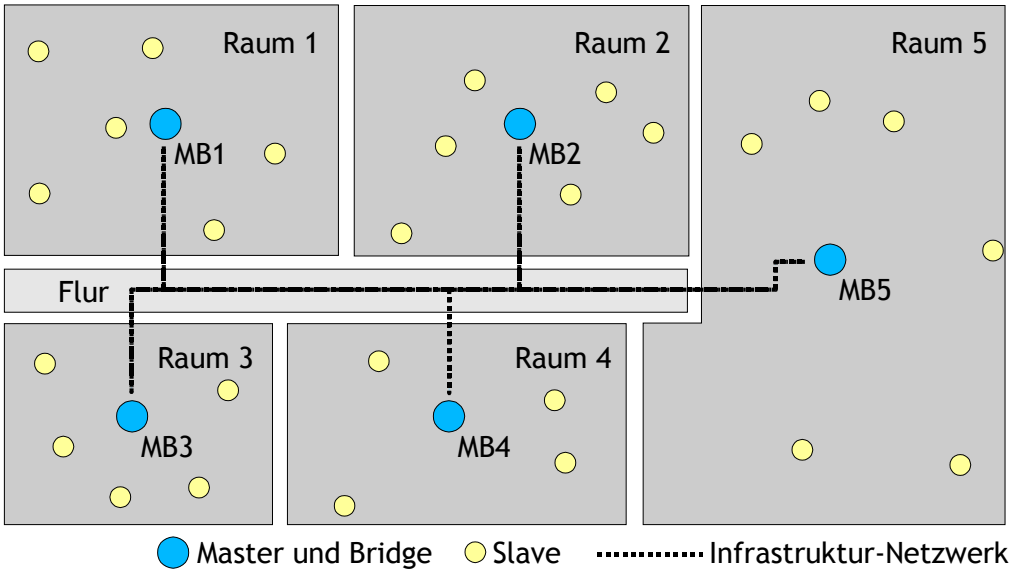

Abbildung 2.2: *Zelluläres Infrastrukturnetz*

Als Gateway zwischen Sensornetz und herkömmlichen Netz dienen so genannte X-Bridges. Diese Übernehmen die Kapselung der Particle-Pakete in UDP-Pakete und die Aussendung von gekapselten Particle-Paketen.

2.5 Andere MAC und RF-Konzepte für drahtlose Sensornetzwerke

Die Domäne der drahtlosen Sensornetzwerke ist ein junges Forschungsgebiet. Insbesondere zum Thema Medienzugriff gibt es noch wenige Veröffentlichungen. Die ersten Sensornetzwerke wurden mit herkömmlicher Hardware und Software realisiert, wie sie schon länger zur drahtlosen Datenübertragung genutzt wird (z. B. IEEE 802.11). Erst in jüngster Vergangenheit setzt eine Spezialisierung und Ausrichtung von Hard- und Software auf die speziellen Anforderungen von drahtlosen Sensornetzwerken ein. Im folgenden werden einige beispielhafte Arbeiten beschrieben.

Die Universität Berkeley in Kalifornien betreibt im PicoRadio-Projekt großen Aufwand, um Knoten zu bauen, die extrem wenig Energie verbrauchen. Das erklärte Projektziel ist, dass der Energiebedarf so gering werden soll, dass die Knoten die Energie selbst gewinnen (z. B. aus Licht, Vibration, Geräuschen, . . .). Die RF- und MAC-Schichten werden dabei in eigenentwickelter, spezialisierter Hardware realisiert um den hohen Anforderungen an die Energieeffizienz zu genügen und trotzdem ein einfaches CS/MA-Verfahren zum Medienzugriff realisieren zu können. Im Vergleich dazu verwendet unser Ansatz standardisierte Hardware mit einer Universal-MCU[5], die bereits in großen Stückzahlen günstig verfügbar ist. Der große Vorteil sind die Kosten: das verwendete Board ist wesentlich billiger als die spezielle Hardware. Dafür braucht die Chipcon-Hardware mehr Energie als die PicoRadio-Knoten. [RASJP+00]

An der University of Southern California wurde S-MAC entwickelt. Bei S-MAC (Abk. für Sensor Medium Access Control) hat jeder Knoten eine individuelle Folge von Wach- und Schlafphasen, die er seinen Nachbarknoten mitteilt. Jeder Knoten merkt sich für jeden seiner Nachbarn diese Folge. Möchte er mit einem Nachbarn kommunizieren so weiß er, wann dieser Empfangsbereit ist und kann ihn in seiner Wachperiode kontaktieren. Dieses Verfahren ist speicherintensiv, weil ein Knoten sämtliche Folge von Wach- und Schlafphasen seiner Nachbarknoten speichern muss. Ebenso werden ein Reservierungsschema ähnlich wie in 802.11 benutzt, was den Speicherbedarf weiter erhöht. Schließlich sorgt ein RTS/CTS-Mechanismus[6] zwar für die Eliminierung des Problems der versteckten Knoten (siehe Abschnitt 3.5.4 auf Seite 30) sorgt aber für zusätzlichen Overhead. Alles in allem ist das Protokoll sehr auf Unicast-Übertragung Glossar) ausgelegt, während die Anforderungen der oberen Schichten des Particle-Protokolls standardmäßig Broadcasts vorsehen. [YeHE02]

[5] engl. für Micro Computer Units, auch Mikrocontroller, Integrierte Schaltung, die Prozessor, Speicher und weitere Peripherie auf einem Chip vereint.

[6] Ready to Send / Clear to Send. Ein Knoten, der ein Paket an einen anderen Knoten senden möchte, sendet zunächst ein „Bereit zum Senden“-Paket an den Zielknoten und erst wenn dieser mit einem „Frei zum Senden“-Paket antwortet, beginnt der Quellknoten mit seiner Übertragung.

2.6 Hardware für drahtlose Sensornetzwerke

Drahtlose Sensornetzwerke sind eine Klasse von Netzwerken, die sich von herkömmlichen Computernetzwerken erheblich unterscheidet. Die Hardware auf der sie aufbauen ist viel einfacher, als bei Computern der PC- oder PDA-Klasse. Dafür braucht sie wenig Energie, bietet aber auch nur niedrigere Datenraten, kleine Speicher und weniger leistungsfähige Prozessoren an.

Die Industrie bietet im Moment zwei gebrauchsfertige Architekturen an, um drahtlose Sensornetzwerke aufzubauen. Dies sind Bluetooth, das auf dem IEEE 802.15.1 Standard aufsetzt, und ZigBee – basierend auf IEEE 802.15.4. Die IEEE-Standards definieren in beiden Fällen die Physische und Medienzugriffs-Schicht (Schicht 1 und 2b), während die Bluetooth und ZigBee-Spezifikationen die höheren Schichten definieren. Von den beiden Standards ist ZigBee der eindeutig am besten auf die Bedürfnisse von Sensornetzwerken zugeschnittene – das Protokoll ist für niedrigen Energieverbrauch optimiert und bietet vergleichsweise niedrige Datenraten an. Bluetooth hingegen wurde von Anfang an als Kabelersatz entworfen und bietet daher höhere Datenraten bei erheblich höherem Energieverbrauch. [IEEE03a] [IEEE03b]

Die Tabelle 2.1 auf der nächsten Seite zeigt einen Vergleich zwischen den genannten Netzwerken, die explizit für kleine Geräte maßgeschneidert sind, die keine Infrastruktur benötigen, und GSM und W-LAN, die für andere Zwecke entwickelt wurden. Zwar lässt sich theoretisch mit jedem drahtlosen Netz ein Sensornetzwerk aufbauen, jedoch machen es die Batterielebensdauer bzw. die Übertragungskosten nicht optimierter Netze äußerst ineffizient und unwirtschaftlich.

Name	ZigBee™	Bluetooth™	WiFi™ alias W-LAN	GPRS/GSM
Hauptandwendungs-gebiet	Überwachung und Steuerung	Kabelersatz	Web, E-Mail, Video, Dateiaustausch	Flächendeckende Sprach- und Datendienste
Benötigter Speicher	4KB-32KB	250KB+	1MB+	16MB+
Batterielebensdauer in Tagen	100-1000+	1-7	0,5-5	1-10
Netzwerkgröße	Quasi unlimitiert (2^{64})	7	32	1
Bandbreite (KB/s)	20-250	720	11000+	64-128+
Reichweite (m)	1-100+	1-100	1-100	1000+
Bewertungskriterien	Zuverlässigkeit, Kosten, Energieverbrauch	Kosten, Komfort	Geschwindigkeit, Flexibilität	Abdeckung, Qualität

Tabelle 2.1: *Vergleich verschiedener drahtloser Netzwerkarchitekturen, Quelle:* http://www.zigbee.org/*, teilweise aktualisiert.*

2.7 CAN-Bus Medienzugriff

Der CAN-Bus ist eine gemeinsame Entwicklung der Firmen Bosch und Intel für die Verkabelung von digitalen Komponenten in Automobilen. Es handelt sich um ein serielles Bussystem.

Die Art, wie der Medienzugriff in CAN realisiert ist, unterstützt eine große Anzahl von Knoten auf einem gemeinsamen Medium und ist im Kern auch für drahtlose Anwendungen realisierbar. Deshalb wurde bei dem Protokoll, dass im Rahmen dieser Arbeit entwickelt wurde, eine Variante des CAN-Medienzugriffs eingesetzt (siehe Abschnitt 3.5 auf Seite 25). An dieser Stelle soll zunächst das ursprüngliche Verfahren dargestellt werden, wie es der CAN-Bus benutzt.

Im CAN-Bus-Protokoll werden Bits in NRZ-L Codierung auf das Medium gegeben. Dies ist die einfache Codierung, bei der einem 1-Bit und einem 0-Bit jeweils andere Symbole[7] zugeordnet werden. Wichtig ist dabei, dass eines der Symbole *dominant* ist und das andere *rezessiv*.

Die Eigenschaften dominant und rezessiv bedeuten folgendes: Wenn kein angeschlossener Knoten ein dominantes Symbol auf den Bus legt, sehen alle Knoten den Bus im rezessiven Zustand. Sobald mindestens ein Knoten ein dominantes Symbol auf den Bus legt, lesen alle den Bus im dominanten Zustand.

Beim CAN-Bus werden die Bits durch die Spannungsdifferenz zwischen zwei Adern definiert. Liegt an den beiden Übertragungsadern eine annähernd gleiche Spannung an, ist dies der rezessive Zustand. Liegt eine hohe Spannungsdifferenz vor, repräsentiert das den dominanten Zustand. Diese Codierung der Bits erhöht die Robustheit gegenüber Rauschen, denn ein äußerer Einfluss, der auf die eine Ader des Kabels einwirkt, tut dies mit hoher Wahrscheinlichkeit auch auf die zweite Ader.

Das Datenpaketformat des CAN-Busses wird CAN-Frame genannt. Ein solcher Frame beginnt mit einem Start-Of-Frame-Bit (ein dominantes Bit) und dem Identifier (11 oder 20 Bit je nach Protokollversion). Ein Identifier ist ein vorher festgelegte Zahl, die den Typ der zu übertragenden Information bezeichnet (z. B. Drehzahl, Status der Zentralverriegelung, . . .).

Möchte nun ein Knoten einen Frame auf den Bus schicken, so schickt er den gerade beschriebenen Anfang des Frames und hört Bit für Bit mit, welchen Zustand der Bus hat. Sobald auf dem Bus ein dominantes Bit zu hören ist, obwohl der Knoten ein rezessives Bit sendet, bricht er die Übertragung ab und empfängt den Rest des Frames. Da keine zwei Knoten Nachrichten mit dem gleichen Identifier verschicken, diese sich also in mindestens einem Bit unterscheiden, ist sicher, dass es zu keiner Kollision auf dem Bus kommt, da derjenige Knoten die Übertragung abbricht, der als erstes den dominanten Zustand ließt, obwohl der den rezessiven sendet.

Implementiert man den Medienzugriff des CAN-Protokolls für andere Medien als Kabel, so kann man die Bits, abhängig vom Medium, z. B. als Licht an/aus

[7] hier: die Darstellung eines Bits als physikalische Größe

(Glasfaser) oder Sender an/aus (Funk) darstellen. Wichtig dabei nur, dass die oben geschilderte Bedingung eingehalten wird: ein Bit-Wert muss eine dominante Darstellung haben, und der andere eine rezessive.

Wegen ihres mehrstufigen Charakters wird diese Art des Medienzugriffs auch Arbitrierung (von engl. Arbiter: Schiedsrichter, Schlichter) genannt.

3. Protokollentwurf

In diesem Kapitel wird der theoretische Entwurf des Protokolls dargestellt. Der Abschnitt 3.3 zeigt die für Schicht 1 verwendeten Paketformate. Im Abschnitt 3.4 wird ein Verfahren dargelegt, das eine präzise Synchronisation der Zeitgeber der Knoten herbeiführt. Dadurch können netzweit synchronisierte Zeitschlitze (engl. Slots) etabliert werden. Der Abschnitt 3.5 zeigt daraufhin, wie ein effizienter Medienzugriff mit Zeitmultiplexing erfolgen kann – der vorgestellte Mechanismus verhindert weitgehend Kollisionen indem er jeweils nur einem Knoten pro Slot das Senderecht zuteilt. Im Abschnitt 3.6 wird ein alternatives Synchronisationsverfahren (zu dem in Abschnitt 3.4) vorgestellt, das einen verteilten, statistischen Ansatz verfolgt. Abschnitt 3.8 zeigt eine Methode, mit der Energie gespart werden kann, wenn in einem Netz nur wenige oder keine Pakete gesendet werden. Im Abschnitt 3.9 wird erklärt, wie der Energieverbrauch weiter gesenkt werden kann, wenn ein Knoten alleine ist. Der Abschnitt 3.10 befasst sich schließlich mit einer Codierung, die die Robustheit der Übertragung über die Luftschnittstelle verbessert. Abschnitt 3.11 erläutert schließlich eine Methode, mit der die jeweilige Anwendung Energie sparen kann, wenn sie nicht die gesamte Verarbeitungszeit benötigt.

Abbildung 2.1 auf Seite 8 stellt die im Rahmen dieser Arbeit neu entworfenen Protokollteile in grauer Farbe dar.

3.1 Motivation

Die zu entwerfenden Schichten des Protokolls stellen die Basis für jegliche Kommunikation zwischen den Knoten des Particle-Netzwerks dar. Alle darüber liegenden Schichten sind auf die Dienste dieser Schichten angewiesen, und fügen weiteren, meist semantischen Mehrwert hinzu. Fehler, die in den untersten Schichten begangen werden, und Ineffizienzen, die hier in Kauf genommen werden, können durch die weiter oben liegenden Schichten nicht mehr ausgeglichen werden. Deshalb ist eine sorgfältige und ausführliche Herangehensweise nötig.

Das Protokoll muss sich an folgenden Qualitätskriterien messen lassen:

- Geringer Energieverbrauch
  - dabei sollten Betriebszustände berücksichtigt werden, in denen besonders effektiv Energie gespart werden kann wie z. B. die Isoliertheit eines Knotens, das Ausbleiben von Nutzdatenverkehr, etc.
- Effektive Mediennutzung unter Vermeidung von Kollisionen auch bei vielen Knoten auf engem Raum (Skalierbarkeit)
- Minimales Aufkommen von Signalisierungsdaten
- Automatische Konfiguration
- Berücksichtigung von versteckten und ausgelieferten Knoten
- Schnelles Auffinden und Verbinden mit anderen Knoten nach der Initialisierung (engl. Responsiveness)
- Optimale Kanalcodierung um eine möglichst robuste Übertragung über die Funkschnittstelle zu ermöglichen
- Bereitstellen einer angemessenen Nutzdatenrate
- Beschränkung der Übertragungslatenz

Zwischen manchen dieser Ziele ergeben sich Konflikte: Je kürzer z. B. die Zeit sein soll, in der sich ein Knoten nach der Initialisierung mit anderen Knoten verbindet, desto weniger Spielraum bleibt zum Energie sparen. Ebenso soll die Zeit, in der eine Übertragung stattfinden kann, möglichst klein sein, damit die Transceiver die meiste Zeit ausgeschaltet bleiben können – dadurch wird aber zwangsläufig Bandbreite verschenkt, weil das Medium in der übrigen Zeit ungenutzt bleibt. In diesen und ähnlichen Fällen müssen günstige Kompromisse gewählt werden.

3.2 Überblick über das Protokoll

Um einen möglichst geringen Energieverbrauch zu erzielen, ist es notwendig, dass die Transceiver der Knoten die meiste Zeit abgeschaltet sind und nur zu bestimmten Zeitpunkten aktiviert werden. Das bedeutet, dass Knoten die meiste Zeit keine Daten versenden oder empfangen können. Insbesondere bedeutet es, dass die Knoten nicht permanent auf eine eingehende Nachricht warten können wie dies z. B. beim W-LAN-Standard der Fall ist. Ein Transceiver in Empfangsbereitschaft, der das Mediums aktiv abhört und nach einer hereinkommenden Übertragung sucht, verbraucht nämlich wesentlich mehr Energie als ein deaktivierter Transceiver. [RASJP+00]

Eine andere Möglichkeit wäre es, den Transceiver (z. B. durch ein externes Bauteil) dann zu aktivieren, wenn auf der benutzen Übertragungsfrequenz ein Signal zu hören ist. Da sich aber meist viele inkompatible drahtlose Geräte ein Frequenzband teilen, würde auch dies zu vielen unnötigen Aktivierungen des Transceivers und damit Energieverschwendung führen.

Damit die Knoten also keine Übertragung übersehen (obwohl die Transceiver die meiste Zeit deaktiviert sind), müssen sie ihre Transceiver zur gleichen Zeit aktivieren. Zu diesem Zweck definieren wir isochrone Slots von 30 ms

Länge[1], die mittels einer im Folgenden erläuterten Synchronisationsfunktion auf allen Knoten jeweils zur gleichen Zeit beginnen. Relativ zu diesem Zeitraster können nun alle Aktivitäten, die auf allen Knoten synchron durchgeführt werden müssen (also insbesondere die Funkkommunikation) in ihrem Anfangszeitpunkt festgelegt werden.

In jedem Slot wird genau ein oder kein Paket versendet. In den Slots, in denen ein konkurrierender Zugriff auf das Medium erfolgt, wird zu Beginn des Slots und vor dem Senden des Paketes die so genannte Arbitrierung durchgeführt. Diese stellt sicher, dass nur ein Knoten in dem Slot senden darf. In Slots in denen ein Knoten exklusiven Zugriff auf das Medium hat (siehe Abschnitt 3.4 auf der nächsten Seite) erfolgt die Übertragung des Paketes gleich am Anfang des Slots.

Die Herstellung und Aufrechterhaltung synchroner Zeitgeber in allen Knoten, wird durch regelmäßigen Austausch von Synchronisationspaketen (Beacons[2]) aufrecht erhalten. Der Austausch von Nutzdaten erfolgt mit Datenpaketen variabler Länge.

3.3 Paketformat

Für die Übertragung von Daten- und Kontrollnachrichten der Schicht 1 wird ein einfaches Paketformat verwendet, wie es in Abbildung 3.1 dargestellt ist.

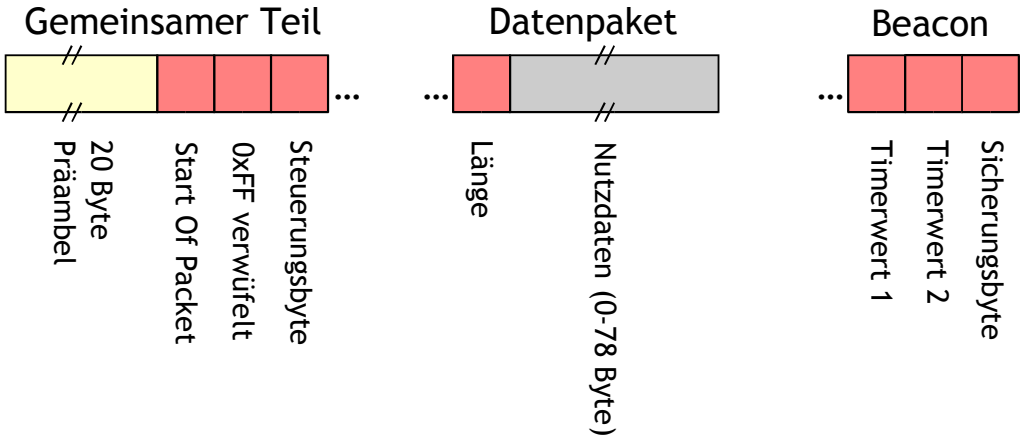

Abbildung 3.1: *Format der Pakete der Schicht 1*

In der RF-Schicht gibt es zwei Arten von Paketen[3]: Beacons und Datenpakete. Die Anfänge beider Paketarten sind gleich:

[1]Eine Länge von 30 ms wurde gewählt, weil sich dadurch in der erstellten Referenzimplementierung ein günstiges Verhältnis ergibt zwischen der Zeit, in der die Transceiver aktiviert sind und der restlichen Zeit

[2]engl. für Funkfeuer, Leuchtfeuer, Bake. Im Folgenden wird der englische Begriff verwendet. Er ist verbreitet für Pakete, die die zeitliche Synchronisation ermöglichen. Insbesondere der W-LAN-Standard verwendet ihn (IEEE 802.11).

[3]Dies sind die grundlegenden Arten von Paketen des Protokolls. In Abschnitt 3.5.4 auf Seite 30 werden noch 4 weitere, optionale Paketarten eingeführt, die dort auch beschrieben sind.

Die Präambel besteht aus 20 Bytes mit dem Wert $AA_{hex} = 10101010_{bin}$. Diese Folge wird vom Receiver benötigt (siehe Abschnitt 4.3.3 auf Seite 52) und ermöglicht die Bitsynchronisation des Empfängers. Eine Länge von 20 Bytes hat sich in der Referenzimplementierung bewährt um eine robuste Übertragung zu gewährleisten.

Das Start-Of-Packet-Byte hat den Wert $5A_{hex}$ und wird für die Synchronisation der Bytegrenzen benutzt. Danach wird ein verwürfeltes Byte mit dem Wert FF_{hex} gesendet (siehe Abschnitt 3.10 auf Seite 40, Scrambling).

Das Steuerungsbyte (engl. Frame Control Byte) unterscheidet die beiden Arten von Schicht-1-Paketen: Die Beacons (Bytewert 1), die der zeitlichen Synchronisation der Knoten dienen, und die Datenpakete (Bytewert 0), die Pakete höherer Schichten transportieren (die für die übergeordnete Schicht wiederum Daten- oder Kontrollpakete darstellen können). Alle anderen Werte dieses Bytes sind für spätere Erweiterungen reserviert. Alle Pakete mit einem unbekannten Wert in diesem Byte werden stillschweigend verworfen.

Ab diesem Punkt unterscheiden sich die beiden Paketarten:

Im Datenpaket folgt das Längenbyte. Es gibt an, wie viele Bytes als Nutzdaten empfangen werden müssen. Danach folgen direkt die Nutzdaten (1-78 Byte).

Im Beacon folgen zwei Bytes die den Wert des Zeitgebers (engl. Timer)[4] im Sender kurz vor dem Senden des Paketes enthalten. Mit dieser Information kann der empfangende Knoten seine Uhr stellen. Es folgt ein Sicherungsbyte, das aus der bitweisen XOR-Verknüpfung der beiden Timerbytes besteht und das zur Überprüfung der Integrität der Übertragung genutzt wird. [TecO]

3.4 Synchronisation mit Master und Slave

Um die Synchronisation der Knoten zu gewährleisten wurde ein Master-Slave-Modell entwickelt. In jedem Sensornetzwerk befindet sich genau ein Master. Die Master- bzw. Slave-Rolle ist einem Knoten nicht fest zugewiesen. Nach dem Einschalten sucht ein Knoten erst eine längere Zeit nach einem Master, d. h. er versucht Beacons zu empfangen. Empfängt er ein Beacon, geht er in den Slave-Modus und erwartet in regelmäßigen Abständen weitere Beacons. Ansonsten geht er davon aus, dass kein anderer Master vorhanden ist und geht in den Master-Modus.

Abbildung 3.2 auf der nächsten Seite zeigt die möglichen Zustände des Protokolls. Dabei sind die im normalen Betrieb zu beobachtenden Übergänge in hellgrün[5] gekennzeichnet. Diese sind

[4] Zeitgeber oder Timer sind Hardwarebausteine. Sie können programmiert werden, periodisch ein Ereignis auszulösen, auf das der Prozessor, bzw. die auf ihm laufende Software reagieren kann. Timer kann man sich als rückwärts laufende Uhren vorstellen, die sich – wenn sie den Zeitpunkt 0 erreicht haben – immer wieder auf einen einmal festgelegten Zeitwert zurückstellen. Meist ist der aktuelle Zeitwert eines Timers (der Timerwert) von der Software les- und schreibbar.

[5] in der Papierversion die Pfeile in mittelgrau

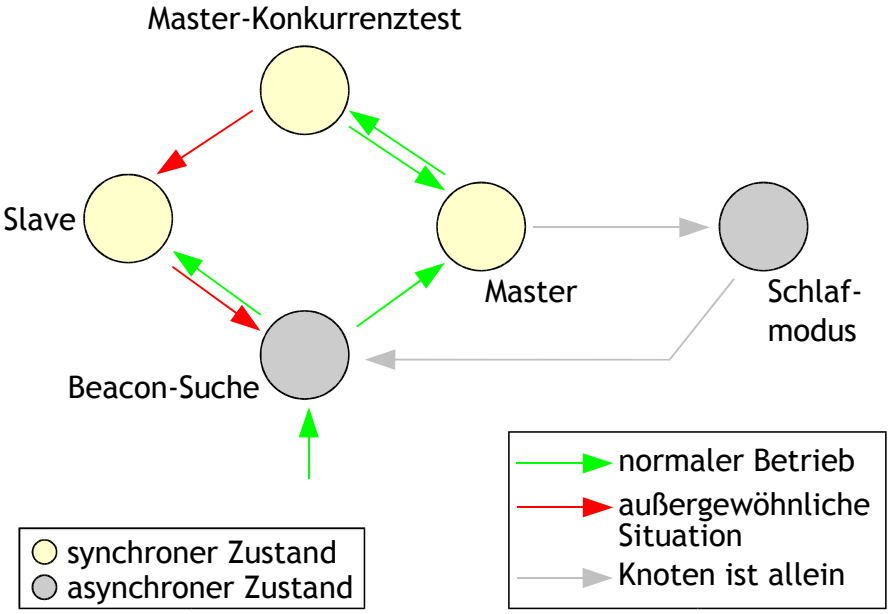

Abbildung 3.2: *Zustandsautomat der Master-Slave-Synchronisation*

- der Eintritt in die intensive Beaconsuche nach dem Einschalten und Initialisieren des Knotens
- der Übergang in den Slavezustand, in dem Fall, dass ein Beacon gefunden wurde
- der Übergang in den Masterzustand, falls kein Beacon empfangen werden konnte
- und der zyklische Übergang zwischen Masterstatus und Master-Konkurrenztest (siehe Abschnitt 3.4.1 auf Seite 21)

Ungewöhnliche Übergänge sind in hellrot[6] gekennzeichnet:

- Der Übergang Master-Konkurrenztest in den Slavezustand, weil ein Beacon empfangen wurde und damit angenommen wird, dass sich ein anderer Master im Empfangsbereich befindet. Die beiden Master können sich entweder räumlich aufeinander zu bewegt haben, oder die Bedingungen des Funkkanals haben sich verbessert (z. B. weil ein Störsignal wegfiel oder schwächer wurde).
- Der Übergang vom Slavezustand in die intensive Beaconsuche, da der Master, mit dessen Beacons der Slave sich synchronisiert hat, nicht mehr sendet, außer Empfangsreichweite gekommen ist oder die Qualität des Kanals sich verschlechtert hat.

