BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Implementierung und Bewertung eines Algorithmus für selektive Retransmissions in einem 6LoWPAN Funknetz

Abschlussarbeit

vorgelegt von

Andreas Michael Bartusch

EDV.Nr.:776120

dem Fachbereich VII – Elektrotechnik und Feinwerktechnik –
der Beuth Hochschule für Technik Berlin vorgelegte Masterarbeit
zur Erlangung des akademischen Grades
Master of Engineering (M.Eng.)
im Studiengang
Kommunikations- und Informationstechnik

Tag der Abgabe 11. Oktober 2012

Gutachter

Prof. Dipl.-Inform. Thomas Scheffler	Beuth Hochschule für Technik
Prof. Dipl.-Ing. Hans-Otto Kersten	Beuth Hochschule für Technik

Kurzfassung

Hardwareplattformen für drahtlose Funknetze basierend auf dem Funkstandard IEEE 802.15.4 sind heutzutage so leistungsfähig, dass sie die Übertragung von IPv6 Paketen erlauben. Ermöglicht wird dies durch den Adaptionslayer IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN), spezifiziert durch die Internet Engineering Task Force (IETF). Ein limitierender Faktor bei der Übertragung stellt die geringe Framegröße des Link-Layers von nur 127 Bytes dar. Folglich muss das IP-Paket in viele Fragmente zerlegt werden! Sollte eines dieser Fragmente während des Transportes innerhalb des Funknetzes verloren gehen, ist es erforderlich, alle Teilstücke des IPv6-Paketes erneut zu übertragen, was die begrenzte Übertragungskapazität solcher Netze enorm belastet.

Aufbauend auf der 6LoWPAN-Implementierung des Betriebssystems Contiki wird in dieser Abschlussarbeit prototypisch ein Lösungsvorschlag für die selektive Bestätigung implementiert. Verloren gegangene Paketfragmente können so wiederholt übertragen und die Datenlast innerhalb des Netzwerkes reduziert werden. Dieser Protokollentwurf wird auf Ebene des Adaptionslayers, zwischen Schicht 2 und 3 des Open Systems Interconnection (OSI)-Referenzmodells realisiert.

Die Implementierung wird mit der ursprünglichen 6LoWPAN-Implementierung verglichen und bewertet. Grundlage dieser Bewertung bilden Messungen von relevanten Parametern wie Übertragungsreichweite, Paketverzögerung, Paketverlust, Retransmissionrate und Datendurchsatz.

Abschließend werden die Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf mögliche weiterführende Projekte gegeben.

Abstract

Nowadays, hardware platforms for wireless radio networks, based on the radio standard IEEE 802.15.4, are so efficient that they allow for the transmission of IPv6 packets. This is made possible by the 6LoWPAN shim layer, specified by IETF. A limiting factor when transmitting IPv6 packets is the small frame size of the link layer of only 127 bytes. Consequently, the IP packet has to be fragmented into several units! In case one of these fragments will be lost during transport within the radio network, all units of the IPv6 packet have to be retransmitted. This stresses massively the limited transmission capacity of such networks.

Based on the 6LoWPAN implementation of the Contiki operating system, in this final thesis an approach for the selective confirmation is implemented prototypically. So lost packet fragments can be retransmitted and data load within the network can be reduced. This protocol draft is realized on the shim layer between layer 2 and 3 of the OSI reference model.

The implementation is compared with the original 6LoWPAN implementation and assessed. Measurements of relevant parameters such as transmission range, packet delay, retransmission rate and data throughput provide the basis for this assessment.

Finally, the findings resulting from this final thesis are summarized and prospects for possible future projects are given.

Erklärung

Ich versichere, dass ich diese Abschlussarbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Datum Unterschrift

Aufgabenstellung der Masterarbeit „Implementierung und Bewertung eines Algorithmus für selektive Retransmissions in einem 6LoWPAN Funknetz“

6LoWPAN ist ein Adaptionslayer für die Übertragung von IPv6 Paketen über Funknetze mit geringem Energiebedarf nach IEEE 802.15.4, welcher von der IETF spezifiziert wird. Ein besonderes Kennzeichen dieser Netze ist unter anderem die geringe Framegröße des Link-Layers von 127 Byte, welche aufgrund der vergleichsweise hohen Bitfehlerraten und der beschränkten Verarbeitungskapazitäten der Netzknoten gewählt wurde.

Diese kleine Framegröße macht es erforderlich, dass der 6LoWPAN-Layer zu überragende IPv6 Pakete gegebenenfalls in eine Vielzahl von Fragmenten zerlegt und beim Empfänger wieder transparent zusammenfügt (Link-Layer Fragmentierung). Dabei sieht der derzeitige Standard keine Bestätigung der korrekten Übertragung der einzelnen Fragmente vor. Sollten dabei ein oder mehrere Fragmente verloren gehen oder beschädigt werden, so wird das komplette IPv6 Paket verworfen und muss erneut übertragen werden.

Im Rahmen dieser Arbeit soll ein Mechanismus zur selektiven Bestätigung der Übertragung von Fragmenten prototypisch implementiert, getestet und bewertet werden. Dazu soll eine existierende 6LoWPAN-Implementierung für das Betriebssystem Contiki angepasst und auf existierenden Sensorknoten des Typs Atmel Raven umgesetzt werden.

Im Rahmen von Messungen soll insbesondere ein Vergleich des Übertragungsverhaltens zwischen bestätigter und unbestätigter Frameübertragung durchgeführt werden. Dabei sind Parameter wie die maximale Übertragungsreichweite, Übertragungszeit, Paketverlustrate/Retransmissionrate und Datendurchsatz zu bestimmen.

Der Laboraufbau ist unter Berücksichtigung der vorhandenen Laborinfrastruktur und der vorhandenen Hardware zu realisieren. Diese sind im Einzelnen:

- 2 x Atmel Raven Board (Sensorknoten)
- 2 x 6LoWPAN Gateway (Notebooks)

Für die Bearbeitung der Aufgabenstellung werden gute Kenntnisse der Grundlagen der Telekommunikations- und Netzwerktechnik vorausgesetzt. Der Bearbeiter muss sich in das Thema 6LoWPAN und die Konfiguration und Programmierung des Contiki Betriebssystems einarbeiten.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Viele Innovationen in den Bereichen der Sensorik, Überwachung und Steuerung haben das Ziel unser Leben komfortabler und auch sicherer zu gestalten. Beispiele für solche Anwendungsfälle sind in der Endverbraucherelektronik, im Heimbereich bei Sicherheitssensoren, im Gesundheitswesen bei der Altenpflege oder in der Automobilindustrie bei Fahrzeugsensoren zu finden (siehe Abschnitt A.4). Sensornetzwerke, basierend auf dem Standard IEEE 802.15.4, stellen eine kostengünstige Lösung für die Umsetzung solcher zukunftsweisenden Ideen dar. Durch das Aufbrauchen des Adresspools des Protokolls Internet Protocol (IP) in der Version 4 und aus dem Wunsch heraus, auf einfache Art und Weise von jedem Ort der Welt auf die Stationen eines Sensornetzwerkes zugreifen zu können, wurde der Einsatz des Protokolls IPv6 auf den Geräten eines sogenannten Low-Rate Wireless Personal Area Network (LR-WPAN) interessant.

Lange Zeit bezweifelte man, dass die Umsetzung des Internetprotokolls auf so begrenzten Hardwareplattformen wie Sensorknoten überhaupt denkbar sei. Mit dem μIP-TCP/IP-Stack wurde eine Möglichkeit geschaffen, das Internetprotokoll in den Versionen 4 und 6 auf wenigen Kilobyte Random-Access Memory (RAM) und Kilobyte Read-Only Memory (ROM) umzusetzen.

In einem Wireless Personal Area Network (WPAN) werden für gewöhnlich nur kleine Datenmengen transportiert. Unter besonderen Umständen ist allerdings eine Übertragung größerer Datagramme nötig. Diese IP-Pakete werden durch den 6LoWPAN-Adaptionslayer in der sendenden Station in mehrere Fragmente aufgeteilt und in der Empfängerstation wieder zu einem vollständigen Datagramm zusammengesetzt. Die geringe Rahmengröße des Link-Layers des Standards IEEE 802.15.4 macht diesen Schritt notwendig, um ein solches Datagramm übertragen zu können.

Der Verlust eines einzelnen Fragmentes eines Datagramms durch äußere Störeinflüsse oder durch hohe Auslastung des Netzwerkes führt zu einem erneuten Senden des gesamten Datagramms, ausgelöst durch die höheren Schichten des OSI-Referenzmodells. Dieses Verhalten kann die Leistungsfähigkeit des Netzwerkes noch weiter beeinträchtigen. Im schlimmsten Fall ist ein Zusammenbruch der Kommunikation denkbar. Dieses unerwünschte Verhalten führte zu der Entscheidung, eine alternative Protokollimplementierung als Thema dieser Abschlussarbeit zu wählen.

Einen möglichen Lösungsansatz bietet der Entwurf „LoWPAN fragment Forwarding and Recovery“ von Pascal Thubert und Jonathan W. Hui an. Die beiden Entwickler beschreiben, wie man auf einfache Art und Weise die Bestätigung der übertragenen sowie das erneute Senden verlorengegangener Fragmente eines Datagramms auf hardwarebegrenzten Systemen implementieren

könnte. Auf diesem Weg hofft man, den zusätzlichen Netzwerkverkehr zu reduzieren und die Gesamtperformance des WPAN zu verbessern.

Ziel dieser Arbeit wird es daher sein, diesen Protokollentwurf auf dem AVR RZ RAVEN 2.4 GHz Wireless Evaluation Kit umzusetzen und die Implementierung mit dem ursprünglich eingesetzten 6LoWPAN-Adaptionslayer (erläutert in 2.2) zu vergleichen und das neue Protokoll zu bewerten.

1.1.1 Einführung

In vielen Anwendungsfällen werden über ein 6LoWPAN nur kleine Datenmengen von wenigen Bytes übertragen. Da ein Rahmen des Standards IEEE 802.15.4 zwischen 74 und 102 (Contiki 2.5) Bytes Nutzdaten übertragen kann, ist eine Fragmentierung oft nicht notwendig. Gelegentlich müssen jedoch größere Datenmengen wie bspw. bei der Realisierung von Firmwareupdates oder beim Abrufen von Logdateien übertagen werden. In diesen beiden Fällen ist ein sicherer Datentransfer von mehr als 10 Kilobytes nötig.

Mechanismen wie das TCP oder eine Segmentierung auf Ebene des Applikations-Layers werden genutzt, um sicheren Ende-zu-Ende-Transport zu gewährleisten. Eine Möglichkeit die Übertragung von großen Datenmengen über 6LoWPAN-Verbindungen zu unterstützen ist es, die Maximum Segment Size (MSS) so anzupassen, dass sie in eine 802.15.4-MTU passt. Leider kann dadurch erheblicher Headeroverhead hinzugefügt werden.

Ein anderer Mechanismus verbindet die 6LoWPAN-Fragmentierung mit einer Transport-Layer- oder Applikation-Layer-Segmentierung. Durch das Erhöhen der MSS wird der Protokolloverhead beim Ende-zu-Ende Transportprotokoll minimiert. Dadurch wird die Anzahl der ausstehenden Datagramme durch das Transportprotokoll verringert, idealerweise auf ein einzelnes Datagramm. Die Ankunft der Pakete beim Empfänger in falscher zeitlicher Reihenfolge – typisch für 6LoWPANs – wird reduziert. Der Standard [RFC4944] (Kushalnagar u. a.) definiert zwar einen Fragmentierungsmechanismus, aber es existiert leider keine Möglichkeit, einzelne, verlorengegangene Fragmente erneut anzufordern. Der Verlust eines einzelnen Fragments führt also zum erneuten Senden des Datagramms mit all seinen Fragmenten. Das bedeutet eine Verschwendung von Ressourcen in einem ohnehin schon ressourcenbegrenzten Netzwerk.

Erfahrungen in der Vergangenheit zeigten, dass falsch weitergeleitete oder verlorene Fragmente zu einem schlechten Verhalten des Netzwerks und möglicherweise zu Schwierigkeiten auf Ebene der Anwendungsschicht führen.

Für den Datenaustausch über einen Hop hinweg wurde bereits ein Bestätigungsmechanismus veröffentlicht, der ausreicht, um die Fragmente zu sichern. In einer Multihop-Umgebung kann ein Ende-zu-Ende-Fragment-Recovery-Mechanismus eine gute Ergänzung zu einem Hop-bei-Hop MAC-Ebene-Recovery sein. Dieser Entwurf stellt ein einfaches Protokoll zum Anfordern individueller Fragmente zwischen zwei 6LoWPAN-Endpunkten vor.

6LoWPAN-Endpunkte bezeichnen 6LoWPAN-Stationen, die 6LoWPAN-Header zum Zwecke des Transportes eines IPv6-Paketes verarbeiten oder erzeugen. In diesen Stationen werden die Fragmentierung und das Zusammensetzen (Reassembly) der Datagramme durchgeführt. [Hui u. Thubert, b, S. 2-3]

1.1.2 Notwendigkeit

Auch wenn in einem 6LoWPAN-Netzwerk üblicherweise nur kleine Datenmengen transportiert werden, existieren Anwendungsfälle, die das Übertragen großer Pakete erfordern. Mit diesen Datagrammen werden unter Umständen Funktionen realisiert, die einen effektiven Transport der Daten durch ein Low-Power Wireless Personal Area Networks (LoWPAN) erfordern.
Beispiele für solche Anwendungsfälle können unterschiedlicher Natur sein wie:

- Das Senden von Befehlspaketen oder vollständigen Konfigurationen an eine Station. Die Befehle können einzeln ausgeführt werden – alternativ kann das System auch in einen früheren Zustand zurückversetzt werden.
- Das Übertragen eines Firmwareupdates, um eine neue Softwareversion auf den Stationen einsetzen zu können. Hierbei wird in der Regel eine große Anzahl von Stationen innerhalb einer kurzen Zeitspanne aktualisiert.
- Das Übertragen von gesammelten Messdaten oder Logdateien zwecks Analyse bzw. Diagnose von einer Station an einen Kommunikationspartner. Die Daten werden hier in einem einzelnen großen Paket versandt.
- Das Versenden komplexer Datentypen, die unter Umständen mehr als ein Fragment erfordern.

Das unkontrollierte Versenden einer neuen Firmware oder das Herunterladen größerer Datenpakete kann leicht zu einem enorm ansteigenden Datenverkehr und somit zu einer Auslastung des Netzwerkes führen. Dadurch können einzelne Fragmente verloren gehen, was zu einem erneuten Senden sämtlicher Fragmente eines Datagramms führt und den Netzwerkverkehr noch weiter belastet. Dieses Verhalten führt möglicherweise zu einem enormen Performanceverlust bis hin zu einem Zusammenbruch der Kommunikation.

Diese Auslastung kann weiterhin zu starken Funkstörungen bzw. Interferenzen und willkürlichem Verwerfen der Datenpakete in den weiterleitenden Knoten führen. Darüber hinaus kann das Warten, bis die Zielstation aus ihrem Ruhezustand erwacht, zu zusätzlichem Zwischenspeichern der Pakete und Speicherauslastung führen.

Um die Schwere des Problems zu demonstrieren, wird eine recht sichere Rahmenübertragungsrate von 99,9 Prozent über einen einzelnen 802.15.4-Hop angenommen. Die erwartete Übertragungsrate eines Datagramms, bestehend aus fünf Fragmenten, würde ungefähr 99,5 Prozent betragen. Die erwartete Übertragungsrate fällt auf 95,1 Prozent, wenn dieses Datagramm über 10 Hops, eine sinnvolle Größe für ein 6LoWPAN-Netzwerk, transportiert wird. Die erwartete Übertragungsrate für ein 1280-Byte-Datagramm beträgt demzufolge 98,4 Prozent für einen einzelnen Hop und fällt nach 10 Hops auf 85,2 Prozent.

Unter der Annahme, dass die minimale Maximum Transmission Unit (MTU) für das Protokoll IPv6 1280 Bytes beträgt und ein 802.15.4-Rahmen Nutzdaten von nur 74 Bytes übertragen kann, wird das Paket durch den 6LoWPAN-Adaptionslayer in mindestens 18 einzelne Fragmente aufgeteilt. Zu bedenken ist, dass der maximale Overhead bei Verwendung der 6LoWPAN-Fragmentierung und des Mesh-Headers zu einer Aufteilung auf ungefähr 32 Fragmente führt. Dieser Fragmentierungsgrad ist deutlich höher, als man ihn bisher bei Internetverkehr mit dem Protokoll IPv4 gewöhnt ist. [Hui u. Thubert, b, S. 3-4]

1.2 Gliederung der Arbeit

Diese Abschlussarbeit setzt sich, inklusive dieser Einleitung, aus sieben Kapiteln zusammen. Nach den einleitenden Worten, welche die Motivation zu dieser Abschlussarbeit beschreiben und einen inhaltlichen Überblick vermitteln, beschäftigt sich das zweite Kapitel mit den theoretischen Hintergründen. Es wird der Standard IEEE 802.15.4 beschrieben und der Fokus auf die Sicherungsschicht, die auf Ebene 2 des OSI-Referenzmodells realisierbaren Netztopologien und die Kommunikation zwischen den verschiedenen Stationen eines WPAN gelegt. Der Sicherheitsaspekt und mögliche Verschlüsselungsverfahren würden den Rahmen dieser Arbeit sprengen – daher wird auf diese Thematik nicht näher eingegangen.

Ein Großteil des Kapitels ist dem 6LoWPAN-Adaptionslayer gewidmet. Nach Erläuterung der Aufgaben eines Adaptionslayers werden der grundlegende Aufbau des 6LoWPAN-Formates und mögliche Routingmechanismen auf den Ebenen 2 und 3 des OSI-Referenzmodells beschrieben. Der 6LoWPAN-Adaptionslayer setzt eigene Headerkomprimierungsverfahren ein, um die begrenzte Übertragungskapazität eines Datenrahmens nach Standard IEEE 802.15.4 nicht unnötig zu reduzieren. Dabei wird die grundlegende Funktionsweise der in diesem Zusammenhang eingesetzten zustandslosen und kontextbasierten Komprimierungsmechanismen betrachtet. Anschließend wird der Fragmentation- und Reassembly-Prozess für die Übertragung größerer IPv6-Datagramme erläutert.

Die letzten Abschnitte des zweiten Kapitels sind dem offenen Betriebssystem Contiki und dem μIP-Stack gewidmet. Der μIP-Stack ist ein wichtiger Bestandteil dieses Betriebssystems – in diesem Zusammenhang wird auch auf die ereignisorientierte Benutzerstelle dieses IP-Stacks eingegangen.

Das dritte Kapitel beschreibt den allgemeinen Aufbau eines Sensorknotens und vermittelt eine Vorstellung davon, wie der 6LoWPAN-Adaptionslayer auf solchen Hardwareplattformen realisiert werden kann. Abgeschlossen wird das Kapitel mit einer allgemeinen Beschreibung der für die Lösung der Aufgabenstellung genutzten Hardwareplattform AVR RZ RAVEN und dem Programmiergerät AVR JTAGICE mkII.

Das folgende Kapitel schildert den Protokollentwurf, der im Rahmen dieser Abschlussarbeit auf der gewählten Hardwareplattform auf der Basis des Betriebssystems Contiki umgesetzt werden soll. Darüber hinaus werden alle für die Lösung der Aufgabenstellung gefällten Entscheidungen und Gedankengänge dargelegt und vorbereitende Arbeitsschritte beschrieben.

Das fünfte Kapitel dokumentiert den praktischen Teil der Arbeiten. Dabei wird auf die Analyse des Betriebssystems Contiki und der in Frage kommenden Quellcodestellen eingegangen und die gewählte Strategie für die Umsetzung des Protokolls dargelegt. Dieses Kapitel beschreibt abschließend mögliche Szenarien für eine Bewertung des Protokollentwurfs und erläutert den dafür entworfenen Messaufbau.

Das Kapitel 6 befasst sich mit der Analyse des Netzwerkverkehrs von Originalimplementierung und Protokollentwurf. Am Beispiel einer Retransmission wird der Beweis der Funktion erbracht. Die durchgeführten Messungen zu relevanten Parametern wie Paketumlaufzeit, Paketverlustrate, Retransmissionrate und Durchsatz werden analysiert, graphisch dargestellt, mit der ursprünglichen Implementierung verglichen und letztendlich bewertet.

Im letzten Kapitel werden die Ergebnisse dieser Abschlussarbeit und gewonnene Erkenntnisse zusammengefasst und ein Ausblick auf mögliche weiterführende Arbeiten gegeben.

Kapitel 2

Theoretische Grundlagen

Ziel dieses Kapitels ist es, ein besseres Verständnis für die Thematik dieser Abschlussarbeit zu schaffen. Dabei wird der damit im Zusammenhang stehende theoretische Hintergrund näher beleuchtet. Da der innerhalb dieser Arbeit implementierte Protokollentwurf auf dem Standard IEEE 802.15.4 aufsetzt, wird dieses drahtlose Konzept beschrieben. Hier wird der Schwerpunkt auf die Sicherungsschicht des OSI-Referenzmodells gelegt.

In diesem Kontext wird der allgemeine Aufbau eines Sensornetzwerkes, speziell die verwendeten Gerätetypen, die Adressierungsarten und die realisierbaren Netztopologien dargelegt. Darüber hinaus wird die Art und Weise betrachtet, wie die einzelnen Stationen eines WPANs auf dieser Ebene des OSI-Referenzmodells miteinander kommunizieren. Der Sicherheitsaspekt und mögliche Verschlüsselungsverfahren spielen eine eher untergeordnete Rolle, daher wird auf diese Thematik nicht näher eingegangen.

Da die Retransmissionimplementierung auf Ebene des Adaptionslayers, also zwischen der Sicherungsschicht und der Vermittlungsschicht, umgesetzt wird, ist ein umfangreicher Teil dieses Kapitels dem 6LoWPAN-Adaptionslayer gewidmet. Nach Erläuterung der Funktionen eines Adaptionslayers, wird der grundlegende Aufbau des 6LoWPAN-Formates und das Konzept des *Dispatch-Bytes* beschrieben. Darüber hinaus werden angewandte Routingkonzepte auf den Ebenen 2 und 3 des OSI-Referenzmodells im Zusammenhang mit 6LoWPANs erklärt. Dabei ist das Ebene 3-Routing das Konzept, welches durch die Entwickler des Betriebssystems Contiki umgesetzt wurde.

Aufgrund der limitierten Übertragungskapazität des Standards IEEE 802.15.4 werden durch den 6LoWPAN-Adaptionslayer eigene Headerkomprimierungsverfahren angewandt. Dieses Kapitel beschreibt unter anderem die grundlegende Funktionsweise der zu diesem Zwecke eingesetzten zustandslosen und kontextbasierten Komprimierungsmechanismen. Diese wurden bereits in Contiki auf Ebene des Adaptionslayers umgesetzt.

Anschließend wird der Fragmentation- und Reassembly-Prozess für die Übertragung größerer IPv6-Datagramme erläutert. Es wird eine Möglichkeit beschrieben, wie dieser Vorgang auf einem Sensorknoten softwaretechnisch umgesetzt werden kann.

Abschließend wird das offenen Betriebssystem Contiki und der μIP-Stack erläutert. Der μIP-Stack ist ein wichtiger Bestandteil dieses Betriebssystems. Dabei wird auch auf die ereignisorientierte Benutzerstelle dieses IP-Stacks und den Rime-Stack für die Realisierung der untersten Schichten des OSI-Referenzmodells eingegangen.

2.1 Der Funkstandard 802.15.4

Ein IEEE 802.15.4-LR-WPAN ist ein Netzwerk, das im Hinblick auf niedrige Kosten und sehr energiesparender drahtloser Kommunikation über eine kurze Entfernung zwischen mehreren Geräten entwickelt wurde. Der Standard beschreibt die Bitübertragungsschicht (auch Physical Layer (PHY) genannt) und die Sicherungsschicht des OSI-Referenzmodells. Die Definitionen des Standards wirken dem aktuellen Trend in der drahtlosen Technologie entgegen, wobei der Fokus auf eine drahtlose Kommunikation unter Verwendung hoher Datenraten und verbessertem Quality of Service (QoS) gelegt wird. Dadurch wird es möglich, anspruchsvolle Anwendungen aus dem Multimediabereich zu realisieren. [Gutierrez u. a., 2011, S. 3]

2.1.1 Protokollstapel

Das OSI-Referenzmodell ist auf einen von der International Standards Organization (ISO) entwickelten Vorschlag zurückzuführen. Es sollte ein Schritt in Richtung Standardisierung der in den verschiedenen Schichten eingesetzten Protokolle sein und wurde 1995 überarbeitet. Es dient dazu, offene Systeme miteinander zu verbinden und besteht aus sieben Schichten. Die Schichten des Modells wurden nach den folgenden Leitsätzen gebildet:

- Eine Schicht sollte geschaffen werden, falls eine abweichende Abstraktion benötigt wird.
- Jede Schicht sollte eine klar abzugrenzende Funktion erfüllen.
- Die Funktion einer jeden Schicht sollte im Hinblick auf international standardisierte Protokolle gewählt werden.
- Die Grenzen der Schichten sollten so gewählt werden, dass der Informationsfluss durch die Schnittstelle minimiert wird.
- Die Anzahl der Schichten sollte groß genug sein, so dass eigenständige und unabhängige Funktionalitäten nicht gezwungenermaßen in einer Schicht realisiert werden, aber dennoch klein genug, damit die Architektur nicht unhandlich wird.

[Tanenbaum, 2011, S. 63-64]

2.1.2 Bitübertragungsschicht

Die Bitübertragungsschicht des OSI-Referenzmodells stellt das Interface und das physikalische Medium bereit, über welches die Kommunikation abgewickelt wird. Die Bitübertragungsschicht ist die niedrigste Schicht innerhalb des OSI-Referenzmodells und ermöglicht im Falle des Standards IEEE 802.15.4 das Aktivieren und Deaktivieren des Funktransceivers, die Energiedetektion auf einem Übertragungskanal, das Steuern der Verbindungsqualität, den Zugriff über das Clear Channel Assessment (CCA)-Verfahren, die Kanalauswahl sowie die Übertragung und den Empfang von Nachrichtenpaketen durch das physikalische Medium.

Die Regierungen in der ganzen Welt verwalten das Funkfrequenzspektrum. Dieses Spektrum wird in verschiedene Bänder unterteilt und den unterschiedlichsten Diensten und Anwendungen zugewiesen, welche bspw. die Bereiche Telefonie, Vernetzung, militärische Kommunikation, Fernsehen oder Radio abdecken. Oft erfordert das Nutzen dieser Frequenzbänder eine bestimmte Lizenz. Einige Bereiche sind jedoch für die lizenzfreie Nutzung reserviert. Sie können genutzt werden, wenn die regionalen Vereinbarungen bezüglich der Grenzen von Ausgangsleistung, Auslastungsgrad, Modulation oder anderen Parametern nicht verletzt werden. Die genauen Einzelheiten dieser unlizenzierten Dienste variieren auf der Welt. Generell ist zu sagen, dass die Geräte einer Reihe behördlicher Beschränkungen unterliegen. Diese sind vom Dienst bzw. der Anwendung abhängig.

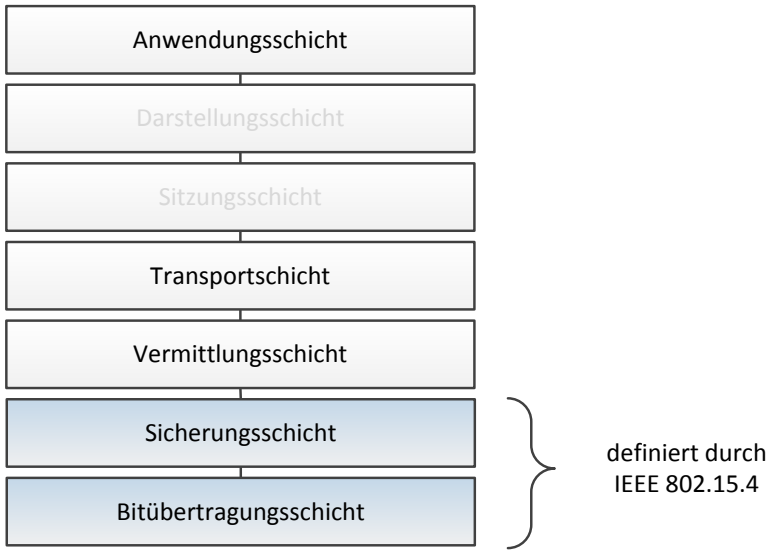


Abbildung 2.1: Einordnung des Standards 802.15.4 in das OSI-Referenzmodell

Implementierungen des Standards IEEE 802.15.4 sind an die örtlichen Bestimmungen des Landes anzupassen, in dem sie eingesetzt werden. Im Fall von Europa, Japan, Kanada und den Vereinigten Staaten von Amerika bestehen die Bestimmungen aus lizenzfreien aber typgenehmigten Direct Sequence Spread Spectrum (DSSS)-Diensten. Die Beschränkungen für diese lizenzfreien Typvorgaben bestimmen letztendlich das zu nutzende Frequenzband und einige weitere Charakteristiken des Dienstes. [Gutierrez u. a., 2011, S. 43-44]

Die Bitübertragungsschicht des Standards IEEE 802.15.4 spielt für die Lösung der Aufgabenstellung dieser Abschlussarbeit eine eher untergeordnete Rolle. Der Anhang (Abschnitt A.1) enthält jedoch zusätzliche Informationen über die Frequenzbänder, die eingesetzten Modulationsverfahren und die Dienste auf der Bitübertragungsschicht.

2.1.3 Die Sicherungsschicht

Die Media Access Control (MAC)-Unterschicht bildet gemeinsam mit der Logical Link Control (LLC)-Unterschicht die Sicherungsschicht des OSI-Referenzmodells. Die Sicherungsschicht ist für den Transport ganzer Datenrahmen in einem bestimmten Netzabschnitt verantwortlich. Die MAC-Schicht ermöglicht den Zugriff auf das gemeinsam genutzte Übertragungsmedium und sorgt für einen sicheren Datentransfer. Die LLC-Schicht stellt eine Zwischenschicht bereit und ist in IEEE 802.2 für sämtliche Protokolle der 802-Familie definiert. Die Unterschicht gestattet es bereits auf Ebene der Sicherungsschicht verschiedene Verkehrstypen bereitzustellen, die Kriterien wie Leitungsvermittlung oder besondere Zuverlässigkeit garantieren.

Das Konzept der LLC-Schicht hat sich bei der Realisierung nicht durchgesetzt, so dass die Netzwerkschicht direkt auf die MAC-Schicht zugreift.

Im Falle des WPAN ist die optimale Nutzung des drahtlosen Übertragungsmediums zwar wün-

schenswert, leider arbeitet dieser Dienst in eingeschränkten lizenzfreien Frequenzbändern, die er sich zudem mit anderen, nicht in jedem Fall standardisierten, drahtlosen Technologien teilen muss. Hierunter fällt auch das Wireless Local Area Network (WLAN) welches sich heutzutage in vielen Bereichen durchgesetzt hat.

Der Standard IEEE 802.15.4 verwendet für den Zugriff auf dieses mehrfachgenutzte Medium den Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)-Algorithmus. Dieses Verfahren macht es erforderlich, vor dem Senden eine gewisse Zeit in den Übertragungskanal hineinzulauschen, um mögliche Kollisionen mit anderen ausgehenden Übertragungen zu vermeiden.

Die MAC-Schicht des Standards 802.15.4 hat die folgenden Aufgaben zu erfüllen:

- Erzeugen von Bestätigungsrahmen
- Eingliedern der Geräte in ein Netzwerk
- Abmelden der Geräte aus einem Netzwerk
- Bereitstellen von Sicherheitsmechanismen
- Erzeugung der Beacon-Signale
- optional das Realisieren des Guaranteed Time Slot (GTS)-Managements für die Verwendung in der Stern-Topologie

Die MAC-Schicht des Standards 802.15.4 wurde entwickelt, um die Implementierung eines sehr einfachen Protokollstacks zu ermöglichen. Das erleichtert die schnelle Entwicklung von Anwendungen und hat eine direkte Auswirkung auf den Energieverbrauch – das Geheimnis liegt hier in der Einfachheit. [Gutierrez u. a., 2011, S. 79] [Kupris u. Sikora, 2007, S. 65]

Die 2006 veröffentlichte Überarbeitung des Original Standards beinhaltete einige neue Funktionen und Modifikationen der MAC-Unterschicht. Mit Ausnahme der Sicherheitsmechanismen sind alle neuen Funktionen abwärtskompatibel. Im Institute of Electrical and Electronics Engineers (IEEE) Standard 802.15.4-2006 wurde die Abarbeitung der Sicherheitsmechanismen schlanker und einfacher gestaltet. Weitere Änderungen sind:

- ein weiteres Datenfeld, welches die Version des Rahmens anzeigt
- eine Funktion, welche die Synchronisation von Endknoten erleichtert (der Mechanismus wird auf höherer Ebene implementiert)
- das Nutzen der garantierten Zeitschlitze, auch GTS genannt, ist nun optional
- die Dienstelemente wurden aktualisiert, um die Kanalseiten basierend auf den neuen Betriebsmodi der Bitübertragungsschicht nutzen zu können
- der Sendezeitpunkt der Beacon-Signale kann nun beeinflusst werden, um das Aneinanderreihen von Superframes zu ermöglichen
- das Broadcasting in Netzwerken mit aktiviertem Beacon-Signal wurde vereinfacht
- durch neue Funktionen wurde die Anmeldezeit in einem Netzwerk mit deaktiviertem Beacon-Signal reduziert

[Gutierrez u. a., 2011, S. 80]

Gerätetypen

Der Standard IEEE 802.15.4 definiert zwei Gerätetypen, das Full Function Device (FFD) und das Reduced Function Device (RFD). Das FFD enthält den vollständigen Satz der MAC-Dienste und darf in beliebigen Netztopologien eingesetzt werden. Dabei kann es in jeder der drei möglichen Rollen auftreten. In einen WPAN ist das FFD also als Personal Area Network (PAN)-Koordinator, Koordinator oder einfaches Netzwerkgerät anzutreffen. In der Rolle das PAN-Koordinator kann dieser Gerätetyp mit jeder Station kommunizieren.

Das RFD kommuniziert nur mit dem Point Coordinator (gleichbedeutend mit Access Point (AP) oder Zugangspunkt) in Gestalt des PAN-Koordinators und wird ausschließlich in der Stern-Topologie verwendet, da es nur einen Teil der Dienste der MAC-Teilschicht enthält. Es ist nicht in der Lage, die Rolle des PAN-Koordinators zu übernehmen und ermöglicht den preisgünstigen Aufbau eines Netzwerkes. Dieser Gerätetyp kann auf sehr einfachen Plattformen mit minimalen Hardwareressourcen bezüglich Rechenleistung und Arbeitsspeicher implementiert werden.
[Gutierrez u. a., 2011, S. 35] [Kupris u. Sikora, 2007, S. 67-68]

Adressierung

Zur Identifikation der Stationen eines Netzwerkes wird auf Ebene der Sicherungsschicht, unabhängig von der Topologie, eine acht Byte lange Adresse verwendet. Diese Adresse kennzeichnet ein individuelles Gerät und wird bei der Herstellung während eines Flashvorgangs mit der Hardware verknüpft.

Die Adresse entspricht dem genormten Extended Unified Identifier (EUI-64) Format des IEEE. Die ersten drei Bytes dieser Adresse kennzeichnen den Hersteller des Gerätes, die restlichen Bytes werden durch das Unternehmen selbst vorgegeben.

Die Verwendung dieser Adressen erzeugt einen erheblichen Overhead innerhalb des Datenrahmens des MAC-Schicht. Um diesen Overhead zu reduzieren, besteht die Möglichkeit, der Station innerhalb des Anmeldevorgangs vom PAN-Koordinator eine zwei Byte lange Kurzadresse zuzuweisen. Der PAN-Koordinator stellt sicher, dass die Adresse innerhalb des Netzwerkes einzigartig ist und erlaubt die Anmeldung von maximal 65535 Stationen – doch solche Netze sind in der Regel nicht praktikabel. [Gutierrez u. a., 2011, S. 80] [Kupris u. Sikora, 2007, S. 69]

Netztopologien

Unter Berücksichtigung der beiden vorgestellten Gerätetypen FFD und RFD sind auf der Ebene der MAC-Schicht die Netztopologien Stern und Peer-to-Peer realisierbar. Die beiden Topologien sind in der Abbildung 2.2 noch einmal graphisch veranschaulicht worden. Das Management für diese Topologien wird in der Netzwerkschicht realisiert und ist nicht im Rahmen des Standards 802.15.4 definiert.

In der *Stern-Topologie* wird die Kommunikation von einem einzigen PAN-Koordinator gesteuert, der in diesem Netzwerk als eine Art Master arbeitet. Er verwaltet alle Netzwerkgeräte (auch Endknoten oder Station genannt) und sendet für ihre Synchronisation Beacon-Signale (inkl. Superframe Control). In dieser Topologieform kommunizieren die Geräte ausschließlich über den PAN-Koordinator. Jedes FFD kann sein eigenes Netzwerk aufbauen, indem es zum PAN-Koordinator ernannt wird. Dabei arbeitet jedes Stern-Netzwerk unabhängig von seinen Nachbarnetzwerken.

Während des Aufbaus eines neuen Stern-Netzwerkes wählt der PAN-Koordinator einen Netzwerk Identifier, die sogenannte PAN-ID. Sie darf in keinem der Netzwerke in unmittelbarer Nähe ver-

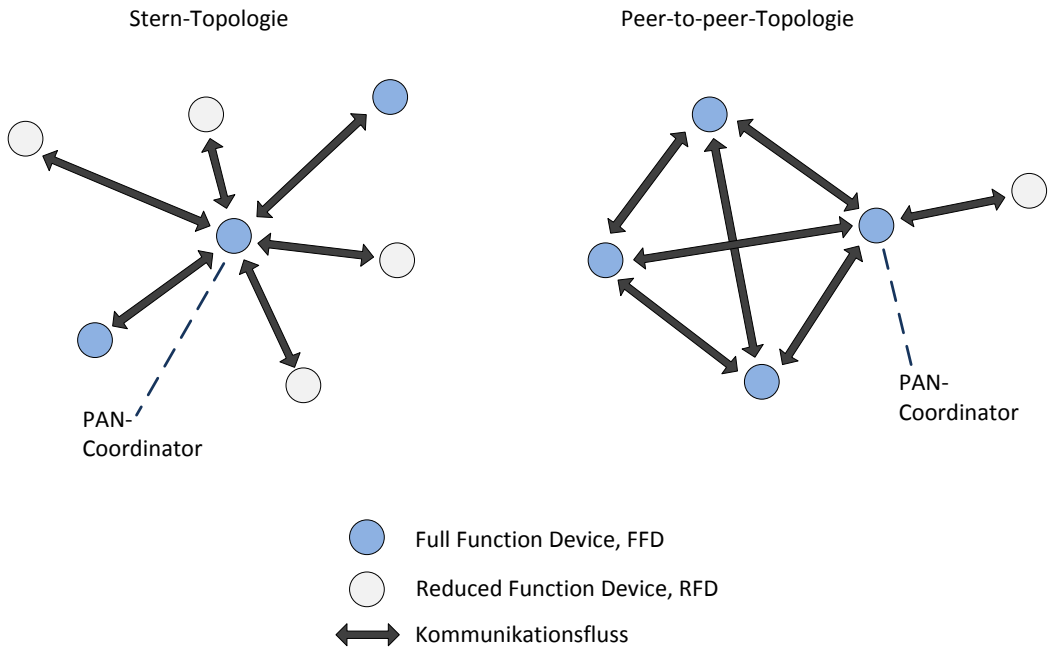


Abbildung 2.2: Netzwerktopologien gemäß Standard IEEE 802.15.4, abgezeichnet von [Kupris u. Sikora, 2007, S. 68]

wendet werden. Das wird dadurch ermöglicht, dass man alle verfügbaren bzw. gewählten Kanäle nach existierenden Netzwerken überprüft und dann eine PAN-ID wählt, die sich von den vorhandenen PAN-IDs unterscheidet. Nach dieser Prozedur beginnt der PAN-Koordinator mit dem Senden von Beacon-Signalen. In festgelegten Intervallen gewährt er Netzwerkgeräten, die dem Netzwerk beitreten wollen, den Zutritt.

Netzwerkgeräte die darauf programmiert sind, einem Stern-Netzwerk beizutreten, müssen zuerst nach Netzwerken innerhalb ihrer Reichweite suchen. Das geschieht durch das Auswerten der Beacon-Signale, die von einem PAN-Koordinator gesendet wurden. Nach der Suche können höhere Schichten des OSI-Referenzmodells das Netzwerk auswählen, dem man beitreten möchte. Dies wird durch das Senden einer „Beitreten Anfrage“ an den PAN-Koordinator realisiert. Der PAN-Koordinator entscheidet im Gegenzug, ob das Gerät beitreten darf oder nicht.

Das Stern-Netz unterstützt auch einen Mode mit nicht aktiviertem Beacon-Signal. In diesem Fall nutzt der PAN-Koordinater die Beacon-Signale nur, um dem Endknoten das Beitreten in das Netzwerk zu ermöglichen. Der Datenaustausch wird durch das zyklische Anpollen der Endknoten durch den PAN-Koordinator erreicht. [Gutierrez u. a., 2011, S. 80-82] [Kupris u. Sikora, 2007, S. 68]

Die *Peer-to-Peer-Topologie* ermöglicht die Kommunikation eines beliebigen FFD mit jedem anderen FFD in seiner Reichweite. Nachrichten für FFD außerhalb seiner Reichweite werden über das Multihop-Routingverfahren weitergegeben. Diese Netztopologie ermöglicht den Aufbau komplexer, großer Netzwerke einschließlich ad hoc, „selbstorganisierender“ und „selbstheilender“ Strukturen. Der Standard IEEE 802.15.4 spezifiziert keine Details dieser Netzwerke, sondern definiert nur die Funktionen auf der MAC-Schicht, um sie realisieren zu können.

