Hochschule für Technik und Wirtschaft Dresden
University of Applied Sciences
Fakultät Informatik/Mathemathik

Abschlussbericht

Forschungs- und Entwicklungsseminar Sensornetze

Ergebnisse des 2. Semesters

Eingereicht von:	Kai Richter
Hochschulbetreuer:	Prof. Dr. Jörg Vogt
Studiengang:	Angewandte Informationstechnologien (Master)
Matrikelnummer:	40701

Dresden, den 23.02.2017

Inhaltsverzeichnis

Abkürzungsverzeichnis

6LoWPAN	IPv6 over Low power Wireless Personal Area Network
CCA	Clear Channel Assessment
OSI	Open System Interconnection
RDC	Radio Duty Cycling
WPAN	Wireless Personal Area Network

Abbildungsverzeichnis

Tabellenverzeichnis

1 Aufgabenstellung

1.1 Zielstellung im 2. Semester

Der vorliegende Abschlussbericht zum Forschungs- und Entwicklungsseminar auf dem Gebiet der Sensornetze knüpft thematisch am Zwischenbericht des 1. Semsters (zu entnehmen aus: [FoS16]) und der darin beschriebenen Ausgangssituation, den durchgeführten Arbeiten sowie den erläuterten Ergebnissen an.

Abweichend von den Zielstellungen im 1. Semester konzentriert sich dieser Abschlussbericht jedoch auf die Bearbeitung weiterer Aufgaben, die sich konkret mit den folgenden zwei Punkten zusammenfassen lassen:

- Theoretische Betrachtung des ContikiMAC Radio Duty Cycling Protokols und Erörterung der Kompatibilität zum IEEE 802.15.4 Standard: Im Rahmen dieser Teilaufgabe soll geklärt werden, inwiefern eine Kommunikation zwischen Sensorknoten in einem Sensornetzwerk theoretisch möglich ist, die einerseits mit ContikiMAC bzw. andererseits mit dem IEEE 802.15.4 Standard betrieben werden. Die beiden Energiesparverfahren werden dabei genauer untersucht und auf eine mögliche Kompatibilität geprüft.

- Versuchsaufbau zur Implementierung des IEEE 802.15.4 Netzwerkstacks in einen Linux Kernel: Um die Kommunikationsmöglichkeiten zwischen verschiedenen Sensornetzwerken (z.B. aufbauend auf unterschiedlichen Betriebssystemen wie Contiki oder RIOT und dessen di-

versen RDC Mechanismen) weiterhin praktisch zu evaluieren, soll ein Raspbery Pi mit dem darauf liegenden Betriebssystem Raspian zunächst so konfiguriert werden, dass dieser in der Lage ist, in verschiedene Sensornetzwerke integriert zu werden. Unterschiedliche und unter Umständen auch nicht zur direkten Kommunikation kompatible Sensornetze könnten somit durch die zusätzlichen unter Linux verfügbaren Funktionen überbrückt oder der Datenverkehr zu Analysezwecken komfortabel mitgeschnitten werden.

1.2 Aufbau des Abschlussberichtes

Im Folgenden soll der Abschlussbericht die durchgeführten Arbeiten sowie die theoretisch und praktisch gewonnenen Erkenntnisse dokumentieren und aufbereiten. Entsprechend der genannten Aufgabenstellungen wird eine Gliederung in die folgenden wesentliche Teile vorgenommen.

Zunächst sollen die möglichen Mechanismen des Energiesparens in Sensornetzen mittels sogenannter Radio Duty Cycling Protokolle genauer untersucht werden. Die bereits genannten Verfahren des IEEE 802.15.4 Standards sowie ContikiMAC werden einzeln vorgestellt und eine Kompatibilität theoretisch abgeleitet. Darauf aufbauend ist eine Einschätzung möglich, ob die Kommunikation zwischen Sensornetzen, die auf diesen unterschiedlichen Verfahren aufsetzen, überhaupt denkbar ist und wenn ja, unter welchen Voraussetzungen bzw. mithilfe welcher Lösungsansätze.

Im zweiten Abschnitt wird der praktische Teil der Aufgabenstellung durchgeführt und der genaue Lösungsweg beschrieben. Das Ziel besteht darin, die einzelnen Schritte und Vorgehensweisen zur Implementierung von IEEE 802.15.4 und 6LoWPAN im Linux Kernel des Raspian Betriebssystems nachvollziehbar bzw. reproduzierbar darzustellen.

Schließlich werden die Ergebnisse im letzten Kapitel zusammengefasst und ein Ausblick für weiterführende Arbeiten am Thema gegeben.

2 Analyse der Kompatibilität von RDC Protokollen

2.1 Theoretische Grundlagen ContikiMAC

Die theoretischen Konzepte des Radio Duty Cyclings unter dem Open Source Betriebssystem Contiki beschreibt Adam Dunkles in seinem Paper [Dun11]. Die für dieses Kapitel wesentlichen Mechanismen lassen sich anhand der Abbildung 1 erläutern.