Die hellgrauen Pfeile zeigen die Zustandsübergänge, die ein Knoten ausführt, wenn er alleine ist und zyklisch in einen Schlafmodus geht (siehe Abschnitt 3.9 auf Seite 36).

Aus zeitlicher Perspektive wird jeder n-te Slot für die Übertragung eines Beacons reserviert. In diesem Slot darf keine Datenkommunikation stattfinden

[6] in der Papierversion die Pfeile in dunkelgrau

– der Master eines Netzes erhält exklusiven Zugriff auf das Medium, eine Arbitrierung wie in Datenslots muss nicht stattfinden. Dies gewährleistet, dass die Synchronisation auch bei komplett ausgelastetem oder überlastetem Medium noch funktioniert. Ein Slot, in dem die Übertragung eines Beacons stattfindet, wird Beaconslot genannt.

n muss so groß wie möglich gewählt werden, da die Übertragung des Beacons Energie verbraucht und mit jedem Beaconslot eine Möglichkeit zur Übertragung eine Nutzdatenpaketes wegfällt. Gleichzeitig darf n aber nicht zu groß gewählt werden, damit die Laufzeitunterschiede der einzelnen lokalen Zeitgeber nicht zu groß werden. Ein Knoten im Mastermodus muss periodisch alle n Slots, also in jedem Beaconslot, seinen aktuellen Zeitgeberwert senden. Alle Slave-Knoten im Empfangsbereich dürfen nichts senden und müssen auf das Beacon warten. Wird es fehlerfrei empfangen, was mit Hilfe des Sicherungsbytes überprüft wird, stellen sie ihre Zeitgeber mit Hilfe der im Beacon enthalten Informationen so ein, dass sie synchron zum Zeitgeber des Masters laufen (siehe Abbildung 3.3).

Dabei ist zu beachten, dass der empfangende Knoten den im Beacon enthaltenen Zeitstempel nicht unverändert auf seinen Timer übertragen darf. Da der gesendete Zeitwert beim Master vor dem Senden ausgelesen wurde und zum Zeitpunkt des vollständigen Empfangs schon wieder Zeit vergangen ist, würde die Uhr des Empfängers nach gehen, wenn er den Zeitstempel unverändert zuweisen würde – der Empfänger würde zu spät aufwachen und den Beginn des nächsten Slots verfehlen. Statt dessen muss auf den empfangenen Zeitstempel die Zeit Δt addiert werden. Aus dem Diagramm in Abbildung 3.3 wird ersichtlich, dass diese Zeit aus folgenden Komponenten besteht:

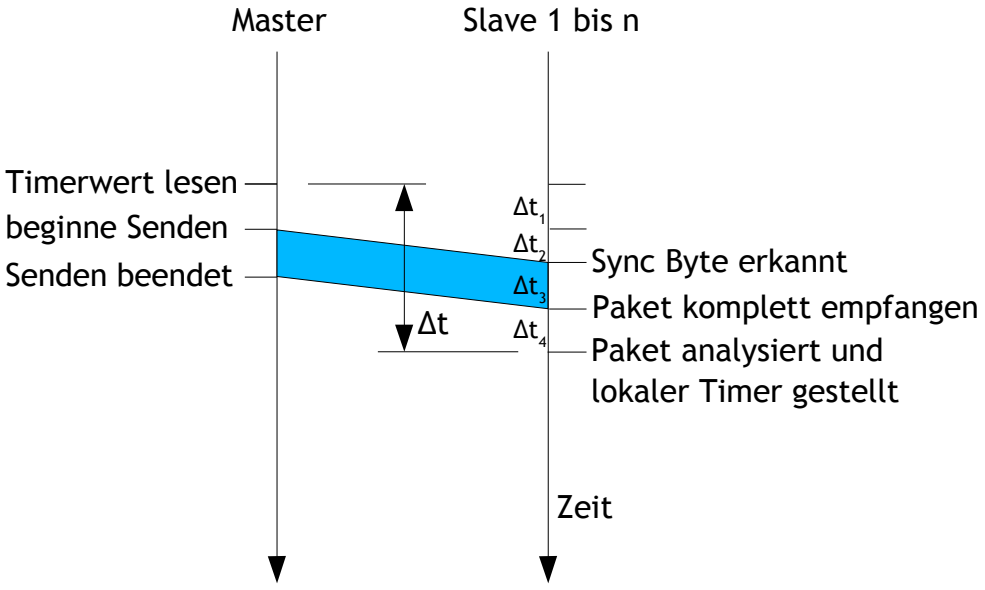

Abbildung 3.3: *Zeitlicher Ablauf der Synchronisation*

Vorbereitungszeit Δt_1 Auslesen des Timerwertes, Hinzufügen von Redundanz zur Fehlererkennung, Senden der Präambel.

Übertragungslatenz Δt_2 Die Zeit vom Senden des ersten Bytes (genauer: das erste Byte nach dem Sync-Byte) bis zum seinem Empfang.

Übertragungsdauer Δt_3 Die Zeit, die für die Übertragung benötigt wird.

Nachbereitungszeit Δt_4 Überprüfung der Integrität der Nachricht, Zuweisung des Timerwertes.

Die Zeit $\Delta t = \sum_{i=1}^{4} \Delta t_i$ wird (von kleinen Schwankungen abgesehen, die die Funktionalität nicht beeinträchtigen) als annähernd konstant angenommen.

In der erstellten Referenzimplementierung addiert der sendende Knoten den durch Messung ermittelten Wert von Δt noch vor dem Senden zu dem ausgelesenen Timerwert. Der empfangende Knoten kann somit den empfangenen Wert unverändert in sein Timerregister schreiben.

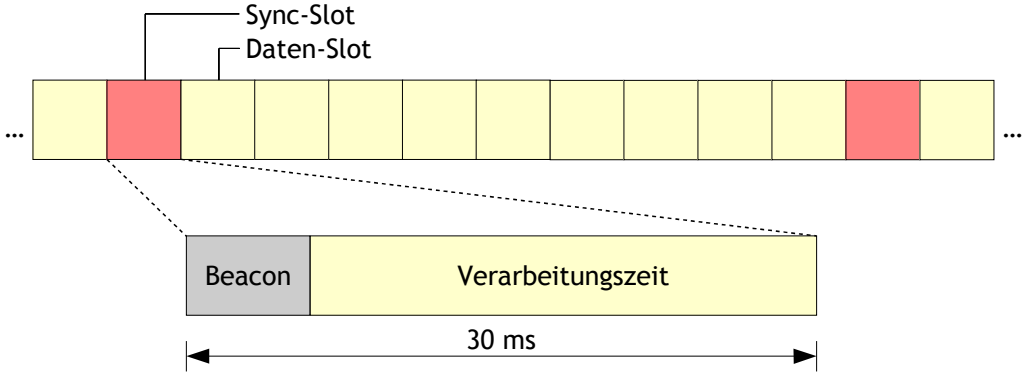

Abbildung 3.4: *zeitliche Betrachtung des Beaconslots*

Abbildung 3.4 zeigt den zeitlichen Ablauf eines Beaconslots. Der Master schickt das Beacon am Anfang des Slots, die Slaves erwarten das Paket dort, empfangen und verarbeiten es.

3.4.1 Vermeiden konkurrierender Master

Wenn sich in einem Netz mehr als ein Master befinden, führt dies zu Problemen. So kann es vorkommen, dass sich verschiedene Teilnetze auf unterschiedliche Master synchronisieren. Diese können dann nur innerhalb des jeweiligen Teilnetzes, nicht aber mit den anderen Teilnetzen kommunizieren – die Übertragungen der jeweils anderen Teilnetze stellen ein Störsignal dar. Im schlimmsten Fall stören sich die Master so stark, dass keines der ausgesendeten Beacons fehlerfrei empfangen werden kann. Daher muss diese Situation möglichst vermieden werden. Weitere Probleme im Zusammenhang mit mehreren Mastern innerhalb eines Netzes zeigt Abschnitt 3.4.2.

Abbildung 3.2 auf Seite 19 fasst die möglichen Zustände der Master-Slave-Synchronisation und die Übergänge zwischen ihnen zusammen[7]. Um zu verhindern, dass ein Netz mehrere Master enthält, führt der einzige Weg zum Master-Zustand über eine intensive Beaconsuche. Bei der intensiven Beaconsuche wird kontinuierlich, unter Vernachlässigung der Slotgrenzen, für ein

[7] der Schlafmodus wird in Abschnitt 3.9 beschrieben

Vielfaches der Slotzeit nach einem Master gesucht[8]. Erst wenn während dieser intensiven Suche kein Beacon empfangen wurde, erhält der Knoten den Masterstatus.

Als zweite Maßnahme zur Vermeidung mehrerer Master in einem Netz dient der so genannte *Master-Konkurrenztest*. Jeder Master führt diesen alle n Beaconslots durch. Sein Zweck es ist, andere Master zu detektieren (z. B. weil sich zwei Master in den gegenseitigen Übertragungsbereich bewegt haben). Der Master-Konkurrenztest beginnt unmittelbar nach Senden des Beacons und dauert bis zum Beginn des nächsten Beaconslots (siehe Abbildung 3.5). Während dieser Zeit ist der Master ständig in Empfangsbereitschaft. Empfängt er während des Tests ein Beacon, so stellt er seinen Timer danach und wird zum Slave.

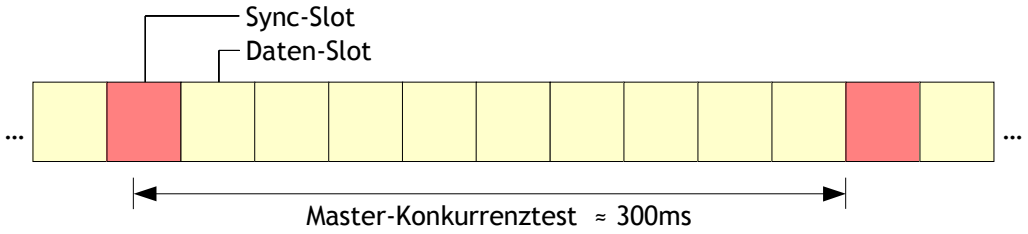

Abbildung 3.5: *Master-Konkurrenztest zur Konfiktauflösung*

Der Master-Konkurrenztest sollte nur selten durchgeführt werden, da die Funktionalität des Masters während der n Slots stark eingeschränkt ist. Zwar empfängt der Knoten in dieser Zeit auch Datenpakete, durch die ständige Empfangsbereitschaft bleibt aber keine Zeit für die auf dem Knoten laufende Anwendung um die Pakete zu verarbeiten. Wenn Datenpakete ankommen, so steht der Anwendung nach Abschluss des Tests nur das zuletzt empfangene zur Verfügung und sie kann es erst nach Senden des Beacons im nächsten Beaconslot bearbeiten.

3.4.2 Einschränkungen des Master-Slave Konzeptes

Für die Synchronisation im Master-Slave-Modus wird angenommen, dass sich alle Knoten jeweils im gegenseitigen Empfangsbereich befinden, d. h. jeder Knoten kann die Pakete eines jeden anderen Knoten empfangen.

Ein Netzwerk wie es in Abbildung 2.2 auf Seite 9 dargestellt ist erfüllt diese Kriterien größtenteils. In der Regel ist sichergestellt, dass sich alle Knoten in einem Raum gegenseitig hören können. Durch geschickte Wahl der Signalstärke kann theoretisch verhindert werden, dass Knoten in verschiedenen Räumen nicht über die Luftschnittstelle kommunizieren, und die raumübergreifende Kommunikation ausschließlich über ein drahtgebundenes Netzwerk erfolgt. In jedem Raum befinden sich Gateways, die Pakete der LL-Schicht in UDP-Pakete kapseln und an das Ethernet weiterleiten, und gekapselte LL-Pakete aussenden. RF-Protokollpakete (wie das Beacon) werden jedoch nicht

[8] In der Referenzimplementierung dauert die Suche nach dem Einschalten 200 mal so lange wie die Länge eines Slots plus einer 8-Bit Zufallszahl. Nach dem Aufwachen aus dem Schlafmodus dauert sie 50 mal so lange wie die Länge eines Slots.

weitergeleitet, so dass sich jeweils nur alle Knoten in einem Raum synchronisieren. Knoten können sich natürlich auch über mehrere Räume synchronisieren – aber nicht durch von den Gateways weitergeleitete Pakete.

Mit dieser Technik lässt sich das Problem eines durch unterschiedlich Synchronisation (mehrere Master) gespaltenen Netzwerks eindämmen. Da sich elektromagnetische Wellen, wie sie für die Funkkommunikation eingesetzt werden, jedoch auch durch Wände fortpflanzen, kann die Zellenplanung nicht immer so perfekt sein, dass ein Fall wie in Abbildung 3.6 ausgeschlossen ist.

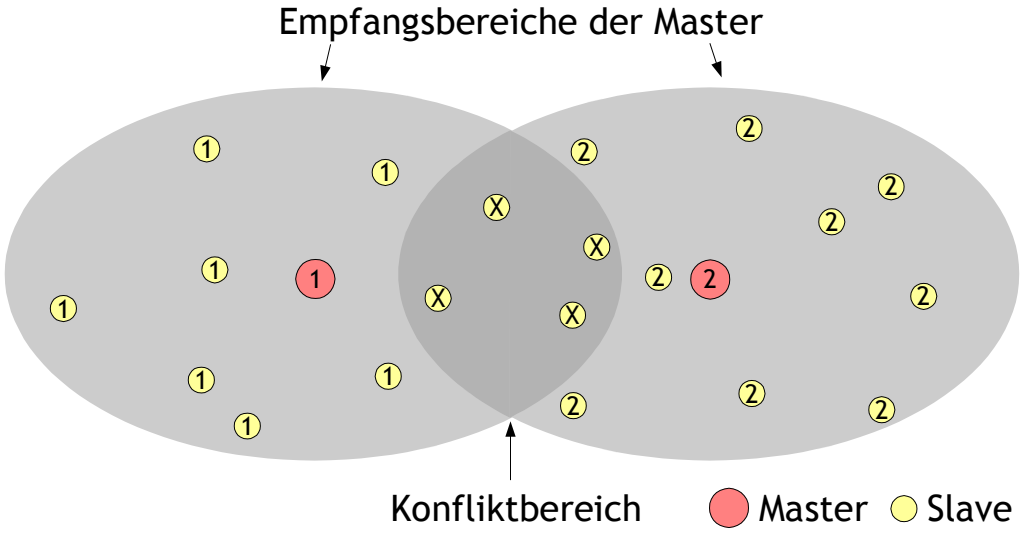

Abbildung 3.6: *Teilnetze und Konfliktbereiche durch mehrere Master*

Wenn zwei Master-Knoten so weit voneinander entfernt sind, dass sie sich nicht mehr hören können, kann es einen Bereich geben, in dem die Signale beider hörbar sind (vgl. Abbildung 3.6). Befinden sich in diesem Bereich Knoten, so ist deren Zustand nicht eindeutig.

Abbildung 3.7 auf der nächsten Seite zeigt in zwei Beispielen die zeitliche Darstellung von Signalen, die bei einem Knoten im Konfliktbereich ankommen. Durch die zwei Master gibt es zwei Teilnetze. Die Knoten von Netz 1 sind mit Master 1 synchronisiert, die von Netz 2 mit Master 2. Die Slotgrenzen sind verschoben. Die Knoten in der Mitte sind entweder auf einen der beiden Master synchronisiert oder hören keine Beacons mehr, weil sich die Beacons der beiden Master jeweils überlagern.

Eine Lösung dieses Problems bringt die später in Abschnitt 3.6 auf Seite 31 eingeführte, alternative Synchronisationsmethode.

3.4.3 Dynamische Weitergabe der Masterrolle

Ein Knoten im Mastermodus ist gegenüber einem Slaveknoten im Nachteil, da er durch das regelmäßige Senden der Beacons mehr Energie benötigt. Daher wäre es günstig, wenn ein Knoten, dessen Energie begrenzt ist, die Masterrolle nach einer gewissen Zeit gezielt an einen anderen Knoten abgeben könnte. Dieser Abschnitt zeigt ein mögliches Verfahren dafür.

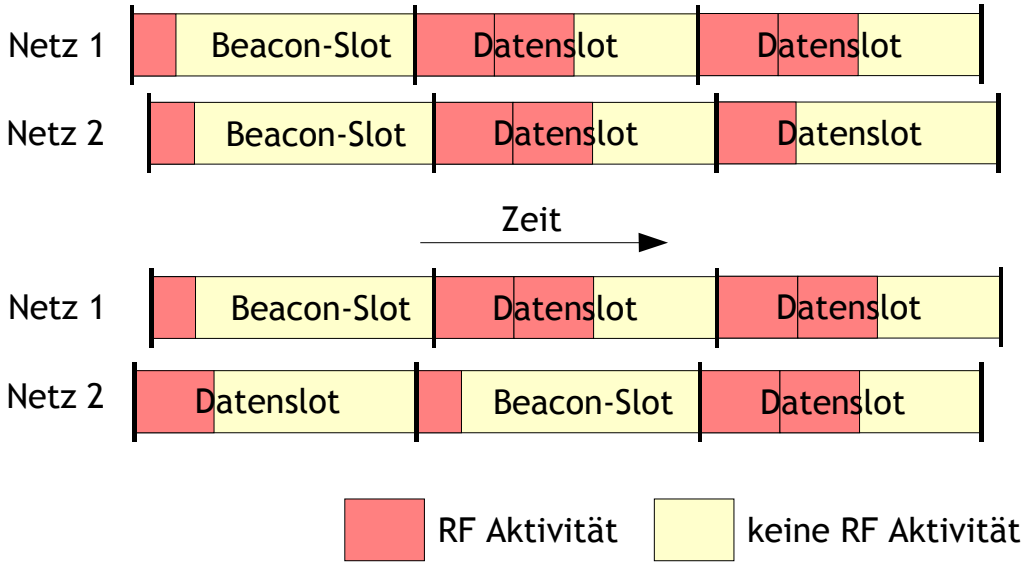

Abbildung 3.7: *zeitliche Betrachtung der Konfliktsituation wenn zwei Master im Empfangsbereich*

Die Weitergabe der Masterrolle ist aber nicht für jede Art von Knoten sinnvoll. Solche, die über eine dauerhafte Energieversorgung verfügen, z. B. Gateways, die an ein Netzteil angeschlossen sind, sollten dies nicht tun. Man könnte definieren, dass diese Knoten die Masterrolle bevorzugen. Alle andern Knoten, bei denen die Energieversorgung begrenzt ist, bevorzugen die Slaverolle. Ob ein Knoten die Master- oder Slaverolle bevorzugt wird seine *Präferenz* genannt.

Die geregelte Übergabe sollte gesichert und ohne lange Übergangszeit funktionieren, in welcher das Netz keinen oder mehrere Master hat. Dies stellt der folgende 4-Wege-Handshake sicher:

- Wenn ein Master seine Rolle an einen anderen Knoten weitergeben möchte, schickt er eine GiveUp-Nachricht als Broadcast[9].
- Jeder Knoten der die Nachricht gehört hat, sendet eine Solicitation(ID, Pref)-Nachricht, die seine ID[10] und seine Präferenz enthält.
- Der Master wartet einige Slots auf Solicitation(ID, Pref)-Nachrichten. Er merkt sich die ID der ersten Nachricht mit Master-Präferenz. Falls nur Nachrichten mit Slave-Präferenz ankommen, merkt er sich von diesen die ID der ersten Nachricht. Nach der Wartezeit schickt der Master eine Grant(ID)-Nachricht an die ID, die er sich gemerkt hat. Falls keine Solicitation-Nachrichten ankommen, bricht er ab.
- Ein Knoten, der eine Grant(ID)-Nachricht empfängt, dessen ID gleich der in der Nachricht ist, ändert seinen Status in Master und sendet eine Accept(ID)-Nachricht. Optional kann die Nachricht auch mehrmals

[9]Broadcast, engl. für Rundruf, ein Signal bzw. Datenpaket, das nicht an einen bestimmten Knoten adressiert, sondern für alle empfangsbereiten Knoten bestimmt ist.

[10]ID steht für engl.: Identifier, also Identifikator. Ein Knoten hat im Particle-Stack in der RF-Schicht eigentlich keine ID, da sämtliche Kommunikation über Broadcasts erfolgt. Hier ist die ID der LL-Schicht gemeint.

gesendet werden. Das Senden mehrerer Accept-Nachrichten erhöht die Zuverlässigkeit der Übergabe, denn hier ist die verwundbare Stelle des Protokolls: Nach Senden der Nachricht ist das Netz in einem inkonsistenten Zustand (2 Master) bis die Accept-Nachricht vom ursprünglichen Master korrekt empfangen und verarbeitet wurde.

- Ein Master, der eine Accept(ID)-Nachricht empfängt, deren enthaltene ID übereinstimmt mit der in der zuletzt gesendeten Grant(ID)-Nachricht, wird zum Slave.

Ein Masterknoten mit Slave-Präferenz (diese ist der Regelfall), zieht nach dem Senden jedes Beacons eine Zufallszahl und gibt mit einer gewissen (sehr niedrigen) Wahrscheinlichkeit seine Rolle auf, indem er eine GiveUp-Nachricht verschickt und dem obigen Protokoll folgt.

Ein Masterknoten, der den Masterzustand bevorzugt, gibt ihn von sich aus nicht auf, d. h. er verschickt keine GiveUp- oder Grant-Nachrichten. Einzig wenn ein bevorzugter Master einen anderen Master hört, wird er zu dessen Slave um die Stabilität des Netzwerks nicht zu gefährden.

Slave-Knoten mit Master-Präferenz könnten auch aktiv versuchen, den Master-Status zu erlangen, indem sie von Zeit zu Zeit Solicitation-Nachrichten verschicken und so beim 2. Punkt des Protokolls einsteigen.

Das Paketformat der einzelnen Pakete wird in Tabelle 3.1 definiert.

Nachricht	Steuerungsbyte	Daten
GiveUp	2	keine
Solicitation(ID, Pref)	3	8 Byte ID des Absenders, 1 Byte Präferenz
Grant(ID)	4	8 Byte ID des Empfängers
Accept(ID)	5	8 Byte ID des Absenders

Tabelle 3.1: *Paketformat für GiveUp-Sequenz*

3.5 Medienzugriff

In den Datenslots, das sind alle Slots zwischen den Beaconslots, darf jeder Knoten Pakete senden. Zu einem festgelegten Zeitpunkt innerhalb des Slots kann der Knoten entweder senden oder empfangen. Um zu verhindern, dass mehrere Knoten gleichzeitig versuchen zu senden und somit Kollisionen (durch Überlagerung unverständliche Signale) auf dem Medium entstehen, muss ein Medienzugriffsverfahren eingesetzt werden, dass nur jeweils einem Knoten den Zugriff gewährt, ihn allen anderen Knoten verweigert und dabei jeden Knoten möglichst gleichberechtigt behandelt.

3.5.1 Verfahren

Im vorliegenden RF-Protokoll wird eine abgewandelte Version des Medienzugriffs von CAN angewandt, die im Grundlagenkapitel im Abschnitt 2.7 auf Seite 13 beschrieben wurde.

Zunächst wollen wir das Verfahren einordnen: Es braucht keine herausgehobene Station, die den Zugriff regelt sondern organisiert ihn verteilt. Es wird kein gesonderter Signalisierungskanal verwendet, sondern der gleiche benutzt, der auch zur Übertragung der Daten verwendet wird (In-Line Signalisierung). Es findet keine Reservierung und Medienzuteilung statt.

Das Verfahren ist optimiert auf Fairness beim Zugriff in einem voll vermaschten Netz ohne versteckte und ausgelieferte Knoten. In diesem Fall liefert es auch bei überlastetem Medium gute Fairness und hilft beim Energie sparen.

Die Beobachtung der Bits auf dem Medium in Echtzeit wie bei CAN (siehe Abschnitt 2.7 auf Seite 13) ist mit dem verwendeten Chip nicht möglich, da er in der Praxis ca. 500 μs benötigt um den Transceiver aus dem ausgeschalteten Zustand in Betriebsbereitschaft zu versetzen (siehe Abschnitt 4.3.3 auf Seite 52). Ähnlich lange dauert das Umschalten zwischen Senden und Empfangen.

Es werden auch keine Identifier für die Arbitrierung verwendet. Statt dessen zieht ein Knoten eine 8-Bit Zufallszahl (Arbitrierungsbyte) am Anfang jedes Slots in dem er ein Paket senden möchte. Der Zugriff auf das Medium wird also nicht mir unterschiedlichen Prioritäten gewährt, sondern mit gleicher Chance für alle Knoten.

Statt einer Busleitung wird die Luftschnittstelle benutzt, die es genauso ermöglicht Broadcastsignale[11] zu senden. Für die 1 wird als dominantes Signal ein Bitmuster ausgesendet. Für jede 0 im Arbitrierungsbyte misst der Knoten die ankommende Signalstärke anhand der er entscheidet, ob ein Signal auf dem Medium liegt oder nicht. Die einzelnen Zeitabschnitte, in denen ein Knoten entweder sendet oder hört nennen wir die Arbitrierungs-Zeitscheiben (engl. Arbitration-Slices).