RFDs können in einem Peer-to-Peer-Netzwerk nur zum Einsatz kommen, wenn sie als Peripheriegeräte verwendet werden, da ihnen die Eigenschaft fehlt, Datenrahmen weiterleiten zu können. Daraus ergibt sich, dass eine ausreichende Anzahl an FFD vorhanden sein muss, um das Netzwerk aufbauen zu können. Es gilt zu beachten, dass die Peer-to-Peer-Kommunikation zusätzliche Ressourcen an Arbeitsspeicher auf dem Endknoten erfordert, um die Routingtabellen der höheren Netzwerkschicht verarbeiten zu können.

Eine besondere Form des Peer-to-Peer-Netzwerkes ist das *Cluster-Tree-Netzwerk*. Es besteht aus einer veränderbaren Anzahl von Koordinatoren, die als Cluster Manager oder Cluster Head innerhalb der Cluster-Tree-Struktur dienen können. Sie können jedoch auch als Router für die Weiterleitung der Nachrichten in das Netzwerk verwendet werden. Der größte Vorteil dieser Topologieform ist ihr hierarchischer Aufbau. Dadurch kann der Routingalgorithmus für den Transport der Nachrichten durch das Netzwerk stark vereinfacht werden. Die Abbildung 2.3 zeigt ein Beispielnetzwerk dieses Typs.

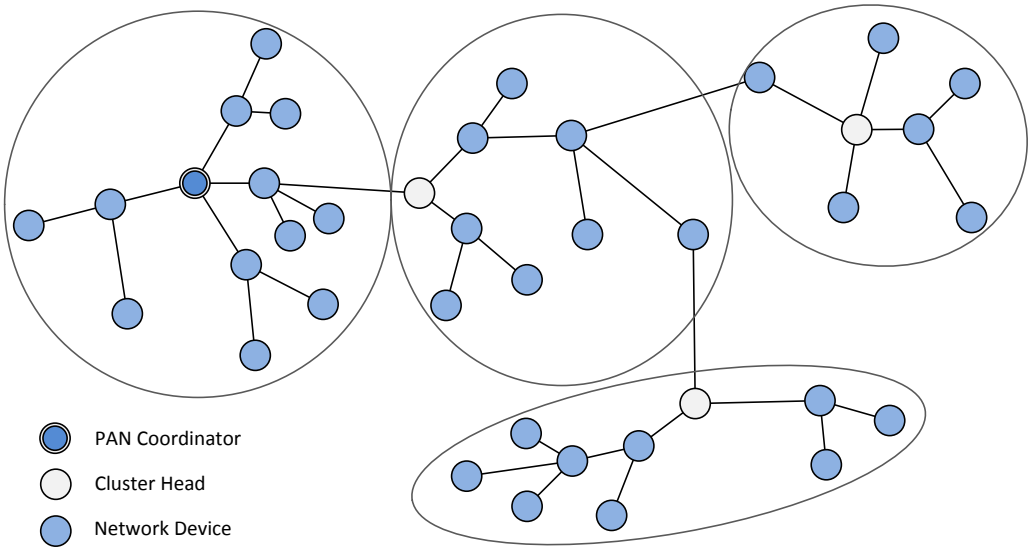


Abbildung 2.3: Cluster-Tree-Topologie gemäß Standard IEEE 802.15.4, abgezeichnet von [Gutierrez u. a., 2011, S. 83]

Ähnlich wie bei dem Vorgang des Startens eines Netzwerkes in Stern-Topologie, wird ein neues Peer-to-Peer-Netzwerk durch ein FFD in der Rolle des PAN-Koordinators aufgebaut. Der PAN-Koordinator wählt auch hier eine PAN-ID die sich von den vergebenen Identifiern der Netzwerke in Reichweite unterscheidet. In den meisten Fällen wird das Peer-to-Peer-Netzwerk mit nicht aktiviertem Beacon-Signal arbeiten – es existiert jedoch eine Betriebsart mit aktiviertem Beacon-Signal.

Endknoten die für den Einsatz in einer Peer-to-Peer-Topologie vorgesehen sind, suchen nach verfügbaren Netzwerken in ihrer Reichweite, um einen PAN-Koordinator in der Nähe zu finden oder alternativ dazu ein Gerät mit Routerfunktionalität, welches stellvertretend für einen PAN-Koordinator im Netzwerk agiert. Der IEEE Standard 802.15.4 bezeichnet Geräte mit Routerfunk-

tionalität als Koordinatoren.

Nach abgeschlossener Suche kann das Gerät durch das Senden einer „Beitreten Anfrage“ an den PAN-Koordinator oder alternativ zu einem der Koordinatoren in der Nähe Mitglied eines der gefundenen Netzwerke werden. Diese Funktionalität wird auf den höheren Schichten des OSI-Referenzmodells implementiert.

Der Koordinator gestattet dem Endknoten im Gegenzug den Beitritt oder verweigert ihn. Für den Fall, dass das kürzlich hinzugefügte Gerät ein FFD ist, wird es auch zu einem Koordinator und stellt die Routingdienste für andere Geräte bereit, die Teil des Netzwerks sind oder ihm beitreten möchten. [Gutierrez u. a., 2011, S. 82-83]

Superframe-Struktur

Der Standard IEEE 802.15.4 erlaubt die Implementierung einer optionalen Superframe-Struktur, die vom PAN-Koordinator verwaltet wird. Sie wird von Beacon-Signalen begrenzt, die durch den Koordinator in regelmäßigen Intervallen gesendet werden. Die Struktur dieses Superframes kann je nach Anwendungsfall konfiguriert werden. Kleine Netze in Stern-Topologie mit niedrigen Latenzzeiten sind ebenso denkbar wie sehr große Netze mit großer Latenz, bei denen die Nachrichten über mehrere Stationen zum Zielendpunkt transportiert werden.

Jedes Beacon-Signal enthält Informationen, welche die Synchronisation der Stationen in einem Netzwerk erleichtern. Diese Informationen enthalten den Netzwerk Identifier (PAN-ID) sowie den zeitlichen Abstand der Beacon-Signale und Superframes. Ein Superframe ist in 16 zusammenhängende Zeitschlitze unterteilt. Der Anfang des ersten dieser Zeitschlitze fällt mit dem Beginn des Beacon-Signals zusammen.

Möchte also eine Station mit dem PAN-Koordinator in Verbindung treten, so hat dies in der Zeit zwischen zwei aufeinanderfolgenden Beacon-Signalen zu erfolgen. Diese Zeitperiode wird Contention Access Period (CAP) oder auch Zugriffskonfliktperiode genannt. Um mit den PAN-Koordinator Nachrichten austauschen zu können, greifen die Stationen auf das Übertragungsmedium unter Verwendung des slotted CSMA/CA-Mechanismus zu. Die Abbildung 2.4 verdeutlicht den allgemeinen Aufbau eines Superframes.

Abbildung 2.4: Allgemeiner Aufbau eines Superframes, abgezeichnet von [Gutierrez u. a., 2011, S. 84]

Auf Wunsch kann der PAN-Koordinator einer bestimmten Station feste Zeitschlitze zuweisen. Dieser Betriebsmodus (GTS) ist optional. Er ist für Anwendungsfälle gedacht, bei denen eine

bestimmte Bandbreite oder eine niedrige Latenz gefordert wird und kommt in der Stern-Topologie zum Einsatz.

Die Zeitschlitze werden rückwärts vom Ende des Superframes ausgehend den Endpunkten gruppenweise zugeordnet. In der Grafik 2.5 wurde diese Einteilung dokumentiert. Die Zeitspanne, die durch die GTSs abgedeckt wird, bezeichnet man als Contention-Free Period (CFP) oder auch als zugriffskonfliktfreie Periode. Die CFP kann einen erheblichen Teil des Superframes einnehmen. Der Standard 802.15.4 fordert allerdings, dass ein kleiner Bereich des Frames als CAP reserviert wird. Er ist für Geräte gedacht, die nicht die CFP nutzen. Es gibt nur eine Ausnahme von dieser Regel und das ist die kurzzeitige Reduzierung zwecks Unterbringung längerer Beacon-Signale für die Erhaltung der GTSs.

Das Verwenden von GTSs macht Protokolldienste auf höheren Schichten des OSI-Referenzmodells möglich, doch für Netzwerke bei denen solche Dienste nicht implementiert wurden bleibt die Nutzung von GTSs optional.

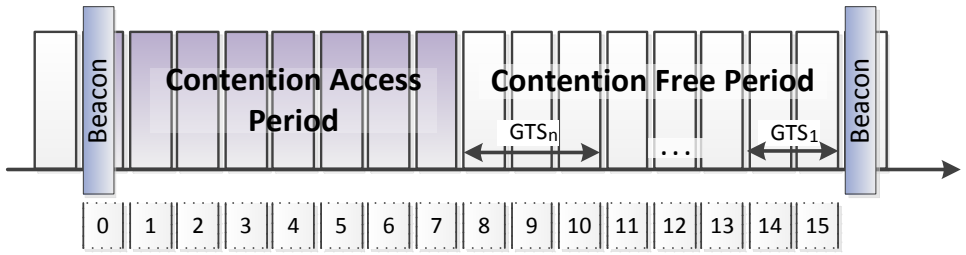


Abbildung 2.5: Aufbau eines Superframes mit GTS, abgezeichnet von [Gutierrez u. a., 2011, S. 85]

Für weniger kritische Anwendungen bezüglich Latenz kann das Superframe auch in einen aktiven und einen inaktiven Teil mit 16 zusammenhängenden Zeitschlitzen geteilt werden, wobei nur der aktive Teil für den Nachrichtentransport zwischen den Stationen genutzt wird. Die Abbildung 2.6 zeigt beispielhaft ein Superframe mit aktivem und inaktivem Teil gleicher Länge. Die beiden Teile müssen nicht zwingend gleich groß sein.

Die Länge des aktiven Teils des Superframes wird durch das Superframe-Intervall festgelegt, während die Frequenz des Beacon-Signales das Beacon-Intervall bestimmt. Dadurch zum Beispiel ist es Netzwerken, die mit batteriegespeisten Koordinatoren aufgebaut wurden, möglich, den Übertragungsauslastungsgrad zu reduzieren und die Lebensdauer der Batterie des Koordinators zu erhöhen.

Ein weiterer Anwendungsfall für diese Betriebsart findet sich in großen Multihop-Netzwerken mit aktiviertem Beacon-Signal. In diesem Fall wird die aktive Periode eines Netzwerkes so abgeglichen, dass sie in die inaktive Periode des Superframes des Eltern-Netzwerkes fällt. Der Eltern-Knoten kann der PAN-Koordinator oder ein weiterer Koordinator sein. Dadurch können mehrere Superframes ineinander geschachtelt werden, ohne sich gegenseitig zu stören. [Gutierrez u. a., 2011, S. 83-86]

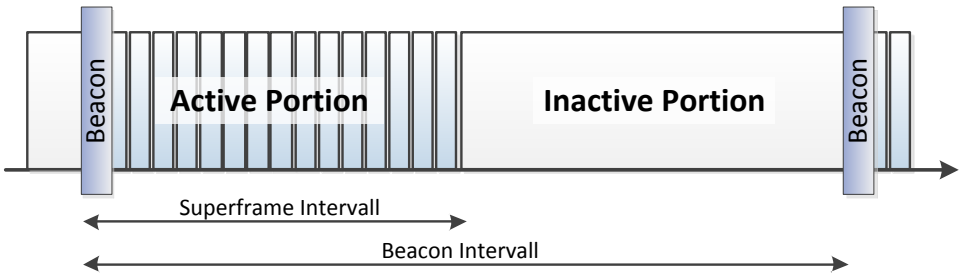


Abbildung 2.6: Superframe-Struktur mit aktivem und inaktivem Teil, abgezeichnet von [Gutierrez u. a., 2011, S. 86]

MAC-Layer Datentransfer

Die drahtlose Kommunikation über Funk stellt einige Herausforderungen an die Entwickler solcher Systeme. Nicht zuletzt handelt es sich dabei um ein Medium, das man sich mit anderen Diensten teilen muss. Der Standard IEEE 802.15.4 berücksichtigt dies und bietet Maßnahmen, um die Sicherheit des Nachrichtentransportes zu verbessern. Hier kommt ein Protokoll mit Bestätigung der Übertragung basierend auf dem CSMA/CA-Algorithmus und einer Rahmenintegritätsprüfung zum Einsatz.

Das Modell für die Datenübertragung ist von der Netzwerktopologie abhängig. Beim Stern-Netzwerk wird die Kommunikation immer über den PAN-Koordinator abgewickelt. In der Peer-to-Peer-Topologie darf eine Station mit jedem anderen Netzwerkgerät innerhalb seiner Reichweite kommunizieren.

Für Netzwerke mit Stern-Topologie existieren zwei Arten von Datentransfermechanismen, abhängig davon, ob der PAN-Koordinator Beacon-Signale verwendet oder nicht. Hierbei handelt es sich um den Datentransfer zu einem PAN-Koordinator oder den Datentransfer von einem PAN-Koordinator. In einem Peer-to-Peer-Netzwerk wird zusätzlich noch der Peer-to-Peer-Datentransfer unterstützt. [Gutierrez u. a., 2011, S. 87]

Datentransfer zu einem PAN-Koordinator

In einem Netzwerk mit aktiviertem Beacon-Signal hat sich die Station, die Daten an den PAN-Koordinator senden möchte, mit dem Beacon-Signal zu synchronisieren. Dieses Signal wird regelmäßig vom PAN-Koordinator gesendet. Falls die Station Teil eines Stern-Netzwerkes ist und ihm ein GTS zugewiesen wurde, wartet es den entsprechenden Zeitpunkt innerhalb des Superframes ab, bevor es seine Daten sendet.

Wird kein GTS verwendet, sendet die Station die Daten innerhalb der CAP des Superframes. Dabei wird das slotted CSMA/CA Verfahren eingesetzt. Nach dem Empfang des Datenrahmens kann der PAN-Koordinator auf Wunsch eine Bestätigung an die Station senden und den Datentransfer abschließen. In der Abbildung 2.7 wird der Vorgang noch einmal durch ein Sequenzdiagramm verdeutlicht.

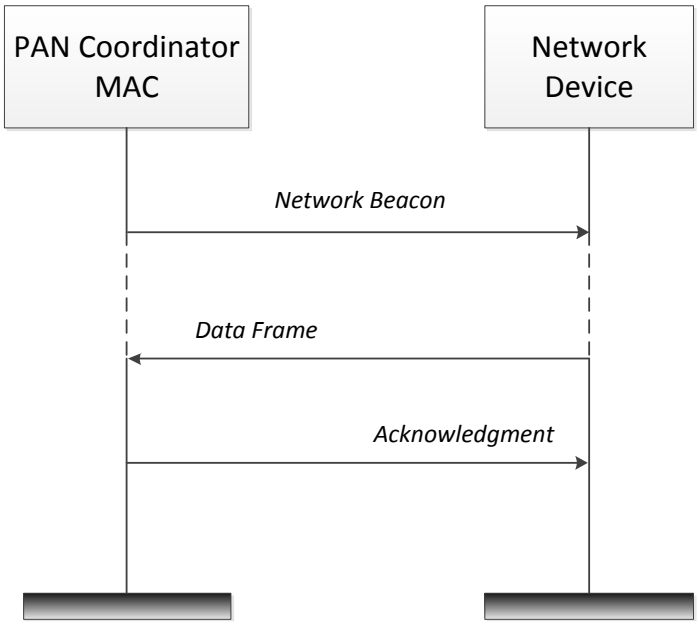


Abbildung 2.7: Datentransfer zu einem PAN-Koordinator in einem Netzwerk mit aktiviertem Beacon-Signal, abgezeichnet von [Gutierrez u. a., 2011, S. 88]

Eine Station, welche in einem Netzwerk mit nicht aktiviertem Beacon-Signal Daten an den PAN-Koordinator senden möchte, überprüft die Verfügbarkeit des Übertragungskanals unter Verwendung des Carrier Sense Multiple Access (CSMA)-Algorithmus. Wenn der Kanal verfügbar ist, sendet die Station die Daten an den PAN-Koordinator. Der PAN-Koordinator sendet, falls gefordert, eine Bestätigung an die Station und quittiert so den empfangenen Datenrahmen. Das Sequenzdiagramm zu diesem Vorgang zeigt die Abbildung 2.8. [Gutierrez u. a., 2011, S. 87-88]

Datentransfer von einem PAN-Koordinator

Wenn ein PAN-Koordinator Daten an eine Station senden möchte, überträgt er diese nicht sofort, sondern fügt stattdessen die Kurzadresse des Zielgerätes in ein bestimmtes Feld des Beacon-Signals ein. Dieses Feld wird auch *Pending Address List* genannt. Damit wird der Station signalisiert, dass Daten beim PAN-Koordinator zum Abrufen bereit liegen. Nachdem das entsprechende Gerät das Beacon-Signal empfangen hat und die Daten empfangen möchte, sendet es einen sogenannten *Data Request MAC Command*-Rahmen zum PAN-Koordinator. Dies geschieht in der CAP des Superframes. Nachdem der PAN-Koordinator den Befehlsrahmen empfangen hat, antwortet er mit einem Bestätigungsrahmen (Acknowledgment Frame) gefolgt von dem anstehenden Datenrahmen. Die Übertragung wird durch das Senden einer Bestätigung durch die Station als Quittung für den einwandfreien Empfang abgeschlossen. Sollten weitere Nachrichten für diese Station bereitliegen, wird dies wieder im Beacon-Signal angezeigt und die Prozedur beginnt erneut. Der Standard IEEE 802.15.4 spricht bei diesem Verfahren von einem indirekten Nachrichtentransfer.

Der von einem PAN-Koordinator ausgehende Vorgang des Datentransfers in einem Netzwerk mit aktiviertem Beacon-Signal wurde in der Abbildung 2.9 noch einmal in einem Sequenzdiagramm zusammengefasst.

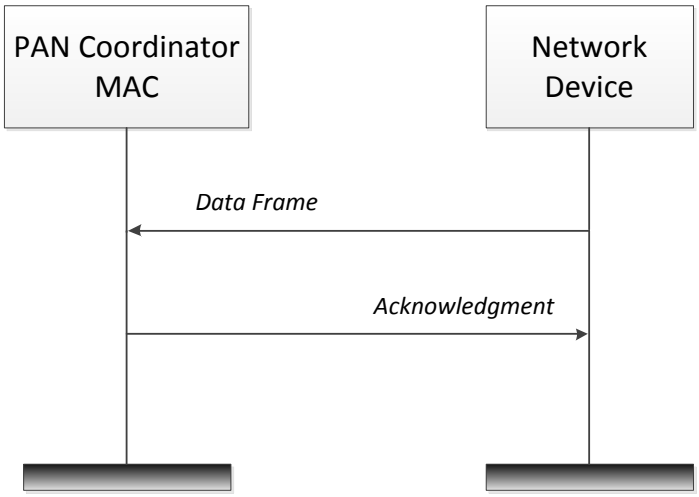


Abbildung 2.8: Datentransfer zu einem PAN-Koordinator in einem Netzwerk mit deaktiviertem Beacon-Signal, abgezeichnet von [Gutierrez u. a., 2011, S. 88]

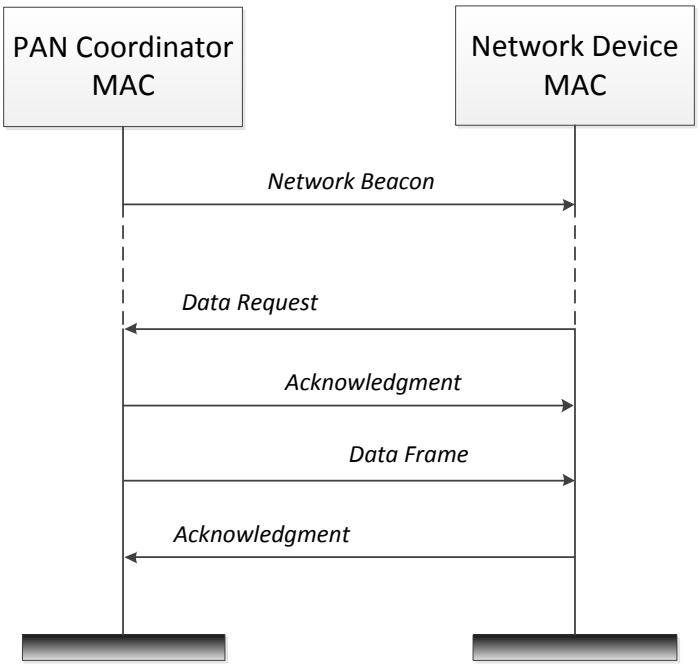


Abbildung 2.9: Datentransfer von einem PAN-Koordinator in einem Netzwerk mit aktiviertem Beacon-Signal, abgezeichnet von [Gutierrez u. a., 2011, S. 89]

Der indirekte Datentransfer wird vom PAN-Koordinator auch in Netzwerken mit deaktiviertem Beacon-Signal verwendet, um Daten an Stationen zu senden. Da es hier keinen Mechanismus von Seiten des PAN-Koordinator gibt, um das Vorhandensein von Nachrichten für die Stationen anzuzeigen, fragen diese in bestimmten Abständen beim PAN-Koordinator an. Hierbei wird von Polling gesprochen.

Die MAC-Unterschicht des Standards 802.15.4 bietet einen Mechanismus der höheren Schichten des OSI-Referenzmodells erlaubt, den PAN-Koordinator anzupollen. Wird der Polling-Vorgang angestoßen, sendet die MAC-Schicht der Station einen *Data Request MAC Command*-Rahmen zum PAN-Koordinator, der den Empfang mit einem Bestätigungsrahmen quittiert. Anschließend werden die anstehenden Daten vom PAN-Koordinator gesendet. Eine von der Station gesendete Bestätigung schließt den Datentransfer ab. Die Abbildung 2.10 dokumentiert den Vorgang anhand eines Sequenzdiagramms.[Gutierrez u. a., 2011, S. 88-90]

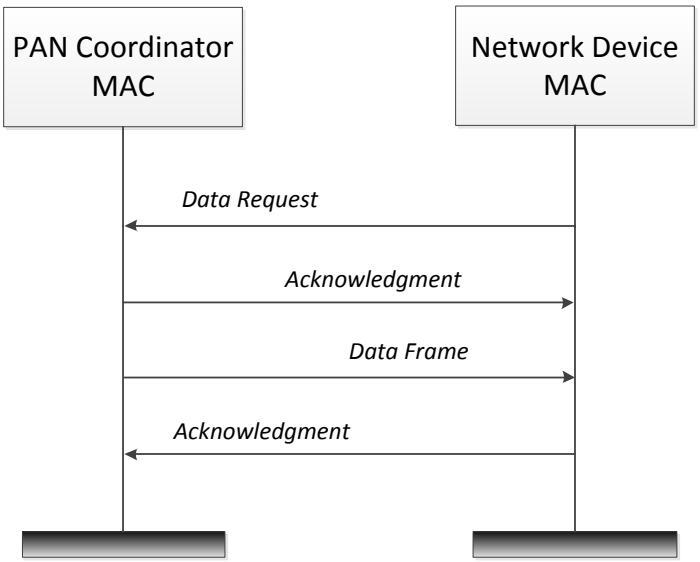


Abbildung 2.10: Datentransfer in einem Stern-Netzwerk mit deaktiviertem Beacon-Signal von einem PAN-Koordinator ausgehend, abgezeichnet von [Gutierrez u. a., 2011, S. 90]

Peer-to-Peer-Datentransfer

Für Peer-to-Peer-Topologien liegt die Datentransferstrategie in der Hand der entsprechenden Netzwerkschicht, die das drahtlose Netzwerk verwaltet. Eine Station kann sich hier im Empfangsmodus befinden oder die HF-Kanäle nach laufenden Übertragungen abhören. Alternativ dazu kann sie auch periodisch Beacon-Signale zwecks Synchronisation mit anderen Geräten senden, die den Übertragungskanal überwachen.

Dienste auf der MAC-Unterschicht

Die MAC-Unterschicht stellt zwei Dienste für höhere Schichten des OSI-Referenzmodells bereit. Ähnlich wie auf der Bitübertragungsschicht existieren Dienste für die Datenübertragung und für das Management. Der Managementdienst trägt auch die Bezeichnung MAC Sublayer Management Entity (MLME).

Auf den Datendienst wird über den MAC Common Part Sublayer Service Access Point (MCPS-SAP) und auf den Managementdienst über den MAC Sublayer Management Entity Service Access Point (MLME-SAP) zugegriffen. Durch den Standard 802.15.4 wird für jeden der Dienste ein Satz Protokolldienstelemente definiert, um sämtliche Funktionen eines LR-WPAN Gerätes bereitzustellen. [Gutierrez u. a., 2011, S. 90]

Übertragungsszenarien

Der Datenaustausch zwischen zwei Transceivern ist aufgrund der Natur des drahtlosen Übertragungsmediums fehleranfällig. Wenn in dem übertragenen Rahmen ein oder mehrere Fehler auftreten, erreicht die Nachricht nicht den vorgesehenen Empfänger, da die Bitübertragungsschicht den Rahmen verwirft oder sich gar nicht darauf synchronisieren kann. Bei letzterem Fall wird die Bitübertragungsschicht die Nachricht gar nicht erst empfangen.

Bei einem Datentransfer mit Bestätigung können drei mögliche Übertragungsszenarien auftreten:

- **Die Daten werden erfolgreich übertragen:** Der Absender sendet eine Nachricht zu einem Empfänger und diese wird fehlerfrei empfangen. Der Bestätigungsrahmen wird an den Absender innerhalb einer festgelegten Zeitspanne (timeout) erfolgreich übertragen (Abbildung 2.11).
- **Der Nachrichtenrahmen geht verloren:** Eine Nachricht für einen bestimmten Empfänger erreicht diesen nie. In diesem Fall wird die Nachricht nach Ablauf einer festgelegten Zeitspanne erneut gesendet. Nach einer bestimmten Anzahl von Versuchen signalisiert die MAC-Schicht der darüber liegenden Schicht, dass ein Fehler auftrat (Abbildung 2.12).
- **Der Bestätigungsrahmen geht verloren:** Der Absender einer Nachricht empfängt keinen Bestätigungsrahmen. Ähnlich wie in dem Beispiel des verlorenen Nachrichtenrahmens wird die Nachricht erneut gesendet und bei mehrmaligem Fehlschlag der darüber liegenden Schicht ein Übertragungsfehler signalisiert (Abbildung 2.14).

[Gutierrez u. a., 2011, S. 92]

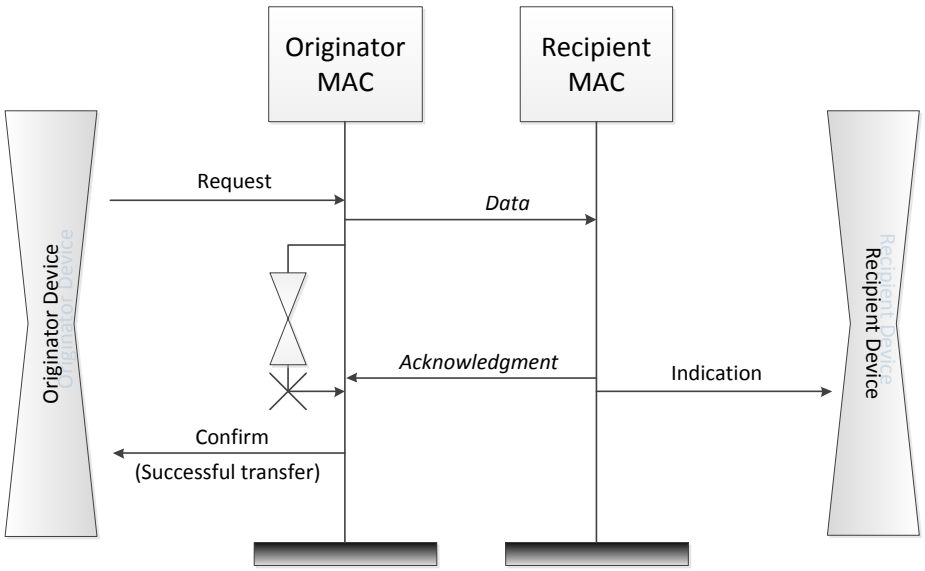


Abbildung 2.11: Übertragungsszenario: Daten werden erfolgreich übertragen, abgezeichnet von [Gutierrez u. a., 2011, S. 93]

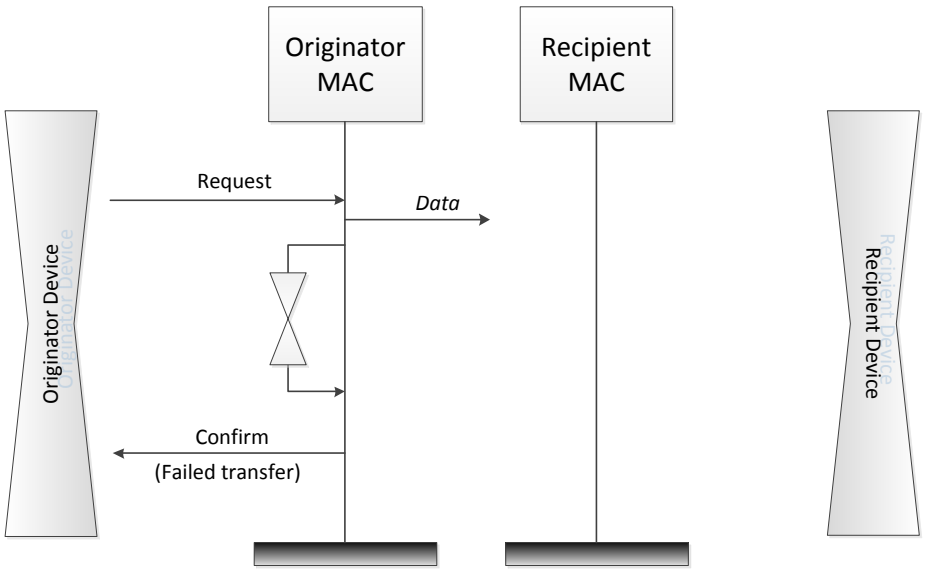


Abbildung 2.12: Übertragungsszenario: Nachrichtenrahmen geht verloren, abgezeichnet von [Gutierrez u. a., 2011, S. 93]

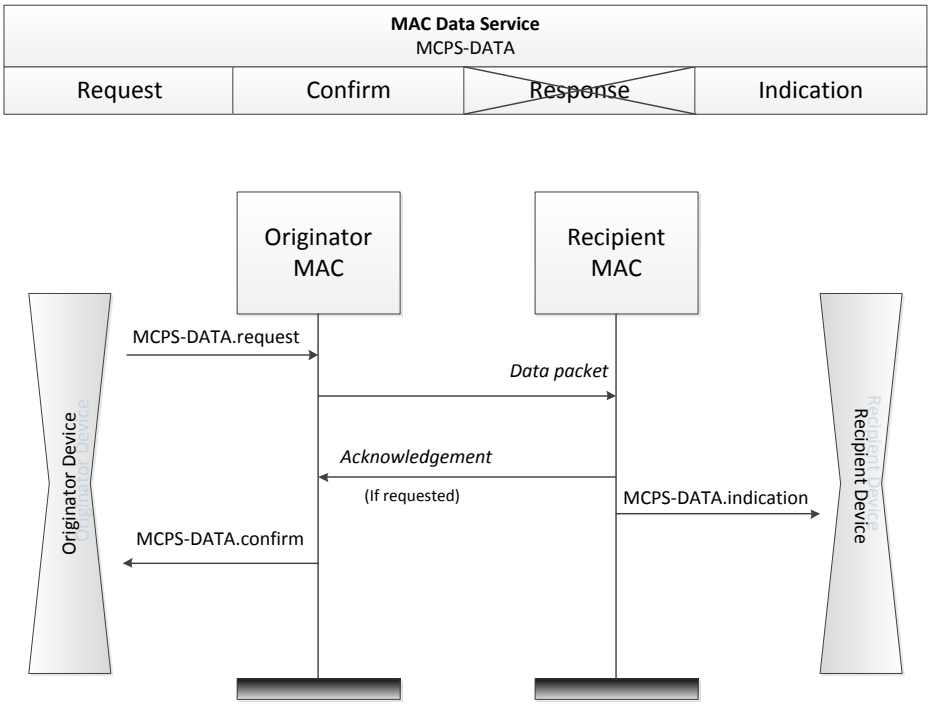


Abbildung 2.13: Sequenzdiagramm für den Mechanismus des Datenaustausches, abgezeichnet von [Gutierrez u. a., 2011, S. 94]

Dienst für den Nachrichtentransfer

Der Dienst für die Datenübertragung auf MAC-Ebene stellt drei Dienstelemente für die Übertragung von Nachrichten zur Verfügung: MCPS-DATA.request, MCPS-DATA.confirm und MCPS-DATA.indication. Die Abbildung 2.13 zeigt das Sequenzdiagramm eines typischen Datentransfers zwischen zwei Stationen. Der Dienst für die Datenübertragung verzichtet hierbei auf das Response-Dienstelement, da es nicht erforderlich ist.

Die Schnittstelle kann verschiedene Adressformate verarbeiten und ermöglicht so das leichte Implementieren der Stern- und Peer-to-Peer-Topologie. Die MAC-Schicht erlaubt den höheren Schichten des OSI-Referenzmodells die Art des übertragenen Adressfeldes in einem Datenpaket zu beeinflussen. So können verschiedenste Arten von Netzwerkschichten inklusive Multihop- und ad hoc-Netzwerken realisiert werden. Der Standard IEEE 802.15.4 verwendet die Standard EUI-64 Adresse des IEEE und eine Kurzadresse, die während des Anmeldevorgangs am Netzwerk vergeben wird. Zusätzlich unterstützt der Dienst zwei optionale Dienstelemente, die es den höheren Schichten gestattet, eine MAC Service Data Unit (MSDU) aus der MAC-Warteschlange zu entfernen. Diese Dienstelemente heißen MCPS-PURGE.request und MCPS-PURGE.confirm und werden für den indirekten Datentransfer verwendet. Das kann zum Beispiel nützlich sein, wenn der abzuholende Datenrahmen bei einem Koordinator ungültig wird, bevor ihn die betreffende Station abholen konnte. [Gutierrez u. a., 2011, S. 94]

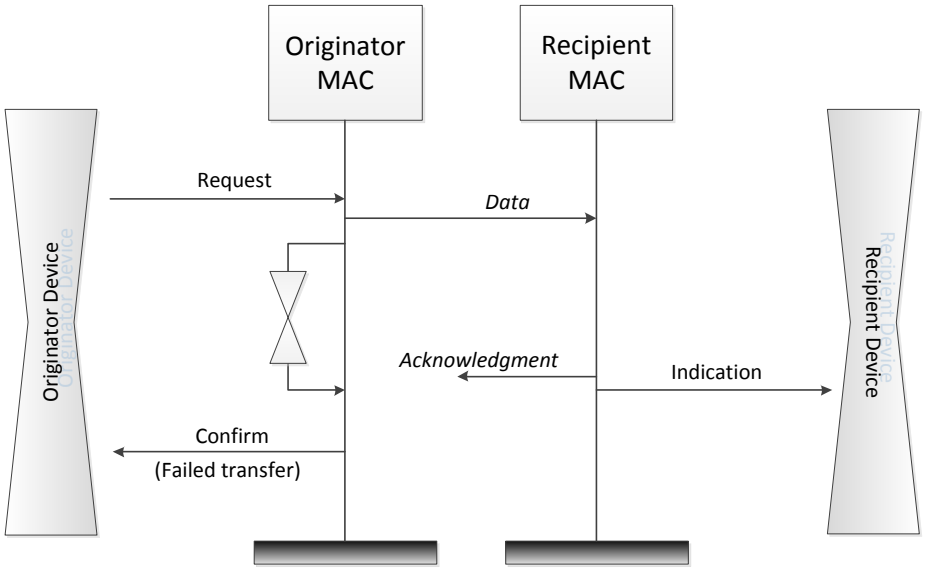


Abbildung 2.14: Übertragungsszenario: Bestätigungsrahmen geht verloren, abgezeichnet von [Gutierrez u. a., 2011, S. 93]

Der Dienst für das Management

Durch den Dienst für das Management auf MAC-Ebene können die Einstellungen für den Nachrichtenverkehr verändert werden. Darüber hinaus ist es möglich, auf das Sendeempfangsmodul zuzugreifen und die Netzwerkfunktionen zu nutzen. Sämtliche Dienstelemente der MLME wurden in der Tabelle 2.1 zusammenfassend aufgeführt. [Gutierrez u. a., 2011, S. 95]

Rahmenaufbau der MAC-Unterschicht

Die Rahmenstruktur der MAC-Schicht des Standards 802.15.4 spiegelt die Einfachheit und Flexibilität des Protokolls wieder. Mit möglichst wenigen Funktionen werden die Probleme, die bei einer drahtlosen Übertragung zu bewältigen sind, gelöst. Der MAC-Rahmen besteht aus drei Teilen: dem Header, dem Payload bzw. den Nutzdaten von variabler Länge und dem Footer.

Der MAC-Header enthält ein Feld zur Steuerung des Rahmens, ein Feld für die Sequenznummer, ein Adressfeld und optional ein zusätzliches Header-Feld für die Sicherheitsmechanismen. Das Rahmensteuerungsfeld beschreibt den Typ des Rahmens, den verwendeten Sicherheitsmechanismus und das Format sowie den Inhalt des Adressfeldes. Darüber hinaus wird hier angezeigt, ob eine Bestätigung durch den Empfänger gewünscht wird. Das Feld für die Sequenznummer enthält eine Zahl, die mit jedem übertragenen Rahmen erhöht wird. Im Adressfeld wird die Quell- oder Zieladresse, abhängig von den Angaben im Rahmensteuerungsfeld, gespeichert. Falls ein Sicherheitsmechanismus verwendet wird, wird diese Information mit dem Rahmensteuerungsfeld übertragen. Der Rahmen enthält dann zusätzlich den entsprechenden Header. Der Header für die Sicherheitsmechanismen ist von variabler Länge, abhängig von der verwendeten Sicherheitsstufe. Die Nutzdaten der MAC-Schicht enthalten spezifische Informationen, die von der Art der Nachricht abhängig sind und von den höheren Protokollschichten ausgewertet werden.

Dem Rahmen wird der sogenannte MAC-Footer angehängt, der aus einer 16 Bit langen Rahmen-Frame Check Sequence (FCS) besteht. Die FCS beruht auf dem standardisierten ITU-T 16 Bit Cyclic Redundancy Check (CRC)-Algorithmus, formal Comité Consultatif International Téléphonique et Télégraphique (CCITT) 16-bit CRC genannt. Der allgemeine Aufbau des MAC-Rahmens wird in Abbildung 2.15 dokumentiert. Wenn alle drei Komponenten des MAC-Rahmens zusammengefügt werden, wird von der MAC Protocol Data Unit (MPDU) gesprochen.

Der Standard 802.15.4 definiert vier Arten von MAC-Rahmen: MAC-Befehls-, Beacon-, Daten-, und Bestätigungsrahmen. [Gutierrez u. a., 2011, S. 110-111]

Tabelle 2.1: Die Dienstelemente des Management Dienstes der MAC-Schicht, entnommen [Gutierrez u. a., 2011, S. 95]

Primitive	**Category**	**Description**	Request	Confirm	Response	Indication
GET	Communication Settings	MAC PAN information base management	X	X		
SET			X	X		
RESET			X	X		
RX-ENABLE	Radio Control	Enables/Disables radio system	X	X		
SCAN		Scan radio channels	X	X		
ASSO CIATE	Networking	Association control with a network coordinator	X	X	X	X
DISAS SOCIATE			X	X		X
GTS		GTS Management	X	X		X
ORPHAN		Orphan device management			X	X
SYNC		Control of device syncronization with network coordinator	X			
SYNC-LOSS						X
START		Beacon Management	X	X		
BEACON-NOTIFY						X
POLL		Beaconless Syncronization	X	X		
COMM-STATUS		Communication Status				X

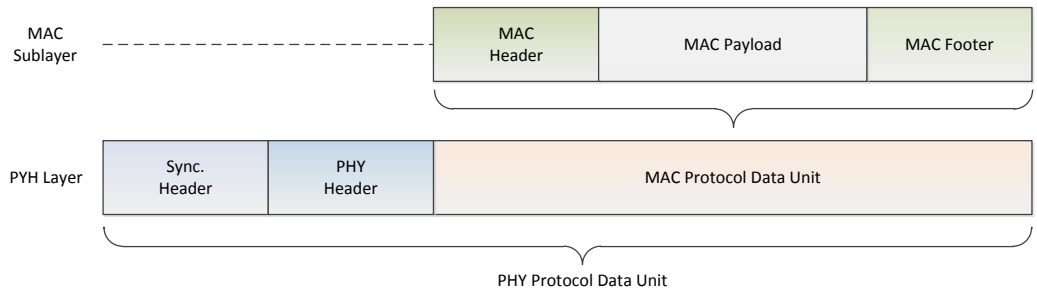


Abbildung 2.15: Allgemeiner Aufbau des MAC-Rahmens, abgezeichnet von [Gutierrez u. a., 2011, S. 110]

Beacon-Rahmen

In einem Netzwerk mit aktiviertem Beacon-Signal darf ein FFD Beacon-Rahmen senden. In einem Beacon-Rahmen sind in dem Adressfeld die PAN-ID und die Adresse der sendenden Station zu finden. Die Nutzdaten werden in vier Datenfelder unterteilt:

- **Superframe Specification Field:** Enthält Parameter, die ein vorhandenes Superframe beschreiben.
- **Pendig Address Specification Field:** Enthält die Anzahl und den Typ der Adressen, die im *Address List Field* aufgeführt werden.
- **Address List Field:** Enthält eine Liste sämtlicher Geräteadressen mit Daten, die beim PAN-Koordinator verfügbar sind.
- **Beacon Payload Field:** Dieses optionale Feld wird von den höheren Protokollschichten verwendet. Darüber können beispielsweise Daten an alle Stationen eines Netzwerkes verteilt werden.