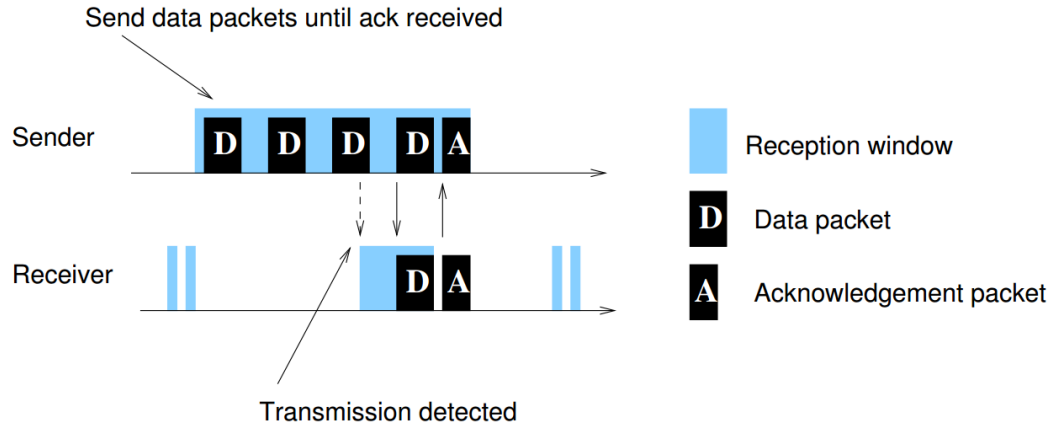

Abbildung 1: Funktionsweise von ContikiMAC[1]

Die Grundidee des Radio Duty Cyclings besteht demnach darin, das Funkmodul als einen der größten Energieverbraucher des Sensorknotens nur periodisch

[1] Entnommen aus [Dun11]

einzuschalten und auf dem Funkkanal nach Datenpaketen zu lauschen. Dieses periodische Einschalten wird in der Abbildung durch die kurzen blauen Reception Windows des Datenempfängers gekennzeichnet. Es handelt sich dabei um zwei aufeinanderfolgende Clear Channel Assessments (CCA), die durchgeführt werden, um die Belegung des Funkkanals mit Daten zu testen. Ist dieser Test negativ, kann das Funkmodul wieder für eine definierte Zeit in den Schlafmodus versetzt werden. Befinden sich dagegen Daten für den Sensorknoten auf dem Funkkanal, bleibt das Funkmodul für die Dauer der Übertragung aktiv, empfängt die Daten und bestätigt den Empfang mit einem Acknowledgement. Aus der Sicht des Datensenders wird das Datenpaket folglich so lange wiederholt gesendet, bis das entsprechende Acknowledgement vom Emfängerknoten erhalten wird.

2.2 Theoretische Grundlagen IEEE 802.15.4

IEEE 802.15.4 definiert zunächst den Standard eines Übertragungsprotokolls für Wireless Personal Area Networks (WPAN) und repräsentiert darin die unteren beiden Schichten des OSI Modells respektive Bitübertragungs- und MAC-Layer. Durch die Entwicklungsziele des Protokolls wie z.B. geringe Leistungsaufnahme, kostengünstige Hardware und sichere Übertragung findet es vor allem Anwendung auf dem Gebiet der Sensornetze. [Wiki]

Auch IEEE 802.15.4 bietet die Möglichkeit eines Stromsparmechanismus an, welcher als „indirekte Kommunikation“ bezeichnet werden kann. Die konkrete Funktionsweise ist in Abbildung 2 dargestellt. Der Sender eines Datenpakets ist dabei zwingend als „always on“ Device konfiguriert, d.h. dieses Gerät wird sein Funkmodul nie in den Schlafmodus versetzen sondern ist ständig für andere Netzwerkteilnehmer erreichbar. Weiterhin werden zu sendende Datenpakete zunächst zwischengespeichert, bis die Empfangende Station selbst gezielt danach fragt. Diese Voraussetzungen sind notwendig, da der Empfänger periodisch beim potenziellen Sender Anfragen nach neuen für ihn verfügbaren Daten stellt (Data Requests). Ist diese der Fall, wird die Datenübertragung

durchgeführt (begrenzt von zwei Acknowledgement-Paketen), ansonsten folgt eine Schlafphase des Empfängers.

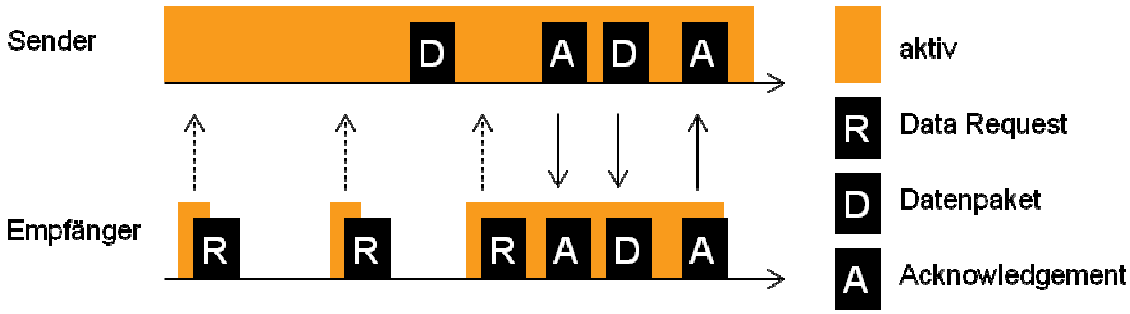

Abbildung 2: Indirekte Kommunikation unter IEEE 802.15.4

Auf diese Weise werden Datenpakete also nicht vom Sender *direkt* gesendet sondern *indirekt* vom Empfänger abgefragt, was den wesentlichen Unterschied zum bereits vorgestellten Übertragungsverfahren mittels ContikiMAC ausmacht.

