HTW Dresden
Fakultät Informatik/Mathematik
Masterarbeit

Masterarbeit

Thema: Betrachtung von Stromsparmodi für den 802.15.4-Linux-Kerneltreiber und Integration eines geeigneten Verfahrens

Autor: Thomas Kühnel

Betreuer und Erstgutachter: Prof. Dr.-Ing. Jörg Vogt

Zweitgutachter: Prof. Dr.-Ing. Robert Baumgartl

eingereicht am: 22.10.2014

Danksagungen

Vor allem möchte ich mich bei Prof. Dr. Ing. Jörg Vogt für die Betreuung dieser Masterarbeit bedanken. Er stand stets hilfreich zur Seite und hat für neue Motivation gesorgt.

Weiterer Dank gilt der Firma Dresden Elektronik, die diese Arbeit durch die Bereitstellung von Hardware unterstützt hat, welche zum Testen der Implementierung verwendet wurde.

Ebenso geht mein Dank an alle Kommilitonen und Freunde, die mich während des Studiums begleitet haben. Speziell möchte ich dabei Michael Hanslik und Thomas Winkler dafür danken, dass sie diese Arbeit Korrektur gelesen und zahlreiche Verbesserungsvorschläge gegeben haben.

Meinen Eltern möchte ich dafür danken, dass sie mich während der Studienzeit nicht nur finanziell, sondern auch emotional stets unterstützt haben. Ohne sie wäre dieses Studium nicht möglich gewesen.

Inhaltsverzeichnis

Listings

Abbildungsverzeichnis

Abkürzungsverzeichnis

6LoWPAN	IPv6 over Low power Wireless Personal Area Network
CAP	Contention-Access-Period
CCA	Clear Channel Assessment
CFP	Contention-Free-Period
CSMA/CA	Carrier sense multiple access with collision avoidance
CoAP	Constrained Application Protocol
GTS	Guaranteed-Timeslots
HAT	Hardware Attached on Top
LQI	Link Quality Indication [Erg04, S. 8]
LR-WPAN	Low-Rate Wireless Personal Area Network
MAC	Medium Access Layer
MCPS	MAC common part sublayer
MFR	MAC Footer
MHR	MAC Header
MLME	MAC sublayer management entity

NAT Network Address Translation

OSI Open Systems Interconnection

PAN Personal Area Network

RDC Radio Duty Cycling

SPI Serial Peripheral Interface

USB Universal Serial Bus

WLAN Wireless Local Area Network

1 Einleitung

1.1 Motivation

Der Bereich der Hausautomatisierung ist aktuell ein stark wachsendes Forschungs- und Entwicklungsgebiet. Grund dafür sind nicht nur die in den letzten Jahren gesunkenen Preise für kommerzielle Hausautomatisierungslösungen, sondern auch die ständige Vernetzung etwa durch Smartphones. Da der Nutzer selbst von überall erreichbar ist, steigt auch das Bedürfnis, auf Geräte im eigenen Haus aus der Ferne zuzugreifen.

Zudem ist die Realisierung dieser Systeme heutzutage nicht nur günstiger, sondern auch einfacher. Früher gab es vorwiegend drahtgebundene Lösungen, die deshalb nur bei Bau des Hauses mit vergleichsweise geringem Aufwand eingebaut werden konnten. Heutige Lösungen arbeiten vorwiegend über Funk und sind batteriebetrieben. Durch optimierte Soft- und Hardware lassen sich für die Geräte lange Batterielaufzeiten realisieren, wodurch auch der Mehraufwand durch nötiges Wechseln der Batterien gering bleibt. Zudem ist es bei einer drahtlosen Lösung auch einfacher, die Module wieder zu entfernen, wodurch sie sich auch für den privaten Einsatz in einer gemieteten Wohnung eignet.

Drahtlose Sensornetze bestehen aus mehreren räumlich verteilten Knoten, die jeweils eine konkrete Aufgabe erfüllen. Dabei können Knoten als Sensoren agieren, die etwa die Raumtemperatur ermitteln, oder als Aktoren, welche beispielsweise eine Heizung steuern. Des Weiteren gibt es auch Module, in denen Sensoren und Aktoren kombiniert sind.

Sämtliche Sensorknoten haben die Gemeinsamkeit, dass die Knoten meist ohne Verbindung zum Stromnetz stehen und deswegen mit Batterien betrieben werden müssen. Eine Alternative zum Batteriebetrieb wäre die Nutzung von Energy Harvesting, also der Gewinnung von Energie aus der Umgebung des Sensors, etwa durch eine kleine Solarzelle. Um die Geräte praktisch nutzen zu können, ist es deshalb notwendig, dass sie möglichst wenig Energie benötigen und dadurch Laufzeiten von mehreren Jahren erreichen, ohne dass in dieser Zeit eine Wartung notwendig ist.

Zur Funkübertragung ist hierfür der Standard 802.15.4 eine gängige Lösung, da dieser speziell für Anwendungen wie Sensornetze entwickelt wurde und daher für niedrigen Energieverbrauch konzipiert wurde. Trotzdem ist der Hauptverbraucher eines Knoten üblicherweise das Funkmodul, dasich dieses dauerhaft im Empfangszustand befinden muss, um keine Pakete zu verpassen. Da üblicherweise aber nur wenige Pakete pro Minute oder sogar noch weniger übertragen werden müssen, ist dies sehr ineffizient.

Deshalb ist es notwendig, mit Verfahren zu arbeiten, bei denen das Funkmodul möglichst lange deaktiviert bleiben kann, aber es trotzdem möglich ist, zuverlässig Daten zu übertragen. Von kommerziellen Lösungen wird zum stromsparenden Betrieb entweder das ZigBee-Protokoll oder eine vollständige Eigenentwicklung verwendet. In freien Systemen gibt es vorwiegend Eigenentwicklungen, die nach keinem Standard entwickelt wurden und nicht zueinander kompatibel sind.

Durch die Verbreitung der ARM-Prozessorarchitektur und Boards wie den Raspberry Pi, ist es möglich, 802.15.4 Technologie direkt unter Linux zu verwenden. Somit kann ein Board mit Raspberry Pi und einem Funkmodul als kleiner und günstiger Router für ein drahtloses Sensornetz agieren. Die Nutzung von Linux bietet gegenüber spezialisierten Lösungen auch den Vorteil, dass dafür eine große Auswahl an freier Software verfügbar ist und diese leicht angepasst werden kann. Zusammen mit einfachen Sensorknoten, welche mit freier Software betrieben werden, ist es damit möglich, ein eigenes „Smart Home“ zu realisieren, ohne auf teure, proprietäre Lösungen zurückgreifen zu müssen.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist es, eine Möglichkeit zu finden, auch unter Linux mit stromsparenden Sensornetzen arbeiten zu können. Dies ist zwar auch jetzt schon durch diverse eigene Lösungen möglich, bei denen es sich aber meist um spezielle Werkzeuge handelt, die nur auf das dazugehörige Sensorbetriebssystem ausgelegt sind. Beispiele für solche Lösungen sind der Border-Router von Contiki [Con14] oder der PppRouter von TinyOS [Tin14]. Bei diesen arbeitet ein Sensorknoten mit dem entsprechenden Embedded Betriebssystem als Router zwischen Sensornetz und Host-PC und sorgt am Host über eine virtuelle Netzwerkschnittstelle für den Zugriff auf das Sensornetz. Weiterhin gibt es auch proprietäre Lösungen, wie etwa deCONZ [dre14] der Firma Dresden Elektronik. Bei dieser Möglichkeit ist der Zugriff nur über die eigene Software möglich und nicht mehr über die Netzwerkschnittstellen von Linux.

Bei der vorzustellenden und zu erweiternden Lösung besitzt der Linuxhost ein eigenes Funkmodul, welches direkt über einen Treiber angesprochen wird, ohne einen speziellen Knoten zu benötigen, der die beiden Systeme verbindet. Dabei liegt der Vorteil darin, dass keine komplexe Software auf der Funkhardware notwendig ist, die als eigenständiges Projekt entwickelt, bzw. angepasst und gewartet werden muss, sondern möglichst viel Code auf der Linux-Plattform implementiert ist. Das macht auch Updates einfacher, da nur ein Paket mit dem entsprechenden Kernelmodul aktualisiert werden muss, anstatt eine neue Firmware auf das Funkmodul aufzuspielen.

Treiber für verschiedene Funkchips und Teile der Medium Access Layer (MAC)-Schicht wurden dabei schon implementiert. Allerdings sind die speziellen Funktionen zum Umgang mit stromsparenden Knoten, welche die sich die meiste Zeit in einem Schlafmodus befinden und damit nicht ansprechbar sind, noch nicht im Treiber umgesetzt. Deshalb soll ein geeignetes Verfahren zur Kommunikation mit stromsparenden Knoten gefunden und in den 802.15.4-Treiber integriert werden.

Hier soll speziell der Modus zur indirekten Übertragung untersucht werden. Dabei handelt es sich um einen im 802.15.4 Standard spezifizierten Modus, bei dem die Kommunikation vom Empfänger ausgeht und dieser bei einem übergeordneten Knoten gelagerte Nachrichten anfragt. Mit diesem Modus kann ein sehr geringer Energieverbrauch erreicht werden. Allerdings wurde er bisher wenig untersucht und unter den freien Betriebssystemen für Mikrocontroller ist TinyOS das einzige, in dem er implementiert wurde.

Die Entwicklung soll unter Linux laufen und als Host die Raspberry Pi-Plattform nutzen. Da dies eine kostengünstige und gut verfügbare Hardwareplattform ist, eignet sie sich ideal für Projekte wie ein eigenes, privates Sensornetz.

Weiterhin wurde zur Entwicklung ein RaspBee-Funkmodul zur Verfügung gestellt, welches speziell zur Verwendung mit dem Raspberry Pi entwickelt wurde. Standardmäßig arbeitet dieses Modul nur mit einer eigenen proprietären Software zusammen. Es soll untersucht werden ob und wie es sich mit den freien Linux-Treibern nutzen lässt.

1.3 Aufbau der Masterarbeit

Diese Arbeit ist in sechs Kapitel eingeteilt. Sie begleitet den Entwicklungsprozess an den Linux-Treibern und einer Firmware für das RaspBee-Funkmodul. Zuerst werden im Kapitel „Stand der Technik“ Grundlagen erklärt und gezeigt, welche Probleme auf diesem Gebiet bereits gelöst sind und wo noch Arbeit notwendig ist. Anschließend werden verschiedene Verfahren zum Stromsparen auf dem MAC-Layer vorgestellt und verglichen. Aus diesem Vergleich soll ein Verfahren hervorgehen, welches im Anschluss implementiert werden soll. Darauf folgen die beiden Kapitel „Entwurf“ und „Implementierung“, in denen die Entwicklungsschritte dokumentiert und aufgetretene Probleme und Entscheidungen erläutert werden. Im abschließenden Kapitel werden die Ergebnisse noch einmal zusammengefasst sowie ausgewertet und Erweiterungsmöglichkeiten genannt.

2 Stand der Technik

2.1 IEEE 802.15.4

Bei 802.15.4 handelt es sich um einen von der IEEE Standards Association entwickelten offenen Standard zur drahtlosen Übertragung von Daten. Dieser wurde speziell für die Anforderungen von Low-Rate Wireless Personal Area Networks (LR-WPANs) entwickelt und kann damit beispielsweise für drahtlose Sensornetze eingesetzt werden. Im Vergleich zu anderen Funkstandards wie etwa WLAN gehört hier der Stromverbrauch zu den wichtigsten Faktoren, da die Knoten in einem solchen Netz üblicherweise nur mit Batterien betrieben werden, welche möglichst lange halten sollen.

In diesem Standard sind die Bitübertragungsschicht und Sicherungsschicht des OSI-Modells spezifiziert. Für höhere Schichten gibt es weitere Protokolle, die auch für den Einsatz auf stromsparenden Sensorknoten gedacht sind. Beispielsweise könnte ein mit 802.15.4 betriebenes Netz auf der Vermittlungsschicht das Protokoll IPv6 over Low power Wireless Personal Area Network (6LoWPAN) einsetzen. Dieses wurde entwickelt, um IPv6 auch in Netzen mit sehr einfachen Knoten verfügbar zu machen. Auf der Anwendungsschicht gibt es das Constrained Application Protocol (CoAP), welches ein vom Funktionsumfang zu HTTP ähnliches Protokoll ist, aber auf möglichst geringen Overhead durch Protokollheader optimiert wurde. Außerdem definiert es zusätzliche Funktionen, die in HTTP nicht vorhanden sind. Ein Beispiel dafür ist das Abonnieren einer Ressource beim Server, um daraufhin automatisch über Änderungen informiert zu werden. Damit ist es ideal

zum Einsatz in Sensornetzen geeignet, denn es muss kein aufwendiges Polling der Ressource mehr betrieben werden.

