HAW HAMBURG

BACHELOR THESIS
Moritz Holzer

Implementierung der indirekten Übertragungen des IEEE 802.15.4 Standards in RIOT OS

FAKULTÄT INFORMATIK UND DIGITALE GESELLSCHAFT

Faculty of Computer Science and Digital Society

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Moritz Holzer

Implementierung der indirekten Übertragungen des IEEE 802.15.4 Standards in RIOT OS

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
der Fakultät Informatik und digitale Gesellschaft
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Schmidt
Zweitgutachter: Prof Dr. Martin Becke

Eingereicht am: 08.07.2026

Moritz Holzer

Thema der Arbeit

Implementierung der indirekten Übertragungen des IEEE 802.15.4 Standards in RIOT OS

Stichwörter

Kurzzusammenfassung

IEEE 802.15.4 bildet die Grundlage vieler energiearmer drahtloser IoT-Netze. Der Standard sieht indirekte Übertragungen vor, bei denen ein Koordinator Daten für schlafende Endgeräte zwischenspeichert und diese erst nach einer Anfrage des Endgeräts ausliefert. Dadurch kann der Empfänger den Funktransceiver über längere Zeit deaktivieren und Energie sparen. RIOT OS unterstützte diese standardisierten Mechanismen bisher jedoch nur unvollständig.

Diese Arbeit entwirft und implementiert ein MAC-Layer für indirekte IEEE 802.15.4 Übertragungen im nicht beacon-aktivierten Betrieb von RIOT OS. Dazu werden die relevanten Standardprimitiven analysiert und auf die Architektur von RIOT OS abgebildet. Die Evaluation untersucht Latenz, Downlinkdatenrate und Energieverbrauch. Die Ergebnisse zeigen den grundlegenden Zielkonflikt indirekter Übertragungen. Gegenüber einem dauerhaft aktiven Empfänger reduziert ein Poll-Intervall von 100 ms die mittlere Stromaufnahme auf etwa 78,6 %. Bei 500 ms wird eine rund 89,2 % erreicht. Gleichzeitig steigt die mittlere Round-Trip-Time von etwa 7,4 ms ohne Duty-Cycling auf 70,7 ms bei 100 ms und 311,7 ms bei 500 ms. Die Downlinkdatenrate fällt dabei von etwa 132 kbit/s ohne Duty-Cycling auf 8,66 kbit/s bei 100 ms und 1,88 kbit/s bei 500 ms. Für die untersuchte Konfiguration liegt ein sinnvoller Kompromiss zwischen Energieeinsparung, Latenz und Downlinkdatenrate insbesondere im Bereich von 100 ms bis 500 ms.

Moritz Holzer

Title of Thesis

Implementation of IEEE 802.15.4 indirect transmissions in RIOT OS

Keywords

Abstract

IEEE 802.15.4 is the basis for many low-power wireless IoT networks. The standard defines indirect transmissions, where a coordinator buffers data for sleeping end devices and delivers it only after the end device polls for pending data. This allows the receiver to keep its radio transceiver disabled for longer periods and reduce energy consumption. RIOT OS, however, previously provided only incomplete support for these standardized mechanisms.

This thesis designs and implements a MAC layer for IEEE 802.15.4 indirect transmissions in the non-beacon-enabled mode of RIOT OS. To this end, the relevant standard primitives are analyzed and mapped to the RIOT OS architecture.

The evaluation examines latency, downlink data rate, and energy consumption. The results show the fundamental trade-off of indirect transmissions. Compared to an always-on receiver, a polling interval of 100 ms reduces the average current consumption by about 78.6 %. At 500 ms, a reduction of about 89.2 % is achieved. At the same time, the mean round-trip time increases from about 7.4 ms without duty cycling to 70.7 ms at 100 ms and 311.7 ms at 500 ms. The downlink data rate decreases from about 132 kbit/s without duty cycling to 8.66 kbit/s at 100 ms and 1.88 kbit/s at 500 ms. For the evaluated configuration, a practical compromise between energy savings, latency, and downlink data rate lies particularly in the range from 100 ms to 500 ms.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

ACK Acknowledgement.

AEAD Authenticated Encryption with Associated Data.

API Application Programming Interface.

BLE Bluetooth Low Energy.

BSP Board Support Package.

CAP Contention Access Period.

CFP Contention-Free Period.

CoAP Constrained Application Protocol.

CSL Coordinated Sampled Listening.

CSMA-CA Carrier Sense Multiple Access with Collision Avoidance.

DMM Digital Multimeter.

DSME Deterministic and Synchronous Multichannel Extension.

DUT Device Under Test.

ED Energy Detection.

HAL Hardware Abstraction Layer.

HR-WPAN High-Rate WPAN.

IE Information Element.

IoT Internet of Things.

IP Internet Protocol.

IPv6 Internet Protocol Version 6.

ISR Interrupt Service Routine.

LLC Logical Link Control.

LLN Low-Power and Lossy Network.

LP-WAN Low-Power WAN.

LQI Link Quality Indicator.

LR-WPAN Low-Rate WPAN.

lwIP lightweight IP.

MAC Medium Access Control.

MCPS MAC Common Part Sublayer.

MCPS-SAP MCPS Service Access Point.

MCU Microcontroller Unit.

MFR MAC Footer.

MHR MAC Header.

MIC Message Integrity Code.

MLME MAC Sublayer Management Entity.

MLME-SAP MAC Sublayer Management Entity Service Access Point.

MPDU MAC Protocol Data Unit.

MSDU MAC Service Data Unit.

PAN Personal Area Network.

PD-SAP Physical Layer Data Service Access Point.

PHY Physical Layer.

PIB PAN Information Base.

PLME-SAP Physical Layer Management Service Access Point.

POS Personal Operation Space.

PPDU PHY Protocol Data Unit.

PSDU Physical Service Data Unit.

QoS Quality of Service.

RIT Receiver-Initiated Transmission.

RPL Routing Protocol for Low-Power and Lossy Networks.

RSSI Received Signal Strength Indicator.

RTOS Real-Time Operating System.

RTT Round-Trip Time.

SoC System-on-a-Chip.

SSID Service Set Identifier.

TMCTP TVWS Multichannel Cluster Tree PAN.

TSCH Timeslotted Channel Hopping.

TVWS Television White Space.

UART Universal Asynchronous Receiver Transmitter.

USB Universal Serial Bus.

WAN Wide Area Network.

WPAN Wireless Personal Area Network.

1 Einleitung

Das Internet of Things (IoT) ist eine Erweiterung des Internets um Sensoren und Aktoren, die die physische Welt repräsentieren. Durch die Einsatzorte, an denen die Geräte verwendet werden, ergibt sich häufig eine Energiebeschränktheit, da sie dort nicht durch das Stromnetz versorgt werden können. Die Geräte müssen daher energieeffizient arbeiten und sind oft so konzipiert, dass sie nur in kurzen Zeitintervallen messen oder Daten übertragen. Zusätzlich spielt der Kostenfaktor der Geräte eine Rolle, es werden günstige oder auch wegwerfbare Geräte in Netzwerken mit einer großen Anzahl drahtloser Knoten benötigt [12].

Diese Rahmenbedingungen legen die Entwicklung eines Standards mit geringer Komplexität und Protokolloverheads nahe. Die IEEE hat hierfür 1999 die 802 Working Group 15 ins Leben gerufen. Ihre Aufgabe ist die Definition verschiedener Standards in der drahtlosen Nahfeldkommunikation, welche allgemein als Wireless Personal Area Networks (WPANs) bezeichnet wird.

WPANs fokussieren sich auf den Personal Operation Space (POS), das nahe Umfeld einer Person oder eines Objektes. Die Working Group 15 hat drei Klassen von WPANs definiert. Diese werden mit Datenrate, Batterieverbrauch und Quality of Service (QoS) unterschieden: High-Rate WPANs (HR-WPANs) (IEEE 802.15.3) sind für multimediale Anwendungen mit hohen QoS-Anforderungen vorgesehen. Medium-Rate WPANs (IEEE 802.15.1/Bluetooth) unterstützen typische Gerätekommunikation mit für Sprachdienste geeigneter QoS. Low-Rate WPANs (LR-WPANs) (IEEE 802.15.4) hingegen adressieren industrielle, private und medizinische Anwendungen mit besonders geringen Anforderungen an Energieverbrauch, Kosten, Datenrate und QoS [12].

Im Gegensatz zu Low-Rate WPANs (LR-WPANs) stehen die Low-Power WANs (LP-WANs), wie z.B. LoRa, die sich auf noch niedrigere Datenraten, jedoch größtmögliche Reichweite spezialisieren [24].

Zur Weiterentwicklung der LR-WPAN Technologie wurde die Task Group 4 eingerichtet, deren Ziel die Spezifikation des späteren IEEE 802.15.4-Standards war [12]. Parallel zu diesen Standardisierungsaktivitäten entstanden alternative Konzepte für energieeffiziente Kurzstreckenkommunikation. Hierzu zählt insbesondere Wibree, eine von Nokia initiierte Low-Power-Technologie, die sich aus der Motivation heraus entwickelte, Bluetooth-fähige Geräte wie Mobiltelefone oder Laptops, um eine energieeffiziente Erweiterung zu ergänzen [7]. Der Ansatz von Wibree zielte dabei auf eine einfache Integration in bestehende Bluetooth-Ökosysteme ab.

Im Rahmen der IEEE-Standardisierung entschied sich die Task Group letztlich für einen anderen Ansatz, aus dem der IEEE 802.15.4-Standard hervorging [7]. Wibree wurde hingegen außerhalb dieses Prozesses weiterentwickelt und später in die Bluetooth-Spezifikation integriert, woraus schließlich Bluetooth Low Energy (BLE) entstand [15].

Der IEEE 802.15.4 Standard befindet sich in den unteren beiden Schichten des OSI-Modells. Die zweite Schicht (Data Link) wurde von IEEE 802.15 weiter in Logical Link Control (LLC) und Medium Access Control (MAC) aufgeteilt. IEEE 802.15.4 deckt dabei das erste Layer (PHY) und den unteren Teil des zweiten Layers (MAC) ab. Dieser Standard bildet die Grundlage für verschiedene weiterführende Kommunikationstechniken. Eine zentrale Erweiterung stellt 6LoWPAN [28] dar. 6LoWPAN definiert eine Adaptionsschicht, die die Übertragung von IPv6-Paketen über IEEE-802.15.4-Netzwerke ermöglicht. Darauf aufbauend definiert Thread [27] eine vollständige, IP-basierte Netzwerkarchitektur. Thread nutzt 6LoWPAN als Adaptionsschicht und integriert zusätzlich Routing-Mechanismen, Sicherheitsfunktionen, Netzwerkkonfiguration sowie Geräteauthentifizierung. Eine weitere darauf aufbauende Technologie ist Zigbee [8]. Zigbee definiert ergänzend zur physikalischen und MAC-Schicht eine eigene Netzwerk-, Sicherheits- und Anwendungsschicht und stellt damit einen vollständigen Protokollstack bereit. Im Gegensatz zu Thread basiert Zigbee jedoch nicht nativ auf IPv6, sondern verwendet ein eigenständiges Netzwerkprotokoll. Der Standard unterstützt zusätzlich indirekte Datenübertragungen. Bei indirekter Übertragung werden Daten für einen schlafenden Endknoten an einem Koordinator zwischengespeichert und später vom Endknoten aktiv abgefragt. Diese bieten den Vorteil, dass der Empfänger nicht immer den Übertragungen des Netzwerkes lauschen muss [17].

RIOT OS ist ein freies und quelloffenes Betriebssystem, das speziell für ressourcenbeschränkte IoT-Geräte entwickelt wurde. Es basiert auf einer modularen Microkernel-

Architektur und bietet Funktionen wie präemptives Multithreading, Echtzeitfähigkeit sowie eine einheitliche Hardwareabstraktion bei geringem Speicherbedarf [6].

Das Internet of Things (IoT) stellt besondere Anforderungen an Betriebssysteme, da viele eingesetzte Geräte nur über sehr begrenzte Ressourcen verfügen, gleichzeitig jedoch energieeffizient, zuverlässig und teilweise echtzeitfähig arbeiten müssen [5]. Zusätzlich zu diesen Anforderungen sollte es auch für Entwickler einfach und mit Standardbibliotheken zu verwenden sein. Dies ist bei vielen klassischen IoT-Betriebssystemen nicht gegeben [5]. RIOT OS bietet außerdem viele IoT-relevante Protokolle wie LoRa, BLE, CoAP und 6LoWPAN. Dies macht es zu einer guten Grundlage für diverse IoT-Anwendungen.

Mit diesen Anforderungen positioniert sich RIOT OS zwischen klassischen IoT-Betriebssystemen und vollwertigen Betriebssystemen wie Linux [5].

Der momentane Stand der Implementierung der MAC-Schicht in RIOT ist unvollständig und bildet nicht die im Standard definierten (MAC-)Schnittstellen ab. Bisher gibt es keine Möglichkeit, mit definierten Schnittstellen das Gerät mit indirekten Übertragungen zu betreiben. Da diese Art der Übertragung in Versuchen 25% Energieersparnis gezeigt hat[26], ist dies eine sinnvolle Erweiterung der Implementierung in RIOT OS.

1.1 Ziele

Ziel dieser Arbeit ist es, die indirekten Übertragungsmechanismen des IEEE 802.15.4-Standards in RIOT OS verfügbar zu machen und damit die Unterstützung des Standards innerhalb des Betriebssystems zu erweitern.

Der Standard unterscheidet zwischen einem synchronisierten und mehreren unsynchronisierten Kommunikationsmodi. Während im synchronisierten Betrieb eine zeitliche Strukturierung durch sogenannte Superframes erfolgt, arbeitet der unsynchronisierte Modus ohne diese Struktur. Im unsychronisierten Modus ist der Koordinator dauerhaft empfangsbereit, da keine fest definierten Kommunikationszeiträume existieren. Im Modus mit Superframes muss das Gerät regelmäßig aufwachen, um sich mit dem Koordinator zu synchronisieren. Dies kann ggf. zu einem höheren Energieverbrauch führen als nötig [21]. Die Verwendung von Superframes erzeugt einen Overhead auf grund ihrer höheren Komplexität durch die Synchronisierung. In industriellen Anwendungen ist der unsynchronisierte Modus weiter verbreitet; beispielsweise verwendet Thread ausschließlich diesen Betriebsmodus [29].

Aus den oben genannten Gründen wird im Rahmen dieser Arbeit der Fokus auf die Umsetzung der indirekten Übertragungen im unsynchronisierten Modus gelegt. Damit wird eine grundlegende Basis geschaffen, um die Standardkonformität von RIOT OS im Hinblick auf IEEE 802.15.4 weiter auszubauen.

1.2 Aufbau der Arbeit

Kapitel 2 betrachtet die Unterstützung indirekter IEEE 802.15.4-Übertragungen in bestehenden IoT-Betriebssystemen und grenzt den Ansatz dieser Arbeit davon ab.

Kapitel 3 beschreibt die technischen Grundlagen von IEEE 802.15.4, RIOT OS und der bestehenden Netzwerkarchitektur. Dabei werden insbesondere die für indirekte Übertragungen relevanten Mechanismen des Standards und die vorhandenen Schnittstellen in RIOT OS betrachtet.

Kapitel 4 analysiert die für indirekte Übertragungen benötigten IEEE 802.15.4-Schnittstellen und leitet daraus die Anforderungen an eine Integration in RIOT OS ab.

Kapitel 5 stellt das Design der entwickelten MAC-Schicht vor. Dazu gehören die Zustandsmaschine, die Callback-Architektur und die Schnittstellen zu höheren Schichten.

Kapitel 6 beschreibt die Implementierung und die durchgeführten Tests. Dabei werden unter anderem die Warteschlangen für indirekte Übertragungen, die Einbindung in den SubMAC-Layer sowie die Integration in GNRC erläutert.

Kapitel 7 evaluiert die Implementierung anhand von Latenz, Downlinkdatenrate und Energieverbrauch. Abschließend fasst Kapitel 8 die Ergebnisse zusammen und gibt einen Ausblick auf mögliche Erweiterungen.

2 IEEE 802.15.4 in IoT-Betriebssystemen

Zur Einordnung der in dieser Arbeit vorgestellten Implementierung werden im Folgenden bestehende IoT-Betriebssysteme und deren IEEE 802.15.4-Unterstützung betrachtet. Der Fokus liegt dabei nicht auf einer vollständigen Gegenüberstellung aller Netzwerkfunktionen, sondern auf der Frage, ob und wie indirekte Datenübertragungen im Betrieb ohne Superframes unterstützt werden.

2.1 RIOT OS

In RIOT OS kann IEEE 802.15.4 entweder mit 6LoWPAN, OpenDSME, OpenThread oder mit einem sogenannten „SubMAC“ Layer betrieben werden. Bei 6LoWPAN wird der GNRC Netzwerk-Stack verwendet. Der SubMAC-Modus bildet zentrale Funktionen des Betriebs ohne Superframes weitgehend standardnah ab. Für eine vollständige Unterstützung dieses Betriebsmodus fehlen jedoch indirekte Übertragungen über definierte MAC-Schnittstellen. Das SubMAC Layer wird auch in OpenThread verwendet [25]. Die Funktionen der einzelnen Transceiver werden auf eine HAL Abstraktion abgebildet. Diese bietet eine allgemeine Schnittstelle für IEEE 802.15.4 Funktionen.

Damit existiert in RIOT OS bereits eine hardwareunabhängige Grundlage für den in dieser Arbeit betrachteten Betriebsmodus. Die für indirekte Übertragungen notwendige schnittstellengestützte Unterstützung ist jedoch bisher nicht vorhanden. Einzelheiten hierzu werden in Kap. 3.2 erläutert.

2.2 Zephyr RTOS

Zephyr ist genau wie RIOT OS ein für die Nutzung auf ressourcenbeschränkten Systemen ausgelegtes Betriebssystem [31].

Die IEEE 802.15.4 Implementierung in Zephyr bietet kein eigenes MAC Layer an [30]. Der Standard kann über eine native „SoftMAC“, welche herstellerspezifische HAL Schnittstellen abstrahiert, verwendet werden. Diese kann dann direkt, mit 6LoWPAN bzw. über OpenThread verwendet werden. Indirekte Übertragungen werden nur mittels OpenThread unterstützt. Wobei die für die indirekten Übertragungen benötigten Schnittstellen vorhanden sind und darauf aufbauend diese Funktionalität implementiert werden könnte.

2.3 Contiki

Contiki ist ein Betriebssystem für ressourcenbeschränkte eingebettete Systeme und Sensornetze. Es wurde für IP-basierte Kommunikation auf kleinen Sensorknoten entwickelt und unterstützt typische Protokolle des Internet of Things (IoT), darunter IEEE 802.15.4, 6LoWPAN, IPv6 und RPL [19]. Die 6LoWPAN-Funktionalität ist in Contiki in den uIP Netzwerkstack eingebunden. Dabei bildet 6LoWPAN die Anpassungsschicht zwischen IPv6 und der IEEE 802.15.4-basierten Funkübertragung [9]. Lightweight IP (lwIP) ist hiervon abzugrenzen. LwIP wurde ebenso wie uIP von Adam Dunkels entwickelt, ist jedoch ein eigenständiger TCP/IP-Stack. LwIP bietet ebenfalls 6LoWPAN Unterstützung, unter anderem für IEEE 802.15.4, ist aber nicht der zentrale Netzwerk-Stack Contikis. Unterhalb von 6LoWPAN nutzt Contiki die jeweils konfigurierte MAC Schicht.

Der energiesparende Betrieb der Funkschnittstelle wird in Contiki jedoch nicht primär über die im IEEE 802.15.4-Standard vorgesehene indirekte Datenübertragung realisiert, sondern über Radio Duty-Cycling Verfahren wie ContikiMAC [10]. Damit unterscheidet sich ContikiMAC grundlegend von der indirekten Datenübertragung nach IEEE 802.15.4. ContikiMAC verwendet stattdessen wiederholte Datenübertragungen durch den Sender, um den periodisch aufwachenden Empfänger zu erreichen [10]. Der energiesparende Betrieb wird somit nicht durch koordinatorseitiges Zwischenspeichern und Polling, sondern durch Radio Duty Cycling umgesetzt.