2.3 Schlussfolgerungen zur Kompatibilität

Aufbauend auf den theoretischen Grundlagen der RDC Mechanismen sei nun zunächst folgende hypothetische Ausgangssituation gegeben: Es bestehe bereits ein funktionierendes Sensornetzwerk auf Basis des Betriebssystems Contiki. Die Sensorknoten dieses Netzwerkes arbeiten dabei mit dem RDC Protokoll ContikiMAC, um möglichst viel Strom sparen zu können. Parallel dazu existiere ein weiteres Sensornetzwerk auf Basis des ebenfalls Open Source Betriebssystems RIOT OS, welches sich sehr stark am IEEE 802.15.4 Standard orientiert. Beide Sensornetze arbeiten zunächst unabhängig voneinander, sollen aber zukünftig miteinander kommunizieren können. Es stellt sich an diesem Punkt also die Frage nach der prinzipiellen Kompatibilität des ContikiMAC RDC Protokolls mit dem IEEE 802.15.4 Standard.

Ein existierender Artikel im Wiki der Contiki Entwickler beschreibt diese Problematik folgendermaßen: (entnommen aus [CoWi])

> *„The Contiki X-MAC **provides compatibility** with the 802.15.4 link layer standard by forming its messages according to the 802.15.4 specification. All messages, both strobes and data messages, use the 802.15.4 frame and addressing format. This makes it possible to interoperate between an **always-on** 802.15.4 network and a Contiki X-MAC network.“*

Diese Aussage lässt zunächst den Schluss zu, dass sich auch das Übertragungsverfahren von Contiki an die generellen Spezifikationen hält, die in IEEE 802.15.4 bezüglich Bitübertragungsschicht und MAC-Layer definiert sind und insofern eine generelle Kompatibilität für die Kommunikation untereinander gegeben ist. Der eigentliche Knackpunkt zur Kompatibilität besteht hier jedoch in der Art und Weise des Stromsparens unter Contiki. Denn wie aus den vorangegangenen theoretischen Betrachtungen ersichtlich wird, unterscheiden sich die RDC Mechanismen von ContikiMAC und denen des IEEE 802.15.4 Standards maßgeblich. Daher heißt es in dem Artikel der Contiki Entwickler weiterhin:

> *„Although the logic is different, and the nodes **cannot directly communicate** with each other, other mechanisms of the 802.15.4 standard work, such as the ability to discard incoming packets that have a destination address that does not match the node's address.“*

Zusammenfassend ergeben sich aus den Aussagen der Entwickler selbst, sowie den theoretischen Betrachtungen der Kapitel 2.1 und 2.2 damit zwei wesentliche Einschränkungen für die Kompatibilität von ContikiMAC Netzwerken und IEEE 802.15.4 Sensorknoten:

1. Die Kompatibilität zu ContikiMAC ist nur bei der Kommunikation mit einem „always-on“ IEEE 802.15.4 Netzwerk gegeben. Das heißt, aufgrund der erörterten Unterschiede der beiden RDC Mechanismen, ist ein IEEE 802.15.4 Standard-Device nicht in der Lage Daten an einen ContikiMAC Empfänger zu versenden, da er den genauen Zeitpunkt, an

dem der ContikiMAC Empfänger gerade aktiv ist, nicht kennt und außerdem den Mechanismus zum wiederholten Senden des Datenpaketes bis zu einem entsprechenden Acknowledgement (Vgl. Abbildung 1) nicht implementiert hat. Entgegengesetzt ist die Kommunikation jedoch denkbar und möglich, da das ContikiMAC Device jederzeit Daten an die „always-on“ Station schicken kann.

2. Damit einher geht die zweite Aussage, dass die beiden unterschiedlichen Sensorknoten nicht *direkt* miteinander kommunizieren können. Möchte ein ContikiMAC Knoten also auch Daten von einer IEEE 802.15.4 Station empfangen, so ist dies nur über die Implementierung der „indirekten Kommunikation“ möglich, d.h. das ContikiMAC Device erfragt indirekt über Data Requests nach verfügbaren Daten beim „always-on“ IEEE 802.15.4 Knoten (Vgl. Abbildung 2). Dieser Mechanismus ist jedoch unter ContikiMAC nicht vorgesehen.

Weiterhin erschwerend auf die fragliche Kompatibilität wirken sich die bislang noch außen vor gelassenen ContikiMAC Zusatzfunktionalitäten Fastsleep und Phase-Optimization aus. Kritisch ist diesbezüglich vor allem die letztgenannte Phasenoptimierung, mithilfe derer sich ein unter ContikiMAC betriebener Sensorknoten mit anderen Stationen im Netzwerk synchronisieren kann. Der Sender eines Datenpaketes versucht dabei abzuschätzen, wann die empfangende Station wahrscheinlich aktiv ist und wann nicht. Die Datenpakete werden folglich erst zum passenden Zeitpunkt gesendet, was einerseits das Datenaufkommen auf dem Kommunikationskanal sowie andererseits den Energiebedarf des Senders verringert. Unter IEEE 802.15.4 ist ein derartiger Mechanismus jedoch nicht vorgesehen. Die zusätzliche Logik, die hinter diesem Algorithmus steckt, wirkt sich daher wohl ebenfalls negativ auf die Kompatibilität aus.