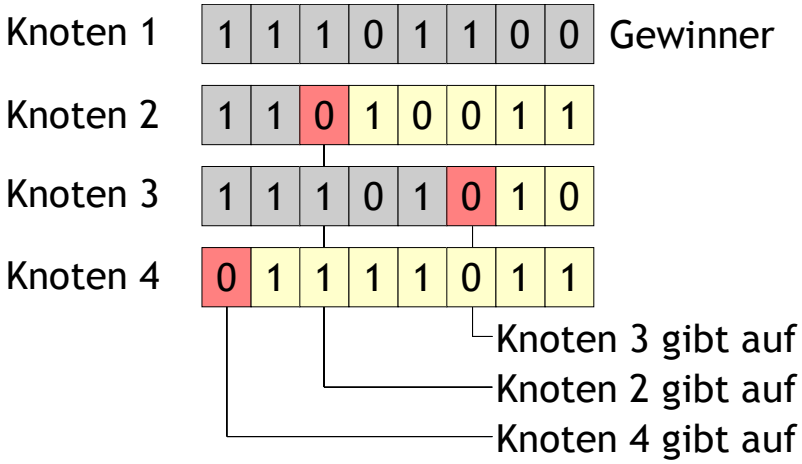

Abbildung 3.8: *Beispiel für Arbitrierung mit 4 Knoten*

Abbildung 3.8 zeigt ein Beispiel für eine Arbitrierung. 4 Knoten bewerben sich gleichzeitig um das Medium. Jeder der vier Knoten würfelt eine Zufallszahl. Nach und nach scheiden alle Knoten bis auf Knoten 1 aus, weil sie, während

[11] Ein **Broadcastsignal** (von Broadcast, engl. Rundruf), ein Signal das von einem Sender gesendet wird, und das alle Empfänger hören können.

Sie sich in einem 0-Slice befinden (also während sie ins Medium hören), ein Signal eines anderen Knoten erkennen, der sich momentan in einem 1-Slice befindet (also gerade ein Signal ins Medium sendet).

Zu einer Kollision kann es nur kommen, falls zwei Knoten eine exakt gleiche Zufallszahl gezogen haben ($P = 1/2^8$) oder einer der Knoten ein oder mehrere Signale auf dem Medium nicht detektiert (also 1-Slices als 0-Slices erkennt während er sich selbst in einem 0-Slice befindet).

Abbildung 3.9 zeigt den zeitlichen Aufbau eines Datenslots, dessen Aufteilung in Arbitrierungs-Teil am Anfang und Paket-Teil in der Mitte, sowie die Vorgänge innerhalb eines einzelnen Arbitrierungs-Slices.

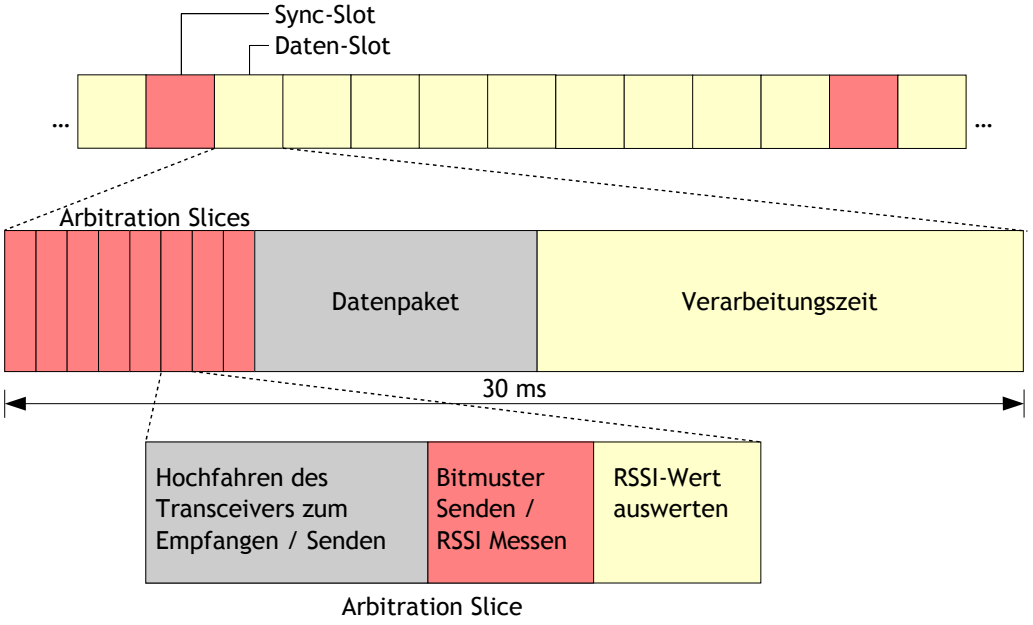

Abbildung 3.9: *Zeitliche Einordnung und Aufbau eines Arbitrierungs-Slices*

Um die Leistungsfähigkeit des Medienzugriffs einschätzen zu können, wurde im nächsten Abschnitt die Wahrscheinlicheit für eine Kollision in Abhängigkeit von der Anzahl der Knoten berechnet.

3.5.2 Kollisionswahrscheinlichkeiten

Bei der Arbitrierung ziehen alle Knoten, die auf das Medium sendend zugreifen möchten, eine 8-Bit Zufallszahl ungleich 0, alle anderen ziehen eine 0. Die Knoten gehen ihre Zahl schrittweise vom höchstwertigen Bit b_7 zum niedrigstwertigen Bit b_0 durch. Wenn mehrere Knoten gleichzeitig auf das Medium zugreifen möchten, erhält derjenige Knoten den Zugriff, der zuerst eine 1-Stelle hat, an der andere Knoten eine 0-Stelle haben.

Dies bedeutet, dass der Knoten gewinnt, der die größte Zahl gezogen hat. Andernfalls müsste es Zahlenpaare (x, y) geben, bei denen $x > y$ ist, aber trotzdem jedes Bit von y größer oder gleich dem entsprechenden Bit von x ist, denn nur dann würde y sich behaupten. Dies ist offensichtlich nicht möglich.

Eine Kollision liegt demnach vor, wenn die größte aller gezogenen Zahlen mehr als einmal vorkommt. In diesem Fall sehen alle Knoten, die diese höchste Zahl gezogen haben, das Medium als frei an und beginnen zu senden. Wie

wahrscheinlich ist es nun, dass wenn n Knoten eine Zahl ziehen, die höchste aller gezogenen Zahlen mehr als einmal vorkommt? Die Antwort liefert die Kombinatorik: Wir berechnen, wie viele Kombinationsmöglichkeiten es gibt und zählen auf, wieviele davon zu einer Kollision führen bzw. kollisionsfrei sind.

Für unsere Überlegungen schließen wir die passiven Knoten, die nicht senden möchten, aus. Diese ziehen eine 0 und nehmen somit nicht aktiv am Wettbewerb teil. Dieser findet nur unter den sendewilligen Knoten statt.

Alle Zahlen, die in einem Slot für die Arbitrierung gezogen werden, kann man als n-Tupel ansehen. Die Anzahl der möglichen n-Tupel ist 255^n, da jeder Knoten eine Zahl zwischen 1 und 255 zieht (ziehen mit Zurücklegen). Versuchen wir nun, die Möglichkeiten aufzuzählen, bei denen es zu keiner Kollision kommt (die *gültigen* Tupel):

Bei allen gültigen Tupeln gibt es genau eine größte Zahl z. Diese Zahl kann an irgendeiner Stelle im Tupel stehen, es gibt also n Möglichkeiten an welcher Stelle im Tupel die höchste Zahl steht.

Betrachten wir nun einen dieser Fälle: Wenn z die höchste Zahl im Tupel ist, gilt für alle anderen Zahlen im Tupel $0 < x < z$. Es gibt also $(z-1)^{n-1}$ Möglichkeiten für gültige Tupel wenn z an einer Stelle festgehalten wird.

Da z jetzt aber an einer beliebigen Stelle im Tupel stehen kann, haben wir insgesamt $n \cdot (z-1)^{n-1}$ gültige Tupel. Und da gilt $1 < z < 256$ müssen wir nur noch über alle z summieren. Dabei kann der Fall z=1 unberücksichtigt bleiben, denn wenn 1 die größte aller gezogenen Zahlen ist, haben alle diese Zahl (weil sie zwischen 1 und 255 gewählt wird) und das Tupel ist ungültig. Damit gibt es also die folgende Anzahl Tupel.

$$n \cdot \sum_{z=2}^{255} (z-1)^{n-1}$$

Damit ist die Wahrscheinlichkeit, dass eine Kollision auftritt gleich:

$$P_{\text{Kollision}}(n) = 1 - \frac{n \cdot \sum_{z=2}^{255} (z-1)^{n-1}}{255^n}$$

Noch allgemeiner ausgedrückt ist die Wahrscheinlichkeit bei einer Zufallszahlenlänge von l Bit gleich:

$$P_{\text{Kollision}}(n,1) = 1 - \frac{n \cdot \sum_{z=2}^{2^l-1} (z-1)^{n-1}}{(2^l-1)^n}$$

Abbildung 3.10 auf der nächsten Seite zeigt die Kollisionswahrscheinlichkeiten in Abhängigkeit der Anzahl der Knoten. Zusätzlich wurde die Länge der gezogenen Zufallszahl zwischen 8 und 11 Bit variiert, um Abschätzen zu können, wie diese sich auswirkt.

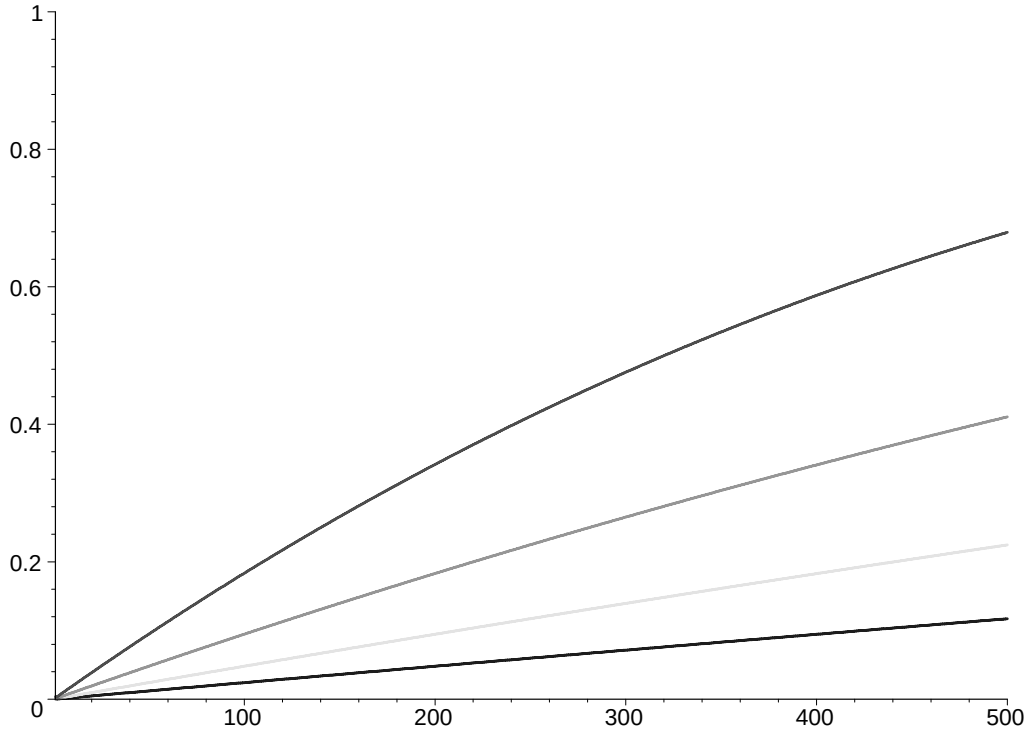

Abbildung 3.10: *Kollisionswahrscheinlichkeit in Abhängigkeit von der Länge der gezogenen Zufallszahl (oberste Kurve 8 Bit bis unterste Kurve 11 Bit) und der Anzahl der Knoten (x-Achse).*

Mit der in der Referenzimplementierung gewählten Länge von 8 Bit für die Zufallszahlen der Arbitrierung ergibt sich selbst bei 100 Knoten – ein Wert der in der Praxis selten erreicht werden sollte – eine Kollisionswahrscheinlichkeit von 0,1836 was für die meisten Anwendungen wohl ausreichen sollte.

3.5.3 Nutzung zur Energieeinsparung

Ein angenehmer Nebeneffekt des Arbitrierungs-Verfahrens ist, dass es zum Energie sparen genutzt werden kann. Wenn ein Knoten in einem Datenslot kein Paket zu Senden hat kann er die Arbitrierung einfach mit einer 0 statt mit einer Zufallszahl durchführen. Das hat zur Folge, dass in allen Arbitrierungs-Slices ins Medium gehört wird – der Knoten beteiligt sich passiv an der Arbitrierung. Beim ersten Slice in dem der Knoten ein Signal hört kann er die Arbitrierung abbrechen und nach allen Arbitrierungs-Slices in Empfangsbereitschaft gehen (da er in diesem Fall ja weiß, dass ein anderer Knoten etwas senden möchte). Hört er aber in allen Arbitrierungs-Slices kein Signal weiß der Knoten, dass kein Knoten im aktuellen Slot ein Paket senden möchte. Er muss also nicht in Empfangsbereitschaft gehen und versuchen, das Signal zu empfangen.

Messungen mit der Referenzimplementierung des Protokolls haben ergeben, dass der Energieverbrauch um ca. 30% gesenkt werden kann, wenn keine Pakete verschickt werden (was in der Regel in den meisten Slots der Fall sein wird).

Wenn ein Knoten in einem Slot nichts senden möchte, hat er zwei Optionen:

- Entweder er führt die Arbitrierung passiv durch (d. h. mit einer 0 als Zufallszahl) und geht abhängig von deren Ergebnis in Empfangsbereitschaft oder nicht,
- oder er macht die Arbitrierung nicht mit und versucht in jedem Fall ein Paket zu empfangen.

Die Messungen zeigen, dass die erste Option weniger Energie verbraucht.

3.5.4 Einschränkungen der Arbitrierung

Bei der Arbitrierung kommt es unter ungünstigen Voraussetzungen, die nachfolgend beschrieben werden, zu zusätzlichen Kollisionen. Dies ist der Fall, weil der Wettbewerb um das Medium bei den jeweiligen Senderknoten stattfindet, die Kollision jedoch bei den Empfängerknoten.

Bei *versteckten Knoten* wie in Abbildung 3.11 kommt es zu Kollisionen bei Knoten mit nicht vollvermaschter 1-Hop-Nachbarschaft. Da die Knoten der Gruppe 1 die Signale der Knoten der Gruppe 2 nicht empfangen können und umgekehrt, erhält jeweils einer aus den Gruppen den Medienzugriff und darf senden. Dadurch kommt es beim Knoten im Konfliktbereich zu zusätzlichen Kollisionen. Diese werden toleriert, da eine Auflösung aufwändig wäre und mit viel zusätzlichem Signalisierungsaufwand verbunden. Der Zielknoten müsste die ID (siehe Glossar) mindestens einer der Knoten erfahren, die ihm ein Paket schicken wollen und müsste dann einem der beiden ein Senderecht erteilen. Der zusätzliche Energie- und Zeitaufwand wird durch den gewonnenen Nutzen nicht aufgewogen.

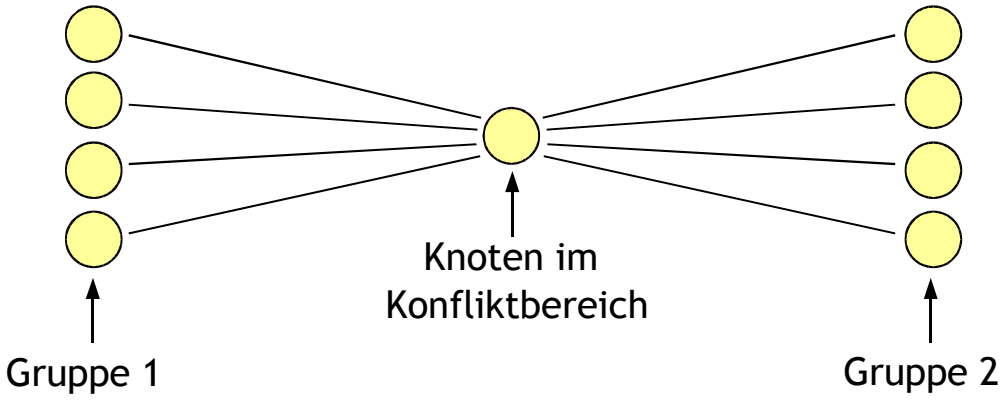

Abbildung 3.11: *Versteckte Knoten bei der Synchronisation. Die Knoten der Gruppe 2 sind für die Knoten der Gruppe 1 versteckt und umgekehrt. Beim Konfliktknoten ist die 1-Hop-Nachbarschaft nicht vollvermascht.*

Bei *ausgelieferten Knoten* wird der Medienzugriff verwehrt, obwohl er ohne Kollisionen möglich wäre (siehe Abbildung 3.12 auf der nächsten Seite). Dieses Problem betrifft das Particle-Protokoll nur am Rande, denn ausgelieferte Knoten spielen nur bei Unicast-Verkehr eine Rolle. Im Particle-Protokoll werden hingegen standardmäßig Broadcasts verschickt und Unicast-Sendungen (siehe Glossar) kommen nur im Einzelfall vor.

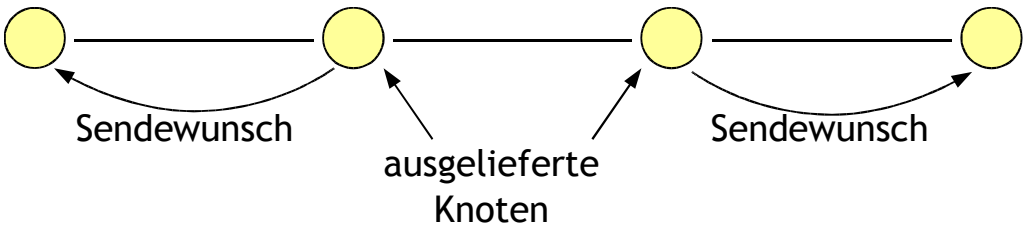

Abbildung 3.12: *Ausgelieferte Knoten bei der Synchronisation. Die mittleren Knoten sind einander ausgeliefert, obwohl sie senden könnten, ohne dass Konflikte auftreten.*

3.6 Verteilte Synchronisation

Die in Abschnitt 3.4.2 auf Seite 22 geschilderte Situation ist im Master-Slave-Modus nicht leicht auflösbar. Allgemein kann man sagen, dass die Erzielung und Aufrechterhaltung eines korrekten Master-Slave-Zustandes über das gesamte Netz hinweg aufwändig ist. Im zeitlich letzten Drittel dieser Arbeit wurde eine Alternative zum bisherigen Ansatz entwickelt – eine Synchronisation, die nicht auf die Status Master und Slaves angewiesen ist.

Das neue Verfahren geht wie folgt vor: Die Unterscheidung zwischen Beaconslots und Datenslots wird aufgehoben. Jeder Knoten entscheidet nun in jedem Slot zufällig, ob er ein Beacon sendet oder eine normale Arbitrierung durchführt. Die Wahrscheinlichkeit P_{Sync} ein Beacon zu senden ist abhängig von der Nummer des aktuellen Slots. Die Slotnummer wird zu Beginn jedes Slots um 1 erhöht und beim Empfang oder beim Senden eines Beacons jeweils auf 0 zurückgesetzt. Isoliert für jeden Slot betrachtet sind die genauen Wahrscheinlichkeiten wie folgt gewählt:

$$P_{sync}(slot) = \begin{cases} 0 & \text{für } slot < 10 \\ 1/4^{15-slot} & \text{für } 10 \leq slot \leq 15 \end{cases}$$

Abbildung 3.13 auf der nächsten Seite zeigt die Werte für die maßgeblichen Slots.

Damit lässt sich nun berechnen, wie wahrscheinlich es ist, dass im n-ten Slot nach dem letzten Beaconslot das nächste Beacon gesendet wird ($P_{\text{beacon}}(slot)$). Diese berechnet sich aus der Wahrscheinlichkeit, dass bisher noch kein Beacon gesendet wurde multipliziert mit der, dass im aktuellen Slot eines gesendet wird:

$$P_{\text{beacon}}(slot) = P_{sync}(slot) \cdot \prod_{i=0}^{slot-1} (1 - P_{sync}(i))$$

Durch die statistische Verteilung kann nun kein Slot mehr explizit als Beaconslot ausgewiesen werden, in dem kein Knoten versucht, ein Datenpaket zu senden und genau ein Knoten (der Master) exklusiven Zugriff auf das Medium erhält. Daher ist es nötig, die Arbitrierung so zu erweitern, dass Beacons gegenüber normalen Datenpaketen bevorzugt werden. Zusätzlich muss garantiert sein, dass nur ein Knoten zum Senden des Beacons zugelassen wird, falls mehrere Knoten gleichzeitig versuchen eines zu senden.

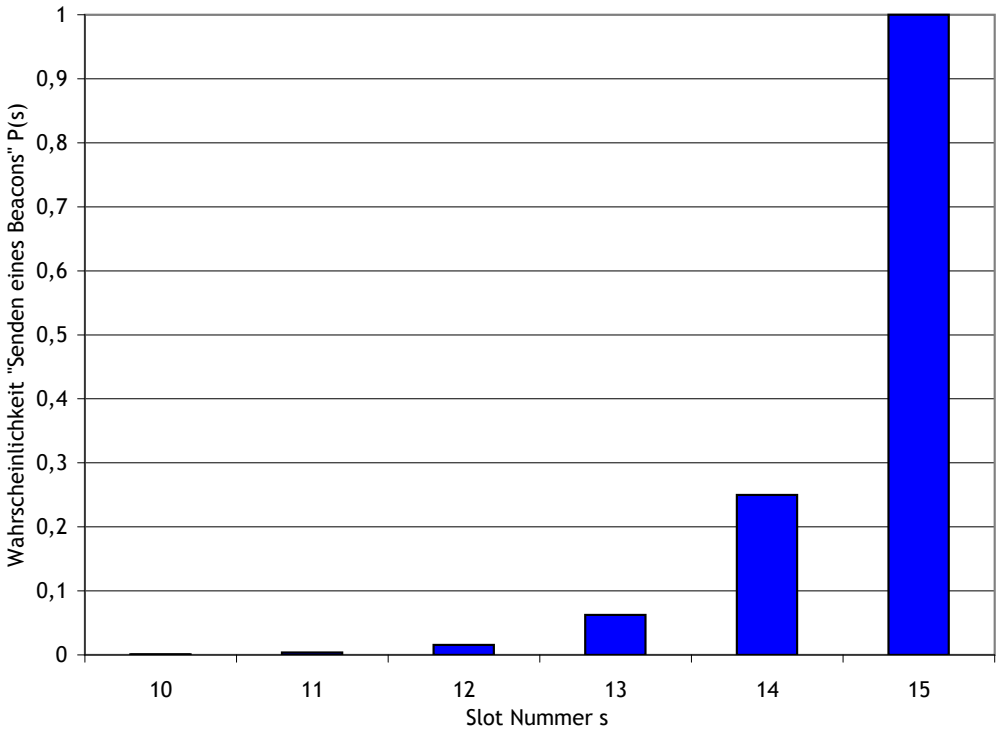

Abbildung 3.13: *Wahrscheinlichkeiten dass ein Beacon gesendet wird, in Abhängigkeit von der aktuellen Slotnummer, unabhängig von den voraus gehenden Slots:* $P_{sync}(slot)$

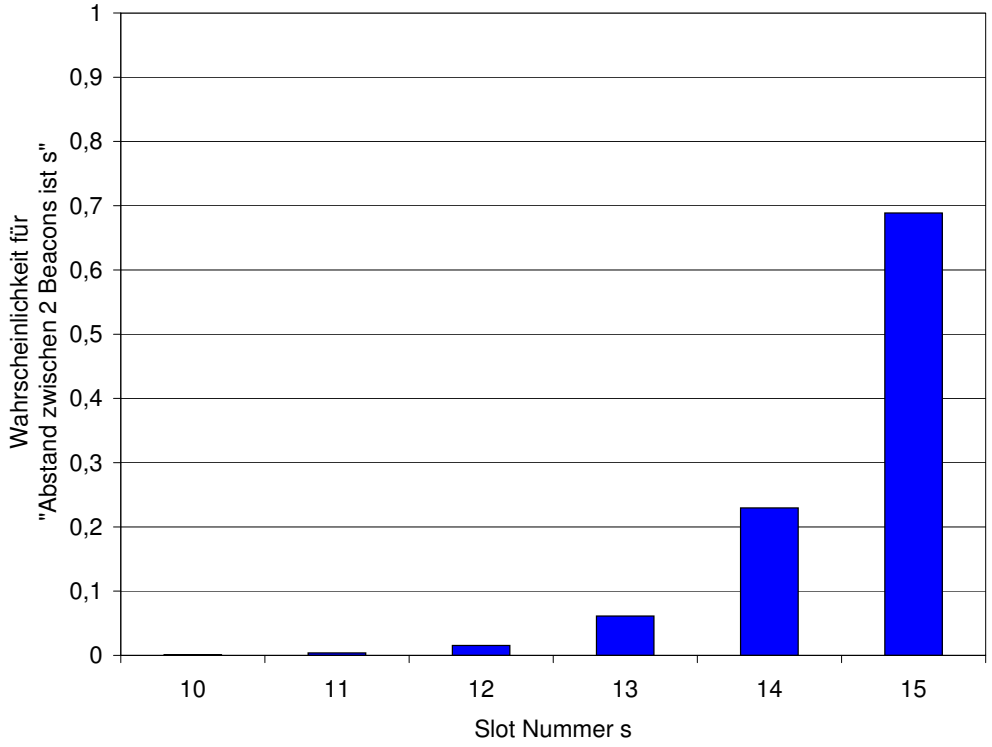

Abbildung 3.14: *Wahrscheinlichkeiten dass nach dem letzen Beacon im angegebenen Slot ein weiteres Beacon gesendet wird:* $P_{beacon}(slot)$

Beide Ziele lassen sich durch eine einfache Erweiterung des Arbitrierungs-Konzeptes erreichen: Den 8 Arbitrierungs-Slices wird einfach ein weiterer Arbitrierungs-Slice vorangestellt (siehe Abbildung 3.15). In diesem Slice dürfen nur die Knoten senden, die im aktuellen Slot ein Beacon senden möchten – alle anderen Knoten hören in diesem Slice. Falls ein Knoten, der kein Beacon schicken möchte, ein Signal in diesem Slice hört scheidet er sofort aus. So ist sichergestellt, dass die nachfolgenden 8 Slices entweder nur den Wettbewerb unter Datenpaketen oder nur unter den Beacons regeln, wobei Beacons priorisiert werden.