2.4 NuttX

NuttX stellt mit `mac802154` eine generische IEEE 802.15.4 MAC Schicht bereit, die über MAC Sublayer Management Entity (MLME)- und MAC Common Part Sublayer

(MCPS)-Primitive angesprochen werden kann [2]. Es enthält eine indirekte Sendewarteschlange, in der ein Koordinator Frames bis zu einer Anfrage des Zielgeräts zwischenspeichern kann [3]. Damit existiert in NuttX grundsätzlich ein koordinatorseitiger Mechanismus für indirekte Datenübertragung.

2.5 Weitere Systeme

TinyOS war eines der ersten Systeme mit IEEE 802.15.4-Unterstützung [20, 14]. Es implementiert Beacon- und Non-Beacon-Betrieb sowie indirekte Übertragungen. Jedoch wird es seit Jahren nicht mehr aktiv weiterentwickelt (letzte Version 2012).

Contiki-NG ist das Nachfolgeprojekt von Contiki und implementiert unter anderem den TSCH Modus des Standards, der in Kap. 3.1.4 näher beschrieben wird [23]. Dieser Modus basiert auf festgelegten Sende- und Empfangszeiträumen. Der energiesparende Betrieb wird daher nicht durch klassische indirekte Übertragungen im non beacon Betrieb erreicht, sondern durch synchronisierte Zeitschlitze und geplante Kommunikation.

Mbed ist ein RTOS für ARM Cortex basierte Systeme [4]. Der Netzwerk-Stack unterstützt natives IEEE 802.15.4 mit indirekten Übertragungen. Das Projekt wird jedoch ab Juli 2026 als End-of-Life behandelt [22].

Apache MyNewt ist ein modulares Echtzeitbetriebssystem für ressourcenbeschränkte eingebettete Systeme. Im Gegensatz zu den zuvor genannten Systemen stellt MyNewt Core jedoch keine generische IEEE 802.15.4 Unterstützung bereit.

FreeRTOS stellt keine zentrale HAL und keinen eigenen IEEE 802.15.4 MAC/PHY Stack bereit. Die Hardwareanbindung erfolgt über portierungs- und herstellerspezifische Treiber bzw. BSPs; FreeRTOS dient dabei nur als Laufzeitumgebung für Tasks und Synchronisation [1].

2.6 Abgrenzung dieser Arbeit

Die Betrachtung bestehender Betriebssysteme zeigt, dass die Unterstützung indirekter Datenübertragung gemäß IEEE 802.15.4 im Betrieb ohne Superframes sehr unterschiedlich realisiert ist. Während einige Systeme energiesparende Kommunikation über alternative Mechanismen wie Radio Duty Cycling oder Timeslotted Channel Hopping (TSCH)

umsetzen, unterstützen andere Systeme indirekte Übertragungen nur innerhalb bestimmter Netzwerk-Stacks oder durch herstellerspezifische Implementierungen.

Zephyr stellt zwar die für indirekte Übertragungen benötigten Schnittstellen grundsätzlich bereit, bietet jedoch keine eigenständige und allgemein nutzbare MAC-Schicht, welche diese Funktionalität unabhängig von OpenThread bereitstellt. NuttX verfügt mit `mac802154` über eine generische MAC-Schicht und unterstützt indirekte Übertragungen koordinatorseitig über eine Warteschlange. Die Abstraktion ist jedoch stärker an die dortige MAC-Architektur gebunden und unterscheidet sich damit grundlegend vom in RIOT verfolgten Ansatz.

RIOT verfolgt ähnlich wie Zephyr das Ziel, hardwarespezifische Funktionen über eine allgemeine Schnittstelle bereitzustellen. Mit dem SubMAC-Layer existiert bereits eine weitgehend hardwareunabhängige Implementierung für den Betrieb ohne Superframes. Diese generalisiert zentrale MAC Funktionen und ermöglicht dadurch eine Nutzung verschiedener IEEE 802.15.4 Transceiver über eine gemeinsame Schnittstelle. Für eine vollständige standardkonforme Unterstützung des Betriebs ohne Superframes fehlt jedoch bislang die schnittstellengestützte Unterstützung indirekter Datenübertragungen.

An diesem Punkt setzt die vorliegende Arbeit an. Ziel ist es, die bestehende Generalisierung des RIOT SubMAC Layers so zu erweitern, dass indirekte Datenübertragungen im Betrieb ohne Superframes unabhängig von konkreter Hardware unterstützt werden können. Die Arbeit grenzt sich damit von Betriebssystemen ab, die indirekte Übertragungen entweder nur innerhalb vollständiger Netzwerk-Stacks, über herstellerspezifische Implementierungen oder über alternative Energiesparmechanismen realisieren.

3 Technische Grundlagen

Dieses Kapitel beschreibt die technischen Details, die für die Analyse und die spätere Implementierung benötigt werden. Dabei werden insbesondere die Mechanismen für indirekte Übertragungen im Betrieb ohne Superframes betrachtet. Zunächst wird der IEEE 802.15.4-Standard mit seiner Schichtenarchitektur und den relevanten Funktionen beschrieben. Anschließend folgt ein Überblick über RIOT OS und dessen Unterstützung der IEEE 802.15.4-Mechanismen.

3.1 IEEE 802.15.4

Alle Angaben in diesem Kapitel beziehen sich auf die Revision von 2024 des IEEE 802.15.4 Standards [17].

3.1.1 Architektur

Der Standard ist in einer Schichtenarchitektur aufgebaut, die sich am OSI-Modell orientiert. Es werden dabei die untersten beiden Schichten abgebildet: PHY und MAC. Die Schichten kommunizieren dabei, wie in dieser Architektur üblich, nur mit den ihnen anschließenden Schichten. Die Schichten und ihre Schnittstellen sind in Abb. 3.1 zu sehen. Die dort beschriebenen „Next higher layers“ bestehen aus der Netzwerkschicht und der Applikationsschicht. Diese gehören jedoch nicht zum Umfang des Standards.

3.1.1.1 PHY

Das PHY Layer ist für die Aktivierung und Deaktivierung des Transceivers, die Steuerung der Channel-Einstellungen und das Senden und Empfangen von Paketen zuständig. Es bildet dabei die Schnittstelle zwischen Transceiver(-Firmware) und dem MAC Layer. Die

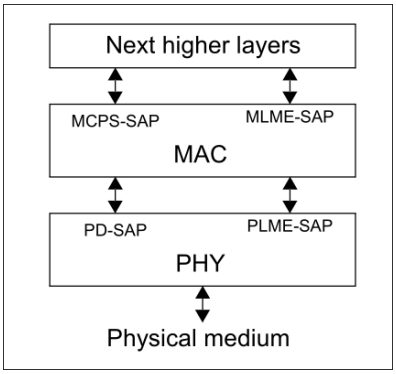

Abb. 3.1: LR-WPAN Architektur. siehe [17]

in Abb. 3.1 gezeigten Schnittstellen (PD-SAP, PLME-SAP) zum MAC-Layer sind im Standard nicht definiert, da sie in den typischen Implementierungen erwartungsgemäß nicht bereitgestellt werden [17].

Es werden verschiedene PHYs mit unterschiedlichen Frequenzbändern und -modulationen definiert. Dies ermöglicht eine Anpassung an Vorgaben von lokalen Gesetzen und Umgebungsumständen.

3.1.1.2 MAC

Das MAC-Layer besteht aus zwei Services, dem MAC Data Service und der MAC Sublayer Management Entity (MLME). Die beiden Services sind über die Schnittstellen MLME-SAP und MCPS-SAP für die „next higher layer“ erreichbar.

Es ist für das Senden und Empfangen von MAC Protocol Data Units (MPDUs) über die PHY Data Services zuständig [17].

3.1.2 Netzwerktopologie

Das Netzwerk besteht aus verschiedenen Geräten, die jeweils über eine Funkschnittstelle verfügen, welche die MAC und PHY des Standards implementiert.

Der Standard kann mit zwei verschiedenen Topologien betrieben werden, aus denen applikationsabhängig ausgewählt werden kann. Dies kann einerseits eine Stern-Topologie und andererseits einer peer-to-peer-Topologie sein, welche beispielhaft in Abb. 3.2 dargestellt sind.

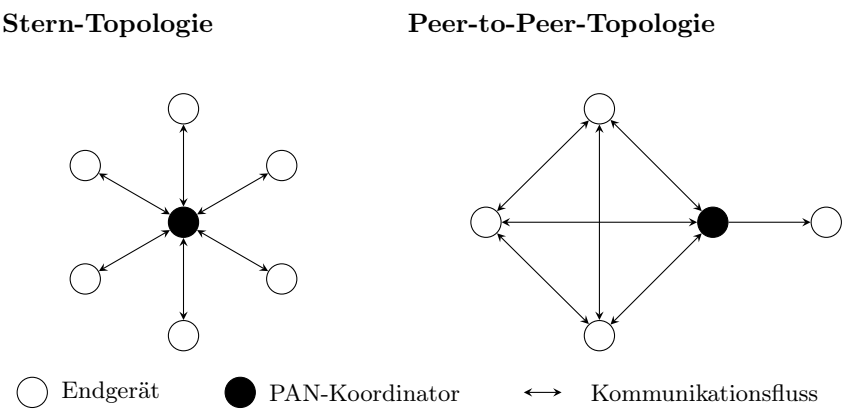

Abb. 3.2: Netzwerktopologien IEEE 802.15.4

3.1.3 Frame Struktur

Die MAC Frames werden an die PHY als PSDU übergeben, welche dann zum PHY payload werden. Die Struktur der PPDU ist in Abb. 3.3 dargestellt.

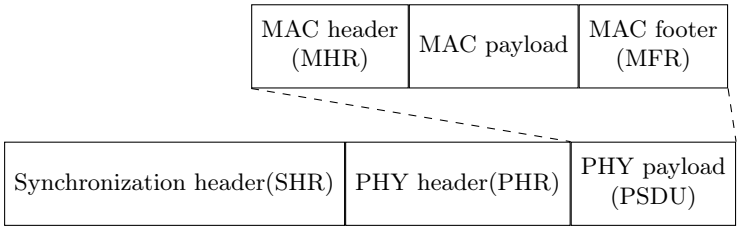

Abb. 3.3: Frame Struktur

Das MAC-Frame ist in drei Teile aufgeteilt: MAC Header (MHR), MAC-Payload und MAC Footer (MFR). Eine Übersicht über das gesamte MAC-Frame ist in Abb. 3.4 abgebildet. Im folgenden Abschnitt sind die einzelnen Komponenten des Frames genauer beschrieben.

Der **Frame Header** enthält unter anderem Informationen bezüglich der Version, des Types und der Adressierungsfelder des Frames. Es werden verschiedene MAC Frame Typen definiert:

- Beacon

<table>
<tr><td>Octets:
1/2</td><td>0/1</td><td>0/2</td><td>0/2/8</td><td>0/2</td><td>0/2/8</td><td>variable</td><td colspan="2">variable</td><td>variable</td><td>2/4</td></tr>
<tr><td rowspan="2">Frame Control</td><td rowspan="2">Sequence Number</td><td>Destination PAN ID</td><td>Destination Address</td><td>Source PAN ID</td><td>Source Address</td><td rowspan="2">Auxiliary Security Header</td><td colspan="2" rowspan="2">IE</td><td rowspan="2">Frame Payload</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Addressing fields</td></tr>
<tr><td colspan="7">MHR</td><td>Header IEs</td><td>Payload IEs</td><td>MAC Payload</td><td>MFR</td></tr>
</table>

Abb. 3.4: MAC Frame Struktur

- Data
- Acknowledgement
- MAC command
- Multipurpose
- Fragment oder Frak
- Extended

Beacons werden zur Synchronisation von Geräten und Koordinatoren verwendet. Für die Datenübertragung ist das Data Frame zuständig. Das Acknowledgement Frame wird als Antwort auf ein empfangenes Frame versendet und bestätigt den Empfang für den Sender. Mithilfe dieses Frametypes kann auch auf indirekt versendete Pakete in der Warteschlange hingewiesen werden, dies wird dann über das Frame-pending Bit signalisiert. Die Übertragung von Befehlen wie der Datenanfrage (mehr dazu in Kap.4) wird mit dem MAC Command realisiert. Dieser Frametyp wird noch in weitere Untertypen eingeteilt, die den genauen Befehl definieren. Die für diese Arbeit benötigten Befehle werden im Kap. 4 beschrieben. Das Multipurpose Frame basiert auf Information Elements (IEs), die das Frame sehr variabel machen. Es wurde als Erweiterung in der Revision von 2015 des Standards eingeführt und kann applikationsspezifische Aufgaben übernehmen. Mithilfe des Multipurpose Frames soll eine einfache Erweiterung des Standards ohne Änderung der Framestruktur möglich sein [16]. Die Frames können aus Latenzgründen auch in kleinere Frames geteilt werden, hierfür sind die Fragment und die Frak Frames verfügbar. So müssen Acknowledgement und Retransmissions nur für kleinere Frames gesendet werden.

Die in Abb. 3.4 erwähnten IEs wurden in der 2015 Version des Standards hinzugefügt. IEs sind Informationsfelder variabler Anzahl und Länge. Dies wurde aufgrund von den neu dazugekommen Betriebsmodi: TSCH und DSME eingeführt, da dort zusätzliche Informationen benötigt wurden. Durch die IEs müssen nicht bei jeder Erweiterung neue Frame Typen hinzugefügt werden.

Im **Payload** befinden sich, wenn IEs verwendet werden, ihre Payloads. Beim MAC command Frame befinden sich im Payload die Untertypen des Befehls. Ansonsten wird dort der eigentliche Payload des Frames gespeichert.

Abschließend enthält der **Footer** eine Prüfsumme, die über den Header und den Payload berechnet wird.

3.1.4 Kommunikationsarten

Der Standard bietet verschiedene Arten der Kommunikation: synchronisiert und unsynchronisiert. Die unsynchronisierte Kommunikation wird in älteren Versionen des Standards „Non-Beacon Mode“ genannt. In der aktuellsten Version jedoch nicht mehr namentlich genannt, sondern nur als „ohne Superframes“ betitelt. Die synchronisierte Kommunikation wird in verschiedene Superframes aufgeteilt:

- **Beacon:** Ein Koordinator sendet periodische Beacons, die verwendet werden, um die Geräte zu synchronisieren. Die Zeit zwischen Beacons wird in zwei Teile geteilt: Contention Access Period (CAP) und Contention-Free Period (CFP). Die CAP kann verwendet werden, um Frames mit CSMA-CA an den Koordinator zu senden. Die CFP bietet reservierte Slots für Übertragungen.
- **Deterministic and Synchronous Multichannel Extension (DSME):** Die grundlegende Funktion ähnelt dem des Beacon Superframes. Die Kommunikation besteht jedoch aus klar zugeordneten Zeitslots, die über mehrere Kanäle verteilt sein können.
- **Timeslotted Channel Hopping (TSCH):** Das Superframe besteht aus Zeitslots, die nacheinander wiederholt werden. Je Zeitslot dürfen immer Paare aus zwei Geräten einen Frame der maximalen Länge und einen Ack-Frame austauschen.
- **TVWS Multichannel Cluster Tree PAN (TMCTP):** Es kann ein großes Netzwerk aus Clustern baumartig erstellt werden. Die einzelnen Cluster bilden

PANs ab, die von eigenen Koordinatoren verwaltet werden. Die Endgeräte in den PANs senden immer nur an ihren nächstliegenden Knoten im Baum.

Es wird im Standard zwischen optionalen und nicht optionalen Funktionen unterschieden; die o.g. Superframes gehören dabei zu den optionalen.

3.1.5 Übertragungsarten

Im Standard werden drei verschiedene Arten der Übertragung beschrieben:

- Datenübertragung zu einem Koordinator von einem Gerät
- Indirekte Übertragung von einem Koordinator zu einem Gerät
- Übertragung zwischen zwei Peer-Geräten

Um Frames zu senden, kann dies entweder mit oder ohne Superframes passieren. Ohne Superframes können die Frames direkt gesendet werden. Bei Nutzung von Superframes passiert immer eine Art der Synchronisierung per Beacons. Dies muss bei den Peer-to-Peer Übertragungen beachtet werden, da die Daten dann nicht einfach zu einem anderen Gerät übertragen werden können.

Für LE-Anwendungen gibt es noch ein optionales Superframe (Coordinated Sampled Listening (CSL)), in dem die Endgeräte während der CAP Channel-Scans durchführen und damit in Arbeitszyklen funktionieren können. Als Alternative für Geräte, die keine Superframes verwenden, gibt es noch die Receiver-Initiated Transmission (RIT), in der von den Empfängern in periodischen Abständen Datenanfragen gestellt werden, auf die dann ein Sender antworten kann. Die RIT ermöglicht es damit auch in festgelegten Abständen mit Arbeitszyklen den Energieverbrauch zu senken.

3.1.6 Sicherheit

Durch die Natur von drahtloser Kommunikation sind diese Netze anfällig für verschiedene Arten von Angriffen. Darüber hinaus sind solche Netze häufig Low-Power and Lossy Networks (LLNs). LLNs zeichnen sich durch begrenzte Energie-, Speicher- und Rechenressourcen sowie durch verlustbehaftete Funkverbindungen aus. Der physikalische Zugriff auf das Medium ist ohne große Schwierigkeiten möglich, was das Mithören sehr einfach macht. Durch dies können Replay-Angriffe, also das Abspielen von davor aufgenommenen

Paketen, leicht möglich gemacht werden. Aufgrund dieser Restriktionen sind die einsetzbaren kryptografischen Verfahren und Sicherheitsprotokolle eingeschränkt. Insbesondere fehlen typischerweise eine Trusted Computing Base sowie leistungsfähige Zufallszahlengeneratoren, was die Implementierung komplexer kryptografischer Funktionen erschwert.

Die Verschlüsselung in IEEE 802.15.4 basiert auf symmetrischen Schlüsseln, die von höheren Protokollschichten bereitgestellt werden. Der Standard selbst definiert keine Mechanismen zur Schlüsselverteilung, sondern ausschließlich deren Verwendung.

IEEE 802.15.4 adressiert die klassischen Sicherheitsziele wie Vertraulichkeit, Authentizität und Schutz vor Replay Angriffen [17]. Frames können optional geschützt werden. Dafür wird ein Sicherheits-Header eingefügt, der das Schutzniveau und den verwendeten Schlüssel beschreibt, sowie ein fortlaufender Zähler, der alte Pakete erkennbar macht [17]. Der Standard definiert mehrere Sicherheitsstufen, entweder nur Integritätsschutz über einen MIC oder Integritätsschutz plus Verschlüsselung. Eine reine Verschlüsselung ohne Integritätsschutz ist veraltet und wird nicht empfohlen [17]. Authenticated Encryption with Associated Data (AEAD) kombiniert Vertraulichkeit und Integrität. Die Nutzdaten werden verschlüsselt, während bestimmte Header-Felder als „assoziierte Daten“ in die Integritätsprüfung einfließen, ohne selbst verschlüsselt zu werden. Der Empfänger entschlüsselt nur, wenn die Integritätsprüfung erfolgreich ist [17].

3.2 RIOT OS

3.2.1 Systemarchitektur

Die Entwicklung von RIOT verfolgt das Ziel, ein Betriebssystem mit minimalem Ressourcenverbrauch hinsichtlich RAM, ROM und Energie bereitzustellen. Gleichzeitig soll eine hohe Flexibilität erreicht werden, um unterschiedliche Hardwarekonfigurationen von 8- bis 32-Bit-Mikrocontrollern zu unterstützen. Ein weiteres Ziel ist die Reduktion von Code-Duplikation sowie die Sicherstellung der Portabilität über verschiedene Hardwareplattformen hinweg. Darüber hinaus zielt RIOT auf eine einfache Programmierbarkeit und die Unterstützung von Echtzeitfähigkeiten für zeitkritische Anwendungen ab [6].