Unproblematischer ist dagegen der sogenannte Fastsleep Mechanismus, welcher nach [Dun11] dazu verwendet wird, um während einer Aufwachphase möglichst schnell zu prüfen, ob auf dem Funkkanal tatsächlich Daten zur Übertragung vorhanden sind oder ob der Sensorknoten nur aufgrund von Rauschen „geweckt“ wurde. In letzterem Fall kann der Empfang abgebrochen und

das Funkmodul des Emfängers noch vor Abschluss der Übertragung schnellstmöglich wieder in den Schafmodus versetzt werden, um weitere Energie zu sparen. Auf eine Kompatibilität zum IEEE 802.15.4 Standard sollte dieses Kriterium aber keinen Einfluss haben, da es für die sendenden Station letztendlich unerheblich ist, ob ein Sensorknoten inaktiv ist, für den das Datenpaket ohnehin nicht bestimmt ist. Außerdem wirkt sich ein aktiviertes Fastsleep auch während des Sendevorgangs nicht auf die Kommunikation aus.

2.4 Lösungskonzepte

Aus den theoretischen Überlegungen in Kapitel 2.3 ergibt sich, dass ContikiMAC nicht kompatibel zur indirekten Übertragung des IEEE 802.15.4 Standards sein kann. Eine funktionierende *direkte* Kommunikation zwischen einem Sensorknoten, der mit ContikiMAC betrieben wird, und einer Station, die z.B. das Betriebssystem RIOT verwendet, lässt sich damit zunächst ausschließen. Zwei Ansätze, um solche heterogenen Sensornetze dennoch zu ermöglichen, sollen durch die folgenden Lösungskonzepte aufgezeigt werden.

2.4.1 Indirekte Kommunikation mit „always on“ Netzwerken

Der erste Ansatz wurde im Verlaufe der Diskussion des Kapitels 2.3 bereits genannt und besteht in der Realisierung einer indirekten Kommunikation zwischen einem „always on“ IEEE 802.15.4 Netzwerk (z.B. auf Basis von RIOT OS), sowie dem ContikiMAC. (Vgl. Abbildung 3) Das IEEE 802.15.4 Standard Netzwerk ist dabei stets empfangsbereit, kann damit keine zusätzliche Energie sparen aber jederzeit Daten des ContikiMAC Netzwerkes empfangen. Da es den ContikiMAC Algorithmus zum Versenden von Daten jedoch nicht „versteht“, kann es die Contiki Netzwerkknoten auch nicht auf direktem Wege ansprechen. Hierfür müssen die ContikiMAC betriebenen Netzwerkteilnehmer selbst periodisch nach für sie verfügbaren Datenpaketen anfragen und sich die

Daten damit auf „indirektem Wege“ besorgen. Diese Methode bedingt jedoch eine Implementierung dieses „indirekten“ Mechanismus‘ in ContikiMAC.

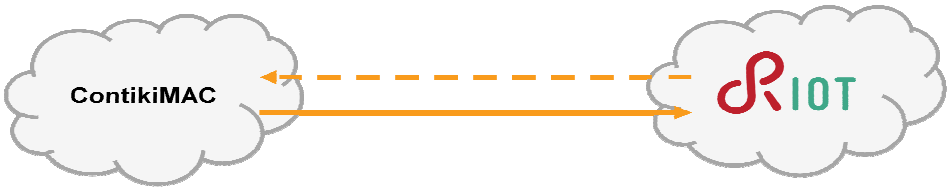

Abbildung 3: Indirekte Kommunikation mit „always on“ Netzwerken

2.4.2 Indirekte Kommunikation mittels Routing

Eine „indirekte Kommunikation“ könnte alternativ aber auch als Datenübertragung aufgefasst werden, die nicht über zwei direkt miteinander verbundene Sensorknoten durchgeführt wird, sondern bei der eine zusätzliche Zwischeninstanz in Form eines Routers zum Einsatz kommt (Vgl. Abbildung 4).

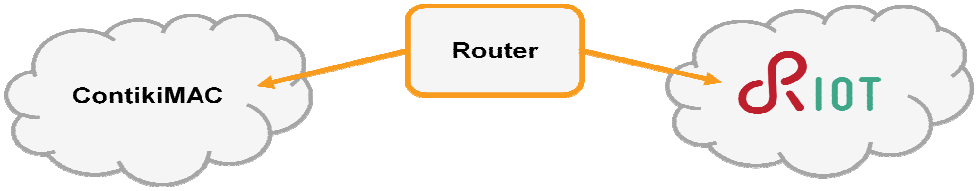

Abbildung 4: Indirekte Kommunikation über einen Router

Hierbei wird sämtliche Kommunikation zwischen beiden Netzwerken über den zwischengelagerten Router geleitet. Dieser muss dabei in der Lage sein, sowohl die RDC Mechanismen des ContikiMAC als auch des IEEE 802.15.4 Standards verstehen zu können. Diese Variante scheint zunächst aufwändiger in der Realisierung zu sein, bringt aber auch einige Vorteile. So ist es z. denkbar, dass über den Router gleichzeitig die Kopplung zu IP-basierten Netzwerken wie dem Internet realisiert wird. Außerdem könnte ein entsprechend konfiguriertes Gerät, z.B. basierend auf einem vollwertigen Linux Betriebssystem, auch für diverse Zusatzfunktionalitäten wie dem Sniffing des Datenverkehrs sowohl innerhalb als auch zwischen den beiden Sensornetzwerken genutzt werden.