2.1.1 Geräteklassen

Die Knoten in einem 802.15.4 Netzwerk lassen sich in zwei verschiedene Klassen einteilen: Full Function Devices und Reduced Function Devices.

Ein Full Function Device kann mit jedem anderen Knoten in seiner Reichweite kommunizieren und ist immer empfangsbereit für eingehende Nachrichten. Diese Knoten können deshalb als Koordinator oder PAN Koordinator agieren und damit Aufgaben zur Verwaltung des Netzes erfüllen. Dazu gehören die Verwaltung der sich im Netz befindenden Knoten und deren Adressen, gegebenenfalls das Zwischenspeichern von Daten für Knoten, die aufgrund von Stromsparfunktionen nicht direkt ansprechbar sind und das Weiterleiten von Paketen für weiter entfernte Knoten.

Ein Reduced Function Device darf nur als einfacher Netzknoten agieren. Es kann keine Verwaltungsfunktionen übernehmen und keine Daten zu anderen Reduced Function Devices übertragen. Nur eine Kommunikation mit Full Function Devices ist möglich. Die Funktionen dieser Knoten sollen sehr einfach sein und damit eine hohe Batterielebensdauer erreichen. Dies wird durch Algorithmen zur Kommunikation mit den Full Function Devices erreicht, die es ermöglichen, dass die Knoten die meiste Zeit deaktiviert sein können und trotzdem eine Funkübertragung möglich ist.

2.1.2 Paketarten

Es gibt vier verschiedene Arten von Paketen (Frames), die auf der MAC-Schicht in einem 802.15.4-Netzwerk übertragen werden können: Beacon, Command, Data und Acknowledgement Frames.

Adressiert werden diese Pakete entweder über die MAC-Adressen der Kommunikationspartner oder alternativ über die im Standard spezifizierten Kurzadressen. Diese sind im Gegensatz zur MAC-Adresse (sechs Byte) nur zwei Byte lang und verringern damit den nötigen Overhead. Die Vergabe und Verwaltung dieser Adressen erfolgt durch den Koordinator.

Eine weitere in 802.15.4 verwendete Adresse ist die PAN-Id. Dies ist eine drei Byte lange Adresse, die zur Identifikation des Netzwerkes dient. Damit können mehrere Netzwerke auf der gleichen Frequenz betrieben werden, ohne dass diese sich gegenseitig stören.

Jedes der Pakete beginnt mit einem MAC Header (MHR), darauf folgt die Payload, in der die eigentlichen Nutzdaten gespeichert sind. Beendet wird jedes Paket mit einem MAC Footer (MFR)-Feld, in dem eine Prüfsumme der gesamten Paketdaten enthalten ist.

Ein **Beacon Frame** wird von einem Koordinator periodisch übertragen und enthält Informationen zum Status des Netzwerks. Dazu gehört zum Beispiel, zu welchen Zeitpunkten Knoten Daten senden dürfen oder für welche Knoten der Koordinator ausstehende Übertragungen zwischengespeichert hat. Außerdem kann ein Beacon Frame verwendet werden, um potentielle neue Knoten über die Existenz des Netzwerkes zu informieren.

Octets: 2	1	4/10	0-14	2	variable	variable	variable	2
Frame Control	Sequence Number	Addressing Fields	Auxiliary Security Header	Superframe Specification	GTS fields	Pending address fields	Beacon Payload	FCS
MHR				MAC Payload				MFR

Abbildung 2.1: Beacon Frame

In einem **Command Frame** sind Befehle enthalten, die zur Verwaltung des Netzes auf MAC-Schicht verwendet werden. Dazu gehören etwa die Anfrage zum Beitreten und Verlassen des Netzwerkes oder das Anfordern von Paketen vom Koordinator. Antworten auf diese Befehle werden ebenfalls als Command Frames versendet.

Octets: 2	1	4/10	0-14	1	variable	2
Frame Control	Sequence Number	Addressing Fields	Auxiliary Security Header	Command Frame Identifier	Command Payload	FCS
MHR				MAC Payload		MFR

Abbildung 2.2: Command Frame

In den **Data Frames** sind Daten enthalten, die in ihren Payload Feldern Pakete aus höheren Schichten enthalten und von diesen dann weiterverwendet werden können.

Octets: 2	1	4/10	0-14	variable	2
Frame Control	Sequence Number	Addressing Fields	Auxiliary Security Header	Data Payload	FCS
MHR				MAC Payload	MFR

Abbildung 2.3: Data Frame

Anschließend gibt es noch die **Acknowledgement Frames**. Diese werden verwendet, um den Empfang eines Command oder Data Frames zu bestätigen, wenn dies angefordert wurde. Um diese Pakete möglichst kurz zu halten, gibt es in ihnen keine Sender- oder Empfängeradresse, sondern nur die Sequenznummer des zu bestätigenden Pakets. Außerdem kann in einem Bit angegeben werden, ob der Koordinator ausstehende Daten besitzt.

Octets: 2	1	2
Frame Control	Sequence Number	FCS
MHR		MFR

Abbildung 2.4: Acknowledgement Frame

2.2 Aufbau des Treibers

Der 802.15.4-Linuxtreiber implementiert die unteren beiden Schichten des OSI-Modells, also MAC und PHY. Darauf aufbauend gibt es eine Netlink Socket-

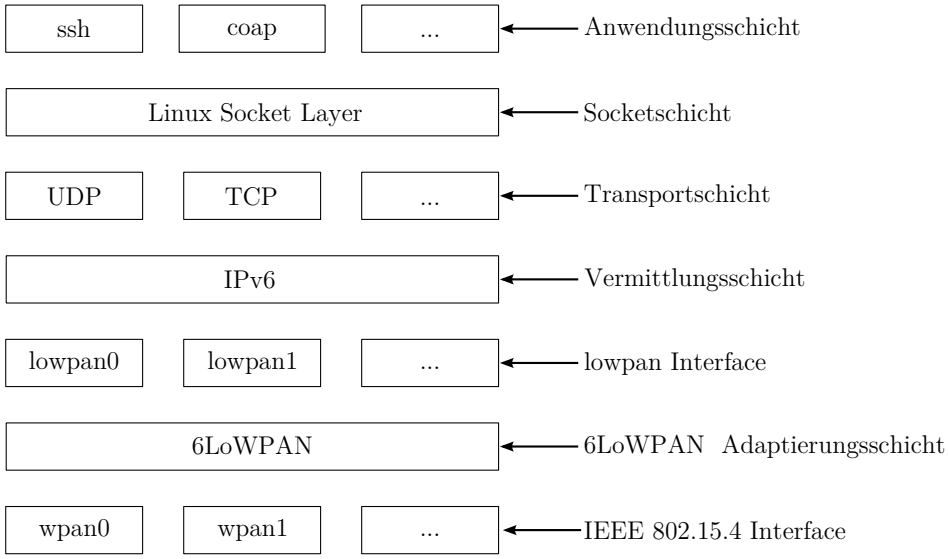

Abbildung 2.5: Schichtenmodell für 802.15.4 unter Linux

Schicht, über die Verwaltungsaufgaben wie Setzen der MAC-Adresse erfolgen, und eine Implementierung des 6LoWPAN-Protokolls. Jede dieser Schichten ist als eigenes Modul implementiert, welches auf der jeweiligen darunter liegenden Schicht aufbaut.

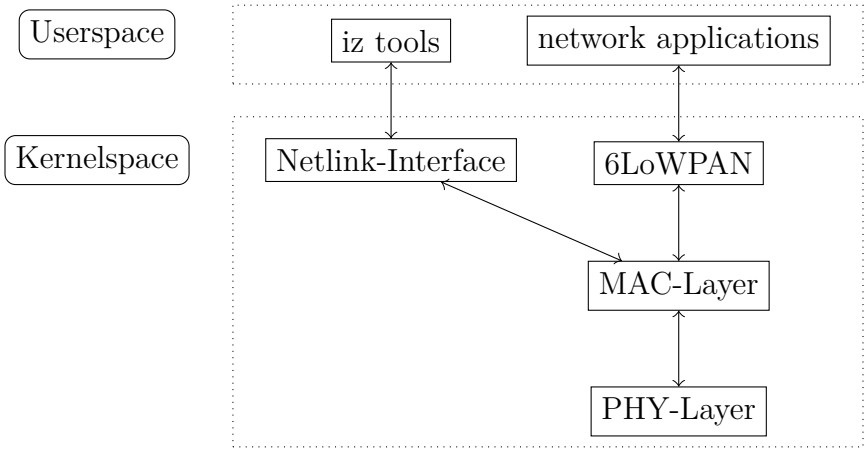

Abbildung 2.6: Unterteilung von 802.15.4 unter Linux in User- und Kernelspace

2.3 unterstützte Hardware

Derzeit gibt es Treiber für die folgenden drei Funkmodule:

- Atmel AT86RF230/231/233/212
- Microchip MRF24J40
- Texas Instruments CC2520

Diese werden direkt über Serial Peripheral Interface (SPI) angeschlossen. Das heißt, es werden direkt Befehle an den Funkchip gesendet und das Modul besitzt keine Firmware, die eine Schnittstelle zwischen den beiden Modulen bereitstellt.

Um die höheren Schichten zu testen, ohne echte Hardware einzusetzen, existiert zudem noch ein `fakelb`-Treiber. Mit diesem lassen sich ein oder mehrere virtuelle Funkmodule erstellen. Damit lassen sich dann auch Szenarien zwischen mehreren Knoten testen und durch die Funkübertragung entstandene Fehler können ausgeschlossen werden.

Weiterhin gibt es einen Treiber namens „serial“, der für Module verwendet wird, die einen eigenen Mikrocontroller besitzen und über die serielle Schnittstelle angesprochen werden können [LZ14]. Dafür wurde ein Protokoll entwickelt, über welches der Treiber mit dem Funkmodul kommuniziert. Auf dem Funkmodul muss deshalb eine Firmware installiert werden, die dieses Protokoll implementiert. Über diese Schnittstelle ist es vergleichsweise einfach, neue Funkmodule unter Linux zu verwenden. Ein Nachteil an diesem Treiber ist allerdings, dass er noch nicht im offiziellen Kernel enthalten ist und deshalb erst manuell auf diesen portiert werden muss. Weiterhin gibt es einen Entwurf für eine verbesserte Version des Protokolls zur Kommunikation mit dem Funkmodul, allerdings wurde dieser noch nicht als Treiber implementiert. In diesem gibt es zusätzliche Befehle und die Möglichkeit, Fehlercodes in der Antwort auf einen Befehl zu senden.

Das Protokoll besteht aus wenigen Befehlen, die alle synchron abgearbeitet und vom Kommunikationspartner mit einer Antwort bestätigt werden. Bis auf den

Befehl „Transmit Block“, welcher bei Empfangen eines Paketes aufgerufen wird, werden alle Befehle von Seite des Linux-Treibers aus an das Funkmodul gesendet. Die Befehle bestehen jeweils aus den zwei Startbyte „zb“, gefolgt von einer einen Byte langen Id, um den Befehl zu identifizieren. Anschließend kann abhängig vom Befehl eine beliebige Anzahl Bytes als Parameter für den Befehl folgen. Die Antwort beginnt wie der Befehl, nur dass das high bit in der Id invertiert ist. Nach der Id folgt hier der Status, welcher den Erfolg des Befehls angibt, und danach ggf. mit dem Befehl angeforderte Daten.

```
1 Befehl:  'z' 'b' 0x09 <len> <data * len>
2 Antwort: 'z' 'b' 0x89 <status>
```

Listing 2.1: Beispiel Serial-Protokoll

Die verfügbaren Befehle sind `Open` und `Close`, um die Verbindung zu initialisieren, bzw. zu beenden. Mit den Befehlen `Set Channel`, `Set PAN ID`, `Set Short Address`, `Set Long Address` können die entsprechenden Informationen an das Funkmodul übertragen werden und mit `Address` die MAC-Adresse ausgelesen werden. Weiterhin gibt es die beiden Funktionen `Transmit Block` und `Receive Block` mit denen Daten übertragen werden können.

In einer zweiten Revision des Protokolls [Che14] gibt es noch die beiden Befehle `Enable/Disable promiscuous mode`, was etwa nützlich ist, wenn das Gerät als Sniffer verwendet werden soll und `Enable/Disable hardware auto-acknowledgment`. Außerdem werden in dieser Version in den Antworten auf Befehle zusätzlich Fehlercodes ausgegeben, damit der Host weiß, weshalb ein Fehler aufgetreten ist und möglicherweise besser darauf reagiert werden kann.