RIOTs Softwarestruktur ist modular aufgebaut, wobei alle Komponenten zur Kompilierzeit aggregiert werden. Dadurch besteht das System ausschließlich aus den für den jeweiligen Anwendungsfall benötigten Modulen, was sowohl den Speicherbedarf als auch

die Systemkomplexität reduziert. Ein Überblick der Module ist in Abb. 3.5 gegeben. Die Architektur trennt klar zwischen hardwareunabhängigen und hardwareabhängigen Komponenten. Der Kernel (core) stellt grundlegende Betriebssystemfunktionen bereit, während Systemdienste (sys, sys/net) und externe Bibliotheken (pkg) zusätzliche Funktionalität ergänzen. Die Hardware-Abstraktionsschicht umfasst die Module cpu, boards, periph und drivers und kapselt plattformspezifische Details. Die eigentliche Anwendungslogik ist in der Applikationschicht angesiedelt und damit von der Hardware entkoppelt, wodurch Portabilität und Wartbarkeit des Systems erhöht werden [6].

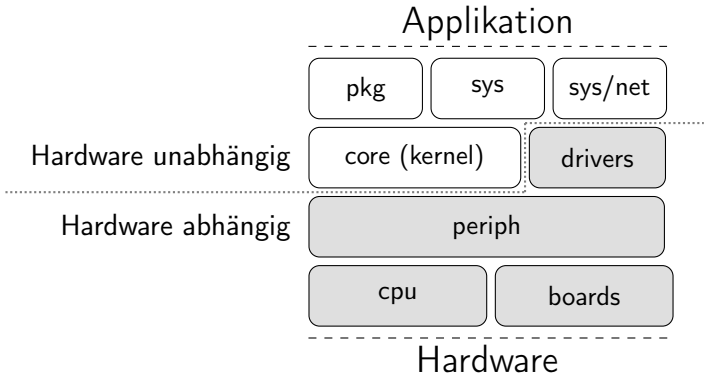

Abb. 3.5: RIOT Software-Struktur

3.2.2 Netzwerkkomponenten

Die grundlegende Struktur der Netzwerkschnittstellen in RIOT OS ist in Abb. 3.6 dargestellt. Netdev ist für die Netzwerkstacks eine generische Low-Level Schnittstelle auf die Gerätetreiber. Die Schnittstelle für die Applikationsschicht wird durch die Sock-API abgebildet [6].

Die direkte Verwendung von netdev in Netzwerkstacks wie GNRC wirkt sich nachteilig auf die Funktionalität und Weiterentwicklung von IEEE 802.15.4 aus. Einerseits bestehen die Probleme darin, dass Teile des Standards verteilt in netdev und GNRC implementiert sind. Andererseits verursacht die Verwendung von netdev Codeduplizierung der PHY/MAC-Komponenten, da netdev hardwareabhängig ist und für jedes Gerät separat implementiert werden muss. In netdev lassen sich die MAC-Funktionalitäten nicht frei konfigurieren. Dies hat zur Folge, dass einige MAC-Funktionen ausschließlich dann verfügbar sind, wenn sie hardwareseitig implementiert sind. Zudem fehlt ein Mechanismus,

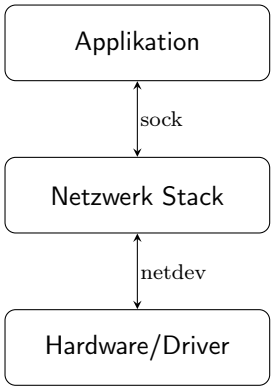

Abb. 3.6: RIOT OS Netzwerkschnittstellen

um anzugeben, welche MAC-Funktionen eine konkrete netdev-Implementierung bereitstellt. Dadurch, dass Teile des Standards direkt in GNRC implementiert sind, gibt es keine Möglichkeit, diese in anderen Netzwerkstacks wiederzuverwenden [18].

Diese Nachteile haben den Anreiz für eine dedizierte HAL Implementierung für IEEE 802.15.4 gegeben. Diese soll für den Standard netdev ersetzen und bietet eine einheitliche Schnittstelle für eine MAC Implementierung und darauf aufbauend für die Netzwerkstacks an. Infolgedessen wurde das SubMAC Layer implementiert, welches für CSMA-CA, Retransmissions und das Speichern von Teilen der MAC PIB zuständig ist. Des Weiteren macht es das Layer möglich Geräte, die hardwareseitiges CSMA-CA nicht unterstützen, softwareseitig zu erweitern. In Abb. 3.7 wird der Vergleich der alten gegen die neue Struktur und deren Schnittstellen skizziert, dabei ist die skizzierte MAC noch zu implementieren.

3.2.3 Unterstützte Transceiver

In RIOT OS werden von der neuen Radio-HAL die in Tabelle 3.1 beschriebenen Tranceiver unterstützt. Die dabei unterstützten PHY Funktionen sind ebenfalls aufgeschlüsselt.

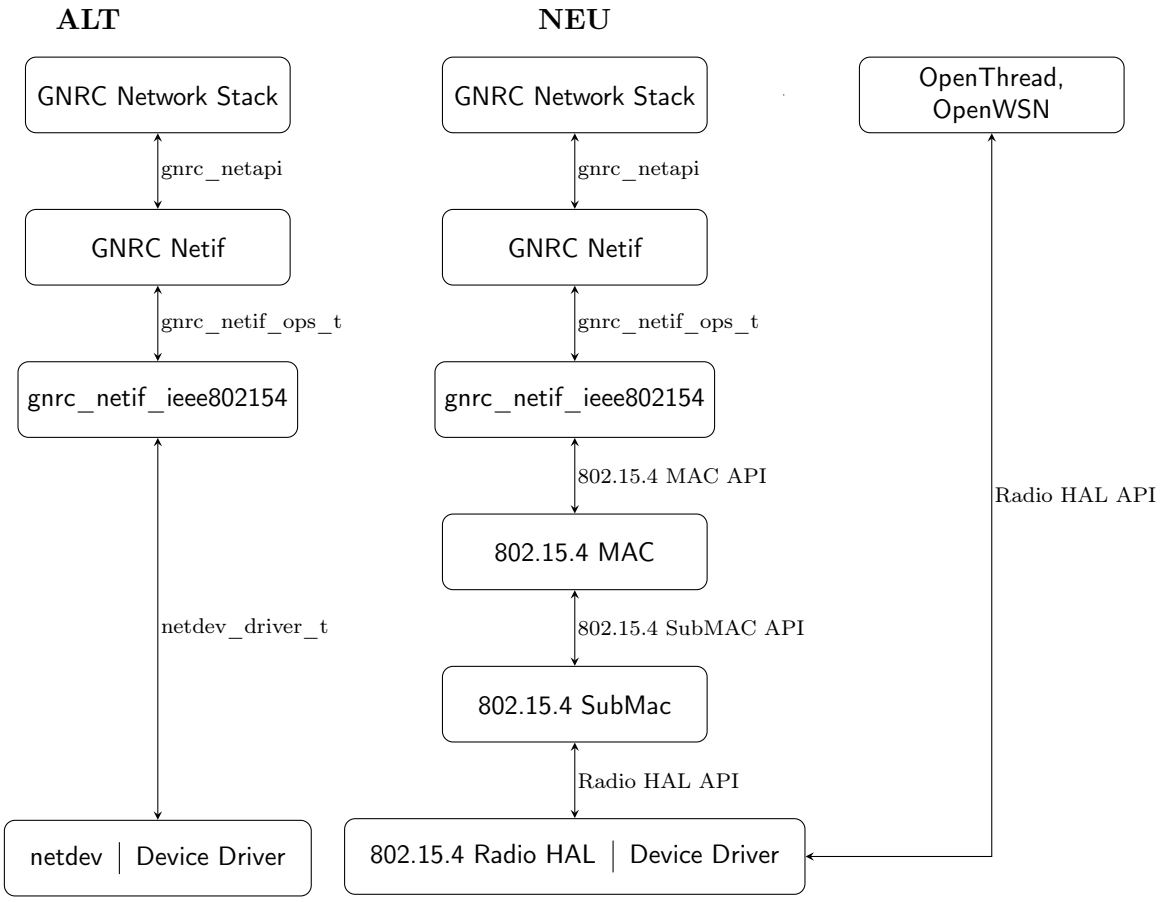

Abb. 3.7: RIOT Netzwerk Architektur

		Hardwareunterstützung		
MCU	Hersteller	CSMA-CA	CSMA-CA/ Retransmission	Auto-ACK/ ACK Timeout
CC2538	Texas Instruments	X		
KW2XRF	NXP			X
MRF24J40	Microchip	X	X	
NRF52	Nordic Semiconductor			
ESP32	Espressif	X		X

Tabelle 3.1: Unterstützte Radio HAL Transceiver

4 Anforderungen und Analyse

In diesem Kapitel wird erläutert, welche Teile des Standards implementiert werden sollen und was dafür benötigt wird.

4.1 Erforderliche Schnittstellen von IEEE 802.15.4

In den folgenden Abschnitten werden einzeln für die verschiedenen Vorgänge, die über das MAC verfügbar sein sollen, die benötigten Schnittstellen und MAC Funktionalitäten analysiert und festgelegt.

Die Services im Standard beschreiben immer den Informationsfluss zwischen dem MAC-Layer und dem nächsthöheren Layer. Das nächsthöhere Layer ist das des Service-Benutzers, was beispielsweise ein Netzwerkstack oder OpenThread sein kann.

Infolgedessen werden vier Primitive beschrieben:

- Request - Anforderung eines Services
- Indication - Meldung eines Events an das nächsthöhere Layer
- Response - Abschluss einer Indication
- Confirm - Rückmeldung des Abschlusses in Folge eines Request

Daraus ergibt sich die immer gleiche Reihenfolge *Request* → *Indication* → *Response* → *Confirm*. Die Verfügbarkeit der Primitiven wird in Abb. 4.1 grafisch dargestellt.

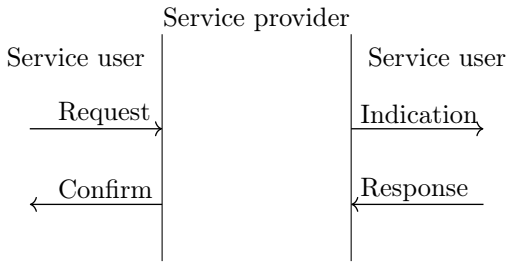

Abb. 4.1: Service Primitiven

4.1.1 Erstellung und Verwaltung von PANs

Für das Erstellen von PANs existieren verschiedene Methoden, abhängig davon, ob der Koordinator mit oder ohne Superframes agiert.

Beim Betrieb mit Superframes wird das PAN durch einen Energy Detection (ED)-Scan, gefolgt von einem aktiven Scan, gestartet. Die Ergebnisse dieser Scans dienen der nächsthöheren Layer als Grundlage zur Auswahl eines geeigneten Kanals, welcher anschließend vom MAC-Schicht konfiguriert wird.

Im Betrieb ohne Superframes beginnt die Erstellung mit dem Reset des MAC-Layers. Nach dem Zurücksetzen kann ein ED-Scan folgen, welcher vom Standard jedoch nicht zwingend vorgeschrieben ist. Um vorhandene Koordinatoren ausfindig zu machen, wird ein aktiver Scan ausgeführt. Dieser wird in 4.1.3.1 näher beschrieben. Vor dem Starten des PANs muss mit MLME-SET.request die PANId und die Adresse des Gerätes gesetzt werden. Danach wird mit einem MLME-START.request mit der Angabe der PANId und des Channels das PAN gestartet. Der Ablauf ist auch in Abb. 4.2 skizziert.

4.1.2 Einem PAN beitreten

Um einem PAN beizutreten, gibt es zu einem die Möglichkeit, ohne vorherige Kommunikation auf dem Channel des Koordinators an dessen Adresse zu senden. Als optionale Möglichkeit wird im Standard der MLME-ASSOCIATE.request gegeben [17, Kap. 10.21] (siehe Sequenzdiagramm 4.3). Die Zuordnung zum Netzwerk kann entweder mit einer indirekten Datenanfrage für die Antwort geschehen oder als direkte Antwort, dann wird das Verfahren zu einer Schnellzuordnung.

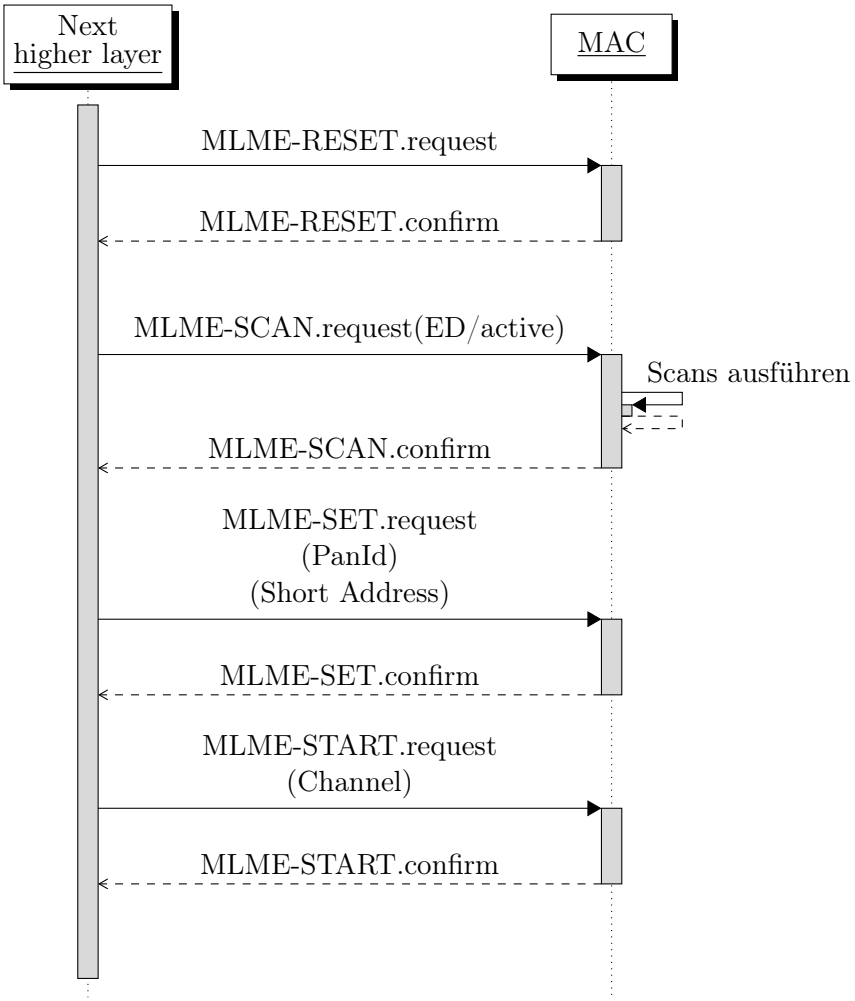

Abb. 4.2: Starten eines PANs

Für die Anfrage des Beitritts wird der MAC-Befehl *Association Request* versendet. Die Antwort des Koordinators darauf wird mit *Association Response* durchgeführt.

4.1.3 Aus einem PAN austreten

Wenn die Assoziierung mit einem PAN mithilfe der MLME-ASSOCIATE.request Primitive erfolgt ist, kann ein Gerät anschließend mittels der MLME-DISASSOCIATE.request-Primitive aus diesem PAN austreten.

Der Disassoziierungsvorgang kann entweder von einem Koordinator oder von einem mit diesem assoziierten Gerät initiiert werden. Ein Koordinator kann über diese Primitive

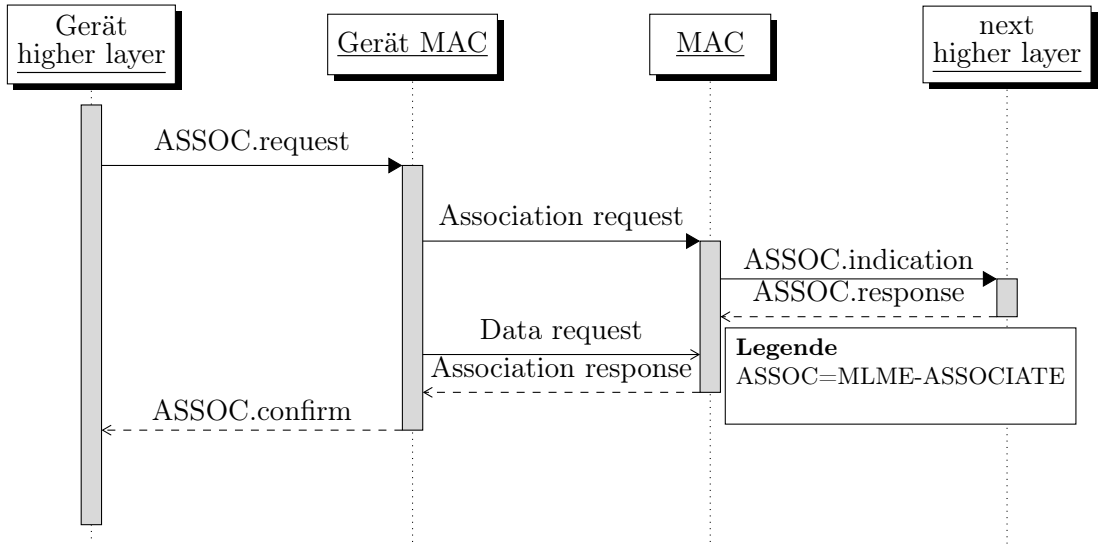

Abb. 4.3: Ablauf Beitreten zu einem PAN

einem Gerät signalisieren, dass es sich vom PAN trennen soll. Die Übertragung des Disassoziierungsbefehls kann direkt oder indirekt erfolgen. Schlägt diese Übertragung fehl, unabhängig davon, ob sie direkt oder indirekt durchgeführt wurde, betrachtet der Koordinator das betreffende Gerät als nicht mehr Teil des PANs.

Möchte ein Gerät selbst die Disassoziierung initiieren, erfolgt dies ausschließlich über eine direkte Übertragung. Schlägt dabei das Kanalzugriffsverfahren fehl, wird die nächsthöhere Schicht entsprechend benachrichtigt. Das Gerät betrachtet sich als disassoziiert, sobald entweder ein ACK auf den Disassoziierungsbefehl empfangen wird oder kein ACK eingeht.

Der dazugehörige MAC-Befehl ist die *Disassociation Notification*.

4.1.3.1 Aktiver Scan

Der aktive Scan kann verwendet werden, um Koordinatoren im PAN zu lokalisieren. Der Scan sendet in jedem Channel eine Beacon-Anfrage und wartet auf eingehende Beacons. Anschließend werden die gefundenen Koordinatoren inklusive ihrer zugehörigen PAN-ID und Adresse zurückgegeben.

Der dazugehörige MAC-Befehl ist der *Beacon Request*.

4.1.3.2 Passiver Scan

Der passive Scan kann wie im aktiven Scan zur Findung von Koordinatoren verwendet werden. Er sendet jedoch nicht aktiv eine Beacon Anfrage, sondern meldet nur die empfangenen Beacons. Dieser Scan kann nur Koordinatoren die mit Superframes, also periodischen Beacons arbeiten, erkennen.

4.1.3.3 ED Scan

Der Energie-Scan läuft ähnlich wie der Aktive-Scan ab, jedoch wird statt der Beacon Anfrage der Energiewert des jeweiligen Channels gemessen und zurückgemeldet. Dieser Energiewert beschreibt die während der Messdauer empfangene Funkenergie innerhalb des betrachteten Kanals. Ein höherer Energiewert spricht für einen stärker belasteten Kanal und kann von der nächsthöheren Schicht zur Auswahl eines möglichst störungsarmen Kanals verwendet werden. Für den ED-Scan werden Funktionalitäten benötigt, die bisher in der Radio-HAL nicht implementiert sind.

4.1.4 Übertragung

Um Daten zu übertragen wird die MCPS-DATA Primitive bereitgestellt. Die Übertragung kann entweder mit oder ohne der Anfrage eines ACKs durchgeführt werden. In Abb.4.4 wird der Ablauf der Übertragung mit ACK skizziert, der Ablauf ohne ACK ist derselbe bis auf das skizzierte ACK. Innerhalb der Primitiven kann auch zwischen direkten oder indirekten Übertragungen ausgewählt werden. Beim Modus ohne Superframes werden bei direkten Übertragungen die Frames einfach mit CSMA-CA versendet. Für die indirekten Übertragungen wird eine weitere Primitive (MLME-POLL) bereitgestellt, die es Geräten ermöglicht, verfügbare Frames beim Koordinator anzufragen.