3 IEEE 802.15.4 und 6LoWPAN im Raspberry Pi Linux Kernel

Auf den theoretischen Betrachtungen aus Kapitel 2 und insbesondere Punkt 2.4 aufbauend, soll in diesem Abschnitt die Implementierung des IEEE 802.15.4 Netzwerkstacks in einen Linux Kernel sowie ein anschießender Funktionstest betrachtet werden. Faktisch wird damit der erste Schritt zur Umsetzung des in Kapitel 2.4.2 vorgeschlagenen theoretischen Lösungsansatzes praktisch realisiert, indem ein Raspberry Pi mit seinem Linux Betriebssystem Raspian zur generellen Kommunikation mit IEEE 802.15.4 basierten Senoretzwerken befähigt wird. Alle hierzu im Folgenden Kapitel beschriebenen und praktisch durchgeführten Vorgehensweisen zur Konfiguration des Linux Kernels wurden auf Grundlage der verschiedenen Quellen [RIOT], [Rasp], [OpLb] und [Wpan] hergeleitet.

3.1 Implementierung in Raspbian

3.1.1 Vorbereitungen

Zunächst sollte sicher gestellt werden, dass auf der SD Karte des Raspberry Pi's eine aktuelle Version des Betriebssystems Raspbian installiert ist. Dieses ist auf der Website des Herstellers *raspberrypi.org* erhältlich und kann unter Windows beispielsweise mit Hilfe des Tools Win32DiskImager leicht auf die SD Karte geflasht werden. Im folgenden wird das Betriebssystem *Raspbian*

Jessie with Pixel in der Version *Januar 2017* mit dem *Linux Kernel 4.4* verwendet. Ein erster Boot des Pi's stellt dessen Funktionsfähigkeit sicher.

Da der neu zu konfigurierende Kernel aus Gründen der Laufzeit nicht direkt auf dem Pi kompiliert werden soll, wird außerdem ein Cross Compiler benötigt. Hier wird auf *gcc-linaro-arm-linux-gnueabihf* unter dem Betriebssystem Ubuntu zurückgegriffen. Diesen erhält man entweder über den Konsolenbefehl `sudo apt-get- install gcc-linaro-arm-linux-gnueabihf` oder über die ebenfalls unter raspberry.org erhältliche Toolchain.

Aufbauend auf dieser vorbereiteten Arbeitsumgebung sind Nachfolgend alle durchgeführten Schritte zur Konfiguration des Raspian Kernels reproduzierbar aufgeführt.

3.1.2 Konfiguration des Kernels

Zunächst erfolgt das Klonen der neuesten Raspberry Pi Linux Quellen (hier Kernel Version 4.4.y, dieser hat den 802.15.4 Stack schon implementiert, er muss im Verlaufe der Konfiguration aber noch eingebunden werden) sowie die Durchführung einer ersten Default-Konfiguration über die beiden nachstehenden Befehlsblöcke:

```
git clone --depth=1 https://github.com/raspberrypi/linux
```

```
cd linux
KERNEL=kernel
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcmrpi_defconfig
```

Anschließend folgt das Einfügen des später verwendeten Funkchips (AT86RF233 auf einem Atmel REB233-XPRO) in den Device Tree des Kernels über das Hinzufügen der folgenden Codezeilen am Ende der Datei *arch/arm/boot/dts/bcm2835-rpi-b.dts.*

```
&spi0 {
    status="okay";
    spidev@0{
        status = "disabled";
    };
    spidev@1{
        status = "disabled";
    };
    at86rf233@0 {
        compatible = "atmel,at86rf233";
        reg = <0>;
        interrupts = <23 4>;
        interrupt-parent = <&gpio>;
        reset-gpio = <&gpio 24 1>;
        sleep-gpio = <&gpio 25 1>;
        spi-max-frequency = <3000000>;
        xtal-trim = /bits/ 8 <0x0F>;
    };
};
```

Nach dem Patch des .dts Files ist ein Rebuild der .dtb Files notwendig:

```
sudo ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make dtbs
```

An dieser Stelle folgt die eigentliche Kernel Konfiguration über die den Aufruf eines Menüs (siehe nachstehender Befehl, sollte dieser scheitern muss ggf. zuerst `sudo apt-get install libncurses5-dev` ausgeführt werden) in welchem folgende Einstellungen vorzunehmen sind:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

1. 802.15.4 und 6LoWPAN support aktivieren

```
Networking support
  --> Networking options
    --> IEEE Std 802.15.4 Low-Rate Wireless Personal Area Networks
support
```

Alle Optionen sollten in diesem Punkt entweder als Modul <**M**> oder direkter Teil des Kernels <*> konfiguriert werden. Danach <**Exit**> und das gleiche Prozedere für den Punkt:

```
Networking support
  --> Networking options
    --> 6LoWPAN support
```

2. Aktivieren des Treibers für das 802.15.4 Device

Im Rahmen der späteren Teststellung wird ein AT86RF233 SPI Funkchip auf einem Atmel REB233-XPRO verwendet. Aus dem Root Menü ist zur Aktivierung des dafür benötigten Treibers folgender Punkt aufzurufen:

```
Device Drivers
  --> Network device support
    --> IEEE 802.15.4 drivers
```

In dem Untermenü ist der entsprechende Treiber wieder auf <**M**> oder <*> zu setzen.

3. Default Boot Optionen setzen

```
Boot options
  --> () Default kernel command string
```

An dieser Stelle ist die Zeichenkette *console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 root=/dev/mmcblk0p2 rootfstype=ext4 rootwait* einzutragen. Letztlich ist die Konfiguration über <**Save**> in der vorgeschlagenen *.config* Datei zu speichern. Nach dem Speichern ist die Konfiguration abgeschlossen und das Menü kann über <**Exit**> verlassen werden.