Den Aufbau des Beacon-Rahmens zeigt die Abbildung 2.16. [Gutierrez u. a., 2011, S. 111]

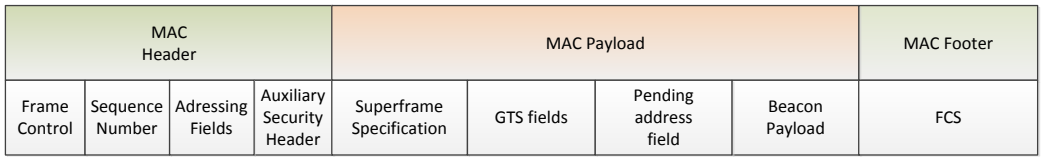


Abbildung 2.16: Format des Beacon-Rahmens, abgezeichnet von [Gutierrez u. a., 2011, S. 111]

Datenrahmen

Datenrahmen werden von der MAC-Unterschicht zum Übertragen der Nutzdaten für die höheren Protokollschichten verwendet. Das Adressfeld enthält hier die PAN-ID und die Geräte-ID von der Quell- und/oder Zielstation – so wie in dem MCPS-DATA.request-Dienstelement festgelegt. Die Abbildung 2.17 erläutert den Aufbau des Rahmens noch einmal graphisch. [Gutierrez u. a., 2011, S. 111]

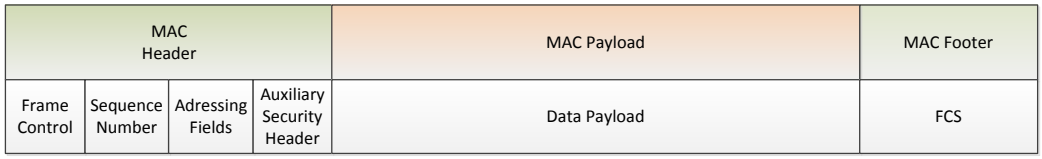


Abbildung 2.17: Format des Datenrahmens, abgezeichnet von [Gutierrez u. a., 2011, S. 112]

Bestätigungsrahmen

Bestätigungsrahmen werden von der MAC-Unterschicht zum Quittieren des erfolgreichen Empfangs eines Rahmens an den Sender einer Nachricht geschickt. Er wird nur auf ausdrücklichen Wunsch gesendet und setzt voraus, dass die FCS korrekt ist. Der Bestätigungsrahmen enthält kein Adressfeld im MAC-Header und es werden keine Nutzdaten übertragen. Dadurch kann der Netzwerkverkehr reduziert werden.

Wenn eine Station einen Bestätigungsrahmen empfängt, kontrolliert sie, ob sie eine Bestätigung erwartet. Wenn auch die empfangene Sequenznummer übereinstimmt, wird der Nachrichtentransfer abgeschlossen. Andernfalls wird der Rahmen verworfen. [Gutierrez u. a., 2011, S. 112]

Die Abbildung 2.18 veranschaulicht den Aufbau des Rahmens.

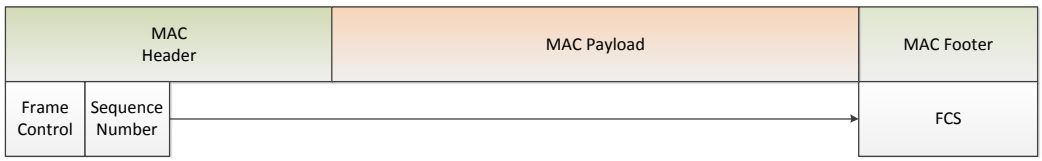


Abbildung 2.18: Format des Bestätigungsrahmens, abgezeichnet von [Gutierrez u. a., 2011, S. 112]

MAC-Befehlsrahmen

Der MAC Befehlsrahmen ist ein fester Bestandteil der MAC-Unterschicht und wird für sämtliche Aktivitäten aus Tabelle 2.2 eingesetzt. Die Nutzdaten der MAC-Schicht enthalten zwei Felder, eines welches den MAC-Befehl enthält und eines für zusätzliche Informationen. Diese Informationen sind von der Art des Befehls abhängig. Die Abbildung 2.19 veranschaulicht den MAC-Befehlsrahmen noch einmal. [Gutierrez u. a., 2011, S. 112-113]

Tabelle 2.2: MAC-Befehlsrahmen, entnommen [Gutierrez u. a., 2011, S. 113]

Command Identifier	**Command Type**
1	Association Request
2	Association Response
3	Disassociation Notification
4	Data Request
5	PAN ID Conflict Notification
6	Orphan Notification
7	Beacon Request
8	Coordinator Realignment
9	GTS Request
10-255	Reserved

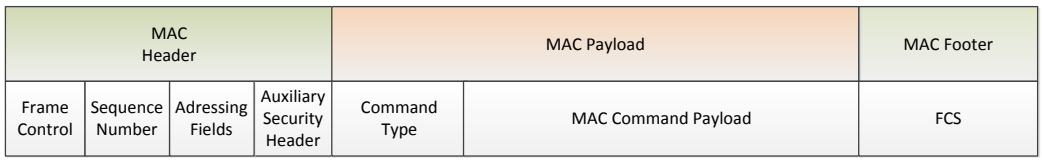


Abbildung 2.19: Format des Befehlsrahmens, abgezeichnet von [Gutierrez u. a., 2011, S. 113]

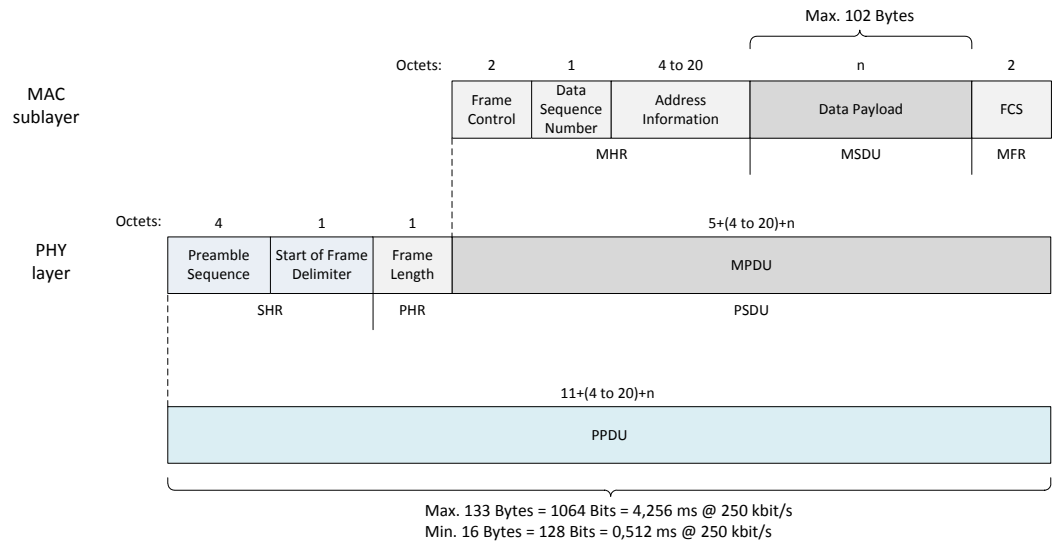


Abbildung 2.20: Datenrahmen des Standards IEEE 802.15.4 auf PHY- und MAC-Ebene, abgezeichnet von [Kupris u. Sikora, 2007, S. 66]

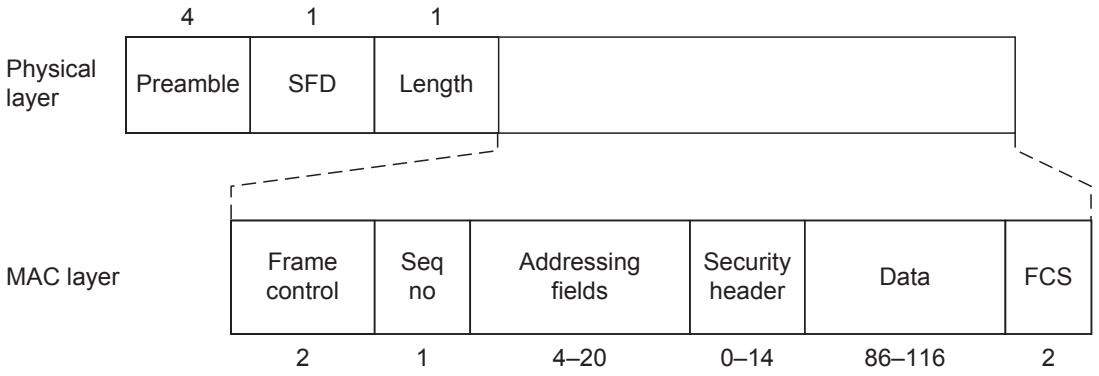


Abbildung 2.21: Die Headerformate der Bitübertragungs- und der MAC Schicht des Standards IEEE 802.15.4, entnommen aus [Vasseur u. Dunkels, 2010, S. 157]

2.2 Der Adaptionslayer 6LoWPAN

Das Protokoll IP wurde entwickelt, um die Netzwerkkommunikation über die Grenzen eines Netzwerkes hinaus zu ermöglichen. Das Internet setzt sich aus vielen Teilnetzen zusammen – eine bestimmte Anzahl dieser Netze muss ein Paket bei seinem Weg vom Sender zum Empfänger typischerweise durchlaufen. Für jede Art Netzwerk wird eine „IP-over-X"-Spezifikation benötigt, die beschreibt, wie ein IP-Paket in diesem Netz zu transportieren ist. Die Komplexität dieser Spezifikationen kann sich je nach Art des Netzwerkes unterscheiden. Ein Beispiel für eine relativ einfach gehaltene Spezifikation wäre „IPv6 over Ethernet" ([RFC2464] (Crawford)), die in etwas mehr als vier Seiten American Standard Code for Information Interchange (ASCII)-Text beschrieben wird.

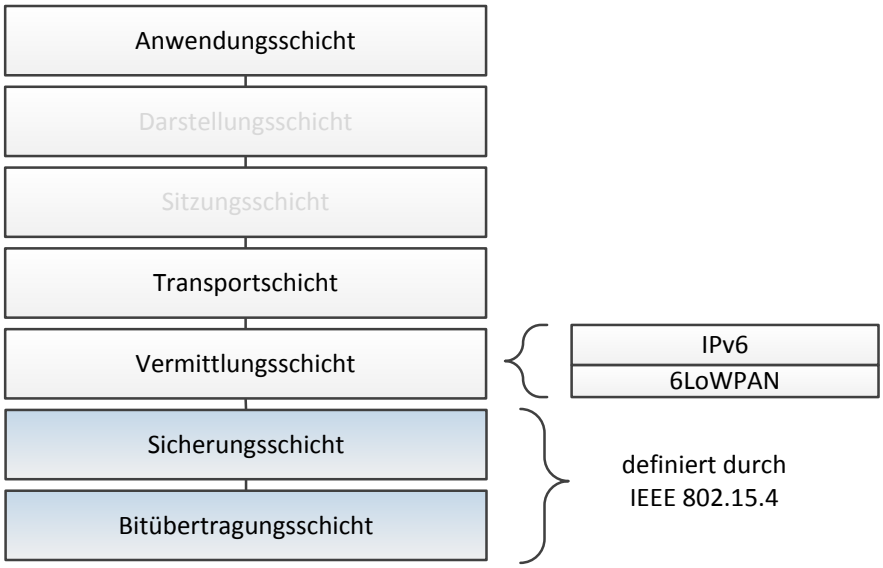


Abbildung 2.22: Einordnung des 6LoWPAN-Adaptionslayers in das OSI-Referenzmodell

Die 6LoWPAN-Spezifikation für die Übertragung des Protokolls IPv6 über ein Standard IEEE 802.15.4-Netzwerk ist etwas umfangreicher als das erwähnte Beispiel. Die Dienste des Standards IEEE 802.15.4 sind nicht so mächtig wie jene, die durch die Ethernet-Technologie bereitgestellt werden. Darüber hinaus führt die begrenzte Leistungsfähigkeit der Standards IEEE 802.15.4 und der Wunsch, möglichst wenig Energie für das Senden und Empfangen der Daten einzusetzen, zu strengeren Optimierungsanforderungen an diesen Adaptionslayer. [Shelby u. a., 2009, S. 27]

2.2.1 Aufgaben des Adaptionslayers

Ein „IP-over-X“-Adaptionslayer muss die IP-Datagramme an die Dienste anpassen, die von dem Teilnetz, üblicherweise auf Ebene 2 des OSI-Referenzmodells, bereitgestellt werden. Dabei gibt es verschiedene Probleme zu lösen:

- Es können Punkt-zu-Punkt-Verbindungen existieren oder solche, die mehrere IP-Knoten miteinander verbinden. Bei einer Punkt-zu-Punkt-Verbindung ist der Kommunikationspartner eindeutig. Wird ein Kommunikationsmedium von mehreren Stationen genutzt, wird für gewöhnlich eine Form der Ebene 2-Adressierung verwendet. Besonders in drahtlosen Netzen kann ein Paket von mehreren Stationen empfangen werden, auch von solchen, für die es nicht gedacht war. Die Ebene 2-Adressierung ist hier eine effiziente Möglichkeit, den Empfänger zu bestimmen. Nachdem die Netzwerkschicht die IP-Adresse des nächsten Hops bestimmt hat ist es die Aufgabe des Adaptionslayers, die Ebene 2-Adresse zu bestimmen, die das Paket näher an sein Ziel auf Ebene der Vermittlungsschicht heranbringt.
- Es kann passieren, dass das Teilnetz nicht sofort einen Weg zum nächsten IP-Knoten bereitstellen kann. Ein Beispiel hierfür wären verbindungsorientierte Netze wie Integrated Services Digital Network (ISDN) oder Asynchronous Transfer Mode (ATM), bei denen der Adaptionslayer zuerst eine Verbindung aufbauen müsste. Obwohl 6LoWPANs nicht verbindungsorientiert arbeiten, müsste der Adaptionslayer bei eingesetztem „Mesh-Under“-Routingverfahren den nächsten Ebene 2-Hop bestimmen und diese Station mit den nötigen Informationen versorgen, damit dieser das Paket weiterleiten kann. „Mesh-Under“ bezeichnet das Routing und Forwarding auf Ebene 2 des OSI-Referenzmodells und wird in Abschnitt 2.2.3 näher erläutert.
- Das IP-Paket muss so „eingepackt“/gekapselt werden, dass es durch das Teilnetz transportiert und durch den Ebene 2-Empfänger wiedergewonnen werden kann. Das führt zu weiteren Problemen:
  - Es muss möglich sein, über die Verbindung noch andere Pakettypen als IP-Datagramme zu übertragen. Das macht es nötig, zwischen verschiedenen Arten der Kapselung zu unterscheiden. Die meisten Ebene 2-Protokolle liefern Informationen über den als Payload enthaltenen Pakettyp (der 16-Bit-Ethertype bei Ethernet oder die PID (Protokoll-ID) beim Point-to-Point Protocol (PPP)) – der Standard IEEE 802.15.4 allerdings nicht. Der Standard 6LoWPAN nutzt einen eigenen Mechanismus, um die übertragenen Pakete zu identifizieren.
  - Es ist möglich, dass ein IP-Paket nicht in einen Ebene 2-Datenrahmen passt. Eine IP-Netzwerkschnittstelle wird durch die Paketgröße charakterisiert, die sie übertragen kann. Sie wird als MTU bezeichnet. Ethernet-Schnittstellen besitzen oft eine MTU von 1500 Bytes. Der Standard IPv6 definiert für die MTU einen minimalen Wert von 1280 Bytes. Der Adaptionslayer muss daher in der Lage sein, Paketgrößen von 1280 Bytes zu übertragen. Der Standard IEEE 802.15.4 kann auf Ebene der Sicherungsschicht nur Rahmen mit einer Größe von 127 Bytes übertragen. Um größere IPv6-Pakete transportieren zu können, muss der Adaptionslayer in der Lage sein, ein IP-Paket auf mehrere Ebene 2-Rahmen aufzuteilen und am Zielknoten wieder zusammenzufügen, um das IP-Paket zu rekonstruieren. Der Standard 6LoWPAN bezeichnet diesen Prozess in Anlehnung an den IP-Layer-Fragmentierungsprozess *Fragmentation* und *Reassembly*.
  - Das Protokoll IP wurde so entwickelt, dass jedes Paket unabhängig von anderen Paketen transportiert werden kann. Das führte dazu, dass der IP-Header viele Informationen enthält, die eigentlich aus dem Zusammenhang ermittelt werden könnten. In einem 6LoWPAN würde die typische IP/UDP-Headergröße von 48 Bytes einen Großteil des durch den Standard IEEE 802.15.4 bereitgestellten Platzes für die Nutzdaten

verbrauchen. Der Standard 6LoWPAN setzt deshalb eigene Kompressionsverfahren ein, um die Größe der Header zu reduzieren.

[Shelby u. a., 2009, S. 28-29]

2.2.2 Grundlegender Aufbau des 6LoWPAN-Formates

Das Format des Standards IEEE 802.15.4 enthält keine Felder für die Identifikation des Payloads und der Daten die darin enthalten sind. Es gibt keine Multiplexing-Information, die es dem Empfänger ermöglicht, die 6LoWPAN-Pakete von den anderen Datenpaketen zu unterscheiden. Es ist auch nicht möglich, die Art des gesendeten 6LoWPAN-Paketes zu ermitteln. Daher ist es eine der Aufgaben des 6LoWPAN-Kapselungs-Formates, einen zusätzlichen Pakettyp-Identifier bereitzustellen. 6LoWPAN entwickelte seine Typ-Identifier nicht auf Grundlage existierender Entwürfe wie dem 2-Byte IEEE 802.3 Ethertype oder dem 8 Byte Subnetwork Access Protocol (SNAP)-Header, die oft von IEEE 802.2-basierten Kapselungen genutzt werden ([RFC1042] (Postel u. Reynolds)). Diese Typ-Identifier bieten eine sehr gute langfristige Erweiterbarkeit. Sie sind jedoch recht verschwenderisch bezüglich des Overheads. Bei relativ kleinen Paketen, wie sie durch den Standard IEEE 802.15.4 vorgesehen sind, führt dies zu Problemen. Hier wird stattdessen das erste Byte des Payloads als sogenanntes *Dispatch Byte* verwendet. Dadurch ist es möglich, einen Typ-Identifier zu realisieren und man ist in der Lage, mögliche zukünftige Informationen zu kodieren.

Tabelle 2.3: Bedeutung der zwei höchstwertigsten Bits im Dispatch Byte, Inhalt entnommen aus [Shelby u. a., 2009, S. 33]

Bits	**Bedeutung**
00	Not a LoWPAN packet (NALP)
01	Normal dispatch
10	Mesh header
11	Fragmentation header

Das *Dispatch Byte* kann 256 verschiedene Werte annehmen. Davon werden die 64 Werte, bei denen die 2 höherwertigsten Bits auf null gesetzt sind, für den Standard IEEE 802.15.4 reserviert. Sie stehen dem 6LoWPAN-Standard also nicht zur Verfügung. Die anderen 192 Werte (3 x 64) wurden grob in 3 Klassen eingeteilt, die auch durch die 2 höherwertigsten Bits bestimmt werden. Diese Zuordungen wurden in der Tabelle 2.3 dokumentiert. Das ist eine vorläufige Einteilung. Mit künftigen Entwicklungen des 6LoWPAN können den ungenutzten Bereichen nach und nach weitere Funktionen zugeordnet werden. Die Zuweisungen für das *Dispatch Byte* werden von der Internet Assigned Numbers Authority (IANA) verwaltet (Internetlink: IANA).

Einer der Werte (01000001_2) ist für den Transport unveränderter IPv6-Pakete, wie in Abbildung 2.23 vorgeschlagen, gedacht. Das spiegelt den allgemeinen Fall im Ethernet wieder, der für diesen Zweck den Ethertyp 0x86DD bereithält. In einem 6LoWPAN ist es jedoch sehr viel wahrscheinlicher, dass eine sendende Station eine Form der Headerkompression und/oder einige der anderen Funktionen des Standards verwendet. Hier wurde die bisherige 32-Bit/64-Bit-Einteilung des IPv6 verworfen. Dabei wurde dem Einsatz von Mikrokontrollern mit reduzierten Ressourcen Rechnung getragen.

Zwei weiteren Werten innerhalb des 01-Raumes wurden bereits Funktionen zugewiesen. Die Folge 01010000_2 entspricht dem LOWPAN_BC0 und enthält eine Sequenznummer, um Duplikate von

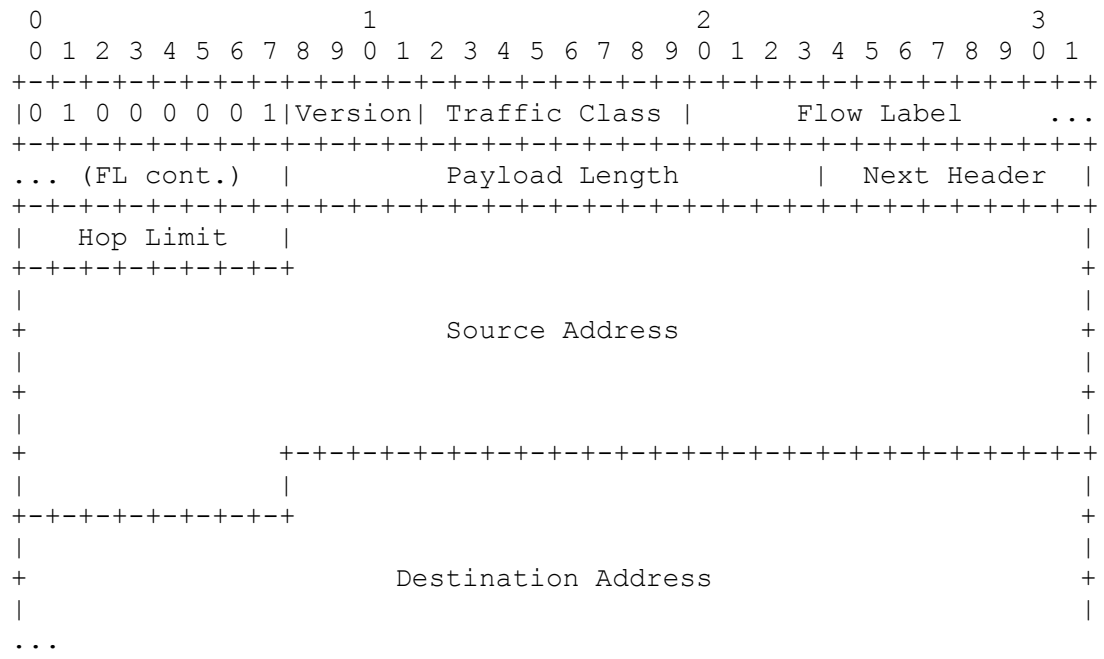


Abbildung 2.23: Nicht komprimiertes IPv6-Paket mit 6LoWPAN-Header, entnommen aus [Shelby u. a., 2009, S. 33]

Paketen bei *flodding*-basierten Broadcasting-Mechanismen detektieren zu können. 01111111_2=ESC ist reserviert für das Ausweiten des Wertebereichs des *Dispatch Bytes* über 8 Bit hinaus.

Einige der durch den Standard 6LoWPAN definierten Formate wurden entworfen, um weitere 6LoWPAN-Protocol Data Unit (PDU)s als Payload übertragen zu können. Falls mehrere Header vorhanden sein müssen, stellt sich die Frage, in welcher Reihenfolge diese übertragen werden sollen. Um dies praktikabel zu machen, definiert 6LoWPAN eine wohlgeordnete Aufeinanderfolge. Wenn vorhanden, sollten die 6LoWPAN-Header in der folgenden Reihenfolge übertragen werden:

- **Adressierung:** Der Mesh-Header ($10nnnnnn_2$) verweist auf die Layer 2 Original Quell- und finale Zieladresse und einen Hop-Counter, gefolgt von einer 6LoWPAN-PDU.
- **Hop-by-Hop Verarbeitung:** Header die in erster Linie Layer 2 Hop-by-Hop-Optionen wie den Broadcast-Header bereitstellen. Der Broadcast-Header (LOWPAN_BC0, 01010000_2) enthält eine Sequenznummer, die mit jedem weiterleitenden Hop überprüft wird, gefolgt von einer 6LoWPAN-PDU.
- **Zielknoten Verarbeitung:** Der Fragmentierungs-Header ($11nnnnnn_2$) beschreibt die Fragmente, die, nach einem Transport über möglicherweise mehrere Schicht 2-Hops, am Zielknoten wieder zu einer 6LoWPAN-PDU zusammen gefügt werden.
- **Payload:** Die Header verweisen auf Pakete der Netzwerkschicht, wie bspw. IPv6 ($0100-0001_2$), LOW-PAN_HC1 (01000010_2), oder LOWPAN_IPHC ($011nnnnn_2$).

Dabei handelt es sich um die gleiche Reihenfolge, in welcher die entsprechenden Header auf IPv6-Ebene angeordnet werden müssen. Der entscheidende Unterschied ist, dass IPv6 ein *Next-Header*-Feld besitzt, welches die Art der folgenden PDUs identifiziert, während der Standard 6LoWPAN ein Dispatch Byte am Anfang einer jeden PDU für die Identifizierung verwendet. Ursprünglich rührt diese Lösung von den fehlenden Multiplexinformationen der IEEE 802.15.4 MAC-Schicht

her. Man dachte dabei auch an eine einfachere Art der Implementierung. Die Tabelle 2.4 fasst die aktuellen und beabsichtigten Zuweisungen durch das Dispatch Byte noch einmal zusammen. Die Abbildung 2.24 veranschaulicht den Aufbau des 6LoWPAN-Protokollstacks graphisch. [Shelby u. a., 2009, S. 32-34]

Tabelle 2.4: Aktuelle und geplante Zuweisungen für das *Dispatch Byte*, Inhalt entnommen aus [Shelby u. a., 2009, S. 35], [Kushalnagar u. a., S. 8]

Pattern	**Header Type**
00 $xxxxxx_2$	NALP - Not a LoWPAN frame
01 000001_2	IPv6 - Uncompressed IPv6 Addresses
01 000010_2	LOWPAN_HC1 - LOWPAN_HC1 compressed IPv6
01 000011_2	reserved - Reserved for future use
...	reserved - Reserved for future use
01 001111_2	reserved - Reserved for future use
01 010000_2	LOWPAN_BC0 - LOWPAN_BC0 broadcast
01 010001_2	reserved - Reserved for future use
...	reserved - Reserved for future use
01 111110_2	reserved - Reserved for future use
01 111111_2	ESC - Additional Dispatch byte follows
10 $xxxxxx_2$	MESH - Mesh Header
11 $000xxx_2$	FRAG1 - Fragmentation Header (first)
11 001000_2	reserved - Reserved for future use
...	reserved - Reserved for future use
11 011111_2	reserved - Reserved for future use
11 $100xxx_2$	FRAGN - Fragmentation Header (subsequent)
11 101000_2	reserved - Reserved for future use
...	reserved - Reserved for future use
11 111111_2	reserved - Reserved for future use

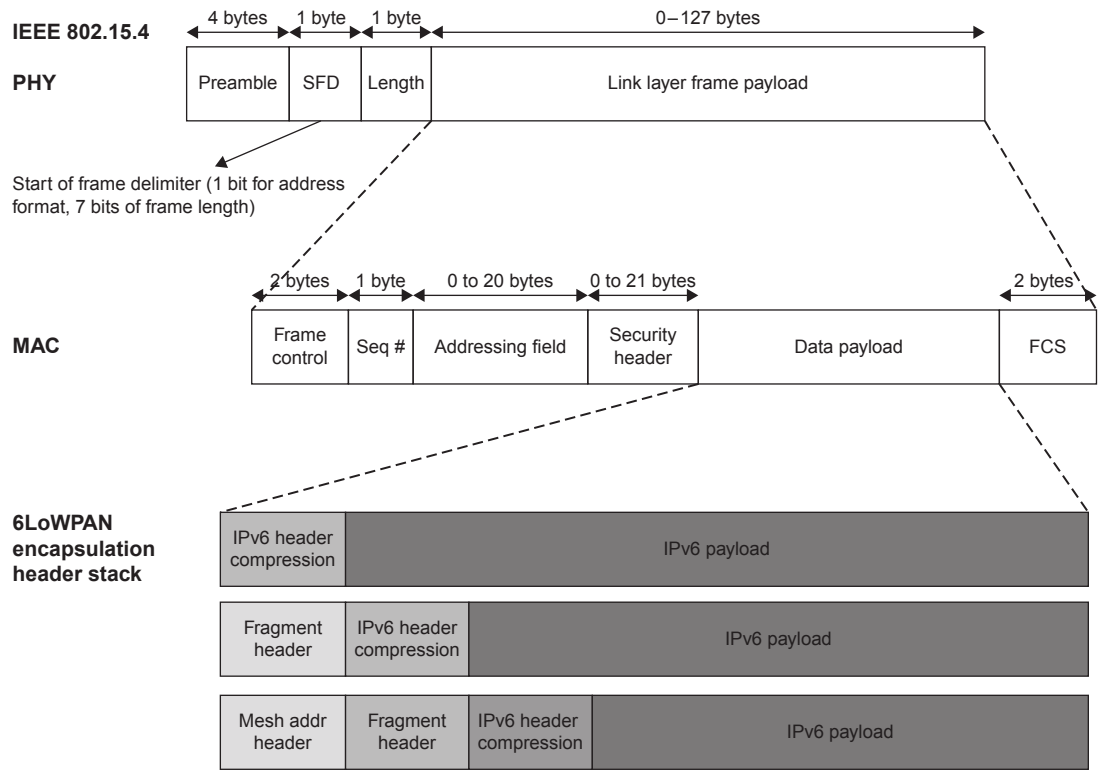


Abbildung 2.24: Der 6LoWPAN-Protokollstack, entnommen aus [Vasseur u. Dunkels, 2010, S. 233]

2.2.3 Routing und Forwarding

Pakete passieren auf ihrem Weg durch das LoWPAN oft mehrere Stationen. Bei diesem Vorgang sind die zwei Prozesse Forwarding und Routing beteiligt. Beide Prozesse können auf Schicht 2 oder Schicht 3 des OSI-Referenzmodells realisiert werden. Für das Routing wird üblicherweise mindestens ein Routingprotokoll eingesetzt. In jeder Station wird durch das Routingprotokoll eine Routing Information Base (RIB) in Gestalt einer Tabelle oder Datenbank geführt. Die RIB enthält alle benötigten Informationen für den Einsatz des Routingprotokolls. Die RIB kann üblicherweise zu einer Forwarding Information Base (FIB) vereinfacht werden. Die FIB wird herangezogen wenn ein Paket empfangen und weitergeleitet werden muss. Manche Routingprotokolle füllen die FIB proactiv/vorausschauend. Bei diesem Beispiel hält die FIB für jedes möglicherweise auftretende Paket einen Eintrag für das Weiterleiten bereit. Andere Routingprotokolle arbeiten reaktiv und fügen den Eintrag in die FIB nur hinzu, wenn das entsprechende Paket empfangen wurde.

Die Abbildung 2.25 zeigt eine übliche Darstellung des Routings auf Ebene der Netzwerkschicht. Pakete werden über eine beliebige Verbindung übertragen und erreichen den Router auf einer Schnittstelle, in diesem Beispiel das Interface *if0*. Der Router schlägt die Zieladresse in seiner FIB nach und wählt dann die passende Schnittstelle und leitet das Paket darüber weiter - in dem Beispiel das Interface *if1*. Das gekapselte Paket wird mit der Zieladresse auf Ebene 2 des OSI-Referenzmodells übertragen.

Die treibende Kraft für das Forwarding in einem LoWPAN ist die Tatsache, dass sich der Zielknoten außerhalb der Reichweite der sendenden Station befindet. Bei den Knoten, welche die Aufgabe des Routing übernehmen, ist die Schnittstelle die das Paket empfängt in der Regel die gleiche, über die das Paket zur nächsten Station gesendet wird. Diese besondere Art der Nachrichtenweiterleitung wird manchmal auch *Router on a Stick* genannt. Dieses Beispiel zeigt die Abbildung 2.26

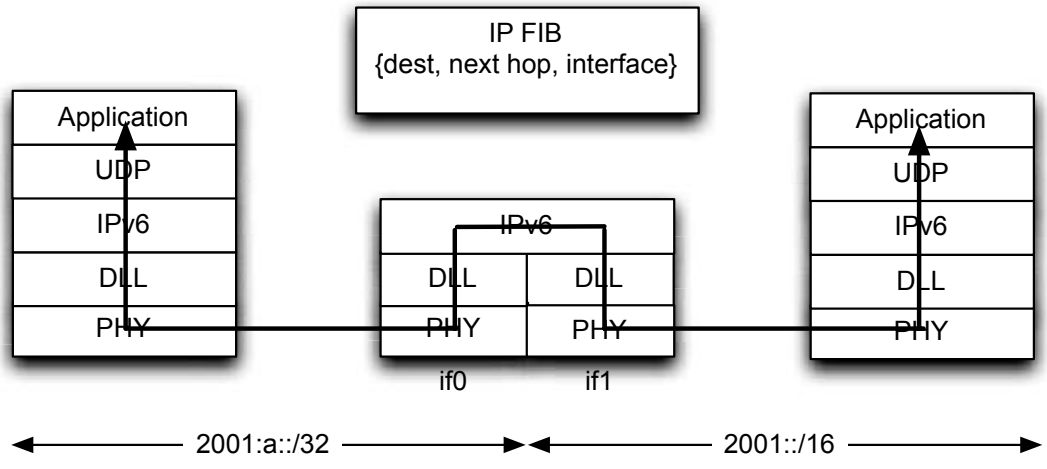


Abbildung 2.25: Das IP-Routingmodell, entnommen aus [Shelby u. a., 2009, S. 38]

Die Abbildungen 2.25 und 2.26 zeigen das Routing auf IP-Ebene. Routing und Forwarding in einem LoWPAN kann auf IP-Ebene aber auch auf der Schicht darunter erfolgen. [Shelby u. a., 2009, S. 37-38]

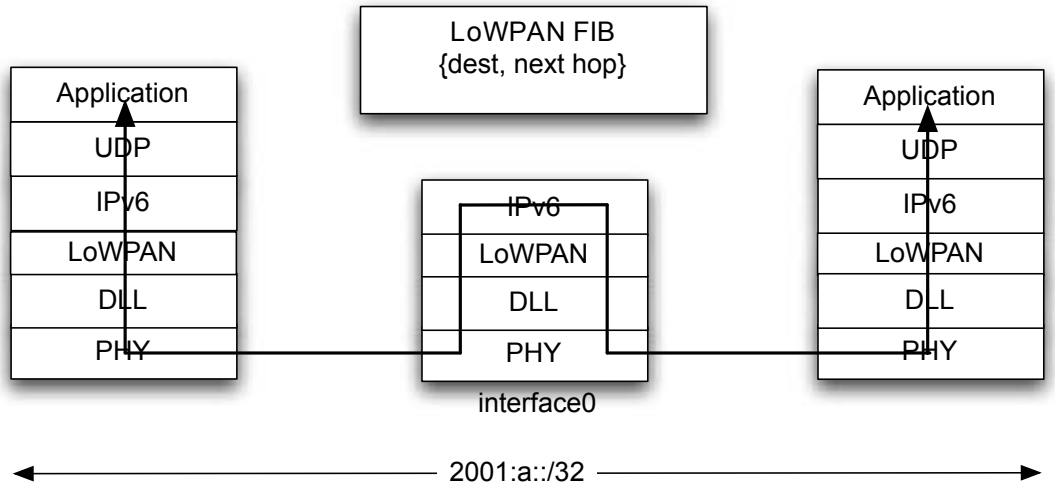


Abbildung 2.26: Das LoWPAN-Routingmodell, entnommen aus [Shelby u. a., 2009, S. 38]

Forwarding auf Schicht 2 des OSI-Referenzmodells („Mesh-Under“)

Wenn das Routing und Forwarding auf Schicht 2 umgesetzt wird, basiert es auf den Adressformaten der Sicherungsschicht wie dem EUI-64 oder den 16 Bit Kurzadressen. Die IETF arbeitet für gewöhnlich nicht mit Routingprotokollen auf Schicht 2 (Mesh Routing). ISA 100 definiert jedoch ein solches Protokoll. Gemeinsam mit einigen Erweiterungen des Data Link Layer (DLL) ist das Routing und Forwarding auf der Sicherungsschicht für den LoWPAN-Adaptionslayer grundsätzlich unsichtbar. Dies verdeutlicht die Abbildung 2.27. Die Situation ist ähnlich der, wenn ein IEEE-definiertes Mesh-Protokoll für den Standard IEEE 802.15.4 benutzt wird (so wie in IEEE 802.15.5 vorgeschlagen).

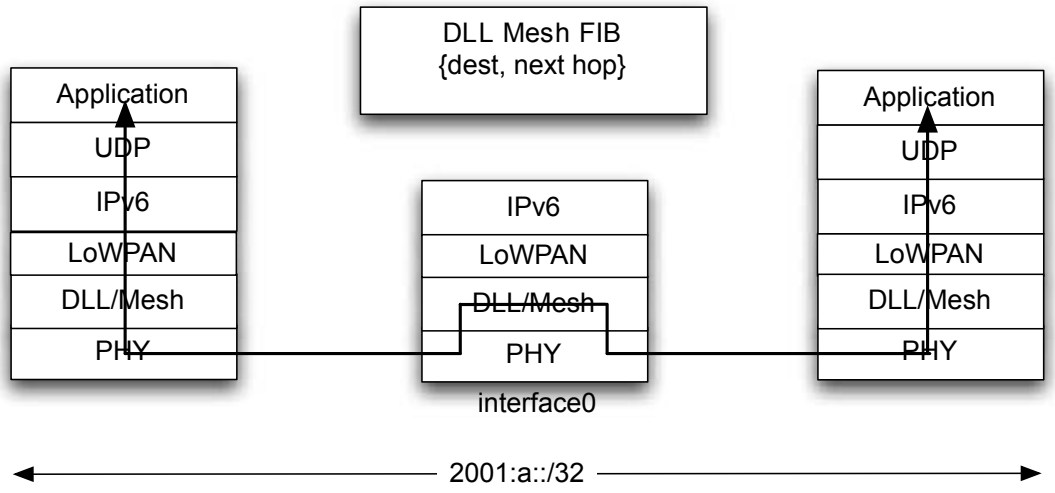


Abbildung 2.27: Mesh-Forwarding auf Ebene der Sicherungsschicht unter dem LoWPAN-Adaptionslayer, entnommen aus [Shelby u. a., 2009, S. 39]

In dem Fall, dass das Forwarding auf Ebene 2 vor dem LoWPAN-Adaptionslayer nicht verborgen wird, gibt es ein Problem zu lösen (siehe Abbildung 2.28). Der Header auf Sicherungsschicht beschreibt die Quell-und die Zieladdresse für den aktuellen Hop. Um das Paket zu seinem eigentlichen Ziel transportieren zu können, muss die Station die endgültige Ebene2-Zieladresse kennen. Darüber hinaus muss für Dienste wie bspw. das Reassembly die Ebene2-Adresse der sendenden Station bekannt sein. Da mit jedem Forwarding-Schritt die Zieladresse auf Sicherungsschichtebene durch die Adresse des nächsten Hops und die Adresse der sendenden Station durch die Adresse des weiterleitenden Knotens ersetzt wird, muss diese Information festgehalten werden.

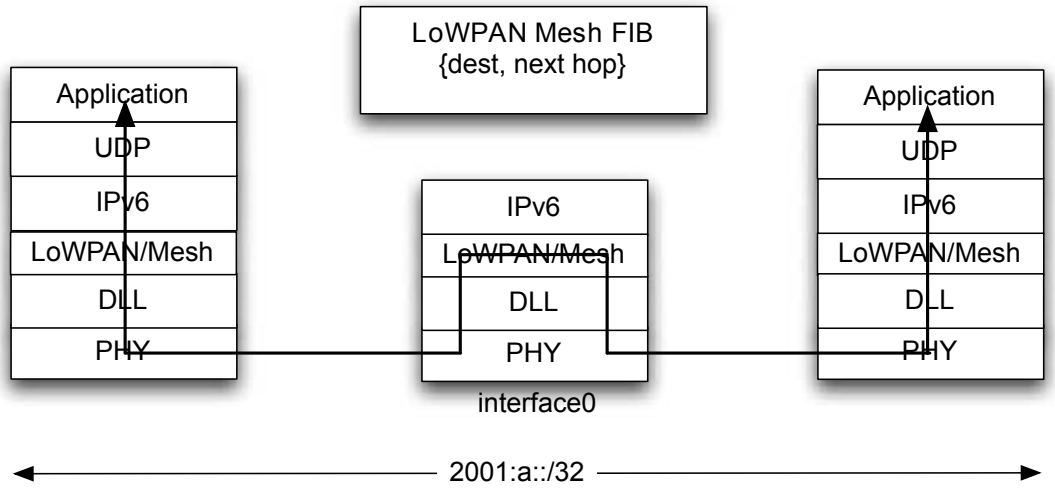


Abbildung 2.28: LoWPAN-Adaptionslayer-Mesh-Forwarding, entnommen aus [Shelby u. a., 2009, S. 39]

6LoWPAN definiert dafür den Mesh-Header. Zusätzlich enthält der Mesh-Header das Äquivalent des IPv6-Hop-Limits. Da die Durchmesser sinnvoller drahtloser Multihop-Netzwerke für gewöhnlich klein sind, wurde das Format der *Hops Left* durch das Zuweisen der unteren 4 Bits im *Dispatch Byte* für Werte unter 15 optimiert (erster Fall in der Abbildung 2.29). Werden Zahlenwerte grösser oder gleich 15 benötigt, fügt der 6LoWPAN-Encoder ein Erweiterungsbyte (*Deep Hops Left*) ein und markiert dies durch Setzen der entsprechenden 4 Bits im *Dispatch Byte* auf den Wert 0xF (zweiter Fall in der Abbildung 2.29).

Der Wert muss durch die weiterleitende Station erniedrigt werden, bevor das Paket zum nächsten Hop gesendet wird. Erreicht dieser Wert die Zahl Null, so wird das Paket stillschweigend verworfen. Die Routenverfolgungs-Funktion, eines der Erfolgsfaktoren des IP-Networking, kann für 6LoWPAN-Mesh-Under-Netzwerke nicht implementiert werden. In einer Implementierung muss das Erweiterungsbyte durch den Dekrementierungsprozess nicht entfernt werden, wenn der *Hops Left*-Wert auf 15 oder darunter fällt. Das Paket kann so wie es ist weitergesendet werden. Alternativ kann hier optimiert werden, indem dieses Byte entfernt und der Wert in das *Dispatch Byte* geschrieben wird.