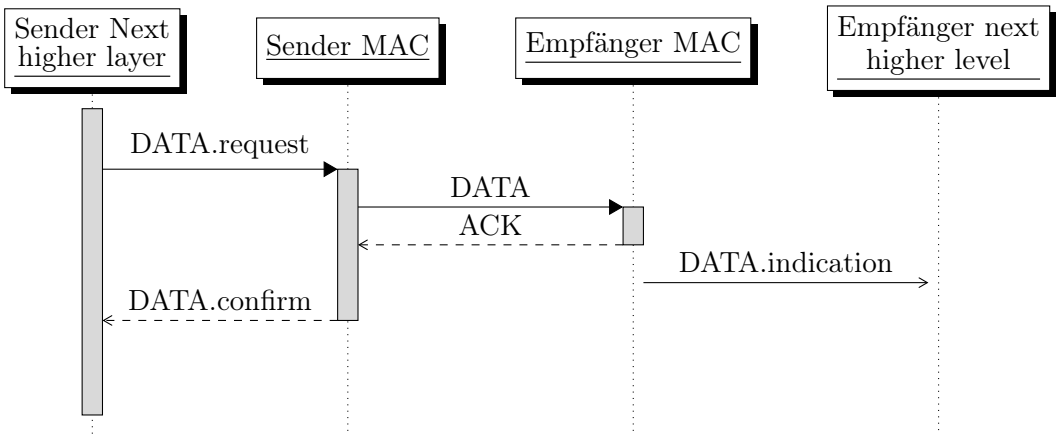

Abb. 4.4: Ablauf Übertragung mit ACK

4.1.5 Zusammenfassung der Schnittstellen

In diesem Abschnitt werden die Schnittstellen, die für die Funktion der indirekten Übertragungen benötigt werden, zusammengefasst. In Tabelle 4.1 befinden sich alle Schnittstellen die der MLME zugeordnet werden. Folgend in Tabelle 4.2 sind alle MCPS Schnittstellen dargestellt.

Schnittstelle	Request	Indication	Response	Confirm	Zweck
ASSOCIATE	X	X	X	X	Zuordnung eines Gerätes zu einem PAN
DISASSOCIATE	X	X		X	Trennung eines Gerätes von einem PAN
GET	X			X	Lesen von Werten aus der PIB
POLL	X			X	Daten von einem Koordinator anfragen
RESET	X			X	MAC-Sublayer reset
SCAN	X			X	Scan von Channels
SET	X			X	Setzen von Werten in der PIB
START	X			X	Starten eines PANs

Tabelle 4.1: Schnittstellen MLME

Schnittstelle	Request	Indication	Confirm	Zweck
DATA	X	X	X	Anfrage einer Übertragung
PURGE	X	X		Enfernen einer MSDU aus der Warteschlange

Tabelle 4.2: Schnittstellen MCPS

In Tabelle 4.3 sind die in diesem Kapitel beschriebene MAC Commands zusammengefasst. Diese werden dann im Kap. 6 Implementierung genauer beschrieben.

MAC Command	Zweck
Association Request	Anfrage an einen Koordinator zum Beitreten eines PANs
Association response	Antwort des Koordinator auf ein *Association Request*
Disassociation Notification	Benachrichtigung von Koordinator oder Gerät, dass aus dem PAN ausgetreten werden soll/möchte
Beacon request	Anfrage in einen bestimmten Channel nach dem Beacon eines Koordinators

Tabelle 4.3: Übersicht MAC-Commands

4.2 Anforderungen an RIOT OS

Die in Kap. 4.1 beschriebenen Schnittstellen und Services des IEEE 802.15.4 Standards müssen im Rahmen der Implementierung in die Systemarchitektur von RIOT OS intigriert werden. Wie in Kap. 3.2.2 beschrieben muss dafür die zu implementierende MAC, die in Abb. 3.7 zu sehen ist, integriert werden. Die Schnittstellen aus 4.1 bilden die Verbindung zu den „Upper Layern“. Die Anbindung der unterhalb der MAC liegenden Schichten erfolgt über die in RIOT OS vorgesehenen SubMAC-Schnittstellen sowie über die IEEE 802.15.4 spezifischen Hilfsfunktionen, die von der Radio-HAL bereitgestellt werden. Diese abstrahieren die zugrundeliegende Funkhardware und ermöglichen eine klare Trennung zwischen MAC-Logik und hardwareabhängigen Komponenten.

Die bestehende Implementierung des SubMAC-Layers in *RIOT* bietet derzeit keine vollständige Unterstützung für alle über das Radio-HAL angebundenen Transceiver. Insbesondere ist die Integration bestimmter Geräteklassen, wie beispielsweise AT86-basierter Transceiver, nur eingeschränkt möglich. Zudem ist die automatische Bestätigung empfangener Frames (Auto-ACK), wie sie im IEEE 802.15.4-Standard vorgesehen ist, bislang nur für solche Radios verfügbar, die diese Funktionalität hardwareseitig bereitstellen.

Für die Erweiterung des SubMACs um diese beiden Funktionalitäten bestehen momenten offene Pull Requests im Repository von *RIOT* [1].

[1] vgl. Pull Requests:
AT86: https://github.com/RIOT-OS/RIOT/pull/21674
Auto-ACK: https://github.com/RIOT-OS/RIOT/pull/21973

5 Design

Die mit dem MAC Layer assoziierten Komponenten werden in Abb. 5.1 dargestellt. Im Design wird die konzeptionelle Umsetzung der in der Analyse definierten Primitiven und Funktionen dargestellt und es wird festgelegt, welche davon konkret implementiert werden.

Abb. 5.1: Komponentendiagramm der MAC-Integration

Da die Radio-HAL aktuell keine ED-Scan Funktionalität bereitstellt und diese für indirekte Übertragungen nicht zwingend erforderlich ist, wird der ED-Scan auf spätere Weiterentwicklungen verschoben. Der passive Scan wird nicht berücksichtigt, da er ausschließlich Koordinatoren erkennt, die mit Superframes agieren und dauerhaft Beacons aussenden.

5.1 Callback-Architektur

Um asynchrone Ereignisse unabhängig vom Ausführungsmodell an das Upper-Layer zu melden, nutzt der MAC-Layer ein Callback-Interface. Dieses Modell passt zu den asynchronen Funkabläufen, bei denen Ereignisse durch die Hardware und das Medium bestimmt werden. Die Callback-Lösung erhöht die Portabilität und erlaubt die Einbindung in unterschiedliche Scheduler- und Event-Modelle. Damit bleibt der MAC-Layer generisch, während das Upper-Layer flexibel an verschiedene Architekturvorgaben und Anwendungskontexte angepasst werden kann. Durch eine genaue Beachtung des ISR-Kontextes kann das Layer mit dieser Architektur theoretisch auch synchron betrieben werden, sofern die Aufrufkette berücksichtigt wird.

5.2 State-Machine und Events

Um die Logik klar verständlich und wartbar zu halten, wird für das Design eine State-Machine verwendet. Dies fördert auch die Erweiterbarkeit des MAC Layers.

In Abb. 5.2 wird die State-Machine mit ihren Zuständen und Transitionen abgebildet. Die Zustände bilden die Rollen und Vorgänge des MAC-Layers ab. *IDLE* als Startzustand, *COORD* für den Koordinator, *DEVICE* für assoziierte Endgeräte sowie *SLEEP* als Zustand mit deaktiviertem Transceiver und inaktivem MAC-Layer. Ein aktiver Scan wird in *SCAN* modelliert. Passive und ED-Scans werden in dieser Arbeit nicht behandelt und sind deshalb in diesem Design nicht enthalten. Die Assoziierung wird durch den Zustand *ASSOC* abgebildet, um nach dem versendeten *Association Request* auf die eintreffende *Association Response* zu warten. Ein Beitritt ist nur aus *IDLE* möglich, da der Koordinator keinem PAN beitritt und ein Gerät jeweils nur einem PAN angehören kann. Das Event *MLME_RESET_REQ*, das durch die *MLME.RESET* Primitive ausgelöst wird, setzt die State-Machine jederzeit nach *IDLE* zurück.

Tabelle 5.1 listet die verwendeten Ereignisse auf und gruppiert sie nach der Quelle von der das Event stammt. In Tabelle 5.2 sind die Zustände und die dort verarbeiteten Events in diesen aufgeschlüsselt.

5.3 Upper-Layer Schnittstellen

In Tabelle 5.3 sind die im Design vorgesehenen Upper-Layer Primitiven und deren Rückmeldungen zusammengefasst. Die Parameter in Tabelle 5.3 sind bewusst reduziert. Das Design konzentriert sich auf die für die indirekte Übertragung notwendigen Funktionen und verzichtet auf Optionen, die im betrachteten Szenario nicht benötigt werden (z. B. Superframe-bezogene Parameter oder erweiterte Scan-Typen). Das Parsing des Headers ist in der RIOT OS Implementierung auf dem Stand der 2006 Version des Standards. Dadurch werden viele der neuen Funktionen nicht unterstützt. Unter anderem können Information Elements (IEs) nicht verarbeitet werden. Parameter, die mit den IEs zusammenhängen, werden aus diesem Grund ebenfalls weggelassen.

In den folgenden Abschnitten werden die Abweichungen der Parameter vom Standard beschrieben und begründet.

Event (Kürzel)	**Bedeutung / Quelle**
Higher-Layer Requests	
COORD_START	Start der Koordinatorrolle
MLME_SCAN_REQ	Start eines aktiven Scans
MLME_RESET_REQ	MLME-RESET Request
MLME_ASSOC_REQ	MLME-ASSOCIATE Request
MLME_POLL_REQ	MLME-POLL Request
MLME_ASSOC_RES	MLME-ASSOCIATE Response
MLME_DISASSOC_REQ	MLME-DISASSOCIATE Request
MCPS_DATA_REQ	MCPS-DATA Request
TX_REQUEST	Sendeanforderung aus der MAC-Logik
SLEEP	Übergang in Schlafmodus
WAKE	Aufwachen aus Schlafmodus
Empfangene Frames	
BEACON	Beacon
DATA	Datenframe
CMD_DATA_REQ	Data Request Command
CMD_BEACON_REQ	Beacon Request Command
CMD_ASSOC_REQ	Association Request Command
CMD_ASSOC_RES	Association Response Command
CMD_DISASSOC	Disassociation Notification Command
Timer/Timeouts	
SCAN_TIMER	Scan-Timer abgelaufen
SCAN_DONE	Scan abgeschlossen
ASSOC_TIMEOUT	Timeout während Assoziierung

Tabelle 5.1: Ereignisse der MAC-State-Machine

Event	**IDLE**	**SCAN**	**COORD**	**DEVICE**	**ASSOC**	**SLEEP**
Higher-Layer Requests						
MLME_SCAN_REQ	X					
COORD_START	X					
MLME_RESET_REQ	X	X	X	X		
MLME_ASSOC_REQ	X					
MLME_POLL_REQ	X			X	X	
MLME_ASSOC_RES			X			
MLME_DISASSOC_REQ			X	X	X	
TX_REQUEST	X		X	X		
MCPS_DATA_REQ	X		X	X		
SLEEP	X		X	X		
WAKE						X
Timer/Timeouts						
SCAN_TIMER		X				
SCAN_DONE		X				
ASSOC_TIMEOUT					X	
Empfangene Frames						
BEACON	∼	X	∼	∼		
DATA	X	X	X	X		
DISASSOC	∼	∼	∼	X		
Empfangene MAC-Commands						
CMD_DATA_REQ	X	X	X	∼		
CMD_BEACON_REQ	∼	∼	X	∼		
CMD_ASSOC_REQ	∼	∼	X	∼		
CMD_ASSOC_RES	∼	∼	∼	∼	X	
CMD_DISASSOC	∼	∼	X	X	X	

Tabelle 5.2: Events pro Zustand (X = erlaubt, ∼ = ignoriert, leer = ungültig)

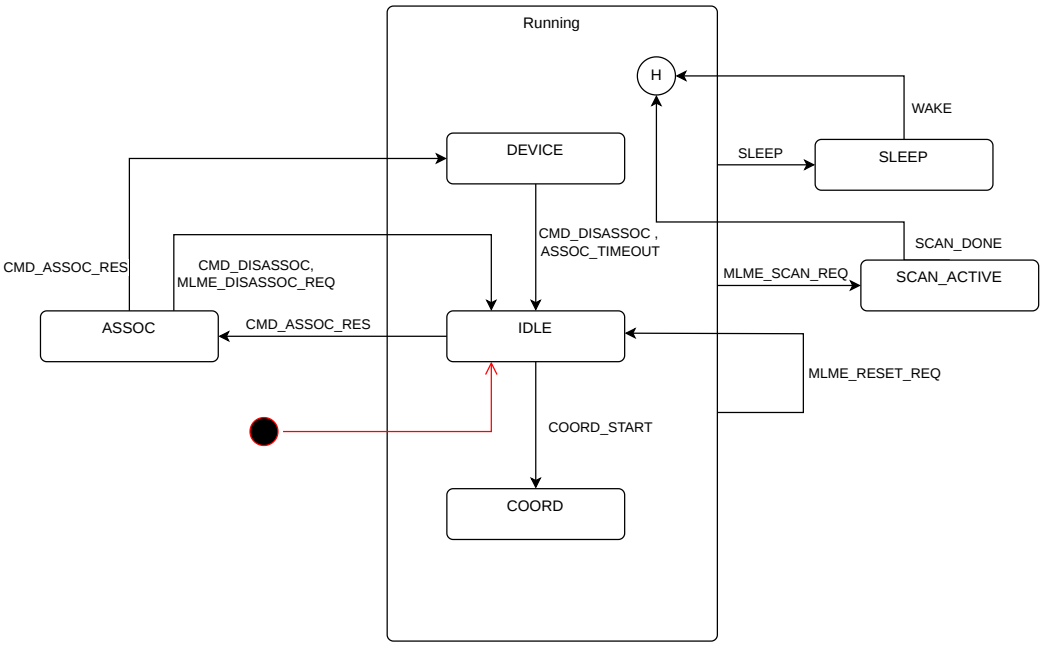

Abb. 5.2: State-Machine des MAC-Layers

Primitive	**Parameter**	**Rückmeldung**
MLME-SCAN.request	Scan-Typ, Kanäle, Dauer, Ergebnis-Puffer	MLME-SCAN.confirm
MLME-GET.request	Attribut-ID	MLME-GET.confirm (Wert/Status)
MLME-SET.request	Attribut-ID, Wert	MLME-SET.confirm (Status)
MLME-START.request	Kanal	MLME-START.confirm
MLME-ASSOC.request	Koordinator-Adresse, PAN-ID, Capability	MLME-ASSOC.confirm
MLME-ASSOC.response	Ziel-Adresse, Status, Short-Addr	–
MLME-POLL.request	Koordinator-Adresse/Mode, PAN-ID	MCPS-DATA.indication
MCPS-DATA.request	Src/Dst-Adresse, PAN-ID, MSDU, Handle, ACK, indirekt	MCPS-DATA.confirm

Tabelle 5.3: Upper-Layer Schnittstellen

5.3.1 MLME-SCAN

5.3.1.1 Request

Im Standard wird der *LinkQualityScan* als Parameter angegeben aber in der Radio-HAL wird die Verbindungsqualität immer mitgegeben. Somit macht es keinen Sinn, diese explizit deaktivierbar zu machen. Auf Grund der Version des Headerparsings wird das Unterdrücken der *PANid/Sequenznummer* nicht unterstützt, was von neueren Versionen beim Scan vorgesehen ist.

5.3.1.2 Confirm

Im Confirm wird die *DetectedCategory* angegeben, die die PHY Kategorie, die gefunden wurde anzeigt. Dies wird von der aktuellen Radio-HAL nicht unterstützt.

5.3.2 MLME-GET

Der Standard erlaubt den Zugriff auf eine umfangreiche Menge an PIB-Attributen. Hier werden nur die für die betrachteten Abläufe relevanten Attribute berücksichtigt.

5.3.3 MLME-SET

Analog zu *MLME-GET* werden nur die benötigten PIB-Attribute abgedeckt. Standardoptionen ohne Einfluss auf die indirekten Übertragungen sind nicht Teil der Schnittstelle.

5.3.4 MLME-START

Der Standard sieht weitere Superframe- und Beacon-Parameter vor. Da Superframes nicht implementiert werden, sind diese Parameter in der Schnittstelle reduziert.

5.3.5 MLME-ASSOC

Im Standard existieren zusätzliche Optionen zur Assoziierung. Das Design beschränkt sich auf die Parameter, die für den Beitritt zu einem PAN im betrachteten Szenario erforderlich sind.

5.3.6 MLME-ASSOC.response

Die Antwort enthält im Standard weitere Informationen. In diesem Design werden nur *Status* und *Short-Address* verwendet, da zusätzliche Felder hier nicht genutzt werden.

5.3.7 MLME-POLL

Der Standard erlaubt weitere Konfigurationen für Polling. Die Schnittstelle bildet nur die für indirekte Übertragungen notwendige Adressierung und den PAN-Kontext ab.

5.3.8 MCPS-DATA

Der Standard definiert zusätzliche Parameter und Optionen für den Datenversand. Hier werden die für direkte und indirekte Übertragungen benötigten Felder abgebildet; weitergehende Optionen bleiben unberücksichtigt.

6 Implementierung und Tests

6.1 Projektstruktur

Das Projekt ist in mehrere Module unterteilt. Die Headerdatei der API für die höheren Layer liegt in `sys/include/net/ieee802154`. Die Implementierungen dieser liegen in `sys/net/link-layer/ieee802154`. Die Implementierung des öffentlichen Headers ist für die Übersichtlichkeit und die Aufgabentrennung in mehrere Komponenten getrennt. Die Funktionen, die nur innerhalb der MAC verwendet werden, sind für die einzelnen Komponente jeweils in einem eigenen Header angegeben. Die Struktur der Header und deren Implementierung werden in Abb. 6.1 visuell dargestellt.

Im Header `mac_tx.h` werden sämtliche Funktionen definiert, die den Versand von Paketen betreffen. Der Header `mac_bh.h` enthält alle Funktionen, die dem „Bottom Half Processor“ des SubMAC-Layers zugeordnet sind. Die Verwaltung der Warteschlange für das indirekte Senden wird durch die in `mac_queue.h` definierten Funktionen realisiert. Des Weiteren sind in `mac_fsm.h` alle Funktionen zusammengefasst, die im Zusammenhang mit der Zustandsmaschine des MAC-Layers stehen. Die Initialisierung des MAC-Layers erfolgt über die in `mac_internal.h` definierten Funktionen. Abschließend umfasst `mac_pib.h` sämtliche Funktionen, die die PAN Information Base (PIB) betreffen.

6.2 Zentrale Datenstruktur

Die zentrale Datenstruktur `ieee802154_mac_t` repräsentiert den vollständigen Zustand des MAC-Layers. Sie umfasst unter anderem:

- den aktuellen Betriebszustand (`ieee802154_mac_state_t`),
- die PAN Information Base (PIB) (`ieee802154_pib_t`),

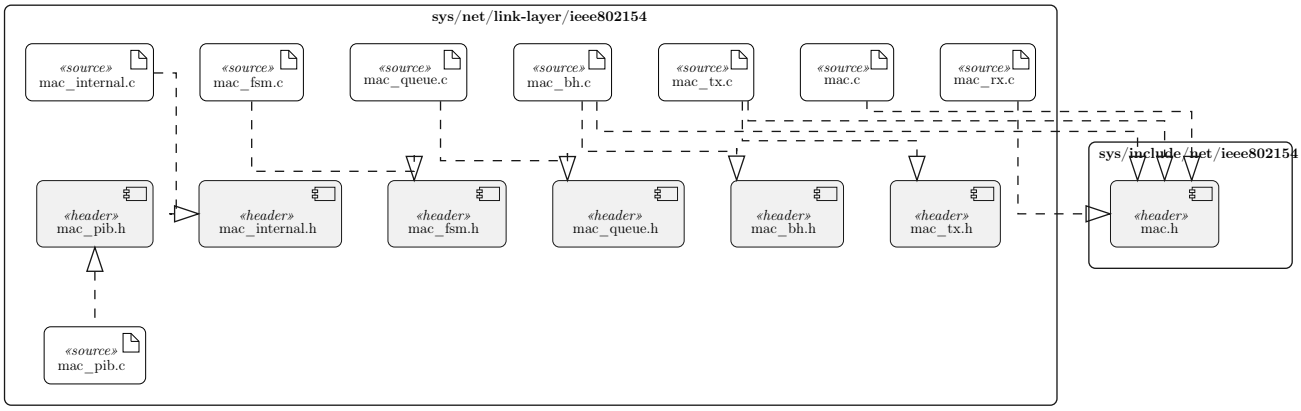

Abb. 6.1: Projektstruktur

- eine Instanz des SubMAC (`ieee802154_submac_t`),
- sowie verschiedene Timer zur Behandlung zeitkritischer Abläufe (z. B. ACK-Timeouts oder Scan-Operationen).