2.3.1 Einbinden der Hardware

Die Funkmodule werden über die SPI-Schnittstelle mit dem Board verbunden. Da die Geräte über diese Schnittstelle keine Informationen zur Identifikation bei ihrem Host übermitteln, wie das etwa bei USB durch Hersteller- und Produkt-ID geschieht, muss dem System auf andere Weise mitgeteilt werden, dass ein Funkmo-

dul verbunden ist. Bis vor kurzem musste dafür bei jeder Änderung der Hardware das betroffene Kernelmodul neu erstellt werden, da diese Konfiguration nur über Einstellungen in Header-Dateien möglich war. Deshalb wurden Device Trees entwickelt und in den Kernel eingebaut. Dabei handelt es sich um ein Dateiformat zur Beschreibung der vorhandenen Hardware. Diese Datei wird dem Kernel beim Bootvorgang übergeben. Dadurch weiß der Kernel, welche Hardware angeschlossen ist und kann mit diesen Informationen die passenden Treiber laden.

Listing 2.2 zeigt ein Beispiel für einen Ausschnitt eines solchen Device Trees. Darin ist konkret der Teil zur Beschreibung des Atmel RF233 Funkmoduls zu sehen. Am Anfang wird mit `&spi` festgelegt, wo im Baum das Gerät eingehängt werden soll. Hier im Beispiel als Unterknoten des SPI-Controllers. Anschließend wird durch die status-Variable der Controller aktiviert. Darauf folgen Informationen über das Funkmodul. `compatible` gibt an, welcher Treiber zu laden ist, um das Gerät anzusprechen. `spi-max-frequency` gibt die maximale Geschwindigkeit für den Datentransfer über SPI an. Die restlichen Variablen geben an, an welchen Pins des SPI-Controllers die Leitungen für Interrupts, Reset des Moduls und Schlafmodus angeschlossen sind.

```
1  &spi {
2    status = "okay";
3    at86rf231@0 {
4      compatible = "atmel,at86rf233";
5      spi-max-frequency = <7500000>;
6      reg = <0>;
7      interrupts = <25 1>;
8      interrupt-parent = <&gpio>;
9      reset-gpio = <&gpio 23 0>;
10     sleep-gpio = <&gpio 24 0>;
11   };
12 };
```

Listing 2.2: Beispiel: Ausschnitt aus einem Device Tree

Speziell für den neuen Raspberry Pi B+ wurde ein System namens Hardware Attached on Top (HAT) [Ada14] entworfen, welches Entwickler von Zusatzhardware für den Raspberry Pi verwenden sollen. Die Besonderheit hierbei ist, dass

ein kompatibles Modul einen EEPROM-Speicher besitzt, in dem sich der Device Tree mit Informationen über das Board befindet. Der Raspberry Pi kann diese Informationen automatisch auslesen, weshalb es als Nutzer auch nicht mehr notwendig ist, selbst Device Tree-Dateien anzupassen. Dieses System wurde erst Ende Juli 2014 vorgestellt und derzeit gibt es ein paar wenige Module, die dieses System verwenden [Upt14]. Wenn es sich durchsetzt, könnte damit die Erweiterung des Raspberry Pi mit neuen Modulen um einiges vereinfacht werden, da weniger manuelle Konfiguration notwendig ist.

Und auch wenn keine nach HAT-Spezifikation gebauten Module verwendet werden, entsteht durch diese Entwicklung der Vorteil, dass nun auch der offiziell für den Raspberry Pi verwendete Kernel und Bootloader Device Trees unterstützen, was vorher nicht der Fall war.

2.4 Funktionsumfang des MAC-Stack

Die 802.15.4 MAC-Schicht lässt sich in zwei Teile gliedern: den MAC common part sublayer (MCPS) und den MAC sublayer management entity (MLME). Hiervon wurde der MCPS implementiert, da über diesen Teil die Übertragung von Daten abläuft.

Der MLME, welcher für Verwaltungsaufgaben benötigt wird, existiert als unfertige Implementierung, wurde aber bisher nicht in den offiziellen Kernel aufgenommen. Aufgaben dieses Teils sind etwa die Verwaltung und Vergabe von Kurzadressen für die Knoten des Netzes. Außerdem ist dort eine Implementierung des Beacon-Modus enthalten, in dem der Koordinator periodisch sogenannte Beacon-Frames als Broadcast an alle Teilnehmer sendet.

Eine weitere noch nicht implementierte Funktionen ist der sogenannte promiscuous Mode. Wenn dieser aktiviert ist, verwirft das Funkmodul nicht automatisch alle Pakete, die an einen anderen Empfänger gesendet wurden, sondern gibt sie an die nächsthöhere Schicht weiter. Dies ist zum Beispiel notwendig, um einen Sniffer

zu implementieren. 802.15.4 Funkmodule besitzen meist einen Befehl, um diesen Modus zu steuern, aber derzeit gibt es kein Interface, über welches der Nutzer den Modus regeln könnte.

Eine weitere Funktion, die sich noch nicht steuern lässt, ist das automatische Senden von ACK-Paketen. Dies ist standardmäßig aktiviert und hat den Vorteil, dass die Pakete schneller bestätigt werden können, da sie nicht erst vom Funkmodul auf den Linux-Host übertragen werden müssen. Allerdings kann es Fälle geben, in denen es nützlich ist, ein ACK manuell zu versenden. Ein Beispiel dafür ist die indirekte Übertragung. Bei dieser gibt es die Möglichkeit, das Vorhandensein von beim Koordinator bereitstehenden Daten in das ACK-Paket einzubinden. Dafür muss es möglich sein, auf dieses Paket zuzugreifen und es zu modifizieren.

Allerdings ist derzeit eine umfangreiche Überarbeitung des Treibers geplant, bei der unter anderem diese Funktionen hinzugefügt werden sollen und vorhandener Code und Interfaces überarbeitet werden.

Eine der neuesten Erweiterungen des Linux 802.15.4 MAC-Stack ist die Unterstützung von Verschlüsselung. Damit ist es möglich, die in 802.15.4 spezifizierte Methode zu verwenden, Pakete auf der MAC-Schicht mit AES zu verschlüsseln. Da damit eine einheitliche Verschlüsselungsmethode für 802.15.4 gegeben ist, haben die Funkmodule meist die Möglichkeit, Pakete in Hardware zu ver- und entschlüsseln. Dies ist besonders bei Sensorknoten nützlich, da kein zusätzlicher Speicher für eine Softwareimplementierung der Verschlüsselungsalgorithmen benötigt wird. Außerdem ist eine Hardwareimplementierung schneller und effizienter und spart somit auch Energie ein.

Auf der MAC-Schicht aufbauend wird üblicherweise 6LoWPAN-Protokoll verwendet. Deshalb wird die Implementierung dieses Protokolls unter Linux auch zusammen mit dem 802.15.4-Treiber verwaltet und entwickelt. 6LoWPAN ist ein Protokoll, um auch in Einsatzbereichen, bei denen es Knoten mit niedriger Übertragungsrate und wenig Speicher gibt, mit IPv6 arbeiten zu können. Das hat den Vorteil, dass auch bei umfangreichen Netzen jeder Knoten eine eigene öffentli-

che Adresse bekommen kann und keine Mechanismen wie NAT verwendet werden müssen.

Um vom Userspace aus Einstellungen an den Geräten vornehmen zu können, existieren verschiedene Tools.

```
#Anzeigen vorhandener Geraete
iz listphy
#Erstellen eines zu wpan-phy0 gehoerenden Device auf MAC-Schicht
iz add wpan-phy0
#Setzen von PAN ID, Kurzadresse und Funkkanal fuer wpan0
iz set wpan0 777 8001 11
```

Listing 2.3: Beispiel für die Verwendung von iz

Das wichtigste von ihnen ist `iz`. Damit können unter anderem vorhandene Geräte aufgelistet und aktiviert werden. Außerdem erfolgt über dieses Tool der Zugriff auf die MAC-Funktionen zum Beitreten und Verlassen eines Netzwerkes. Listing 2.3 zeigt einige Beispiele für die Verwendung.

Ein weiteres wichtiges Werkzeug ist der `izcoordinator`, welcher auf einem als PAN-Koordinator agierenden Gerät dauerhaft laufen muss. Darüber wird die Vergabe von Kurzadressen verwaltet und deren Assoziation zur entsprechenden vollständigen Adresse gespeichert. Zumindest in Verbindung mit dem offiziellen Kernel ist dieses Programm jedoch ohne jegliche Funktion, da die benötigten MAC-Funktionen nicht implementiert sind. Daher wird, selbst wenn ein Befehl empfangen wird, dieser verworfen und kommt nicht beim Koordinator an.

3 Untersuchung verschiedener Stromsparverfahren

Prinzipiell gibt es zwei verschiedene Methoden, um einen Sensorknoten stromsparend zu betreiben, aber trotzdem eine Funkkommunikation mit diesem zu ermöglichen.

Zum einen eine vom Sender initiierte Übertragung, bei der der Sender mehrmals die zu übertragenden Pakete sendet, bis der Empfänger das Ankommen der Nachricht bestätigt. Das mehrfache Senden von Paketen ist hierbei notwendig, da der Empfänger nicht dauerhaft empfangsbereit ist.

Die Alternative dazu ist eine vom Empfänger initiierte Übertragung. Bei dieser Methode ist jedem Knoten ein Koordinator zugeordnet. Dieser ist der zentrale Punkt des Netzes und speichert ankommende Nachrichten für alle ihm zugeordneten Knoten. Ein Knoten kann dann beim Koordinator anfragen, ob Nachrichten für ihn vorhanden sind.

3.1 ContikiMAC

Bei ContikiMAC [Dun11] handelt es sich um einen Radio Duty Cycling (RDC)-Algorithmus, der für das namensgebende Embedded-OS Contiki entwickelt wurde. Der Sender verschickt hierbei mehrfach die zu versendenden Daten, bis entweder der Empfang bestätigt wurde oder die maximale Zeitdauer für die Wiederholungen überschritten wurde.

Bei diesem Verfahren kann jeder Knoten die meiste Zeit seinen Funkchip deaktivieren, so lange er ihn in einem global festgelegten Intervall aktiviert und dann für eine bestimmte Zeit auf eingehende Nachrichten wartet. Standardmäßig beträgt das Intervall zum Überprüfen des Kanals 8 Hz und der Knoten misst den Kanal auf aktive Übertragungen mit zwei Clear Channel Assessments (CCAs). Zwischen diesen wird für t_c Zeiteinheiten gewartet.

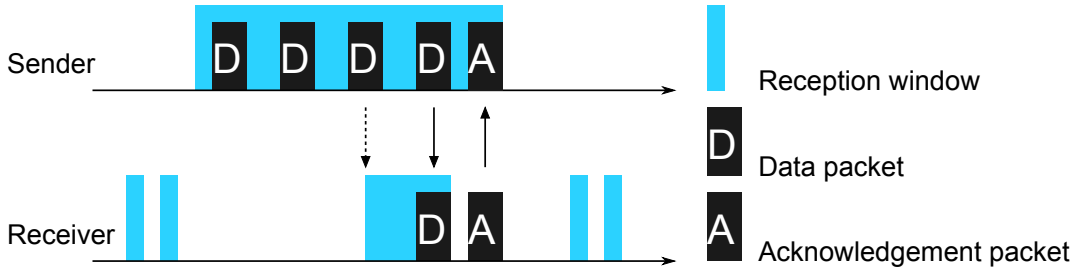

Abbildung 3.1: ContikiMAC

Wenn ein aktiver Sendevorgang detektiert wurde, bleibt das Funkmodul aktiv, bis der Sender das Paket erneut sendet. Damit wird sichergestellt, dass das komplette Paket empfangen wird, da der Empfangsvorgang möglicherweise erst nach Beginn des Paketes begonnen hat und somit Daten am Anfang fehlen würden. Nachdem das Paket empfangen wurde, überprüft der Empfänger, ob das Paket für ihn bestimmt war und bestätigt gegebenenfalls den Empfang. Der Sender wartet nach jedem gesendeten Paket für t_i auf eine Empfangsbestätigung und sendet keine weitere Wiederholung, wenn eine Bestätigung empfangen wurde. Wichtig ist, dass t_i größer als t_c ist, da sonst beide CCAs in den Bereich zwischen zwei übertragenen Paketen fallen könnten und damit die Übertragung nicht erkannt wird.

Unter ContikiMAC sind die Netzwerkknoten in einer Mesh-Topologie wie in Abbildung 3.2 zu sehen organisiert. Jeder Knoten ist dabei gleichberechtigt und kann mit jedem anderen Knoten direkt kommunizieren. Weiterhin kann jeder Knoten auch als Router agieren und damit Nachrichten weiterleiten, die nicht direkt übertragen werden können, weil etwa die beiden Kommunikationspartner zu weit voneinander entfernt sind.