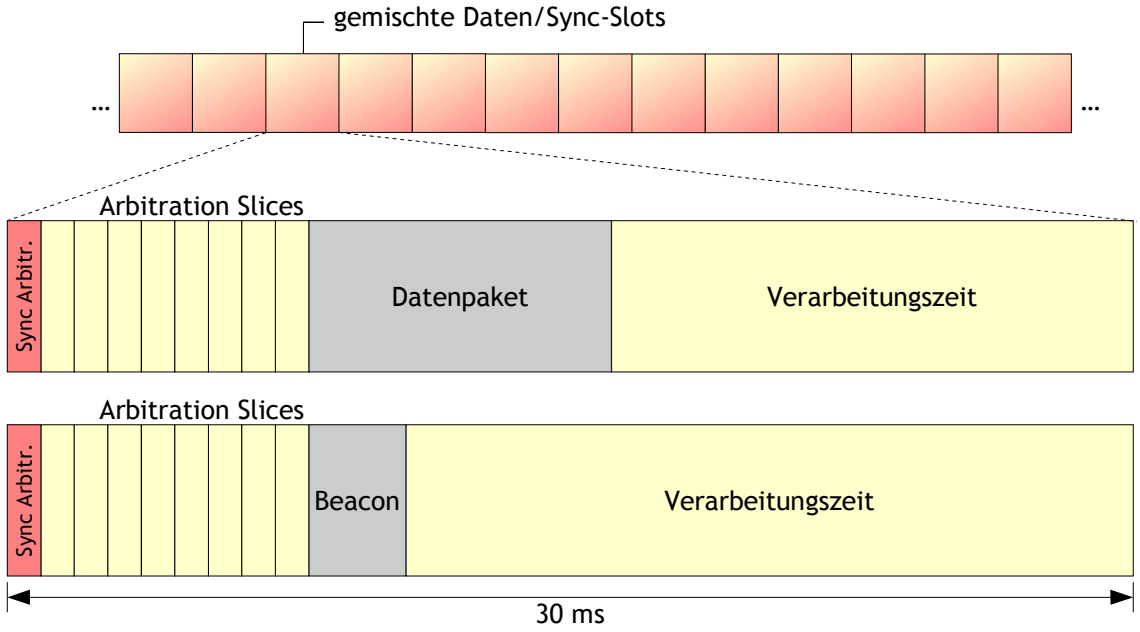

Abbildung 3.15: *Daten- und Beaconslots mit 9 Arbitrierungs-Slices für die verteilte Synchronisation*

Die Wahrscheinlichkeit, dass ein Knoten in einem Slot ein Beacon sendet, ist abhängig von der Slotnummer so gewählt. Bis zum 9. Slot ist die Wahrscheinlichkeit 0, vom 10 bis zum 15 Slot wächst sie exponentiell bis sie schließlich im 15. Slot eine Wahrscheinlichkeit von 1 erreicht. In jedem Netz wird somit unabhängig von der Anzahl der Knoten mindestens alle 15 Slots ein Beacon gesendet. Die Wahrscheinlichkeit von 0 für die ersten 9 Slots stellt sicher, dass die Beacons nicht zu oft gesendet werden, und damit Bandbreite verschwenden.

Der exponentielle Anstieg der Wahrscheinlichkeit zwischen Slot 10 und Slot 15 stellt sicher, dass sich in sehr großen Netzen (Größenordnung 1000 Knoten) nicht zu viele Knoten gleichzeitig entschließen, ein Beacon zu senden. Ausgehend von einer Wahrscheinlichkeit von 1/1024 entscheiden sich in jedem Slot vier mal mehr Knoten dazu, ein Beacon zu senden.

Theoretisch sollte der Fall, dass sehr viele Knoten gleichzeitig ein Beacon senden wollen zwar von der Arbitrierung abgefangen werden und Abbildung 3.10 auf Seite 29 zeigt dass das theoretisch auch gut funktioniert, es liegen jedoch keine praktischen Erfahrungen vor, ob diese bei extrem vielen, gleichzeitigen Sendewünschen noch zuverlässig funktioniert. Eine vom Slot unabhängige

Wahrscheinlichkeit (z. B. $P_{sync}(\text{slot}) = 0,5$) oder gar festgelegte Anzahl von Slots (z. B. $P_{sync}(\text{slot}) = 0$ für slot < 10 und $P_{sync}(\text{slot}) = 1$ für slot ≥ 10) wären fatal, weil eben zu viel Knoten gleichzeitig versuchen würden ein Beacon zu senden.

Durch die verteilte Synchronisation fällt jeglicher Aufwand weg, der mit der eindeutigen Festlegung und dem Aufrechterhalten der Master- und Slave-Rollen verbunden ist. Dies macht das Protokoll leichter verständlich und auch leichter implementierbar. Abbildung 3.16 zeigt, dass das Verfahren quasi zustandslos arbeitet – die einzigen anderen Zustände außerhalb des Normalbetriebs sind das gründliche Suchen nach Beacons, falls länger keines Empfangen wurde und der Tiefschlaf, wenn der Knoten alleine ist (siehe Abschnitt 3.9 auf Seite 36). Der Vergleich mit Abbildung 3.2 auf Seite 19 zeigt die deutlich verminderte Komplexität.

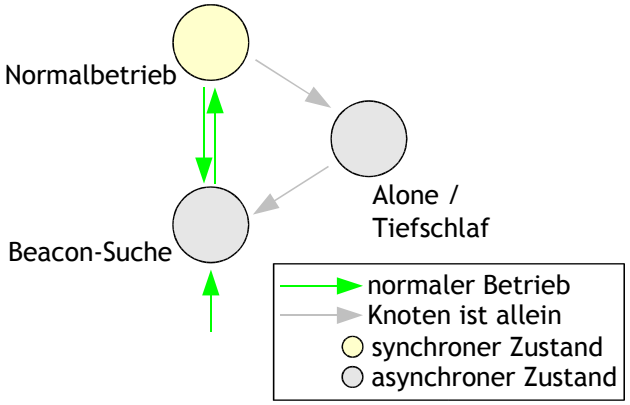

Abbildung 3.16: *Zustandsautomat des Synchronisationsteils bei verteilter Synchronisation*

3.7 Vergleich der Synchronisationsmethoden

Beide bisher vorgestellten Synchronisationsmethoden haben ihre Vor- und Nachteile, die in diesem Abschnitt kurz aufgeführt werden.

Die Stärken der Master-Slave-Synchronisation liegen in voll vermaschten Netzen, d. h. Netzen in denen sich jeder Knoten in der Übertragungsreichweite eines jeden anderen Knoten befindet. Dort findet die Synchronisation zuverlässig und präzise statt.

In einem Multi-Hop-Netz, das ist ein Netz, in dem manche Teilnehmer nur über Zwischenknoten erreichbar sind, funktioniert die Master-Slave-Synchronisation nicht. Hier sind mehrere Master nötig, um alle Knoten synchronisiert zu halten. Knoten im Konfliktbereich, d. h. Knoten die die Beacons mehr als eines Masters empfangen können, haben demnach einen undefinierten Zustand. Hier funktioniert die verteilte Synchronisation besser. Man muss jedoch bedenken, dass sich die zeitlichen Abweichungen per Hop summieren.

3.8 Verkehrsanzeige

Mit den bisher vorgestellten Mechanismen ist es den Knoten möglich, ihre Transceiver die meiste Zeit ausgeschaltet zu lassen, ohne ein Paket zu

versäumen. Die Knoten müssen jedoch immer noch in jedem Slot (bei Master-Slave-Synchronisation in jedem Datenslot) die Arbitrierung durchführen, auch wenn über eine längere Zeit keine Daten gesendet werden. Wenn für die Anwendung eine leichte Erhöhung der Latenz (das ist die Zeit zwischen Beauftragung des Sendens eines Paketes durch die Anwendung und dem Start der Übertragung) vertretbar ist, kann auch die Arbitrierung in allen Slots zwischen zwei Beaconslots entfallen, wenn im Netz keine Datenpakete verschickt werden müssen. Damit können die Transceiver komplett deaktiviert bleiben (bis auf eine kleine Außnahme bei verteilter Synchronisation).

Um dies zu erreichen, wird das Prinzip der Verkehrsanzeige (kurz VA, engl. auch Traffic Indication) benutzt: In den Beaconslots wird ein kleines Zeitfenster für das Senden der VA freigehalten. Dabei kommt die gleiche Technik wie bei einem Arbitrierungs-Slice zum Einsatz. Alle Knoten, die ein Paket senden möchten, senden an der definierten Stelle im Beaconslot ein Signal aus. Alle Knoten, die nichts zu senden haben, hören an gleicher Stelle ins Medium.

Die Slots zwischen einem Beaconslot in dem eine VA gesendet oder gehört wurde und dem nächsten Beaconslot, werden eine *Verkehrsgruppe* (engl. Traffic Group) genannt. Die Slots zwischen einem Beaconslot in dem keine VA gesendet wurde und dem nächsten Beaconslot, bilden analog eine *Leerlaufgruppe* (engl. Idle Group) – siehe Abbildung 3.17.

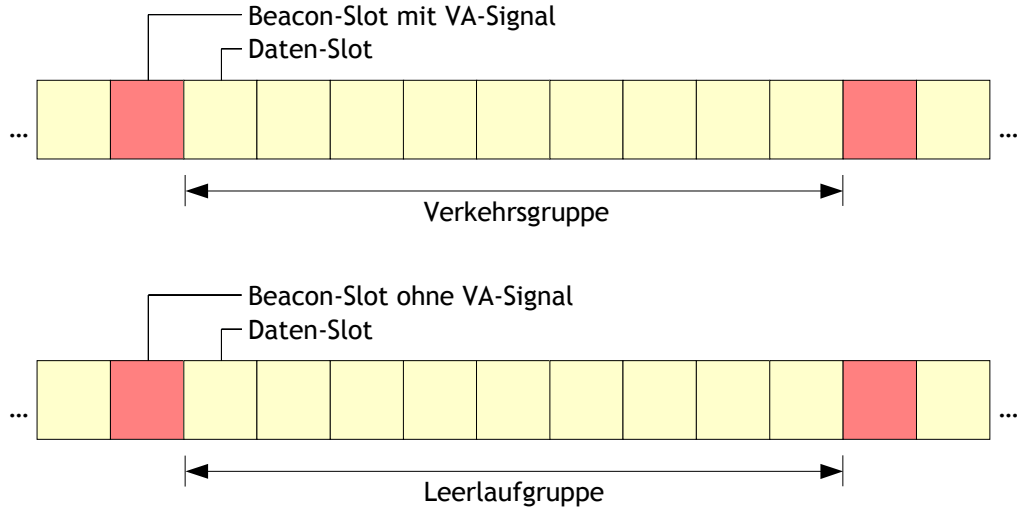

Abbildung 3.17: *Verkehrsgruppe und Leerlaufgruppe bei Verwendung von Verkehrsanzeige*

Wenn die Anwendung das Senden eines Paketes beauftragt und sich das Netz in einer Verkehrsgruppe befindet, darf das Paket sofort gesendet werden. Befindet sich das Netz aber in einer Leerlaufgruppe, wird das Senden des Paketes bis zur nächsten Verkehrsgruppe verzögert.

Durch diese Maßnahmen wird sichergestellt, dass Pakete nur in Verkehrsgruppen, nicht aber in Leerlaufgruppen gesendet werden. Somit kann ein Knoten in einem Netz mit Master-Slave-Synchronisation die Arbitrierung in allen Leerlaufgruppen weglassen und somit den Transceiver bis zum nächsten Beaconslot komplett deaktivieren.

In einem Netz mit verteilter Synchronisation dagegen kann in Leerlaufgruppen nicht komplett auf die Arbitrierung verzichtet werden. Da es keine für Beacons reservierten Slots gibt, würde das auf eine Leerlaufgruppe folgende Beacon überhört werden. Um dies zu verhindern genügt jedoch, das erste Arbitrierungs-Slice auszuwerten. Falls in diesem Slice ein Signal zu hören ist, weiß der Knoten, dass ein anderer Knoten ein Beacon senden möchte und führt die Arbitrierung wie gewohnt fort.

Der Zeitpunkt des VA-Slices innerhalb des Sync-Slots hängt davon ab, ob Master-Slave- oder verteilte Synchronisation verwendet wird. Abbildung 3.18 skizziert die verschiedenen Zeitpunkte.

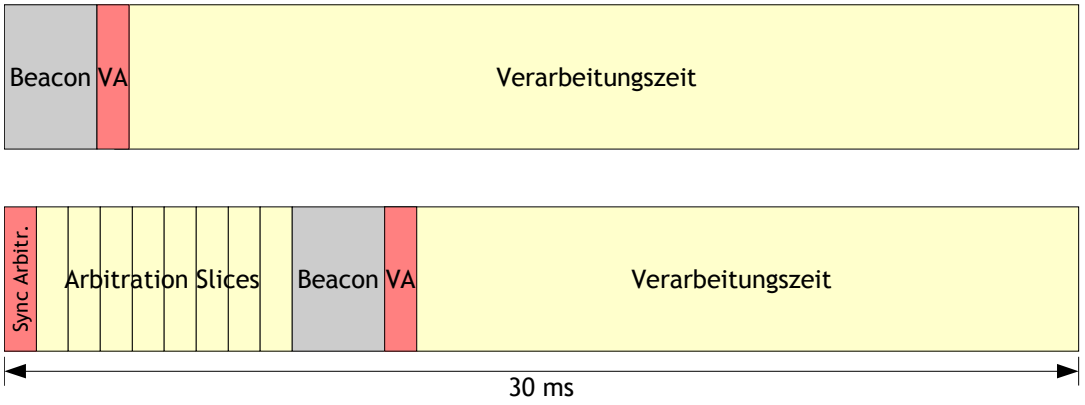

Abbildung 3.18: *Zeitliche Lage der VA im Beaconslot bei Master-Slave- und verteilter Synchronisation*

3.9 Schlafmodus

In einem Sensornetzwerk kann es vorkommen, dass ein Knoten alleine ist, d. h. es ist keine Kommunikation zu anderen Knoten möglich. Dies kann z. B. daran liegen, dass er sich räumlich vom Netz wegbewegt hat, oder ein Störsignal den Funkkanal zeitweise unbrauchbar macht. In diesem Fall ist es günstig, wenn der Knoten das Wissen um die Unmöglichkeit der Kommunikation nutzt, um Energie zu sparen.

Die Aufgabe Energie zu sparen, wenn ein Knoten alleine ist, kann man in folgende Teilaufgaben zerlegen:

- Es muss zuverlässig festgestellt werden können, dass ein Knoten alleine ist, also nicht mit anderen Knoten kommunizieren kann.
- Wenn die Isoliertheit festgestellt wurde, muss möglichst viel Energie gespart werden. Dabei geht er für eine begrenzte Zeit in einen *Schlafmodus* während dessen er keine Pakete verschickten oder empfangen kann. Nach dieser Zeit wird der Normalbetrieb fortgesetzt.
- Das hin- und herwechseln zwischen Schlafmodus und Normalbetrieb muss so gestaltet werden, dass sich zwei Knoten, die sich in diesem Zyklus befinden, innerhalb angemessener Zeit finden und in den dauerhaften Normalbetrieb übergehen. Die dafür zugestandene Zeit kann wesentlich länger sein (in der Größenordnung von Minuten) als die für

die Aufnahme eines neuen Knotens in ein Netz im Normalbetrieb (wenige Sekunden).

3.9.1 Erkennen der Isoliertheit

Voraussetzung Energie sparender Maßnahmen ist also, dass ein Knoten zuverlässig erkennen kann, dass er alleine ist. Die Tatsache, dass er keine Datenpakete empfängt, reicht für diese Schlussfolgerung nicht aus. Schließlich gibt es keine Obergrenze für die Zeit zwischen zwei Datenpaketen – es gibt durchaus Anwendungen, bei denen es genügt, alle paar Stunden ein Paket zu senden. Bessere Grundlage für die Entscheidung ist der Empfang von Beacons.

Bei verteilter Synchronisation sendet jeder Knoten von Zeit zu Zeit ein Beacon. Wenn ein Knoten also längere Zeit keine Beacons hört, kann er davon ausgehen, dass er alleine ist. Schwieriger wird es bei Master-Slave-Synchronisation. In diesem Falle wissen zwar die Slaves trivialerweise, dass sie nicht alleine sind, ein Master der längere Zeit keine Datenpakete empfängt kann aus dem genannten Grund aber nicht einfach davon ausgehen, dass er alleine ist. Eine mögliche Lösung für dieses Problem ist eine Rückmeldung aller Slaves nach Senden des Beacons im Beaconslot. Dazu kann zu einem festgelegten Zeitpunkt die gleiche Technik wie bei einem Arbitrierungs-Slice eingesetzt werden: der Master hört, ob auf dem Medium ein Signal liegt und alle Slaves senden ein Signal an dieser Stelle (siehe Abbildung 3.19).

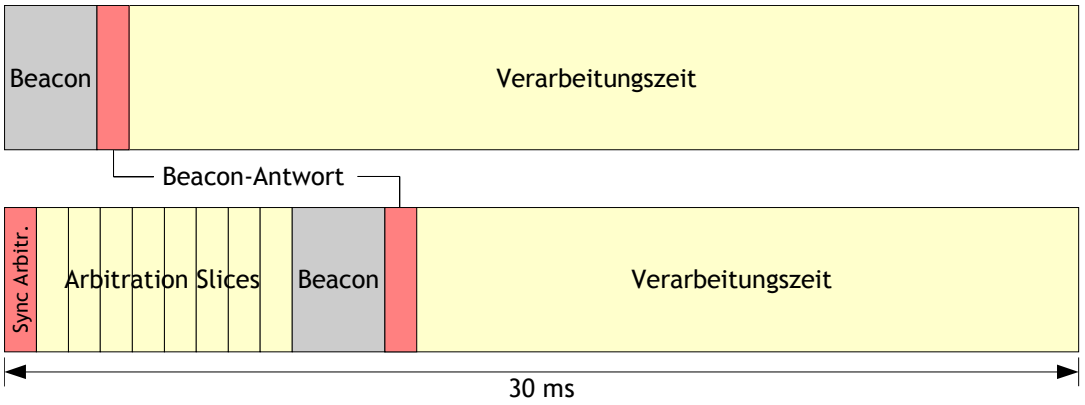

Abbildung 3.19: *Beaconantwort im Beaconslot bei Master-Slave- und verteilter Synchronisation*

Während der Implementierung des Schlafmodus für die verteilte Synchronisation wurde die Entscheidung getroffen, diese Form der Rückmeldung auch dort einzusetzen. Durch die statistische Verteilung und Arbitrierung beim Senden des Beacons kann es nämlich vorkommen, dass ein bestimmter Knoten in einer langen Folge von Slots das Beacon sendet. Kommt dann der Umstand hinzu, dass die Anwendung gerade keine Datenpakete sendet, bekommt der dieser Knoten keine Rückmeldung von anderen Knoten und glaubt, er sei alleine. Die Wahrscheinlichkeit, dass nur ein einziger Knoten das Beacon in Folge sendet, nimmt natürlich mit der Anzahl der Slots ab, die man für das Erkennen der Isoliertheit eines Knotens berücksichtigt – eine sichere Anzahl von Slots kann jedoch nicht angegeben werden. Außdem gilt: Je länger man

das Detektionsintervall wählt, desto länger muss man die durchschnittliche Dauer der Schlafphase wählen um ein niedriges Verhältnis zwischen Wach- und Schlafphasen zu erreichen und je länger man diese Phasen wählt, desto länger dauert die Konfliktauflösung, wenn sich zwei Knoten im Schlafmodus begegnen (s. Abschnitt 5.3).

Folgendes ist zu beachten: Wenn das Verfahren Verkehrsanzeige (siehe Abschnitt 3.8) und die Beaconantwort gleichzeitig eingesetzt werden, müssen die beiden Slices im Beaconslot natürlich hintereinander angeordnet werden. Hier wird als Reihenfolge festgelegt, dass zuerst die Beaconantwort und dann die Verkehrsanzeige erfolgt. Abbildung 3.20 stellt diesen Umstand dar.

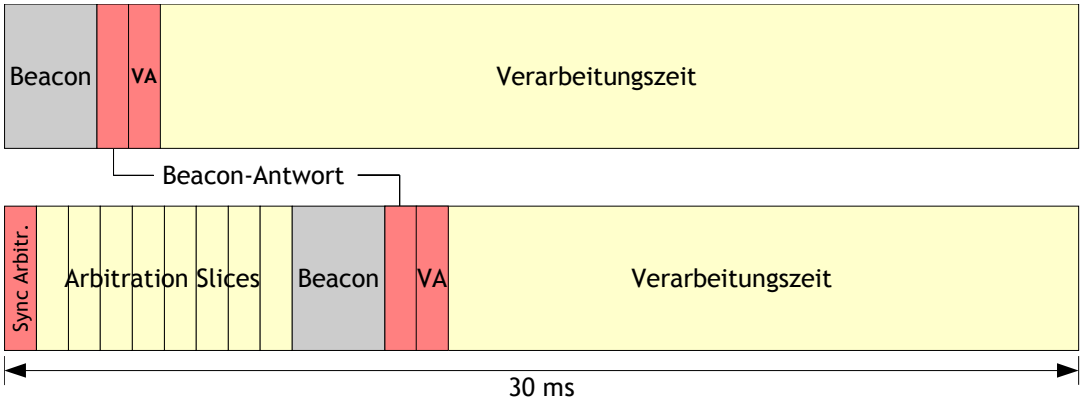

Abbildung 3.20: *Gleichzeitige Nutzung von Beaconantwort und Verkehrsanzeige im Beaconslot bei Master-Slave- und verteilter Synchronisation*

3.9.2 Nutzung der Isoliertheit zur Energieeinsparung

Nun, da der Knoten weiß, dass er alleine ist, kann er davon ausgehen, für eine gewisse Zeit nicht mit anderen Knoten kommunizieren zu können. Dieses Wissen kann er nutzen um möglichst viel Energie zu sparen. In diesem Falle kann das Netzwerkprotokoll Pakete, die die Anwendung zum Senden beauftragt hat, einfach ignorieren. Es verhält sich gegenüber der Anwendung so, als würde es die Pakete verschicken (es meldet das erfolgreiche Senden zurück), tatsächlich aber führt es weder die Arbitrierung noch das Senden des Paketes durch. Ebensowenig versucht der Stack Pakete zu Empfangen.

Abhängig vom Typ der Anwendung kann man noch einen Schritt weiter gehen: Bei einem Knoten, der alleine ist, kann man auch die Anwendung in regelmäßigen Intervallen für längere Zeit deaktivieren, schließlich kann der Knoten keine Pakete empfangen, die er verarbeiten müsste, und auch das Senden der eigenen Sensordaten ist überflüssig. Die letztliche Entscheidung darüber, ob solch ein temporäres Aussetzen der Verarbeitung möglich ist, sollte der Anwendung überlassen werden, die dies zur Laufzeit verbieten oder erlauben können sollte. Falls die Anwendung den Tiefschlaf erlaubt, kann ein verwaister Knoten nun regelmäßige Tiefschlaf-Phasen einlegen.

3.9.3 Auffinden von Knoten im Schlafmodus

Nach beiden Stufen des Energie sparens (Kommunikationsunterdrückung oder Tiefschlaf) läuft der Knoten wieder im Normalzustand. Stellt der Knoten fest, dass er immer noch alleine ist, betritt er wieder den Energiesparzustand.

Bei diesem zyklischen hin- und herwechseln zwischen Energiespar- und Normalzustand muss sichergestellt sein, dass zwei partiell tiefschlafende Knoten sich in akzeptabler Zeit finden können[12]. Zu dieser Situation kann es beispielsweise kommen, weil sie zunächst weit voneinander entfernt waren und nun aufeinander zukommen. Dies kann erreicht werden, indem das Tiefschlaf-Intervall zufällig gewählt wird. Dies ermöglicht zwar keine garantierte Höchstzeit, in der sich die Knoten finden, aber die Wahrscheinlichkeit dafür nimmt mit der Zeit stetig zu. In Abschnitt 5.3 auf Seite 64 werden die Ergebnisse der Simulation zu diesem Problem vorgestellt.

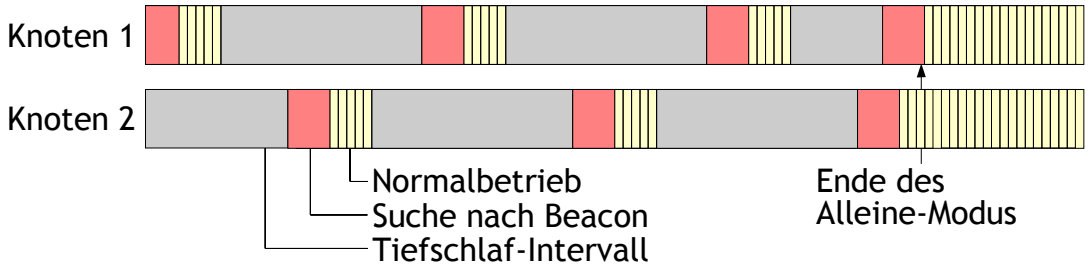

Abbildung 3.21: *typischer zeitlicher Verlauf von zwei Knoten im Schlafmodus und Auflösung des Konfliktes*

Abbildung 3.21 zeigt einen möglichen Ablauf der Rückkehr zweier Knoten aus dem Schlafmodus mit Intervall-Tiefschlaf zum Normalbetrieb. Entscheidend ist hier, dass nach dem Aufwachen aus dem Tiefschlaf eine Beaconsuche stattfinden muss (der Knoten hat nämlich das strenge Zeitraster der Slotgrenzen verlassen und muss versuchen sich einer evtl. bereits aufgebauten Synchronisation zwischen Knoten anzuschließen), gefolgt von einer Periode in Normalbetrieb, die eine gewisse Mindestlänge aufweisen muss. Dies ermöglicht einen reibungslosen Übergang in den dauerhaften Normalbetrieb sowohl bei Master-Slave- als auch bei verteilter Synchronisation.