Zusätzlich enthält die Struktur Mechanismen zur Synchronisation wie Mutexe, um konkurrierende Zugriffe auf gemeinsame Ressourcen zu vermeiden.

6.3 PAN Information Base (PIB)

Die PAN Information Base (PIB) wird durch die Struktur `ieee802154_pib_t` abgebildet und enthält alle konfigurierbaren Parameter des MAC-Layers, wie beispielsweise PAN-ID, Adressen oder CSMA-CA-bezogene Parameter.

Zur generischen Verwaltung der Attribute wird ein typisiertes Zugriffssystem verwendet, bestehend aus:

- einer Attribut-Enumeration (`ieee802154_pib_attr_t`),
- einem generischen Wertecontainer (`ieee802154_pib_value_t`),
- sowie Metadaten zur Beschreibung der Attribute.

Dieses Design ermöglicht eine flexible Implementierung der MLME-GET und MLME-SET Primitiven. Die PIB-Implementierung folgt einem tabellengetriebenen Ansatz. Jedes Attribut wird durch einen Metadateneintrag beschrieben, der den Datentyp, die

Zugriffsrechte, die Position innerhalb der PIB-Struktur sowie Default-, Minimal- und Maximalwerte enthält. Dadurch können Zugriffe auf PIB-Attribute generisch implementiert werden, ohne für jedes Attribut separate Zugriffsfunktionen bereitzustellen.

Die Funktionen `ieee802154_mac_mlme_set` und `ieee802154_mac_mlme_get` implementieren den generischen Schreib- bzw. Lesezugriff auf PIB-Attribute. Auf Basis der Attributmetadaten wird zur Laufzeit der Speicherbereich des gewünschten Attributs bestimmt und abhängig vom hinterlegten Datentyp verarbeitet. Durch die Verwendung von Offsets und Größenangaben wird ein einheitlicher Zugriff auf heterogene Attributtypen ermöglicht, darunter boolesche Werte, Ganzzahlen, Adressen und Bytesequenzen.

Die Initialisierung der PIB erfolgt in `ieee802154_mac_pib_init`. Dabei werden zunächst alle Attribute mit den in der Metadatentabelle hinterlegten Standardwerten belegt. Anschließend werden die Sequenznummern für Beacon- und Datenframes mit zufälligen Startwerten initialisiert.

Da auf die PIB potenziell aus mehreren Ausführungskontexten zugegriffen wird, werden Lese- und Schreibzugriffe über einen Mutex synchronisiert.

6.4 Warteschlange für die indirekten Übertragungen

Für die Verwaltung ausgehender Frames wird eine mehrstufige Warteschlangenstruktur verwendet. Einzelne Frames werden durch `ieee802154_mac_tx_desc_t` beschrieben, die neben den Nutzdaten auch Metainformationen wie den aktuellen Sendezustand enthalten.

Diese Deskriptoren werden in einer ringpufferbasierten Warteschlange (`ieee802154_mac_txq_t`) organisiert. Für indirekte Übertragungen, wie sie im IEEE 802.15.4-Standard vorgesehen sind, wird zusätzlich eine strukturierte Sammlung solcher Warteschlangen (`ieee802154_mac_indirect_q_t`) verwendet, die eine Zuordnung zu Zielknoten ermöglicht.

6.5 Zeitabhängige Funktionen

Zur Behandlung zeitabhängiger Abläufe verfügt das MAC-Layer über eine periodisch ausgeführte Tick-Funktion. Diese dient als zentrale Zeitbasis für interne Timeoutme-

chanismen und überwacht insbesondere laufende Assoziationsvorgänge, Polloperationen sowie ausstehende indirekte Übertragungen. Die Tick-Funktion läuft nur, während sich Frames in der ausgehenden Warteschlange befinden oder auf ein indirektes Frame gewartet wird. Ist dies nicht der Fall wird der wiederkehrende Timer deaktiviert.

Bei jedem Tick wird zunächst ein interner Zeitzähler erhöht. Anschließend werden zeitkritische Zustände darauf geprüft, ob ihre jeweilige Gültigkeitsdauer überschritten wurde. Für Frames, die sich in der indirekten Warteschlange befinden, gilt pro Frame eine Zeitspanne in der diese gültig sind. Wenn diese Zeitspanne abgelaufen ist werden die Frames verworfen und dies per MCPS-DATA.confirm an das höhere Layer gemeldet.

Während der Assoziierung und der Anfrage eines Frames ist der Transceiver für das Empfangen der Antwort des Koordinators eingeschaltet. Damit dieser sich bei nicht Empfangen einer Antwort wieder deaktiviert. Wird auch hierfür der Ablauf einer Zeitspanne überprüft. Wenn diese abgelaufen ist, wird der Transceiver wieder deaktiviert.

6.6 Callback-System

Die Interaktion mit dem SubMAC sowie mit darüberliegenden Schichten erfolgt über ein Callback-basiertes Ereignissystem, das in `ieee802154_mac_cbs_t` definiert ist.

Dieses umfasst unter anderem:

- Bestätigungen von Sendeoperationen (MCPS-DATA.confirm),
- Indikationen empfangener Frames (MCPS-DATA.indication),
- sowie MLME-bezogene Ereignisse.

Darüber hinaus werden hardware- und zeitbezogene Ereignisse, wie Radio-Interrupts oder Timer-Abläufe, über dedizierte Callback-Schnittstellen in den MAC-Layer integriert. Zeitkritische oder interruptbasierte Ereignisse werden nicht direkt verarbeitet, sondern über sogenannte Bottom-Half-Mechanismen in den Thread-Kontext überführt. Hierzu dienen Funktionen wie `ieee802154_mac_bh_process` oder `ieee802154_mac_handle_radio`.

Dieses Design vermeidet lange Verarbeitungszeiten im Interrupt-Kontext und erhöht die Systemstabilität.

6.7 Implementierung der Zustandsmaschine

Die detaillierte Beschreibung der Zustände und Zustandsübergänge ist im Kapitel 5 dargestellt.

Die Zustandsmaschine wurde als ereignisgesteuerte Steuerlogik realisiert. Für jeden Zustand existiert eine separate Funktion, die abhängig vom eintreffenden Ereignis entsprechende Aktionen ausführt und den nächsten Zustand bestimmt. Die Verarbeitung erfolgt über eine zentrale Dispatchfunktion, welche basierend auf dem aktuellen Zustand die jeweilige Zustandsroutine aufruft. Dieses Vorgehen ermöglicht eine klare Trennung der zustandsspezifischen Logik und verbessert die Wartbarkeit der Implementierung.

Innerhalb der Zustandsmaschine werden zentrale Aufgaben wie das Einreihen von Frames in die Sendewarteschlange, das Auslösen von Übertragungen über das SubMAC sowie die Weiterleitung empfangener Daten an höhere Schichten koordiniert.

Zur Gewährleistung threadsicherer Ausführung erfolgt die Verarbeitung von Zustandsübergängen unter Verwendung von Mutexen. Zusätzlich kann der vorherige Zustand gespeichert werden, um beispielsweise bei Übergängen zwischen Schlaf- und Aktivzuständen in den korrekten Modus zurück zu wechseln.

6.8 Interaktion mit dem SubMAC

Der entwickelte MAC-Layer baut auf dem in *RIOT* vorhandenen SubMAC-Layer auf, welches die direkte Interaktion mit der Radio-HAL kapselt. Während der SubMAC grundlegende Funktionen wie das Senden und Empfangen von Frames sowie die Steuerung des Transceivers bereitstellt, übernimmt das MAC-Layer die Protokolllogik und Koordination der Abläufe.

Beim Senden von Frames erfolgt die Interaktion in mehreren Schritten. Zunächst werden die zu übertragenden Daten durch das MAC-Layer in einen entsprechenden Frame eingebettet und in der Sendewarteschlange abgelegt. Anschließend wird der eigentliche Sendevorgang über den SubMAC durch Aufruf der entsprechenden Sendeoperation ausgelöst.

Empfangene Frames werden zunächst vom SubMAC verarbeitet und anschließend über Callback-Mechanismen an das MAC-Layer weitergeleitet. Die weitere Verarbeitung erfolgt innerhalb der Zustandsmaschine, welche abhängig vom Frametyp entsprechende Aktionen ausführt, wie beispielsweise die Weitergabe von Nutzdaten an höhere Schichten oder die Verarbeitung von MAC-Kommandos.

Die Kommunikation zwischen MAC und SubMAC erfolgt ereignisbasiert über ein Callback-System. Zeitkritische oder interruptbasierte Ereignisse werden dabei zunächst durch den SubMAC erfasst und anschließend über sogenannte Bottom-Half-Mechanismen in den Threadkontext überführt, wo sie durch das MAC-Layer verarbeitet werden.

6.9 Ablauf einer indirekten Datenübertragung

Der Schwerpunkt der Implementierung liegt auf der Unterstützung indirekter Datenübertragungen. Diese werden insbesondere im Koordinatorbetrieb benötigt, wenn Frames nicht unmittelbar an ein Endgerät übertragen werden können, sondern zunächst zwischengespeichert werden müssen, bis das jeweilige Gerät diese durch einen Pollmechanismus anfordert.

Wird durch eine höhere Schicht eine *MCPS-DATA.request* mit indirekter Übertragung ausgelöst, erzeugt der MAC-Layer zunächst einen vollständiges Frame und legt dieses in der Warteschlange ab. Die Zuordnung erfolgt dabei zielknotenspezifisch, sodass ausstehende Frames einem Endgerät eindeutig zugeordnet werden können. Im Unterschied zu direkten Übertragungen wird der Frame in diesem Fall nicht unmittelbar an den SubMAC zum Senden übergeben.

Der zwischengespeicherte Frame verbleibt in der indirekten Warteschlange, bis das zugehörige Endgerät einen *Data-Request* an den Koordinator sendet. Dieser Request wird zunächst durch den SubMAC empfangen und anschließend über die Callback- und Bottom-Half-Mechanismen an den MAC-Layer weitergeleitet. Die weitere Verarbeitung erfolgt in der Zustandsmaschine des MAC-Layers.

Nach Empfang eines gültigen *Data-Requests* wird geprüft, ob für das anfragende Endgerät ein passender Frame in der indirekten Warteschlange vorhanden ist. Ist dies der Fall, wird der entsprechende Warteschlangeneintrag ausgewählt und der eigentliche Sendevorgang ausgelöst. Die Übertragung erfolgt anschließend über den SubMAC-Layer.

Auf diese Weise wird sichergestellt, dass Daten erst dann gesendet werden, wenn das Endgerät empfangsbereit ist.

Kann kein passender Frame gefunden werden, erfolgt keine Datenübertragung. Zusätzlich werden zwischengespeicherte Frames über die periodische Tick-Funktion auf Ablauf ihrer Gültigkeitsdauer geprüft. Wird das konfigurierte Timeout überschritten, wird der betreffende Eintrag aus der Warteschlange entfernt und die höhere Schicht über eine entsprechende *MCPS-DATA.confirm* mit einem Fehlerstatus informiert.

Der Ablauf einer indirekten Übertragung wird im Sequenzdiagramm in Abb. 6.2 genau skizziert.

6.10 Behandlung von Fehlerfällen

Neben der regulären Verarbeitung berücksichtigt die Implementierung verschiedene Fehler- und Sonderfälle, die während des Betriebs auftreten können.

Ein wesentlicher Aspekt ist der Umgang mit begrenzten Ressourcen. Da ausgehende Frames in Warteschlangen organisiert werden, kann es bei hoher Last dazu kommen, dass keine freien Einträge mehr verfügbar sind. In diesem Fall werden entsprechende Fehlercodes zurückgegeben, sodass höhere Schichten auf diese Situation reagieren können.

Weiterhin werden zeitabhängige Fehlerzustände berücksichtigt. Hierzu zählen insbesondere Timeoutsituationen, wie das Ausbleiben von Bestätigungen oder Antworten während der Assoziation und bei Polloperationen. Diese werden durch die periodische Tickfunktion erkannt und führen zur entsprechenden Behandlung, beispielsweise dem Abbruch laufender Operationen oder der Rückmeldung an höhere Schichten.

Auch innerhalb der Zustandsmaschine werden ungültige Ereignisse erkannt. Tritt ein Ereignis in einem Zustand auf, für den keine definierte Behandlung existiert, wird dies als Fehlerzustand interpretiert und entsprechend behandelt, um inkonsistente Zustände zu vermeiden.

Darüber hinaus werden hardware- und treiberspezifische Fehler berücksichtigt. Fehlschläge bei der Interaktion mit dem Transceiver, beispielsweise beim Senden von Frames, werden erkannt und führen zu einer entsprechenden Fehlerbehandlung innerhalb des MAC-Layers.

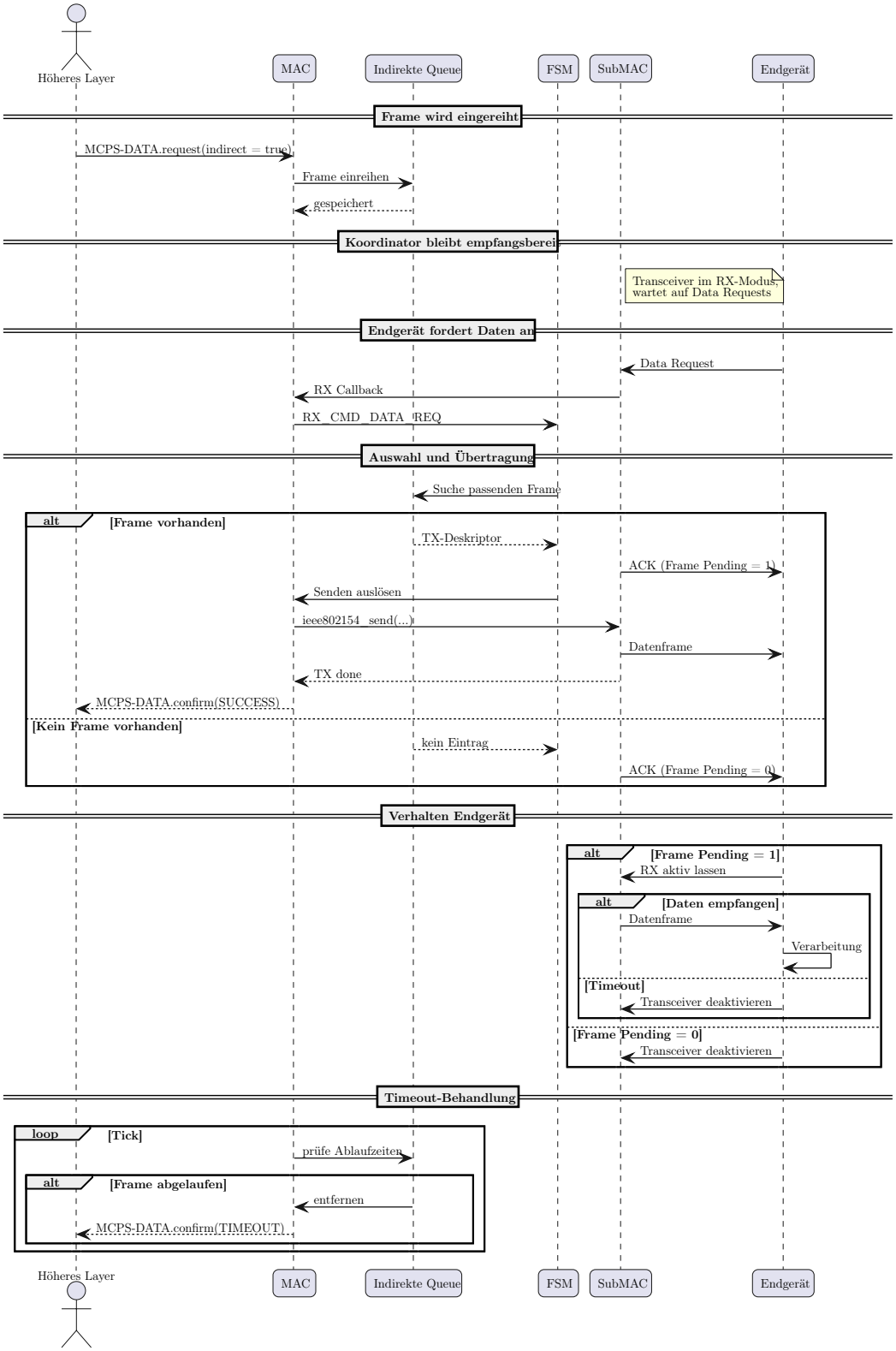

Abb. 6.2: Sequenzdiagramm indirekte Datenübertragung

Insgesamt stellt die Implementierung sicher, dass sowohl interne als auch externe Fehlerzustände erkannt und kontrolliert behandelt werden. Hierdurch wird die Robustheit des Systems erhöht.

6.11 Konfigurationsparameter

Der MAC-Layer erlaubt die Anpassung zentraler Systemparameter über Compiletimekonstanten. Diese Parameter beeinflussen sowohl das Laufzeitverhalten als auch den Ressourcenverbrauch des Systems und ermöglichen eine Anpassung an unterschiedliche Hardwareplattformen und Einsatzszenarien.

Ein wesentlicher Parameter ist die Größe der Sendewarteschlange (`IEEE802154_MAC_TXQ_LEN`), welche die maximale Anzahl gleichzeitig gepufferter Frames bestimmt. Eine größere Warteschlange erhöht die Robustheit gegenüber kurzfristigen Lastspitzen, führt jedoch zu einem höheren Speicherverbrauch.

Für indirekte Übertragungen wird zusätzlich die Anzahl verwalteter Warteschlangen über `IEEE802154_MAC_TX_INDIRECTQ_SIZE` konfiguriert. Dieser Parameter bestimmt, wie viele Endgeräte gleichzeitig mit ausstehenden Daten bedient werden können.

Zeitabhängige Abläufe werden über mehrere Parameter gesteuert. Das Tick-Intervall (`IEEE802154_MAC_TICK_INTERVAL_MS`) definiert die zeitliche Auflösung der internen Ablaufüberwachung. Ein kleineres Intervall ermöglicht eine feinere Reaktion auf Timeout-Ereignisse, erhöht jedoch die CPU-Last.

Zusätzlich wird über `IEEE802154_MAC_FRAME_TIMEOUT` festgelegt, wie lange Frames in der Warteschlange gültig bleiben, bevor sie verworfen werden.

Durch diese Konfigurationsmöglichkeiten kann die Implementierung flexibel an unterschiedliche Anforderungen angepasst werden, beispielsweise hinsichtlich Speicherverbrauch, Reaktionszeit oder Netzwerkstruktur.

6.12 GNRC Netif für das MAC Layer

Für die Nutzung der neuen MAC-Implementierung innerhalb des RIOT-Netzwerkstacks wurde zusätzlich eine Anbindung an GNRC Netif umgesetzt. Hierzu wurde das Modul

gnrc_netif_ieee802154_mac eingeführt. Es stellt gegenüber GNRC weiterhin eine reguläre gnrc_netif_t-Schnittstelle bereit, kapselt intern jedoch eine Instanz von ieee802154_mac_t und verbindet diese mit der Radio-HAL. Die Adapterstruktur enthält dafür sowohl eine netdev_t-Komponente als auch die Zustände und Puffer, die für Empfang, Senden, Scan, Assoziierung und periodisches Polling benötigt werden.