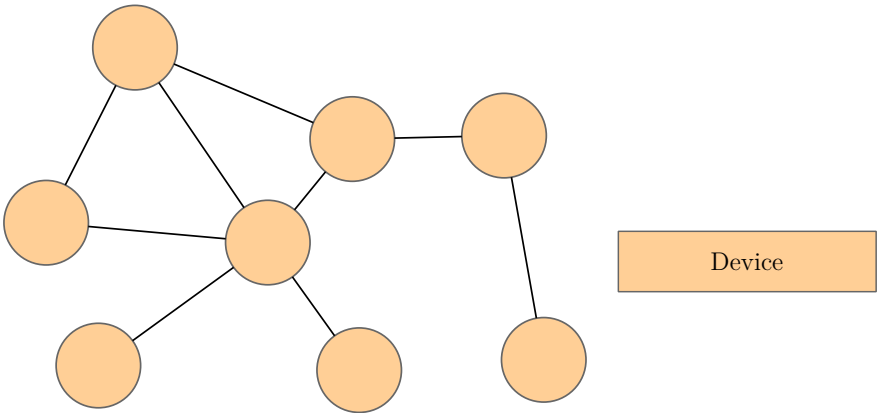

Abbildung 3.2: Mesh-Netztopologie

3.1.1 verwandte Verfahren

Ein mit ContikiMAC verwandtes Verfahren ist Low Power Listening [DM08], welches in TinyOS verwendet wird. Auch hier gibt es ein Intervall t_{dc} in dem die Knoten aufwachen und auf aktive Übertragungen warten. Die Anzahl der durchzuführenden CCAs ist bei diesem Verfahren nicht festgelegt, sondern kann vom Nutzer gewählt werden. Ebenso wie bei Contiki versucht der Sender die Übertragung maximal für die Dauer eines Intervalls t_{dc} zu wiederholen und wartet zwischen jedem Paket t_{IPS} Zeiteinheiten auf eine Empfangsbestätigung.

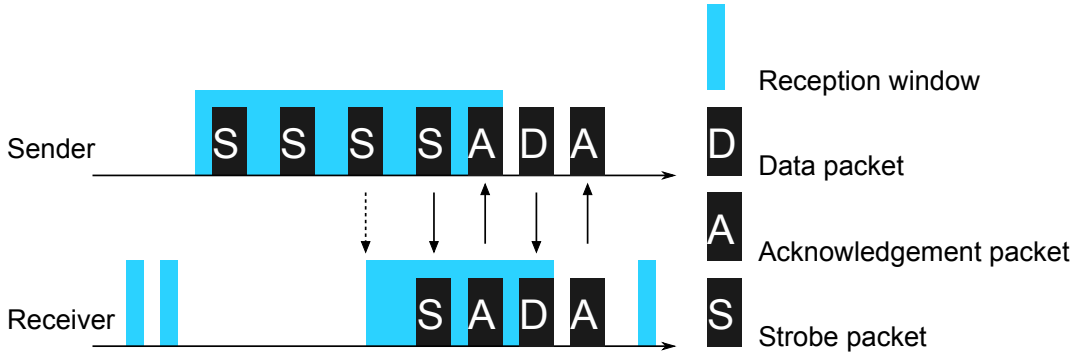

Abbildung 3.3: X-MAC

Ein weiteres Verfahren ist Contiki X-MAC. Dieses unterscheidet sich von ContikiMAC und LPL dadurch, dass der Sender statt den zu übertragenden Daten erst

sogenannte Strobe-Pakete überträgt, um den Empfänger über ein ausstehendes Paket zu informieren. In diesen Strobe-Paketen befindet sich die Empfängeradresse des zu sendenden Paketes, aber keine weiteren Daten. Dadurch bleiben die Strobe-Pakete unabhängig von den zu sendenden Daten immer gleich kurz. Für den Knoten, der die Nachricht erhalten soll ist dies vorteilhaft, da es weniger Verzögerung bis zum Senden der eigentlichen Daten gibt. Alle anderen Knoten können durch das kurze Paket schnell ermitteln, dass das Paket für einen anderen Empfänger bestimmt ist, und für die restliche Dauer der Periode wieder das Funkmodul deaktivieren.

Auch hier gibt es ein global festgelegtes Intervall, in dem die Knoten aufwachen und auf eingehende Daten warten. Dieses Intervall ist auch die maximale Zeit, für die die Strobe-Pakete übertragen werden, danach wird die Übertragung als fehlgeschlagen betrachtet.

Empfängt ein Knoten ein Strobe-Paket, welches für ihn bestimmt ist, so antwortet er mit einem Strobe-Acknowledgement und stellt sein Funkmodul auf Empfangsmodus. Der Sender übermittelt nach Erhalt des Strobe-Acknowledgement die eigentlichen Daten. Diese können dann wieder bestätigt werden, wenn dies angefordert wurde.

3.1.2 Interoperabilität

Verfahren wie ContikiMAC und TinyOS Low Power Listening wurden speziell für ein konkretes Betriebssystem entwickelt. Da sie nach keinem Standard implementiert sind, gibt es von System zu System Unterschiede, wodurch die Interoperabilität erschwert wird.

Da ContikiMAC und TinyOS LPL sich sehr ähnlich sind, wurde in [KTDT12] analysiert, ob es möglich ist, Knoten aus beiden Systemen in einem gemeinsamen Netz zu verwenden. Dabei wurde untersucht, wie sich eine Anpassung der beiden Parameter Aufwachintervall t_{dc} und Zeit zwischen zwei gesendeten Paketen t_{IPS} auf die Kommunikation auswirkt und ob ein idealer Wert für beide Systeme gefunden

werden kann. Die Standardwerte der beiden Algorithmen sind sehr unterschiedlich, etwa ist t_{IPS} bei ContikiMAC 0.4 ms lang und bei TinyOS LPL 8 ms.

Dabei wurde ermittelt, dass besonders der Wert von t_{IPS} eine entscheidende Rolle spielt und bei etwa 1.1 ms ein Idealwert gefunden wurde, bei dem wesentlich weniger Übertragungen, als bei jedem anderen höheren oder niedrigeren Wert notwendig waren. Ein weiteres in Experimenten ermitteltes Ergebnis war, dass es auch große Unterschiede macht, welcher der beiden Knoten Sender und Empfänger ist und sich dies auf den Duty Cycle beider auswirkt.

3.2 indirekte Übertragung

Bei der indirekten Übertragung handelt es sich um ein Verfahren, welches im IEEE 802.15.4-Standard [IEE11] beschrieben wird. Dabei werden Datenpakete nicht direkt vom Sender an den Empfänger übertragen, sondern erst im Sender zwischengespeichert, bis der Empfänger ausstehende Pakete anfordert. Die Idee hinter diesem Verfahren ist, dass Endknoten die meiste Zeit in einem Schlafmodus verbringen und daher auch keine Pakete empfangen können.

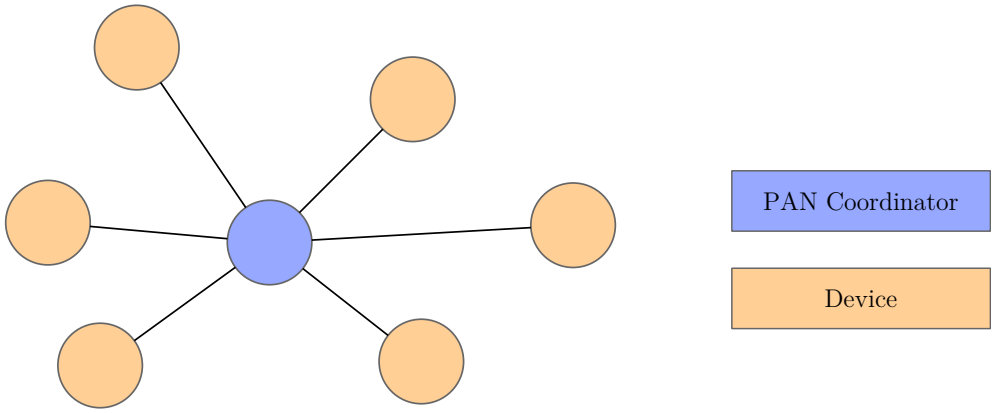

Abbildung 3.4: Stern-Netztopologie

Dieses Verfahren funktioniert nur in einer Stern-Netztopologie und die Kommunikation ist ausschließlich zwischen Knoten und dem Koordinator möglich. Der

Koordinator kann nicht direkt Daten an den Knoten senden, da nicht bekannt ist, wann dieser aktiv ist. Deshalb behält er zu sendende Pakete zusammen mit ihrer Ankunftszeit in einer Warteschlange. Die Ankunftszeit wird benötigt, da die Pakete laut Standard nur maximal für $maxTransactionPersistenceTime$ gespeichert bleiben dürfen, nach Ablauf dieser Zeit muss ein Timeout ausgelöst und dem nächsthöheren Layer mitgeteilt werden, dass die Übertragung fehlgeschlagen ist. Der Knoten kann Nachrichten an den Koordinator direkt senden, da dieser dauerhaft erreichbar ist.

Im Gegensatz zu Verfahren wie ContikiMAC ist auch kein Wissen über die Schlafzyklen der Knoten nötig. Außerdem können die Aufwachintervalle der einzelnen Knoten unterschiedlich sein und müssen auch nicht unbedingt periodisch stattfinden. Zu beachten ist allerdings, das Abfrageintervall kleiner zu wählen, als den Timeout der Warteschlange, da der Koordinator sonst Pakete verwerfen kann.

3.2.1 Beacon-Modus

In diesem Modus sendet der Koordinator periodisch Beacon-Pakete. In diesen befindet sich unter anderem eine Liste mit Zieladressen, für die der Koordinator Pakete in der Warteschlange besitzt. Wenn ein Knoten einen Beacon empfängt, der seine Adresse enthält, sendet er einen Data-Request-Befehl an den Koordinator, um die Daten zu erhalten. Diese Beacons werden immer in einem festen Intervall gesendet. Dieses Intervall kann in einem Bereich zwischen 15.36 ms und etwa 4 Minuten liegen.

Die Zeitdauer von Beginn eines Beacon bis zum Beginn des nächsten Beacon kann als sogenannter Superframe organisiert werden. Dieser Superframe kann aus vier verschiedenen Abschnitten bestehen.

- Den Beginn des Superframes markiert ein Beacon, der Informationen über das Netzwerk und die Struktur des Superframes enthält. Dieser wird als Broadcast an alle Teilnehmer der Netzwerkes gesendet.

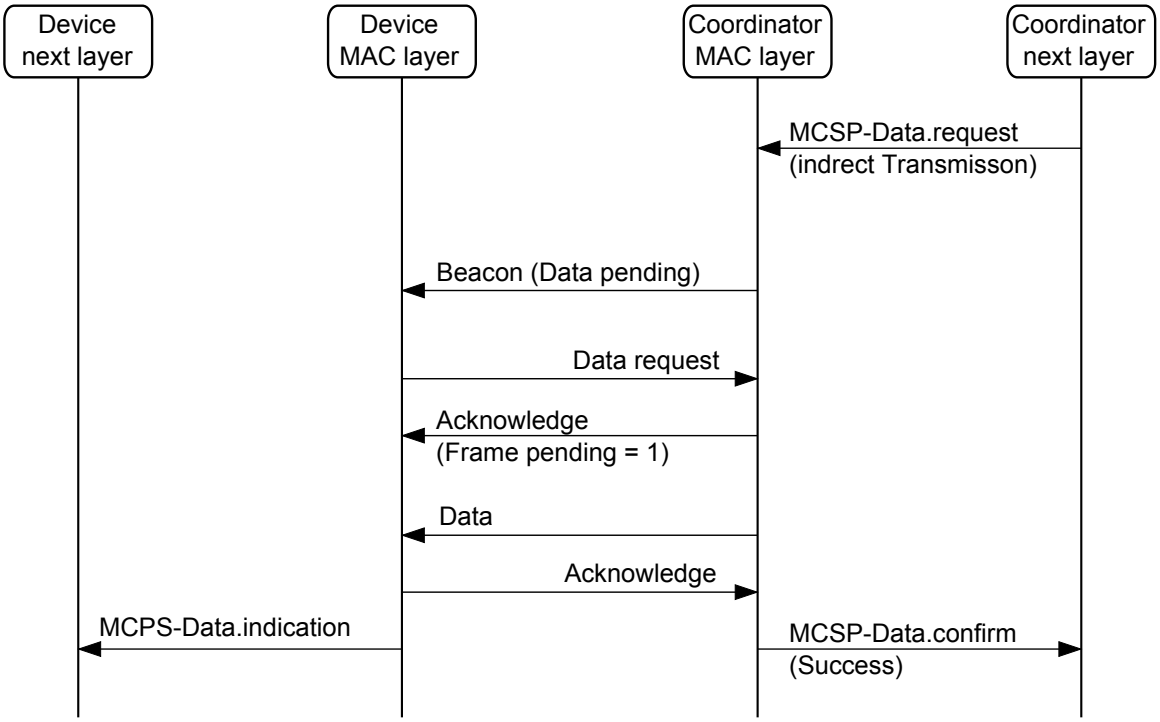

Abbildung 3.5: indirekter Transfer mit Beacons [Wik14, mit kleinen Änderungen]

- Anschließend folgt eine Contention-Access-Period (CAP). In diesem Zeitabschnitt darf jeder Knoten senden und mögliche Kollisionen werden mit dem Carrier sense multiple access with collision avoidance (CSMA/CA)-Algorithmus vermieden.