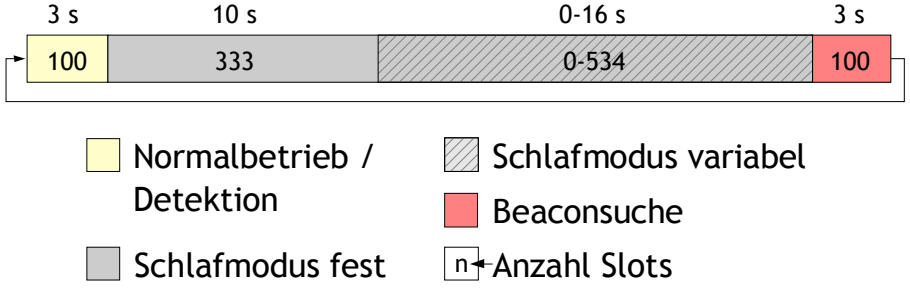

Abbildung 3.22: *genaue Zeitverhältnisse beim Wechseln zwischen Normal- und Schlafmodus bei einem verwaisten Knoten*

Abbildung 3.22 zeigt die genaue Dauer für Normal- und Schlafintervalle, die in der Referenzimplementierung gewählt wurden und die Verhältnisse dazwischen. Eine Begründung für die Wahl genau dieser Verhältnisse liefert Abschnitt 5.3 auf Seite 64.

[12] Zwei Knoten sind dabei der ungünstigste Fall. Sind mehr als zwei Knoten im Schlafmodus so findet ein aufwachender Knoten mit höherer Wahrscheinlichkeit ein Beacon auf dem Medium.

Wie man sieht, ist ein verwaister Knoten regelmäßig 6 Sekunden wach und befindet sich durchschnittlich 18 Sekunden im Energiesparmodus. Das ergibt ein Verhältnis von Wach- zu Gesamtzeit von $^1/_4$.

3.10 Scrambling

Um die Übertragung robuster zu machen – insbesondere um Gleichstromfreiheit und lange 0- bzw. 1-Folgen zu vermeiden – werden die zu sendenden Daten verwürfelt (engl. scrambled). Dazu wurde der Scrambling-Algorithmus des populären drahtlosen LAN-Standards W-LAN implementiert, der im IEEE[13] 802.11-Standard definiert ist. Es wird das Polynom $G(z) = Z^{-7} + Z^{-4} + 1$ benutzt. [802.97]

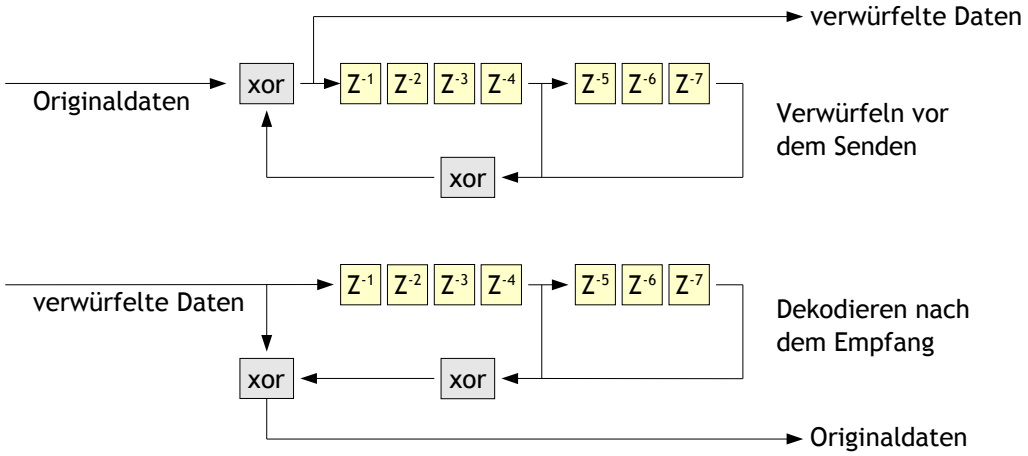

Abbildung 3.23: *Scrambling und Descrambling*

Abbildung 3.23 zeigt das Verfahren anschaulich. Der Vorgang ist selbst synchronisierend, d. h. der Wert des verwendeten 7-Bit-Schieberegisters ist nach spätestens 7 übertragenen Bits bei Sender und Empfänger gleich. Daher wird am Anfang des RF-Headers, gleich nach dem Sync-Byte ein verwüfeltes Byte mit dem Wert 255 (nur 1-Bits) gesendet. Nach Empfang und Descrambling dieses Bytes kann der Empfänger alle nachfolgenden Bytes korrekt decodieren, unabhängig davon, welchen Wert das Register vorher hatte.

Der Mikrokontroller-Kern des CC1010 hat genügend Rechenkapazität um die Ver- und Entwürfelung in Echtzeit durchzuführen, d. h. nachdem ein Byte in das Sende-Pufferregister geschrieben wurde bleibt genug Zeit für die Codierung bis der Puffer bereit ist, das nächste Byte aufzunehmen (siehe hierzu Abschnitt 4.3.3 auf Seite 52).

3.11 Aktive Unterbrechung der Anwendung

In diesem Abschnitt wird eine weitere Maßnahme zum Energie sparen vorgestellt, die eigentlich nicht Teil des Protokolls ist. Sie ist stark abhängig von der später dokumentierten Implementierung des Protokolls und der Hardware auf der die Implementierung ausgeführt wird. Daher muss an dieser

[13] Abkürzung für Institute of Electrical and Electronics Engineers

Stelle auf die Abschnitte 4.3 und 4.4.1 vorgegriffen werden. Die verwendeten, noch nicht eingeführten Begriffe werden dort erklärt.

Da die vorliegende Implementierung ohne Betriebssystem durchgeführt wurde, werden alle zeitkritischen Vorgänge in einer *Interrupt Service Routine* (ISR) durchgeführt (die zu Beginn eines Slots aufgerufen wird). In der restlichen Ausführungszeit außerhalb der ISR wird die eigentliche Anwendung ausgeführt. Diese greift über asynchrone, nicht blockierende Funktionen mit hohem Abstraktionsgrad auf die Protokolldienste (z. B. Senden und Empfangen von Paketen) zu.

Kann eine Anwendung nun voraussehen, dass sie im aktuellen Slot keine weiteren Aktionen durchführen muss, so kann sie den Prozessor in einen Energiesparmodus versetzen, in dem er nur wenig Energie verbraucht. Dieser Modus wird mit dem Beginn des nächsten Slots (also mit Aufruf der ISR) automatisch unterbrochen.

Anwendungsbeispiel: Ein Programm, dass auf das Eintreffen neuer Pakete wartet, fragt in einer Schleife fortwährend den Status des Protokolls ab. Ist ein neues Paket eingetroffen, so verarbeitet es dessen Inhalt. Nach der Abfrage kann es den Prozessor in den Energiesparmodus versetzen. Anstatt ständig den Status des Protokolls abzufragen ob ein neues Paket eingetroffen ist – der sich bis zum nächsten Slot bzw. Aufruf der ISR definitiv nicht ändern kann – wird die Ausführung der Anwendung angehalten und erst mit im nächsten Slot, nach Rückkehr aus der ISR, fortgeführt. Abbildung 3.24 zeigt das Prinzip.

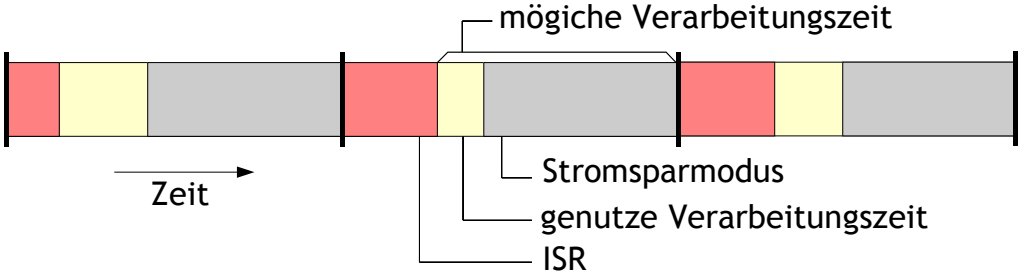

Abbildung 3.24: *von der Anwendung veranlasste Unterbrechung*

Für das Auslösen der Unterbrechung wird in der Referenzimplementierung ein C-Makro (`RF_LOW_POWER_SLEEP_UNTIL_NEXT_SLOT()`) zur Verfügung gestellt. Es entspricht prinzipiell der `yield()`-Funktion, die in vielen Mehrprozess-Systemen dafür genutzt wird, um die Ausführung des aktuellen Prozesses zu unterbrechen, da dieser gerade keine weitere Rechenzeit benötigt. Da auf dem CC1010 aber immer nur ein Prozess läuft, nämlich entweder die Anwendung oder der Stack, wird der Prozessor eben so lange in den Energiesparmodus versetzt, bis die nächste Ausführung des Stacks ausgelöst wird.

3.12 Bewertung

Das in diesem Abschnitt entworfene Protokoll erfüllt die eingangs gestellten Anforderungen an ein Netzwerkprotokoll für drahtlose Sensornetzwerke.

- Geringer Energieverbrauch
  - Die Knoten verbrauchen wenig Energie, weil sie ihre Transceiver nur während kurzer Zeitintervalle aktivieren, was auf allen Knoten synchronisiert geschieht.
  - Der Medienzugriff wird zum Energie sparen mitverwendet. Nicht sendewillige Knoten beteiligen sich passiv an der Arbitrierung und versuchen nur dann ein Paket zu empfangen, wenn sie während der Arbitrierung ein Signal eines sendewilligen Knotens hören.
  - Zwischen zwei Beaconslots können Knoten ihre Transceiver ganz oder größtenteils deaktiviert lassen, wenn die Verkehrsanzeige genutzt wird und gerade keine Daten zu verschicken sind. Dieses Verfahren ist optional und kann zur Übersetzungszeit aktiviert werden. Die durchschnittliche Latenz erhöht sich dann um 5 Slots (150 ms).
  - Knoten, die räumlich oder durch einen gestörten Kanal isoliert sind erkennen diesen Umstand und können ihn zum Energie sparen nutzen.
  - Die Anwendung kann, wenn sie nicht die gesamte Rechenzeit benötigt, den Prozessor bis zum nächsten Slot in einen Schlafmodus versetzen.
- Effektive Mediennutzung
  - Die Arbitrierung stellt ein wirksames Mittel dar, um Kollisionen innerhalb eines Slots zu vermeiden und den einzelnen Knoten gleichberechtigten Medienzugriff zu ermöglichen.
- Minimales Aufkommen von Signalisierungsdaten und schnelles Auffinden und Verbinden mit anderen Knoten nach der Initialisierung
  - Diese Ziele stehen in Konflikt. Um eine schnelle Verbindung nach dem aktivieren eines Knotens zu gewährleisten, müssen die Beacons häufig gesendet werden, um den Signalisierungsverkehr zu minimieren, sollten sie möglichst selten gesendet werden. In unserem Fall wurde der Schwerpunkt auf eine schnelles Einbuchen gelegt. Der aufwändige 4-Wege-Handshake bei der Weitergabe der Masterrolle wird nur sehr selten durchgeführt, was den Protokollaufwand rechtfertigt.
- Automatische Konfiguration
  - Nach dem Einschalten verbinden sich die Knoten automatisch – das Protokoll läuft ohne Administrations-Aufwand.
- Berücksichtigung von versteckten und ausgelieferten Knoten
  - Ausgelieferte Knoten spielen bei Broadcastnachrichten, wie sie im vorliegenden Protokoll der Standardfall sind, keine Rolle. Durch versteckte Knoten kommt es zu zusätzlichen Kollisionen, die aber akzeptiert werden, da die Beseitigung des Problems erheblichen

Protokollaufwand und Signalisierungsverkehr mit sich bringen würden.

- Optimale Kanalcodierung um eine möglichst robuste Übertragung über die Funkschnittstelle zu ermöglichen
  - Als Kanalcodierung wurde der Scrambling-Algorithmus von IEEE 802.11 gewählt, weil er in Echtzeit während der Übertragung ausgeführt werden kann und das Datenvolumen nicht erhöht. Auf die Verwendung von aufwändigeren Codes, die Redundanz hinzufügen und Fehlerkorrektur erlauben, wurde verzichtet, weil dies die Übertragungsrate herabsetzen würde und die Grenzen der Rechenkapazität der Knoten sprengen würde.
- Bereitstellen einer angemessenen Nutzdatenrate
  - Das Protokoll stellt eine Nutzdatenrate von 15,36–15,93 kBit / s zur Verfügung. Dies ist auf den ersten Blick wenig, sollte aber genügen um Kontexte, die oft nur mit wenigen Bytes repräsentiert werden können, zu übertragen.
- Beschränkung der Übertragungslatenz
  - Das Protokoll garantiert keine obere Schranke für die Übertragungslatenz. Dieses Ziel steht im Konflikt mit der Forderung nach gleichberechtigtem Zugriff, dem hier Vorrang gewährt wurde. Bei gleichberechtigtem Zugriff kommt es bei überlastetem Medium zu häufiger Ablehnung des Medienzugriffs und somit erhöhter Latenz.

4. Implementierung

Der in Kapitel 3 beschriebene Protokollentwurf wurde für den CC1010-Chip von Chipcon[1] unter Verwendung des Open-Source-Compilers *Small Devices C Compiler* (kurz SDCC) implementiert. [sdcc]

4.1 Verwendete Hardware

Als Hardware zum Programmieren und Verbinden der seriellen Ports mit dem PC diente das *CC1010DK Development Kit* (Abschnitt 4.1.2), bestehend aus einem Evaluation Board und mehreren Evaluation Modules.

Um die Sensoren flexibler einsetzen zu können wurde als kleinere Alternative zum Evaluation Module das PartC-Board (Abschnitt 4.1.3) entwickelt.

Zum Testen von zeitlichen Parametern wurde ein 2-kanaliges Speicheroszilloskop von Hewlett-Packard verwendet (Abschnitt 4.1.4).

4.1.1 Übersicht CC1010 Chip

Der CC1010 vereint einen 8-Bit-Mikrocontroller, Speicher, einen Transceiver und viele Peripheriebausteine auf einem Chip. Der Mikrocontroller ist kompatibel zum Intel 8051 und erreicht das 2,5-fache dessen Performance. Der Transceiver ist in seiner Frequenz von 300-1000 MHz frei per Software programmierbar und erreicht Übertragungsraten bis zu 76,8 kBit / s. Auf dem Chip sind 32 kB Flash ROM und 2176 Byte SRAM untergebracht.

Zusätzliche Bausteine auf dem Chip sind ein 3-kanaliger 10 Bit Analog-Digital-Wandler, 4 Timer, 2 serielle Schnittstellen, ein Hardware-Watchdog (siehe Glossar), ein SPI-Interface zur Kommunikation mit anderen integrierten Schaltungen, DES Verschlüsselung in Hardware sowie 26 I/O Leitungen.

[1] http://www.chipcon.com/

4.1.2 CC1010 Development Kit

Das CC1010 Development Kit enthält alle Komponenten, die für die Programmierung und den Betrieb des CC1010 nötig sind. Es ist in zwei Versionen erhältlich: Eine Version für die Übertragung auf einer Frequenz von 433 MHz und eine für die Frequenzbereiche 868 und 915 MHz. Der verbaute CC1010 Chip ist in beiden Versionen der gleiche, mit der verwendeten Übertragungsfrequenz ändern sich jedoch die Parameter der benötigten externen Bauteile.

Das *Evaluation Board (EB)* (Abbildung 4.1 auf der nächsten Seite) wird mit einem parallelen und zwei seriellen Kabeln mit dem PC verbunden. Über die parallele Schnittstelle erfolgt die Programmierung des CC1010. Über die eine serielle Schnittstelle kann der CC1010 Daten mit dem PC austauschen (z. B. über ein Terminalprogramm Debugausgaben machen oder Tastaturbefehle entgegennehmen). Der zweite serielle Port kann entweder ebenfalls zur Datenübertragung genutzt, oder für *In-Circuit-Debugging* verwendet werden (siehe Abschnitt 4.2 auf Seite 48).

Weiterhin enthält das EB Komponenten, die per Software angesteuert werden können. Darunter sind 4 farbige LEDs mit denen man z. B. den aktuellen Zustand der laufenden Anwendung anzeigen kann, 4 Tippschalter, deren jeweiliger Zustand abgefragt werden kann und die somit zum Auslösen von Ereignissen verwendet werden können, zwei Potentiometer, mit denen die Spannung an den analogen Eingangspins des CC1010 verändert werden kann, die durch Sampling mit dem auf dem Chip befindlichen Analog-Digital-Wandler gemessen werden kann sowie ein Reset-Schalter, mit dem der Chip zurückgesetzt werden kann. Die restlichen Bauteile gewährleisten eine geregelte Spannungsversorgung und dienen der Pufferung der Kommunikation über die verfügbaren Schnittstellen.

Weiterhin gehören zum CC1010DK zwei „Evaluation Modules“ (EMs) (Abbildung 4.2 auf der nächsten Seite). Diese bestehen aus einer kleinen Platine die mittels Steckverbindungen auf das EM aufgesteckt werden können. Auf dieser befindet sich der eigentliche CC1010-Chip sowie einige externe, passive Bauteile, die zum Betrieb des Chips notwendig sind (Quarze, Kondensatoren, Widerstände, etc.), sowie ein Temperatursensor und ein Antennenanschluss, auf den die ebenfalls mitgelieferten Antennen geschraubt werden können. Da das Platinenlayout der EMs, das in 4-Ebenen-Technik entworfen ist, im Internet verfügbar war, konnten zu Testzwecken einige weitere Kopien der EMs angefertigt werden.

Auf alle EMs wurden Energieversorgungs-Anschlüsse gelötet, um diese mit den im TecO verwendeten Energieversorgungs-Steckern zu verbinden. Somit konnten die EMs auch unabhängig vom EB betrieben werden.

4.1.3 PartC-Board

Um die Knoten flexibler einsetzen zu können, wurde eine kleinere Platine entwickelt, das PartC-Board. Diese Platine war zu klein, um die Konnektoren anzubringen, die die Verbindung zum EB herstellen. Statt dessen wurde eine kleinere Steckverbindung verwendet, die auch bei den PIC-Particles zum Einsatz kommt. Um das PartC-Board dennoch mit dem EB verbinden zu können,

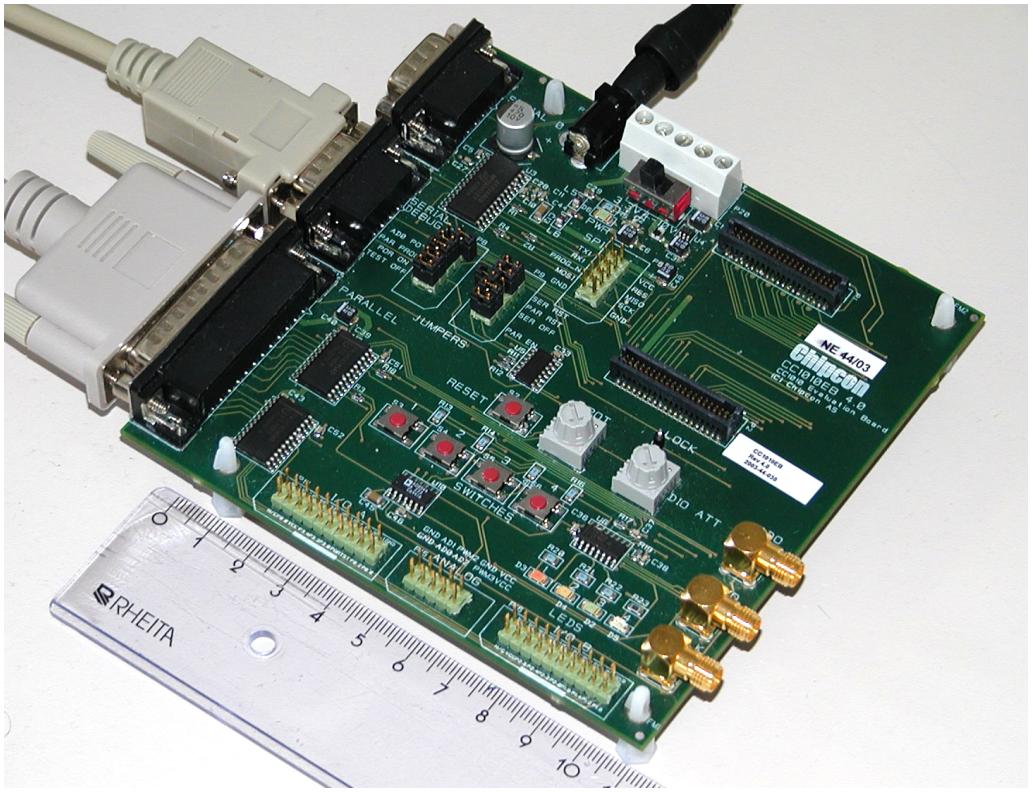

Abbildung 4.1: *Evaluation Board (EB) mit Steckern zum Aufstecken eines EMs*

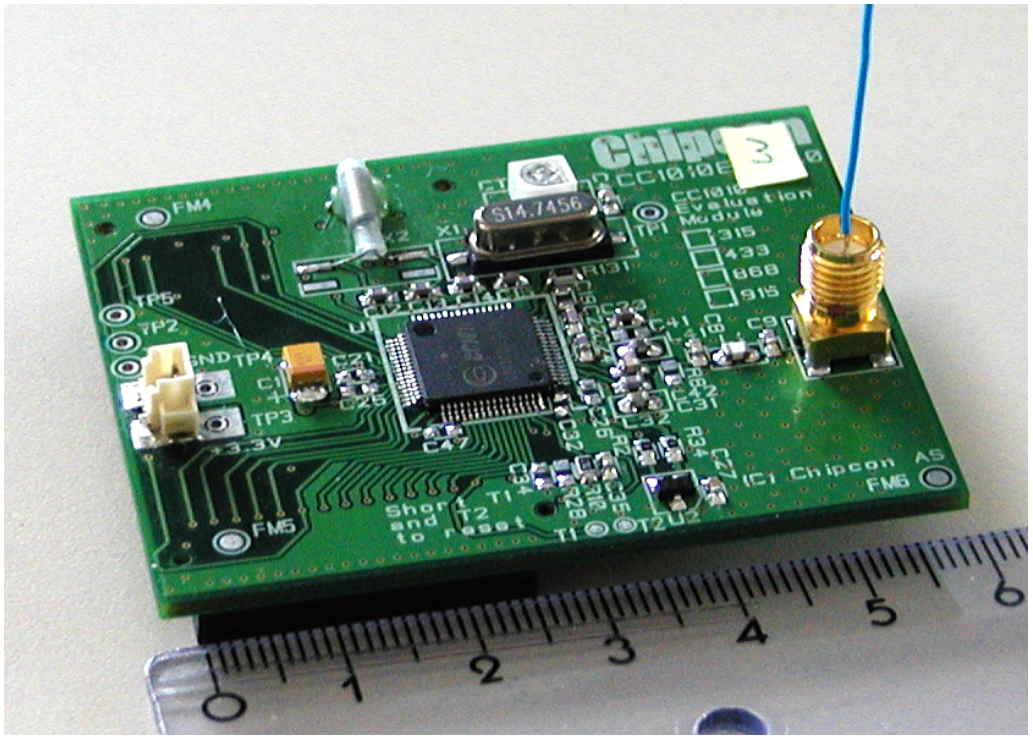

Abbildung 4.2: *Evaluation Module (EM) mit Drahtantenne*

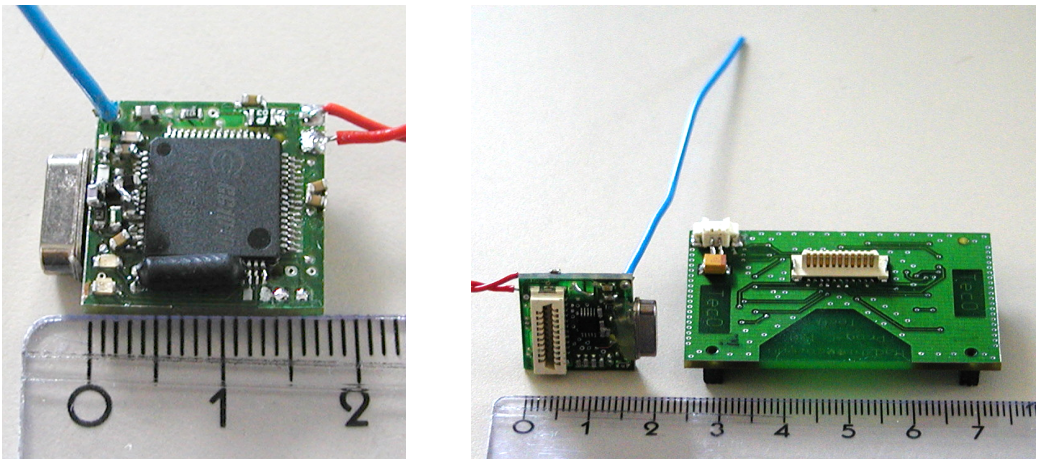

Abbildung 4.3: *PartC Board und EB-Verbindungsplatine*

wurde daher eine Zwischenplatine entworfen. PartC-Board und Zwischenplatine sind in Abbildung 4.3 zu sehen.

4.1.4 Oszilloskop

Zum Überprüfen und Justieren der zeitlichen Synchronität auf verschiedenen Knoten wurde ein 2-Kanal Speicheroszilloskop von Hewlett-Packard verwendet (siehe Abbildung 4.4 auf der nächsten Seite). An bestimmten Stellen im Code wurde ein Pin des CC1010 als digitaler Ausgang konfiguriert und an- bzw. ausgeschaltet. Dieser Pin wurde mit einem Eingang des Oszilloskops verbunden, der gleiche Pin auf dem zweiten EM mit dem zweiten Kanal. Sobald einer der Knoten ein Beacon geschickt und der andere es korrekt empfangen hat, sollten die auf den Ausgangspins der beiden Knoten erzeugten Signale gleich laufen. Die noch verbleibenden Laufzeitunterschiede konnten mit dem Oszilloskop gemessen werden. Es blieb eine Rest-Laufzeitschwankung von typischerweise $\pm 5\mu s$, was für den Betrieb des Protokolls ausreicht.

Die Laufzeitschwankungen kamen vor allem durch unterschiedliche Laufzeit des Descrambling des letzten Bytes zu Stande (siehe Abschnitt 3.10 auf Seite 40).

4.2 Verwendete Software

Als Compiler für den Chipcon CC1010 wurde SDCC (Small Devices C Compiler) verwendet. SDCC ist ein Open-Souce-Compiler der unter der *GNU Public Licence*[2] verfügbar ist. Die Wahl fiel auf diesen Compiler und nicht auf die mitgelieferte *Keil µVision2* Entwickungsumgebung von Keil Inc., weil die Demoversion stark eingeschränkt war und die Einzellizenz zu teuer erschien.