Die Bits, die mit V und F gekennzeichnet wurden, zeigen an, ob es sich bei der Senderadresse und der Zieladresse um 16 Bit-Kurzadressen (1) oder um einen EUI-64 (0) handelt. Der Mesh-Header, sofern er eingesetzt wird, muss der erste Header in einem 6LoWPAN-Headerstack sein, da er die Adressen für die Weiterleitung enthält. Der nächste Header kann ein LOWPAN_BC0-Header für Multicasting sein (Siehe Abbildung 2.28). Weitere Header im Stack könnten Fragmentierungs-

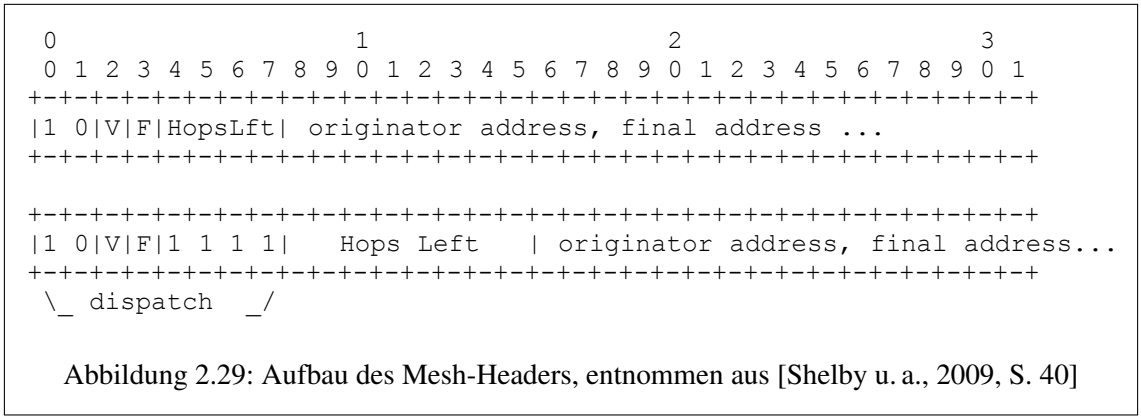


Abbildung 2.29: Aufbau des Mesh-Headers, entnommen aus [Shelby u. a., 2009, S. 40]

header sein, die während des Mesh-Forwarding-Vorganges nicht verändert werden. Die Fragmente werden unabhängig voneinander weitergeleitet. [Shelby u. a., 2009, S. 38-40] [Kushalnagar u. a., S. 9]

Ebene 3-Routing („Route-Over“)

Das Route-Over-Forwarding auf Ebene 3 des OSI-Referenzmodells ist in Abbildung 2.26 dargestellt. Im Gegensatz zum Mesh-Forwarding auf Ebene der Sicherungsschicht werden hier keine besonderen Anforderungen an den Adaptionslayer gestellt. Bevor die Forwarding-Implementierung der Netzwerkschicht das Paket sieht, wurde es durch den Adaptionslayer „entkapselt“ – wenigstens konzeptionell. Die Implementierungen können in der Lage sein, Optimierungen anzuwenden indem sie die gekapselte Form beibehalten, wenn sie wissen, wie sie das Paket für den nächsten Ebene 3 Hop zu bearbeiten haben. Hierbei gilt zu bedenken, dass der Schritt der Fragmentierung und Defragmentierung für jeden Hop beim Route-Over-Forwarding durchgeführt werden muss – auch wenn das schwer vorstellbar ist, da die Adressen der Netzwerkschicht Teil der ersten Bytes des IPv6-Headers sind, die nur im ersten Fragment eines größeren Paketes auftreten. Hier können mögliche Implementierungen wieder Optimierungen anwenden, indem sie einen virtuellen Reassambly-Buffer führen und sich die IPv6-Header inklusive der relevanten Adressen (und den Inhalt jedes ankommenden Fragments unabhängig von der logischen Reihenfolge) merken. [Shelby u. a., 2009, S. 40-41]

2.2.4 Headerkomprimierung

Zustandslose Headerkomprimierung

Die 6LoWPAN-Spezifikation definiert zwei Headerkompressionsentwürfe die gemeinsam eingesetzt werden können. HC1 um IPv6-Header zu komprimieren und analog dazu HC2 für UDP-Header. HC2 kann also in Paketen eingesetzt werden, in denen auch das Verfahren HC1 verwendet wird. Die Auswahl von HC1 erfolgt über das Dispatch-Byte LOWPAN_HC1 (01000010_2, siehe Tabelle 2.4). Das nächste Byte kodiert die möglichen Kompressionsoptionen. Das gesetzte Bit 7 signalisiert hierbei, dass auch das Kompressionsverfahren HC2 eingesetzt wird. Die Abbildung 2.30 zeigt die ersten Bytes von HC1 oder HC1/HC2-komprimierten Nutzdaten. Das Betriebssystem Contiki verarbeitet die Komprimierungsverfahren HC1 und HC2 gemeinsam unter einer Implementierung für HC1.

Das Ziel von HC1/HC2 war es, eine Headerkompression auf gänzlich zustandslose Art und Weise zu ermöglichen. Das heißt, es müssen vor dem Austausch von komprimierten Paketen keine Vereinbarungen zwischen den beiden Kommunikationspartnern getroffen werden. Die Verfahren

HC1 und HC2 nutzen dabei in erster Linie das Vorhandensein von Redundanzen in den Paketen aus.

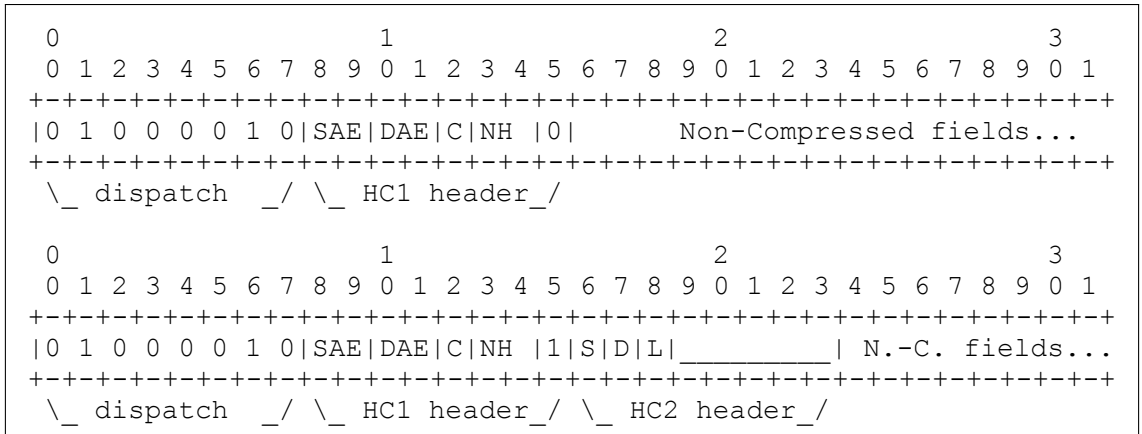


Abbildung 2.30: HC1-komprimiertes IPv6-Paket: ohne und mit HC2, entnommen aus [Shelby u. a., 2009, S. 43]

Die wichtigste Redundanz in 6LoWPAN-Paketen besteht darin, dass sich IP-Adressen aus den Ebene 2-Adressen ableiten lassen. Die Ebene 2-Adresse entspricht der MAC-Adresse der Station und wird für die Generierung der unteren Hälfte der IP-Adresse, dem Interface Identifier (IID), herangezogen. Dieser Teil der IPv6 Sender- und Empfängeradressen kann daher ausgelassen werden.

Schwieriger gestaltet sich das Komprimieren der ersten 64 Bit einer IPv6-Adresse. HC1 berücksichtigt hier nur den Fall, dass es sich um eine link-lokale Adresse mit dem Präfix „FE80::/64“ handelt. HC1 ermöglicht für diesen besonderen Fall das Weglassen des oberen Teils der Sender- oder Empfängeradresse. Für die Kodierung von Sender- und Empfängeradresse werden im HC1-Header jeweils 2 Bits zur Verfügung gestellt, genannt Source Address Encoding (SAE) und Destination Address Encoding (DAE). Die Bedeutung der Bitkombinationen wird in Tabelle 2.5 erläutert.

Tabelle 2.5: Bedeutung der Werte für SAE und DAE bei HC1, Inhalt entnommen aus [Shelby u. a., 2009, S. 44], [Kushalnagar u. a., s. 17-18]

Bitmuster	Prefix	IID
00	unkomprimiert	unkomprimiert
01	unkomprimiert	ermittelt aus L2- bzw. Mesh-Adresse
10	link-lokal (FE80::/64) angenommen	unkomprimiert
11	link-lokal (FE80::/64) angenommen	ermittelt aus L2- bzw. Mesh-Adresse

Tabelle 2.6: Bedeutung der Werte für die NH-Bits bei HC1, Inhalt entnommen aus [Shelby u. a., 2009, S. 44], [Kushalnagar u. a., s. 18]

Bitmuster	Bedeutung
00	Next-Header wird unkomprimiert gesendet
01	Next-Header entspricht 17 (UDP)
10	Next-Header entspricht 1 (ICMP)
11	Next-Header entspricht 6 (TCP)

Die restlichen Bits des HC1-Headers ermöglichen die Kompression der nicht adressbezogenen Komponenten des IPv6-Headers, dargestellt in der Abbildung 2.31. Hier werden einige Annahmen für den üblichen Pakettransport in einem 6LoWPAN getroffen:

- Die IP-Protokollversion ist offensichtlich 6 und braucht demzufolge nicht gesendet werden.
- Die Werte der Felder für das *Flow Label* und die *Traffic Class* werden in der Regel auf null gesetzt. Das gesetzte C-Bit im HC1-Header berücksichtigt diesen Fall und diese Felder werden nicht gesendet. Ist es nicht gesetzt, wird deren Inhalt in den unkomprimierten Datenfeldern übertragen, die dem HC1-Header folgen.
- Das Feld für die Länge der Nutzdaten (*Payload Length*) kann von der verbleibenden Länge der 6LoWPAN-PDU abgeleitet werden. Diese kann aus dem Ebene 2-Rahmen oder über den Fragmentierungsmechanismus ermittelt werden. Dieses Feld muss demzufolge bei verwendetem HC1-Verfahren nicht übertragen werden.
- Das ein Byte große Next-Header-Feld kann einige Werte annehmen, die wahrscheinlicher sind als andere. Die zwei NH-Bits im HC1-Header kodieren hier, wie in Tabelle 2.6 zu sehen, drei der am häufigsten auftretenden Header-Felder. Sind die beiden Bits auf null gesetzt, wird der Wert des Next-Headers unkomprimiert in dem auf den HC1-Header folgenden und dafür vorgesehenen Datenfeld übertragen.

Der Wert für das *Hop Limit* ist zu schwierig zu komprimieren und wird demzufolge in den unkomprimierten Datenfeldern übertragen, die dem HC1- oder HC1/HC2-Header folgen. Die Datensequenz beginnt dabei immer mit den 8 Bits, die das *Hop Limit* enthalten. Anschließend folgen, falls vorhanden, die unkomprimierten Daten in der Reihenfolge, wie sie im HC1-Header kodiert werden:

- Das Präfix der Senderadresse (64 Bits), wenn das höherwertige Bit des SAE null ist.
- Der IID (64 Bits) der Senderadresse, wenn das niederwertige Bit des SAE null ist.
- Das Präfix der Empfängeradresse (64 Bits), wenn das höherwertige Bit des DAE null ist.
- Der IID (64 Bits) der Empfängeradresse, wenn das niederwertige Bit des DAE null ist.
- Die *Traffic Class* (8 Bits) und das *Flow Label* (20 Bits), wenn das C-Bit null ist.
- Der Next-Header, falls die NH-Bits null sind.
- Falls vorhanden, sämtliche unkomprimierten Daten die durch das Verfahren HC2 erzeugt wurden.
- Sämtliche Next-Header und Nutzdaten die von der Komprimierung nicht betroffen sind.

Wenn das Bit ganz rechts im HC1-Header gesetzt ist, folgt ihm unmittelbar der HC2-Header. Die NH-Bits des HC1-Headers verweisen dann auf das Vorhandensein eines Next-Headers für das Protokoll UDP. Der HC2-Header verwendet dabei 3 Bits eines Bytes für die Kompression des *Source Ports* , des *Destination Ports* und der Länge des Datagramms, bestehend aus den Nutzdaten und dem Header, in Bytes. Die UDP-Prüfsumme wird niemals komprimiert.

Die Länge des Datagramms kann relativ einfach aus der Anzahl der verbleibenden Nutzdatenbytes wiedergewonnen werden – das für die Längenkodierung im HC2-Header zugewiesene Bit L ist dementsprechend fast immer gesetzt.

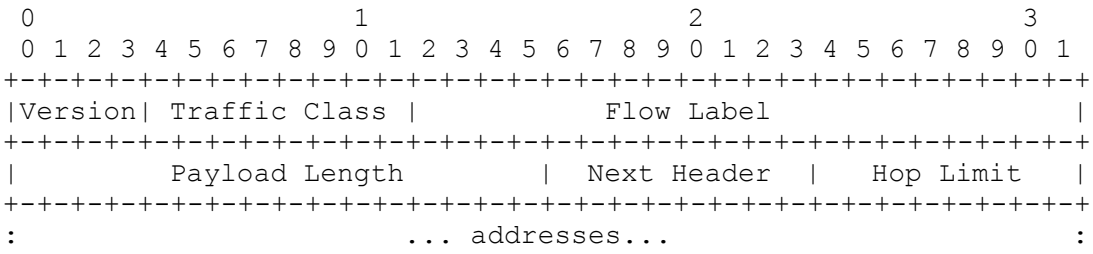


Abbildung 2.31: Nicht adressbezogene Komponenten des IPv6-Headers, entnommen aus [Shelby u. a., 2009, S. 44]

Bei den Portnummern beschränkt man sich darauf, nur die Werte zu komprimieren, die relativ häufig auftreten. Die Wahl fiel hier auf den Bereich von 61616 bis 61631 (0xF0Bn). Jede Portnummer in diesem Bereich kann auf die unteren 4 Bits komprimiert werden, indem das S-Bit für den *Source Port* und das D-Bit für den *Destination Port* innerhalb des HC2-Headers gesetzt wird. Sämtliche UDP-Felder die nicht komprimiert wurden, werden den unkomprimierten Datenfeldern im HC1-Paket in der Reihenfolge angehangen, in der sie im UDP-Header erscheinen.

Das Ergebnis der Headerkomprimierung muss nicht zwingend ein Vielfaches von 8 Bit lang sein. Hardwarekomponenten senden aber üblicherweise keine Daten in Einheiten kleiner als ein Byte. So kann es nötig sein, die restlichen Bits, die zu einem kompletten Byte fehlen, mit Nullen aufzufüllen. Anschließende Datenfelder wie unkomprimierte Header und Nutzdaten beginnen nun an einer Bytegrenze. Das wirkt einem möglichen Datenversatz aufgrund der Headerkomprimierung entgegen. [Shelby u. a., 2009, S. 43-45] [Kushalnagar u. a., S. 16-20]

Kontext-basierte Headerkomprimierung

Die im Standard [RFC4944] (Kushalnagar u. a.) definierten Kompressionstechniken sind sehr effektiv für link-lokale Unicast-Adressen, haben aber einen sehr begrenzten Effekt beim Einsatz von globalen IPv6-Adressen oder Multicastadressen. Für miteinander kommunizierende Stationen in verschiedenen IP-Subnetzen (denkbar wären hier verschiedene PANs auf Grundlage des Standards IEEE 802.15.4), wo routingfähige Adressen erforderlich sind, werden daher effiziente Komprimierungsverfahren für globale IPv6-Adressen benötigt. Bei eingesetztem Verfahren HC1 würden die Sender- und Empfängeradressen des Protokolls IPv6 unkomprimiert übertragen werden.

Der Standard [RFC6282] (Hui u. Thubert [a]) , eine Erweiterung des Standards [RFC4944], beschreibt zu diesem Zwecke zwei Komprimierungstechniken, bezeichnet mit LOWPAN_IPHC (kurz IPHC) und LOWPAN_NHC (kurz NHC). Diese Komprimierungsverfahren werden die Techniken HC1 und HC2 langfristig ersetzen.

Das Verfahren wurde in der Version 2.5 des quelloffenen Betriebssystems Contiki bereits implementiert und trägt dort die Bezeichnung HC06. Das IPHC-Encoding, dokumentiert in Abbildung 2.32, erfordert (bis zu) 13 Bits. 5 Bits davon wurden auf die niederwertigen Bits des *Dispatch Bytes* gelegt, um den Protokolloverhead zu reduzieren und dafür zu sorgen, dass die aufgewendeten Bits auf ein Vielfaches von Acht enden. Die restlichen 8 Bits können dem Header optional als zusätzliches Byte hinzugefügt werden.

Alle nicht komprimierbaren Daten werden den IPHC-Encoding-Bits in der gleichen Reihenfolge wie in einem unkomprimierten IPv6-Header angehängt. Das Feld für die IP-Protokollversion wird in jedem Fall verworfen und die Angabe der Länge der IPv6-Nutzdaten kann aus dem Längenfeld des IEEE 802.15.4-Rahmens oder aus dem 6LoWPAN-Fragmentation-Header, sofern vorhanden, abgeleitet werden.

In den IPHC-Bits werden die folgenden Informationen kodiert:

- **Bits 3 bis 4 (TF):** Diese Bits ermöglichen eine noch differenziertere Kompression des *Traffic Class*- und des *Flow Label*-Feldes des IPv6-Headers:
  - **00:** Beides, *Traffic Class* und *Flow Label*, werden unkomprimiert übertragen. Der Fall ist in der Abbildung 2.32 dokumentiert. Hierbei wird mit 4 Bits aufgefüllt, um ein komplettes Byte zu erhalten. Die *Traffic Class* setzt sich allgemein aus der Explicit Congestion Notification (ECN) [RFC3168] (Floyd u. a.) und dem Differentiated Services Code Point (DSCP) [RFC2474] (Nichols u. a.) zusammen.
  - **01:** Das *Traffic Class*-Feld wird auf die 2 Bits der ECN gemäß [RFC3168] komprimiert. Das *Flow Label* bleibt unverändert. Nach dieser Komprimierung wird mit 2 Bits aufgefüllt, um ein drittes komplettes Byte zu erhalten.
  - **10:** Das *Flow Label*-Feld wird komprimiert und das *Traffic Class*-Feld unkomprimiert übertragen.
  - **11:** Sowohl das *Flow Label*-Feld als auch das *Traffic Class*-Feld werden komprimiert und somit ausgelassen.
- **Bit 5 (NH):** Über dieses Bit wird der 1 Byte große Next-Header kodiert:
  - **0:** Die 8 Bit des Next-Headers werden nicht komprimiert.
  - **1:** Das IPv6-Next-Header-Feld wird ausgelassen und ein weiterer Encoding-Mechanismus, das sogenannte Next Header Coding (NHC) wird angewendet.
- **Bits 6 bis 7 (HLIM):** Im Gegensatz zur HC1-Kompressionstechnik ermöglicht IPHC die Komprimierung des *Hop Limit*-Feldes des IPv6-Headers.
  - **00:** Das *Hop Limit*-Feld wird unkomprimiert übertragen.
  - **01:** Das *Hop Limit*-Feld wird ausgelassen und sein Wert als 1 angenommen.
  - **10:** Das *Hop Limit*-Feld wird ausgelassen und sein Wert als 64 angenommen.
  - **11:** Das *Hop Limit*-Feld wird ausgelassen und sein Wert als 255 angenommen.
- **Bit 8 (Context Identifier Extension (CID)):** Ein Bit, mit dem das Vorhandensein der CID angezeigt wird:
  - **0:** Es wird keine zusätzliche Kontext-Information (CID) verwendet. Wird eine kontextbasierte Koprimierung durch Source Address Compression (SAC) oder Destination Address Compression (DAC) angegeben, wird der Kontext '0' verwendet.
  - **1:** Die 1 Byte große CID wird hinter dem Destination Address Mode (DAM)-Feld hinzugefügt.
- **Bit 9 (SAC):** Gibt die Art der Komprimierung an:
  - **0:** Die Senderadresskomprimierung ist zustandslos.
  - **1:** Die Sendersdresskomprimierung ist kontextbezogen.

- **Bits 10 bis 11 (Source Address Mode (SAM)):**

  Sollte der Wert in SAC=0 sein:

  - **00:** Die komplette 128-Bit-Adresse wird unkomprimiert übertragen.
  - **01:** Die ersten 64 Bits der IP-Adresse werden ausgelassen und dafür das link-lokale Präfix, aufgefüllt mit Nullen, angenommen. Die verbleibenden 64 Bits werden unkomprimiert übertragen.
  - **10:** Die ersten 112 Bits der IPv6-Adresse werden ausgelassen und ihr Wert als link-lokales Präfix, aufgefüllt mit Nullen, angenommen. Die folgenden 64 Bits entsprechen 0000:00FF:FE00:XXXX. Die 16 Bits XXXX werden unkomprimiert übertragen.
  - **11:** Die IPv6-Adresse wird komplett ausgelassen. Die ersten 64 Bits werden durch das link-lokale Präfix gebildet, die verbleibenden 64 Bits werden, wie bei HC1, aus dem IEEE 802.15.4-Rahmen abgeleitet.

  Sollte der Wert in SAC=1 sein:

  - **00:** Die IPv6-Adresse ist eine spezifizierte Adresse (::).
  - **01:** Das 64-Bit-Präfix wird aus der Kontext-Information abgeleitet, die verbleibenden 64 Bits werden unkomprimiert übertragen. Die durch die Kontextinformation bestimmten Bits werden immer verwendet. Für Bits des IID, die nicht durch die Kontextinformation bestimmt werden können, werden die entsprechenden unkomprimiert übertragenden Bits übernommen. Sämtliche verbleibenden Bits werden auf null gesetzt.
  - **10:** Die ersten 112 Bits der Adresse werden aus der Kontext-Information abgeleitet und die verbleibenden 16 Bits werden unkomprimiert übertragen. Die Bits aus der Kontextinformation werden immer verwendet. Die Bits des IID, die nicht aus den Kontextinformationen gewonnen werden können, werden aus der Bitmap 0000:00FF:FE00:XXXX übernommen. XXXX geben hierbei die unkomprimiert übertragenen Bits an. Sämtliche verbleibenden Bits werden auf null gesetzt.
  - **11:** Die Adresse wird komplett aus der Kontext-Information und gegebenenfalls aus dem Ebene 2-Rahmen abgeleitet. Die Bits aus der Kontextinformation haben hierbei Vorrang. Alle verbleibenden Bits werden auf null gesetzt.

- **Bit 12 (M):** Kodierung von Multicastadressen:

  - **0:** Die Empfängeradresse ist keine Multicastadresse.
  - **1:** Die Empfängeradresse ist eine Multicastadresse.

- **Bit 13 (DAC):**

  - **0:** Die Kompression der Empfängeradresse ist zustandslos.
  - **1:** die Kompression der Empfängeradresse ist zustandsbehaftet, abhängig vom Kontext.

- **Bits 14 bis 15 (DAM):**

  Sollte der Wert in M=0 und in DAC=0 sein:

  - **00:** Die komplette 128-Bit-Adresse wird unkomprimiert übertragen.
  - **01:** Die ersten 64 Bits der IPv6-Adresse werden ausgelassen und es wird das link-lokale Präfix, aufgefüllt mit Nullen, angenommen. Die verbleibenden 64 Bits werden unkomprimiert übertragen.

- **10:** Die ersten 112 Bits der IPv6-Adresse werden ausgelassen und für die ersten 64 Bits das link-lokale Präfix, aufgefüllt mit Nullen, angenommen. Die folgenden 64 Bits entsprechen der Bitmap 0000:00FF:FE00:XXXX. Die 16 Bits XXXX werden unkomprimiert übertragen.
- **11:** Die IPv6-Adresse wird komplett ausgelassen. Die ersten 64 Bits werden durch das link-lokale Präfix gebildet, die verbleibenden 64 Bits werden, wie bei HC1, aus dem IEEE 802.15.4-Rahmen abgeleitet.

Sollte der Wert in M=0 und DAC=1 sein:

- **00:** Dieser Wert ist für eine spätere Anwendung reserviert.
- **01:** Die Adresse wird aus der Kontext-Information abgeleitet, die unteren 64 Bits werden unkomprimiert übertragen. Die durch die Kontextinformation bestimmten Bits werden immer verwendet. Für Bits des IID, die nicht durch die Kontextinformation bestimmt werden können, werden die entsprechenden unkomprimiert übertragenden Bits übernommen. Alle verbleibenden Bits sind null.
- **10:** Die oberen Bits der Adresse werden aus der Kontext-Information abgeleitet und die unteren 16 Bits werden unkomprimiert übertragen. Sämtliche Bits des IID, die nicht durch die Kontextinformation abgedeckt werden, können aus der Bitmap 0000:00-FF:FE00:XXXX übernommen werden. XXXX enthält dabei die 16 Bits, die unkomprimiert übertragen werden. Alle verbleibenden Bits werden auf null gesetzt.
- **11:** Die Adresse wird komplett aus der Kontext-Information und gegebenenfalls aus dem Ebene 2-Rahmen abgeleitet. Die Bits aus der Kontextinformation haben hierbei Vorrang. Alle verbleibenden Bits werden auf null gesetzt.

Sollte der Wert in M=1 und in DAC=0 sein:

- **00:** Die 128 Bits der Adresse werden unkomprimiert übertragen.
- **01:** Die Adresse wird mit 48 Bits kodiert und besitzt die Form FFXX::00XX:XXXX:X-XXX.
- **10:** Die Adresse wird mit 32 Bits kodiert und besitzt die Form FFXX::00XX:XXXX.
- **11:** Die Adresse wird mit 8 Bits kodiert und besitzt die Form FF02::00XX.

Sollte der Wert in M=1 und in DAC=1 sein:

- **00:** Die Adresse wird mit 48 Bits kodiert und besitzt die Form FFXX:XXLL:PPPP:PP-PP:PPPP:PPPP:XXXX:XXXX. Die mit X gekennzeichneten Nibbles werden unkomprimiert übertragen. P kennzeichnet Nibbles zum Kodieren des Präfix (welches sich beispielsweise aus einem Kontext ergibt). L kennzeichnet Nibbles, die zum Kodieren der Präfixlänge verwendet werden. Die Präfixinformation P und L wird aus dem Kontext selbst abgeleitet. Dieses Format ist konform mit den unicast-präfixbasierten IPv6-Multicast-Adressen, definiert in [RFC3306] (Haberman u. Thaler) und [RFC3956] (Savola u. Haberman).
- **01:** Dieser Wert ist für eine spätere Anwendung reserviert.
- **10:** Dieser Wert ist für eine spätere Anwendung reserviert.
- **11:** Dieser Wert ist für eine spätere Anwendung reserviert.

[Vasseur u. Dunkels, 2010, S. 242-244] [Hui u. Thubert, a, S. 5-9]

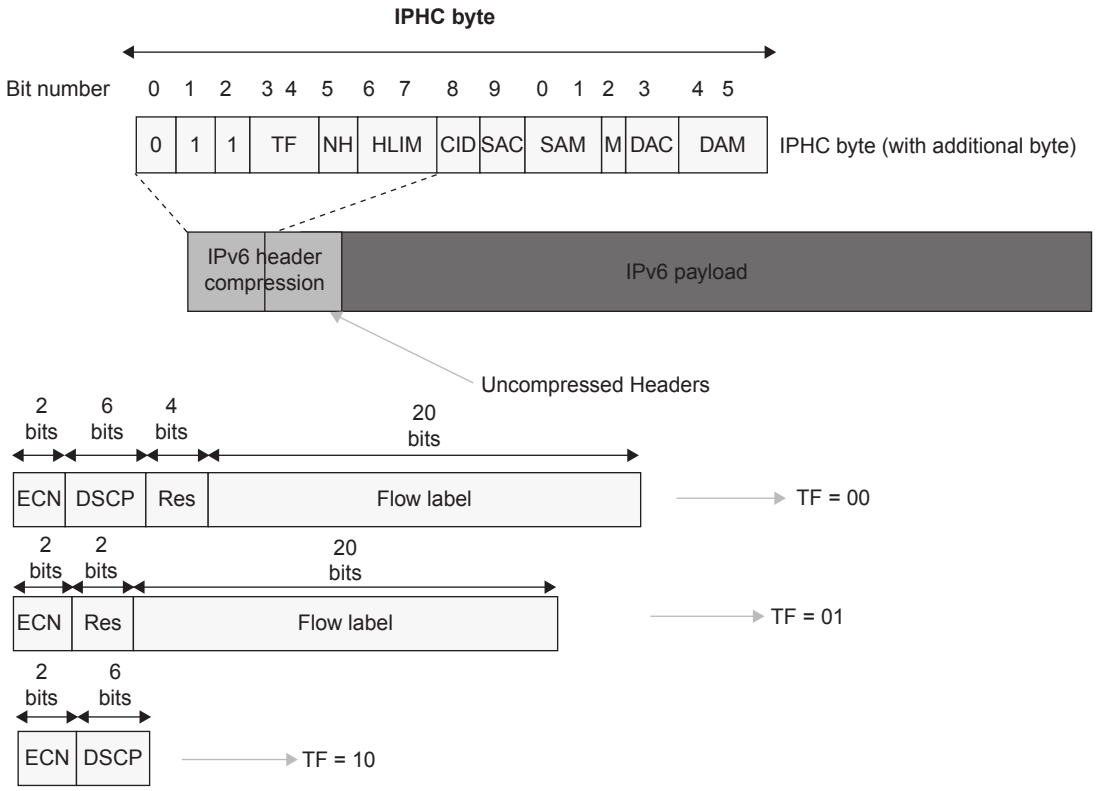


Abbildung 2.32: Das IPHC-Encoding, entnommen aus [Vasseur u. Dunkels, 2010, S. 242]

Der Context Identifier

Die Kompressionstechnik IPHC ist darauf angewiesen, dass zwischen der sendenden Station, die das IPv6-Paket komprimiert, und der empfangenden Station, die dieses Paket wieder dekomprimiert, Kontextinformationen ausgetauscht werden. Wie diese Informationen geteilt und aktuell gehalten werden, wird im Standard [RFC6282] (Hui u. Thubert [a]) nicht beschrieben – ebenso wenig wie die Reaktionen auf unbekannte und ungültige Kontextinformationen. Hier existiert durchaus Potential für Protokollerweiterungen. Die Spezifikation ermöglicht das Nutzen von bis zu 16 Kontexten. Dabei muss für das Kodieren der Sendeadresse nicht der gleiche Kontext wie für das Kodieren der Empfängeradresse verwendet werden.

Bei gesetztem CID-Bit im IPHC-Encoding wird ein zusätzliches Byte hinter den DAM-Bits angehangen. Dieses Byte kodiert das Paar von Kontextinformationen, das für die Kompression der Sender- und Empfängeradresse verwendet wird. Die Abbildung 2.33 zeigt die entsprechende Kodierung. Die vorhandenden Bits werden dabei ihrem Verwendungszweck zugeordnet.

Der Source Context Identifier (SCI) identifiziert dabei das verwendete Präfix der IPv6-Sendeadresse und der Destination Context Identifier (DCI) das Präfix der IPv6-Empfängeradresse, wenn die kontextbasierte Komprimierung verwendet wird. [Vasseur u. Dunkels, 2010, S. 245] [Hui u. Thubert, a, S. 9-10]

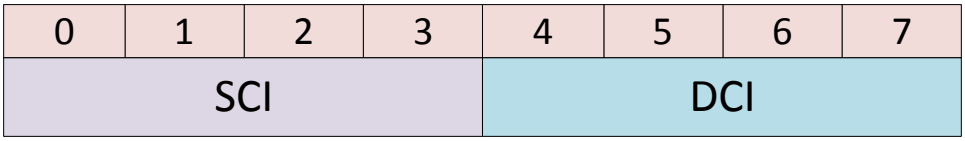


Abbildung 2.33: Aufbau des Context Identifiers, Inhalt aus [Hui u. Thubert, a, S. 10]

Die IPv6-Next-Header-Kompression

Das Protokoll IPv6 verwendet aufeinander folgende Header bei denen das Next-Header-Feld die Art des Headers anzeigt, der folgt. Vergleichbar mit der HC1-Technik, bei der die entsprechende Information in den Bits 5 bis 6 des Encoding-Bytes kodiert wird, bietet IPHC einen Mechanismus, um das IPv6-Next-Header-Feld auszulassen. Die IPv6-Next-Header-Kompression wird durch das Setzen von Bit 5 innerhalb des IPHC-Bytes angezeigt. Anschließend wird den unkomprimierten Daten des IPv6-Header-Feldes, wie in Abbildung 2.34 zu sehen, ein zusätzliche Byte angefügt. Dieses Byte wird LOWPAN_NHC (kurz NHC) genannt.

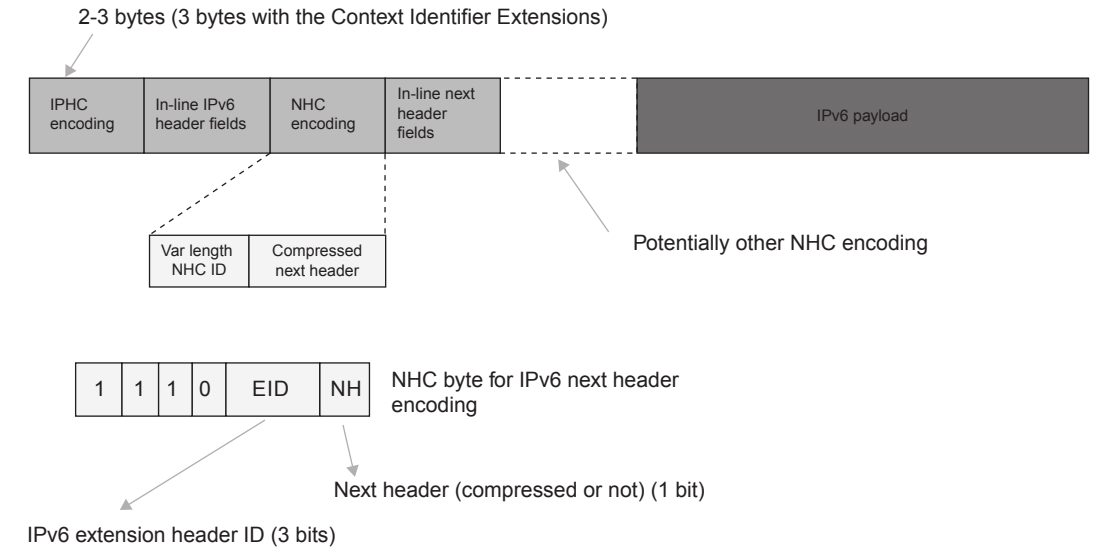


Abbildung 2.34: IPv6-Paket mit IPHC-und NHC-Kompression, übernommen aus aus [Vasseur u. Dunkels, 2010, S. 245]

Die NHC-Daten sind, abhängig von der Art des Next-Headers, unterschiedlich lang. Dadurch wird eine effizientere und flexiblere Kompressionstechnik ermöglicht. Wie in der Abbildung 2.34 zu sehen, bestimmen die ersten sieben Bits des NHC-Bytes den folgenden Next-Header. Die drei rechtseitigen Bits davon werden als Extension Header ID (EID) bezeichnet und kodieren den IPv6-Erweiterungsheader, der unmittelbar folgt. Die folgende Auflistung zeigt die Zuordnung der Headertypen zu den Bitmustern:

- **0:** IPv6-Hop-by-Hop-Header [RFC2460] (Deering u. Hinden)
- **1:** IPv6-Routing-Header [RFC2460]
- **2:** IPv6-Fragment-Header [RFC2460]
- **3:** IPv6-Destination-Options-Header [RFC2460]
- **4:** IPv6-Mobility-Header [RFC6275] (Arkko u. a.)
- **5:** reserviert für eine spätere Anwendung
- **6:** reserviert für eine spätere Anwendung
- **7:** IPv6-Header

Das gesetzte letzte, rechtseitige Bit (NH) des NHC-Bytes signalisiert, dass ein weiterer Next-Header, komprimiert mit der LOWPAN-NHC-Technik, folgt. Bei nicht gesetztem Bit folgt der Next-Header unverändert. Unter Einsatz der LOWPAN-NHC-Technik wird der IPv6-Header nicht verändert, mit zwei Ausnahmen, dem Längenfeld und dem Next-Header-Feld:

- Das Next-Header-Feld wird ausgelassen, wenn das NH-Bit gesetzt ist. Dadurch werden Redundanzen verhindert.
- Die Bedeutung des Längenfeldes im Erweiterungsheader wird verändert. Vorher definierte es die Länge des Headers in 8-Byte Schritten ohne die einleitenden acht Bytes. Durch das byteweise Zählen werden ein möglicher unnötiger Overhead sowie eine zusätzliche Fragmentierung verhindert.

[Vasseur u. Dunkels, 2010, S. 245-246] [Hui u. Thubert, a, S. 14-16]

Die Kompression des UDP-Headers unter Verwendung von LOWPAN_NHC

NHC bietet eine weitere UDP-Header-Kompressionstechnik mit einigen Verbesserungen gegenüber HC2. Ein Bereich von 16 aufeinanderfolgenden wahrscheinlichen Ports, kodiert in der Form 0xF0BX, wird von der HC2-Technik übernommen. Dies führt allerdings zu Kompatibilitätsproblemen bei Anwendungen, die diesen Portbereich bereits verwenden. Darum empfiehlt die Spezifikation [RFC6282] (Hui u. Thubert [a]) einen Transport Layer Security (TLS) Message Integrity Check (MIC), definiert in [RFC5246] (Dierks), um den Inhalt und seine Integrität zu bestätigen. Obwohl in [RFC2460] (Deering u. Hinden) das Verwenden einer UDP-Prüfsumme verlangt wird, ermöglicht die Spezifikation [RFC4944] (Kushalnagar u. a.) das Umgehen dieser Regel, falls dies eine höhere Protokollschicht zulässt. Obwohl die Spezifikation [RFC4944] einer dazwischenliegenden Station erlaubt die UDP-Prüfsumme zu ignorieren, selbst wenn das empfangende Paket eine UDP-Prüfsumme besitzt, sollte dies nicht ohne die Bestätigung durch den Sender geschehen. Umgekehrt kann sich eine dazwischenliegende Station dazu entscheiden, eine UDP-Prüfsumme einzufügen, wenn ein Paket ohne UDP-Prüfsumme empfangen wurde. Die Berechnung der UDP-Prüfsumme hat entsprechend den Regeln spezifiziert in [RFC0768] (Postel [b]) und [RFC2460] zu erfolgen. Die Abbildung 2.35 illustriert die NHC-Bytes für den UDP-Header:

- **Bit 5 (C):** UDP-Prüfsumme
  - **0:** Die 16 Bit UDP-Prüfsumme wird unkomprimiert übertragen.
  - **1:** Die 16 Bit UDP-Prüfsumme wird ausgelassen und von dem 6LoWPAN-Endpunkt erneut berechnet.
- **Bits 6 bis 7 (P):** Source- und Destination-Ports

  - **00:** Sowohl der Source-Port als auch der Destination-Port wird unkomprimiert übertragen.
  - **01:** Die 16 Bits des Source-Ports werden unkomprimiert übertragen. Die ersten 8 Bits des Destination-Ports werden ausgelassen und als 0xF0 angenommen. Die verbleibenden 8 Bits werden unkomprimiert übertragen.
  - **10:** Die ersten 8 Bits der Source-Ports werden ausgelassen und als 0xF0 angenommen. Die verbleibenden 8 Bits werden unkomprimiert übertragen. Die 16 Bits des Destination-Ports werden unkomprimiert übertragen.
  - **11:** Die ersten 12 Bits der Source- und Destination-Ports werden ausgelassen und als 0xF0B angenommen. Die verbleibenden 4 Bits der Source- und Destination-Ports werden unkomprimiert übertragen.

Daten die unkomprimiert übertragen werden (teilweise oder komplett) erscheinen in der gleichen Ordnung, wie im UDP-Header-Format laut [RFC0768] (Postel [b]).

Die Abbildung 2.35 zeigt ein vollständiges Beispiel bei dem

- IPHC für die Headerkomprimierung verwendet wurde.
- NHC für die Komprimierung eines erweiterten IPv6-Fragmentierungsheaders verwendet wurde.

NHC wird verwendet, um den UDP-Header auf die effizienteste Weise zu komprimieren. Hier wird die UDP-Prüfsumme ausgelassen und der Source- sowie der Destination-Port komprimiert. [Vasseur u. Dunkels, 2010, S. 246-248] [Hui u. Thubert, a, S. 16-19]

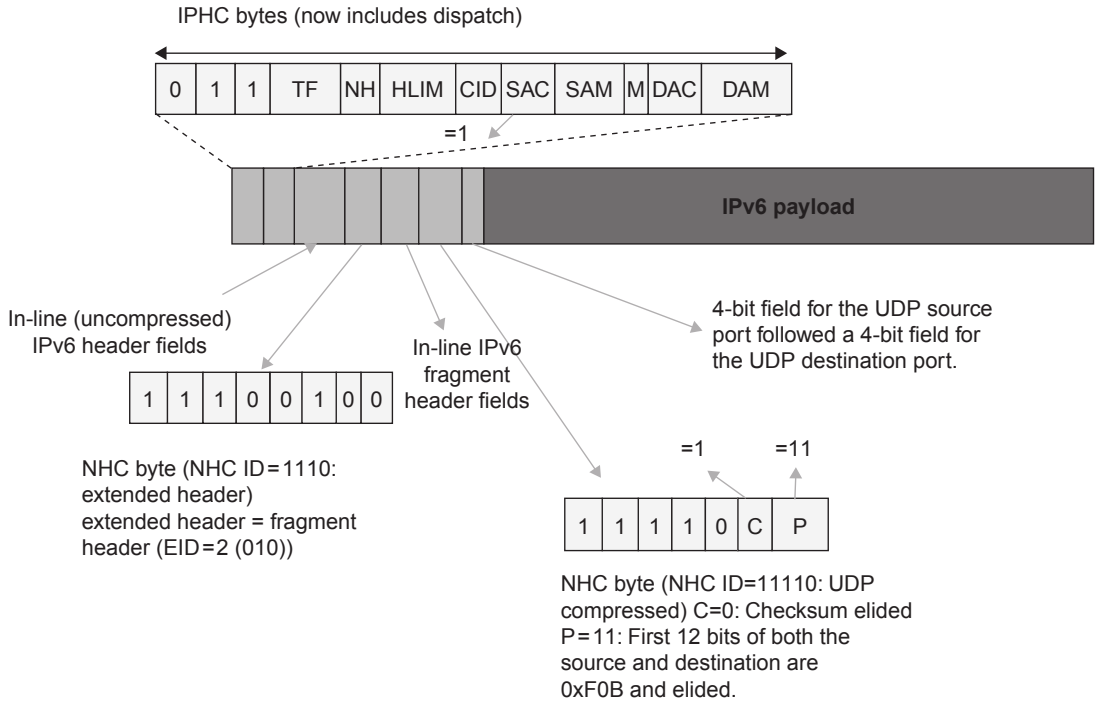


Abbildung 2.35: IPv6-Paket mit IPHC-Kompression und NHC-Kompression für Erweiterungs- sowie UDP-Header, übernommen aus aus [Vasseur u. Dunkels, 2010, S. 247]

2.2.5 Fragmentation und Reassembly

IP-Pakete werden in Größen unterschiedlicher Länge übertragen. Dabei weist das kleinste mögliche IPv6- Paket eine Länge von nur 40 Bytes auf und enthält nur den IP-Header. Das größte IPv6-Paket kann, unter Verwendung der selten eingesetzten Jumboframe-Option [RFC2675] (Borman u. a.) und einer Kodierung mit 32 Bit, Nutzdaten von 4.294.967.295 Bytes enthalten. Ohne diese Option können IPv6-Pakete eine Nutzlast von 65.535 transportieren, was zusammen mit dem Header eine Gesamtpaketlänge von 65.575 Bytes ergibt. Nur wenige Netzwerke können Pakete dieser Größe effizient übertragen. In der Regel wird eine deutlich geringere MTU, bspw. 1500 für das Ethernet, verwendet.