- Darauf kann eine Contention-Free-Period (CFP) folgen. Dieser Bereich ist in mehrere Abschnitte eingeteilt, die jeweils zu einem bestimmten Knoten gehören. Nur dieser eine Knoten darf in diesem Abschnitt Daten senden.

- Die restliche Zeit des Superframes ist die „Inactive Period“. In dieser Zeit findet keine Übertragung statt und sowohl Knoten als auch der Koordinator können sich im Stromsparmodus befinden.

3.2.2 beaconloser Modus

Hierbei muss jeder Knoten selbstständig den Zustand seiner Warteschlange abfragen. Dies geschieht, indem ein Data-Request-Befehl an den Koordinator gesendet wird. Dieser wird zunächst mit einem ACK bestätigt. Zu beachten ist hier, dass das ACK-Paket ein Feld „Frame Pending“ besitzt. Wenn dieses Feld auf 1 gesetzt ist, weiß der Knoten, dass der Koordinator im Anschluss an das ACK noch ein Datenpaket senden wird. Kann der Koordinator nicht schnell genug den Wert dieses Feldes ermitteln, so wird es immer auf 1 gesetzt. Anschließend sendet der Koordinator die Daten aus der Warteschlange oder ein leeres Datenpaket, wenn keine zu sendenden Daten existieren.

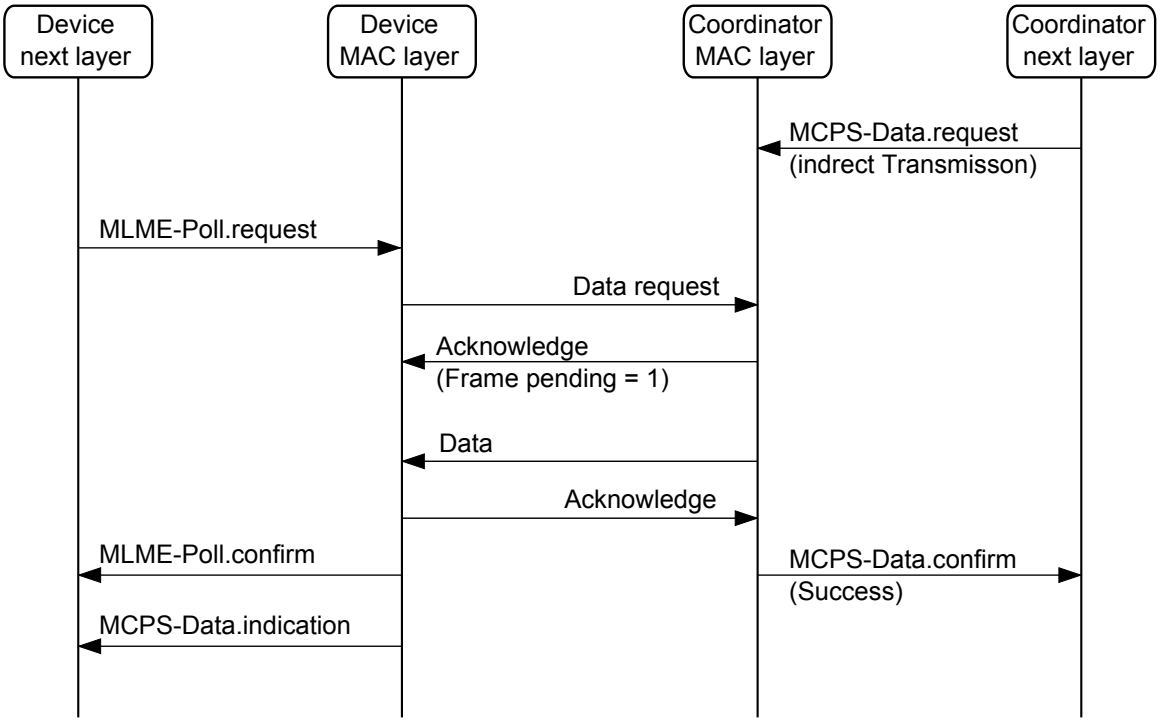

Abbildung 3.6: indirekter Transfer ohne Beacons [Wik14, mit kleinen Änderungen]

3.3 Beispielszenarien

In diesem Abschnitt werden verschiedene praktische Szenarien betrachtet. Es soll hierbei ermittelt werden, wie sehr sich die MAC-Verfahren für das jeweilige Szenario eignen. Zudem wird angenommen, dass als Protokoll auf der Anwendungsschicht CoAP verwendet wird. Da sich die Übertragungsverfahren in zwei Klassen einteilen lassen, werden exemplarisch die indirekte Übertragung des 802.15.4 Standard und ContikiMAC verglichen.

3.3.1 periodisch zu erfassender Sensor

In diesem Fall gibt es Sensorknoten, die zum Messen bestimmter Werte, wie etwa Temperatur oder Luftfeuchtigkeit, verwendet werden. Diese sind in einem bestimmten Intervall zu erfassen, um damit beispielsweise Statistiken zu erstellen, oder eine Aktionen auszulösen, wenn einer der gemessenen Werte eine Grenze überschreitet. Zum Beispiel könnte die Heizung aktiviert werden, wenn die Temperatur zu gering ist.

Auf Anwendungsschicht lässt sich dieses Problem sehr gut mit CoAP lösen, da es bei diesem Protokoll die Möglichkeit gibt, eine Ressource zu abonnieren und dann automatisch vom Server neue Werte der Ressource zu erhalten. Je nach Anwendungsfall kann der Server den Messwert periodisch oder auch nur bei einer Änderung des Wertes senden. Nachrichten vom Client zum Server werden in beiden Fällen nur zum Abonnieren und Kündigen der Ressource benötigt, was nur selten auftreten sollte.

Bei Nutzung der indirekten Übertragung kann hierbei ein großes Abfrageintervall im Minutenbereich oder höher verwendet werden, da außer Abos von Ressourcen oder Kündigungen dieser keine Nachrichten zu erwarten sind. Da diese Nachrichten nur bei Start und Beenden des Systems versendet werden, ist das zu erwartende Nachrichtenaufkommen zum Sensor sehr gering. Die Übertragung vom Sensorknoten zum Server wird durch das hohe Intervall nicht beeinträchtigt. Da

der Koordinator der einzige Punkt ist, mit dem der Sensorknoten kommunizieren kann und dieser dauerhaft aktiv sein muss, werden Nachrichten zum Koordinator direkt übertragen. Die für die indirekte Übertragung notwendigen Data-Requests zum Anfordern von beim Koordinator zwischengespeicherten Paketen können beispielsweise vor oder nach jedem übertragenen Sensormesswert gesendet werden.

Das zu wählende Abfrageintervall kann hier von jedem Teilnehmer des Netzes frei gewählt werden und muss nicht unbedingt für jeden Knoten gleich sein. Zu beachten ist allerdings, dass das Intervall kleiner ist, als die Zeitdauer, für die der Koordinator die ausstehenden Daten speichert, da sonst Pakete verloren gehen können.

Wird ContikiMAC für das gleiche Problem eingesetzt, so sollte beachtet werden, dass dieses Protokoll standardmäßig sehr kleine Intervalle zum Aufwachen der Knoten nutzt. Der Standardwert beträgt 8 Hz. Da unter Contiki alle Knoten als gleichberechtigt betrachtet werden, findet die Übertragung auf beiden Richtungen auf die gleiche Weise statt. Bei Knoten die nicht mit Batterien betrieben werden müssen, kann konfiguriert werden, dass das Funkmodul nicht deaktiviert wird, um Pakete schneller erhalten zu können.

Für Perioden im Bereich mehrerer Sekunden oder Minuten ist ContikiMAC nicht geeignet. Aufgrund des wiederholten Sendens, bis der Empfänger das Paket bestätigt, wird im schlimmsten Falle der Kanal für die komplette Dauer einer Periode mit Wiederholungen des Datenpaketes ausgelastet.

Ein ähnlicher Anwendungsfall wäre ein Sensor, der nur bei bestimmten Ereignissen ein Signal auslöst. Etwa ein Sensor, der das Öffnen von Tür oder Fenster registriert und als Teil einer Anlage zum Einbruchschutz eingesetzt wird. In diesem Fall ist der Betrieb als indirekter Knoten empfehlenswert. Da diese Ereignisse nur sporadisch auftreten, muss der Sensor entsprechend auch nur sehr selten Daten übertragen und kann für lange Zeit inaktiv bleiben.

3.3.2 Aktor, Licht über Web schalten

In diesem Beispiel soll ein Knoten betrachtet werden, der als Aktor eingesetzt wird und dafür auf eingehende Signale reagieren muss. Dies könnte etwa eine Lampe sein, die Befehle zum An- und Ausschalten, Dimmen oder Ändern der Farbe empfangen kann. Dabei ist die Reaktionszeit auf ein empfangenes Signal ein entscheidender Faktor und sollte möglichst gering sein.

Der Einsatz der indirekten Übertragung ohne Beacons ist für diesen Anwendungsfall weniger geeignet. Um in diesem Modus eine geringere Latenz zu erreichen, muss der Knoten öfter Data-Requests an den Koordinator senden. Zusätzlich dazu steigt die Anzahl der gesendeten Data-Requests auch noch mit der Anzahl der im Netz vorhandenen Knoten. Damit würden bei einem Einsatz mit dieser Methode sehr viele unnötige Anfragen versendet, obwohl tatsächlich zu übertragende Nutzdaten nur selten auftreten.

Eine Verbesserung wäre in diesem Falle die Nutzung von Beacons. Der Vorteil hierbei ist, dass die Knoten selbst keine Anfragen mehr senden müssen. Es genügt, die im Beacon enthaltenen Informationen auszulesen, und anschließend einen Data-Request zu senden, wenn im Beacon bereitstehende Daten angekündigt wurden. Die Netzlast verringert sich somit, da nur noch der Koordinator periodisch Beacons versenden muss und die Anzahl dieser unabhängig von der Anzahl der Knoten im Netz ist. Die Knoten müssen nur noch periodisch empfangsbereit sein, anstatt Daten zu senden, was den Energieverbrauch auch etwas verringert.

Ähnlich verhält es sich bei ContikiMAC. Auch hier müssen die Knoten periodisch empfangsbereit sein. Statt den Beacons werden hier die zu sendenden Nutzdaten übertragen oder es findet keine Übertragung statt. Damit ist mit diesem Verfahren bei niedriger Anzahl von Nutzdaten die Netzauslastung am geringsten, denn es werden nur Daten übertragen, wenn dies wirklich notwendig ist. Weiterhin kann abhängig von der gewählten Periode die Latenz sehr gering sein.

3.3.3 örtlich stark verteiltes Netz

Bei einem Netz, dessen Knoten über größere Distanzen verteilt sind, gibt es weitere Probleme, die zu beachten sind. Da durch die begrenzte Übertragungsreichweite möglicherweise nicht jeder Knoten direkt zu dem gewünschten Kommunikationspartner Pakete übertragen kann, muss es einen oder mehrere Knoten zwischen den beiden geben, welche die Pakete weiterleiten.