Die gnrc_netif_ops_t-Operationen leiten ausgehende Pakete an die MCPS-DATA.request-Schnittstelle des MAC-Layers weiter. Empfangene Frames werden über die Callbacks der MAC-Implementierung in eine Empfangswarteschlange des Adapters eingetragen und anschließend durch GNRC abgeholt. Verwaltungsoperationen wie Scan, Starten eines Koordinators, Beitritt zu einem Netzwerk und Polling werden über netopt-Aufrufe auf die entsprechenden MLME-Primitiven abgebildet. Dadurch kann der bestehende GNRC-Pfad weiterverwendet werden, während die IEEE 802.15.4-spezifische Protokolllogik im neuen MAC-Layer liegt. Das Modul wird über gnrc_netif_ieee802154_mac aktiviert und zieht dabei die Module ieee802154_mac, ieee802154_submac und die benötigten Radio-Schnittstellen ein.

Für diese Anbindung wurden mehrere netopt-Optionen verwendet beziehungsweise ergänzt. Bestehende Optionen wie NETOPT_ADDRESS, NETOPT_ADDRESS_LONG, NETOPT_NID und NETOPT_CHANNEL werden auf die entsprechenden Attribute der PIB abgebildet. NETOPT_SCAN startet einen aktiven Scan und liefert pro gefundenem Koordinator Kanal, PAN-ID, Adresse, LQI, RSSI und den Beacon-Payload zurück. NETOPT_CONNECT löst die Assoziierung mit einem Koordinator aus und nutzt intern MLME-ASSOCIATE.request; während eines laufenden Connect-Vorgangs wird automatisch gepollt, um die Assoziierungsantwort abzuholen. NETOPT_START startet einen Koordinator auf einem angegebenen Kanal. NETOPT_POLL sendet gezielt eine MLME-POLL.request an einen Koordinator, während NETOPT_POLL_INTERVAL das periodische Polling für indirekte Übertragungen konfiguriert. Zusätzlich werden NETOPT_BEACON_PAYLOAD, NETOPT_PAN_COORD, NETOPT_ACK_REQ und NETOPT_TX_INDIRECT unterstützt. Letzteres wird beim Senden aus dem GNRC-Paketheader übernommen und entscheidet, ob ein ausgehendes Frame direkt oder indirekt über die Warteschlange des Koordinators verschickt wird.

Die neuen Funktionen sind auch über die RIOT-Shell zugänglich. Der Befehl ifconfig <if> scan [all|<channel>] startet einen aktiven Scan und gibt die gefundenen Koordinatoren tabellarisch aus. Mit ifconfig <if> start <channel> wird ein Knoten als Koordinator gestartet. Der Beitritt eines Endgeräts erfolgt mit ifconfig

`<if> connect <channel> <panid> <coord_addr> [capability]`, wobei die Koordinatoradresse als kurze oder lange IEEE 802.15.4-Adresse angegeben werden kann. Für manuelle Datenanfragen steht `ifconfig <if> poll <panid> <coord_addr>` zur Verfügung. Zusätzlich können `ifconfig <if> set poll_interval <ms>` und `ifconfig <if> set beacon_payload <payload>` verwendet werden. Für Test- und Debugzwecke wurde außerdem der bestehende Textsende-Befehl erweitert, sodass `txtsnd -indirect <if> <addr> <data>` ein Paket mit gesetztem Indirect-Flag an die neue MAC-Implementierung übergibt.

6.13 Anpassungen an bestehendem Code

Im Rahmen der Implementierung wurden mehrere Anpassungen an bestehenden Komponenten vorgenommen, um eine korrekte Funktionalität des MAC-Layers sicherzustellen.

6.13.1 Framefilterung im Koordinatorbetrieb

Für den Betrieb des MAC-Layers als Koordinator ist es erforderlich, auch Frames ohne gesetzte Zieladresse zu empfangen, insbesondere bei MAC-Kommandos wie dem *Association Request*. Diese Funktionalität war in den vorhandenen Implementierungen der Radio-HAL für die unterstützten Transceiver nicht vollständig umgesetzt. Für die Transceiverfamilien KW2XRF [1] und AT86RF2XX [2] wurde dieses Verhalten durch die Konfiguration eines entsprechenden Hardware-Registers realisiert, welches den Transceiver in den Koordinatorbetrieb versetzt und die Filterung entsprechender Frames ermöglicht. Für Transceiver der NRF52-Familie [3], die keine vergleichbaren Hardwaremechanismen bereitstellen, erfolgt die Framefilterung vollständig in Software innerhalb der Radio-HAL. In diesem Zusammenhang wurde die bestehende Filterlogik erweitert, sodass auch Frames ohne gesetzte Zieladresse akzeptiert werden, sofern sich das Gerät im Koordinatorbetrieb befindet. Hierzu wurde eine zusätzliche Zustandsvariable eingeführt, die den aktuellen Betriebsmodus des Transceivers berücksichtigt.

[1] https://www.nxp.com/docs/en/fact-sheet/KNTSKW2XFS.pdf
[2] https://www.microchip.com/en-us/product/at86rf233
[3] https://docs.nordicsemi.com/r/bundle/additionalresources/page/additionalresources/nrf52-series

6.13.2 Frame Pending und Source Address Matching

Ein weiteres Problem trat bei der Behandlung indirekter Übertragungen auf. Hierbei ist das *Frame Pending* Bit relevant, welches signalisiert, ob weitere Daten für ein Endgerät bereitstehen. Dieses Bit wird in der bestehenden Implementierung ausschließlich von Transceivern gesetzt, die entsprechende Hardwareunterstützung bieten. Da dies beispielsweise bei der NRF52-Familie nicht der Fall ist, wurde die Signalisierung angepasst. Die Information über gesetzte Pending-Frames wird nun im Callback aus dem Statusparameter in die übergebene Informationsstruktur integriert. Zusätzlich wurde die Entscheidung über das Setzen des *Frame Pending*-Bits in das SubMAC-Layer verlagert, sodass diese Funktionalität auch für Transceiver ohne Hardwareunterstützung einheitlich verfügbar ist. Diese Verlagerung baut auf der Erweiterung des SubMAC um softwareseitige ACK-Erzeugung auf.[4].

Für die Kopplung der indirekten Warteschlange an das *Frame Pending* Bit wurde außerdem die Source Address Matching Schnittstelle der Radio-HAL erweitert. Die Aktualisierung dieses Zustands erfolgt an den Stellen, an denen ein indirekter Warteschlangeneintrag angelegt oder entfernt wird. Dazu wurde die Hilfsfunktion `ieee802154_mac_indirect_fp_update` eingeführt, welche den Pending-Zustand für die betroffene Zieladresse aktualisiert. Unterstützt die Plattform adressgenaues Source Address Matching, werden kurze beziehungsweise erweiterte Adressen an die SubMAC übergeben. Dadurch kann ein Transceiver mit entsprechender Hardwareunterstützung das Pending-Bit nur für Endgeräte setzen, für die tatsächlich indirekte Daten vorliegen. Für Plattformen ohne adressgenaue Unterstützung wird stattdessen ein globaler Zustand verwendet. Der SubMAC leitet diese Konfiguration entweder an die Radio-HAL weiter oder speichert sie für die softwareseitige ACK-Erzeugung. Damit bleibt die Entscheidung über das Pending-Bit zentral im SubMAC gekapselt, während die MAC-Schicht nur die Zustandsänderungen der indirekten Warteschlange melden muss.

6.13.3 Short-Adressen im KW2XRF-Treiber

Während der Implementierung und Nutzung des KW2XRF-Transceivers wurde zudem ein Fehler im Zusammenhang mit der Verwendung von Short-Adressen festgestellt. Die

[4] vgl. Pull Request:
https://github.com/RIOT-OS/RIOT/pull/21973

Ursache lag in einer fehlerhaften Endianessumwandlung innerhalb der Radio-HAL. Dieser Fehler wurde identifiziert und behoben[5].

6.14 Tests

Um die Funktionalität des MAC-Layers sicherzustellen wurden einige Unit- und Systemtests implementiert. Die Unittests umfassen die Warteschlangen für die einzelnen Endgeräte und die Warteschlange die diese enthalten. In den Systemtests werden die einzelnen Primitiven des Upper-Layers getestet und die Post-Conditions abgefragt.

6.14.1 Unittests

Die Unit-Tests des MAC-Layers befinden sich in `tests/unittests/tests-ieee802154_mac` und werden mit dem in RIOT verwendeten EmbUnit-Framework ausgeführt. Sie testen die Warteschlangenlogik isoliert von realer Funkhardware.

Ein erster Teil der Tests behandelt die Sendewarteschlange `ieee802154_mac_txq_t`. Geprüft werden die Zustände leer und voll, das Reservieren neuer Deskriptoren, das Bestätigen eines reservierten Eintrags über `commit`, das Lesen des vordersten Elements mit `peek` sowie das Entfernen mit `pop`. Dabei werden auch Randfälle abgedeckt, etwa `NULL`-Argumente, das Reservieren in einer vollen Warteschlange und das Entfernen aus einer leeren Warteschlange. Zusätzlich prüfen die Tests das korrekte Umlaufen der Ringpuffer-Indizes für Kopf und Ende der Warteschlange. Damit wird sichergestellt, dass die Anzahl der belegten Einträge und die internen Indizes auch an den Grenzen der Warteschlangenlänge konsistent bleiben.

Der zweite Teil testet die Warteschlangenverwaltung für indirekte Übertragungen, die durch `ieee802154_mac_indirect_q_t` repräsentiert wird. Hier wird geprüft, dass ein freier Slot korrekt reserviert wird, dass bei vollständig belegter Slotmaske kein weiterer Slot vergeben wird und dass die Reservierung das zugehörige Bit in der Maske löscht. Ebenso wird getestet, dass beim Freigeben eines Slots die Freimaske wieder gesetzt und der Inhalt der zugehörigen Zielwarteschlange zurückgesetzt wird. Abschließend prüfen die Tests, ob die indirekte Warteschlangenstruktur korrekt zwischen vollständig leerem Zustand und mindestens einem belegten Slot unterscheidet.

[5] vgl. Pull Request:
https://github.com/RIOT-OS/RIOT/pull/22057

6.14.2 Systemtests

Die Systemtests wurden mit der pythonbasierten Testumgebung Bream umgesetzt. Bream stellt dabei die Verbindung zu den Testknoten her, startet die Firmware, sendet Shell Befehle an einzelne RIOT-Knoten und wertet die seriellen Ausgaben aus. Die Tests werden als Pytests ausgeführt und können gegen serielle Knoten laufen. Im Gegensatz zu den Unit-Tests wird dabei nicht nur eine einzelne Datenstruktur geprüft, sondern das Zusammenspiel aus MAC-Layer, SubMAC, Radio-HAL, Shell Befehle und realer Funkübertragung.

Ein erster Teil der Systemtests prüft das Auffinden und Beitreten zu einem Netzwerk. Dazu wird zunächst ein Knoten als Koordinator gestartet. Der Koordinator wird mit Kanal, PAN-ID und SSID konfiguriert und anschließend über `ifconfig` kontrolliert. Weitere Knoten führen danach einen Scan beziehungsweise eine Suche nach verfügbaren IEEE-802.15.4-Netzwerken aus. Die Testerwartung ist, dass die vom Koordinator ausgesendete PAN-ID in den Suchergebnissen erscheint. Zusätzlich werden Szenarien mit mehreren Koordinatoren getestet. Dabei laufen zwei Koordinatoren entweder auf demselben Kanal mit unterschiedlichen PAN-IDs oder auf unterschiedlichen Kanälen. Die scannenden Knoten müssen die erwarteten Netze erkennen, wodurch insbesondere Beacon Erzeugung, Kanalwahl, PAN-ID Filterung und die Auswertung der Suchergebnisse geprüft werden.

Darauf aufbauend wird der Beitritt eines einzelnen Knotens zum Netzwerk getestet. Der Test startet einen Koordinator, liest dessen Adresse aus und führt auf einem zweiten Knoten eine Join-Operation aus. Nach dem Join wird der Zustand des Knotens erneut über `ifconfig` geprüft. Entscheidend ist hierbei, dass der Knoten eine gültige Kurzadresse erhält und damit als assoziiertes Endgerät im Netzwerk betrieben werden kann. Dieser Test deckt die MLME-ASSOCIATE Abläufe, die Adressvergabe und die grundlegende Zustandsänderung nach einer erfolgreichen Assoziation ab.

Für indirekte Datenübertragungen wird ein Koordinator mit einem assoziierten Endgerät aufgebaut. Anschließend sendet der Koordinator mit `txtsnd -indirect` eine Nutzlast an das Endgerät. Der Test wartet auf die Ausgabe des Paket-Dumps am Endgerät und prüft, ob die empfangene Nutzlast exakt dem gesendeten Wert entspricht. Danach wird in Gegenrichtung eine direkte Nachricht vom Endgerät zum Koordinator gesendet und ebenfalls anhand der empfangenen Nutzlast validiert. Dadurch wird geprüft, dass indi-

rekte Frames korrekt in die Warteschlange des Koordinators eingetragen, durch Polling des Endgeräts ausgeliefert und anschließend aus der Warteschlange entfernt werden.

Ein weiterer Systemtest betrachtet zwei unabhängige Koordinator-Endgeräte Paare auf demselben Kanal. Beide Koordinatoren verwenden unterschiedliche PAN-IDs und je ein Endgerät tritt dem jeweiligen Netzwerk bei. Danach werden Nachrichten parallel an beide Endgeräte gesendet. Die Tests prüfen nicht nur, dass die erwartete Nutzlast beim richtigen Endgerät ankommt, sondern auch, dass keine Nachricht versehentlich im falschen Netzwerk zugestellt wird. Damit wird die Trennung der PANs auf demselben Funkkanal überprüft.

7 Evaluation

7.1 Versuchsaufbau

Die Messungen werden mit einem nrf52840 SoC durchgeführt.

Der Aufbau besteht aus zwei Boards. Das Erste davon wird als Koordinator eingerichtet, das Zweite als Endgerät, welches mit diesem assoziiert wird.

Der Nachrichtenablauf funktioniert so, dass der Koordinator dabei ein einzelnes Nutzdatenframe in die indirekte Sendewarteschlange einreiht. Das assoziierte Endgerät ruft dieses Frame durch Polling ab und sendet anschließend eine Antwort an den Koordinator. Der nächste Request wird erst gestartet, nachdem die Antwort des vorherigen Requests beobachtet wurde oder ein Timeout abgelaufen ist.

Das Gerät kann in einen Duty-Cycle mit einer definierten Inaktivitätszeit versetzt werden. Im Schlafmodus wird das Radio abgeschaltet und der Thread blockiert mit einem sleep für die eingestellte Inaktivitätszeit. Der Idle Thread von RIOT OS schaltet dann die MCU in einen Schlafmodusm, der nur durch Interrupts verlassen werden kann. Der Interrupt des sleeps weckt die MCU dann wieder auf. Sobald das Gerät aus dem Schlafmodus aufwacht, sendet es mithilfe der MLME-POLL Primitiven einen *Data Request* an den Koordinator. Anschließend wird bis zu 10 ms auf ein eintreffendes Frame gewartet. Trifft dieses ein, wird mit einem Timeout von 10 ms auf den Abschluss der an den Koordinator gesendeten Antwort abgewartet und erst danach die MCU und den Transceiver in den Schlafmodus zurückversetzt. Trifft kein Frame ein, werden der Transceiver und die MCU direkt in den Schlafmodus zurückversetzt.

Für jeden untersuchten Duty-Cycle werden zehn unabhängige Läufe durchgeführt. Innerhalb eines Laufs werden jeweils 30 Requests ausgeführt. Beantwortete Requests liefern einen RTT-Messwert. Dieses Vorgehen reduziert die zeitliche Kopplung der Einzelmessungen innerhalb eines Laufs, um eine Pseudoreplikation zu vermeiden.

7.1.1 Exponentialverteilter Anfragejitter

Der nächste Request wird erst gestartet, nachdem der vorherige Request beantwortet wurde oder in einen Timeout gelaufen ist. Zwischen zwei aufeinanderfolgenden Requests wird eine zufällige Wartezeit eingefügt. Diese Wartezeit dient als Jitter, um unterschiedliche Phasenlagen zwischen Anfragezeitpunkt und Duty-Cycles des Endgeräts abzudecken.

Der Jitter J wird als exponentialverteilte Wartezeit modelliert [13]. Für den Parameter $\lambda > 0$ gilt

$$f_J(t) = \lambda e^{-\lambda t}, \qquad t \geq 0,$$

wobei $f_J(t)$ die Dichtefunktion der Wartezeit ist. Die zugehörige Verteilungsfunktion lautet

$$F(t) = P(J \leq t) = 1 - e^{-\lambda t}. \qquad t \geq 0.$$

Der Erwartungswert der Wartezeit beträgt

$$\mathbb{E}[J] = \frac{1}{\lambda}.$$

Kurze Abstände treten dadurch häufig auf, während längere Abstände mit abnehmender Wahrscheinlichkeit möglich bleiben. Dadurch werden unterschiedliche relative Phasenlagen zwischen Anfragezeitpunkt und Duty-Cycle über einen Messlauf zufällig abgedeckt.

Für die Parametrisierung wird die mittlere Jitterzeit relativ zum betrachteten Duty-Cycle D gewählt. Konkret wird

$$\mathbb{E}[J] = D$$

gesetzt. Somit entspricht die mittlere Jitterzeit einem Duty-Cycle. Für den Fall ohne Duty-Cycling ($D = 0$) wird die Messung mit Jitterwerten $\mathbb{E}[J] = 100\,\text{ms}$ durchgeführt.

$$D_{\text{eff}} = \begin{cases} D, & D > 0, \\ 100\,\text{ms}, & D = 0. \end{cases}$$

Zur konkreten Erzeugung eines Jitterwerts wird eine gleichverteilte Zufallsvariable $U \sim \mathcal{U}(0,1)$ verwendet. Der Jitter ergibt sich durch inverse Transformation als

$$J = -D_{\text{eff}} \ln(1 - U), \qquad U \in \mathcal{U}(0,1).$$

7.2 Einzelergebnisse zur Latenz

Die Zeitmessung beginnt beim Einreihen des indirekten Frames in die Sendewarteschlange und endet, wenn die zugehörige Antwort des Endgeräts beobachtet wird.

In Abb. 7.1 ist der Vergleich des Medians und des Mittelwerts dargestellt. Dort ist zu sehen, dass Median und Mittelwert denselben grundsätzlichen Anstieg mit dem Duty-Cycle zeigen. In der Abbildung wurde zur Darstellung der Streuung der Messwerte das 5 %-Quantil und 95 %-Quantil bestimmt, das Intervall zwischen diesen zeigt die von Ausreißern bereinigte Streuung grau eingefärbt.

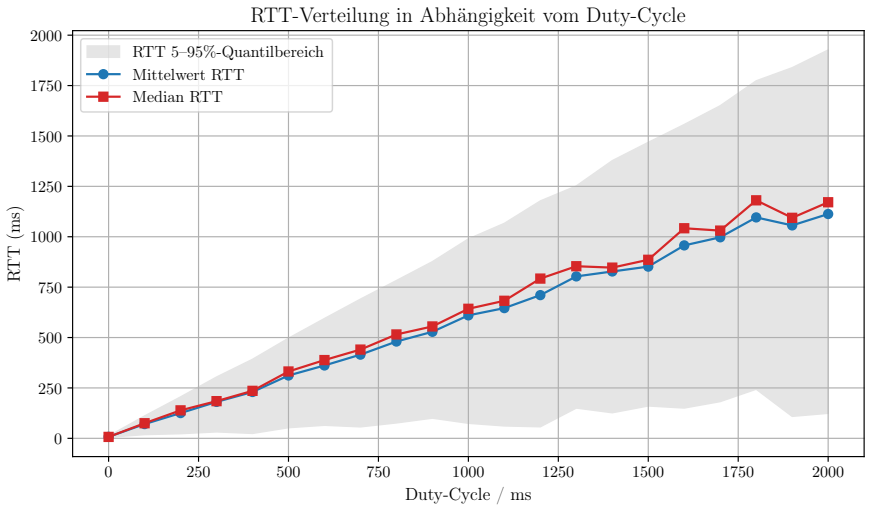

Abb. 7.1: Vergleich von Mittelwert und Median sowie 5 %- bis 95 %-Quantilbereich der RTT-Messwerte.