Dem Paket lag eine Demoversion bei, deren Einschränkung darin besteht, dass Größe des übersetzten Programms auf 2KB beschränkt ist. Da schnell klar wurde, dass die zu entwickelnde Software größer als 2KB werden würde, wurde bei Keil Inc. nach einer vergünstigten Hochschullizenz für die Vollversion angefragt. Da die Einzellizenz für die Entwicklungsumgebung mit

[2] http://www.gnu.org/

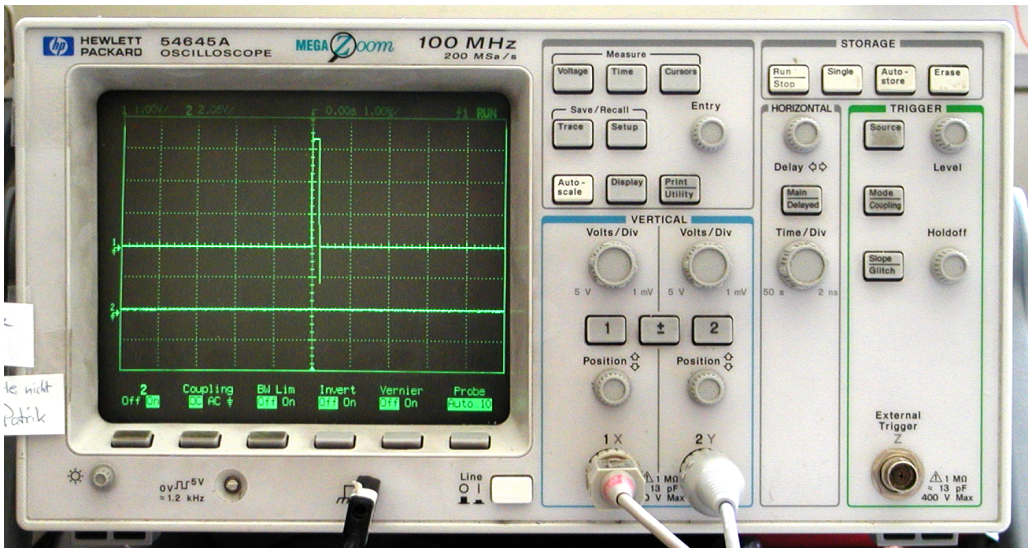

Abbildung 4.4: *2-Kanal Speicheroszilloskop*

€ 2.300 abzüglich 10 % Hochschulrabatt + 16 % Mehrwehrtsteuer (also letztlich mit € 2.401,20) zu Buche geschlagen hätte (Stand Dezember 2003), wurde entschieden, den kostenlos verfügbaren SDCC zu verwenden.

Durch diese Entscheidung wurde auf die Möglichkeit des interaktiven *In-Circuit-Debuggings* verzichtet. Mit dieser Form der Fehlerdiagnose kann man Haltepunkte im Code setzen, den Code Befehl für Befehl ausführen und Variablen- und Speicherinhalte am PC beobachten während der Code auf dem Mikrocontroller läuft. Dies wird durch proprietäre Treiber und Schnittstellen ermöglicht, die derzeit nur von der Keil Software unterstützt werden. Der Verzicht auf interaktives Debugging fällt allerdings nicht stark ins Gewicht. Die Entwicklung des RF-Protokolls ist auf Unterbrechungen (engl. Interrupts) angewiesen und es müssen harte zeitliche Vorgaben eingehalten werden, was das interaktive Debugging erschwert. Kompliziertere Phänomene wie die Synchronisation mehrerer Knoten lassen sich auf diesem Wege ebenfalls nicht analysieren. Hierbei ist die Benutzung eines Oszilloskops unabdingbar (siehe Abschnitt 4.1.4 auf der vorherigen Seite).

SDCC erwies sich als instabil, was die Lauffähigkeit des erzeugten Codes betrifft. Am stabilsten lief der Code, der mit der Version 2.4.0 vom 26. Februar 2004 übersetzt wurde. Die von den Entwicklern des Compilers angeratene Benutzung der neusten Entwicklungsversionen, die täglich automatisch erstellt werden, führte zu Code, der nicht fehlerfrei lief.

Ebenfalls wurden einige nicht erklärbare Fehler entdeckt, die mit Hilfe des Ausschlussprinzips isoliert wurden und deren Lösung unlogisch erscheint. So genügt es z. B. nicht zum Abschalten des Watchdog-Timers (siehe Glossar), das Register das diesen kontrolliert einmal zu beschreiben – der entsprechende Wert muss zweimal zugewiesen werden. Um anderen Benutzern des SDCC eine schnelle Fehlersuche zu ermöglichen, wurde eine Homepage zur dieser Diplomarbeit angelegt, auf der einige Fehler und mögliche Wege zur Umgehung derselben aufgelistet wurden. Sie ist unter der URL http://www.teco.edu/~spiess/ zu finden.

Als integrierte Entwicklungsumgebung wurde jEdit[3] benutzt. Der vollständig in Java implementierte Editor ist leicht zu bedienen, zu konfigurieren und lässt sich vielseitig erweitern. So kann er die Funktionen, Variablen und Defines einer C-Quellcodedatei auflisten. Er verfügt über eine beachtliche Anzahl von Funktionen, unter anderem inkrementelle Suche, Suche mit Regulären Ausdrücken, Auflistung aller Suchergebnisse und Suche in Dateien. Außerdem kann er die verschiedenen Sprachbestandteile vieler Programmiersprachen farblich hervorheben.

Weiterhin wurde ein Build-Tool in der Skriptsprache Perl[4] implementiert, dass nur jene Programmteile neu übersetzt, die sich seit dem letzten Build verändert haben. Ein weiteres Perl-Skript ruft das Flash-Brennprogramm nur dann auf, wenn sich die übersetzte Binärdatei geändert hat. Diese Skripte wurden in jEdit so integriert, dass per Tastendruck übersetzt und programmiert werden konnte.

4.3 Modelle und Konzepte des CC1010

In diesem Kapitel wird das Modell der CC1010 aus Sicht des Programmierers beschrieben. Es setzt sich aus der Abstraktionen der einzelnen Komponenten und deren Zusammenspiel zusammen. Dem Programmierer wird ein übersichtliches *Application Programming Interface (API)* angeboten. In manchen Fällen muss jedoch direkt, unter Umgehung des API auf die so genannten *Special Function Registers (SFRs)* zugegriffen werden. Sämtliche Interaktion der Software mit der Hardware erfolgt über diese SFRs.

4.3.1 Interrupts

Interrupt ist das englische Wort für Unterbrechung und bezeichnet bei Mikorprozessoren Unterbrechnungen des normalen Programmablaufs als Reaktion auf bestimmte Ereignisse. Das auslösende Ereignis wird als Quelle des Interrupts (engl. Interrupt Souce) bezeichnet.

Ein Interrupt kann unterschiedlichste Quellen haben:

- Ablauf eines Timers (siehe Abschnitt 4.3.2 auf der nächsten Seite)
- Empfang eines neuen Bytes an der seriellen Schnittstelle oder durch den Transceiver
- Bereitschaft zum Senden des nächsten Bytes an der seriellen Schnittstelle oder durch den Transceiver
- Externer Interrupt – eine Spannung liegt am Interrupt-Pin des Chips an
- Analog-Digital-Wandler[5] (ADC) – eine Spannung niedriger als ein festgelegter Schwellwert liegt an einem ADC-Pin

[3] http://www.jedit.org/
[4] http://www.perl.com/
[5] engl. Analog Digital Converter, daher die Abkürzung ADC

Die verschiedenen Arten von Interrupts können einzeln aktiviert oder deaktiviert werden (man spricht von Interrupt-Maskierung oder -Masking). Dazu ist ein SFR vorgesehen, dessen Bits die Interrupts repräsentieren und die gesetzt oder gelöscht sein können. Zusätzlich ist es durch ein weiteres Bit möglich alle Interrupts auf einmal zu deaktivieren.

Wenn ein Interrupt auftritt und dieser aktiviert ist, unterbricht der Prozessor den aktuellen Programmablauf und springt der Prozessor zu der zum Interrupt passenden *Interrupt Service Routine (ISR)*. Dies ist eine spezielle Funktion. Für jede Art von Interrupt kann genau eine ISR definiert werden. Eine ISR wird in SDCC wie folgt definiert:

```
void isr_x() interrupt <Nummer des Interrupts> {
    // Code
}
```

Ist die Ausführung der ISR abgeschlossen, setzt der Prozessor die Ausführung des Programms an der Stelle fort, an der es unterbrochen wurde. War der Interrupt zum Zeitpunkt des Auftretens nicht aktiviert, wird er ignoriert – der Programmablauf wird wie gewohnt fortgesetzt.

Zur weiter gehenden Steuerung der einzelnen Interrupts werden so genannte *Flags* (engl. Merker) verwendet. Dies sind Bits in einem dafür vorgesehenen SFR. Beim Auslösen eines Interrupts wird das zugehörige Interrupt-Flag gesetzt. In der ISR muss dieses Bit wieder gelöscht werden, damit ein neuer Interrupt ausgeführt werden kann. Tritt neuer Interrupt auf (siehe Liste oben) solange das Interrupt-Bit nicht gelöscht wurde, so wird dieser ignoriert und nicht ausgeführt. Gleiches gilt für die Situation, dass während der Ausführung der ISR ein neuer Interrupt auftritt.

Die Flags erlauben die Nutzung der Interrupts auch ohne ISRs. Dazu kann in einer Schleife wiederholt der Status eines Interrupt-Flags abgefragt werden. Hier folgendes Codestück ist aus einem Makro zum Senden eines Bytes über die serielle Schnittstelle. Es wird gewartet, bis die Schnittstelle bereit ist, das nächste Byte zu senden. Das Flag für den zugehörigen Interrupt heißt TI_1:

```
while (!TI_1);  // Warten auf Interrupt-Flag
TI_1=0;         // Löschen des Bits
SBUF1=(x);      // Schreiben des nächsten Bytes
                   in den Sendepuffer
```

4.3.2 Timer

Der CC1010 verfügt über 4 Hardware-Timer (engl. für Zeitgeber), die von 0 bis 3 durchnummeriert sind. Jeder Timer kann bei Ablauf einen Interrupt auszulösen.

Timer 0 und Timer 1 haben jeweils zwei 8-Bit SRFs, die wie ein großes 16-Bit-Zählerregister behandelt werden. Diese Register können von der Software gelesen und geschrieben werden. Die übliche Funktionsweise kann folgendermaßen beschrieben werden:

1. Zählerregister mit einem Anfangswert geschrieben, der zur gewünschten Periode führt.
2. In regelmäßigen Zeitabständen wird der Wert des Zählerregisters um 1 erhöht. Die Länge dieser Zeitabschnitte ist definiert durch den Prozessortakt und einen Divisor, der darauf angewendet wird. So kann z. B. festgelegt werden, dass das Zählerregister alle 4 Taktzyklen erhöht wird.
3. Bei Überlauf des Zählerregisters, also dem Übergang von FFFF_{hex} zu 0000_{hex} wird ein Interrupt ausgelöst.
4. Die aufgerufene ISR setzt das Interrupt-Flag zurück, lädt wieder den Anfangswert in das Zählerregister und führt alle Operationen zur Bearbeitung des Interrupts aus.
5. nun geht es weiter mit 2.

Die Timer 2 und 3 funktionieren ähnlich, ihr Zählerregister kann aber weder gelesen noch geschrieben werden. Der Anfangswert wird nach Auslösen des Interrupts automatisch neu geladen, so dass dieser Schritt in der ISR entfallen kann.

Neben der normalen Funktion als Timer können Timer 0/1 auch als Ereigniszähler (Spannungsänderungen an einem Eingangspin) und Timer 2/3 auch als Pulse Width Modulator (PWM) eingesetzt werden. Davon wird im vorliegenden Projekt aber kein Gebrauch gemacht.

Zur Konfiguration der Timer gibt es folgende API-Funktionen:

```
ulong halConfigTimer01(byte options, ulong period, word clkFreq,
    xdata word* modulo);
ulong halConfigTimer23(byte options, ulong period, word clkFreq);
```

In der Referenzimplementierung wird Timer0 als der Slot-Timer eingesetzt, Timer1 wird von der API im Hintergrund verwendet, um Ausgaben über den seriellen Port zu ermöglichen und Timer 2 wird vom Stack verwendet, um Timeouts zur realisieren. Der Anwendung steht somit Timer 3 in vollem Umpfang zur Verfügung.

4.3.3 Transceiver

Der auf dem Chip untergebrachte Transceiver nutzt Frequenzmodulation (FSK)[6] zur Übertragung. Per Software kann die Mittenfrequenz f_{center} und die Kanaltrennung f_{sep} eingestellt werden. Der Transceiver verwendet zum Senden und Empfangen eines Bits eine der Frequenzen $f_{center} \pm \frac{1}{2} \cdot f_{sep}$.

Die Sende- und Empfangsfrequenzen sind über einen weiten Bereich von 300 bis 1000 MHz per Software programmierbar. Bei stark unterschiedlichen Frequenzen ändern sich jedoch die Parameter der externen Komponenten, die für den Betrieb des Transceivers notwendig sind.

[6] engl. Frequency Shift Keying

Bevor der Transceiver zum Senden oder Empfangen benutzt werden kann, muss er kalibriert werden. Dazu ruft man eine Kalibrierungsfunktion auf, die eine Datenstruktur zurück gibt, in der die Ergebnisse der Kalibrierung stehen. Der Inhalt dieser Datenstruktur wird nachfolgend bei jedem Sende- und Empfangsvorgang zur Einstellung optimaler Parameter verwendet.

Um ein Byte zu senden, muss dieses in ein spezielles RF-Pufferregister geschrieben werden. Während der Übertragung wird das zu sendende Byte jeweils aus dem Puffer kopiert und er kann das nächste Byte aufnehmen. Die Bereitschaft, das nächste Byte in den Puffer aufzunehmen, zeigt das System mit dem RF-Interrupt an. Wenn man mehrmals das gleiche Byte hintereinander senden möchte, muss man das Pufferregister nur einmal beschreiben.

Das Senden eines Byte-Stromes wird von der API wie folgt abstrahiert:

- Transceiver in Sendebereitschaft versetzen, Warten auf PLL-Lock[7]
- Schreiben des Präambelbytes, das aus alternierenden Bits bestehen muss, in das RF-Pufferregister.
- n mal auf den RF-Interrupt warten, n ist hierbei die Länge der zu sendenden Präambel.
- Das Synchronisationsbyte in den Puffer laden, mit dessen Hilfe erkennt der Empfänger den Anfang der Übertragung.
- Auf den nächsten RF-Interrupt warten.
- Schreiben des ersten Bytes der Nachricht in den Puffer.
- Wiederholen der zwei voraus gehenden Schritte, mit allen folgenden Bytes der Nachricht.
- Noch zwei RF-Interrupts abwarten, damit der Wert im Pufferregister noch übertragen wird.
- Herunterfahren des Transceivers.

Beim Empfang gibt der RF-Interrupt an, dass das erste oder nächste Byte empfangen wurde. Die Schritte zum Empfangen von Bytes sind folgende:

- Transceiver in Empfangsbereitschaft versetzen, Warten auf PLL-Lock.
- Start des Mittelungsfilters. Dieser misst den Durchschnitts-Signalpegel[8]
- Starten des Empfangs. Der Receiver versucht eine Präambel zu detektieren und falls dies erfolgreich war, das Synchronisationsbyte zu empfangen.
- Warten auf den RF-Interrupt, der anzeigt, dass das Synchronisationsbyte empfangen wurde.

[7] PLL: Abk. für Phase Lock Loop, eine Schaltung, die einen Taktsignal erzeugt, das Phasengleich zu einem Referenztakt ist. PPL-Lock bedeutet, dass der Zustand der Phasengleichheit (nach einer gewissen Einschwingzeit) erreicht ist.

[8] Der durchschnittliche Pegel wird verwendet um 0- und 1-Symbole unterscheiden zu können. Daher ist es wichtig, das Präambel und Synchronisationsbyte gleichstromfrei sind, also gleich viele 0en und 1en haben.

- Deaktivieren des Mittelungsfilters.
- Warten auf den RF-Interrupt, der anzeigt, dass das erste Byte empfangen wurde.
- Lesen des Bytes aus den Puffer.
- Wiederholen der zwei voraus gehenden Schritte, bis die vollständige Nachricht empfangen wurde.
- Herunterfahren des Transceivers.

Alternativ zum gerade beschriebenen *Bytemodus*, in dem der Transceiver immer ganze Bytes sendet bzw. empfängt, kann er auch im *Bitmodus* betrieben werden. Dann wird jeweils nur das LSB aus dem Pufferregister zum Senden ausgelesen bzw. das zuletzt empfangene Bit in das Pufferregister geschrieben.

4.3.4 AD-Wandler und Feldstärkemessung

Auf dem CC1010 ist ein Analog-Digital-Wandler (ADC) untergebracht, der die Spannungen an 3 analogen Pins mit einer Auflösung von 10 Bit digitalisieren kann. Dabei entspricht ein Wert von 0 einer Spannung von 0 Volt und ein Wert von 1023 einer Spannung von 1.25 V. Alternativ kann auch eine externe Referenzspannung für den Maximalwert verwendet werden, die entweder auch 1.25 V oder VDD[9] betragen muss.

Der ADC kann auch verwendet werden um die Stärke des Signal zu messen, das der Transceiver gerade empfängt. Zu diesem Zweck verfügt der Transceiver-Teil des Chips über einen RSSI[10]-Ausgang, wenn sich der Transceiver in Empfangsbereitschaft befindet. An diesem liegt die Referenzspannung an, wenn kein Signal gemessen wird und eine Spannung von 0 Volt, wenn ein sehr starkes Signal empfangen wird.

Dieser Ausgang des Transceivers kann nun intern mit einem der Eingänge des ADC elektrisch verbunden werden, und mit dem ADC digitalisiert werden. Die ermittelten Werte sind logarithmisch zu interpretieren, d. h. eine Differenz von $\pm n$ steht für eine Änderung der Signalstärke um $n \cdot k$ mit einem Gewichtungsfaktor k.

4.3.5 Systemtakte und Arbeitsmodi des Mikrocontrollers

Auf dem EM stehen zwei Systemtakte zur Verfügung: einmal 14,7456 MHz und 32,768 kHz. Der Prozessorkern des CC1010 kann mit einer der beiden Taktraten betrieben werden. Läuft er mit 32,768 kHz ist – neben der stark verminderten Rechenleistung – die Benutzung des Transceivers nicht möglich. Dafür verringert sich die typische Leistungsaufnahme von 14,8 mA auf 1,3 mA.

Weiterhin kann man den Prozessorkern in einen von drei Arbeitsmodi versetzen. Der Standardmodus ist *Active*. Hier holt sich der Prozessor Anweisung

[9] in diesem Fall die Versorgungsspannung

[10] engl. für Received Signal Strength Indication, Angabe der Stärke des empfangenen Signals

für Anweisung aus dem Flash-Speicher und führt sie aus. In den Modi *Idle* und *Power Down* ist die Ausführung angehalten. Im Idle-Modus wird die Verarbeitung wieder fortgesetzt, wenn ein Interrupt auftritt oder das System zurückgesetzt wird. Ist der Knoten im Power-Down-Modus, kann er nur noch durch einen Reset aufgeweckt werden.

Ein Reset löscht alle Register und beginnt die Ausführung des Programms ab Adresse 0, der RAM-Speicher bleibt vom Reset aber unberührt.

Tabelle 4.1 auf der nächsten Seite zeigt detailliert, wie viel Strom der CC1010 in den verschiedenen Betriebsmodi aufnimmt.

Modus	Prozessor	Peripherie	Stromaufnahme	beendet durch	Verwendung
Active	14 MHz	14 MHz	14,8 mA	Ändern eines SFRs durch Software	Normalbetrieb
	32 kHz	32 kHz	1,3 mA	Ändern eines SFRs durch Software	
Idle	Angehalten	14 MHz	12,8 mA	Interrupt, Reset, Aus-/Einschalten	Anwendung wartet auf nächsten Slot
		32 kHz	29,4 μA	Interrupt, Reset, Aus-/Einschalten	Tiefschlaf: Veranlasst durch den Stack wenn Knoten alleine ist oder durch die Anwendung
Power Down	Angehalten	32 kHz	200 μA	ADC-Wert über Schwellwert, Reset, Aus-/Einschalten	
		Angehalten	0,2 μA	Reset, Aus-/Einschalten	

Tabelle 4.1: *Stromaufnahme in den verschiendenen Betriebsmodi, Quelle CC1010 Data Sheet V1.2*

4.4 Anwendung des Modells bei der Implementierung

Dieser Abschnitt erläutert die Interaktion der verschiedenen Komponenten des vorgestellten Hardwaremodells bei der Implementierung des Protokollentwurfs.

4.4.1 Synchronisation und Ausführungsebenen

Die in dieser Diplomarbeit gestellte Aufgabe wurde ohne Zuhilfenahme eines Betriebssystems implementiert. Eine Alternative wäre evtl. TinyOS[11] – ein Open-Source-Betriebssystem für drahtlose Sensornetzwerke – gewesen. Aus Zeitgründen wurde auf die Einarbeitung in ein solches Betriebssystem und dessen Nutzung verzichtet.

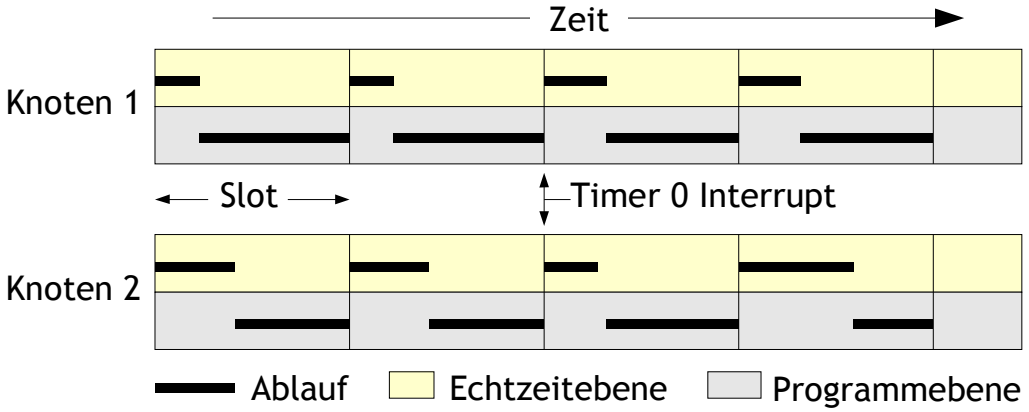

Abbildung 4.5: *Programm- und Echtzeitebene*

Um die Synchronisation zwischen den Knoten zu ermöglichen wurde ein einfaches Zeitschema verwendet:

- Timer0 wird so initialisiert, dass er alle 30 μs eine ISR aufruft.
- Alle sychnronen Vorgänge werden innerhalb der ISR ausgeführt. Die Zeit während der der Prozessor die ISR ausführt, wird die *Echtzeitebene* genannt.
- Alle nicht zeitkritischen Vorgänge werden in der restlichen Zeit ausgeführt. Diese stellt die *Programm-* bzw. *Echtzeitebene* dar.
- Sämtliche Funktionen zum Senden und Empfangen von Daten werden asynchron mit nicht blockierenden Funktionen realisiert.
  - Zum Senden füllt das Programm einen Puffer und teilt dem System mit, dass der Inhalt des Puffers versendet werden muss. Danach kann der Status der Sendung abgefragt werden (noch nicht abgeschickt, erfolgreich verschickt, Fehler).

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

– Während der Ausführung der Echtzeitebene empfangene Pakete werden in einen Puffer geschrieben. Die Programmebene kann den Status des Empfangspuffers nachprüfen (neue Daten angekommen, ...).

4.4.2 Rendezvous

Um synchron zu erledigende Aufgaben parallel auf verschiedenen Knoten durchzuführen (insbesondere Senden und Empfangen) genügt es nicht, synchrone Slotgrenzen auf allen Knoten zu haben. Auch innerhalb eines Slots, müssen diese Aufgaben an genau definierten Zeitpunkten beginnen.

Dafür werden so genannte Rendezvous-Punkte verwendet. Ein Rendezvous-Punkt ist ein Timerwert. Es wird eine Funktion (bzw. aus Effizienzgründen ein Makro) zur Verfügung gestellt, das aktiv auf einen Timerwert wartet, d. h. in einer Endlosschleife den Timerwert so lange abfragt, bis er größer oder gleich dem zu erreichenden ist. Diese Rendezvous-Anweisung muss vor den gleichzeitig auszuführenden Befehlen stehen. Gibt es Befehle oder Unterprogrammaufrufe, die eine unbestimmt Zeit dauern können, muss danach wieder ein Rendezvous eingefügt werden.

Ein Beispiel für den Einsatz von Rendezvous ist die Arbitrierung. Da das Hochfahren der Transceiver eine nicht vorhersehbare, zufällige Zeit dauert, müssen die nächsten Schritte genau zur gleichen Zeit erfolgen: Das Senden des Signals oder die Messung der Feldstärke muss genau synchronisiert werden. Dazu wird jeweils ein Rendezvous verwendet.

Zur Feststellung eines geeigneten Timerwertes wurde das Rendezvous zunächst durch Befehle ersetzt, die den aktuellen Timerwert über den seriellen Wert ausgeben. Von diesen Timerwerten wurde dann der maximale Wert für das Rendezvous festgelegt.

4.4.3 Arbitrierungs-Slice und RSSI

In einem Arbitrierungs-Slice wird entweder ein Signal ins Medium gesendet, oder es wird versucht zu ermitteln, ob mindestens ein anderer Knoten ein Signal sendet oder nicht. Während ersteres kein Problem darstellt (man sendet einfach ein paar Bytes beliebige Daten ins Medium) erweist sich letzteres als kompliziert.

Die herkömmliche Methoden des Receivers eine ankommende Übertragung zu erkennen, nämlich Präambel-Detektion, kann hier nicht angewendet werden, da viele Knoten gleichzeitig senden. Wenn diese nun mehrere Knoten gleichzeitig eine Präambel senden, kommt es zu Überlagerungen des Signals. Die erreichbare Synchronität liegt bei $\pm 10\mu$s. Bei einer Übertragungszeit von ca. 13 μs für ein Bit reicht diese Genauigkeit nicht aus um in der Summe ein korrekt überlagertes Präambel-Signal zu erzeugen.

Statt Präambeln zu senden und zu empfangen werden einfach beliebige Daten gesendet. Bei der Detektion wird statt auf eine Präambel zu warten, der Wert des hereinkommenden Signals gemessen. Dazu wird wie in Abschnitt 4.3.4

auf Seite 54 beschrieben zu geeignetem Zeitpunkt im Arbitrierungs-Slice die hereinkommende Feldstärke gemessen.