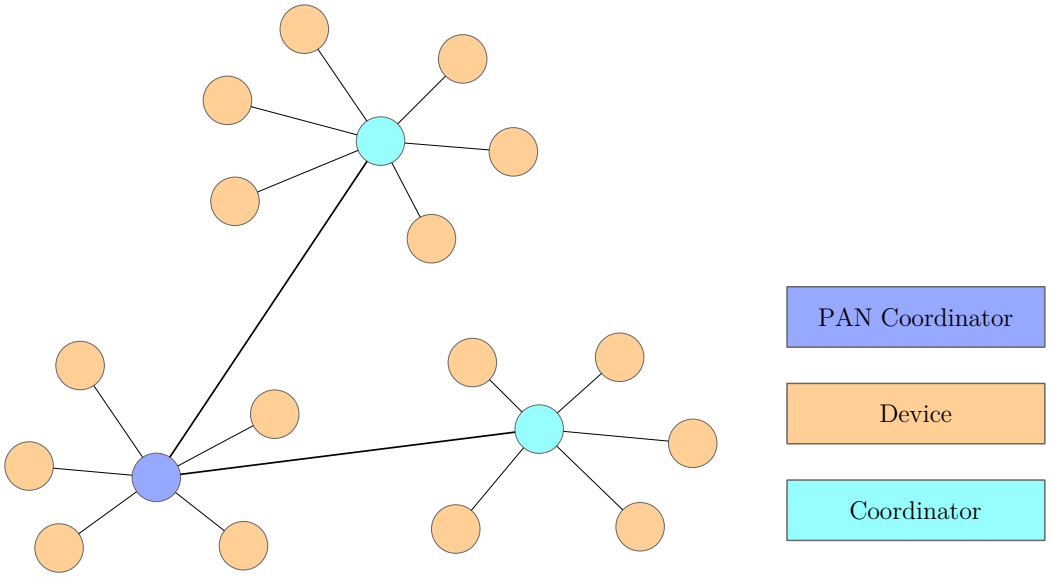

Abbildung 3.7: Cluster Stern-Netztopologie

In 802.15.4 gibt es dafür die Cluster Stern-Netztopologie [GWCB11, S. 36], wie in Abbildung 3.7 zu sehen. In dieser Topologie gibt es zusätzlich zu einem PAN-Koordinator mehrere diesem untergeordnete Koordinatoren. Die Koordinatoren sind alle Full Function Devices und können untereinander kommunizieren. Die Endknoten, die als Reduced Function Device betrieben werden, verbinden sich zu einem Koordinator in ihrer Nähe und verwenden diesen dann als einzigen Kommunikationspartner.

Bei ContikiMAC ist jeder Knoten gleichberechtigt und kann auch als Router agieren und Nachrichten weiterleiten. Dadurch ist es nicht notwendig, zusätzliche Netzknoten, die nur als Router agieren zum Netz hinzuzufügen. Allerdings ist zu be-

achten, dass dadurch die Latenz bei Übertragungen über mehrere Knoten schnell ansteigen kann, da die Übertragung in jedem Schritt mehrmals wiederholt werden muss. Eine Möglichkeit dem entgegen zu wirken ist es, doch wieder Knoten mit fester Stromversorgung in das Netz aufzunehmen und bei diesen den Stromsparmodus zu deaktivieren, so dass der Knoten dauerhaft empfangsbereit ist. Damit kommen zumindest Übertragungen zu diesen Knoten beim ersten Versuch an.

3.4 Ergebnisse

Es wurde gezeigt, dass die untersuchten Verfahren sich je nach Einsatzgebiet sehr unterschiedlich verhalten. Für die in den nächsten Kapiteln folgende Implementierung eines Verfahrens wurde die Methode der indirekten Übertragung gewählt. Dafür sprechen folgende Gründe:

Besonders im häufigsten Fall, in dem der Knoten nur ein einfacher Sensor ist, der nur zu bestimmten Zeitpunkten Daten sendet und nur sehr selten empfangen muss, zeigt die Nutzung der indirekten Übertragung ihre Vorteile. Wenn der Knoten seine Data-Requests immer vor oder nach dem Senden neuer Messwerte durchführt, sind dafür keine zusätzlichen Aufwachvorgänge nötig. Nur die Dauer des Wachzustandes wird etwas verlängert. Damit ist die Aufwachperiode nur von den zu übertragenden Messwerten abhängig und kann somit nicht mehr weiter verringert werden, ohne dabei auch die Anzahl der Messungen zu verringern.

Bei Knoten, die mit geringer Latenz Befehle empfangen können sollen, ist ein Verfahren wie ContikiMAC zwar besser geeignet, durch Nutzung des Beaconmodus lässt sich aber auch mit der indirekten Übertragung ein annähernd gleich gutes Verhältnis aus Stromsparen und geringer Latenz erreichen. Wenn der zu steuernde Knoten beispielsweise eine Lampe ist, ist es außerdem sowieso naheliegender, den Knoten auch gleich mit deren Energieversorgung zu betreiben. Damit kann der Knoten dauerhaft aktiv bleiben und kann so noch zusätzlich die Aufgabe als Router übernehmen.

Ein weiterer Vorteil ist, dass dieses Verfahren standardisiert ist. Damit hat es auch höhere Chancen, offiziell in den Linux-Kernel aufgenommen zu werden, als etwa eine Implementierung von ContikiMAC.

4 Entwurf

Der Entwurf setzt sich, genauso wie die Implementierung, aus den folgenden zwei Teilproblemen zusammen:

1. Entwicklung einer Firmware für das RaspBee-Funkmodul
2. Erweiterung des Linux-Moduls, um die indirekte Übertragung zu unterstützen

4.1 RaspBee-Firmware

Um das RaspBee Modul nutzen zu können, war es nötig eine neue Firmware dafür zu implementieren. Diese realisiert das Protokoll zur Kommunikation mit dem serial-Linuxtreiber.

Zuerst wurde ein passendes Framework ausgewählt, um die Implementierung zu vereinfachen. Ein weiterer Vorteil eines Frameworks anstelle der Verwendung plattformspezifischer Funktionen ist es, dass die Implementierung dadurch portabel wird und auch auf anderen Modulen genutzt werden kann, sofern das entsprechende Framework auch zur Verfügung steht.

Zur Auswahl standen TinyOS, Contiki, der offizielle Atmel MAC-Stack und µRacoli. Letztendlich fiel die Wahl auf µRacoli. Im Folgenden sollen die Vor- und Nachteile der verschiedenen Systeme erläutert werden, welche zur Wahl von µRacoli geführt haben.

TinyOS ist ein leichtgewichtiges Betriebssystem für Mikrocontroller und für verschiedene Modelle der Hersteller Atmel und Texas Instruments verfügbar. Außerdem ist eine vollständige Implementierung des 802.15.4 MAC-Stacks enthalten. Allerdings ist es ein umfangreiches und entsprechend komplexes System und verwendet zur Implementierung eine eigene Sprache. Diese orientiert sich an C, wurde aber stark erweitert. Dadurch wäre bei einer Implementierung mit TinyOS aufgrund fehlender Erfahrung mit diesem System noch Mehraufwand für die Einarbeitung in das System entstanden, weshalb sich gegen TinyOS entschieden wurde.

Contiki ist vergleichbar komplex wie TinyOS, nutzt aber Standard-C. Auch hier gibt es einen fertigen 802.15.4-Stack, welcher zwar nicht vollständig ist, aber die fehlenden Funktionen wären ohnehin nicht relevant, da diese auf Linuxseite implementiert werden. Da jedoch im Forschungsprojekt schlechte Erfahrungen bei der Verwendung von Contiki auf Atmel-Hardware gesammelt wurden, fiel die Entscheidung gegen Contiki. Der Grund dafür ist, dass die Implementierung der Atmel-Plattform in Contiki selten gewartet wird und damit oft Fehler enthält, die behoben werden müssen, um das System überhaupt nutzen zu können.

Der offizielle MAC-Stack von Atmel ist mit allen Atmel-Chips kompatibel und bietet mehrere Abstraktionsebenen, um auf die Funkhardware zuzugreifen. Dieser ist zwar kostenlos verfügbar, steht aber unter einer proprietären Lizenz. Dies sprach als Grund gegen den Atmel-Stack.

Bei µRacoli handelt es sich um ein sehr leichtgewichtiges Framework für diverse Atmel-Chips. Der Fokus liegt auf einer Abstraktion der Netzwerkfunktionen und der seriellen Kommunikation zwischen Host und Mikrocontroller. Damit bleibt das System relativ simpel, bietet aber trotzdem alle für die Entwicklung des Treibers nötigen Funktionen. Weiterhin ist µRacoli freie Software, die unter der BSD-Lizenz steht, weshalb es kaum Einschränkungen bei der Nutzung oder Verbreitung der damit entwickelten Firmware gibt und auch eine kommerzielle Nutzung möglich ist.

4.2 indirekter Modus

4.2.1 Grundlagen zum Linux-Kernel

Der Linux-Kernel ist ein sehr komplexes System und das zeigt sich auch in seinem Entwicklungsmodell. Es existieren verschiedene Branches des Kernels, die entweder zum Entwickeln und Testen neuer Funktionen genutzt werden oder als stabile Version gedacht sind. Eine stark vereinfachte Hierarchie der Branches am Beispiel des 802.15.4 Treibers ist in Abbildung 4.1 zu sehen. Hierbei zeigt die Pfeilrichtung den Weg an, den neue Funktionen gehen, bis sie in einer für den Anwender nutzbaren Kernelversion enthalten sind. Zudem gibt es noch weitere Branches, in denen keine neuen Funktionen entwickelt, sondern nur Fehler behoben werden. Diese sind der Übersichtlichkeit halber nicht in der Grafik enthalten.

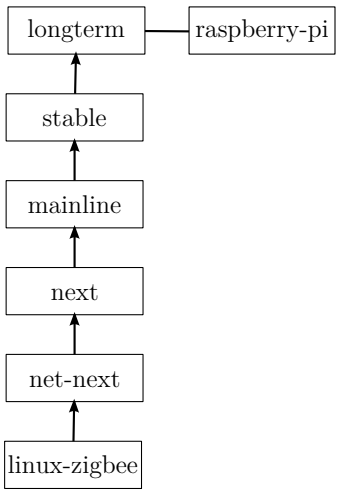

Abbildung 4.1: Hierarchie der Linux-Branches

Der raspberry-pi-Branch wird direkt von Entwicklern der Raspberry Pi Foundation betrieben und basiert auf dem longterm-Branch. Zusätzlich sind darin diverse Patches enthalten, die plattformspezifische Änderungen implementieren und für eine bessere Unterstützung der Hardware sorgen als der offizielle Kernel. Anfangs war dies notwendig, da passende Treiber nicht im offiziellen Kernel enthalten waren und der Raspberry Pi nicht genutzt werden konnte. Seit Anfang 2014 [EW14]

sind auch im offiziellen Kernel Treiber enthalten, um die USB-, HDMI- und SD-Kartenanschlüsse zu nutzen. Der aktuelle Status zum Fortschritt der Implementierung des offiziellen Kernels wird unter [RW14] dokumentiert.

Zum Implementieren von neuen Funktionen am 802.15.4-Treiber eignet sich nur der linux-zigbee-Branch. Dieser wird von den Entwicklern des Treibers betrieben und enthält immer den aktuellsten Code. Ein Problem dabei ist, dass die Patches für den Raspberry Pi dafür nicht verfügbar sind. Deshalb wurde versucht, die Patches zu portieren. Aufgrund der Versionsunterschiede zwischen den beiden Branches konnten die Patches nicht direkt übernommen werden, sondern mussten an einigen Stellen angepasst werden. Die Portierung war zwar erfolgreich, wurde wegen des großen Aufwands aber wieder verworfen, da dieser bei einem Update von Raspberry Pi Kernel oder dem 802.15.4-Treiber erneut entstehen würde.

Parallel dazu wurde die Möglichkeit untersucht, den Raspberry Pi mit dem unmodifizierten linux-zigbee-Kernel zu betreiben. Eines der Hindernisse dabei war, dass in dieser Version zwingend mit den in Abschnitt 2.3.1 beschriebenen Device Trees gearbeitet werden muss. Der Bootloader des Raspberry Pi unterstützt diese allerdings nicht. Deshalb war es zunächst notwendig den alternativen Bootloader u-boot[1] zu installieren, welcher dann den Linux-Kernel lädt.

4.2.2 Netzwerkmanagement

Die Funktionalität zum Verwalten des Netzwerkes ist im aktuellen Kernel nicht vorhanden, aber in einer älteren Entwicklerversion[2]. Deshalb wurden für die indirekte Übertragung notwendige Funktionen daraus übernommen und portiert. Benötigt wird der Associate Request, welcher einer der MAC Kommandos ist. Mit diesem kann sich ein Knoten beim Koordinator als Teilnehmer des Netzes anmelden und eine Kurzadresse anfordern.

[1] `http://elinux.org/RPi_U-Boot`
[2] `http://sourceforge.net/p/linux-zigbee/kernel/ci/devel/tree/`

Es gibt noch weitere Funktionen, die für eine vollständige Implementierung der MAC-Schicht notwendig wären, diese wurden aber weggelassen, da sie für das Testen der indirekten Übertragung nicht direkt erforderlich sind. Zudem wird daran derzeit von einem der Hauptentwickler des Treibers gearbeitet, weshalb diese Funktionen in Zukunft offiziell verfügbar sein sollten.