3.1.3 Build und Übertragung des Kernels

Der anschließende Buildvorgang wird über folgenden Befehl gestartet:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs
```

Der Build kann trotz Cross Compiling einige Zeit in Anspruch nehmen. Nach dem erfolgreichen Build sind die neuen Dateien auf die SD Karte zu übertragen. Nach Anschluss der unter Abschnitt 3.2.1 bereits vorbereiteten SD-Karte, können dessen Partitionen über `lsblk` eingesehen werden. Folgende Ausgabe sollte dabei zu erkennen sein:

```
sdb
   sdb1
   sdb2
```

Hierbei stellt `sdb1` im Normalfall die FAT (boot) Partition und `sdb2` die ext4 (root) Partition dar. Folglich können folgende Befehle zum mounten der SD Karte genutzt werden:

```
mkdir /mnt/fat32
mkdir /mnt/ext4
sudo mount /dev/sdb1 /mnt/fat32
sudo mount /dev/sdb2 /mnt/ext4
```

Das Übertragen der Dateien erfolgt dann über die Anweisungen in den folgenden zwei Befehlsblöcken:

```
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- IN-
STALL_MOD_PATH=/mnt/ext4 modules_install
```

```
sudo cp /mnt/fat32/$KERNEL.img /mnt/fat32/$KERNEL-backup.img
sudo scripts/mkknlimg arch/arm/boot/zImage /mnt/fat32/$KERNEL.img
sudo cp arch/arm/boot/dts/*.dtb /mnt/fat32/
sudo cp arch/arm/boot/dts/overlays/*.dtb* /mnt/fat32/overlays/
sudo cp arch/arm/boot/dts/overlays/README /mnt/fat32/overlays/
```

Abschließend erfolgt das Unmounten der SD-Karte über die Befehle:

```
sudo umount /mnt/fat32
sudo umount /mnt/ext4
```

An dieser Stelle ist der neue Kernel erfolgreich installiert und der Raspberry Pi bereit zum Boot. Nach dem Boot kann über `uname -a` verifiziert werden, dass auch die konfigurierte Kernel Version genutzt wird.

3.1.4 Installation der Userspace Tools

Damit das 802.15.4 Device auch ordnungsgemäß eingerichtet und genutzt werden kann, wird außerdem noch das Userspace Tool **Linux-WPAN** benötigt. Es lässt sich auf dem Raspberry Pi über folgende Befehle leicht installieren:

```
sudo apt-get install libnl-3-dev libnl-genl-3-dev wireshark
sudo apt-get install dh-autoreconf

git clone https://github.com/linux-wpan/wpan-tools
cd wpan-tools

./autogen.sh
./configure CFLAGS='-g -O0' --prefix=/usr --sysconfdir=/etc --
libdir=/usr/lib
make
sudo make install
```

3.2 Test der Konfiguration mittels Simulation

An diesem Punkt sollte sowohl der Raspberry Pi samt Linux Kernel und Raspian Betriebssystem sowie die darauf aufsetzenden Userspace Tools fertig konfiguriert und eingerichtet sein. Das heißt, dass die Userspace Tools nun dazu verwendet werden können, die Konfiguration auf ihre tatsächliche Funktionsfähigkeit zu testen. Hierfür bietet die Implementierung des Kernels einen sogenannten *fakelb* Treiber an, welcher in der Lage ist, physische 802.15.4 Interfaces zu emulieren und damit eine IEEE 802.15.4 / 6LoWPAN basierte Netzwerkkommunikation zu simulieren.

Der fakelb Treiber muss zunächst mit einem Parameter für die Anzahl zu simulierender Devices gestartet werden, was mit dem Befehl `sudo modprobe fakelb numlbs=2` erfolgen kann. Im Beispiel werden zwei simulierte IEEE

802.15.4 Interfaces erstellt. Um diese auch für die entsprechende Ben8utzung zu konfigurieren, werden folgende Befehle verwendet:

```
iwpan phy0 set netns name wpan0

ip netns exec wpan0 iwpan dev wpan0 set pan_id 0xfefe
ip netns exec wpan0 ip link add wpan0 name lowpan0 type lowpan
ip netns exec wpan0 ip link set wpan0 up
ip netns exec wpan0 ip link set lowpan0 up
```

Hier dargestellt ist die exemplarische Konfiguration des physischen Interfaces *phy0*, dem zunächst ein zugehöriges *wpan0* Interface zugeordnet, die pan_id gesetzt, ein 6LoWPAN Interface erstellt, und alle Schnittstellen aktiviert werden. Um weitere physische Interfaces zu konfigurieren, müssen die entsprechenden Schritte auf diese Weise wiedeholt werden.

Die erstellten physischen Interfaces sowie die gesamte IP-Konfiguration des Raspberrys lassen sich an dieser Stelle über die Befehle `wpan dev` bzw. `ifconfig` anzeigen. Weiterhin ist es nun möglich, die 6LoWPAN Interfaces direkt über einen ping Befehl anzusprechen sowie die Kommunikation mithilfe eines Sniffing Tools wie. Wireshark mitzuschneiden. Eine auf diese Weise mitgeschnittene Kommunikation ist der Abbildung 5 zu entnehmen.

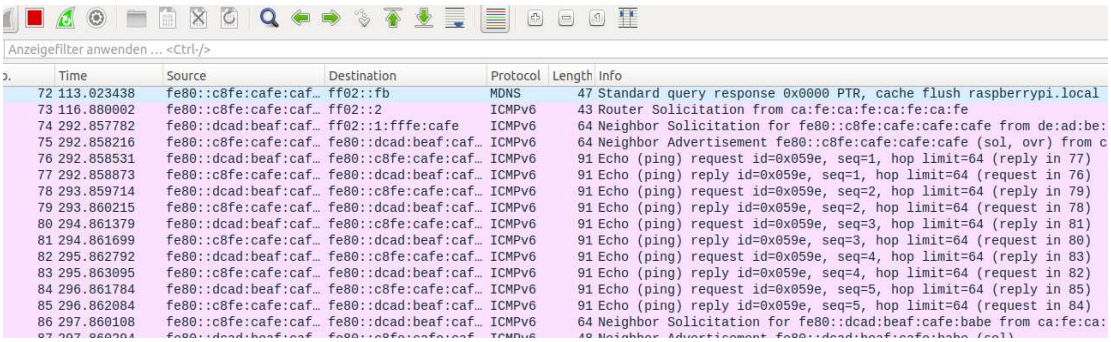

Abbildung 5: Wireshark Mittschnitt der Kommunikation im Testnetzwerk

Die abgebildete Kommunikation der emulierten Interfaces liefert an dieser Stelle den Nachweis dafür, dass der in den Linux Kernel implementierte und

im Verlaufe der Arbeit konfigurierte IEEE 802.15.4 Netzwerkstack grundsätzlich funktionsfähig ist. Die Simulation zeigt, dass der Raspberry Pi also nun mit Sensornetzen kommunizieren kann.

3.3 Verwendung eines Funkchips

Nachdem die generelle Funktionstüchtigkeit der Implementierung von IEEE 802.15.4 und 6LoWPAN in den Linux Kernel mittels Simulation eines Testnetzwerkes nachgewiesen wurde, soll anschließend auch ein physisches Testnetz aufgebaut werden. In diesem Zusammenhang soll das Funkmodul REB233-XPRO samt Atmel AT86RF233 Funkchip über die SPI Schnittstelle mit dem Raspberry Pi verbunden werden. Während der Konfiguration des Kernels wurde hierfür bereits der Treiber für den AT86RF233 im Kernel aktiviert und der zugehörige Devicetree kompiliert und eingebunden (Kap. 3.1.2).

Im ersten Schritt muss der Funkchip physich mit dem Raspberry Pi verkabelt werden. Dies erfolgt über entsprechende Pins am REB233-XPRO sowie den GPIO Header des Raspberry Pi. Die genauen Pinbelegungen lassen sich den jeweiligen Handbüchern bzw. Schnittstellenbeschreibungen entnehmen ([AtReb] und [Rasw]). Letztlich wurden 6 Pins entsprechend der nachstehenden Tabelle 1 miteinander verkabelt.

	Pin auf Raspberry Pi	Pin am REB233-XPRO
VCC 3,3V (Spannung)	1	20
GDN (Masse)	6	19
SPI_CLK (Clock)	23	18
SPI_MISO (Datenleitung)	21	17
SPI_MOSI (Datenleitung)	19	16
SPI_SS (Slave Select)	24	15

Tabelle 1: Pinbelegung zur Verkabelung des Funkchips

Weiterhin muss die SPI Schnittstelle auf dem Raspberry Pi aktiviert werden. Das ist komfortabel über ein Konfigurationsmenü möglich, welches mit Hilfe des Befehls `sudo raspi-config` aufgerufen werden kann. Unter den Advanced Options ist das SPI Interface dann aktivierbar.

Nach einem Neustart des Pi's kann schließlich über den Befehl `wpan dev` der WPAN Userspace Tools verifiziert werden, ob das 802.15.4 Device auch ordnungsgemäß vom Kernel erkannt wurde. Ist dies der Fall, kann das nun vorhandene physische Interface analog zu den in Kapitel 3.2 gegebenen Befehlen weiter konfiguriert und die benötigten WPAN und 6LoWPAN Schnittstellen aufgesetzt werden. Genau wie im Szenario der Simulation sollte anschließend ein funktionsfähiges Interface zur Kommunikation mit einem Sensornetz zur Verfügung stehen.

Um diese Konfiguration nun ebenfalls auf ihre Funktionsfähigkeit zu testen, wurde eine Sensornode des ersten Semesters direkt neben dem Raspberry Pi mit angeschlossenem Funkmodul platziert und eingeschaltet. Die Sensornode, welche mit Contiki betrieben wird, versucht direkt nach dem Einschalten eine Verbindung mit benachbarten Netzwerkknoten aufzunehmen und versendet hierfür IEEE 802.15.4 Datenpakete. Die Idee des Tests besteht darin, die Kommunikation auf dem konfigurierten Interface des Raspberry Pi mitzuschneiden und damit den Traffic der Sensornode sichtbar zu machen. Auf diese Weise könnte die Funktionsfähigkeit des angeschlossenen Funkchips direkt nachgewiesen werden.

Bis zum Ende dieses Forschungsseminars blieb das Mittschneiden des von der Sensornode ausgesendeten Datenverkehrs jedoch erfolglos. Die Funktionsfähigkeit des angeschlossenen AT86RF233 Funkchips konnte somit nicht direkt nachgewiesen werden. Dennoch seien an dieser Stelle einige Vermutungen genannt, warum dieser Test nicht erfolgreich war. Eventuell können daraus Aufgabenstellungen für folgende Forschungsseminare entstehen:

- Die Verkabelung des Funkmoduls mit dem Raspberry Pi und die benötigten Pins wurden mehrfach überprüft. Die Möglichkeit, dass ein wichtiger Pin dabei jedoch außer Acht gelassen wurde oder die Verkabelung nicht ganz korrekt war, ist jedoch nicht komplett auszuschließen.

- Es wurde ebenfalls vermutet, dass der Promiscuous Mode für den Fehlschlag des Tests verantwortlich gemacht werden kann. Eventuelle muss dieser eingeschaltet sein, damit Datenpakete, die nicht für den Raspberry Pi bestimmt sind, überhaupt mitgeschnitten werden können.

- Als eine weitere Fehlerquelle kann der Devicetree identifiziert werden, der während der Konfiguration des Kernels verwendet wurde. Unter Umständen sind in diesem einige Parameter falsch definiert oder fehlen.

- Schließlich ist auch eine simple Fehlkonfiguration des Testnetzwerkes mithilfe der Userspace Tools denkbar. Eventuell sind bei der Vergabe von Pan ID's oder Netzwerkadressen weitere Fehler unterlaufen.

4 Zusammenfassung und Ausblick

Die Ergebnisse im 2. Semester des Forschungs- und Entwicklungsseminars im Bereich Sensornetze können an dieser Stelle mit der Abarbeitung der beiden unter Kapitel 1.1 genannten Aufgabenstellungen zusammengefasst werden.

Im Laufe des Semesters wurde so die Frage nach der prinzipiellen Kompatibilität zwischen Standard IEEE 802.15.4 Netzwerken sowie ContikiMAC gestützten Sensornetzen über rein theoretische Überlegungen erörtert. Es kann festgehalten werden, dass die RDC Mechanismen zum Energiesparen bislang nicht direkt kompatibel zueinander sind. Um eine Kommunikation dennoch zu ermöglichen, kann jedoch eine der vorgeschlagenen Lösungsmöglichkeiten implementiert werden.

Der darauf aufbauende Versuch der Konfiguration eines Linux Kernels zur Verwendung von IEEE 802.15.4 / 6LoWPAN im Raspbian Betriebssystem auf dem Raspberry Pi war zunächst erfolgreich. Die prinzipielle Funktionsfähigkeit dieser Konfiguration konnte über eine einfache Emulation von physischen Netzwerkinterfaces und dessen Kommunikation untereinander nachgewiesen werden. Die Verwendung eines tatsächlich angeschlossenen Funkchips blieb jedoch bis zum Ende dieses Forschungsseminars erfolglos. Es wird allerdings vermutet, dass es sich hierbei um einen wahrscheinlich eher kleineren Fehler entweder bei der physischen Verkabelung, bei der Definition des Devicetrees oder der generellen Konfiguration des Testnetzes handelt.

Eine Fortsetzung der Aufgabenstellung in einem weiteren Forschungsseminar ist insofern denkbar, um die bestehenden Fehler zu korrigieren und weitere Szenarien mithilfe des Raspberry Pis zu testen. Bspw. kann zukünftig ein realistischeres Testnetz zur Kommunikation heterogener Sensornetze erprobt werden.

Literatur- und Quellenverzeichnis

[AtReb] Atmel Corporation: REB233-XPRO User Guide, 2013, http://www.atmel.com/Images/Atmel-42188-REB233-XPRO-User-Guide.pdf, Abrufdatum: 22.02.2017.

[CoWi] Contiki OS Wiki: Radio duty cycling, The ContikiX-MAC, 802.15.4 Compatibility, 18.09.2015, https://github.com/contiki-os/contiki/wiki/Radio-duty-cycling#802154_Compatibility, Abrufdatum: 22.02.2017.

[Dun11] Dunkels, Adam: The ContikiMAC Radio Duty Cycling Protocol, SICS Technical Report T2011:13, Dezember 2011, http://dunkels.com/adam/dunkels11contikimac.pdf, Abrufdatum: 22.02.2017.

[FoS16] Gräb, Alexander; Richter, Kai: Zwischenbericht für das Forschungsseminar Sensornetze, Sommersemester 2016, Dresden, 31.08.2016.

[OpLb] openlabs.co: 6LoWPAN kernel on a Raspberry Pi, 27.05.2014, http://openlabs.co/blog/archives/1-6LoWPAN-kernel-on-a-Raspberry-Pi, Abrufdatum: 22.02.2017.

[Rasp] raspberry.org: Kernel Building, https://www.raspberrypi.org/ documentation/linux/kernel/building.md, Abrufdatum: 22.02.2017.

[Rasw] raspberrywebserver.com: Raspberry Pi GPIO, http://raspberrywebserver.com/gpio/gpio.html, Abrufdatum: 22.02.2017

[RIOT] RIOT OS Wiki: How to install 6LoWPAN Linux Kernel on Raspberry Pi, 19.10.2016, https://github.com/RIOT-OS/RIOT/wiki/ How-to-install-6LoWPAN-Linux-Kernel-on-Raspberry-Pi, Abrufdatum: 22.02.2016.

[Wpan] wpan.cakelab.org: linux-wpan, 10.08.2016, http://wpan.cakelab.org/, Abrufdatum: 22.02.2017.

[Wiki] Wikipedia.de: IEEE 802.15.4, https://de.wikipedia.org/wiki/IEEE_802.15.4, Abrufdatum: 22.02.2017.