Die gemessenen RSSI-Werte werden mit einem geeigneten Schwellwert verglichen. Liegen sie unter dem Schwellwert wird angenommen, dass ein Signal auf dem Medium zu hören ist – liegen sie darüber, geht der Knoten davon aus, dass kein anderer ein Signal sendet.

Die Ermittlung eines geeigneten Schwellwertes erwies sich schwierig. So ist das Signal eines weit entfernten Knotens kaum vom Hintergrundrauschen zu unterscheiden, während ein Knoten in unmittelbarer Nähe ein erhöhtes Störspektrum erzeugt, auch wenn er nur ins Medium hört. Um zu einem möglichst guten Kompromiss zu kommen, wurde ein spezieller Testmodus für die Arbitrierung implementiert, in der ein Senderknoten immer ein bestimmtes Byte als Arbitrierungsbyte verwendet, und ein Messknoten die gemessenen RSSI-Werte über die serielle Schnittstelle an einen PC sendet.

Auf die ermittelten Daten wurden dann alle in Frage kommenden Schwellwert angewandt und derjenige ausgewählt, der zu den wenigsten Fehlern führte. Abschnitt A.4 auf Seite 79 zeigt die genaue Konfiguration der Software, um die Messungen durchzuführen.

5. Evaluation

Dieses Kapitel beschäftigt sich mit der Bewertung der theoretischen Möglichkeiten des entworfenen Protokolls und der messbaren Ergebnissen seiner Implementierung.

5.1 Theoretische Leistungsfähigkeit des Protokolls

Die maximale Übertragunsrate der LL-Schicht des vorliegenden Protokolls beträgt 2133 kBit/s. Dieser Wert errechnet sich aus der Länge eines Slots (30 ms) aus der sich die Anzahl der Slots pro Sekunde ergibt ($33^1/_3$) und der Anzahl der übertragenen Bytes pro Slot (64).

5.2 Simulation der verteilten Synchronisation

5.2.1 Beschreibung der Simulation

Zur Messung der theoretischen Leistungsfähigkeit der verteilten Synchronisation wurden zwei Typen von Netzen simuliert:

Ein einfaches 9-hop Netzwerk mit 10 Knoten simuliert, wie es in Abbildung 5.1 zu sehen ist

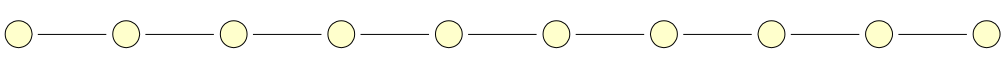

Abbildung 5.1: *9-hop Netzwerk*

und eines, in dem zwischen allen Knoten Kommunikation möglich ist, wie in Abbildung 5.2 auf der nächsten Seite gezeigt.

Zunächst wurde den einzelnen Knoten ein individueller Gangunterschied zugewiesen. Dazu wurde am Oszilloskop ein typischer Gangunterschied ermittelt. Der Gangunterschied ist die Differenz der Slotlänge eines Knotens zu

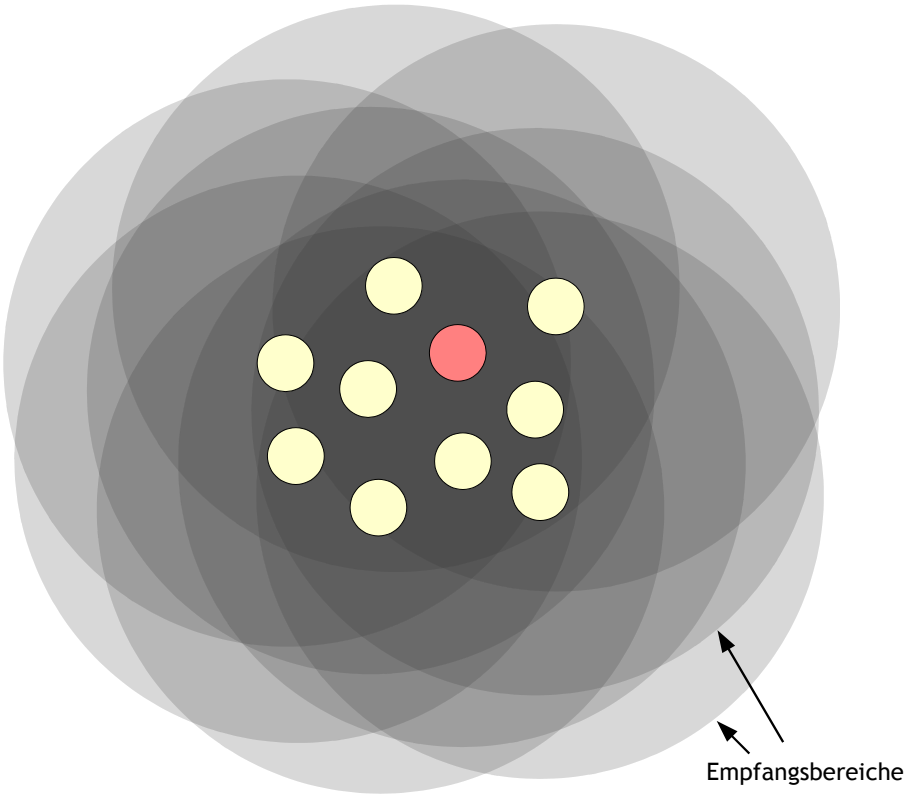

Abbildung 5.2: *Netzwerk in dem jeder Knoten im Empfangsbereich jedes anderen Knotens ist*

einer gedachten, idealen Slotlänge, die der Mittelwert aller real auftretenden Slotlängen ist. Abbildung 5.3 zeigt ein (übertriebenes) Beispiel für den Gangunterschied zwischen zwei Knoten. In der Realität beträgt der Gangunterschied für einen Slot von 30 ms nur $\pm$ 1,5 μs.

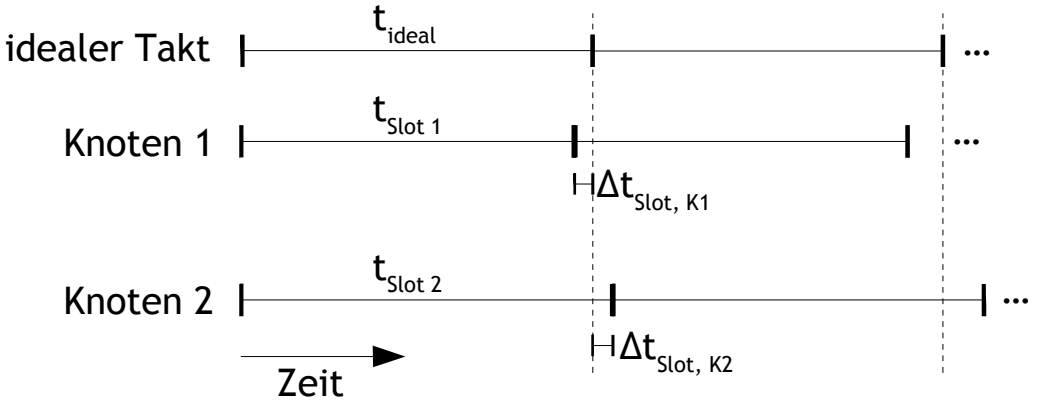

Abbildung 5.3: *Gangunterschied der Timer zwischen zwei Knoten*

Zu unterschiedlich langen Slots kommt es, weil der Betriebstakt des CC1010 Chips mit einem Quarz erzeugt wird und die unterschiedlichen Quarze auf den verschiedenen Knoten kleine Fertigungstoleranzen haben, was wiederrum zu leichten Unterschieden in der Frequenz des jeweiligen Taktes führt.

Ein Knoten wird in der Simulation als Tupel aus folgenden Eigenschaften dargestellt: Der Beginn des nächsten Slots, der Gangunterschied zur idealen Slotlänge und die aktuelle Slotnummer.

Die Simulation schreitet Slot für Slot voran. Am Anfang jedes Slots wird der individuelle Gangunterschied angewandt. Dann wird entscheiden, ob ein Knoten ein Beacon senden möchte oder nicht, die Arbitrierung wird durchgeführt und die Beacons versendet. Dies geschieht indem alle Knoten im Empfangsbereich des Beacons dessen Zeitinformationen auswerten und ihre Timer stellen. Dies wird über simulierte 50 Stunden bzw. 6 Millionen Slots fortgeführt.

Der Empfang der Beacons wurde in zwei Varianten simuliert: in einem Fall stellte der empfangende Knoten den Timer nur, wenn der empfangene Timerwert kleiner ist als der gerade aktuelle vor dem Setzen des Timers. In der zweiten Variante wurden alle Timerwerte akzeptiert und auf den Timer angewandt.

	nur kleinere Timerwerte werden akzeptiert	alle Timerwerte werden akzeptiert
jeder Knoten hört alle anderen	A = 44,0580 μs B = 13,2053 μs	A = 26,8144 μs B = 14,3409 μs
jeder Knoten hört linken und rechten Nachbarn	A = 23,8165 μs B = 117,6294 μs C = 37,8863 μs D = 122,9208 μs	A = 46,4915 μs B = 352,2928 μs C = 59,7456 μs D = 381,7222 μs

A: durchschnittliche Abweichung zwischen Nachbarn
B: maximale Abweichung zwischen Nachbarn
C: durchschnittliche Abweichung im Gesamtnetz
D: maximale Abweichung im Gesamtnetz

Tabelle 5.1: *Simulation der verteilten Sychronisation*

Die Ergebnisse der Simulation sind in Tabelle 5.1 dargestellt. In allen Simulationen wurden die durchschnittliche und maximale Abweichung (C und D) zwischen Nachbarn im Netz ermittelt. Dabei wurden wie folgt vorgegangen: In jedem Slot wird die maximale Zeitdifferenz zwischen zwei Knoten im Netz ermittelt und über die Folge dieser Werte wird einerseits der Mittelwert (C) und andererseits der Maximalwert (D) festgehalten.

Bei der Simulation des 9-Hop Netzwerkes wurden zusätzlich die durchschnittliche und maximale Abweichung (A und B) zwischen benachbarten Knoten berechnet (diese Angaben sind für das voll vermaschte Netz identisch mit C und D, weil jeder Knoten Nachbar eines jeden anderen ist).

5.2.2 Interpretation der Ergebnisse

Die Ergebnisse zeigen, dass sich die Abweichungen in einem Netz mit mehreren Hops verringern, wenn ein Knoten nur Beacons mit Timerwerten akzeptiert, die kleiner sind als sein eigener Timerwert. Die führt dazu, dass

sich die Abweichungen nur „in eine zeitliche Richtung“ fortpflanzen, d. h. in diesem Fall, dass sich die langsameren Uhren durchsetzen.

Allerdings zeigen die Ergebnisse auch, dass sich die Werte für ein voll vermaschtes Netz (was wohl den häufigsten Fall darstellt) signifikant verschlechtern, wenn man diese Einschränkung einführt. Das resultiert aus der Tatsache, dass im Durchschnitt nun nur noch die Hälfte der Beacons verwendet und die andere verworfen wird.

Weiterhin zeigte eine testweise Implementierung, dass durch den Vergleich des ankommenden Timerwertes mit dem lokalen zusätzliche Ungenauigkeit hinzugefügt wird, so dass die gemessene Abweichung höher ist, als hier simulativ ermittelt.

Schlussendlich wurden beschlossen, alle ankommenden Beacons anzuwenden und den erhöhten Gangunterschied in n-Hop-Netzen hinzunehmen.

5.3 Simulation des Schlafmodus

Beim Schlafmodus gibt es eine Situation, die schwierig aufzulösen ist: Wenn zwei Knoten aufeinander treffen, die sich im Schlafmodus befinden. Die Frage ist hierbei: wie lange dauert es, durchschnittlich, bis sich die beiden Knoten gegenseitig erkennen und in den normalen, synchronisierten Betriebszustand übergehen?

Um diese Frage zu beantworten genügt es, die Situation mit zwei Knoten zu untersuchen, denn diese stellt den ungünstigsten anzunehmenden Fall dar. Würden sich drei oder mehr Knoten im Schlafmodus aufeinander zu bewegen, wäre die Zeit, nach der sie sich finden geringer, da es häufiger zu Phasen des Normalbetriebs und der Beaconsuche kommt. Zunächst würden zwei Knoten sich finden und in den normalen, synchronisierten Zustand übergehen. Alle weiteren Knoten würden sich unmittelbar in ihrer nächsten Beaconsuche diesem Netz anschließen.

Damit sich zwei Knoten im Schlafmodus finden muss sich der eine im Zustand der Beaconsuche befinden und der andere im Normalbetrieb – die überlappende Zeit muss dabei pessimistisch gesehen größer als eine habe Sekunde sein.

Um eine realistische Zeit zu ermitteln, die für die Auflösung eines derartigen Konfliktes benötigt wird, wurde eine ereignisdiskrete Simulation erstellt, d. h. die Zeit schreitet von Zustandsübergang zu Zustandsübergang voran. Die Knoten wechseln dabei zyklisch zwischen den Zuständen Normalbetrieb, Energiesparzustand und Beaconsuche. Die Simulation schreitet so lange voran, bis die Bedingung für die gegenseitige Erkennung gegeben ist.

Zunächst wurde mit Hilfe der Simulation ein geeignetes Verhältnis zwischen Dauer der Beaconsuche und Normalbetrieb gesucht. Tabelle 5.2 auf der nächsten Seite zeigt dabei, dass der günstigste Auflösungszeit wohl dann auftritt, wenn die Dauer für beide Zustände gleich gewählt werden.

Danach wurde untersucht, wie sich die Bemessung der Ober- und Untergrenzen des Schlafintervalls auf die Auflösungszeit auswirkt. Tabelle 5.3 auf der nächsten Seite zeigt, dass die Auflösungszeit nur von der durchschnittlichen

t_{Suche} [s]	t_{Normal} [s]	$\emptyset\, t_{Auflösung}$ [min]
1,5	4,5	3,3
2,0	4,0	2,6
2,5	3,5	2,1
3,0	3,0	2,1
3,5	2,5	2,6
4,0	2,0	3,4
4,5	1,5	5,0

Tabelle 5.2: *Auflösungszeiten bei zwei Knoten im Schlafmodus in Abhängigkeit der Dauer der Beaconsuche und des Normalbetriebs bei einer Untergrenze des Schlafintervalls von 10 s und einer Obergrenze von 26 s.*

$min\, t_{\text{Schlaf}}$ [s]	$max\, t_{\text{Schlaf}}$ [s]	$\emptyset\, t_{AuflösungA}$ [min]	$\emptyset\, t_{AuflösungB}$ [min]
18	18	∞	∞
17	19	11,4	13,1
16	20	4,0	5,1
15	21	2,8	3,9
10	26	2,1	3,3
10	42	3,6	5,7
10	64	6,3	10,3
16	32	3,4	5,3
32	64	10,3	16,7

Tabelle 5.3: *Auflösungszeiten bei zwei Knoten im Schlafmodus in Abhängigkeit der Unter- und Obergrenzen des Schlafintervalls t_{Schlaf}. Im Falle von $t_{AuflösungA}$ bei jeweils 3 s Beaconsuche und Normalbetrieb. Im Falle von $t_{AuflösungB}$ bei 1,5 s Beaconsuche und 4,5 s Normalbetrieb.*

Schlafdauer abhängt (also dem Mittel zwischen Minimal- und Maximalwert), vorausgesetzt, minimale und maximale Dauer sind nicht zu nahe beieinander gewählt. Eine breitere Streuung der Schlafdauer führt demnach zu einer günstigeren Auflösungszeit, weil sich die Zustandsabfolge auf den beiden Knoten stärker unterscheidet.

Für die Referenzimplementierung wurde daher jeweils 3 Sekunden (100 Slots) als Dauer für Beaconsuche und Normalbetrieb gewählt sowie eine zufällige Dauer des Schlafintervalls zwischen 10 und 26 Sekunden, da dies einen guten Kompromiss zwischen Auflösungszeit und Energieeinsparung darstellt. Zwei verwaiste Knoten finden sich in diesem Fall theoretisch nach durchschnittlich 2,8 Minuten.

5.4 Messungen mit den PartC-Boards

Zur Bewertung der Implementierung wurden Versuche mit den CC1010 Evaluation Modules (siehe Abbildung 4.2 auf Seite 47) vorgenommen. Die PartC-Boards waren zum Ende dieser Arbeit noch in der Entwicklung, so dass nur zwei zur Verfügung standen. Zusätzlich gab es Probleme mit der Interoperabilität zwischen den großen EMs, von denen 5 zur Verfügung standen, und den PartC-Boards. Also wurden die Tests mit 4-5 EMs durchgeführt.

Das Szenario war wie folgt aufgebaut: 4-5 Knoten im Abstand von jeweils 30 cm bis 1 m senden alle 3 Sekunden ein Datenpaket. Eines der EMs ist mit dem Evaluation Board verbunden und sendet Messdaten über die serielle Schnittstelle an den PC. Diese werden mit Hilfe eines Terminalprogramms in eine Datei geschrieben und später ausgewertet.

Mit dem oben beschriebenen Szenario wurden vier Messungen durchgeführt, die jeweils ca. 20 Minuten dauerten. Zwei der Messungen wurden mit Master-Slave-Synchronisation durchgeführt und zwei mit verteilter Synchronisation. Es wurde berechnet, welche Anzahl von Paketen zu erwarten ist. Diese wurde mit der Anzahl der tatsächlich empfangenen bzw. gesendeten Pakete verglichen und der fehlende Anteil ermittelt. Die Ergebnisse sind in Tabelle 5.4 auf Seite 68 aufgeführt und in 5.4 grafisch dargestellt.

Bei Master-Slave-Synchronisation wurde der Messknoten als Slave gewählt, um den Anteil der verloren gegangenen Beacons einschätzen zu können. Da ein Slave keine Beacons versendet kann dieser Wert auch nicht angegeben werden.

Die Ergebnisse zeigen, dass es noch viel Optimierungsbedarf für die vorläufige Implementierung des Protokolls gibt. Die hohen Paketverluste bei eingeschalteter Verkehrsanzeige machen deutlich, dass das Konzept der Arbitrierungs-Slices noch verbessert werden kann. So kann es z. B. sein, dass zum Zeitpunkt der Messung der Schwellwert der Arbitrierung nicht richtig kalibriert war (siehe Abschnitt A.4 auf Seite 79). An der Hardware wurden Veränderungen vorgenommen, die wohl Software seitig noch nicht berücksichtigt wurden. Der hohe Aufwand für die Kalibrierung der Arbitrierung lohnt sich jedoch erst, wenn die endgültige Version der Hardware zur Verfügung steht.

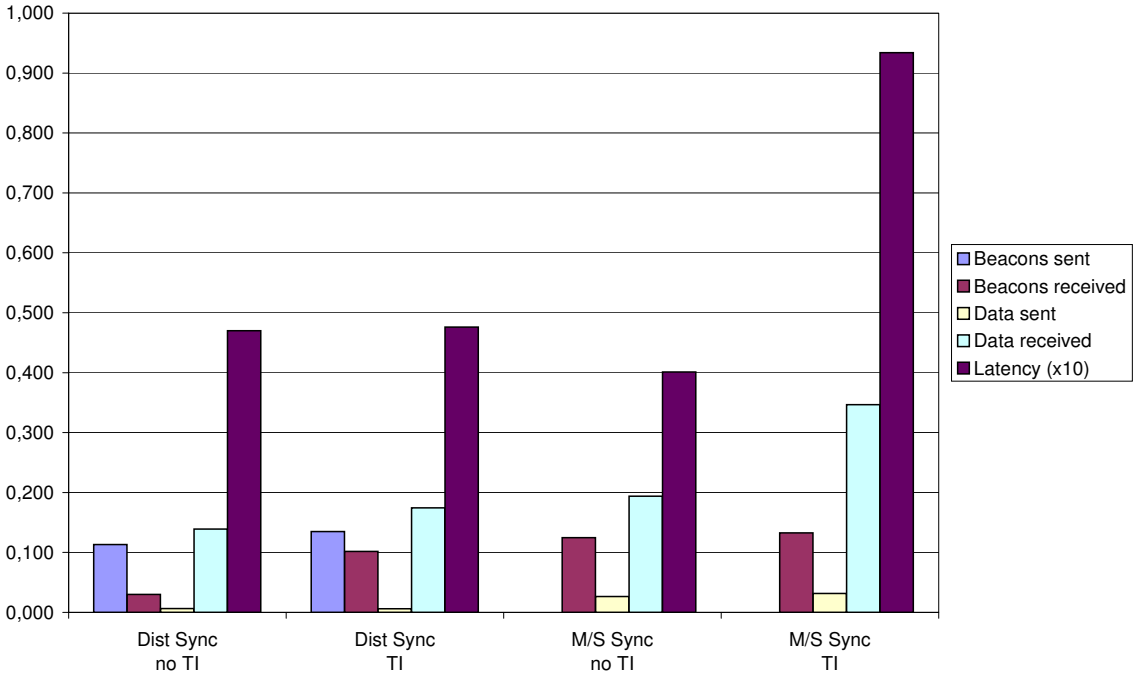

Abbildung 5.4: *Grafische Darstellung der Messergebnisse*

Sync.-Verf.	**Verkehrsanz.**	**Beacons ges.**	**Beacons empf.**	**Daten ges.**	**Daten empf.**	**Latenz**
verteilt	inaktiv	11,3 %	3,0 %	0,6 %	13,9 %	4,70
verteilt	aktiv	13,5 %	10,2 %	0,6 %	17,4 %	4,76
Master-Slave	inaktiv	n/a	12,5 %	2,7 %	19,4 %	4,01
Master-Slave	aktiv	n/a	18,5 %	7,0 %	26,2 %	10,9
		Anteil der Beacons, die weniger gesendet werden konnten als vorhergesagt	Anteil der Beacons, die weniger empfangen wurden als vorhergesagt	Anteil der Datenpakete, die weniger gesendet werden konnten als vorhergesagt	Anteil der Datenpakete, die weniger empfangen wurden als vorhergesagt	Durchschnittliche Latenz von der Beauftragung des senden eines Datenpakete bis zu dessen Sendung

Tabelle 5.4: *Ergebnisse der Messungen mit CC1010 EMs*

6. Zusammenfassung und Ausblick

Im Rahmen dieser Diplomarbeit wurde das Kommunikationsprotokoll der am TecO entwickelten Particle-Computer auf eine neue Hardware portiert. Durch die Portierung können nun wesentlich preiswertere Knoten hergestellt werden, da der verwendete CC1010 von Chipcon Mikrocontroller und Funkeinheit auf einem Chip vereint und nur auf wenige passive, externe Bauteile angewiesen ist.

Die Gegebenheiten der neuen Hardware erforderten eine komplette Neuentwicklung der Bitübertragungs- und der Medienzugriffsschicht, die unter besonderer Berücksichtigung eines niedrigen Energieverbrauchs durchgeführt wurde. Das Protokoll etabliert eine synchrone Zeitbasis auf allen Knoten, so dass die Transceiver der Knoten die meiste Zeit deaktiviert bleiben können. Es wurden zwei Methoden zur Synchronisation vorgestellt.

Das Protokoll ermöglicht einen effizienten Zugriff auf das gemeinsam genutzte Medium, den Funkkanal. Die hierfür verwendete so genannte Arbitrierung, ist ein mehrstufiger Prozess, der ähnlich funktioniert wie das beim CAN-Bus verwendete Zugriffsverfahren.

Zur weiteren Energieeinsparung wurde das Verfahren der Verkehrsanzeige angewendet, das den Energieverbrauch dann senkt, wenn nur wenige Daten gesendet werden – was bei Sensornetzwerken der Regelfall ist. Es erhöht die Latenz der Übertragung, die in Sensornetzwerken eine unkritische Anforderung darstellt.

Knoten, die zeitweilig mit keinem anderen Knoten kommunizieren können, weil sie sich räumlich entfernt haben, oder weil sich die Bedingungen des Funkkanals verschlechtert haben, erkennen diesen Umstand und nutzen dieses Wissen um noch mehr Energie zu sparen. Zusätzlich kann die Anwendung ihre eigene Ausführung Energie sparend unterbrechen, bis die nächste Funkkommunikation stattgefunden hat.

Zuletzt wurde der Verwürfelungs-Algorithmus des bekannten W-LAN-Standards eingesetzt, der einen gleichstromfreien Bitstrom erzeugt und lange 0- bzw. 1-Folgen verhindert, um so eine robustere Übertragung zu ermöglichen.

In Zukunft sollten größere Stückzahlen des PartC-Boards angefertigt werden. Steht eine große Anzahl von Knoten zur Verfügung kann die Leistungsfähigkeit des neuen Protokolls mittels Feldtests unter realistischen Bedingungen bewertet werden und es können letzte Anpassungen an die Hardware vorgenommen werden. Das neue Particle muss softwareseitig in die vorhandenen Gateways integriert werden, die das Sensornetz mit einem lokalen Netzwerk verbinden. Schließlich muss die Software zum Auslesen von Sensordaten noch an die neue Hardware angepasst werden.

Es kann auch versucht werden, die Leistungsfähigkeit der Implementierung der Arbitrierung zu erhöhen oder auf ein komplett anderes Medienzugriffsverfahren wie CSMA umzusteigen.

A. Anhang

A.1 Glossar

Alleine Ein Knoten ist alleine, wenn er mit keinem anderen Knoten bidirektional kommunizieren kann. Diese Eigenschaft (Isoliertheit) wird im Beaconslot mit der gleichen Methode getestet, die auch bei der Arbitrierung eingesetzt wird. Erkennt die Protokollinstanz, dass sie alleine ist, kann sie, um Energie zu sparen, für eine gewisse Zeit die Kommunikation unterdrücken (mittlere Energie-Ersparnis) oder den Knoten in Tiefschlaf versetzen (höchste Energie-Ersparnis). Abschnitt 3.9

Alleine-Test Beim Alleine-Test bekommt ein Knoten, der ein Beacon sendet eine Rückmeldung von den Knoten, die das Beacon empfangen haben. Für den Masterknoten bei der Master-Slave-Synchronisation ist dies die einzige Möglichkeit festzustellen, ob er isoliert ist. Mit diesem Wissen kann er ggf. Energiesparmaßnahmen wie Kommunikationsunterdrückung und Tiefschlaf einleiten. Abschnitt 3.9.1

Anwendungsebene Die Anwendungsebene bezeichnet den Teil der Ausführungszeit, den der Knoten nicht mit der Ausführung der Interrupt-Service-Routine des Stacks beschäftigt ist. Hier läuft die eigentliche Anwendung, die Sensordaten erfasst und den Stack beauftragt diese als Kontextdaten zu verbreiten, bzw. Pakete von anderen Knoten empfängt und verarbeitet. Abschnitt 4.4.1

Arbitrierungs-Slice (engl. für Schlichter-Zeitscheibe) Eine von 8-9 kurzen Zeitabschnitten zu Beginn eines Slots, der für die Arbitrierung benutzt wird. In einem Arbitrierungs-Slice wird entweder ein Funksignal gesendet oder versucht eines zu empfangen. Abschnitt 3.5.1

Arbitrierung Medienzugriff erfolgt bei der gewählten Implementierung per Arbitrierung (svw. Schlichtung). Dieses Verfahren ist mehrstufig und läuft in mehreren, kurzen Zeitabschnitten ab. Diese Zeitabschnitte befinden sich am Anfang des größeren Slots. Sie dauern ungefähr 800 μs.