Versendet wird dieser Befehl, wenn das Userspace Tool `iz` mit dem Parameter assoc und der PAN Id und dem Funkkanal des Netzes aufgerufen wurde. Der Koordinator antwortet dann mit einer Kurzadresse für den Knoten.

4.2.3 Koordinator

Im Userspace gibt es ein Tool `izcoordinator`, welches die Vergabe von Kurzadressen verwaltet. Wenn dieses aktiviert wird, sollen Übertragungen indirekt stattfinden, also intern zwischengespeichert werden, bis sie von dem Empfänger abgeholt werden. Diese werden in einer doppelt verkettete Liste sortiert nach Ankunftszeit gespeichert.

Da die Pakete nur eine bestimmte Zeit in der Liste bleiben sollen und die Übertragung danach als fehlgeschlagen gelten soll, muss dafür eine Lösung gefunden werden. Die einfachste Möglichkeit ist es, bei jeder Änderung an der Liste, also entweder beim Eintragen eines neu angekommenen Paketes oder Entfernen eines abgeholten, alle Einträge am Anfang der Liste zu entfernen, bei denen der Timeout überschritten ist. Um Einträge unabhängig von Listenänderungen zu entfernen, müsste ein Timer oder ähnlicher Mechanismus pro Listeneintrag verwendet werden, um nach Ablauf des Timeouts den Eintrag zu löschen.

Die Suche nach einem Paket wird bei jedem empfangenen Data-Request ausgelöst. Diese Anfrage wird zunächst mit einem ACK bestätigt, in dem das Frame Pending Bit auf 1 gesetzt ist, um ein verfügbares Paket anzuzeigen. Dieser Wert ist bei der Implementierung immer 1, da nicht schnell genug festgestellt werden kann, ob es wirklich ein Paket für den Knoten gibt. Im Anschluss folgen entweder die ange-

forderten Daten oder ein Datenpaket mit leerer Payload, welches dem Empfänger mitteilt, dass es keine Daten für ihn gibt.

4.2.4 Knoten

Die Funktionalität, als indirekt arbeitender Knoten zu arbeiten, sollte in den Treiber überwiegend zum Testen des Koordinators integriert werden. Da der Stromverbrauch eines Raspberry Pi oder ähnlicher Hardwareplattformen wesentlich höher ist als der eines Mikrocontrollers, ist ein Batteriebetrieb über längere Zeit nicht möglich. Auf [Ake13] wurde die mögliche Laufzeit im Batteriebetrieb untersucht. Als Ergebnis wurde ermittelt, dass selbst mit dem sparsameren Raspberry Pi Modell A und Hardwaremodifikationen nur eine Laufzeit von knapp über einem Tag möglich war. Dafür werden 6 AA Batterien benötigt.

Im Userspace muss für den Knoten das `iz` Tool um einen Parameter zum Auslösen des POLL-Befehls erweitert werden, damit dieser auch vom Nutzer ausgeführt werden kann. Von diesem wird dann die eigentliche Implementierung des Befehls auf der MAC-Schicht aufgerufen. Für den Empfang der mit dem Data-Request angeforderten Daten sind keine besonderen Änderungen notwendig. Auch das Senden von Datenpaketen an den Koordinator kann unverändert ausgeführt werden.

5 Implementierung

In diesem Kapitel soll die konkrete Implementierung der Firmware und der indirekten Übertragung erläutert sowie dabei aufgetretene Probleme und deren Lösungen gezeigt werden.

5.1 RaspBee-Firmware

Die Firmware kommuniziert über die serielle I²C-Schnittstelle mit dem Linux-Treiber. Dabei wird in der Hauptschleife der Firmware entweder auf einen eingehenden Befehl oder ein empfangenes Paket gewartet. Beim Funkmodul ist hierbei der automatische Modus für Empfang und Senden von Paketen eingestellt. Dieser bewirkt, dass ACKs für empfangene Pakete automatisch gesendet werden (`RX_AACK_ON`). Bei zu sendenden Paketen wird ein CCA aufgeführt, um zu überprüfen, ob der Kanal frei ist. Die Übertragung wird automatisch wiederholt, wenn kein ACK empfangen wurde (`TX_ARET`). Da dies alles von dem Funkmodul erledigt wird, kann damit auch der Mikrocontroller entlastet werden.

Die implementierte Firmware nutzt als Vorlage die von Mariano Alvira für den Redbee Econotag entwickelte Firmware[1]. Dies war bis jetzt das einzige Modul, für welches die Schnittstelle zum serial-Treiber implementiert wurde.

[1] `https://github.com/malvira/linux802154-serialdev`

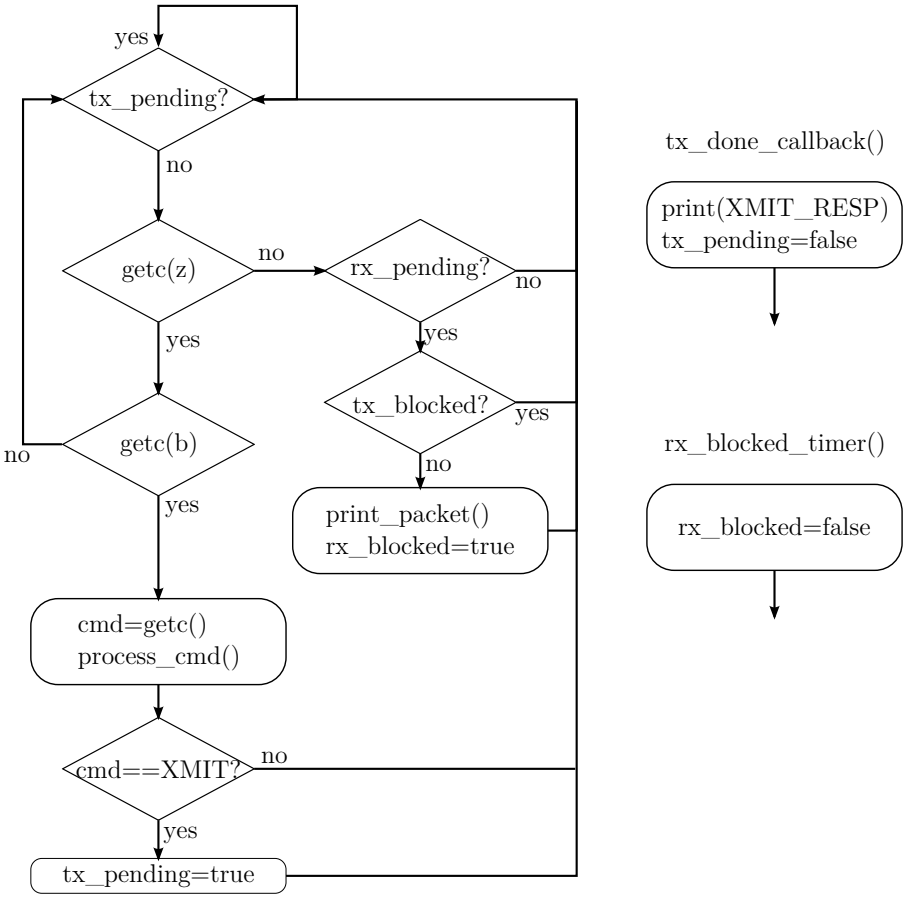

Abbildung 5.1: Ablauf auf der RaspBee Firmware

5.1.1 Paketempfang

Die vom Funkmodul empfangenen Pakete werden in einem Array vom Datentyp buffer_t gespeichert. Dieser Datentyp gehört zu µRacoli und nutzt die Macros `BUFFER_IS_LOCKED()` und `BUFFER_SET_LOCK()`. Mit `BUFFER_SET_LOCK()` wird der Puffer nach dem Empfang des Pakets als belegt markiert und in der Hauptschleife kann dann überprüft werden, in welchem Puffer sich Daten befinden. Nachdem ein Puffer an den Linux-Host übergeben wurde, bricht die Schleife ab und es wird zunächst wieder auf eingehende Befehle gewartet.

Intern wird bei jedem empfangenen Paket ein Interrupt ausgelöst. µRacoli bietet die Möglichkeit an, eine Funktion mit dem Namen `usr_radio_receive_frame()` festzulegen, die am Ende dieses Interrupts aufgerufen wird und als Parameter das empfangene Paket und Metadaten wie den LQI-Wert und die CRC-Prüfsumme erhält. In dieser Funktion wird das empfangene Paket dann in einen freien Puffer geschrieben oder verworfen, wenn kein Platz mehr vorhanden ist. Die Id des zu verwendenden Puffers ist eine statische Variable und nach jedem Aufruf um Eins erhöht bzw. wieder auf Null gesetzt. Somit liegen die Pakete immer in der Empfangsreihenfolge in den Puffern. Der Suchaufwand nach einem freien Puffer ist in diesem Fall auch nur $O(1)$. Denn wenn der verwendende Puffer bereits belegt ist, kann angenommen werden, dass dies der erste in der Liste ist und somit alle anderen Plätze auch belegt sind.

Beim Testen der Firmware zeigte sich, dass es zu fehlerhaften Übertragungen kommt, wenn mehrere Pakete direkt hintereinander von der Firmware an den Host gesendet werden. Dafür verantwortlich ist ein Fehler im Linux-Treiber. Der Treiber überprüft zwar, beim Versuch, einen Befehl zu senden, ob ein möglicherweise vorher gesendeter Befehl noch nicht abgeschlossen ist, aber ob gerade ein Datenpaket empfangen wird, überprüft der Treiber nicht. Dadurch ist es teilweise aufgetreten, dass Befehle abgesendet wurden, während Daten empfangen werden. Der Treiber kann die Daten dann nicht mehr richtig zuordnen und es kam zum Beispiel vor, dass sich im empfangenen Datenpaket auch die Antwort auf den danach ausgeführten Befehl enthalten war. Listing 5.1 zeigt solch ein empfangenes Paket. Beim

markierten Teil handelt es sich hierbei um Antworten auf zwei Befehle, die nicht Teil des Paket sein sollten.

```
0000   7b 33 3a 87 00 9e a4 00 00 00 00 fe 80 00 00 00  {3:.............
0010   00 00 00 02 21 2e ff 7a 62 89 00 7a 62 8b ff 58  ....!..zb..zb..X
0020   61 cc 6b fe ca cd 47 00 ff ff 2e                 a.k...G....
```

Listing 5.1: Fehlerhaft empfangenes Datenpaket

Es wurde versucht, den Fehler im Linux-Treiber zu finden und zu beheben, allerdings ohne Erfolg. Die Fehlersuche wurde dadurch erschwert, dass der Fehler zeitabhängig ist und somit nicht immer auftritt.

Als Workaround wurde deshalb die RaspBee-Firmware so implementiert, dass nach jedem an den Host gesendeten Paket mittels eines Timers das Senden weiterer Pakete an den Host blockiert wird. In dieser Zeit ist es nur möglich, Befehle vom Host an die Firmware zu senden. Die Idee dahinter war, dass auf empfangene Pakete üblicherweise eine Antwort kommt und für diese Antworten ein kurzer Zeitraum freigehalten wird. Beim Testen der Firmware traten daraufhin keine Probleme mehr auf.

5.1.2 Senden von Paketen

Der zweite grundlegende Befehl in der Firmware ist das Senden von Daten an einen anderen Netzwerkteilnehmer. Dafür wird der Befehl `DATA_XMIT_BLOCK`, die Anzahl zu sendender Bytes und darauf folgend die eigentlichen Daten an die Firmware übertragen und zur Weiterverarbeitung in einen Puffer geschrieben. Im Anschluss wird die Variable `tx_pending` auf `true` gesetzt, um einen laufenden Sendevorgang zu signalisieren und danach die Übertragung gestartet. Die Hauptschleife wartet anschließend darauf, dass der Wert von `tx_pending` wieder auf `false` gesetzt wird, damit die Ausführung fortgesetzt werden kann.

Bei Beendigung des Sendevorganges wird eine vorher festgelegte Callback-Funktion aufgerufen. In dieser wird zum einen die Antwort für den XMIT-Befehl zurück

an den Host gesendet, um ihn über den Erfolg der Übertragung zu informieren. Weiterhin wird auch `tx_pending` wieder zurückgesetzt, damit auch die Firmware weiter ausgeführt werden kann.

5.1.3 weitere Befehle

- `OPEN` und `CLOSE` dienen zum Starten und Deaktivieren des Funkmoduls. `OPEN` initialisiert das Funkmodul mit der im EEPROM gespeicherten MAC-Adresse. Diese Adresse befindet sich bereits im Auslieferungszustand im Speicher und wird von der offiziell für das Modul verfügbaren ZigBee-Firmware verwendet. Daher ist es für den Anwender nicht notwendig, die Adresse manuell festzulegen. Anschließend wird der Empfangsmodus aktiviert. `CLOSE` versetzt das Funkmodul wieder in den Schlafmodus.