Für die Abschätzung der Unsicherheit der Messwerte werden bootstrap Konfidenzintervalle berechnet [11]. Als statistische Einheit dient dabei nicht ein einzelner Request, sondern ein vollständiger Messlauf. Für einen Duty-Cycle d mit $m = 10$ Messläufen und n_i Requests im Lauf i wird zunächst der Mittelwert pro Messlauf

$$\bar{r}_{d,i} = \frac{1}{n_i} \sum_{j=1}^{n_i} r_{d,i,j}$$

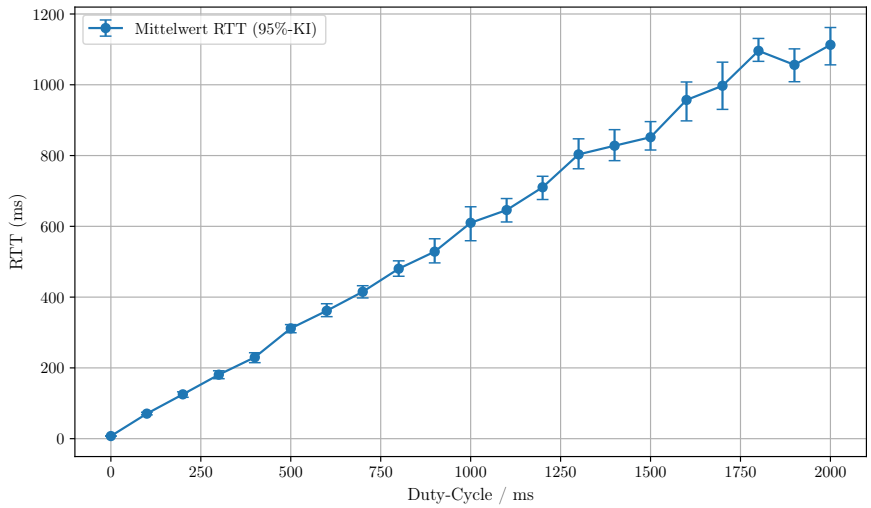

Abb. 7.2: Mittlere RTT mit 95 %-Konfidenzintervallen im serialisierten Aufbau.

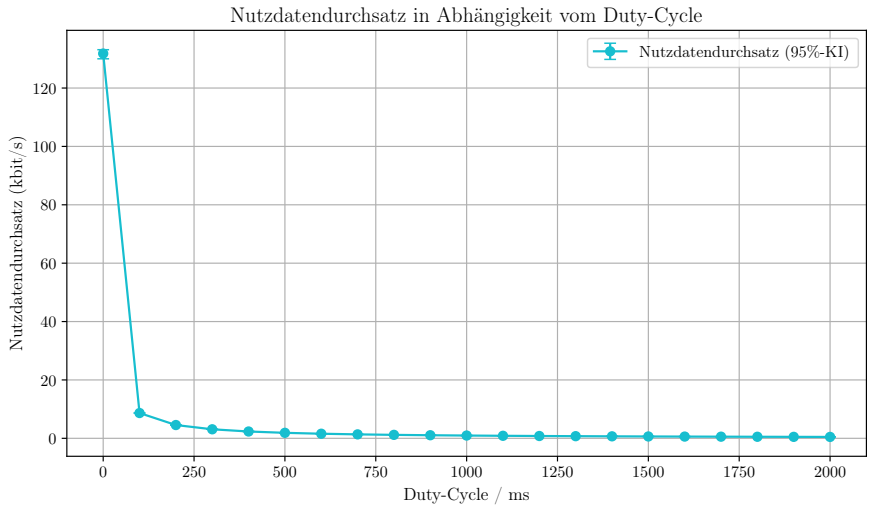

Abb. 7.3: Downlinkdatenrate mit 95 %-Konfidenzintervallen.

berechnet. Der dargestellte Schätzwert ergibt sich anschließend als Mittelwert über die m Messläufe:

$$\hat{\theta}_d = \frac{1}{m} \sum_{i=1}^{m} \bar{r}_{d,i}.$$

Für das Bootstrap-Verfahren wird die Menge $\{\bar{r}_{d,1}, \ldots, \bar{r}_{d,m}\}$ mit Zurücklegen neu gezogen. Für jede der $B = 10.000$ Bootstrap-Stichproben wird erneut ein Mittelwert $\hat{\theta}_d^{*(b)}$ berechnet. Das 95 %-Konfidenzintervall ergibt sich aus dem Perzentilintervall.

$$\left[q_{0{,}025} \left(\hat{\theta}_d^* \right),\ q_{0{,}975} \left(\hat{\theta}_d^* \right) \right].$$

Dieses Vorgehen verhindert, dass die innerhalb eines Messlaufs zeitlich gekoppelten Einzelmessungen als unabhängige Stichproben behandelt werden. Die Mittelwerte und deren Konfidenzintervalle sind in Abb. 7.2 dargestellt. Die aus der Empfangsdauer berechnete Downlinkdatenrate ist in Abb. 7.3 dargestellt. Die Messung der Downlinkdatenrate verwendet einen separaten Downlink-Aufbau. Als Downlink wird hier die Übertragung vom Koordinator zum Endgerät bezeichnet. Dabei sendet der Koordinator pro Messlauf eine feste Anzahl von 30 Frames mit einer Nutzlast von 116 Byte an das Endgerät. Für den Betrieb ohne Duty-Cycling werden die Frames direkt übertragen, für alle Duty-Cycles über die indirekte MAC-Übertragung. Die Downlinkdatenrate ergibt sich aus den am Endgerät empfangenen Nutzdatenbytes und der vom Empfänger gemessenen Empfangsdauer. Sie beschreibt daher nicht die maximale Rohdatenrate des Funkstandards, sondern die unter dem jeweiligen Duty-Cycle tatsächlich erzielte Datenrate in Richtung des Endgeräts.

Ohne Duty-Cycling erreicht der Aufbau etwa 132 kbit s^{-1}. Sobald das Endgerät periodisch schläft, sinkt die Downlinkdatenrate stark ab. Bereits bei einem Duty-Cycle von 100 ms liegt sie nur noch bei etwa 8,66 kbit s^{-1}. Bei 500 ms werden etwa 1,88 kbit s^{-1} erreicht, bei 1000 ms etwa 0,95 kbit s^{-1} und bei 2000 ms etwa 0,48 kbit s^{-1}.

Der starke Abfall zwischen dem Betrieb ohne Duty-Cycling und dem Betrieb mit Schlafintervallen zeigt, dass die Downlinkdatenrate bei indirekter Übertragung vor allem durch die Poll-Perioden des Endgeräts begrenzt wird. Mit zunehmendem Duty-Cycle verlängert sich die Zeit, bis die feste Anzahl an Downlink-Frames vom Endgerät abgeholt wurde, nahezu proportional. Die Downlinkdatenrate entwickelt sich damit gegensätzlich zur Latenz. Größere Duty-Cycles verschlechtern sowohl die Reaktionszeit als auch die erreichbare Downlinkdatenrate.

7.2.1 Einfluss des Duty-Cycles auf die Latenz

Der Duty-Cycle wirkt sich in den Messungen deutlich auf die MAC-Latenz aus. Ohne Duty-Cycling liegt die mittlere RTT bei etwa 7,4 ms. Dieser Wert beschreibt im Wesentlichen die reine Übertragungs- und Verarbeitungszeit des indirekten MAC-Frames und der zugehörigen Antwort. Sobald das Endgerät periodisch schläft, kommt zur eigentlichen Übertragungszeit die Wartezeit bis zum nächsten Poll des Endgeräts hinzu. Dadurch steigt die mittlere RTT bereits bei einem Duty-Cycle von 100 ms auf etwa 70,7 ms.

Mit zunehmendem Duty-Cycle wächst die mittlere RTT insgesamt deutlich an. Bei 500 ms beträgt sie etwa 311,7 ms, bei 1000 ms etwa 610,0 ms und bei 2000 ms etwa 1112,1 ms. Da der Koordinator die Anfrage an ein schlafendes Endgerät erst nach dessen nächsten Poll zustellen kann, ist die zusätzliche Wartezeit durch die relative Phasenlage zwischen Anfragezeitpunkt und Duty-Cycle bestimmt. Der exponentialverteilte Anfragejitter verteilt diese Phasenlage über die Messläufe, sodass sich im Mittel ein mit dem Duty-Cycle wachsender Anteil an Poll-Wartezeit ergibt.

Die Konfidenzintervalle in Abb. 7.2 werden bei größeren Duty-Cycles breiter. Dies ist plausibel, da mit längerer Schlafperiode auch der mögliche Bereich der Wartezeit bis zum nächsten Poll größer wird. Für die Bewertung der Reaktionszeit ist damit der Duty-Cycle der dominante Parameter. Kleine Duty-Cycles liefern kurze und eng streuende Antwortzeiten, während große Duty-Cycles die Latenz in den Sekundenbereich verschieben können.

7.3 Einzelergebnisse zum Energieverbrauch

7.3.1 Messmethodik

Der Messaufbau zur Erfassung des Versorgungsstroms ist in Abb. 7.4 dargestellt. Das Endgerät wird über das Netzteil versorgt, wobei das Multimeter (DMM7510) in Reihe der positiven Versorgungsleitung liegt. Der Computer steuert die Messung, liest die Stromwerte des Multimeters aus und kommuniziert über den UART-Adapter mit dem Endgerät. Der Adafruit nRF52840 Feather Express dient als Koordinator und kommuniziert über IEEE 802.15.4 mit dem nRF52840-DK.

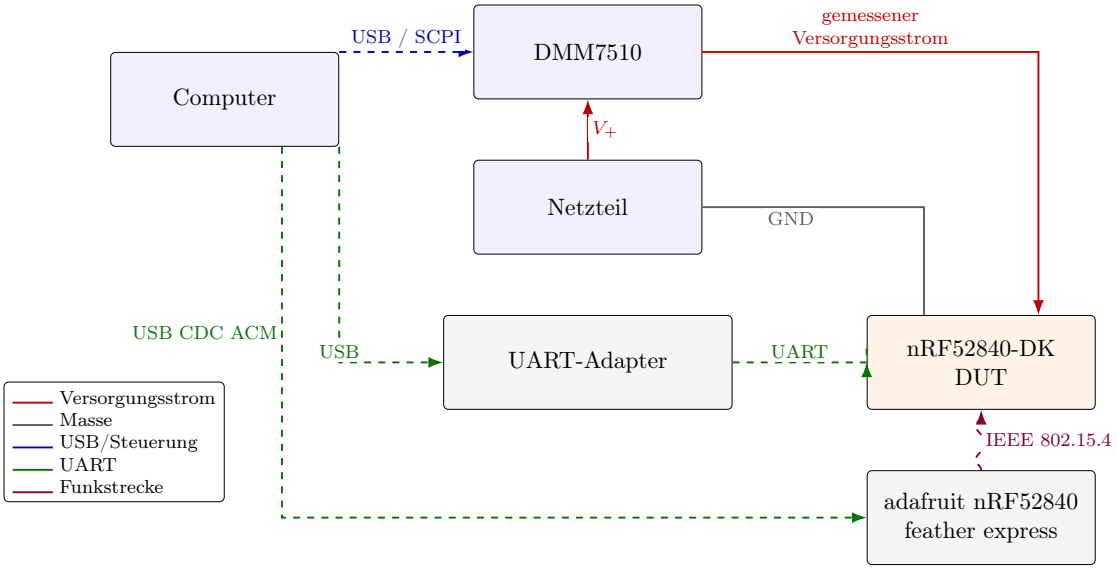

Abb. 7.4: Messaufbau zur Strommessung

Die Messungen wurden äquivalent zu den Latenzmessungen durchgeführt. Somit werden die gleiche Anzahl der Frames und Messläufe ausgeführt und über diese den Durchschnitt gebildet.

7.3.2 Ergebnisse des Energieverbrauchs

Zur Einordnung der Duty-Cycle-Messungen wurden zunächst mehrere Basiszustände des Endgeräts gemessen. Der dauerhaft aktive Zustand mit eingeschaltetem und empfangsbereitem Transceiver dient als Referenz für den Betrieb ohne Energiesparmechanismus. Der Schlafzustand mit deaktiviertem Transceiver und schlafender MCU beschreibt die untere Grenze der erreichbaren Stromaufnahme.

Im dauerhaft aktiven Zustand beträgt die mittlere Stromaufnahme 6,03 mA. Im Schlafzustand sinkt sie auf 0,45 mA. Das Abschalten des Transceivers und der Wechsel in den Schlafmodus reduziert die Stromaufnahme damit um 5,58 mA beziehungsweise 92,54 %.

Für den Pollbetrieb mit einem Intervall von 500 ms wurden zwei Fälle betrachtet. Polling ohne das Empfangen von Datenframes und Polling mit Empfangen von Datenframes vom Koordinator. Die mittlere Stromaufnahme liegt dabei bei 0,66 mA ohne Empfang beziehungsweise 0,65 mA mit Empfang. Der Unterschied zwischen beiden Fällen ist im Mittel sehr klein. Damit wird sichtbar, dass der Grundanteil des periodischen Aufwachens

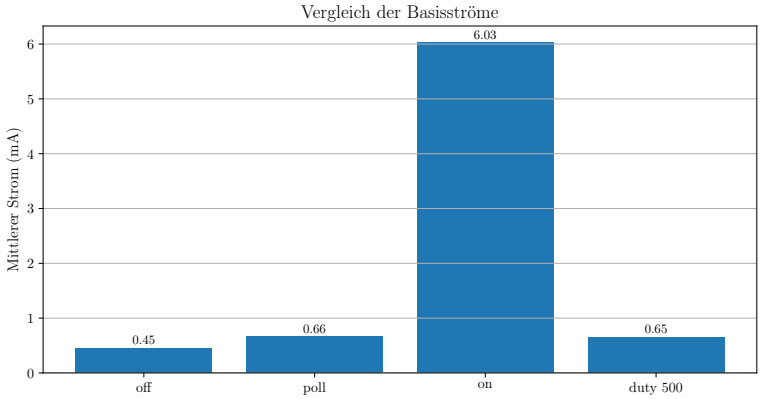

Abb. 7.5: Vergleich der mittleren Stromaufnahme verschiedener Zustände und einen repräsentativen Duty-Cycle von 500 ms.

und Pollings den Energiebedarf in diesem Betriebsbereich stärker prägt als der Empfang eines einzelnen Datenframes.

Die gemessenen Basiszustände sind in Abb. 7.5 dargestellt.

7.3.3 Einfluss des Duty-Cycles auf den Energieverbrauch

Der mittlere Strom sinkt mit zunehmendem Duty-Cycle, weil das Endgerät länger schlafen kann und seltener aktiv werden muss. Der stärkste Effekt tritt dabei im unteren Bereich der untersuchten Duty-Cycles auf. Bei 100 ms beträgt die mittlere Stromaufnahme etwa 1,29 mA. Bei 500 ms sinkt sie bereits auf etwa 0,65 mA. Eine weitere Vergrößerung des Duty-Cycles reduziert den Strom nur noch wenig. Bei 1000 ms werden etwa 0,56 mA erreicht, bei 2000 ms etwa 0,51 mA.

Dieser Verlauf ist plausibel, da der Energiebedarf aus einem nahezu konstanten Schlafanteil und kurzen aktiven Phasen zum Polling besteht. Mit größerem Duty-Cycle verteilt sich die Energie dieser aktiven Phasen auf ein längeres Intervall. Der mittlere Strom nähert sich dadurch dem Schlafstrom von 0,45 mA an. Der Grenznutzen zusätzlicher Schlafzeit nimmt deshalb mit zunehmendem Duty-Cycle ab.

Gegenüber dem dauerhaft aktiven Zustand mit 6,03 mA ergibt sich bereits bei 100 ms eine Einsparung von etwa 4,74 mA beziehungsweise 78,6 %. Bei 500 ms steigt die Einsparung auf etwa 5,38 mA beziehungsweise 89,2 %. Zwischen 1000 ms und 2000 ms nimmt die

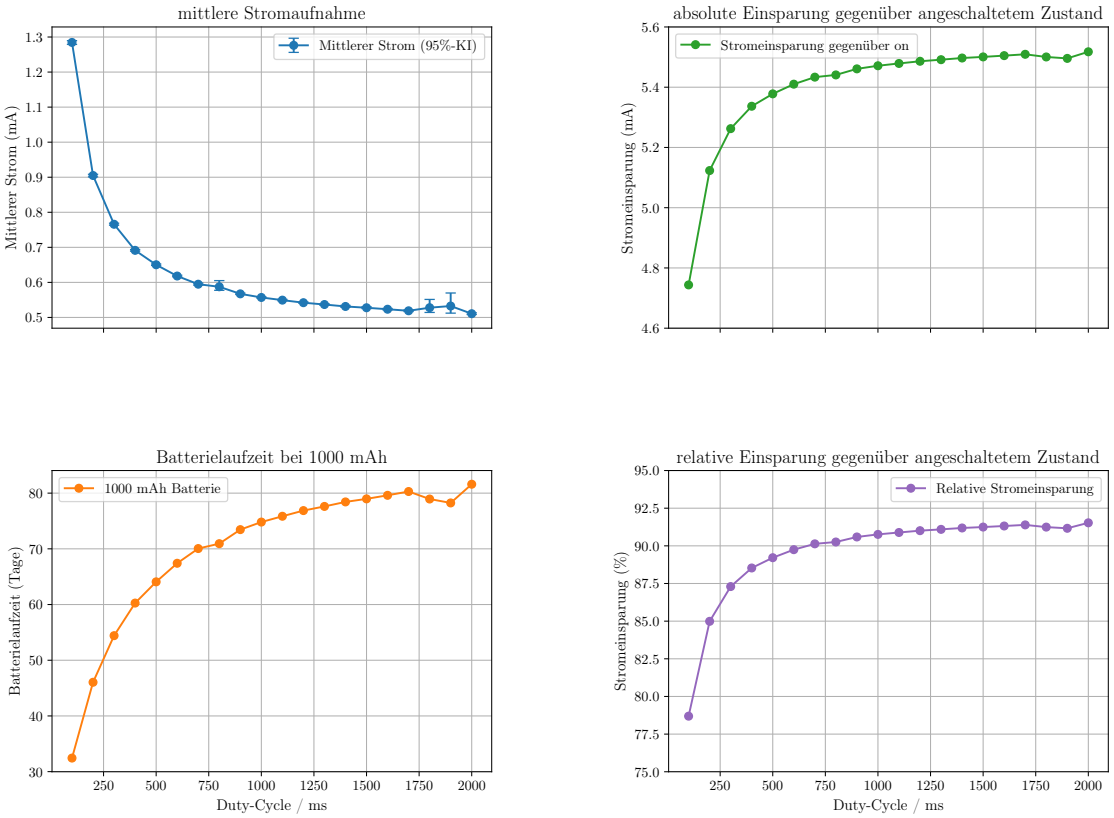

Abb. 7.6: mittlerer Strom, absolute Stromeinsparung, relative Stromeinsparung und geschätzte Batterielaufzeit in Abhängigkeit vom Duty-Cycle.

relative Einsparung nur noch von etwa 90,8 % auf etwa 91,5 % zu. Die zusätzliche Einsparung durch eine Verdopplung des Duty-Cycles in diesem Bereich ist damit deutlich kleiner als die Einsparung durch den Wechsel vom dauerhaft aktiven Betrieb zu kurzen Schlafintervallen.

Die geschätzte Batterielaufzeit zeigt denselben Zusammenhang in umgekehrter Richtung. Für eine angenommene Kapazität von 1000 mA h steigt sie von etwa 32 Tagen bei 100 ms auf etwa 64 Tage bei 500 ms. Bei 1000 ms liegt sie bei ungefähr 75 Tagen und bei 2000 ms bei etwa 82 Tagen. Der Duty-Cycle verbessert die Laufzeit somit deutlich. Die Zuwächse werden bei größeren Intervallen jedoch zunehmend kleiner.