Innerhalb eines Arbitrierungs-Slices kann ein Knoten entweder ein Signal senden oder das Medium abhören. In letzterem Fall erfährt der Knoten, ob mindestens ein anderer Knoten ein Signal sendet oder keiner dies tut. Arbitrierungs-Slices werden außer zur Arbitrierung auch beim Alleine-Test und für die Verkehrsanzeige benutzt. Abschnitt 3.5

Beacon (engl. für Bake, Leuchtfeuer) Ein Paket, das Informationen zur zeitlichen Synchronisation enthält. 3.4, 3.6

Beaconslot Jeder Slot in dem ein Beacon gesendet wird ist ein Beaconslot. Bei Master-Slave-Synchronisation ist das jeder n-te Slot, bei verteilter Synchronisation kann dies jeder Slot sein. Abschnitt 3.4, Abschnitt 3.6

Broadcast bezeichnet den Umstand, dass ein Signal bzw. Datenpaket nicht an einen bestimmten anderen Knoten adressiert ist, sondern an alle empfangsbereiten Knoten in Empfangsreichweite. Siehe auch: Unicast.

C-Präprozessor Der C-Präprozessor erlaubt es vor dem Übersetzen des Quellcodes Teile dessen zu ersetzen oder auszublenden. Mit der sog. Define-Anweisung kann man in einer Quellcode-Datei Makros festlegen und ihnen einen Wert zuweisen. Jedes Auftreten des Makros im nachfolgenden Code wird vor dem Übersetzen des Programms durch den Wert des Makros ersetzt. Man kann auch Makros ohne Wert definieren und im Quellcode abhängig davon, ob ein Makro definiert ist oder nicht, Teile des Codes ausblenden. Dies kann für Debug-Code genutzt werden, der dann nicht mit übersetzt wird, wenn das zugehörige Makro nicht definiert ist.

Debugging Dieser Begriff bezeichnet das Aufspüren und Entfernen von Fehlern in Hard- und Software. Zu diesem Zweck können in der vorliegenden Implementierung Debug-Meldungen über den seriellen Port des CC1010 ausgegeben werden und, wenn die Schnittstelle mit dem PC verbunden wird, mit einem herkömmlichen Terminalprogramm gelesen werden. Zum (de)aktivieren verschiedener Debug-Meldungen gibt es C-Präprozessor-Defines, die entweder definiert werden oder nicht.

Duty Cycle Der Duty Cycle ist das Verhältnis der Zeit, in der ein Knoten aktiv bzw. kommunikationsbereit ist, zur Gesamtzeit. Je geringer der Duty Cycle, desto weniger Energie wird benötigt, umso höher ist aber auch die Latenzzeit für das Senden von Daten und der Datendurchsatz pro Zeiteinheit.

Echtzeitebene Die Echtzeitebene bezeichnet den Teil der Ausführungszeit, den der Knoten mit der Ausführung der Interrupt-Service-Routine des Stacks beschäftigt ist. Hier werden alle zeitkritischen Aufgaben ausgeführt, die auf allen Knoten synchron ablaufen müssen, wie das Senden und Empfangen von Paketen und die Arbitrierung. Abschnitt 4.4.1

ID Abk. für Identifikator. Jedes Particle hat einen weltweit eindeutigen Identifikator. Dieser ist 8 Byte lang. Abschnitt 2.4

Knoten Ein Knoten bezeichnet einen Kommunikations-Teilnehmer am drahtlosen Sensornetzwerk. Ein Knoten kann Sensoren oder Aktoren beinhalten oder nur zur Kommunikation dienen (z. B. als Gateway-Modul, das die Pakete des Netzwerks in ein lokales Netz weiterleitet).

Kommunikationsunterdrückung Die Kommunikationsunterdrückung ist eine Maßnahme zum Energie sparen, wenn die Protokollinstanz glaubt, dass sie alleine ist. Während der Kommunikationsunterdrückung läuft die Anwendung weiter, während jedoch Pakete, die diese zum Senden beauftragt verworfen werden und auch keine Pakete Empfangen werden. Abschnitt 3.9

Master Dies ist ein Protokollzustand bei der Master-Slave-Synchronisation. Im Mastermodus sendet ein Knoten alle n Slots ein Beacon. Dieses enthält den aktuellen Wert seines Timers 0. Abschnitt 3.4

n-Hop-Nachbarschaft Alle Knoten, die von einem Ausgangsknoten über n-1 oder weniger Zwischenknoten erreichbar sind, bilden die n-Hop-Nachbarschaft dieses Knotens. Die 1-Hop-Nachbarschaft besteht danach aus allen Knoten, mit denen der Knoten direkt kommunizieren kann, die 2-Hop-Nachbarschaft stellen die Knoten dar, die über einen Zwischenknoten erreichbar sind usw.

Particle Particles sind eine Plattform (Hard- und Software) mit der schnell prototypische Anwendungen im Bereich drahtlose Sensornetzwerke, ubiquitäre Computer, Gebäudewahrnehmung und -steuerrung, Computer in Textilien etc. realisiert werden können. Die Particles wurden vom TecO im Rahmen des EU-Projektes SmartIts entwickelt. Abschnitt 2.4

Rendezvous Ein Rendezvous dient zur Synchronisation von Aufgaben auf Knoten, synchronisiert sind. Da der Timer 0 auf synchronisierten Knoten zu jedem Zeitpunkt den gleichen Wert hat (mit vernachlässigbarer Abweichung), kann man die Werte des Timers mit Zeitpunkten innerhalb eines Slots gleichsetzen. Ein solcher Timerwert stellt ein Rendezvous dar. Die Synchronisation kommt dadurch zu Stande, dass jeder Knoten aktiv wartet (in einer Schleife den aktuellen Timerwert mit dem zu erreichenden vergleicht), bis dieser erreicht wird. Die Befehle nach einem solchen Rendezvous-Punkt werden auf allen synchronisierten Knoten quasi parallel ausgeführt. Abschnitt 4.4.2

RF Englische Abkürzung für „Radio Frequency“. Wird meist als Präfix verwendet und entspricht der deutschen Vorsilbe „Funk-“.

Slot (engl. für Zeitschlitz) Ein Slot ist die gemeinsame Zeitbasis aller synchronisierten Knoten. Den meisten Aufgaben der RF-Schicht sind Zeitpunkte relativ zum Slot zugewiesen (s. a. Rendezvous). Ein Slot dauert in der gewählten Implementierung 30ms und beginnt auf allen synchronisierten Knoten zur gleichen Zeit. Zu Beginn eines Slots wird eine Interrupt-Service-Routine aufgerufen, die den normalen Programmablauf („Anwendungsebene“) unterbricht und zur „Echtzeitebene“ wechselt. Abschnitt 3.4 und Abschnitt 3.6

Scrambling (engl. für verwürfeln) bezeichnet ein Verfahren zur Kodierung eines binären Datenstroms. Die Ausgabe dieser Codierung ergibt einen neuen Datenstrom, der weitgehend gleichstromfrei ist (d. h. gleich viele 0-en und 1-en enthält) und keine langen 0- bzw. 1-Folgen enthält. Diese Eigenschaften machen die Übertragung über die Funkschnittstelle robuster. Abschnitt 3.10

Slave Dies ist ein Protokollzustand bei der Master-Slave-Synchronisation. Im Slavemodus erwartet ein Knoten alle n Slots ein Beacon. Nach Empfang des Beacons stellt es seinen Timer 0 und läuft danach synchron mit dem Master. Abschnitt 3.4

Synchronisation Die Synchronisation ist ein Verfahren um auf mehreren Knoten des Netzwerkes eine gleiche Zeitbasis herzustellen. Sie ähnelt einem Uhrenvergleich: Ein Knoten sendet eine Darstellung seiner aktuellen Zeit, alle Knoten, die diese Information empfangen verwenden sie um ihre innere Uhr (in vorliegenden Fall einen Timer) zu stellen. Dies ermöglicht es, bestimmte Vorgänge auf verschiedenen Knoten zeitgleich auszuführen. Abschnitt 3.4 und Abschnitt 3.6

Synchronisation mit Master und Slave Bei dieser Form der Synchronisation wird ein Knoten zum Master bestimmt und alle anderen zu Slaves. Der Master sendet alle n Slots ein Beacon, alle Slaves empfangen dieses Beacon und stellen ihre internen Timer. Abschnitt 3.4.

Synchronisation, verteilte Bei verteilter Synchronisation entscheidet jeder Knoten zufällig in jedem Slot, ob er ein Beacon senden möchte – abhängig von der Anzahl der Slots, die seit dem Senden oder Empfangen des letzten Beacons vergangen sind. Sollen im Netz sowohl Datenpakete als auch Beacons versendet werden, bevorzugt die Arbitrierung die Beacons. Wollen mehrere Knoten ein Beacon im gleichen Slot senden, wird eine Arbitrierung wie zwischen Datenpaketen durchgeführt. Abschnitt 3.6

TecO Das Telecooperation Office des Instituts für Telematik an der Universität Karlsruhe.

Tiefschlaf Der Tiefschlaf ist ein Betriebszustand in den der Stack den Knoten versetzt um Energie zu sparen. Während des Tiefschlafs wird die Anwendung nicht ausgeführt. Dieser wird periodisch angewendet, wenn der Knoten glaubt, alleine zu sein. Abschnitt 3.9

Unicast bezeichnet den Umstand, dass ein Signal bzw. Datenpaket an einen bestimmten anderen Knoten adressiert ist. Alle anderen außer dem Zielknoten, die die Übertragung empfangen, sollten sie ignorieren. Siehe auch: Broadcast.

Watchdog Ein Watchdog-Timer (deutsch: Wachhund-Zeitgeber) ist eine Hardware-Einrichtung, die ein System neu startet, falls das laufende Programm in einen undefinierten Zustand gerät. Das Programm muss in regelmäßigen Abständen ein Bit in einem Register setzten oder löschen, sonst startet der Watchdog das System neu.

A.2 LED-Anzeige

Auf dem Chipcon Evaluation Board sind 4 farbige LEDs angebracht, auf den PartC-Boards jeweils zwei. Diese können mittels Software an- und ausgeschaltet werden. Zur Übersetzungszeit muss entschieden werden, ob die LEDs des Evaluation Boards benutzt werden oder die der PartC-Boards. Ersteres erreicht man durch inkludieren der Header-Datei `cc1010eb.h`, letzteres indem man `partCboard.h` einbindet.

Name	CC1010 Evaluation Board	TecO PartC Board
LED 1	Rote LED	nicht vorhanden
LED 2	Gelbe LED	rote LED
LED 3	Grüne LED	nicht vorhanden
LED 4	Blaue LED	blaue LED

Tabelle A.1: *Zuordnung der verwendeten Namen zu den LEDs*

Vom Blinken der LEDs kann man nun auf den aktuellen Zustand eines Knotens schließen. Zum weiter gehenden Debugging kann über das CC1010 Evaluation Board der serielle Port des Knoten mit dem Computer verbunden werden. Es ist dann möglich, über ein Terminal-Programm Debug-Ausgaben des Knotens zu verfolgen.

LED 1 blinkt langsam (ca. 1 Hz) Der Knoten ist im Energiesparmodus. Er geht in diesen Modus, wenn er der Meinung ist, dass sich keine weiteren Knoten in (bidirektionaler) Kommunikationsreichweite befinden. Dieser Modus ist in 2 Varianten realisiert: Entweder der Knoten betreibt nur Kommunikationsunterdrückung oder geht in Tiefschlaf. Bei der Kommunikationsunterdrückung läuft die Anwendung weiter, der Stack verwirft aber die von der Anwendung zum Senden beauftragte Pakete und versucht auch nicht Pakete zu Empfangen. Im Tiefschlaf, bei dem der Knoten nur ca. 30 μA braucht, wird auch die Anwendung angehalten.

LED 2 blinkt sehr schnell Der Knoten macht eine intensive Suche nach einen Beacon um sich an die Synchronisation eines bestehenden Netzes anzukoppeln. Dieser Modus wird aktiviert, wenn ein Knoten die Synchronisation mit anderen Knoten verloren hat, z. B. nach dem Einschalten oder wenn ein er aus dem Energiesparmodus aufwacht. Bei Master-Slave-Synchronisation wird dieser Modus auch aktiviert wenn ein Slave in mehreren Beaconslots kein Beacon empfangen hat.

LED 2 blinkt schnell (ca. 3Hz) Dies zeigt an, dass sich der Knoten regelmäßig mit einem oder mehreren anderen Knoten synchronisiert.

LED 4 blinkt oder flackert LED 4 leuchtet so lange der Knoten die ISR des Stacks ausführt. Je heller diese LED leuchtet, desto länger wird also für die Ausführung des Protokolls benötigt. Je dunkler sie leuchtet, desto mehr Zeit steht für die Ausführung der Anwendung zur Verfügung.

LED 1 und LED 3 flackern abwechselnd Dies zeigt den Ausgang der Arbitrierung des Netzwerkzugriffs an. Wenn ein Signal auf dem Medium

entdeckt wurde, der Knoten also nicht senden darf, wird die LED 1 aktiviert. Wurde kein Signal auf dem Medium gefunden wird die LED 3 aktiviert.

keine LED blinkt Es liegt ein Fehler vor.

A.3 Sourcecode-Struktur und Konfiguration

Die beiden Synchronisationsvarianten des Stacks sind in jeweils einer Datei implementiert. Die Datei für den Stack mit Master-Slave-Synchronisation heißt `stackMasterSlaveSync.c` und die mit verteilter Synchronisation heißt `stackDistSync.c`. Zu den c-Quellcodedateien gehört jeweils eine UV2-Datei. Diese enthält Konfigurationsdaten für die mit dem Evaluation Kit mitgelieferte Entwicklungsumgebung von Keil Inc., insbesondere die Information darüber, welche anderen übersetzten Dateien der Linker benötigt, um ausführbaren Code zu erzeugen. Diese Dateien werden von `keilbuild.pl` verwendet, ein in Perl entwickeltes Buildscript, das alle benötigten Programmteile übersetzt und alle Parameter für den Aufruf von SDCC bereitstellt. Das Keilbuild-Skript stellt damit, zusammen mit SDCC, eine Alternative zur Keil IDE zur Verfügung. Es ist zu beachten, dass man zunächst die Pfadangaben im Keilbuild-Skript anpassen muss, bevor man es verwenden kann.

Die Implementierung kann eine Vielzahl von Debug-Meldungen[1] über die serielle Schnittstelle des CC1010 EBs ausgeben. Dies führt in manchen Fällen dazu, dass die Anwendung nicht mehr zuverlässig funktioniert, liefert aber wichtige Informationen über das Particle.

Die Steuerung der Debug-Meldungen erfolgt über C-Makros ohne Inhalt (auch Schalter bzw. Switch genannt), die entweder definiert werden oder nicht. Im Code werden so mittels Präprozessor-Anweisungen Teile des Codes ein- bzw. ausgeblendet, die die Debug-Ausgaben an- bzw. ausschalten. Im folgenden werden die einzelnen Switches und ihre Bedeutung aufgelistet.

`DEBUG_USE_UART` Dieser Schalter muss gesetzt sein, damit Debug-Ausgaben über die serielle Schnittstelle erfolgen. Ist er nicht gesetzt so werden sämtliche anderen Einstellungen, die Debug-Ausgaben betreffen, überschrieben, und es finden keine Ausgaben statt.

`DEBUG_USE_UART1_FOR_DUMPING` Dieser Schalter legt fest, über welche Schnittstelle die Ausgabe der Debugmeldungen erfolgen soll. Ist er gesetzt, so erfolgt die Ausgabe über den seriellen Port 1, ansonsten über den seriellen Port 0.

`DEBUG_DUMP_RSSI_SAMPLES` Wenn dieser Schalter gesetzt ist, werden die bei der Arbitrierung gemessenen RSSI-Werte ausgegeben. Es muss beachtet werden, dass die Ausgabe der Werte so lange dauern kann, dass die nach der Arbitrierung zu sendenden Pakete erst nach dem Rendezvous, d. h. nicht synchron gesendet und empfangen werden, so dass alle diese Pakete verloren gehen können.

`DEBUG_DUMP_RF_STATE` Der aktuelle Status der RF- und LL-Schichten wird am Ende der ISR ausgegeben, wenn dieser Schalter aktiviert ist.

`DEBUG_DUMP_SYNC_VALUES` Ist dieser Schalter gesetzt, so wird beim Empfang eines Beacons der aktuelle Wert des Timers 0 ausgelesen bevor der

[1] Meldungen zur Fehlersuche und -beseitigung

gerade empfangene zugewiesen wird. Beide Werte werden danach ausgegeben. Unterscheiden sie sich stark, funktioniert die Synchronisation nicht gut und muss neu kalibriert werden – sind sie annähernd gleich, läuft die Synchronisation effektiv.

DEBUG_DUMP_RF_STATISTICS Mit der Funktion DebugDumpStats() können allgemeine Statistiken zum Stack ausgegeben werden, wenn dieser Schalter aktiviert ist, z. B. die Anzahl der Slots, in denen die Arbitrierung gewonnen wurde.

DEBUG_ARBITRATION_TEST_... Diese Schalter ermöglichen die Kalibrierung der Arbitrierung (siehe Abschnitt A.4 auf der nächsten Seite)

Außer zum Debugging werden die Präprozessor-Makros auch benutzt, um Parameter des Protokolls zu festzulegen.

RF_SYNC_MAX_SLOT_COUNT In diesem Makro wird die Anzahl der Slots gespeichert, nach der der Slotzähler für die Synchronisation spätestens auf 0 zurückgesetzt wird. Bei Master-Slave-Synchronisation ist dies die Anzahl der Slots zwischen zwei Beacons. Bei verteilter Synchronisation wird die aktuelle Slotnummer zunächst der Funktion übergeben, die dann (zufällig aber mit bekannter Wahrscheinlichkeit) zurück gibt, ob ein Beacon gesendet werden soll oder nicht.

RF_USE_ALONE_DETECTION Mit diesem Schalter kann die Erkennung der Isoliertheit und das damit verbundene, periodische Schlafen deaktiviert werden, wie es in Abschnitt 3.9 auf Seite 36 beschrieben ist.

RF_ALONE_TEST_FAILED_THRESHOLD Gibt die Anzahl der Slots ohne Empfang eines Beacons, eines Datenpaketes oder einer Beaconantwort an, nach der der Schlafmodus ausgelöst wird.

RF_ALONE_DEEP_SLEEP Ist dieser Schalter gesetzt, wird Tiefschlaf als Schlafmodus gewählt,

RF_PREAMBLE_LENGTH_TX Die Anzahl der Präambel-Bytes, die vor einem Paket gesendet werden.

RF_PREAMBLE_LENGTH_RX Die Anzahl der Präambel-Bits, die der Empfänger korrekt empfangen haben muss, bevor er nach dem Start-Of-Packet-Byte sucht.

RF_MAX_SYNC_JITTER Die Anzahl der Mikrosekunden, die ein Knoten auf ein ankommendes Paket (inklusive Präambel) wartet.

RF_SLOT_LENGTH_IN_US Länge eines Slots in Mikrosekunden.

RF_MASTER_SEARCH_SLOTS_STARTUP Anzahl der Slots in denen ein Knoten nach dem Einschalten nach einem Sync-Signal sucht, bevor er in den Master- (bei Master-Slave-Synchronisation) bzw. Normalzustand (bei verteilter Synchronisation) geht.

RF_MASTER_SEARCH_SLOTS_AFTER_ALONE Anzahl der Slots in denen ein Knoten nach dem Aufwachen aus dem Schlafmodus nach einem Sync-Signal sucht, bevor er in den Master- (bei Master-Slave-Synchronisation) bzw. Normalzustand (bei verteilter Synchronisation) geht.

RF_MAX_FAILED_SYNC_COUNT Nur Master-Slave-Synchronisation: Nach so vielen Beaconslots geht ein Slave in die Beaconsuche um wieder der Synchronisation anzuschließen.

RF_MASTER_RIVAL_TEST_SYNC_COUNT Nur Master-Slave-Synchronisation: Nach so vielen Beaconslots sucht ein Master bis zum nächsten Beaconslots nach einem Beacon um festzustellen, ob sich noch ein weiterer Master in Empfangsreichweite befindet.

RF_SYNC_ADDITIONAL_OFFSET Wenn ein Beacon gesendet wird, so wird vor dem Senden der lokale Timerwert ausgelesen. Zu diesem wird der Offset `RF_SYNC_ADDITIONAL_OFFSET` addiert und im Beacon gesendet. Dies hat den Vorteil, dass ein Knoten, der das Beacon empfängt, den Wert sofort und ohne weitere Rechenoperationen zuweisen kann.

RF_ARBITRATION_RSSI_THRESHOLD Während eines Arbitrierungs-Slices wird entweder ins Medium gehört oder gesendet. Beim Hören ins Medium wird ein RSSI-Wert gemessen, der umso kleiner ist, je stärker ein ankommendes Signal ist. Dieses Makro legt denjenigen RSSI-Wert fest, ab dem angenommen wird, dass ein Signal auf dem Medium liegt. Abschnitt 3.5.1 auf Seite 25

RF_ALONE_TEST_RSSI_THRESHOLD Dieses Makro legt denjenigen RSSI-Wert fest, ab dem angenommen wird, dass eine Beaconantwort auf dem Medium liegt. Abschnitt 3.9.1 auf Seite 37

RF_TRAFFIC_INDICATION_TEST_RSSI_THRESHOLD Dieses Makro legt denjenigen RSSI-Wert fest, ab dem angenommen wird, dass eine Beaconantwort auf dem Medium liegt. Abschnitt 3.8 auf Seite 34

RF_RENDEZVOUS_... Dies sind Rendezvous-Werte für verschiedene Aktionen, die synchron auf allen Knoten ausgeführt werden müssen. Abschnitt 4.4.2 auf Seite 58

RF_ARBITRATION_SLICE_TIMER_TICS Legt die Länge eines Arbitrierungs-Slices in Timer-Tics fest.

RF_BUFFER_SIZE Legt die ACL-Nutzdatenlänge eines Datenpaketes in Byte fest.

A.4 Kalibrierung der Arbitrierung

Um die in Abschnitt 4.4.3 auf Seite 58 beschriebenen Messungen durchzuführen benötigt man zwei Knoten:

Auf dem einen muss der Schalter `DEBUG_ARBITRATION_TEST_0` aktiviert sein (zusammen mit `DEBUG_DUMP_RSSI_SAMPLES`). Dies ist der Messknoten.

Auf dem anderen Knoten muss einer der beiden anderen Schalter aktiviert sein. Dies ist der Senderknoten. Auf beiden Knoten sollte die Anwendung keine Datenpakete verschicken.

Jeder dieser Schalter bewirkt, dass die Arbitrierung nicht wie gewöhnlich abgebrochen wird, sobald bei einem lokalen 0-Bit eine 1 im Medium gehört wurde, sondern immer bis zum letzten Bit durchgeführt wird.

Der Messknoten führt die Arbitrierung mit einem 0-Byte statt einer Zufallszahl durch, d. h. er hört alle 8 Zeitschlitze ins Medium. Da zusätzlich die Ausgabe der RSSI-Werte aktiviert ist, werden diese ausgegeben. Zusätzlich wird auch die Klassifikation des Wertes nach dem aktuellen Schwellwert angegeben: eine 1 bedeutet, dass der Wert nach dem aktuellen Schwellwert als Singal interpretiert wird, eine 0 bedeutet, es wird angenommen, dass kein Signal auf dem Medium ist.

Der andere Knoten benutzt als Arbitrierungsbyte entweder eine 01-Folge (beim Schalter `PATTERN`) oder eine fortlaufend erhöhte Zahl (beim Schalter `COUNT`). Somit kann der Messknoten feststellen, ob die gesendeten Arbitrierungsbytes beim Empfänger auch korrekt interpretiert werden.

A.5 API-Dokumentation

Während der Implementierung des neuen Particle-Protokolls wurden eine Dokumentation aller verwendeten Variablen, Makros und Funktionen erstellt. Die Dokumentation erfolgte mit speziellen Kommentaren innerhalb des Quellcodes, die vom automatischen Dokumentationssystem *Doxygen*[2] ausgelesen und zu einer ansprechenden HTML-Dokumentation verarbeitet wurden.

Die Dokumentation ist auf der Homepage zur Diplomarbeit zu finden. Sie liegt dort unter http://www.teco.edu/~spiess/apidoku/.

[2] http://www.doxygen.org/

Literaturverzeichnis

[802.97] IEEE 802.11. Wireless Lan Medium Access Control (MAC) And Physical Layer (PHY) Specifications. IEEE standard for information technology, IEEE, 1997. 40

[BeGe] Michael Beigl und Hans Gellersen. Smart-Its: An Embedded Platform for Smart Objects. 8, 9

[BKZD+03] M. Beigl, A. Krohn, T. Zimmer, C. Decker und P. Robinson. AwareCon: Situation Aware Context Communication, 2003. 8

[IEEE03a] IEEE. IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks specific requirements part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs). IEEE standard for information technology, IEEE, 2003. 11

[IEEE03b] IEEE. IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks specific requirements part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs). IEEE standard for information technology, IEEE, 2003. 11

[RASJP+00] Jan M. Rabaey, M. Josie Ammer, Julio L. da Silva Jr., Danny Patel und Shad Roundy. PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking. *IEEE Computer Magazine*, July 2000, S. 42–48. 8, 10, 16

[sdcc] 45

Homepage of the Small Devices C Compiler (SDCC).

[TecO] TecO. Particle Computer Homepage. 18

[Weis91] Mark Weiser. The Computer for the Twenty-First Century. *Scientific American*, September 1991, S. 94–10. 5

[YeHE02] W. Ye, J. Heidemann und D. Estrin. An energy-efficient MAC protocol for wireless sensor networks, 2002. 10

Index

Abbildungsverzeichnis