Auf einem Host mit einer einzigen Schnittstelle ist es für gewöhnlich möglich, ihre MTU zu ermitteln. Das versetzt eine Anwendung in die Lage, die Größe der gesendeten Pakete anzupassen. Werden während einer Übertragung mehrere Schnittstellen verwendet, ist dies nicht mehr so einfach. Im schlimmsten Fall wird sich die MTU des Paketes auf ihrem Weg zum Ziel mit jedem Zwischenschritt verändern. Das macht es für die Anwendung sehr schwer, eine Paketgröße zu wählen, die über den gesamten Pfad übertragen werden kann. IPv4 hat hier sehr lockere Anforderungen an die MTU einer Verbindung: Ein IPv4 Netzwerk muss in der Lage sein, Pakete mit einer Mindestlänge von 68 Bytes zu übertragen. Da die meisten Verbindungen in der Lage sind, größere Pakete zu übertragen, besteht keine Notwendigkeit, Anwendungen durch das Senden von Paketen dieser Größe oder darunter einzuschränken. IPv4 setzt voraus, dass jeder Knoten (sendender Host oder ein Router) in der Lage sein muss, Pakete in kleinere Stücke zu zerlegen. Jedes Zielgerät einer Übertragung muss in der Lage sein, diese Fragmente wieder zusammenzufügen. Um ein unfragmentiertes Paket von einem fragmentierten unterschieden zu können, wird das Erstgenannte oft Datagramm genannt.

Oft wird die minimale MTU des Protokolls IPv4 mit der minimalen Größe des *Reassembly*-Buffers verwechselt. Jedes mögliche Zielgerät muss bei IPv4 in der Lage sein, ein Datagramm von 576 Bytes zu empfangen. Das kann in einem Stück erfolgen oder in Form von Paketfragmenten, die wieder zusammengefügt werden müssen [RFC0791] (Postel [a]). Einige Protokolle wie das ursprüngliche Domain Name System (DNS) bis zur Einführung des Extension Mechanisms for DNS (EDNS0) [RFC2671] (Vixie) waren nicht in der Lage, größere Datagramme zu senden.

Das Protokoll TCP kann höhere Datagrammgrößen durch seine MSS-Option während des Verbindungsaufbaus aushandeln. Implementierungen dieser Option waren zwingend erforderlich, um diese sehr wichtige Performanceverbesserung zu ermöglichen. Die Verwendung größerer Datagrammlängen erhöht die gesamte TCP-Performance jedoch nur, wenn das Datagramm bei der späteren Übertragung nicht fragmentiert wird. Die Hosts, die sich auf eine MSS-Aushandlung einlassen, kennen die MTU auf ihren eigenen Verbindungen. So können sie sicherstellen, dass eine Fragmentierung weder auf dem ersten noch auf dem letzten Übertragungsabschnitt geschieht. Ein Mechanismus für die Bestimmung der minimalen MTU auf den beteiligten Netzwerkabschnitten wird Path MTU Detection (PMTUD) ([RFC1191] (Mogul u. Deering) für IPv4) genannt. Dieser Algorithmus verwendet das *Don't Fragment*-Flag (DF) des Protokolls IPv4, welches einen Netzknoten, der normalerweise ein Paket fragmentieren würde, dazu zwingt, stattdessen die ICMP-Fehlermeldung *Destination unreachable* zurück zu senden. Diese Fehlermeldung wird bei gesetztem DF-Flag und benötigter Fragmentierung üblicherweise als *Packet too big* interpretiert!

Da sich der Pfad zwischen den TCP-Instanzen verändern kann, muss der Algorithmus adaptiv/anpassbar sein. Wenn sich bspw. eine Route zu einem Weg mit kleinerer Pfad-MTU verändert, werden evtl. zu lange Pakete das Senden eines ICMP-Pakets auslösen, was den Sender dazu befähigt, die Datagrammgröße schnell zu reduzieren. Eine Erhöhung der verfügbaren Pfad-MTU kann

vom Sender jedoch nur bemerkt werden, wenn er gelegentlich den Weg durch das Senden größerer Pakete, als es die bekannte MTU zulässt, erkundet.

Das Resultat des verbreiteten Einsatzes der PMTUD ist, dass bei typischem Internetverkehr weniger als 1 Prozent der Pakete fragmentiert werden. Dennoch werden in jedem IPv4-Paket 32 Bit (siehe 2.36) des Header-Raumes freigehalten, um eine spätere Fragmentierung auf dem Übertragungspfad zu ermöglichen. Der Header-Raum wird auch reserviert, wenn das DF-Flag gesetzt ist.

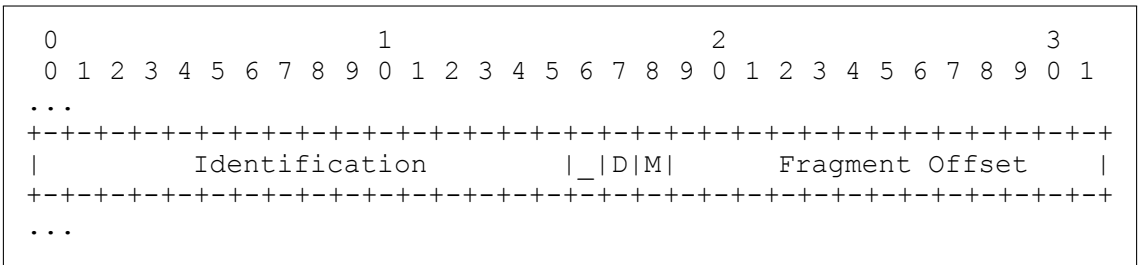


Abbildung 2.36: *Fragmentation*-Felder im IPv4-Header (D=*Don't Fragment*, M=*More Fragments*), entnommen aus [Shelby u. a., 2009, S. 53]

Das IPv4-Reassambly wird auf dem Zielhost des Datagramms ausgeführt. Fragmente mit der gleichen Quelladresse, Zieladdresse sowie identischem Eintrag im IP-Protokoll- und *Identifiation*-Feld werden zusammengefasst. Das erste Fragment eines Datagramms wird durch das auf den Wert null gesetzte *Fragment Offset* identifiziert. Das letzte Fragment wird durch ein auf null gesetztes *More Fragments*-Flag gekennzeichnet. Ist beides auf null gesetzt, handelt es sich um ein unfragmentiertes Datagramm. Das Internet garantiert nicht, dass die Pakete nacheinander übertragen werden – sie können den Empfänger in beliebiger Reihenfolge erreichen.

Fragmente werden in den *Reassembly*-Buffer an eine Position entsprechend ihres *Fragment Offsets* kopiert. Der *Fragment Offset* wird dabei in Einheiten von 8 Bytes gezählt und braucht demzufolge nur 13 Bits anstelle von 16 Bits. Das Datagramm ist vollständig, wenn das letzte Fragment vorhanden ist und die vorigen Fragmente bereits empfangen und in den Buffer geschrieben wurden. Bevor nicht das letzte Fragment empfangen wurde, hat der IPv4-Reassembly-Prozess keine Information darüber, wie groß das zu empfangende Datagramm sein wird.

Die Fragmente eines Datagramms können bei der Übertragung verloren gehen. Dadurch kann der Reassembly-Prozess nicht vollendet werden. Aus diesem Umstand heraus ist es nötig, den Vorgang mittels eines Timers (timeout) nach einer definierten Zeit abzubrechen. Bereits empfangene Fragmente müssen dann erneut übertragen werden. Im Gegensatz zum TCP-Selective Acknowledgment (SACK) bietet IP keine Möglichkeit, die Informationen in bereits empfangenen Fragmenten zu nutzen. In einem unsicheren Übertragungsmedium kann der Verlust eines einzelnen Paketes das Verwerfen weiterer Pakete zur Folge haben, wenn es Teil eines fragmentierten Datagramms war. Darüber hinaus verbrauchen diese teilweise zusammengesetzten Datagramme beachtlichen Speicherplatz, bevor sie letztendlich verworfen werden. Außerhalb von quasi-verlustfreien Netzwerken wie bspw. LANs, wird die IP-Fragmentierung möglichst selten eingesetzt.

Um den IP-Header zu vereinfachen, wurde bei dem Protokoll IPv6 das *Fragmentation*-Feld entfernt. Wenn die Quelle eines Datagramms die Fragmentierungsoption verwenden möchte, muss sie einen zusätzlichen Fragmentierungsheader als Erweiterungsheader anfügen. Eine Fragmentierung

ist nur beim sendenden Gerät möglich. IPv6 unterstützt keine Fragmentierung während der Übertragung – daher kann auch das DF-Flag eingespart werden! Um diesen Umstand angenehmer zu gestalten, wurde die minimale MTU deutlich angehoben und beträgt nun 1280 Bytes. Alle Netzwerke müssen daher in der Lage sein, MTUs dieser Größe übertragen zu können. Die empfohlene MTU wurde auf 1500 Bytes festgesetzt ([RFC2460] (Deering u. Hinden)), damit sie sich mit der MTU des Ethernets deckt. Diese Technologie wird auf Internetpfaden sehr häufig eingesetzt. Die kleinere MTU von 1280 Bytes wurde verwendet, um Platz für *Tunneling*-Header zu schaffen, in die das Paket gebettet wird, bevor es über das Ethernet übertragen wird. Darüber hinaus wird noch etwas Platz für den IP-Header und weiterer Header, die einer voraussichtlichen Payloadgröße von 1024 Bytes vorausgestellt werden, reserviert. Im Gegensatz zu IPv4 definiert IPv6 eine minimale Reassembly-Größe von 1500 Bytes.

Höhere Schichten des OSI-Referenzmodells, die größere Datagramme senden möchten als es die Pfad-MTU erlaubt, verwenden den IPv6-Fragmentierungsheader (siehe Abbildung 2.37). Abgesehen von dem *Next Header*-Feld, welches in die IPv6-Erweiterungsheaderkette eingefügt werden muss, ist der Inhalt ziemlich ähnlich dem der entsprechenden IPv4-Felder. Allerdings wurde das *Identification*-Feld doppelt so groß gewählt wie die 16 Bits, die durch IPv4 bereitgestellt werden. Die geringe Anzahl an Bits war bei den heutigen hohen Datenübertragungsraten nicht mehr ausreichend ([RFC4963] (Heffner u. a.)). Die IPv6-Spezifikation befürwortet, dass IPv6 Knoten PMTUD implementieren ([RFC1981] (McCann u. a.) für IPv6), so dass eine größere Pfad-MTU von 1280 ermittelt und die daraus entstehenden Vorteile genutzt werden können. Die Spezifikation bedenkt vernünftigerweise auch, dass ressourcenbegrenzte IPv6-Implementierungen sich einfach selbst begrenzen können, indem sie keine Pakete größer als 1280 Bytes senden. Das erspart ihnen die Implementierung einer PMTUD.

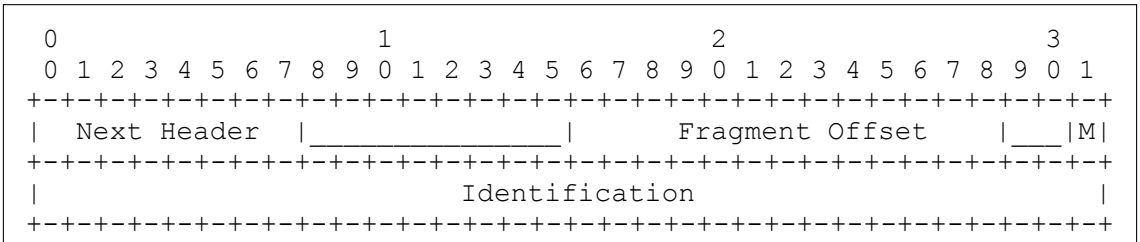


Abbildung 2.37: Der IPv6-Fragmentierungsheader, entnommen aus [Shelby u. a., 2009, S. 55]

IPv6 bietet also eine eigene Fragmentierungsoption für Datagramme größer als die minimale MTU von 1280 Bytes oder der ermittelten Pfad-MTU. Das Protokoll verlässt sich darauf, dass ein Netzwerk in der Lage ist, Pakete von mindestens 1280 Bytes zu transportieren. Für Verbindungen, die von Hause aus nicht in der Lange sind diese Paketgröße zu transportieren, muss mehr Aufwand bei der IPv6-Implementierung betrieben werden, indem Pakete in Ebene 2-Rahmen aufgespalten und am Ziel-IPv6-Knoten wieder zusammengefügt werden. [Shelby u. a., 2009, S. 52-55]

2.2.6 Das Fragmentierungsformat

Die Anforderungen an das Fragmentierungsformat des Standards 6LoWPAN sind mit denen, die an das Protokoll IPv4 gestellt werden, vergleichbar. 6LoWPAN übernimmt daher einen ähnlichen Mechanismus, der sich nur in den Details unterscheidet. Dadurch kann eine effizientere Kodierung und Implementierung ermöglicht werden. Anstelle des *More Fragments*-Flags kopiert der Standard 6LoWPAN die Größe des zusammenzusetzenden Paketes (IPv6-Header und IPv6 Payload) in jedes Fragment. Das ermöglicht dem Zielknoten bereits mit dem Empfang des ersten

Fragmentes ausreichend Buffer für das gesamte IP-Paket zu reservieren. Dabei spielt es keine Rolle, welches Fragment den Zielknoten zuerst erreicht!

Der Standard [RFC4944] (Kushalnagar u. a.) bezeichnet die wieder zusammengesetzten Pakete etwas irreführend als Datagramme, und nennt das Feld für die Größe eines Paketes daher *datagram_size*. Es hat eine Länge von 11 Bits und ermöglicht das Zusammenfügen von Datenpaketen in Größen von bis zu 2047 Bytes. Die MTU der 6LoWPAN-Schnittstellen von 1280 Bytes liefert also Paketgrößen, die sich problemlos fragmentieren lassen.

Um die verschiedenen, wieder zusammenzusetzenden Pakete unterscheiden zu können, wird ein 16 Bit *datagram_tag* in Kombination mit der Ebene 2-Adresse des Senders, der Ebene 2-Adresse des Ziels und der *datagram_size* verwendet. Dies ist vergleichbar mit dem IPv4-*Identification*-Feld.

Die Größe des *datagram_tag* von 16 Bits wurde im Hinblick auf die begrenzte Übertragungsdatenrate in 6LoWPANs gewählt. Vorausgesetzt, es wird eine Fragmentierung durch das Senden von Paketen auf Ebene der Netzwerkschicht von mindestens 128 Bytes erzwungen. Darüber hinaus wird angenommen, dass eine einzelne Station die 250 kBit/s einer IEEE 802.15.4-Verbindung komplett auslastet. Selbst unter diesen ungünstigen Bedingungen würde sich das übertragene *datagram_tag* erst nach vier Minuten wiederholen!

Ein 8 Bit großes *datagram_offset* zeigt die Position des Fragments in einem wieder zusammengesetzten IPv6-Paket an. Wie beim IP-Protokoll wird das Offset in Einheiten von 8 Byte gezählt, so dass mit 8 Bit die kompletten 2047 Bytes kodiert werden können. So wie bei allen 6LoWPAN-Rahmen, identifiziert das erste Byte (*Dispatch Byte*) eines Fragments die Art des Rahmens. Acht der möglichen Werte für dieses Byte ($11100nnn_2$) wurden den Fragmenten zugewiesen. Dabei gehören diese drei Bits eigentlich zur *datagramm_size*. So muss kein Padding (Auffüllen) durchgeführt werden, um die anderen Bits unterzubringen. Das sich daraus ergebende Format, gezeigt in Abbildung 2.38, wird für alle außer dem ersten Fragment einer 6LoWPAN-Fragment-Sequenz verwendet.

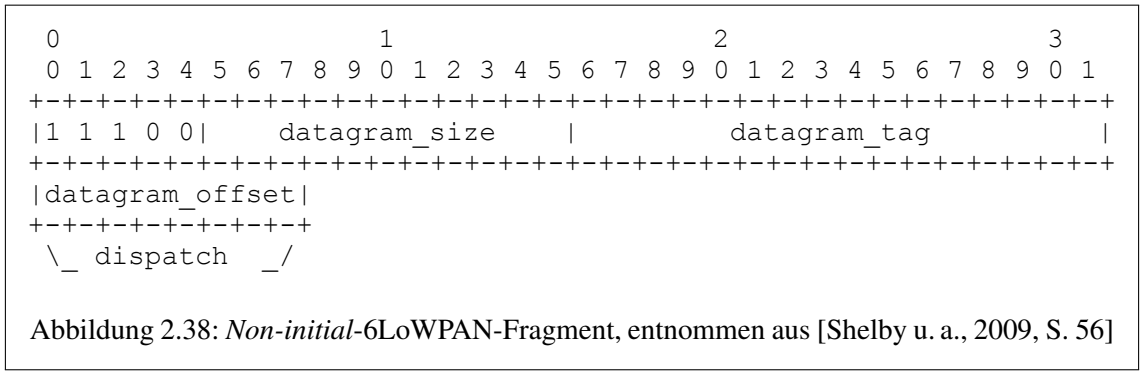


Abbildung 2.38: *Non-initial*-6LoWPAN-Fragment, entnommen aus [Shelby u. a., 2009, S. 56]

Da die meisten Pakete die in einem 6LoWPAN gesendet werden relativ klein sind, ist der Anteil der einleitenden Rahmen im Falle einer Fragmentierung sehr hoch. Das daraus resultierende Fragment-Offset von null erlaubt eine Optimierung durch das Weglassen dieses Wertes. Dem *Dispatch Byte* wurden acht weitere Werte ($11000nnn_2$) zugewiesen, um ein alternatives Fragment-Format zu realisieren, das ein *datagram_offset* von null voraussetzt (siehe Abbildung 2.39).

Eine Station, die eine 6LoWPAN-PDU senden möchte, die zu groß ist um in einen Ebene 2-Rahmen zu passen, kann das folgende Verfahren anwenden:

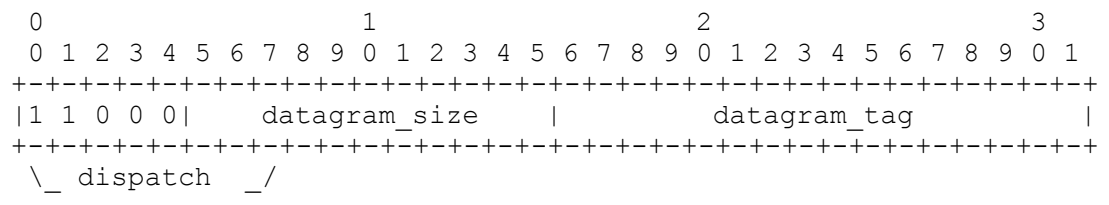


Abbildung 2.39: *Initial*-6LoWPAN-Fragment, entnommen aus [Shelby u. a., 2009, S. 56]

- Die Variable *packet_size* wird auf die Größe des IPv6-Paketes (Header und Payload) gesetzt und die Variable *header_size* auf die Größe der 6LoWPAN-Header, die nur in dem ersten Fragment vorhanden sein müssen. Das sind zum Beispiel das *Dispatch Byte* sowie der komprimierte oder unkomprimierte IPv6-Header (einschließlich der *non-compressed*-Felder für den letzten Fall). Wenn ein Teil des IPv6-Paket-Headers oder des Payloads (wie zum Beispiel der UDP-Header) zu dem *Compressed Header* komprimiert wird, muss die *header_size* um den entsprechenden Betrag korrigiert werden. Das führt in der Regel dazu, dass die *header_size* einen negativen Wert enthält. Zusammengefasst: *header_size* + *packet_size* ist die Grösse der 6LoWPAN-PDU, die sich ergeben würde, wenn sie unfragmentiert gesendet werden könnte.

- In die Variable *max_fram* wird der verbleibende Platz im Ebene 2-Rahmen geschrieben, nachdem PHY, MAC, Address- und Security-Headers und Trailer verrechnet wurden. Dazu zählen auch sämtliche 6LoWPAN-Header, die jedem Fragment oder ganzen Paket vorangestellt werden müssen (so wie Mesh-Header für Mesh-Under). Hierbei ist zu bedenken, dass sich der Inhalt der Variable *max_fram* mit jedem Hop aufgrund der entsprechenden Sicherheitseinstellungen oder Einstellungen für die Adressgröße ändern kann. Solange die Bedingung *header_size*+*packet_size*>*max_frame* nicht erfüllt ist, wird keine Fragmentierung benötigt und die PDU kann einfach in den Ebene 2-Rahmen kopiert werden.

- Die globale Variable *datagram_tag* muss auf einen neuen Wert erhöht werden, der in allen Fragmenten dieser PDU verwendet wird.

- Der nun schwierige Teil ist es, nur so viele Daten mit dem ersten Fragment zu senden, dass es nahezu ausgefüllt ist und auf einen vielfachen Wert von 8 Bytes innerhalb des IPv6-Paketes endet.

  Die Variable *max_frag_initial* wird auf den Wert [(*max_frame*-4-*header_size*)/8]*8 gesetzt. Dabei wird ein Platz von vier Bytes für den *Initial Fragment*-Header freigelassen.

- Die ersten *max_frag_initial* + *header_size* Bytes der 6LoWPAN-PDU werden in einem ersten Fragment übertragen. Diesen Bytes werden die vier Bytes des *Initial Fragment*-Headers vorangestellt.

- Die Variable *position* wird auf den Wert der Variablen *max_frag_initial* gesetzt.

- Die Variable *max_frag* wird auf den Wert [(*max_frame*-5)/8]*8 gesetzt. Es werden fünf Bytes für jeden *non-initial*-Fragment-Header frei gehalten.

- Solange der Wert *packet_size-position* größer ist als *max_frame*-5:

  – Die nächsten *max_frag* Bytes werden in einem *non-initial*-Fragment gesendet. Diesen Bytes wird der fünf-Byte lange *non-initial*-Fragment-Header vorangestellt. Der Wert in *datagramm_offset* wird mit dem Wert *position*/8 ausgefüllt.

  - Der Wert *position* wird um *max_frag* erhöht.

- Die verbleibenden Bytes werden in einem *non-initial*-Fragment übertragen. Der Wert in *datagram_offset* wird mit dem Wert *position*/8 ausgefüllt.

Die etwas überraschenden Rahmenbedingungen dieser Prozedur werden durch die Vorgabe bestimmt, dass das *datagram_offset* nur Positionen innerhalb des Paketes beschreiben kann, die ein Vielfaches von Acht sind. Die Fragmente füllen daher nicht unbedingt den gesamten verfügbaren Raum aus.

Der Empfang der Fragmente und ihr *Reassembly* können nach dieser Prozedur abgearbeitet werden:

- Ein Vier-Tupel/Quadrupel (endliche Liste mit 4 Elementen) muss kreiert werden, bestehend aus:
  - der Quell-Adresse,
  - der Ziel-Adresse,
  - der *datagram_size*,
  - und dem *datagram_tag*.
- Ein *Reassembly*-Buffer muss für dieses Vier-Tupel reserviert werden. Als Buffergröße wird die *datagram_size* verwendet. Der Buffer wird leer initialisiert, bis eine Liste von entsprechenden Fragmenten empfangen wurde.
- Für das *initial*-Fragment:
  - Muss die Variable *datagram_offset* auf null gesetzt werden.
  - Der 4 Byte Fragment-Header wird verworfen.
  - Sämtliches Dekoding und Dekomprimieren muss auf das enthaltene *Dispatch Byte* und jeden *Compressed Header* durchgeführt werden, so als wäre es ein komplettes Paket. Es wird die komplette *datagram_size* für die Rekonstruktion der Längen-Felder verwendet, so wie zum Beispiel die IPv6-Payload-Länge und die UDP-Länge.
  - Die temporäre Variable *data* wird auf den Inhalt und die Variable *frag_size* auf die Größe des sich ergebenden dekomprimierten Paketes gesetzt.
- Für das *non-initial*-Fragment:
  - Die Variable *datagram_offset* wird auf den Wert von dem Feld aus dem Paket gesetzt.
  - Der 5 Byte Fragment-Header wird verworfen.
  - Die temporäre Variable *data* wird auf den Inhalt und *frag_size* auf die Größe des Datenanteils von den empfangenen Fragment gesetzt. Das bedeutet der 5 Byte Header wird subtrahiert.
- Die Variable *byte_offset* wird auf den Wert der Variablen *datagramm_offset**8 gesetzt.
- Der Wert in *frag_size* wird nach folgenden Methoden überprüft:
  - Ist er ein Vielfaches von Acht (erlaubt weitere Fragmente in einer Reihe bis zum Ende), oder
  - *byte_offset*+*frag_size*=*datagram_size* (das bedeutet, das ist das letzte Fragment)

Sollte keine der beiden Bedingungen erfüllt sein, so wird der Vorgang als gescheitert betrachtet und abgebrochen (es wird keinen Weg geben, die verbleibenden Bytes auszufüllen).

- Falls ein Eintrag in der Liste der zuvor empfangenen Fragmente den Bereich *byte_offset*, *byte_offset+frag_size* abdeckt, wird folgendes überprüft:
  - Ist der überlappende Eintrag identisch, wird das aktuelle Fragment als ein Duplikat verworfen,
  - Falls nicht, muss der Vorgang abgebrochen werden.
- Falls ein Eintrag in der Liste der zuvor empfangenen Fragmente den Bereich *byte_offset*, *byte_offset+frag_size* nicht abdeckt, wird dieser Bereich zu der Liste hinzugefügt.
- Der Inhalt ab *data* wird an die Buffer-Position, beginnend an der Stelle auf die der Wert *byte_offset* zeigt, kopiert.
- Falls die Liste der Bereiche nun die ganze Spanne abdeckt, ist das *Reassembly* beendet und der Buffer enthält das wieder zusammengesetzte IPv6-Paket mit der Länge von *datagram_size*. Nun müssen alle abschließenden Verarbeitungsschritte ausgeführt werden, die für ein komplettes Paket nötig sind, so wie zum Beispiel die Rekonstruktion der bei der Komprimierung entfernten UDP-Checksumme.

Im Falle eines Fehlers durch Überdeckung der Intervalle muss der Reassembly-Buffer verworfen werden. Es ist möglich, das Reassembly/den Wiederzusammenbau mit einem neuen Reassembly-Buffer zu wiederholen. Die Prozedur wird viel einfacher, wenn die Anforderung der 6LoWPAN-Format-Spezifikation ignoriert wird, eine *Overlap Detection* durchzuführen. [Shelby u. a., 2009, S. 55-58]

2.3 Das Betriebssystem Contiki

Das Betriebsystem Contiki ist ein quelloffenes Betriebssystem und für den Einsatz auf vernetzten eingebetteten Systemen bzw. Sensorknoten gedacht. Die erste Version des Betriebssystems wurde 2003 veröffentlicht. Das Projektteam unter der Leitung von Adam Dunkels setzt sich aus Akademikern und Entwicklern der Industrie zusammen.

Contiki stellt Mechanismen bereit, die den Programmierer bei der Entwicklung von Software für Anwendungen auf Sensorknoten unterstützen. Darüber hinaus existieren Kommunikationsmechanismen, die es diesen Stationen erlauben, untereinander und mit der Außenwelt zu kommunizieren. Contiki bietet Bibliotheken zur Speicherverwaltung und für die Manipulation von verketteten Listen ebenso wie Kommunikationsabstraktionen und energiesparende Funknetzwerkmechanismen. Contiki verwendet ein Dateisystem, „Coffee“ genannt, dass es Programmen ermöglicht, Flash-ROM als einen üblichen Datenspeicher zu verwenden. Darüber hinaus bietet Contiki eine Sammlung von Simulatoren, welche das Entwickeln und Experimentieren mit Netzwerken bestehend aus Sensorknoten erleichtert.

Contiki war das erste Betriebssystem für Sensorknoten, welches eine IP-Kommunikation, basierend auf dem μIP TCP/IP-Stack, ermöglichte. Im Jahr 2008 übernahm das Betriebssystem Contiki μIPv6, den kleinsten IPv6-Stack der Welt. Der Speicheraufwand für den μIP- und den μIPv6-Stack ist gering - weniger als 5 kB für den μIP-Stack und ungefähr 11 kB für μIPv6. Das macht ihn geeignet für den Einsatz auf hardwarebegrenzten Plattformen wie einem Sensorknoten.

Viele Komponenten von Contiki sind in der Industrie weit verbreitet. Der μIP TCP/IP-Stack und sein größerer Vetter lightweight IP (lwIP) werden zurzeit von hunderten von Unternehmen in ihren Produkten, angefangen von Automobilmotoren und Flugzeugen bis hin zu globalen Frachtschiff-Trackingsystemen oder Satellitensystemen, eingesetzt. Die Protothread-Programmierabstraktion, die in Contiki verwendet wird, kommt auf Systemen wie digitalem Fernsehen, Set-Top-Boxen und hochperformanten Serverclustern zum Einsatz.

Sowohl das Contiki-Betriebssystem als auch die Anwendungen für das System werden in der Programmiersprache C implementiert. Der Einsatz dieser Programmiersprache ermöglicht eine hohe Portierbarkeit des Betriebssystems. Contiki kann daher auf mehr als zwölf unterschiedlichen Mikroprozessor- bzw. Mikrokontrollerarchitekturen eingesetzt werden. [Vasseur u. Dunkels, 2010, S. 129]

2.3.1 Contiki und μIP

Lange glaubte man, dass das Protokoll IP zu komplex und umfangreich sei, um auf Sensorknoten eingesetzt werden zu können. Maßgeblich dafür waren die begrenzten Hardwareressourcen der eingesetzten Mikrokontrollerplattformen.

Um über das Protokoll IP kommunizieren zu können benötigt jedes Gerät eine Softwareimplementierung des IP-Stacks. Das bedeutet, dass auf jedem Computer im Internet ein IP-Stack eingesetzt wird, der Bestand der Betriebsysteme wie bspw. Microsoft Windows, Linux oder Mac OS ist. Sensorknoten sind im Gegensatz zu gewöhnlichen Computersystemen, bei denen für den IP-Stack verhältnismäßig viel Arbeitsspeicher reserviert wird, mit begrenzten Ressourcen ausgestattet. Wo für ein Linux-System näherungsweise ein Megabyte für den ein- und ausgehenden Datenverkehr bereitgestellt wird, sind auf einem Sensorknoten nur wenige Kilobytes verfügbar.

Verallgemeinert sind die Aufgaben des IP-Stacks einfach zu beschreiben. Es werden Pakete mit dem *Communication Device*-Treiber ausgetauscht. Die Anwendungen kommunizieren mit dem IP-Stack über das Betriebssystem oder auf direktem Weg. Der IP-Stack analysiert den Header eines empfangenen Paketes, gewinnt die Nutzdaten aus dem Paket und übergibt sie an die Anwendung. Möchte die Anwendung Daten senden, gibt sie diese an den IP-Stack weiter. Der Stack kreiert ein Paket, indem er den nötigen Header voranstellt und dieses Paket dem *Communication Device*-Treiber für die Übertragung durch das Kommunikationsgerät übergibt. Zusätzlich, um auf direkte Anfragen der Anwendungen und auf ankommende Pakete reagieren zu können, beschäftigt sich der IP-Stack mit periodischer Protokollverarbeitung wie bspw. dem Ausführen von Retransmissions.

Der μIP TCP/IP-Stack ist eine Implementierung des IP-Stacks, eigens dafür geschaffen, um mit den limitierten Hardwareressourcen auf Sensorknoten und eingebetteten Systemen auszukommen. Die erste Version wurde im September 2001 unter einer Open Source Lizenz veröffentlicht, die es ermöglichte, die Software frei in kommerziellen und nicht kommerziellen Systemen zu nutzen. Der μIP-Stack wurde von der Industrie sehr gut angenommen und ist heutzutage in Überwachungssystemen für Öl-Pipelines, globalen Container-Trackingsystemen, Automotoren und Pico-Satelliten (ein Satellitensystem von ca. 1 kg Gewicht) zu finden. μIP ist die wichtigste Komponente für die IP-Kommunikation des Betriebssystems Contiki, obwohl der μIP-Stack als eigenständiges Softwarepaket verwendet werden kann und auch wird. μIP wird im Rahmen des Contiki-Projektes weiterentwickelt.

Die Anforderungen von μIP an den Arbeitsspeicher sind sehr gering und es werden die Protokolle der IP-Protokoll-Familie auf den Ebenen der Netzwerkschicht und der Transportschicht reali-

siert. In seiner Grundkonfiguration wird ungefähr ein Kilobyte RAM und wenige Kilobytes ROM verwendet. In dieser Konfiguration ist der Einsatz der Protokolle IP, ICMP, UDP und TCP möglich. Die exakte Größe des Programmcodes ist hierbei von der verwendeten Prozessorarchitektur abhängig. Es ist möglich, den benötigten RAM weiter zu reduzieren, allerdings auf Kosten der Konformität zum Standard. Die kleinste Konfiguration erfordert ungefähr 100 Bytes an RAM, aber sie ist nicht zwangsläufig standardkonform. Darüber hinaus beeinflusst der bereitgestellte Arbeitsspeicher auch den erreichbaren Datendurchsatz. Für viele Anwendungsfälle mag jedoch ein geringer Aufwand bezüglich des Arbeitsspeichers wichtiger sein als eine hohe Datenrate.

Die erste Version des μIP-Stacks ermöglichte nur IPv4-Kommunikation, aber 2008 wurde er durch das Unternehmen Cisco Systems um die IPv6-Fähigkeit erweitert. Diese Erweiterung wurde von Julien Abeillé und Mathilde Durvy entwickelt. Der μIPv6-Stack war der erste Stack, der alle IPv6 Anforderungen erfüllte, was es ihm erlaubte, das „IPv6 Ready“-Logo (Abbildung in Abschnitt 2.3.1, Quelle[1]) zu tragen/nutzen. Die μIPv6-Erweiterungen wurden als Open Source Software veröffentlicht und dem Betriebssystem Contiki hinzugefügt, wodurch sie einer breiten Masse zugänglich waren.

μIP bedient sich dreier Methoden, um die Größe des Programmcodes und den Speicherverbrauch zu reduzieren: eine ereignisorientierte Programmierschnittstelle, ein absichtlich einfach gehaltenes Bufferverwaltungsmodell und eine speichereffiziente TCP-Implementierung. [Vasseur u. Dunkels, 2010, S. 167-169]

2.3.2 Grundlagen der Verarbeitung

Die grundlegende Arbeitsweise des μIP-Stacks ist unkompliziert. Der μIP-Stack hat drei Aufgaben. Er verarbeitet Pakete, die vom *Communikation Device*-Treiber geliefert werden, er bearbeitet Anfragen von Anwendungen und er nimmt periodisch besondere Aufgaben wahr. Das μIP-*Forwarding*-Modul ist hierbei für das Weiterleiten des Datenverkehrs zu anderen Stationen zuständig. Das *Forwarding*-Modul stellt Anfragen an das *Routing Protokoll*-Modul, um zu ermitteln, wohin die Pakete weitergeleitet werden sollen.

Die Eingabeverarbeitung beginnt, wie in der Abbildung 2.40 zu sehen, wenn der *Communikation Device*-Treiber ein Paket empfangen hat. Der Treiber ruft die Eingabeverarbeitungsfunktion von μIP auf. Die Funktion analysiert den Header, bestimmt ob das Paket Anwendungsdaten enthält und falls dies der Fall ist, gibt sie die Daten an die Anwendung weiter. Die Anwendung kann die eingehenden Daten quittieren. Der Fall wird von dem Teil von μIP abgearbeitet, der für die Ausgaben verantwortlich ist.

Die Ausgabeverarbeitung wird durchgeführt, nachdem die Anwendung von μIP aufgerufen wurde und die Anwendung Daten für den μIP-Stack zum Versenden bereithält. Der Programmcode für die Ausgabeverarbeitung fügt dem zu sendenden Paket den Protokoll-Header hinzu und übergibt es dem *Communicaction Device* für die Übertragung.

Die zyklische Verarbeitung wird ausgeführt, um zeitgesteuerte Aktionen wie Retransmissions anzustoßen. Der zyklische Verarbeitungsmechanismus in μIP wurde absichtlich einfach gestaltet. Die zyklische Verarbeitungsfunktion von μIP wird regelmäßig aufgerufen, um zu überprüfen, ob eine Retransmission nötig ist. Falls dies der Fall ist, erzeugt sie das Paket, welches erneut übertragen werden soll und übergibt es dem *Communikation Device*-Treiber, welcher es dann versendet.

[1] http://www.ipv6ready.org/ (Abrufdatum 30.9.2012)

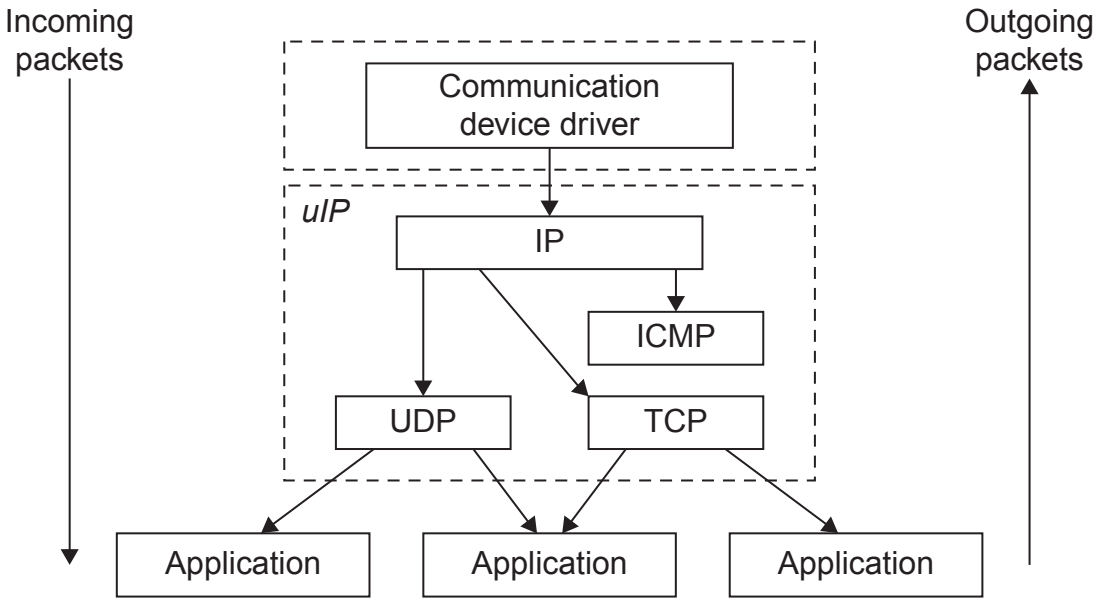


Abbildung 2.40: Die grundlegende Arbeitsweise des μIP-Stacks, entnommen aus [Vasseur u. Dunkels, 2010, S. 170]

Forwarding und *Routing* werden getrennt voneinander ausgeführt. *Forwarding* ist der Vorgang des Weiterleitens eines empfangenen Paketes an einen Nachbarn. *Routing* hingegen ist der Vorgang, der durchgeführt wird, um den Kommunikationspartner für das *Forwarding* zu bestimmen. Das μIP-*Forwarding*-Modul führt eine Tabelle mit Zieladressen und den Adressen der *next-hop*-Nachbarn. Routingprotokolle, die typischerweise oberhalb der Protokolle UDP oder TCP implementiert werden, erstellen die *Forwarding*-Tabelle auf Grundlage der empfangenen Routinginformationen. Ergänzende Informationen zu Contiki und μIP sind im Anhang in Abschnitt A.2 zu finden. [Vasseur u. Dunkels, 2010, S. 169]

2.3.3 Die μIP-Programmierschnittstelle

Die Programmierschnittstelle (Application Programming Interface (API)), stellt den Anwendungen zwei Möglichkeiten bereit, mit dem TCP/IP-Stack zu interagieren. Die bekannteste API für IP-Stacks ist die Berkeley Software Distribution (BSD)-Socket-API. Die BSD-Socket-API wird in den meisten UNIX-Systemen eingesetzt und hat die Entwicklung der Microsoft Windows WinSock-API stark beeinflusst. Die Socket-API wurde für ein multithreadingbasiertes Programmiermodell entwickelt. Dieses Modell belastet die Speicherressourcen des Computersystems stark – dieses Verhalten ist daher nicht ideal für Sensorknoten und ihren limitierten Arbeitsspeicher.