Abb. 7.6 fasst den mittleren Strom, die absolute und relative Stromeinsparung sowie die daraus abgeleitete Batterielaufzeit zusammen.

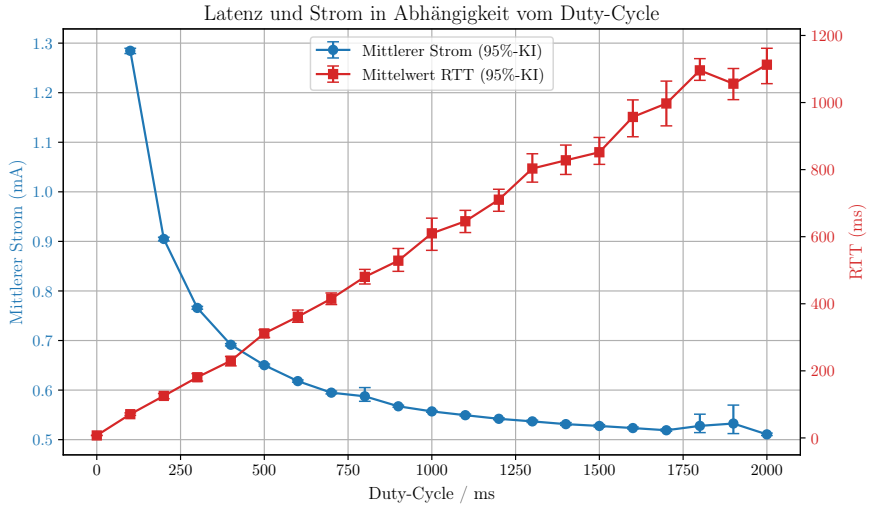

Abb. 7.7: Gemeinsame Darstellung von mittlerem Strom und Mittelwert der RTT im serialisierten Aufbau mit 95 %-Konfidenzintervallen.

7.4 Gemeinsame Bewertung von Latenz, Downlinkdatenrate und Energieverbrauch

Abb. 7.7 zeigt die gemeinsame Darstellung von Latenz und Stromaufnahme. Kleine Duty-Cycles verbessern die Reaktionszeit und erlauben eine höhere Downlinkdatenrate. Sie erhöhen aber die mittlere Stromaufnahme. Größere Duty-Cycles reduzieren den Energiebedarf, verschlechtern jedoch sowohl die Latenz als auch die erreichbare Downlinkdatenrate.

Die mittlere RTT liegt mit etwa 7,4 ms sehr niedrig und die Downlinkdatenrate erreicht etwa 132 kbit s^{-1}. Gleichzeitig ist die mittlere Stromaufnahme mit 6,03 mA deutlich höher als in allen Duty-Cycle-Konfigurationen. Dieser Betriebsmodus ist daher vor allem dann sinnvoll, wenn kurze Reaktionszeiten oder eine hohe Downlinkdatenrate wichtiger sind als eine lange Batterielaufzeit.

Bereits ein Duty-Cycle von 100 ms senkt die mittlere Stromaufnahme auf etwa 1,29 mA und spart damit gegenüber dem dauerhaft aktiven Zustand rund 78,6 % Strom ein. Die mittlere RTT steigt dabei auf etwa 70,7 ms, die Downlinkdatenrate sinkt auf etwa 8,66 kbit s^{-1}. Dieser Bereich stellt damit einen deutlichen Energiespareffekt bereit, ohne die Reaktionszeit bereits in den Bereich mehrerer hundert Millisekunden zu verschieben.

Bei 500 ms wird mit etwa 0,65 mA bereits ein Großteil der möglichen Stromeinsparung erreicht. Die mittlere RTT beträgt dann etwa 311,7 ms, die Downlinkdatenrate etwa 1,88 kbit s^{-1}. Für Anwendungen mit seltenen Statusmeldungen oder geringen Datenmengen kann dieser Bereich ein sinnvoller Kompromiss sein, da die Stromaufnahme nahe am unteren Bereich der Messreihe liegt, die Reaktionszeit aber noch unterhalb einer halben Sekunde bleibt.

Eine weitere Vergrößerung des Duty-Cycles bringt energetisch nur noch geringe Vorteile. Zwischen 1000 ms und 2000 ms sinkt die mittlere Stromaufnahme nur von etwa 0,56 mA auf etwa 0,51 mA. Gleichzeitig steigt die mittlere RTT von etwa 610,0 ms auf etwa 1112,1 ms, während die Downlinkdatenrate von etwa 0,95 kbit s^{-1} auf etwa 0,48 kbit s^{-1} fällt. In diesem Bereich ist der zusätzliche Gewinn an Batterielaufzeit daher klein im Vergleich zu den Einbußen bei Reaktionszeit und Downlinkdatenrate.

Für die untersuchte Konfiguration liegt der praktisch relevante Kompromissbereich daher vor allem zwischen 100 ms und 500 ms. Kürzere Duty-Cycles eignen sich für interaktivere Anwendungen mit höheren Anforderungen an Latenz und Downlinkdatenrate. Längere Duty-Cycles sind dagegen vor allem für stark energieoptimierte Sensorknoten geeignet, bei denen seltene Kommunikation und Antwortzeiten im Sekundenbereich akzeptabel sind.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

In dieser Arbeit wurden indirekter Datenübertragungen nach IEEE 802.15.4 für RIOT OS implementiert und analysiert. Ziel war es, die im Standard vorgesehenen Mechanismen für den Betrieb ohne Superframes verfügbar zu machen und damit eine Grundlage für energieeffiziente Kommunikation mit schlafenden Endgeräten zu schaffen.

Dazu wurden zunächst die relevanten IEEE 802.15.4 Primitiven analysiert und auf die Architektur von RIOT OS abgebildet. Auf dieser Grundlage wurde ein neues MAC-Layer entworfen, der eine zustandsbasierte Verarbeitung, eine PIB, Callback-Schnittstellen sowie eine Warteschlange für indirekte Übertragungen bereitstellt. Zusätzlich wurde die Integration in das bestehenden SubMAC-Layer und in GNRC umgesetzt, sodass Funktionen wie Scan, Start eines Koordinators, Assoziierung, Polling und indirekter Versand über RIOT Schnittstellen nutzbar sind.

Die Evaluation zeigt den zentralen Zielkonflikt indirekter Übertragungen. Durch Duty-Cycling sinkt die mittlere Stromaufnahme deutlich, gleichzeitig steigen Latenz und sinkt die erreichbare Downlinkdatenrate. Gegenüber dem dauerhaft aktiven Betrieb mit etwa 6,03 mA reduziert bereits ein Duty-Cycle von 100 ms die Stromaufnahme auf etwa 1,29 mA. Bei 500 ms werden etwa 0,65 mA erreicht. Der zusätzliche Energiegewinn bei noch größeren Duty-Cycles fällt dagegen geringer aus, während die mittlere RTT von etwa 311,7 ms bei 500 ms auf über eine Sekunde bei 2000 ms steigt.

Damit konnte gezeigt werden, dass indirekte Übertragungen in RIOT OS eine deutliche Reduktion des Energieverbrauchs ermöglichen, sofern höhere Latenzen und geringere Downlinkdatenraten für die jeweilige Anwendung akzeptabel sind. Für die untersuchte Konfiguration liegt ein sinnvoller Kompromiss insbesondere im Bereich zwischen 100 ms und 500 ms.

8.2 Ausblick

Diese Arbeit bildet eine Grundlage für eine vollständigere IEEE 802.15.4-Unterstützung in RIOT OS. Ein nächster Schritt ist die Erweiterung der implementierten Schnittstellen um bisher nicht betrachtete Funktionen des Standards. Dazu zählen insbesondere die Unterstützung des Betriebs mit Superframes sowie die Unterstützung von ED-Scans.

In dieser Arbeit wurde der Fokus bewusst auf den Betrieb ohne Superframes gelegt, da dieser für indirekte Übertragungen in vielen praktischen Szenarien relevant ist und eine geringere Synchronisationskomplexität aufweist. Der IEEE 802.15.4-Standard definiert jedoch auch einen beacon-aktivierten Betriebsmodus, in dem Koordinatoren periodisch Beacons aussenden und die Kommunikation innerhalb einer Superframe-Struktur zeitlich organisiert wird.

Eine zukünftige Erweiterung könnte daher die Unterstützung dieses beacon-aktivierten Betriebs umfassen. Dazu zählen insbesondere die Erzeugung und Verarbeitung periodischer Beacons, die Verwaltung der Superframe-Parameter sowie die Synchronisation von Endgeräten mit dem Koordinator. In diesem Zusammenhang würde auch der passive Scan relevant, da er auf periodisch ausgesendeten Beacons basiert und im Betrieb ohne Superframes keinen praktischen Nutzen hat.

Besonders interessant wäre anschließend ein Vergleich zwischen dem in dieser Arbeit betrachteten unsynchronisierten Polling-Betrieb und einem Superframe-basierten Ansatz hinsichtlich Energieverbrauch, Latenz und Downlinkdatenrate.

Weiterhin sollte die Implementierung auf weiteren Transceivern und Plattformen untersucht werden. Besonders relevant ist dabei das Zusammenspiel mit hardwareseitiger und softwareseitiger Unterstützung für ACKs, Frame Pending und Source Address Matching.

Auch die Evaluation kann erweitert werden. In dieser Arbeit wurden vor allem Szenarien mit einem Gerät und einem Koordinator betrachtet. Für zukünftige Untersuchungen wären größere Netze mit mehreren Endgeräten, höherer Last und unterschiedlichen Poll Intervallen sinnvoll. Dadurch könnte genauer analysiert werden, wie sich Warteschlangenlänge, Paketverluste und Latenz unter realistischeren Netzwerkbedingungen verhalten.

Schließlich kann die Integration in höhere Protokollschichten weiter ausgebaut werden. Da IEEE 802.15.4 häufig als Grundlage für 6LoWPAN, OpenThread oder andere IoT-

Protokolle dient, wäre eine Untersuchung der Auswirkungen indirekter Übertragungen auf diese Techniken ein sinnvoller nächster Schritt.

Literatur

[1] Amazon Web Services. *FreeRTOS Documentation*. 2026. URL: https://www.freertos.org/Documentation/00-Overview (besucht am 07. 06. 2026).

[2] Apache NuttX Project. *IEEE 802.15.4*. Apache NuttX documentation – IEEE 802.15.4 wireless subsystem. Apache Software Foundation. 7. Juni 2026. URL: https://nuttx.apache.org/docs/latest/components/wireless/ieee802154.html (besucht am 07. 06. 2026).

[3] Apache NuttX Project. *mac802154.c – IEEE 802.15.4 MAC implementation*. Apache NuttX source code, inspected revision on master branch. Apache Software Foundation. 7. Juni 2026. URL: https://github.com/apache/nuttx/blob/master/wireless/ieee802154/mac802154.c (besucht am 07. 06. 2026).

[4] Arm Mbed. *Mbed OS — Make your next idea a success with an open source IoT operating system*. Official overview of Mbed OS features and capabilities. Arm Mbed. 21. Jan. 2026. URL: https://os.mbed.com/mbed-os/ (besucht am 21. 01. 2026).

[5] Emmanuel Baccelli u. a. „RIOT OS: Towards an OS for the Internet of Things“. In: *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2013, S. 79–80. DOI: 10.1109/INFCOMW.2013.6970748.

[6] Emmanuel Baccelli u. a. „RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT“. In: *IEEE Internet of Things Journal* 5.6 (Dez. 2018), S. 4428–4440. URL: http://doi.org/10.1109/JIOT.2018.2815038.

[7] Devang G. Chavda u. a. „WIBREE Technology with Bluetooth“. In: *International Journal of Engineering Research and Applications (IJERA)* 2.3 (Mai 2012). Vol. 2, Issue 3 (May–Jun 2012), S. 400–403. ISSN: 2248-9622. URL: https://www.ijera.com/papers/Vol2_issue3/BQ23400403.pdf (besucht am 11. 02. 2026).

[8] Connectivity Standards Alliance. *Zigbee Specification. Zigbee Document 05-3474-23*. Version Revision 23.2, Version 1.0. Connectivity Standards Alliance. 10. Nov. 2025. URL: https://csa-iot.org/developer-resource/specifications-download-request/ (besucht am 01. 07. 2026).

[9] Contiki OS. *Contiki 2.6: 6LoWPAN implementation*. URL: https://contiki.sourceforge.net/docs/2.6/a01794.html (besucht am 02. 07. 2026).

[10] Adam Dunkels. *The ContikiMAC Radio Duty Cycling Protocol*. Techn. Ber. T2011:13. Swedish Institute of Computer Science, Dez. 2011. URL: http://dunkels.com/adam/dunkels11contikimac.pdf (besucht am 07. 06. 2026).

[11] Bradley Efron und Robert J. Tibshirani. *An Introduction to the Bootstrap*. New York: Chapman & Hall, 1993.

[12] J.A. Gutierrez u. a. „IEEE 802.15.4: a developing standard for low-power low-cost wireless personal area networks“. In: *IEEE Network* 15.5 (2001), S. 12–19. DOI: 10.1109/65.953229.

[13] Mor Harchol-Balter. „The Poisson Process“. In: *Introduction to Probability for Computing*. Cambridge: Cambridge University Press, 2023, S. 210–228.

[14] Jan-Hinrich Hauer. *TKN15.4: An IEEE 802.15.4 MAC Implementation for TinyOS 2*. Techn. Ber. TKN-08-003. Berlin, Germany: Telecommunication Networks Group (TKN), TU Berlin (TUB), März 2009.

[15] M. Honkanen, A. Lappetelainen und K. Kivekas. „Low end extension for Bluetooth“. In: *Proceedings. 2004 IEEE Radio and Wireless Conference (IEEE Cat. No.04TH8746)*. 2004, S. 199–202. DOI: 10.1109/RAWCON.2004.1389107.

[16] IEEE 802.15 Working Group. „Application of IEEE Std 802.15.4“. In: *IEEE 802.15 document 15-14-0226-01-0000* (Mai 2014). URL: https://mentor.ieee.org/802.15/dcn/14/15-14-0226-00-0000-802-15-4-applications.pdf.

[17] IEEE 802.15 Working Group. „IEEE Standard for Low-Rate Wireless Networks“. In: *IEEE Std 802.15.4-2024 (Revision of IEEE Std 802.15.4-2020)* (Dez. 2024), S. 1–967. DOI: 10.1109/IEEESTD.2024.10794632.

[18] José Álamos. *RDM0004: The IEEE802.15.4 radio HAL*. RIOT Design Memo RDM0004. RIOT-OS. URL: https://github.com/RIOT-OS/RIOT/blob/master/doc/memos/rdm0004.md (besucht am 05. 12. 2025).

[19] Matthias Kovatsch, Simon Duquennoy und Adam Dunkels. „A Low-Power CoAP for Contiki“. In: *Proceedings of the IEEE Workshop on Internet of Things Technology and Architectures*. Valencia, Spain, Okt. 2011. URL: http://dunkels.com/adam/kovatsch11low-power.pdf (besucht am 07. 06. 2026).

[20] Philip Levis u. a. „TinyOS: An Operating System for Sensor Networks“. In: *Ambient Intelligence*. Hrsg. von Werner Weber, Jan M. Rabaey und Emile Aarts. Berlin, Heidelberg: Springer, 2005, S. 115–148. ISBN: 978-3-540-27139-0. DOI: 10.1007/3-540-27139-2_7. URL: https://doi.org/10.1007/3-540-27139-2_7.

[21] Xiaoyun Li, Chris J. Bleakley und Wojciech Bober. „Enhanced Beacon-Enabled Mode for improved IEEE 802.15.4 low data rate performance“. In: *Wireless Networks* 18.1 (2012), S. 59–74. ISSN: 1572-8196. DOI: 10.1007/s11276-011-0387-y. URL: https://doi.org/10.1007/s11276-011-0387-y.

[22] Will Lord. *Important Update on Mbed*. Official Mbed blog announcement about platform status and future plans. Arm / Mbed. 30. Aug. 2024. URL: https://os.mbed.com/blog/entry/Important-Update-on-Mbed/ (besucht am 21. 01. 2026).

[23] George Oikonomou u. a. „The Contiki-NG Open Source Operating System for Next Generation IoT Devices“. In: *SoftwareX* 18 (2022), S. 101089. ISSN: 2352-7110. DOI: 10.1016/j.softx.2022.101089. URL: https://www.sciencedirect.com/science/article/pii/S2352711022000620.

[24] Usman Raza, Parag Kulkarni und Mahesh Sooriyabandara. „Low Power Wide Area Networks: An Overview“. In: *IEEE Communications Surveys & Tutorials* 19.2 (2017), S. 855–873. DOI: 10.1109/COMST.2017.2652320.

[25] RIOT Project. *IEEE 802.15.4 — RIOT OS API Documentation*. RIOT API documentation — IEEE 802.15.4 networking component. RIOT Project. 21. Jan. 2026. URL: https://api.riot-os.org/group__net__ieee802154.html (besucht am 21. 01. 2026).

[26] Michel Rottleuthner, Thomas C. Schmidt und Matthias Wählisch. *Duty-Cycling is Not Enough in Constrained IoT Networking: Revealing the Energy Savings of Dynamic Clock Scaling*. Technical Report 2508.09620. Aug. 2025. URL: https://arxiv.org/abs/2508.09620.

[27] Thread Group. *Thread 1.4 Features White Paper*. Online. Revision 1.0, September 4, 2024. Accessed: 2026-07-01. Sep. 2024.

[28] Pascal Thubert u. a. *IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header*. RFC 8138. Apr. 2017. DOI: 10.17487/RFC8138. URL: https://www.rfc-editor.org/info/rfc8138.

[29] Ishaq Unwala, Zafar Taqvi und Jiang Lu. „Thread: An IoT Protocol“. In: *2018 IEEE Green Technologies Conference (GreenTech)*. 2018, S. 161–167. DOI: 10.1109/GreenTech.2018.00037.

[30] Zephyr Project. *IEEE 802.15.4*. Zephyr Project Documentation (latest). Zephyr Project. 21. Jan. 2026. URL: https://docs.zephyrproject.org/latest/connectivity/networking/api/ieee802154.html#id6 (besucht am 21. 01. 2026).

[31] Zephyr Project. *Introduction – Zephyr Project Documentation*. Official documentation — Introduction to Zephyr OS. Zephyr Project. 21. Jan. 2026. URL: https://docs.zephyrproject.org/latest/introduction/index.html (besucht am 21. 01. 2026).

A Anhang

A.1 Verwendete Hilfsmittel

In der Tabelle A.1 sind die im Rahmen der Bearbeitung des Themas der Bachelorarbeit verwendeten Werkzeuge und Hilfsmittel aufgelistet.

Tabelle A.1: Verwendete Hilfsmittel und Werkzeuge

Tool	Verwendung
LaTeX	Textsatz- und Layout-Werkzeug verwendet zur Erstellung dieses Dokuments
PlantUML	Erstellung von Sequenz- und Komponentendiagrammen
draw.io	Skizzierung der Statemachine
python, matplotlib	Generierung von Graphen
ChatGPT	Verbesserung von Formulierungen

Glossar

6LoWPAN IPv6-Pakete werden über IEEE 802.15.4 gesendet.

GNRC Generischer, standard Netzwerkstack in RIOT.

OpenDSME Open-Source-Implementierung der IEEE 802.15.4 DSME-Erweiterung, verfügbar unter http://opendsme.org/.

OpenThread Mesh-Netzwerktechnologie basierend auf IEEE 802.15.4.

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit in allen Teilen selbstständig angefertigt und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt habe. Die in meiner Arbeit verwendeten KI-basierten Hilfsmittel habe ich (ggf. mit Produktnamen) angegeben.

Ich verantworte die Übernahme jeglicher von mir verwendeter maschinell generierter Passagen vollumfänglich selbst und trage die Verantwortung für eventuell durch die KI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate.

Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschungsversuch behandelt werden können.

Ort	Datum	Unterschrift im Original