- `SET_CHANNEL`, `SET_PAN_ID`, `SET_SHORT_ADDRESS` und `SET_LONG_ADDRESS` lesen den jeweiligen übergebenen Wert ein und schreiben ihn in die entsprechenden Register des Funkmoduls. Das Modul benötigt diese zum Beispiel, um zu wissen, welche Pakete es mit einem automatischen ACK beantworten muss.

- Der `ADDRESS`-Befehl liest die MAC-Adresse auf dem EEPROM und übergibt sie an den Host, damit sie auch dort genutzt werden kann, ohne manuell eingegeben werden zu müssen.

5.2 indirekte Übertragung

5.2.1 Koordinator

Die Implementierung der indirekten Übertragung besteht aus dem Eintragen zu sendender Pakete in eine Liste und dem Auslesen dieser, wenn ein Knoten Daten anfordert.

Das Eintragen in die Liste erfolgt in der Funktion `mac802154_wpan_xmit()`. Diese Funktion wird für jedes zu sendende Paket aufgerufen und löst normalerweise den Sendevorgang aus. Diese wurde durch eine Abfrage erweitert, ob die indirekte Übertragung verwendet werden soll. Dies gilt in der aktuellen Implementierung für Pakete an jeden Empfänger. Wenn in dem Netz auch direkt ansprechbare Knoten verwendet werden sollen, funktioniert die aktuelle Implementierung nicht. Dafür müssten beim Koordinator Informationen über die Art der einzelnen Netzknoten vorhanden sein.

Sowohl die Variable, ob der indirekte Modus verwendet werden soll, als auch die Liste mit den gespeicherten Paketen werden in einer Datenstruktur vom Typ `mac802154_sub_if_data` gespeichert. Darin sind bereits Informationen über das Funkmodul enthalten, die vom MAC-Treibers benötigt werden. Eine Instanz dieser Struktur ist über die gesamte Laufzeit des Treibers verfügbar.

Die zwischengespeicherten Pakete werden in eine doppelt verkettete Liste abgelegt. Diese wurde gewählt, da so auch aus der Mitte der Liste Einträge entfernt werden können, ohne dass Lücken entstehen. Außerdem ist dies eine der wenigen komplexen Datenstrukturen, die es standardmäßig im Linux-Kernel gibt.

Laut Standard ist die Größe der Liste und die maximale Zeit, die ein Paket gespeichert wird, begrenzt. Zur Vereinfachung der Implementierung werden diese Werte beide vorerst als unbegrenzt betrachtet.

Anschließend wird in der Funktion `mac802154_cmd_data_req()` die Liste ausgelesen. Diese wird bei jedem empfangenen Data-Request aufgerufen. Dann wird in der Liste mit Paketen nach dem ersten Eintrag gesucht, der dem aktuellen Empfänger zugeordnet ist und das Paket gesendet. Für den Fall, dass kein Paket für den Knoten vorhanden ist, dient die Funktion `mac802154_send_empty()`. Diese erstellt ein neues Datenpaket mit leerer Payload und der als Parameter angegebenen Absender- und Empfängeradresse und verschickt dieses.

5.2.2 Knoten

Das Tool `iz` wurde um den Befehl `poll` erweitert, so kann nun mit `iz poll <Device Name> <Coordinator Address>` der angegebene Koordinator abgefragt werden. Dafür wird eine Nachricht mit dem Befehl `IEEE802154_POLL_REQ` an den MAC-Layer gesendet. In der Funktion `ieee802154_poll_req()` werden die übergebenen Werte dann angenommen und daraus ein MAC-Paket erstellt und an den Koordinator gesendet. Anschließend wird noch auf die Antwort der Anfragen gewartet.

5.2.3 Interoperabilität der Implementierung

Bei der Entscheidung dafür, die indirekte Übertragung zu implementieren, war ein Faktor auch die Interoperabilität zu anderen Systemen. Diese wird durch die Standardisierung des Verfahrens ermöglicht. Deshalb soll auch getestet werden, ob die Linux-Implementierung mit einem anderen MAC-Framework kompatibel ist.

Da die Implementierung unter Linux noch nicht vollständig ist, war davon auszugehen, dass keine vollständige Interoperabilität möglich ist. Es können aber zumindest die bereits implementierten Funktionen getestet werden. Um dies zu Testen, wurde ein Linux-Knoten als Koordinator eingerichtet. Die Gegenstelle dazu ist ein Endknoten, der auf dem offiziellen MAC-Stack von Atmel basiert. Dieser implementiert den kompletten 802.15.4-Standard bis zur MAC-Schicht.

Der Knoten wurde so konfiguriert, dass er zu Beginn ASSOC-Requests an den Koordinator versendet, bis ihm von diesem eine Kurzadresse zugeteilt wird. Anschließend werden periodisch Data-Requests an den Koordinator gesendet. Wenn der Koordinator als Antwort auf den Request Daten an den Knoten sendet, so antwortet dieser kurz danach mit exakt den gleichen Daten. So kann überprüft werden, ob die Übertragung auch in beide Richtungen funktioniert.

Getestet wurde dann mit dem beim Linux-Treiber mitgelieferten Programm `izchat`. Damit lassen sich zwischen zwei Knoten Daten auf der MAC-Schicht übertragen. Der Erfolg der Übertragung konnte dann dadurch beobachtet werden, dass die ge-

sendeten Nachrichten kurze Zeit später erneut als Antwort vom Knoten erscheinen. Zudem wurde die Kommunikation während des Tests mit einem Sniffer mitgelesen, um genau zu sehen, was übertragen wird und mögliche Fehler zu finden.

6 Schlussbemerkungen

6.1 Zusammenfassung

Im Verlauf dieser Arbeit wurden für ein 802.15.4-Netz gängigen Stromsparverfahren vorgestellt und in die beiden Kategorien „vom Sender initiierte Übertragung“ und „vom Empfänger initiierte Übertragung“ eingeteilt. Diese wurden dann auf ihre Tauglichkeit anhand verschiedener praxisnaher Szenarien verglichen. Auf Basis dieses Vergleichs wurde die im 802.15.4 Standard spezifizierte Methode zur indirekten Übertragung ausgewählt und als Erweiterung des 802.15.4-Treibers für Linux implementiert. Die implementierten Funktionen konnten außerdem auch erfolgreich auf ihre Interoperabilität mit einer anderen Implementierung des Standards getestet werden.

Parallel dazu wurde auch eine Firmware für das RaspBee-Funkmodul geschrieben, um dieses mit dem Linux-Treiber verwenden zu können. Die entstandene Firmware ist portabel und sollte auch auf andere Module, in denen Atmel Mikrocontroller verwendet werden, einsetzbar sein. Damit wurde die Auswahl kompatibler Module vergrößert und somit auch die Zugänglichkeit des Treibers verbessert.

Der während dieser Arbeit entstandene Implementierung ist entweder auf der beiliegenden DVD oder im Internet unter den folgenden Adressen verfügbar:

- Linux-Kernel: `https://github.com/kuehnelth/linux/tree/poll-request`
- RaspBee Firmware: `https://github.com/kuehnelth/raspbee-linux802154`

- iz-Tools: `https://github.com/kuehnelth/linux-zigbee`

6.2 Ausblick

Die Implementierung der MAC-Befehle ist noch unvollständig, wodurch noch vollständige richtige Interoperabilität mit anderen standardkonformen Systemen möglich ist. Allerdings wurde bewusst darauf verzichtet, da daran bereits von einem anderen Entwickler gearbeitet wird. Außerdem ist Implementierung der Liste mit zu sendenden Paketen noch nicht vollständig, weshalb sie die maximale Anzahl der Einträge und die Wartezeit in der Liste noch nicht regeln lassen. Sobald diese Voraussetzungen erfüllt sind, kann die Implementierung zum Review an die Mailingliste des Treibers geschickt werden, um im Anschluss eine offizielle Integration in den Kernel zu erreichen.

Ein weiterer Punkt der noch implementiert und getestet werden muss ist die Möglichkeit für den Koordinator zu erkennen, zu welchen Knoten direkt und zu welchen indirekt übertragen werden muss. Dies ist beispielsweise für Netze nötig, die mehrere Koordinatoren besitzen.

Die Implementierung des serial-Treibers kann noch durch die Nutzung der zweiten Protokollversion verbessert werden. Dafür ist es notwendig, sowohl den Treiber, als auch die Firmware zu erweitern. Außerdem muss der Treiber auch noch offiziell in den Kernel aufgenommen werden. Dies ist zumindest derzeit nicht möglich gewesen, da gerade an einer Umstrukturierung des Treibers gearbeitet wird, und erst danach wieder neue Treiber aufgenommen werden.

7 Literaturverzeichnis

[Ada14] Adams, James: *Introducing Raspberry Pi HATs*. Version: 2014. http://www.raspberrypi.org/introducing-raspberry-pi-hats/, Abruf: 01.09.2014

[Ake13] Akerman, Dave: *Running The Raspberry Pi On Batteries*. Version: 2013. http://www.daveakerman.com/?page_id=1294, Abruf: 10.10.2014

[Che14] Cheneau, Tony: *IEEE 802.15.4 serial protocol version 2*. Version: 2014. https://github.com/linux-wpan/ieee802154-serial-protocol-version2/blob/master/ieee802154-serial-protocol-2.md, Abruf: 02.09.2014

[Con14] Contiki: *Contiki: The Open Source Operating System for the Internet of Things*. Version: 2014. http://www.contiki-os.org/, Abruf: 15.08.2014

[DM08] David Moss, Kevin K. Jonathan Hui H. Jonathan Hui: *TEP105: Low Power Listening*. Version: 2008. http://www.tinyos.net/tinyos-2.x/doc/html/tep105.html, Abruf: 12.10.2014

[dre14] dresden elektronik ingenieurtechnik GmbH: *deCONZ*. Version: 2014. http://www.dresden-elektronik.de/funktechnik/products/software/pc/deconz/, Abruf: 15.08.2014

[Dun11] Dunkels, Adam: The ContikiMAC Radio Duty Cycling Protocol / Swedish Institute of Computer Science. Version: Dezember 2011. http://dunkels.com/adam/dunkels11contikimac.pdf. 2011 (T2011:13). – Forschungsbericht

[Erg04] Ergen, Sinem C.: ZigBee/IEEE 802.15.4 Summary / EECS Berkeley. 2004. – Forschungsbericht

[EW14] Elinux-Wiki: *RPi Upstream Kernel Compilation*. Version: 2014. http://elinux.org/RPi_Upstream_Kernel_Compilation, Abruf: 09.09.2014

[GWCB11] Gutierrez, J.A. ; Winkel, L. ; Callaway, E.H. ; Barrett, R.L.: *Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensors With IEEE 802.15.4*. Wiley, 2011 (IEEE standards wireless networks series). – ISBN 9780738162850

[IEE11] IEEE: IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). In: *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)* (2011), Sept, S. 1–314. http://dx.doi.org/10.1109/IEEESTD.2011.6012487. – DOI 10.1109/IEEESTD.2011.6012487. ISBN 0–7381–4996–9

[KTDT12] Ko, JeongGil ; Tsiftes, Nicolas ; Dunkels, Adam ; Terzis, Andreas: Pragmatic low-power interoperability: ContikiMAC vs TinyOS LPL. In: *Proceedings of IEEE SECON 2012*, IEEE, Juni 2012. – ISBN 978–1–4673–1904–1, S. 94–96

[LZ14] Linux-Zigbee: *Contiki: The Open Source Operating System for the Internet of Things*. Version: 2014. http://linux-zigbee.sourceforge.net/cgi-bin/trac.cgi/wiki/SerialV1, Abruf: 30.08.2014

[RW14] RaspberryPi-Wiki: *Upstreaming*. Version: 2014. http://www.raspberrypi.org/hats-in-the-wild-and-a-unicorn/, Abruf:

10.10.2014

[Tin14] TinyOS: *TinyOS*. Version: 2014. http://www.tinyos.net/, Abruf: 15.08.2014

[Upt14] Upton, Liz: *HATs in the wild. And a unicorn.* Version: 2014. http://www.raspberrypi.org/hats-in-the-wild-and-a-unicorn/, Abruf: 10.10.2014

[Wik14] Wikipedia: *IEEE 802.15.4*. Version: 2014. http://commons.wikimedia.org/wiki/User:TestAccount1234/Sandbox, Abruf: 01.10.2014

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Dresden, 22. Oktober 2014 _____

Unterschrift