Anstelle der multithreadingbasierten Socket-API verwendet μIP eine ereignisgesteuerte API. Der Einsatz einer ereignisgesteuerten API im Zusammenhang mit Sensorknoten bietet einige Vorteile. Ereignisgesteuerte Mechanismen verbrauchen weniger Speicherressourcen als multithreadingbasierte. Die ereignisgesteuerte API benötigt, im Gegensatz zu einer traditionellen BSD-Socket-API, keinen zusätzlichen Bufferspeicher zwischen dem μIP-Stack und der Anwendung. Dadurch wird der Speicherverbrauch weiter reduziert. Darüber hinaus weisen ereignisgesteuerte APIs eine höhere Effizienz bezüglich der Ausführungszeit gegenüber multithreadingbasierten APIs auf. Aufgrund der niedrigeren Prozessorgeschwindigkeiten in Sensorknoten ist dies sehr vorteilhaft. Da es Anwendungen nun möglich ist, sofort auf Ereignisse wie eintreffende Daten und Verbindungsanfragen zu reagieren, können auch auf Systemen mit begrenzten Hardwareressourcen kurze Reakti-

onszeiten erreicht werden. Die ereignisgesteuerte API wird hauptsächlich für TCP-Verbindungen verwendet, kann aber auch von UDP-basierten Anwendungen genutzt werden.

Obwohl die ereignisgesteuerte API für die meisten Anwendungsfälle geeignet ist, gibt es Anwendungen die von einer sequentiellen API profitieren. Darum bietet μIP optional eine sequentielle BSD-socketähnliche API basierend auf Protothreads. Diese API wird als Protosockets bezeichnet und ermöglicht das Implementieren von Programmen nach einem Top-Down-Entwurf. Die Protosockets-API stellt Bufferspeicher für Retransmissions bereit, um dem Programmierer das Erzeugen der für das erneute Senden benötigten Daten zu ersparen. Dementsprechend werden höhere Anforderungen an den Arbeitsspeicher gestellt. [Vasseur u. Dunkels, 2010, S. 176]

2.3.4 Der Rime-Stack

Die bisherige Entwicklungsarbeit zeigte, dass die traditionelle Einteilung der Schichten für die Kommunikation in Sensornetzwerken mit enormen Einschränkungen verbunden ist. Das macht eine Optimierung zwischen den Protokollebenen notwendig.

In der Vergangenheit hielt man Mechanismen für die Aggregation der Daten innerhalb des Kommunikationsstacks für zu komplex und vertrat die Ansicht, dies würde zu unsicheren und schwer steuerbaren Systemen führen. Das inspirierte das Entwicklerteam um Adam Dunkels zu der Umsetzung von Rime. Bei Rime handelt es sich um einen Kommunikationsstack für Sensornetzwerke in mehreren Ebenen. Durch ihn ist es möglich, die Schichten unterhalb der Vermittlungsschicht des OSI-Referenzmodells auf einfache Art und Weise zu realisieren. Der Hauptgedanke hinter Rime ist die Reduzierung des Implementierungsaufwandes für die Protokolle von Sensornetzwerken mit minimal erhöhten Hardwareressourcen.

Der Rime-Stack verwendet einen einzigen Buffer für das Senden und Empfangen der Rahmen und die Anforderungen an den Arbeitsspeicher sind gering. Der Rime-Stack wird vom Betriebssystem Contiki für die Kommunikation auf Ebene 2 des OSI-Referenzmodells eingesetzt. Während der Implementierung des Protokollentwurfs existieren beim Empfang und beim Senden von Datenrahmen Berührungspunkte mit diesem Stack. Eine kurze Abhandlung[2] über den Rime-Stack wurde von Adam Dunkels online bereitgestellt. Darüber hinaus ist im Verzeichnis „\doc“ des Betriebssystems Contiki eine Beschreibung des Rime-Stacks zu finden. [Dunkels, S. 1]

[2] http://www.sics.se/ adam/dunkels07rime.pdf (Abrufdatum 09.10.2012)

Kapitel 3

Stand der Technik

In diesem Kapitel wird der allgemeine Hardwareaufbau eines Sensorknotens, oft auch als „Smart Object“ bezeichnet, beschrieben. In diesem Zusammenhang wird die Art und Weise der Umsetzung der 6LoWPAN-Technologie in Sensorknoten erläutert. Darüber hinaus werden die für die Lösung der Aufgabenstellung vorgesehenen Mikrokontrollerplattformen und das Programmiergerät für den dort integrierten Festspeicher vorgestellt. Ergänzende Informationen sind im Anhang in den Abschnitten A.3 und A.4 zu finden.

3.1 Allgemeiner Aufbau von Sensorknoten

Sensorknoten, sogenannte „Smart Objects“ unterscheiden sich in Aufbau und Verhalten von konventionellen Computersystemen. Das liegt unter anderem in ihren limitierten Ressourcen von Arbeitsspeicher und Rechenleistung begründet. Ein solches „Smart Object“ ist aus einer Vielzahl von Hardwarekomponenten aufgebaut, es lassen sich jedoch vier zentrale Baugruppen unterscheiden:

- **Das Kommunikationsgerät:** Über diese Baugruppe ist es dem Sensorknoten möglich, Daten/Nachrichten mit anderen Geräten auszutauschen. Üblicherweise werden Funktransceiver mit einer Antenne oder eine drahtgebundene Schnittstelle verwendet.
- **Der Mikrokontroller:** Bei dieser Baugruppe handelt es sich um einen kleinen Mikroprozessor, auf dem die Software des Sensorknotens ausgeführt wird. Hierdurch wird im Wesentlichen das Verhalten der Station bestimmt.
- **Eine Auswahl von Sensoren und Aktoren:** Hierüber interagiert das Gerät mit seiner Umwelt.
- **Die Spannungsquelle:** Sie wird benötigt, um den Sensorknoten und die verwendeten elektrischen Schaltkreise mit Spannung zu versorgen.

[Vasseur u. Dunkels, 2010, S. 119]

3.1.1 Das Kommunikationsgerät

Das Kommunikationsgerät ermöglicht es dem Sensorknoten, mit anderen Geräten zu kommunizieren. Für drahtlose Geräte wird typischerweise ein Funktransceiver, bestehend aus Sender und Empfänger, eingesetzt. Für Geräte mit drahtgebundener Schnittstelle werden Technologien wie Ethernet oder Powerline verwendet.

Die verschiedenen Arten von Funktransceivern stellen unterschiedliche Verarbeitungskapazitäten bereit. Die einfachsten Funktransceiver können nur einzelne Bits senden oder empfangen, während leistungsfähigere Versionen in der Lage sind, die Informationen Paketen zuzuordnen und mit

einem Header zu versehen. Wurden Sicherheitsmechanismen aktiviert, sind sie sogar in der Lage, die Daten unter Verwendung verschiedener Verschlüsselungsmechanismen zu sichern.

Unter den Hardwarekomponenten eines Sensorknotens ist die drahtlose Schnittstelle für gewöhnlich die Baugruppe mit dem höchsten Energieverbrauch. Verglichen mit dem Energieverbrauch der Mikrokontroller, Sensoren und Aktoren verbrauchen sie oft das Zehnfache an Energie. Die Ursache hierfür liegt in der benötigten Rechenleistung für das Modulieren und Demodulieren des Funksignales.

Bei low-power Funkschnittstellen entfällt der kleinste Teil des Energiebedarfs auf die Übertragung des Signales über die Luft. Daraus folgt, dass für das Senden ebenso viel Energie benötigt wird wie für den Empfang. Dementsprechend ist der Energiebedarf eines Gerätes, das im Leerlauf den Funkkanal „belauscht“ genauso hoch, als würde es Daten senden. Um Energie zu sparen muss die Baugruppe also deaktiviert werden – dann ist der Sensorknoten jedoch nicht in der Lage, Daten zu empfangen. In Multihop-Netzwerken werden daher verschiedene Mechanismen verwendet, um die Stationen zu synchronisieren aber dennoch Energie zu sparen. Die Geräte, die diese Einschaltmechanismen verwenden, können die Funkschnittstelle die meiste Zeit deaktiviert lassen und trotzdem Daten mit anderen Sensorknoten austauschen. [Vasseur u. Dunkels, 2010, S. 121-122]

3.1.2 Der Mikrokontroller

Der Mikrokontroller verleiht dem Gerät seine scheinbare Intelligenz. Auf ihm läuft die Software des Sensorknotens und er ist ebenso dafür verantwortlich, die Funkschnittstelle mit den Sensoren und Aktoren zu verbinden. Ein Mikrokontroller besteht im Wesentlichen aus einem Mikroprozessor mit integriertem Speicher, verschiedenen Timern und Hardwarekomponenten, um externe Geräte wie Sensoren, Aktoren und Funktransceiver anschließen zu können. Äußerlich sieht ein Mikrokontroller wie ein gewöhnlicher Mikrochip mit Plastikgehäuse und metallenen Anschlüssen aus.

Mikrokontroller sind weit verbreitet und die am häufigsten eingesetzte Art von Mikroprozessoren. Ungefähr die Hälfte dieser Mikroprozessoren sind 8 Bit Prozessoren, die im Normalfall eine Speichergröße von 65536 Bytes verwalten können. Da die in Sensorknoten eingesetzten Mikrokontroller möglichst energie- und kostensparend sein sollen, sind sie deutlich kleiner als die Prozessoren, die bspw. in Personalcomputern eingesetzt werden. Für gewöhnlich besitzt der Mikrokontroller eines Sensorknotens nur einige Kilobyte an Arbeitsspeicher und läuft mit einer Taktfrequenz von wenigen Megahertz. Ein moderner Personalcomputer wird mit einigen Gigabytes an Arbeitsspeicher ausgeliefert und arbeitet mit einer Taktfrequenz von einigen Gigahertz. Die Tabelle 3.1 zeigt die Eckdaten einiger in Sensorknoten verwendeter Mikrokontroller.

In Mikrokontrollern werden zwei Sorten von Speicher eingesetzt, ROM und RAM. Der ROM wird verwendet, um den auf dem Sensorknoten eingesetzten Programmcode zu speichern. Dieser Programmcode bestimmt das Verhalten des Gerätes. Der RAM wird für temporäre Daten wie Programmvariablen und Pufferspeicher für empfangende und zu sendende Datenpakete eingesetzt. Der Inhalt des ROM wird durch den Hersteller während des Fertigungsprozesses bestimmt und in der Regel nach der Auslieferung nicht mehr verändert. Die meisten neueren Mikrokontroller ermöglichen es jedoch, den Inhalt des ROM neu zu beschreiben. Dadurch ist es möglich, die Software eines Sensorknotens, der sich bereits im Einsatz befindet, zu aktualisieren.

Zusätzlich zu den Speicherbaugruppen wird auf einem Mikrokontroller ein Satz von Timern sowie Mechanismen für das Interagieren mit externen Geräten wie der Funkbaugruppe, den Sensoren

Tabelle 3.1: In Sensorknoten eingesetzte Mikrokontrollerplattformen, Inhalt übernommen von [Vasseur u. Dunkels, 2010, S. 122]

Name	**Hersteller**	**RAM (kB)**	**ROM (kB)**	**Stromaufnahme (aktiv/inaktiv), mA**
MSP430xF168	Texas Instruments	10	48	2/0,001
AVR ATmega128	Atmel	8	128	8/0,02
8051	Intel	0,5	32	30/0,005
PIC18	Microchip	4	128	2,2/0,001

und Aktoren eingesetzt. Die Timer können durch die auf dem Mikrokontroller eingesetzte Software frei genutzt werden. Die externen Geräte sind mit dem Mikrokontroller über seine Pins fest verdrahtet. Die Software greift auf die Geräte typischerweise über einen seriellen Port oder seriellen Bus zu. Die meisten Mikrokontroller ermöglichen es so genannten Universal Synchronous/Asynchronous Receiver/Transmitter (USART) mit ihren seriellen Ports zu kommunizieren. Manche USART können so konfiguriert werden, dass sie als ein Serial Peripheral Interface (SPI) arbeiten. Sie können so mit den Sensoren und Aktoren des Sensorknotens kommunizieren. [Vasseur u. Dunkels, 2010, S. 122-123]

3.1.3 Realisierung der 6LoWPAN-Technologie

6LoWPAN ist eine Netzwerktechnologie, die in Form des 6LoWPAN-Protokollstacks in eingebetteten Systemen integriert wird. Für die Umsetzung dieser Protokolllösung werden typischerweise drei Lösungsmodelle angewandt: Ein-Chip-, Zwei-Chip- und Netzwerkprozessor-Lösungen. Diese Lösungen bieten, je nach Anwendungsfall, gewisse Vorteile. Bei der Ein-Chip-Lösung handelt es sich um einen Funktransceiver mit integriertem Mikrokontroller, wohingegen sich bei der Zwei-Chip-Lösung der Mikrokontroller und der Funktransceiver auf zwei unterschiedlichen Mikrochips befinden. Die Netzwerkprozessor-Lösung nutzt einen Funktransceiver mit Protokollstack, der von einem separaten Anwendungsprozessor angesteuert wird.

Die Ein-Chip-Lösung

In Geräten bei denen die Minimierung der Kosten und der Größe eine wesentliche Rolle spielt und gleichzeitig die Komplexität der integrierten Anwendung gering ist, bieten sich Ein-Chip-Lösungen an.

In einer Ein-Chip-Lösung (System-on-a-Chip (SoC)) wird eine Funktechnologie verwendet, bei welcher der Funktransceiver und der Mikrokontroller gemeinsam mit dem Flash-Speicher und anderen Peripheriegeräten integriert sind. Zusätzlich zum SoC wird nur eine kleine Menge weiterer Bauteile benötigt, um einen unabhängigen drahtlosen Sensorknoten zu erstellen. Dazu zählen die Antenne, diskrete Bauteile für die Frequenzanpassung, Quarzoszillatoren, Sensoren, Aktoren und die Spannungsquelle. Die Abbildung 3.1 zeigt ein Blockdiagramm für die Ein-Chip-Bauweise.

Die gesamte Software des Geräts wird auf dem SoC-Mikrokontroller ausgeführt und ist im Flash gespeichert. Dies beinhaltet die Hardware-Treiber, ein mögliches Betriebssystem (OS) für eingebettete Systeme, den 6LoWPAN-Protokollstack und die Geräteanwendungen. Beispiele für SoC-Hardwarelösungen sind die Plattformen TI CC2530, TI CC1110 und Jennic JN5139. Die meisten 6LoWPAN-Protokollstacks können heutzutage in Ein-Chip-Lösungen integriert werden.

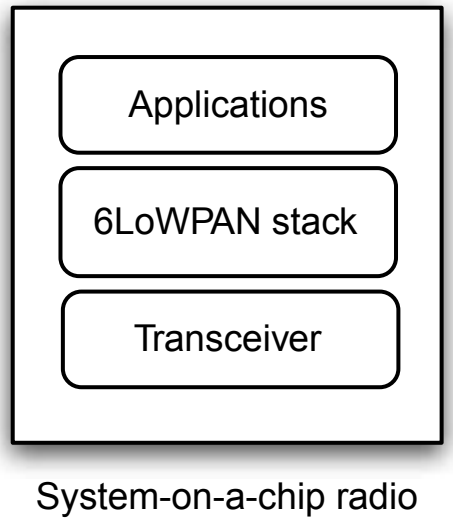


Abbildung 3.1: Ein-Chip-Lösung, entnommen aus [Shelby u. a., 2009, S. 150]

Diese Methode hat den Nachteil, dass die Integration sämtlicher Software auf einem Mikrokontroller die Komplexität und Entwicklungszeit erhöht. Da ein SoC kleine, spezialisierte Mikrokontroller ohne Speicherschutz verwendet, ist die Integration von Anwendungen mit Protokollstack, Basistreibern und Betriebssystem schwieriger, da jede Konfiguration eingehend getestet werden muss. Dieser Ansatz schränkt die Wiederverwertbarkeit der für dieses SoC geschriebenen Anwendungen aufgrund spezialisierter Compiler und Stack-Lösungen ein. [Shelby u. a., 2009, S. 150-151]

Die Zwei-Chip-Lösung

In Geräten, bei denen man sich bereits für eine Mikrokontroller-Architektur entschieden hat, oder bei denen die Anwendungskomplexität und die Leistungsanforderungen höher sind, sollten Zwei-Chip-Lösungen eingesetzt werden. Die Abbildung 3.2 zeigt das Blockdiagramm der Zwei-Chip-Bauweise.

Bei einer Zwei-Chip-Lösung werden der Anwendungsprozessor und der Funktransceiver in zwei unterschiedlichen Mikrochips (Integrated Circuit (IC)) untergebracht. Eine Variation dieses Lösungsansatzes, bei welcher der Transceiver auch den Protokollstack beinhaltet, wird Netzwerkprozessor genannt und im nächsten Abschnitt erläutert.

Der Anwendungsprozessor kommuniziert mit dem Funktransceiver für gewöhnlich über einen USART oder ein SPI (siehe Abschnitt 3.1.2). Der 6LoWPAN-Protokollstack wird gemeinsam mit der Anwendung und dem Betriebssystem auf dem Mikrokontroller integriert. Der Entwickler hat große Freiheiten hinsichtlich der eingesetzten Mikrokontrollerplattform, die spezielle integrierte Steuerungs-, Signalverarbeitungs- oder Leistungsanforderungen haben kann.

Die Zwei-Chip-Lösung kann zum Einsatz kommen, wenn ein bestimmter Mikrokontroller mit definierter Größe an Flash- und Arbeitsspeicher verwendet werden soll. Beispiele für gängige Funktransceiver wären der TI CC2520 und der Atmel AT86RF231.

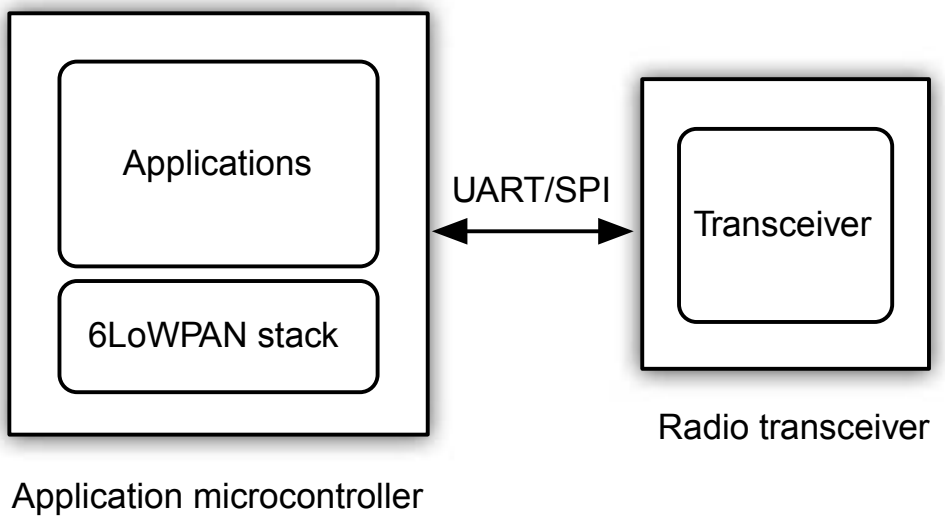


Abbildung 3.2: Zwei-Chip-Lösung, entnommen aus [Shelby u. a., 2009, S. 151]

Der Nachteil dieser Methode ist, dass der 6LoWPAN-Protokollstack und die gewünschte Anwendung im selben Mikrokontroller integriert werden müssen. Genau wie bei der Ein-Chip-Lösung kann diese Integration umfassende Entwicklungsarbeit bedeuten. Viele Protokollstacks sind außerdem auf einen bestimmten Mikrokontroller oder Funktransceiver zugeschnitten. Dies erschwert einen Wechsel der Mikrokontrollerplattform. [Shelby u. a., 2009, S. 151]

Die Netwerkprozessor-Lösung

Eine Zwei-Chip-Netzwerkprozessor-Lösung eignet sich für Projekte, bei denen bereits ein Design oder eine Anwendungssoftware vorliegt oder für neue Geräte, bei denen ein geringer Entwicklungsaufwand gewünscht wird. Ebenso wie die vorangegangenen Zwei-Chip-Lösungen ist auch der Netzwerkprozessor ein separater IC. Jedoch wurde in diesem Fall der 6LoWPAN-Protokollstack in den Funktransceiver integriert. Diese Bauweise wird in Abbildung 3.3 dargestellt.

Ein Netzwerkprozessor auf einem SoC-Funktransceiver beinhaltet oftmals eine Netzwerkprozessor-Firmware. Für gewöhnlich wird ein viel kleineres SoC (weniger Flash-Speicher und RAM) benötigt als bei Ein-Chip-Lösungen, da auf dem IC keine Anwendungen ausgeführt werden. In der Kommunikationsbranche bezieht sich der Begriff Netzwerkprozessor auf eine Central Processing Unit (CPU), die auf die Verarbeitung des Netzverkehrs spezialisiert ist (bspw. in einem IP-Router). In der Low-Power Drahtlos Netzwerk Branche wird der Begriff benutzt um SoC-Netzwerkprozessor-Lösungen mit integriertem Funktransceiver zu beschreiben.

Die Kommunikation mit einem Netzwerkprozessor findet gewöhnlich über einen USART oder ein SPI statt. Häufig wird dafür ein zusätzliches socket-basiertes Protokoll verwendet. Mit diesem Ansatz muss das 6LoWPAN-Netzwerk in der Applikation des Mikrokontrollers nicht gesondert berücksichtigt werden. Hier reicht es, das für die Kommunikation über das lokale Interface verwendete Protokoll zu implementieren. Die Netzwerkprozessor-Lösung wird oft in sogenannten Edge-Routern eingesetzt und ist einfach in Betriebssysteme wie Linux zu integrieren.

Der Nachteil dieser Lösung ist, dass zwei ICs benötigt werden. Bei Geräten, die mit möglichst geringem Kostenaufwand realisiert werden sollen, ist dieses Modell nicht anwendbar. Dement-

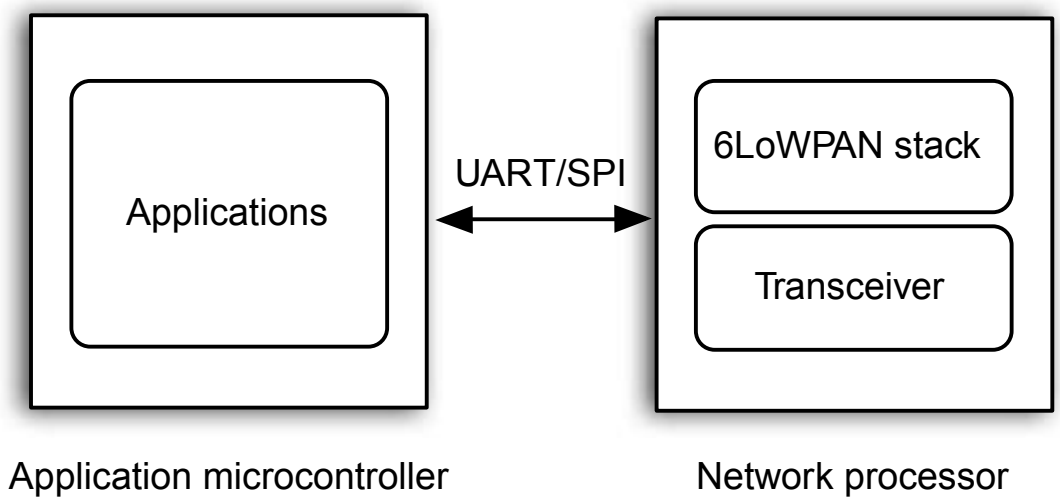


Abbildung 3.3: Netwerkprozessor-Lösung, entnommen aus [Shelby u. a., 2009, S. 152]

sprechend sind die Preise für Netzwerkprozessoren etwas höher als bei Funktransceivern, da sie einen Mikrokontroller, Flash- und Arbeitsspeicher besitzen. [Shelby u. a., 2009, S. 151-152]

3.2 Die Entwicklungsplattform AVR RZ RAVEN

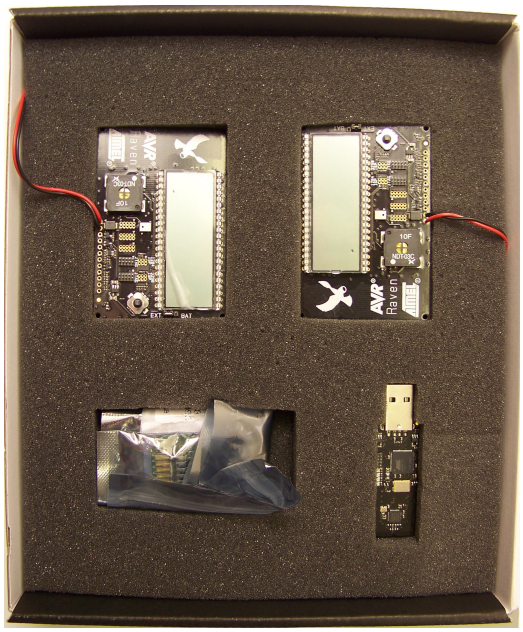
Für die Umsetzung der Aufgabenstellung ist der Einsatz des Atmel AVR RZ RAVEN 2.4 GHz Wireless Evaluation Kits geplant. Dieses Kit besteht aus zwei AVR RZ RAVEN Boards, einem AVR RZ USB Stick und einem Adapter für die Programmierung des nichtflüchtigen Speichers mit einem Gerät wie dem AVR JTAGICE mkII. Diese Entwicklungsplattform ermöglicht das Erstellen und Analysieren einer Vielzahl von Softwarelösungen basierend auf den Protokollen IEEE 802.15.4, 6LoWPAN und ZigBee. Die folgenden Abschnitte geben einen groben Überblick über diese Hardwareplattform.

3.2.1 Das AVR RZ RAVEN Board

Auf dem AVR RZ RAVEN Board werden zwei Mikrokontroller und ein Funktransceiver eingesetzt. Der Mikrokontroller ATmega1284P, der mit dem Atmel AT86RF230 Funktransceiver verbunden ist, enthält einen großen Teil der Software und den Protokoll-Stack für die Kommunikation über die drahtlose Schnittstelle.

Der zweite Mikrokontroller trägt die Bezeichnung Atmel ATmega3290. Er stellt die Benutzerschnittstellen, Aktoren und Sensoren bereit. Er realisiert die Ausgabe auf dem Liquid Crystal Display (LCD). Beide Mikrokontroller gehören zu Atmels stromsparender AVR picoPower®-Familie, die einen minimalen Energieverbrauch bei einer Versorgungsspannung von 1.8V sicherstellt. Für die Kommunikation zwischen den Mikrokontrollern und dem Funktransceiver wird ein USART verwendet.

Als Funktransceiver wird der IC Atmel AT 86RF230 eingesetzt. Für diesen Einsatzfall ist er aufgrund seines geringen Energieverbrauchs und seiner Flexibilität besonders geeignet. Viele Funktionen des Standards IEEE 802.15.4 wurden bereits in die Hardware integriert. Eine Schleifenantenne mit einer Impedanz am Einspeisepunkt von 100 Ω ermöglicht eine Verstärkung von ungefähr



(a) Inhalt

(b) Umverpackung

Abbildung 3.4: Das Atmel AVR RZ RAVEN 2.4 GHz Wireless Evaluation Kit

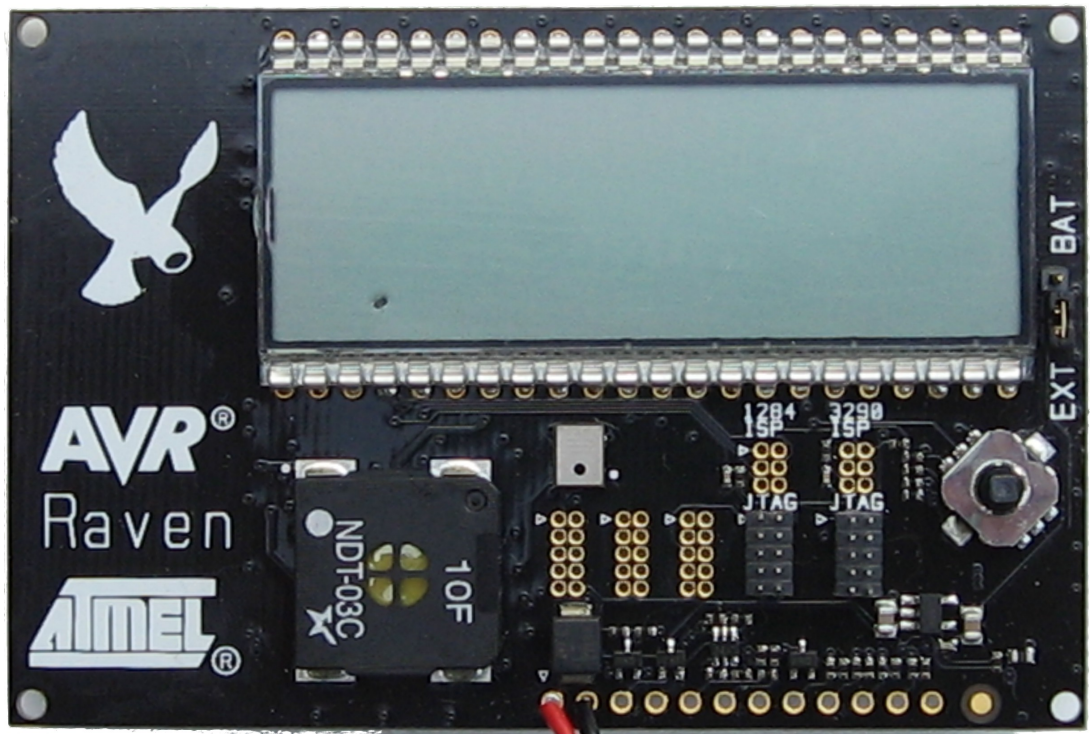


Abbildung 3.5: Das AVR RZ RAVEN Board, Oberseite

5 dB.

Darüber hinaus ist das Board mit einem LCD-Segment, Sensoren, Aktoren und verschiedenen Benutzerschnittstellen ausgestattet. Hierzu zählen ein Vier-Kontakt-Schalter für die Benutzereingaben, ein Mikrophon, ein Lautsprecher und die Benutzerschnittstellen J401 und J201 bis J203 der Mikrokontroller ATmega3290P und ATmega1284P. Über die JTAG-Programmierschnittstellen (J204 für den ATmega 1284P und J301 für den ATmega 3290P) können die nichtflüchtigen Speicher der beiden Mikrokontroller programmiert werden.

Als Spannungsquelle des Boards kann sowohl eine Batterie als auch eine externe Spannungsquelle mit einer Spannung von 5 bis 12 V Gleichspannung dienen. Die Wahl der Spannungsquelle erfolgt über eine Brücke/Jumper auf der rechten Seite des Boards (Draufsicht).

3.2.2 Der AVR RZ USB Stick

Der AVR RZ USB Stick ist im Wesentlichen aus dem bereits vorgestellten Funktransceiver Atmel AT86RF230 und dem Mikrokontroller Atmel AT90USB1287 aufgebaut. Der Funktransceiver ist in diesem Fall an eine gefaltete Dipolantenne mit einer Verstärkung von 0 dB angeschlossen. Der Mikrokontroller Atmel AT90USB1287 verwaltet die USB-Schnittstelle, greift auf den Funktransceiver zu und realisiert die Netzwerkprotokolle über eine Softwareimplementierung. Über die USB-Schnittstelle wird auch die Spannungsversorgung des AVR RZ USB Stick realisiert.

Eine serielle Schnittstelle ermöglicht die Kommunikation mit dem Mikrokontroller Atmel AT90USB1287 und die JTAG-Programmierschnittstelle erlaubt die Programmierung des nichtflüchtigen Speichers.

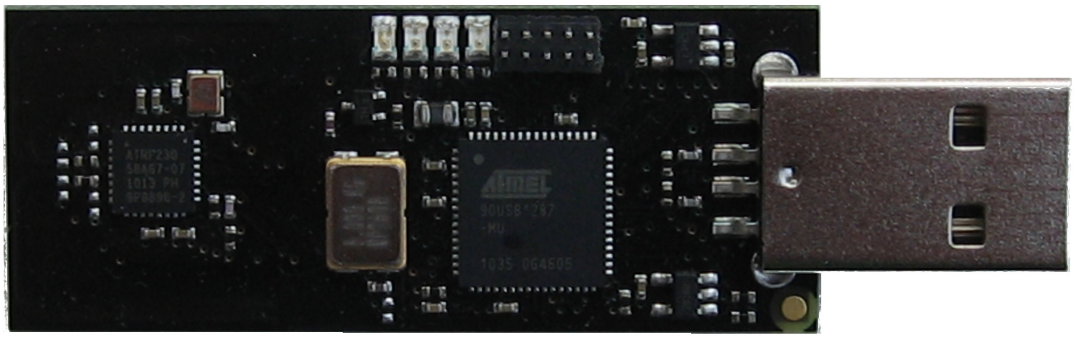


Abbildung 3.6: Der AVR RZ USB Stick, Oberseite

Der AVR RZ USB Stick besitzt kein Ausgabegerät in Form eines LCD. Informationen über den Betriebszustand des Gerätes und die drahtlose Kommunikation mittels des Funktransceivers werden über die vier verschiedenfarbigen Light-Emitting Diode (LED)s gegeben.

Der Hersteller stellt eine umfangreiche Dokumentation[1] der Entwicklungsplattform AVR RZ RAVEN auf seinem Internetauftritt zur Verfügung.

[1] http://www.atmel.com/tools/RZRAVEN.aspx?tab=documents (Abrufdatum 15.9.2012)

3.2.3 Das Programmiergerät AVR JTAGICE mkII

Die Entwicklungsplattform AVR RZ RAVEN stellt für die Programmierung der nichtflüchtigen Speicher (Flash-Speicher und Electrically Erasable Programmable Read-Only Memory (EEPROM)) die Joint Test Action Group (JTAG)-Schnittstelle bereit. Die Programmierung des AVR RZ RAVEN Boards ist alternativ über eine zusätzliche In-circuit Serial Programming (ISP)-Schnittstelle möglich. Trotz preisgünstigerer Alternativen wie dem Atmel AVR Dragon ist für die Realisierung der Aufgabenstellung der Einsatz einer Lösung aus dem mittleren Preissegment in Gestalt des Programmers AVR JTAGICE mkII geplant. Dieser Programmer unterstützt neben der JTAG-Schnittstelle noch alternative Modi wie bspw. ISP für das Programmieren und Debuggen einer Vielzahl von Atmel Mikrokontrollern.

(a) Draufsicht (b) Seitenansicht

Abbildung 3.7: Das Programmiergerät AVR JTAGICE mkII

Das Gerät wird über die RS232- oder alternativ über die USB-Schnittstelle angeprochen. Wird die serielle Schnittstelle RS232 verwendet, ist der Anschluss einer separaten Spannungsversorgung vorgesehen. Der AVR JTAGICE mkII ermöglicht das On-Chip-Debugging und wird durch die Entwicklungsumgebung Atmel® Studio ergänzt. Die Liste der unterstützen Atmelprodukte und Funktionen wird durch den Hersteller über Firmwareupgrades regelmäßig aktualisiert. Eine detailliertere Dokumentation[2] des Programmers wird online zur Verfügung gestellt.

[2] http://www.atmel.com/tools/AVRJTAGICEMKII.aspx?tab=documents (Abrufdatum 17.9.2012)

Kapitel 4

Konzept zur Lösung der Aufgabenstellung

Grundlage des zu implementierenden Algorithmus ist der Entwurf[1] „LoWPAN fragment Forwarding and Recovery“ von Pascal Thubert und Jonathan W. Hui. Die ursprüngliche Idee der Autoren und die weiterführenden Gedanken im Zusammenhang mit dieser Abschlussarbeit werden in diesem Kapitel dargelegt.

4.1 Der Entwurf „LoWPAN fragment Forwarding and Recovery“

Die Problematik bei der Übertragung von IPv6-Paketen stellt die minimale MTU von 1280 Bytes dar. Im schlimmsten Fall stehen einem Rahmen nach Standard 802.15.4 nur 74 Bytes für die Übertagung der Nutzdaten zur Verfügung, so dass Pakete auf Ebene des 6LoWPAN-Adaptionslayers in 18 Fragmente aufgeteilt und am Zielort wieder zusammengefügt werden müssen. Sollte eines der Fragmente verloren gehen, erfordert es die komplette Neuübertragung sämtlicher zum Paket gehörender Fragmente zu Lasten der ohnehin begrenzten Performance.

Der Entwurf von Pascal Thubert und Jonathan W. Hui präsentiert ein einfaches Protokoll für das Weiterleiten und neu Anfordern einzelner Fragmente, die unter Umständen auf dem Weg zwischen zwei 6LoWPAN-Endpunkten (multiple Hops) verloren gehen können.

4.1.1 Anforderungen an den Entwurf

Der Entwurf schlägt eine Methode vor, wie einzelne Fragmente zwischen zwei 6LoWPAN-Endpunkten erneut angefordert werden können. Unabhängig davon, ob Mesh-Under-Routing eingesetzt wird, werden folgende Anforderungen an das LoWPAN gestellt:

- Der Recovery-Mechanismus soll in der Lage sein, stark fragmentierte Pakete mit einer maximalen Anzahl von 32 Fragmenten zu unterstützen.
- Da eine Funkübertagung im Halbduplexmodus arbeitet und eine gewisse Ruhezeit durch die unterschiedlichen Zugriffsmechanismen auf das Übertagungsmedium veranschlagt wird, verbraucht eine Bestätigung ungefähr genauso viel Ressourcen wie ein Datenfragment. Der Recovery-Mechanismus sollte daher in der Lage sein, mehrere Fragmente mit einer einzigen Nachricht zu bestätigen. Darüber hinaus sollte es möglich sein, das Senden der Bestätigungen zu unterdrücken, falls die Übertragung der Fragmente bereits auf einer niedrigeren Schicht des OSI-Referenzmodells gesichert wird.

[1] http://tools.ietf.org/html/draft-thubert-6lowpan-simple-fragment-recovery-07 (Abrufdatum 16.6.2012)

- Der Mechanismus sollte erfolgreich sein oder nach einer definierten Zeitspanne, vorgegeben durch den Recoveryprozess auf einer höheren Ebene des OSI-Referenzmodells, aufgegeben werden.

- Durch den Einsatz eines Mesh-Under-Balancing-Mechanismus, wie bspw. dem ISA100 Data-Link-Layer, können die Rahmen in willkürlicher Reihenfolge eintreffen. Der Mechanismus sollte damit rechnen, dass scheinbar verloren gegangene Fragmente verzögert auf einem alternativen Übertragungspfad am Zielknoten eintreffen.

- Das Zusammenfassen mehrerer paralleler Datenflüsse kann zu einer Auslastung des Netzwerkes und einem möglichen Zusammenbrechen der Kommunikation führen. Der Recovery-Mechanismus sollte in der Lage sein, die Anzahl der Fragmente, die zurzeit über das LoWPAN übertragen werden, zu steuern.

4.1.2 Überblick

Unter Berücksichtigung, dass ein Multihop-LoWPAN aufgrund der eingeschränkten Möglichkeiten des Zwischenspeicherns von Paketen sehr sensibel reagieren kann, empfiehlt dieser Entwurf ein einfaches und vorsichtiges Verhalten der Auslastungskontrolle, basierend auf dem Verfahren TCP-Congestion-Avoidance (Vermeidung von Netzwerküberlastung).

Aus Sicht der sendenden LoWPAN-Station ist ein ausstehendes Fragment ein gesendetes Fragment, für das noch keine Bestätigung empfangen wurde. Das bedeutet, dass sich das Fragment noch auf dem Weg zur Zielstation befindet, es schon empfangen aber noch nicht bestätigt wurde oder aber dass sich die Bestätigung auf dem Weg zur sendenden Station befindet. Darüber hinaus besteht die Möglichkeit, dass entweder das Fragment oder aber der Bestätigungsrahmen bei der Übertragung verloren gegangen sind.

Da in einem „meshed“ LoWPAN die Rahmen nicht zwangsweise in der richtigen Reihenfolge am Empfänger eintreffen, ist es scheinbar unmöglich, zwischen diesen Situationen zu unterscheiden. Für die sendende Station befinden sich die ausstehenden Fragmente also noch auf dem Weg durch das Netzwerk und tragen zu seiner Auslastung bei. Nach einer gewissen Zeit kann jedoch davon ausgegangen werden, dass ein Rahmen beim Empfänger eingetroffen oder verloren gegangen ist, so dass er keine Netzwerkauslastung mehr verursachen kann. Diese Zeitspanne kann auf Grundlage der Round Trip-Verzögerung zwischen den zwei beteiligten LoWPAN-Endpunkten geschätzt werden. Die Methode wird detailliert in [RFC6298] (Sargent u. a.) beschrieben und wird für diese Berechnung empfohlen.

4.1.3 Einführung neuer Dispatch-Bytes und Header

Dieser Entwurf ist als Erweiterung zu „Transmission of IPv6 Packets over IEEE 802.15.4 Networks“ [RFC4944] (Kushalnagar u. a.) gedacht und führt vier neue Dispatch-Typen ein. Sie werden für die Kodierung der „Recoverable Fragments“ (RFRAG)-Header mit oder ohne Bestätigungsanforderung und für den Header des Bestätigungsrahmens (RFRAG-ACK) benötigt. Die Tabelle 4.1 dokumentiert diese neuen Dispatch-Bytes.

Recoverable Fragment Dispatch-Byte und Header

Die Abbildung 4.1 zeigt den Aufbau des Headers für ein Recoverable Fragment. Das gesetzte x-Bit bedeutet hier, dass der Sender eine Bestätigung durch den Empfänger erwartet. Das Sequence-Feld identifiziert die Fragmente und wird von 0 bis 31 durchnummeriert.

Tabelle 4.1: Neu definierte Dispatch-Bytes und Header, Inhalt übernommen von [Hui u. Thubert, b, S. 6]

Bitmuster	Header
11 101000	RFRAG - Recoverable Fragment
11 101001	RFRAG-AR - RFRAG with Ack Request
11 10101x	RFRAG-ACK - RFRAG Acknowledgment

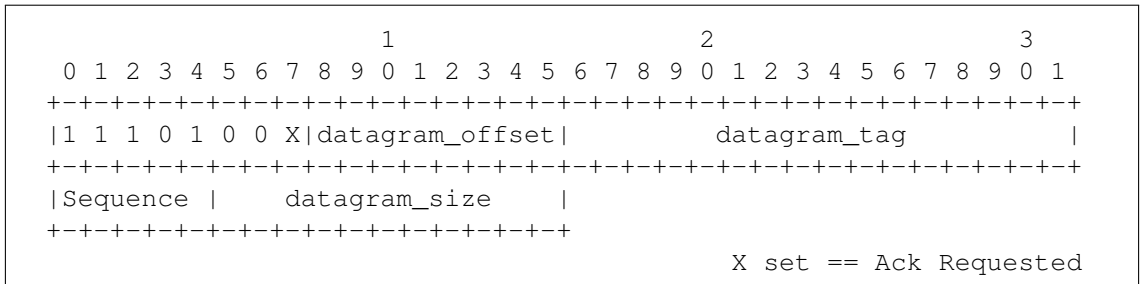


Abbildung 4.1: Recoverable Fragment Dispatch-Typ und Header, übernommen von [Hui u. Thubert, b, S. 7]

Fragment Acknowledgement Dispatch-Byte und Header

Dieser Entwurf definiert eine Bitmap, die verwendet wird, um einzelne Fragmente zu bestätigen. Die Position eines Bits in der Bitmap entspricht dabei dem zu bestätigenden Fragment. Diese Bitmap wird zu einem Datenfeld variabler Länge komprimiert und besteht aus Steuerbits und den Bestätigungsbits. Die höherwertigsten Bits bis zur ersten Null repräsentieren die Steuerbits, die restlichen Bits die Bestätigungsbits. Die Kodierung wird in Abbildung 4.2 noch einmal verdeutlicht.

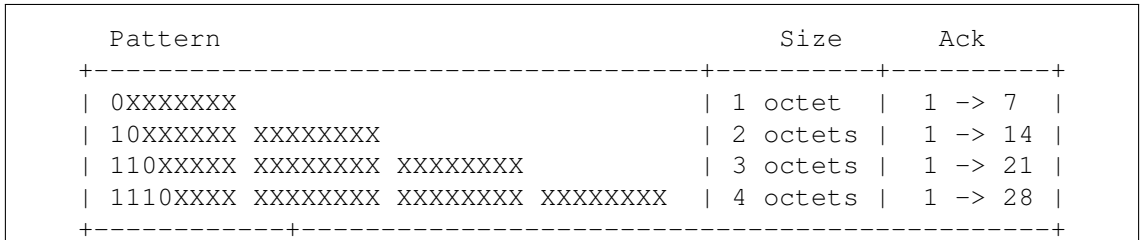


Abbildung 4.2: Kodierung der komprimierten Bestätigungsbitmap, übernommen von [Hui u. Thubert, b, S. 7]

Die Anzahl der Fragmente eines Datagramms bestimmt hierbei das zu verwendende Bitmuster. Das Format kann für zukünftige Anforderungen erweitert werden. Dieser Entwurf erfordert lediglich ein Encoding von vier Bytes, womit bis zu 28 Fragmente bestätigt werden können.

Um die unkomprimierte Form der Bitmap zu erhalten, werden die binären Werte an den Positionen, die mit einem X markiert sind, übernommen und die fehlenden Positionen bis zum 32sten Bit mit Nullen aufgefüllt. Die Abbildungen 4.3 und 4.4 verdeutlichen dies an zwei Beispielen.

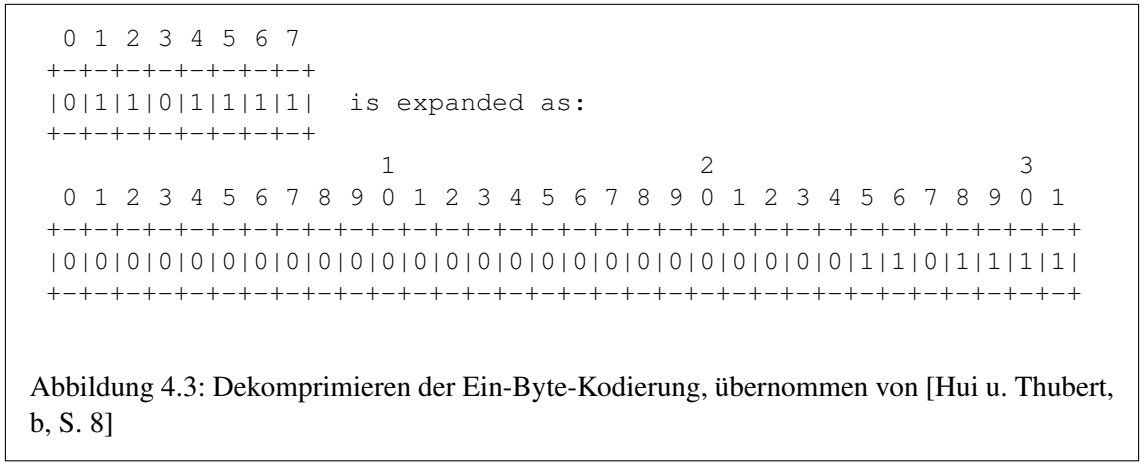


Abbildung 4.3: Dekomprimieren der Ein-Byte-Kodierung, übernommen von [Hui u. Thubert, b, S. 8]

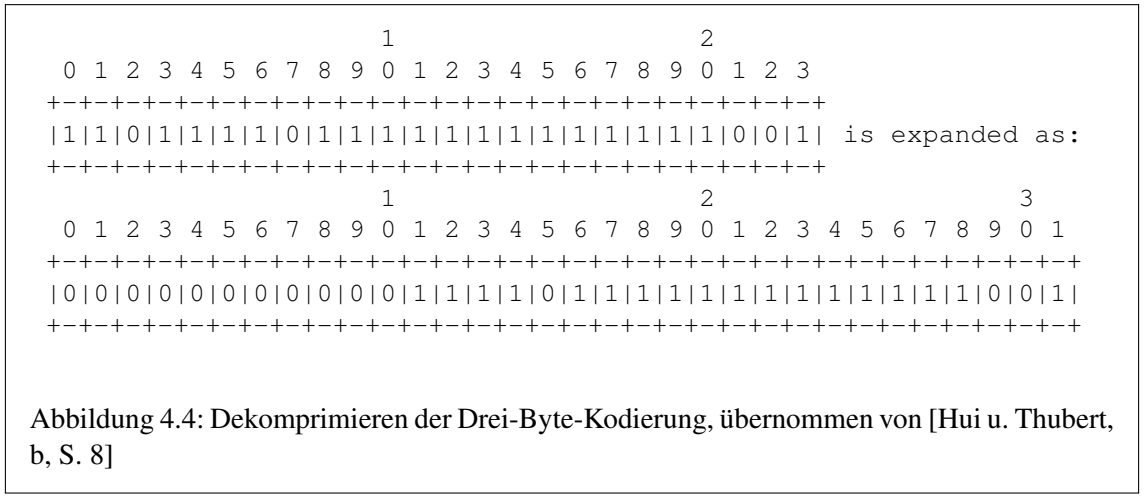


Abbildung 4.4: Dekomprimieren der Drei-Byte-Kodierung, übernommen von [Hui u. Thubert, b, S. 8]

Die unkomprimierte Bitmap wird wie in der Abbildung 4.5 dargestellt interpretiert. Für das Beispiel aus Abbildung 4.4 ergibt sich, dass alle Fragmente von Sequenz 0 bis 20 mit Ausnahme der Fragmente 1, 2 und 16 empfangen wurden. Die ausstehenden Fragmente befinden sich vermutlich noch in der Übertragung oder sind verloren gegangen. Die komprimierte Form der Bestätigungsbitmap wird in einem Bestätigungsrahmen übertragen, der entsprechend der Abbildung 4.6 aufgebaut ist. Die Position die mit einem Y markiert ist, wurde für das Signalisieren einer Netzwerkauslastung reserviert.

4.1.4 Anfordern der Fragmente

Die originalen Fragment-Header des RFC4944 in den Rahmen werden durch die Recoverable Fragments Header RFRAG und RFRAG-AR ersetzt. Der Bestätigungsrahmen mit entsprechendem Header (RFRAG-ACK) wird dabei an die sendende Station, identifiziert durch Sender-MAC-Adresse, zurückgeschickt und kann von dieser anhand des Datagramm-Tags zugeordnet werden.

Die sendende Station steuert dabei die Bestätigungen der Fragmente durch das Setzen des entsprechenden Bits (siehe Abbildung 4.1). Dies kann nach eigenen Richtlinien erfolgen und durch eine Auslastungssteuerung ergänzt werden, die in diesem Entwurf nicht beschrieben wird. Falls durch

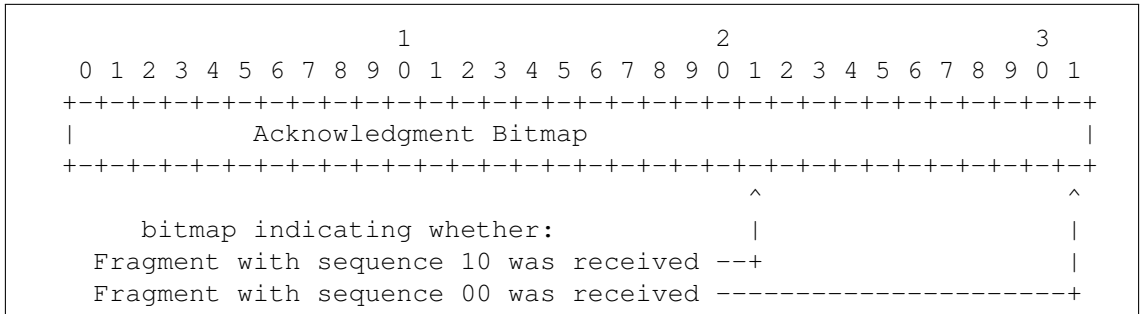


Abbildung 4.5: Dekomprimieren der Drei-Byte-Kodierung, übernommen von [Hui u. Thubert, b, S. 8]

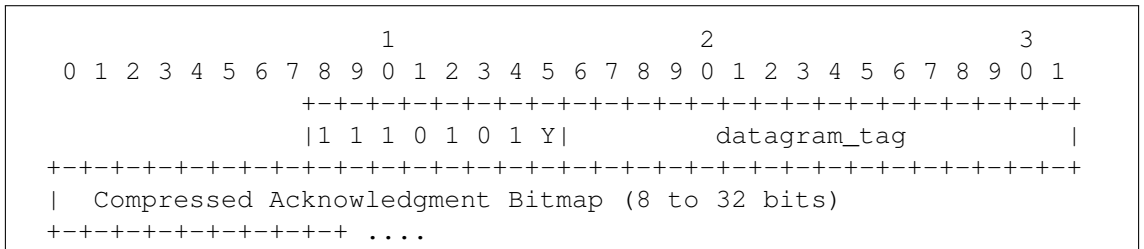


Abbildung 4.6: Fragment Acknowledgement Dispatch-Typ und Header, übernommen von [Hui u. Thubert, b, S. 9]

die sendende Station ein Sicherungsmechanismus unterhalb des 6LoWPAN-Adaptionslayers eingesetzt wird, kann auf den Einsatz des Acknowledgement-Mechanismus verzichtet werden. In diesem Fall wird das Bestätigungsbit nie gesetzt. Falls gefordert, muss die Zielstation die empfangenden Fragmente in jedem Fall bestätigen. Sie kann das Senden des Bestätigungsrahmens jedoch geringfügig verzögern.

Der Sender kann das letzte einer Reihe von zu übertragenden Fragmenten mit einer Bestätigungsanfrage versehen. Der Empfänger muss daraufhin mit dem Senden eines Bestätigungsrahmens reagieren. Falls für zuvor empfangende Fragmente keine Bestätigung angefordert wurde, kann der Empfänger das Senden des Bestätigungsrahmens absichtlich verzögern, bis alle sich auf dem Weg befindenden Fragmente die Zielstation erreicht haben. Das Verzögern des Bestätigungsrahmens kann die Berechnung der Round Trip Verzögerung hinfällig machen. Diese Option sollte daher konfigurierbar sein und nicht standardmäßig aktiviert werden.

Die empfangende Station kommuniziert mit dem Sender durch das Übertragen eines Bestätigungsrahmens, der anhand eines Bitmusters anzeigt, welche der Fragmente bereits empfangen wurden. Für dieses Bitmuster wird der 32 Bit Datentyp *signed Integer* verwendet, mit dem bis zu 32 Fragmente bestätigt werden können und ausreichend ist für die 6LoWPAN-MTU. Für jedes empfangende Fragment, identifiziert durch seine Sequenznummer n, wird das entsprechende Bit n gesetzt. Die Bits für ausstehende Fragmente enthalten den Wert Null. Werden sämtliche Bits auf null gesetzt, wird dem Sender signalisiert, dass der Empfänger das Zusammensetzen der Fragmente für dieses Datagramm abgebrochen hat.

Der Empfänger darf unaufgefordert Bestätigungen senden. Eine unaufgeforderte Bestätigung erlaubt es dem Sender, mit der Übertragung fortzufahren, wenn bereits alle Fragmente einmal gesendet wurden. Zusätzlich besteht für den Empfänger die Möglichkeit, dem Sender mitzuteilen, dass der Empfang eines bestimmten Datagramms abgebrochen wurde. Das Senden der Bestätigungsrahmen verbraucht kostbare Ressourcen – daher sollte das Senden unaufgeforderter Bestätigungen konfigurierbar sein und nicht standardmäßig aktiviert sein.

In der sendenden Station läuft ein Timer für das erneute Senden der Fragmente, für die eine Bestätigung angefordert wurde. Ist der Timer abgelaufen, muss angenommen werden, dass alle Fragmente ihr Ziel erreicht haben oder auf dem Weg dorthin verloren gegangen sind. Dieser Timer sollte auf einen Wert eingestellt werden, der zu keinem Konflikt mit möglichen Retransmissiontimern auf einer höheren Schicht des OSI-Referenzmodells führt. Die im [RFC6298] (Sargent u. a.) beschriebene Methode wird für die Berechnung dieses Timers empfohlen.

Die Fragmente werden nach dem Round Robin-Verfahren übertragen. Dieser Vorgang wird für sämtliche verlorengegangenen Fragmente wiederholt. Sie werden nacheinander übertragen, das älteste Fragment zuerst. Dadurch soll es dem Empfänger möglich sein, Fragmente mit längeren Laufzeiten zu bestätigen, bevor sie erneut gesendet werden.

Sollte der Fragmentierungsprozess durch den Sender abgebrochen werden, kann er dies dem Empfänger durch das Senden eines Pseudofragments mit dem Wert Null für *datagram_offset*, Sequenznummer und *datagram_size* mitteilen. Der Empfänger kann mit dem Empfang dieses Pseudofragmentes die reservierten Ressourcen für das entsprechende Datagramm freigeben. Wird eine Bestätigung verlangt, überträgt der Empfänger ein Bestätigungsbitmuster, das ausschließlich Nullen enthält.

Es kann passieren, dass der Empfänger den Vorgang des Zusammensetzens abbrechen muss. Beispielhaft wäre hier das Fehlen von Speicherressourcen oder der Fall, dass das Datenpaket bereits vollständig zusammengesetzt und an eine höhere Schicht des OSI-Referenzmodells übergeben wurde. In diesem Fall sollte der Empfänger den Sender durch das Übertragen eines Bestätigungsrahmens mit einem Bitmuster aus Nullen informieren. Mit dem Empfang eines solchen Bestätigungsrahmens muss der Sender die Übertragung der Datagrammfragmente abbrechen.

4.1.5 Das Weiterleiten der Fragmente

Dieser Entwurf ermöglicht es zwischenliegenden Routern die Fragmente weiterzuleiten, ohne das Datagramm wieder zusammenzusetzen. Mit dem Weiterleiten des ersten Fragments legt der Router eine Marke entlang des Pfades den das Fragment durch das Netzwerk nimmt. Geroutet wird hier auf Ebene der Netzwerkschicht. Alle nachfolgenden Fragmente werden dann auf dem gleichen Pfad durch das Netzwerk geleitet. Daraus ergibt sich, dass die Fragmente keinen anderen Weg durch das Netzwerk wählen können. Das Datagramm-Tag der Fragmente wird als Träger der Marke gewählt und mit jedem Hop ausgetauscht.

Mit dem Empfang des ersten Fragments

Beim Verfahren „Route Over“, siehe Abschnitt 2.2.3, ändert sich die Ebene 2-Adresse mit jedem Hop. Die Marke, die kreiert und anstelle des Datagramm-Tags übertagen wird, ist mit der MAC-Adresse der sendenden Station verknüpft und nur für diese MAC-Adresse gültig. Es werden die folgenden Annahmen getroffen:

- Die IPv6-Adresse der sendenden Station wird der Variablen *IP_A* zugewiesen und ist unter Umständen mehrere Hops entfernt.
- Die IPv6-Adresse der empfangenden Station wird der Variablen *IP_B* zugewiesen und kann mehrere Hops entfernt sein.
- Die Quell-MAC-Adresse wird der Variablen *MAC_prv* zugewiesen (prv kurz für previous = vorhergehend).
- Das *datagram_tag* wird der Variablen *DT_prv* zugewiesen.

Ein zwischengeschalteter Router der einzelne Fragmente weiterleitet wendet das folgende Verfahren an:

- Er bestimmt die IPv6-Adresse des nächsten Hops in Richtung *IP_B*, und weist diese der Variablen *IP_nxt* zu (nxt kurz für next = nächstfolgend).
- Über ein Neighbor Discovery (ND) wird die MAC-Adresse der Station bestimmt, die der Adresse *IP_nxt* entspricht. Die MAC-Adresse wird der Variablen *MAC_nxt* zugewiesen.

Da es sich um das erste Fragment eines Paketes handelt, werden durch den Router die folgenden Schritte durchgeführt:

- Die mit dem Tupel verknüpften Ressourcen (*MAC_prv* und *DT_prv*) werden freigegeben. Der Variablen *DT_nxt* wird für diesen Datenfluss eine neue Marke zugewiesen. Dafür kann ein Pool oder Ähnliches verwendet werden. Die Marke darf jedoch kürzlich nicht verwendet worden sein.
- Einer Struktur für den Tausch von Marken, indiziert über *MAC_prv* und *DT_prv*, werden die Werte von *MAC_nxt* und *DT_nxt* zugewiesen.
- Einer Struktur für den Tausch von Marken, indiziert über *MAC_nxt* und *DT_nxt*, werden die Werte von *MAC_prv* und *DT_prv* zugewiesen.
- Die Ziel-MAC-Adresse des Fragments wird durch die MAC-Adresse aus *MAC_nxt* ersetzt.
- Das *datagram_tag* wird durch die Marke in *DT_nxt* ersetzt.

Der Router ist nur für diesen Datenfluss konfiguriert und kann die Fragmente zum nächsten Hop weiterleiten.

Empfang weiterer Fragmente

Werden weitere Fragmente des Datagramms empfangen, existiert bereits eine über die Variablen *MAC_prv* und *DT_prv* indizierte Struktur für den Austausch der Marken. Der Router kann die Fragmente durch das Anwenden der folgenden Prozedur weiterleiten:

- Er wertet den Eintrag in der Struktur für den Austausch der Marken, indiziert über *MAC_prv* und *DT_prv*, aus und erhält die Informationen für *MAC_nxt* und *DT_nxt*.
- Die Ziel-MAC-Adresse des Fragments wird durch die MAC-Adresse aus *MAC_nxt ersetzt*.
- Das *datagram_tag* wird durch die Marke in *DT_nxt* ersetzt.

Sollte in der Struktur für den Austausch der Marken kein Eintrag für die Kombination *MAC_prv* und *DT_prv* vorhanden sein, erzeugt der Router einen Bestätigungsrahmen (RFRAG-ACK), um den Fehler anzuzeigen. Diese Nachricht wird anhand der folgenden Richtlinien erzeugt:

- Die Ziel-MAC-Adresse des Fragments wird auf die MAC-Adresse der sendenden Station verändert. Diese Adresse ist in dem Fragment enthalten.
- Das *datagram_tag* wird auf den Wert in *DT_prv* gesetzt.
- Das Bestätigungsbitmuster in dem Rahmen enthält nur Nullen, um den Fehler anzuzeigen.

Empfang eines Bestätigungsrahmens

Sollte ein Bestätigungsrahmen (RFRAG-ACK) empfangen werden, existiert bereits eine Struktur für den Austausch der Marken, indiziert über die Variablen *MAC_nxt* und *DT_nxt*. Ein Bestätigungsrahmen trifft hierbei nie vor einem Fragment ein. Der Router leitet diesen Rahmen durch das Ausführen des folgenden Vorganges weiter:

- Durch das Auswerten des Eintrages in der Struktur für den Austausch der Marken für die Werte in *MAC_nxt* und *DT_nxt* werden die Werte für die Variablen *MAC_prv* und *DT_prv* ermittelt.
- Die Ziel-MAC-Adresse des Fragments wird durch die MAC-Adresse aus *MAC_prv* ersetzt.
- Das *datagram_tag* wird durch die Marke in *DT_prv* ersetzt.

Sollte kein Eintrag in der Struktur für den Tausch der Marken gefunden werden, wird der Rahmen einfach verworfen.

Sollte der Bestätigungsrahmen einen Fehler signalisieren oder das Datagramm bereits vollständig übertagen worden sein, werden die Einträge in der Struktur für den Markenaustausch nicht sofort verworfen. Sollte der Bestätigungsrahmen auf dem Weg zu seinem Ziel verloren gehen, würden die letzten Fragmente erneut übertragen werden, was das Senden eines Bestätigungsrahmens mit einer Fehlernachricht durch andere Router auf dem Pfad zur Folge hätte, da die Ressourcen für die gespeicherten Forwardinginformationen auf diesen Geräten bereits wieder freigegeben wurden. [Hui u. Thubert, b, S. 1, 5-12]

4.2 Einsatz von Werkzeugen und Hilfsmitteln

4.2.1 Das Netzwerkanalysewerkzeug Wireshark

Die Implementierung des Entwurfes auf dem AVR RZ RAVEN Board bzw. AVR RZ USB Stick unterscheidet sich erheblich von konventionellen Softwareentwicklungsprozessen. Faktoren wie die begrenzten Hardwareressourcen auf den Plattformen, die Implementierung eines neuen, auf drahtloser Kommunikation basierenden Protokolls und der Umstand, dass eine bestehende Softwarelösung in Form des Betriebssystems Contiki anzupassen ist, erfordern eine besondere Herangehensweise an die Lösung der Aufgabenstellung. Darüber hinaus sind die Möglichkeiten des Debugging auf den bereitgestellten und für die Implementierung des Protokolls vorgesehenen Plattformen limitiert. Herkömmliche Peripheriegeräte für die Ein- und Ausgabe, die dem Entwickler die Arbeit erleichtern, sind nicht vorgesehen. Selbst bei konzentrierter Arbeitsweise und der Analyse möglicher Konsequenzen vor dem Testen veränderter Softwarefunktionalitäten entspricht das Ergebnis nicht immer den Erwartungen. Um sich einen Überblick über die Programmabläufe innerhalb des Mikrokontrollers verschaffen zu können, ist es nötig, sich im Vorfeld Gedanken über den Einsatz möglicher Debugging- bzw. Analysewerkzeuge zu machen.

Um die Auswirkungen des neuen Protokollentwurfs nachvollziehen und mit der ursprünglichen Implementierung vergleichen zu können, ist eine Analyse der Datenpakete, die über die Netzwerkschnittstelle übertragen werden, sinnvoll. Zu diesem Zweck ist der Einsatz des freien Netzwerkanalysewerkzeuges Wireshark geplant, welches für alle gängigen Betriebssysteme (Linux/Unix, Windows, Mac OS) unter dem entsprechenden Internetauftritt[2] verfügbar ist. Wireshark erlaubt es, Netzwerkverkehr aufzuzeichnen, im Promiscuous-Modus sogar Netzwerkverkehr, der nicht für den entsprechenden Host bestimmt ist. Dieser Datenverkehr wäre sonst für eine weitere Untersuchung unzugänglich. Mit Wireshark ist es unter Zuhilfenahme von sogenannten Capture-Filtern möglich, nur den gewünschten Datenverkehr zu berücksichtigen. Alternativ dazu kann in einer Aufzeichnung mit dem Display-Filter für die Analyse irrelevanter Inhalt unterdrückt werden. Das Werkzeug bietet Funktionalitäten für statistische Auswertung und Analyse des erfassten Netzwerkverkehrs. Das aktuelle Release unterstützt den Protokollstandard 802.15.4 und 6LoWPAN und erleichtert damit die Fehlersuche.

Für das Netzwerkanalysewerkzeug Wireshark sind zwei Einsatzfälle geplant. Der erste Fall ermöglicht es, den eingehenden und ausgehenden Netzwerkverkehr an der Schnittstelle des AVR RZ USB Sticks zu analysieren. Der Stick übernimmt unter anderem die Aufgabe, den 6LoWPAN-Adaptionslayer zu dekodieren um daraus gültige IPv6-Nachrichten zu erzeugen, die über die Ethernet-Schnittstelle gesendet werden. Zusätzlich werden durch den Stick die 8 Byte großen Ebene 2-Adressen des Standards 802.15.4 so angepasst, dass eine Kommunikation über das Netzwerk, über das bspw. Router Advertisement Nachrichten in das WPAN gesendet werden, möglich ist. Ursache für die zwingende Übersetzung der Ebene 2-Adressen ist der Umstand, dass innerhalb der Ethernet-Schicht nur 6 Byte große Adressen verwendet werden. Diese Einsatzmöglichkeit von Wireshark und dem AVR RZ RAVEN Stick unterliegt der Einschränkung, dass nur die Schichten des OSI-Referenzmodells oberhalb des 6LoWPAN-Adaptionslayers untersucht werden können. Fehler in der Implementierung zu entdecken wäre nur durch eine Analyse der Sicherungsschicht bzw. des Adaptionslayers möglich.

```
********** Jackdaw Menu **********
        [Built Sep  1 2012]
*  m        Print current mode    *
*  s        Set to sniffer mode   *
*  n        Set to network mode   *
*  c        Set RF channel        *
*  p        Set RF power          *
*  6        Toggle 6lowpan        *
*  r        Toggle raw mode       *
*  d        Toggle RS232 output   *
*  S        Enable sneezer mode   *
*  e        Energy Scan           *
*  R        Reset (via WDT)       *
*  h,?      Print this menu       *
*                                 *
* Make selection at any time by   *
* pressing your choice on keyboard*
***********************************
```

Abbildung 4.7: Über den Debugport erreichbares „Jackdaw Menu“ des AVR RZ USB Sticks

[2] http://www.wireshark.org/ (Abrufdatum 15.9.2012)

Zu diesem Zweck ist eine alternative Konfiguration auf einem zweiten AVR RZ USB Stick nötig. Dieser Stick wird auf einem separatem Computersystem in Kombination mit einer weiteren Installation von Wireshark eingesetzt. Im Rahmen der Masterarbeit wurde hierfür das Motherboard AT5IONT-I basierend auf dem Intel NM10 Chipsatz mit Dual-Core Intel® Atom™ Prozessor D525 und die Linuxdistribution Debian „Squeeze“ (64 Bit) verwendet. Die Installation des Sticks ist unkompliziert, da durch die Linuxdistribution hervorragender Support geboten wird. Die Hardware wird vom Betriebssystem erkannt und die nötigen Treiberkomponenten automatisch installiert.

```
Currently Jackdaw:
  * Will not send data over RF
  * Will not change link-local addresses inside IP messages
  * Will not decompress 6lowpan headers
  * Will Output raw 802.15.4 frames
  * Will Output RS232 debug strings
  * USB Ethernet MAC: 02:12:13:14:15:16
  * 802.15.4 EUI-64: 02:12:13:ff:fe:14:15:16
  * Operates on channel 26 with TX power  +3.0dBm
  * Current/Last/Smallest RSSI: -92/-92/--dBm
  * Configuration: 129, USB<->ETH is active
  * Never-used stack > 200 bytes
```

Abbildung 4.8: Ausgegebene Konfiguration des AVR RZ USB Sticks für den Sniffermodus

Eine Dokumentation und Installationshinweise für die Arbeit mit dem Stick sind in der Contikidistribution, im Unterordner „\doc“ in der Datei „ravenusbstick-docs.txt“ zu finden. Der Stick wird von den Entwicklern mit dem alternativen Codenamen Jackdaw (deutsch Dohle) bezeichnet. Über eine Applikation, die eine Kommunikation mit dem seriellen Port ermöglicht, kann der Stick konfiguriert werden. Jackdaw stellt hierfür einen Debugport, der unter Linux/Debian mit „ttyACM0“ bezeichnet wird, bereit. Durch die Konfiguration des seriellen Ports auf „dev/ttyACM0“ und das Setzen der Parameter für Bits und Stopbits auf 8 bzw. 1 ohne Parität und Flusskontrolle kann durch das Drücken der Taste „h“ das in Abbildung 4.7 dargestellte Menü aufgerufen werden. Da sich hinter dem Debugport des Jackdaw kein wirkliches physikalisches Gerät verbirgt, kann für die Bitrate ein beliebiger Wert gewählt werden. Jackdaw ist standardmäßig auf den Netzwerkmodus konfiguriert. Hier wird durch Drücken der Taste „s“ der sogenannte „Sniffer“-Modus aktiviert, bei dem der Stick Datenrahmen empfängt, jedoch selbst keine sendet. Das „Übersetzen“ der Ebene 2-Adressen wird nun ebenso verhindert. Durch Betätigen der Taste „6“ wird das Dekodieren des 6LoWPAN-Adaptionslayers deaktiviert bzw. aktiviert. Auf ähnliche Weise wird durch die Taste „r“ das Umschalten zwischen den 802.15.4-Raw-Modus und herkömmlicher Ethernetkommunikation ermöglicht. Durch Drücken der Taste „m“ wird die aktuelle Konfiguration in dem Terminalfenster ausgegeben – die Abbildung 4.8 zeigt die gewünschten Modifikationen beispielhaft.

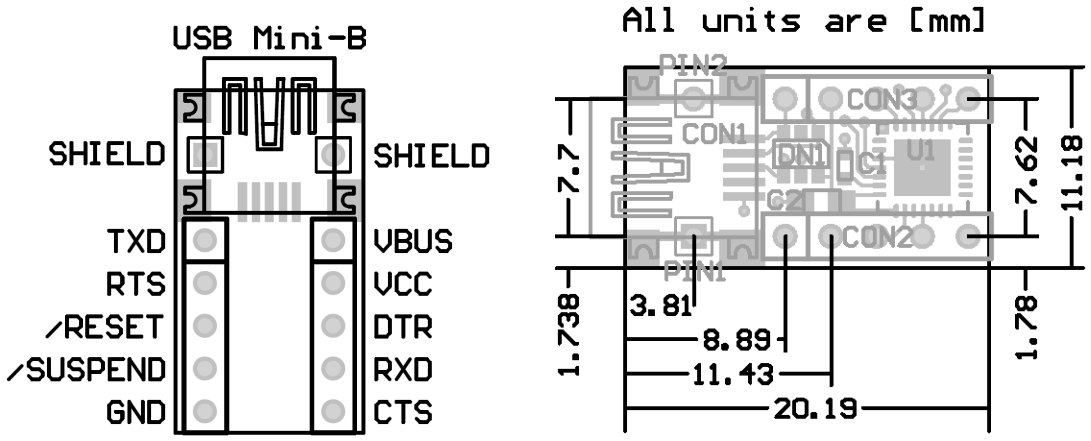
Wird Jackdaw auf die beschriebene Art konfiguriert, ist es unter Zuhilfenahme von Wireshark möglich, die drahtlose Kommunikation zu „belauschen“, aufzuzeichnen und zu analysieren. Die aktuellen Releases von Wireshark ermöglichen es ohne weitere Anpassungen, den Funkstandard IEEE 802.15.4 und den 6LoWPAN-Adaptionslayer zu dekodieren. Durch die Eingabe der Filteroption „wpan“, hierbei handelt es sich um den zuvor erwähnten Display-Filter, kann eine Vorauswahl getroffen werden.

4.2.2 Debugging des AVR RZ RAVEN über die serielle Schnittstelle

Wie bereits im Abschnitt über den Einsatz des Netzwerkanalysetools Wireshark erwähnt, bietet das Programmieren von Mikrokontrollern, im Falle dieser Abschlussarbeit das AVR RZ RAVEN Kit, besondere Herausforderungen, falls sich die Implementierung nicht wie erwartet verhalten sollte und eine Fehleranalyse notwendig ist. Eine Möglichkeit, fehlerhaften Programmcode zu lokalisieren bietet sicher das auf den Plattformen vorhandene JTAG-Interface in Kombination mit dem eingesetzten Programmer und Debugger AVR JTAGICE mkII, der sogenanntes On-Chip-Debugging zulässt. Für die zu lösende Aufgabe wurde diese Art des Debuggings als zu unpraktikabel empfunden, was den Zeitaufwand und die Interpretation der über die Schnittstelle gewonnenen Daten betrifft. Darüber hinaus ist diese Art des Debuggings nur für die im Rahmen von Atmel® Studio (siehe Abschnitt 4.2.4) erstellten Projekte verfügbar.

Ergänzend zum Einsatz von Wireshark wurde daher eine weitere Debuggingmethode in Betracht gezogen. Hier wird eine Terminalverbindung zu der auf dem AVR RZ RAVEN Board vorhandenen seriellen Schnittstelle für die Visualisierung genutzt. Da bereits durch die Entwickler von Contiki eine Ausgabe von Debugginformationen implementiert wurde, die über die Präprozessorvariable DEBUG aktiviert (Wert ungleich 0) bzw. deaktiviert (Wert 0) werden kann, wurde diese Methode favorisiert. Über den Befehl „printf“ könnten so einzelne Variableninhalte ausgegeben oder sogar eine vollständige Ausgabe eines Hexdumpes für empfangene bzw. zu sendende Datenrahmen realisiert werden.

Für die Debugginglösung wird das USB UART Wandler Modul ioMate-USB1 V2.0 verwendet, das auf dem IC CP2102 basiert. Dieses Modul wurde durch das Unternehmen **chip45 GmbH & Co. KG**[3] in Deutschland entwickelt und vertrieben. Das Unternehmen stellt zu dem Produkt ein Datenblatt[4] bereit.



(a) Pinbelegung (b) Dimensionen

Abbildung 4.9: Das USB UART Wandler Modul ioMate-USB1 V2.0, Quelle siehe Fußnote 4

In der Abbildung 4.9 ist in der schematischen Darstellung die Pinbelegung des ioMate zu erkennen. Für das Debugging würde das Verwenden des Pins „RXD“ genügen, der Vollständigkeit halber wird der Pin TXD ebenso mit dem Pin RXD1 auf dem RZ RAVEN verbunden. Darüber

[3] http://www.chip45.com/index.php (Abrufdatum 15.9.2012)

[4] download.chip45.com/ioMate-USB1_V2.0_infosheet.pdf (Abrufdatum 15.9.2012)

hinaus ermöglichen es die Pins VBUS (5 V Gleichspannung) und GND (Masse), das AVR RZ RAVEN über eine externe Spannungsversorgung zu betreiben. Hierfür muss die Jumperbrücke auf der rechten Seite des Boards (Draufsicht) von der Position BAT auf EXT verändert werden. Die Abbildung 4.11 zeigt noch einmal die Platinenrückseite des ioMate und die zu verwendenden Pins.

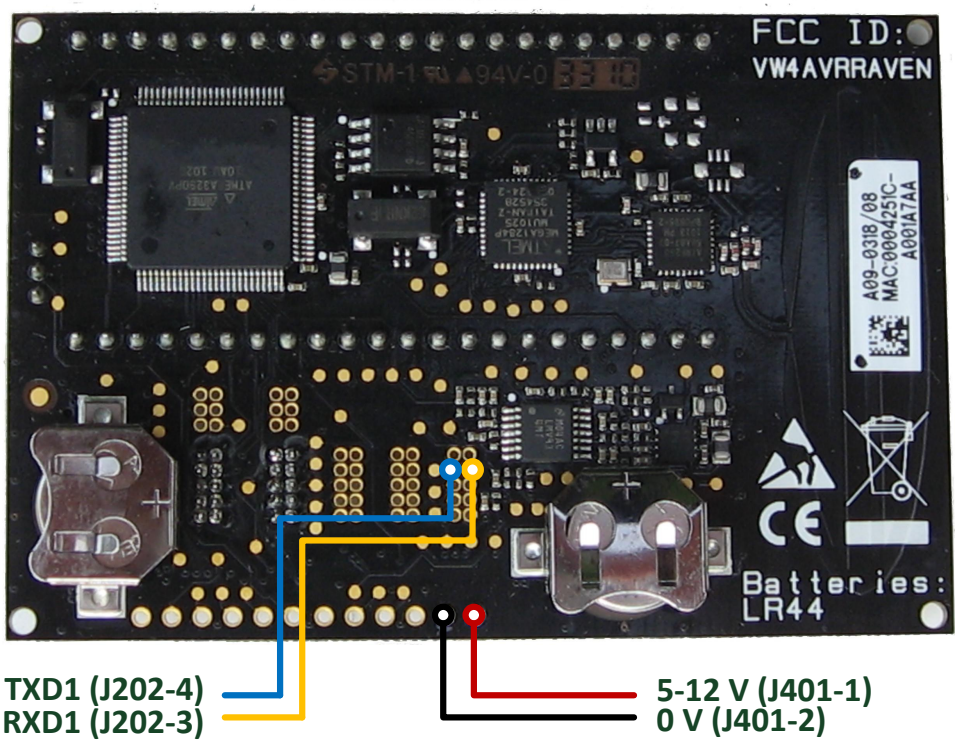


Abbildung 4.10: *AVR RZ RAVEN Board Rückseite mit verwendeten Pins*

Für die serielle Kommunikation ist auf dem AVR RZ RAVEN das ATmege1284P User-Interface vorgesehen. Die Steckerleiste J202 stellt an den Kontakten 3 und 4 die Anschlüsse für Sende (TXD1)- und Empfangsrichtung (RXD1) bereit. Um eine serielle Verbindung zwecks Debugging herstellen zu können müssen die Pin RXD auf dem ioMate mit dem Pin TXD1 auf dem AVR RZ RAVEN verbunden werden. Eine bidirektionale Kommunikation wird durch eine Verbindung zwischen den Pins TXD und RXD1 realisiert. Um die durch das ioMate gelieferte Spannung durch das AVR RZ RAVEN nutzen zu können, stellt das Interface J401 des AVR RZ RAVEN die Pins 1 (5-12V Spannungswandler) und 2 (0 V – Masse) zur Verfügung. Die entsprechenden Kontakte auf dem ioMate werden mit diesen Pins verbunden. In der Abbildung 4.10 ist die Rückseite des AVR RZ RAVEN Boards mit Angabe der zu nutzenden Pins zu sehen. Eine Dokumentation[5] für das AVR RZ RAVEN Kit wird auf der Homepage des Herstellers[6] zur Verfügung gestellt.

Für den Verbindungsaufbau wird die Anwendung PuTTY[7] verwendet. Dabei handelt es sich um einen SSH/telnet-Client, der für Windowsplattformen entwickelt wurde. Über den Windows-Gerätemanager kann der COM-Port, welcher dem ioMate zugewiesen wurde, ermittelt werden. Mit den

[5] http://www.atmel.com/tools/RZRAVEN.aspx?tab=documents (Abrufdatum 15.9.2012)

[6] http://www.atmel.com/ (Abrufdatum 15.9.2012)

[7] http://www.putty.org/ (Abrufdatum 15.9.2012)

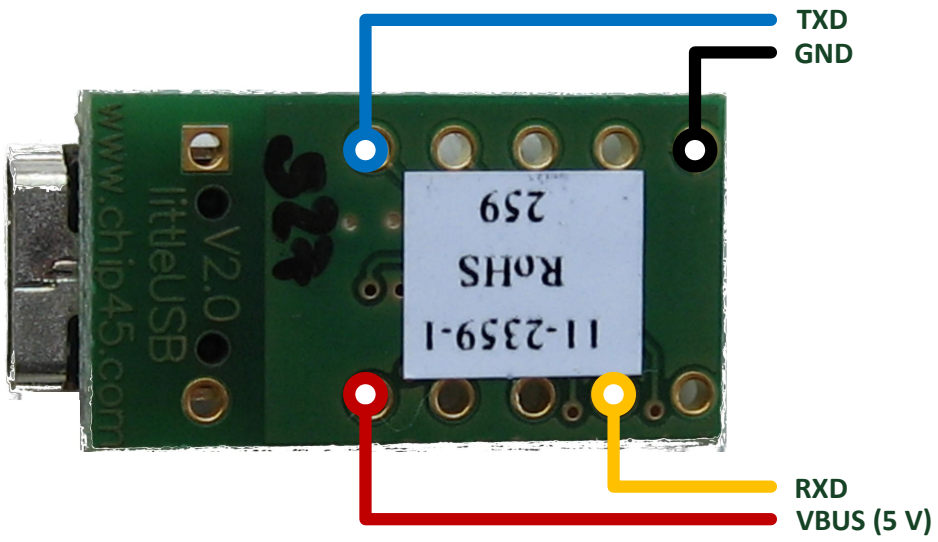


Abbildung 4.11: *ioMate-USB1 V2.0 Rückseite mit verwendeten Pins*

zusätzlichen Parametern 57600 Baud, 8 Datenbits, 1 Stopbit und fehlender Parität kann eine Verbindung zu dem AVR RZ RAVEN aufgebaut werden. Nach dem Zurücksetzen des AVR RZ RAVEN kann der Bootvorgang wie in Abbildung 4.12 dokumentiert beobachtet werden.

```
*******Booting Contiki 2.5*******
MAC address 2:11:22:ff:fe:33:44:55
nullmac sicslowmac, channel 26 power 0
IPv6 Address: fe80::11:22ff:fe33:4455
Contiki-Raven.localhost online with fixed 4687 byte web content

Addresses [4 max]
fe80::11:22ff:fe33:4455

Neighbors [20 max]
  <none>
Routes [20 max]
  <none>
---------
Never-used stack > 3010 bytes
```

Abbildung 4.12: Serielle Ausgabe während des Bootvorganges unter Contiki-2.5

4.2.3 Versionskontrolle über den Subversionclient TortoiseSVN

Während der Softwareentwicklung wird man in der Regel mit dem Problem konfrontiert, dass sich ein Fehler über eine längere Entwicklungsperiode hinweg eingeschlichen hat und erst zu einem späteren Zeitpunkt, meist mit der Implementierung einer neuen Funktionalität, entdeckt wird. Hier ist es wünschenswert, den aktuellen Quellcode mit einer älteren Variante vergleichen und gegebenenfalls wieder verwerfen zu können.

Auch kann der Wunsch, eine neue Implementierung erst ausgiebig zu testen, aber parallel in der Softwareentwicklung fortfahren zu können, zum Einsatz einer Lösung in Gestalt einer Versionsverwaltungssoftware führen. Eine Protokollierung der einzelnen Änderungen während einer Softwareentwicklung ist daher, aus welchen Gründen auch immer, ratsam. Oft arbeiten sogar mehrere Entwickler an einem Projekt, was unter Umständen zu verschiedenen Softwarezweigen, sogenannten Branches, führt. Letzterer Fall wird in dieser Abschlussarbeit eine eher untergeordnete Rolle spielen.

Eine freie Softwarelösung aus dieser Kategorie ist das Werkzeug TortoiseSVN[8], verfügbar für Windowsplattformen. Hierbei handelt es sich um eine Alternative zur bekannten Versionskontrollsoftware Apache™ Subversion®[9]. Dieser Subversion-Client integriert sich in den Windows-Explorer und ermöglicht komfortables Arbeiten. Mit einem Rechtsklick im Explorer kann im entsprechenden Menüpunkt auf die Funktionalitäten von TortoiseSVN zugegriffen werden. Für die Lösung der Aufgabenstellung im Rahmen dieser Abschlussarbeit ist es zuerst nötig, ein Archiv, ein sogenanntes Repository, zu kreieren. Hinter dem Repository verbirgt sich eine Datenbank mit sämtlichen Versionen der zu entwickelnden Software, die vom TortoiseSVN-Client verwaltet wird. Als Speicherort für diese Datenbank wurde ein sogenanntes Network Attached Storage (NAS) vom Typ QNAP TS-219P mit zwei Festplatten im RAID-Verbund (RAID 1) verwendet. In diesem Fall werden die Daten auf zwei Festplatten gespiegelt und bieten ein gewisses Maß an Sicherheit bei einem Ausfall der Hardware.

TortoiseSVN bietet die Funktion „Importieren“, mit der es möglich ist, dem Projektarchiv eine Version von Contiki hinzuzufügen. Die Funktion „SVN Checkout“ erlaubt es, eine Arbeitskopie der verwendeten Version zu erstellen, in diesem Fall steht nur die Revision 1 zur Verfügung. Im Laufe der Implementierungsphase des Protokollentwurfes ist es möglich, an markanten Entwicklungsschritten über die Funktion „SVN Commit“ die geänderten Dateien in das Projektarchiv zu schreiben. Hierfür wird eine neue Version der Datei, identifiziert über eine aufsteigende Revisionsnummer, angelegt. Hierbei gilt es zu beachten, dass für die verschiedenen Dateien im Projektarchiv unterschiedliche Revisionsnummern, abhängig von der Anzahl der Änderungen, existieren können. Die Dateien der aktuellen Arbeitskopie können mit ihren älteren Revisionen verglichen werden. Alternativ kann eine ältere Revision des Projektes in einem neuen Arbeitsverzeichnis „ausgecheckt“ werden. Die Beschreibung sämtlicher Funktionalitäten von TortoiseSVN wäre so umfangreich, dass sie den Rahmen dieser Arbeit sprengen würde. Unter dem Internetauftritt der Entwickler kann auf eine umfangreiche Dokumentation (siehe Fußnote 8) zugegriffen werden.

[8] http://tortoisesvn.net/ (Abrufdatum 15.9.2012)

[9] http://subversion.apache.org/ (Abrufdatum 15.9.2012)

4.2.4 Entwicklungsumgebung Atmel® Studio

Die Integrated Development Environment (IDE) Atmel® Studio[10] ermöglicht das Entwickeln, Erstellen, Simulieren und Debuggen von Softwarelösungen, die auf Atmel Mikrokontrollerplattformen basieren. Atmel® Studio unterstützt den Einsatz der Programmiersprachen C, C++ und Assembler in eigenen Softwareprojekten. In Atmel® Studio ist die GCC toolchain, eine Sammlung von Werkzeugen für das Kompilieren und Erstellen der für das Programmieren des Flashspeichers bzw. EEPROMs benötigten .hex- bzw. .elf-Files, bereits integriert.

Trotz des riesigen Funktionsumfanges der Entwicklungsumgebung Atmel® Studio war für die Lösung der Aufgabenstellung nur die Nutzung eines Moduls dieser Software geplant. Hierbei handelt es sich um das Werkzeug, mit dem die erstellten .hex- bzw. .elf-Files auf den entsprechenden nichtflüchtigen Speicher der eingesetzten Hardwarepattform programmiert werden. Begründet wird diese Entscheidung mit dem Bereitstellen eigener Werkzeuge und Hilfsmittel durch die Entwickler des Betriebssystems Contiki[11]. Hierfür wird ein sogenanntes Instant Contiki[12] zum Download bereitgestellt, eine etwas über ein Gigabyte große komprimierte Datei, welche die Linuxdistibution Ubuntu mit allen für die Entwicklung benötigten Anwendungen enthält. Diese Linuxdistribution kann innerhalb einer Virtualisierungssoftware wie VirtualBox[13] oder VMware[14] eingesetzt werden. Die Instant Contiki Entwicklungsumgebung und die eingesetzte Contiki-Version werden in einem späteren Abschnitt genauer betrachtet.

Während der Gestaltung dieser Abschlussarbeit befand sich die IDE Atmel® Studio in dem Entwicklungsstadium Beta vor der Veröffentlichung der Version 6. Diese Version wurde testweise installiert, führte aber zu einigen Problemen beim Programmieren des AVR RZ RAVEN über die .elf-Files. Hier war es nicht möglich, den Mikrokontroller ATmega 3290p, der hauptsächlich für das Ansteuern des LCD und der zusätzlichen Peripherie sowie das Auswerten des 4-Kontakt-Schalters auf dem Board verantwortlich ist, zu programmieren. Da es sich um ein Softwareprodukt in der Betaphase der Entwicklung handelte, wurde der Einsatz dieser Version letztendlich verworfen.

Die Installation der Vorgängerversion 5 wäre der nächste logische Schritt gewesen. Auf Anraten des Betreuers und eines Entwicklers, der bereits Erfahrung mit dieser Version sammeln konnte, wurde schließlich die Version 4 eingesetzt. Ein wichtiges Feature der Version 4, das Verwenden der .elf-Files für die Programmierung, wurde in Version 5 nicht mehr angeboten. Der Mehraufwand an Arbeit und das Risiko, ein Gerät fehlerhaft zu programmieren, machte den Einsatz der älteren Version sinnvoll. Der Konfigurationsaufwand, speziell die korrekte Auswahl der „Fuses“ und „LockBits“ entfällt durch das Verwenden der .elf-Files, da diese Informationen bereits Teil der Datei sind.

[10] http://www.atmel.com/tools/atmelstudio.aspx?tab=overview (Abrufdatum 15.9.2012)

[11] http://www.contiki-os.org/ (Abrufdatum 15.9.2012)

[12] http://www.contiki-os.org/start.html (Abrufdatum 15.9.2012)

[13] https://www.virtualbox.org/ (Abrufdatum 15.9.2012)

[14] http://www.vmware.com/ (Abrufdatum 15.9.2012)

4.2.5 Notepad++

Im Rahmen dieser Abschlussarbeit war es durch die Aufgabenstellung vorgesehen, dass eine bestehende Version des offenen Betriebssystems Contiki angepasst werden sollte, um den Protokollentwurf zu realisieren. Das Erstellen eines eigenen Projektes innerhalb einer Entwicklungsumgebung würde also entfallen. Die Modifikation des Contiki-Programmcodes würde sich also auf das Editieren und Verändern des bereits vorhandenen Quellcodes beschränken. Da diese Entwicklungsarbeit an einem PC mit installiertem Betriebssystem Microsoft Windows 7 Professional x64 durchgeführt wird, fiel für diese Arbeit die Wahl auf das Editierwerkzeug Notpad++[15].

Bei Notepad++ handelt es sich um einen frei verfügbaren Editor für Quellcode, der eine Vielzahl von Programmiersprachen unterstützt. Der Editor ermöglicht es dem Nutzer, das entsprechend der erkannten oder ausgewählten Programmiersprache voreingestellte Syntax Highlighting anzupassen. Teile des Quellcodes können aus- bzw. eingeblendet werden. Dieses Feature wird als Syntax Folding bezeichnet. In Notepad++ können mehrere Quelltexte zugleich bearbeitet werden und die entsprechenden Dateien stehen nach einem Neustart des Betriebssystems oder des Editors, sofern vorhanden, wieder zum Editieren bereit. Der Funktionsumfang des Werkzeuges geht weit über die in diesem Abschnitt beschriebenen Features[16] hinaus, für das Lösen der Aufgabenstellung ist das hier Beschriebene allerdings ausreichend.

4.3 Eingesetzte Contiki-Version

Zu dem Zeitpunkt, als über eine mögliche Aufgabenstellung für diese Abschlussarbeit nachdacht wurde, veröffentlichten die Entwickler des Betriebssystems Contiki die Version 2.5 (9. September 2011). Mit dem Abgabetermin der Arbeit war das aktuelle Release die Version 2.6 (17 Juli 2012). Es galt also eine Entscheidung bezüglich der für die Implementierung des Protokollentwurfs zugrundeliegenden Version zu treffen.

Da das aktuelle Release von Contiki während der Bearbeitungsphase der Abschlussarbeit erschien, wurden nur die Versionen 2.4 und 2.5 in Betracht gezogen. Da sich die Aufgabenstellung auf eine mögliche Implementierung unter 2.4 bezog und auf durch den Betreuer gesammelte Erfahrungen zurückgegriffen werden konnte, wurde zuerst diese Version näher untersucht. Auf der Internetseite[17] der Entwickler ist es möglich, das aktuelle Release und ältere Versionen des Betriebssystems herunterzuladen. Es wird der Quellcode des Betriebssystems, .elf-Files für das Programmieren auf den nichtflüchtigen Speicher verschiedener Mikrokontrollerplattformen und die zuvor erwähnte Instant-Variante des Betriebssystems bereitgestellt.

```
*******Booting Contiki 2.4*******
                                  Contiki-Raven.localhost online wi...
                IPv6 Address: fe80::11:22ff:fe33:4455
```

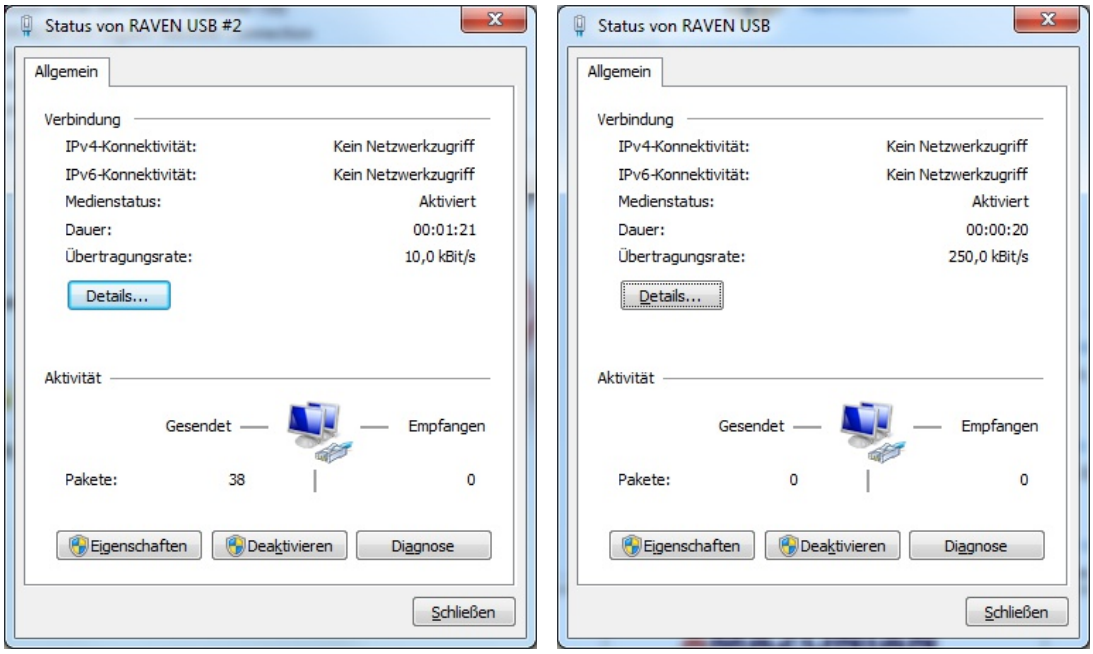
Abbildung 4.13: Serielle Ausgabe während des Bootvorganges unter Contiki-2.4

[15] http://notepad-plus-plus.org/ (Abrufdatum 15.9.2012)

[16] http://notepad-plus-plus.org/features.html (Abrufdatum 15.9.2012)

[17] http://www.contiki-os.org/download.html (Abrufdatum 15.9.2012)

Das AVR RZ RAVEN Board und der AVR RZ USB Stick wurden mit der Version 2.4 des Betriebssystems Contiki programmiert. Ein Performancetest auf dem durch die Entwickler bereitgestellten Anwendungsbeispiel „Webserver-ipv6-raven“ offenbarte eine eher befriedigende Performance beim Aufruf der bereitgestellten Webseite. Für die Kommunikation nach Standard 802.15.4 wurde eine Bitübertragungsschicht verwendet, die eine Datenübertragungsrate von 10 kbit/s unter dem Betriebssystem Microsoft Windows 7 Professional x64 bereitstellte. Der Einsatz der Debugginglösung über die serielle Schnittstelle des AVR RZ RAVEN zeigte kleinere Fehler bei Formatierung der Ausgabe der verschiedenen Informationen durch das Betriebssystem auf. Ein Ausschnitt der Ausgabe während des Bootvorgangs ist in Abbildung 4.13 dokumentiert worden. Die Arbeit mit der Version 2.4 des Instant Contiki führte zu Problemen bei der Installation der Gasterweiterungen, die ein komfortables Arbeiten mit der Virtualisierungssoftware VirtualBox ermöglichen. Eine Inkompatibilität mit dem auf der Linuxdistribution eingesetzten Grafiktreiber erschwerte eine Arbeit mit der bereitgestellten Entwicklungsumgebung. Von einer Aktualisierung des Instant Contiki ist generell abzuraten, da die Distribution im Anschluss nicht mehr fehlerfrei verwendet werden kann.



(a) Contiki 2.4 (b) Contiki 2.5

Abbildung 4.14: Gegenüberstellung der Datenraten der eingesetzten Bitübertragungsschicht

Das ältere Release wurde mit der zu diesem Zeitpunkt aktuellen Version 2.5 verglichen. Contiki 2.5 verwendet für die Kommunikation nach Standard 802.15.4 eine Bitübertragungsschicht, die eine Datenübertragungsrate von 250 kbit/s ermöglicht. Die Abbildung 4.14 zeigt eine Gegenüberstellung der Datenübertragungsraten der beiden Contikiversionen. Performancetests ähnlich derer, die unter Version 2.4 mit dem „Webserver-ipv6-raven“ durchgeführt wurden, machten die Vorteile der neuen Bitübertragungsschicht deutlich.

Eine Auswertung des Bootvorganges über die serielle Verbindung verdeutlichte eine Überarbeitung der Ausgaben. Die verschiedenen Informationen wurden während der Laufzeit fehlerfrei ausgegeben. Eine Überprüfung der Version 2.5 des Instant Contiki lieferte das Ergebnis, dass eine Installation der Gasterweiterungen möglich war. Die Skalierbarkeit der Auflösung des Ubuntu-Desktops in einem Windowsfenster und das Auswählen des Fensters durch einen einfachen Klick gestaltet die Arbeit einfach und komfortabel.

Die enormen Vorteile des Betriebssystems Contiki in der Version 2.5 führten dazu, dass dieses Release als Grundlage für die Implementierung des Protokollentwurfs favorisiert wurde.

Kapitel 5

Realisierung des Konzeptes

5.1 Quellcodeanalyse des Betriebssystems Contiki

5.1.1 Installation des Betriebsystems und der AVR-Werkzeuge

Für die Realisierung des Konzeptes wurde das Betriebssystem Contiki in der Version 2.5 gewählt. Durch den geplanten Einsatz des Netzwerkanalysewerkzeuges Wireshark wurde ein Computersystem basierend auf dem Dual-Core Intel® Atom™ Prozessor D525 und der Linuxdistribution Debian „Squeeze“ (64 Bit) bereitgestellt. Anstatt das angebotene Image der Linuxdistribution Ubuntu mit allen benötigten Dateien und Programmen, das sogenannte „Instant Contiki“, in einer Virtualisierungssoftware zu verwenden, konnte für die Softwareentwicklung nun eine andere Lösung gewählt werden.

Hierfür wird der komplette, online bereitgestellte Quellcode[1] des Betriebssystems heruntergeladen und im „home“ –Verzeichnis des Benutzers entpackt. Über das Advanced Packaging Tool (APT) können die für das Erstellen der .hex- bzw. .elf-Files nötigen Werkzeuge von AVR durch Eingabe des Befehls, dokumentiert in Abbildung 5.1, nachinstalliert werden. Die Debian-Umgebung kann nur ähnlich wie das durch die Entwickler bereitgestellte „Instant Contiki“ eingesetzt werden.

```
apt-get install  avr-libc binutils-avr gcc-avr gdb-avr simulavr
```

Abbildung 5.1: Befehl für das Installieren der AVR-Werkzeuge

5.1.2 Allgemeiner Aufbau des Betriebssystems

Die Ordnerstruktur des Betriebssystem Contiki wurde in der Abbildung 5.2 dargestellt. Verwendet wurde der Befehl „dir /o“ auf einer Computerplattform mit installiertem Windows 7 Professional x64. Die Verzeichnisse sind strukturiert aufgebaut und beinhalten die Quellcodes und benötigten Daten für die Contiki-Werkzeuge wie bspw. den Netzwerksimulator Cooja. Es existiert eine Vielzahl von Beispielprogrammen, die für die bekanntesten Hardwareplattformen bereitgestellt werden. Das Betriebssystem Contiki ist derart komplex, dass im Rahmen dieser Abschlussarbeit nur ein Bruchteil der Verzeichnisse analysiert werden konnte. Für die Lösung der Aufgabenstellung waren drei Verzeichnisse von Belang.

[1] http://www.contiki-os.org/download.html (Abrufdatum 15.9.2012)

```
Verzeichnis von C:\Users\Benutzer\contiki-2.5

17.09.2012  21:56    <DIR>          .
17.09.2012  21:56    <DIR>          ..
17.09.2012  21:56    <DIR>          apps
17.09.2012  21:56    <DIR>          core
17.09.2012  21:56    <DIR>          cpu
17.09.2012  21:56    <DIR>          doc
17.09.2012  21:56    <DIR>          examples
17.09.2012  21:56    <DIR>          platform
17.09.2012  21:56    <DIR>          tools
06.09.2011  23:43             6.978 Makefile.include
06.09.2011  23:43             1.809 README
06.09.2011  23:43             4.738 README-BUILDING
06.09.2011  23:43             6.355 README-EXAMPLES
               5 Datei(en),         19.880 Bytes
               9 Verzeichnis(se), 155.031.433.216 Bytes frei
```

Abbildung 5.2: Verzeichnisstruktur des Betriebssystems Contiki Version 2.5

Das Verzeichnis „\doc“ enthält eine Vielzahl nützlicher Anleitungen für den Umgang mit dem Betriebssystem und für die Konfiguration der verwendeten Mikrokontroller. Besonders interessant für diese Abschlussarbeit sind die Beschreibungen für den μIP- und den Rime-Stack.

Im Ordner „examples“ sind Beispielanwendungen, teilweise den Hardwareplattformen zugeordnet, zu finden. Die Unterverzeichnisse „\examples\webserver-ipv6-raven“ und „\examples\raven-usbstick“ enthalten die Quellcodes und Make-Files für den AVR RZ USB Stick und das AVR RZ RAVEN Board. Diese Make-Files enthalten die Konfigurationen bzw. Befehle, um die für die Programmierung des nichtflüchtigen Speichers benötigten .elf-Files zu erzeugen. Bei dem Programm „make“ handelt es sich um ein mächtiges Softwareentwicklungswerkzeug für das automatisierte Kompilieren des Quellcodes und das Erzeugen der gewünschten Dateien. Die verwendeten Make-Files wurden durch die Contiki-Entwickler bereitgestellt und sind Teil des Betriebssystems. Im Rahmen dieser Abschlussarbeit wurden sie nicht verändert. Im Contiki-Hauptverzeichnis ist in der Datei „README-BUILDING“ eine Beschreibung für den Umgang mit den Make-Files zu finden.

Der Unterordner „\core“, dargestellt in der Abbildung 5.3, enthält sämtliche Quelldateien der Module und Bibliotheken des Betriebssystems. Der Ordner „\net“ ist hier von besonderem Interesse – er enthält die Quelldateien, die für die Netzwerkkommunikation nötig sind. Die Protokolle für die drahtlose Kommunikation können grob in den Rime-Kommunikationsstack und den μIP TCP/IP-Stack unterteilt werden

Wie ein Auszug des Ordnerinhaltes des Unterverzeichnisses „\net“ in der Abbildung 5.4 zeigt, sind dort die Headerdatei „sicslowpan.h“ und die Quellcodedatei „sicslowpan.c“ abgelegt worden. Diese beiden Dateien realisieren den 6LoWPAN-Adaptionslayer und enthalten die benötigten Deklarationen, Definitionen und Funktionen. Der praktische Teil dieser Abschlussarbeit besteht darin, diese Dateien zu analysieren und entsprechend dem Protokollentwurf von Pascal Thubert und Jonathan W. Hui (siehe 4.1) anzupassen. Die nächsten Abschnitte beschreiben die softwaretechnische Realisierung des 6LoWPAN-Adaptionslayers und die Abläufe innerhalb des Programmcodes. Dabei werden allgemein die Codeanalyse und die eingesetzten Werkzeuge beschrieben.

```
Verzeichnis von C:\Users\Benutzer\contiki-2.5\core

17.09.2012  23:59    <DIR>          .
17.09.2012  23:59    <DIR>          ..
17.09.2012  21:56    <DIR>          cfs
17.09.2012  21:56    <DIR>          ctk
17.09.2012  21:56    <DIR>          dev
17.09.2012  21:56    <DIR>          lib
17.09.2012  21:56    <DIR>          loader
17.09.2012  21:56    <DIR>          net
17.09.2012  21:56    <DIR>          sys
06.09.2011  23:43             2.149 contiki.h
06.09.2011  23:43             1.973 contiki-lib.h
06.09.2011  23:43             2.255 contiki-net.h
06.09.2011  23:43             1.961 contiki-version.h
               5 Datei(en),          8.338 Bytes
               9 Verzeichnis(se), 154.952.298.496 Bytes frei
```

Abbildung 5.3: Verzeichnis „\core“ des Betriebssystems Contiki Version 2.5

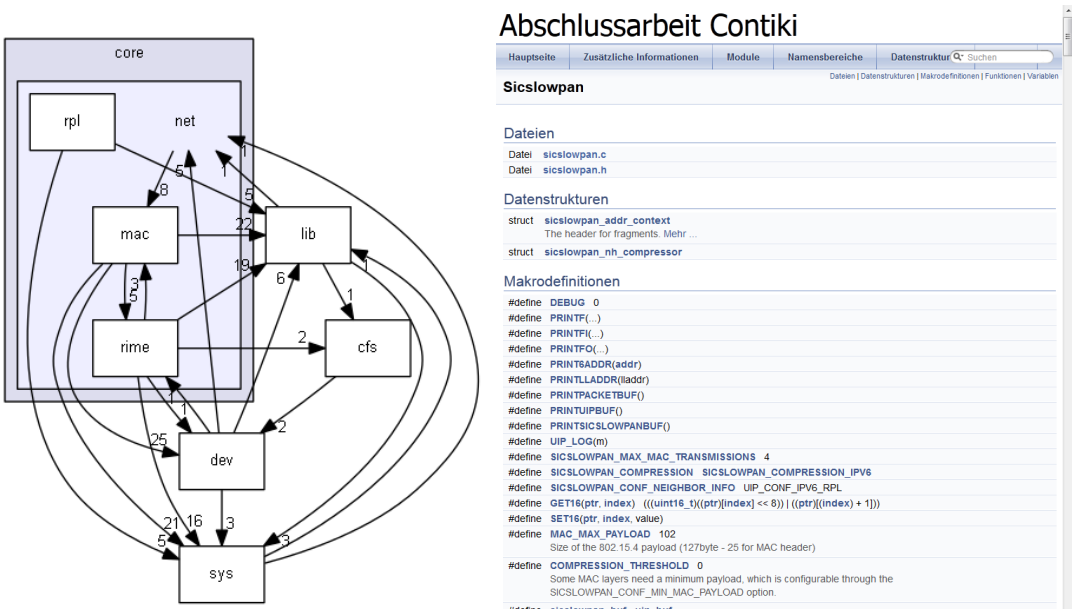
```
Verzeichnis von C:\Users\Benutzer\contiki-2.5\core\net

18.09.2012  00:29    <DIR>          .
18.09.2012  00:29    <DIR>          ..
17.09.2012  21:56    <DIR>          mac
                                    .
                                    .
                                    .
06.09.2011  23:43             3.720 rime.h
06.09.2011  23:43            63.830 sicslowpan.c
06.09.2011  23:43            10.914 sicslowpan.h
06.09.2011  23:43             7.696 simple-udp.c
                                    .
                                    .
                                    .
06.09.2011  23:43             3.181 uip-udp-packet.c
06.09.2011  23:43             2.177 uip-udp-packet.h
              71 Datei(en),        746.882 Bytes
               5 Verzeichnis(se), 155.031.814.144 Bytes frei
```

Abbildung 5.4: Verzeichnis „\core\net“ des Betriebssystems Contiki Version 2.5 - Auszug

5.1.3 Eingesetzte Werkzeuge bei der Quellcodeanalyse

Das Analysieren eines fremden Softwareprojekts, insbesondere, wenn die Programmiersprache C eingesetzt wurde, kann sehr zeitintensiv sein. Das Projekt Contiki ist im Laufe der Jahre derart umfangreich geworden, dass das Verzeichnis „\core“ über 300 Header- und Quellcodedateien enthält. Das in Abschnitt 4.2.5 vorgestellte Werkzeug Notepad++ bietet ein Feature, das ein komfortables Durchsuchen ganzer Verzeichnisse nach einer vorgegebenen Zeichenkette ermöglicht. Ein erheblicher Teil der Codeanalyse wurde auf diese Art und Weise praktiziert.

Der Quellcode des Softwareprojekts Contiki wurde von den Entwicklern ausführlich kommentiert. Zweifelsohne war eines der Ziele, fremden Entwicklern das Verständnis für die Arbeitsweise des Betriebssystems und der implementierten Algorithmen zu erleichtern. Darüber hinaus wurde durch die Art und Weise, wie der Quellcode kommentiert wurde, der Einsatz eines mächtigen Dokumentationswerkzeuges ermöglicht. Hierbei handelt es sich um das Programm Doxygen[2], das von Dimitri van Heesch entwickelt wurde. Doxygen unterstützt gängige Programmiersprachen und ermöglicht es, basierend auf den verwendeten Kommentaren innerhalb des Quellcodes eine Dokumentation des Softwareprojektes zu erstellen. Die Dokumentation wird dabei in gängigen Formaten wie bspw. PDF, PostScript, RTF oder HTML bereitgestellt. Darüber hinaus ist Doxygen in der Lage, aus undokumentiertem Quellcode Diagramme für die Abhängigkeiten, Vererbungen und die Interaktion zwischen den einzelnen Funktionen oder Dateien zu erstellen.



(a) Abhängigkeiten zwischen Verzeichnissen (b) Dokumentationsbeispiel basierend auf HTML

Abbildung 5.5: Diagramm und Quellcodedokumentation, erstellt mit Doxygen

Um dieses Programmfeature nutzen zu können, ist die Installation des Werkzeugs „dot“ aus dem GraphViz-Softwarepaket[3] notwendig. In der Abbildung 5.5 ist ein Beispiel für eine erstellte Dokumentation basierend auf HTML und ein Diagramm für die Abhängigkeiten der Unterverzeichnisse in dem Ordner „\core“ des Betriebssystems Contiki zu sehen. Für das Betriebssystem Contiki

[2] http://www.stack.nl/ dimitri/doxygen/index.html (Abrufdatum 18.9.2012)

[3] http://www.graphviz.org/ (Abrufdatum 18.9.2012)

werden im Internet mit Doxygen erstellte Dokumentationen[4][5] bereit gestellt. Wird das angebotene Feature der graphischen Aufbereitung des Quellcodes nicht benötigt, kann auf diese Quellen zurückgegriffen werden.

5.1.4 Relevante Deklarationen bzw. Definitionen

Ein besonderes Merkmal des Contiki-Quellcodes ist der intensive Gebrauch von Präprozessordefinitionen und –Makros. Ein Großteil der Betriebssystemkonfiguration, das Debugging der Softwaremodule und viele verwendete Konstanten basieren auf Makros und Definitionen des C-Präprozessors. Entsprechend der gewählten Konfiguration werden über die Makros Teile des Quellcodes ein- bzw. ausgeblendet. Dadurch ist es möglich, die verwendeten Hardwareressourcen in Form von Arbeits- und Programmspeicher zu minimieren. Die folgenden beiden Abschnitte beschreiben die für die Lösung der Aufgabenstellung relevanten Definitionen. Die Abschnitte vermitteln nur einen groben Überblick über die wichtigsten Definitionen, der entsprechende Quellcode ist im Anhang (CD) zu finden.

Die Headerdatei „sicslowpan.h“

In der Headerdatei „sicslowpan.h“ werden die durch den 6LoWPAN-Adaptionslayer verwendeten Dispatch-Bytes über eine Präprozessoranweisung definiert, verdeutlicht im Quellcodeausschnitt 5.1. Diese Dispatch-Bytes werden im Rahmen der Implementierung des Protokollentwurfs um weitere Definitionen ergänzt.

```
/**
 * \name 6lowpan dispatches
 * @{
 */
#define SICSLOWPAN_DISPATCH_IPV6                0x41 /* 01000001 = 65 */
#define SICSLOWPAN_DISPATCH_HC1                 0x42 /* 01000010 = 66 */
#define SICSLOWPAN_DISPATCH_IPHC                0x60 /* 011xxxxx = ... */
#define SICSLOWPAN_DISPATCH_FRAG1               0xc0 /* 11000xxx */
#define SICSLOWPAN_DISPATCH_FRAGN               0xe0 /* 11100xxx */
/** @} */
```

Listing 5.1: Definition der Dispatch-Bytes

In dieser Datei sind ebenso die Definitionen der innerhalb des 6LoWPAN-Adaptionslayers verwendeten Kompressionstechniken zu finden. Sie werden im Quellcodeausschnitt 5.2 dokumentiert. Die Kompressionstechniken spielen für die Lösung der Aufgabenstellung eine eher untergeordnete Rolle.

[4] http://dak664.github.com/contiki-doxygen/ (Abrufdatum 18.9.2012)

[5] http://contiki.sourceforge.net/docs/2.6/index.html (Abrufdatum 18.9.2012)

```
/**
 * \name 6lowpan compressions
 * @{
 */
#define SICSLOWPAN_COMPRESSION_IPV6       0
#define SICSLOWPAN_COMPRESSION_HC1        1
#define SICSLOWPAN_COMPRESSION_HC06       2
/** @} */
```

Listing 5.2: Definition der Kompressionstechniken

In den beiden Listings wird die Art und Weise der Quellcodekommentierung verdeutlicht, die für das Erstellen einer Dokumantation mit Doxygen empfohlen werden. Durch Doxygen werden allerdings auch alternative Formatierungen der Kommentarzeilen unterstützt.

Die Quellcodedatei „sicslowpan.c“

Die Präprozessordefinition `DEBUG` in der Datei „sicslowpan.c“ erlaubt das Ein- und Ausschalten der Debugginginformationen für den 6LoWPAN-Adaptionslayer. Bei aktiviertem Debugging werden über den Präprozessor verschieden formatierte Definitionen für den Ausgabebefehl „printf“ realisiert. Den entsprechenden Ausschnitt mit einem Teil der Definitionen des Ausgabebefehls zeigt der Quellcode 5.3

```
#define DEBUG 0
#if DEBUG
/* PRINTFI and PRINTFO are defined for input and output to debug one without
    changing the timing of the other */
u8_t p;
#include <stdio.h>
#define PRINTF(...) printf(__VA_ARGS__)
#define PRINTFI(...) printf(__VA_ARGS__)
#define PRINTFO(...) printf(__VA_ARGS__)
```

Listing 5.3: Ein- und Ausschalten des Debugging über die Variable `DEBUG`

Für das Senden bzw. Empfangen eines Datagramms durch den 6LoWPAN-Adaptionslayer ist der Zugriff auf zwei Paketbuffer, den Rime-Buffer und den μIP-Buffer, nötig. Durch den 6LoWPAN-Adaptionslayer wird kein separater Speicherbereich für zu sendende Datagramme reserviert. Für die Verarbeitung wird der μ-IP-Buffer verwendet. Der definierte 6LoWPAN-Buffer wird nur für den Empfang der 802.15.4-Datenrahmen und das Erzeugen der Datagramme eingesetzt.

```
/** \name Pointers in the rime buffer
 *  @{
 */
#define RIME_FRAG_PTR              (rime_ptr)
#define RIME_FRAG_DISPATCH_SIZE 0    /* 16 bit */
#define RIME_FRAG_TAG              2    /* 16 bit */
#define RIME_FRAG_OFFSET           4    /* 8 bit */
```

Listing 5.4: Pointerdefinition für den Zugriff auf den Rime-Buffer

Innerhalb des Quellcodes werden für den μIP-Buffer und den 6LoWPAN-Buffer mehrere Alias kreiert. Für das Analysieren und Erzeugen der 6LoWPAN-Header wird mittels Präprozessor definierter Konstanten auf den Rime-Buffer zugegriffen. Diese Konstanten sind für die Realisierung des Protokollentwurfs von hoher Bedeutung, da sie durch eigene Definitionen ergänzt werden. Der Quellcodeausschnitt 5.4 zeigt diese Definitionen.

5.1.5 Funktionen für das Senden und Empfangen der Datagramme

Der C-Quellcode des Betriebssystems Contiki, der den 6LoWPAN-Adaptionslayer realisiert, enthält zwei Funktionen, die für die Umsetzung des Protokollentwurfs von Pascal Thubert und Jonathan W. Hui modifiziert werden müssen. Die Methode für das Senden eines Datagramms ist im Rumpf der Funktion `static uint8_t output(uip_lladdr_t *localdest)` zu finden. Die Funktion `static void input(void)` ist hingegen für den Empfang eines Datagramms zuständig. Die folgenden Abschnitte geben eine allgemeine Beschreibung dieser Funktionen vor der Anpassung für die Realisierung des Protokollentwurfs.

Die Funktion `output()`

Diese Funktion hat die Aufgabe, ein Datagramm der Netzwerkschicht nach den Vorgaben des 6LoWPAN-Adaptionslayers aufzubereiten, so dass Datenrahmen über das 802.15.4-Netzwerk versandt werden können. Der Funktion wird die Ebene 2-Adresse des Zielknotens übergeben. Zurückgeliefert wird der Datentyp „unsigned Char“, der die Information über das erfolgreiche Versenden des Datagramms enthält.

Nach der Initialisierung der verwendeten Variablen und dem Zurücksetzen des Rime-Buffers wird für alle TCP-Nachrichten mit Ausnahme der FIN-Pakete der Stream-Mode aktiviert, wodurch eine Kommunikation in beide Richtungen erlaubt wird. Das zu übertragende Datagramm befindet sich im μIP-Buffer. Anschließend wird geprüft, ob die übergebene Ebene 2-Adresse den Wert „NULL“ enthält - sollte dies der Fall sein, wird angenommen, dass die Daten an alle Stationen gesendet werden sollen. Die Ebene 2-Adresse wird in eine Broadcast-Adresse (Kurzadresse 0xffff) umgewandelt. Nach dem Überprüfen eines definierten Schwellwertes für die Datenkompression, dieser ist standardmäßig auf null gesetzt, wird das konfigurierte Kompressionsverfahren angewandt. Die Definition eines Schwellwertes für die Datenkompression berücksichtigt alternative Sicherungsschichten, die ein minimales Datenvolumen für die Nutzdaten erwarten. Die angewandte Kompressionstechnik wird über Präprozessormakros bestimmt – nicht benötigter Programmcode wird hier während des Kompiliervorganges ausgeblendet. Voreingestellt ist das Verfahren IPv6, ein Synonym für einen unkomprimierten IPv6-Header.

Im Anschluss an eine mögliche Headerkompression wird überprüft, ob das Datagramm in einen einzigen 802.15.4-Rahmen passt. Ist diese Bedingung erfüllt, werden hinter dem während der Kompression erzeugten Header die Nutzdaten in den Rime-Buffer kopiert und der 802.15.4-Datenrahmen durch das Aufrufen der entsprechenden Funktion versandt.

Ist eine Fragmentierung des Datagramms nötig, werden die Kompressionsheader im Rime-Buffer verschoben, um Platz für den Fragmentierungsheader zu schaffen. Die Fragmentierungsoption des Datagramms kann über ein Präprozessormakro ein- bzw. ausgeschaltet werden. Nun wird der 6LoWPAN-Header für ein initiales Fragment (FRAG1) erzeugt und die Nutzdaten hinter dem Kompressionsheader in den Rime-Buffer kopiert. Das initiale Fragment wird anschließend über das Netzwerk an die Zielstation übertragen.

Nach dieser Übertragung wird der 6LoWPAN-Header für ein weiteres Fragment (FRAGN) erzeugt und der Sendevorgang innerhalb einer „While“-Schleife solange ausgeführt, bis das kom-

plette Datagramm versandt wurde. Mit jedem Schleifendurchlauf wird dabei das *datagram_offset* angepasst.

Sind alle Fragmente versandt worden, wird der Inhalt der Variablen, die das *datagram_tag* enthält, um eins erhöht und die Kontrolle an die aufrufende Funktion zurückgegeben.

Die Funktion `input()`

Diese Funktion wird vom Betriebssystem aufgerufen, wenn durch den Rime-Stack ein 6LoWPAN-Paket empfangen wurde. Ein empfangenes 6LoWPAN-Paket wird dekodiert und, falls es sich um mehrere Fragmente handelt, wieder korrekt zu einem Datagramm zusammengesetzt. Für diese Aufgabe wird ein spezieller Speicherbereich reserviert, der 6LoWPAN-Buffer. Nach der Initialisierung der für die Aufgabe benötigten Variablen wird als Erstes überprüft, ob ein aktuell zusammenzufügendes Datagramm mindestens 20 Sekunden alt ist. Falls diese Bedingung erfüllt ist, wird das Datagramm verworfen und der Reassembly-Vorgang abgebrochen.

In einem nächsten Schritt wird das Dispatch-Byte analysiert und die *datagram_size*, das *datagram_tag* und, falls vorhanden, das *datagram_offset* bestimmt. An dieser Stelle wird auch die Entscheidung getroffen, ob es sich um ein initiales Fragment (FRAG1), ein weiteres Fragment (FRAGN) oder sogar um das letzte Fragment eines Datagramms handelt.

Anschließend wird bestimmt, ob zur Zeit ein Reassembly-Prozesses durchgeführt wird. Falls ja, wird überprüft, ob das Fragment zu dem Datagramm gehört, das gerade zusammengesetzt wird. Ist dies nicht der Fall, wird es verworfen und die Funktion verlassen. Ist kein Reassembly-Prozess am Laufen, wird dieser Vorgang initiiert und der Timer gestartet, der für das Verwerfen zu alter Datagramme verantwortlich ist. Darüber hinaus wird das erwartete *datagram_tag* und die *datagram_size* zwischengespeichert.

Nach dieser Prüfung wird bestimmt, ob es sich um ein folgendes Fragment (FRAGN) handelt. Ist dies der Fall, wird ein Teil des Programmcodes übersprungen und mit dem Kopieren der Nutzdaten in den 6LoWPAN-Buffer fortgefahren. Handelt es sich allerdings um ein initiales Fragment (FRAG1) oder ein unfragmentiertes Datagramm, werden die folgenden Dispatch-Bytes analysiert und ggf. der IPv6-Header dekomprimiert. Ein Präprozessormakro entscheidet hier wieder, welches Komprimierungsverfahren verwendet wird. Nicht benötigter Programmcode wird während des Kompilierungsvorgangs durch den Präprozessor ausgeblendet.

Nach diesem Schritt wird, wie im Fall eines Folgefragments, mit dem Kopieren der Nutzdaten fortgefahren. Am Ende des Kopiervorganges erfolgt eine Prüfung, ob sich bereits ein komplettes Datagramm im 6LoWPAN-Buffer befindet. War diese Prüfung erfolgreich, wird das Datagramm in den μIP-Buffer kopiert und der μIP-Stack informiert. Das geschieht durch das Aufrufen der Funktion `void tcpip_input (void)`. Andernfalls wird die Kontrolle an das Kernel-Prozessmodul des Betriebssystems (process.c) zurück gegeben.

5.2 Umsetzung des Protokollentwurfs

Der Protokollentwurf „LoWPAN fragment Forwarding and Recovery“ von Pascal Thubert und Jonathan W. Hui wird durch das Anpassen der 6LoWPAN-Implementierung des Betriebssystems Contiki in der Version 2.5 umgesetzt. Der Entwurf schlägt eine Lösung für das Weiterleiten der Fragmente bei eingesetztem Routingverfahren „Route Over“ (siehe Abschnitt 2.2.3) vor. Der Fokus wird innerhalb dieser Abschlussarbeit auf die Implementierung der neuen Dispatch-Bytes, Header und der Retransmission-Funktionalität gelegt, auch wenn der Weiterleitungsmechanismus

im Abschnitt 4.1.5 beschrieben wurde. Darüber hinaus sind die verfügbaren Ressourcen der verwendeten Hardwareplattform durch die Verwendung eines eigenen 6LoWPAN-Retransmission-Buffers belegt.

Das Betriebssystem Contiki unterstützt zur Zeit keinen Mesh-Header. Für die Bewertung des Algorithmus werden daher aussagekräftige Parameter einer drahtlosen Verbindung zwischen dem AVR RZ USB Stick und dem AVR RZ RAVEN Board ermittelt, die im Kapitel 6 diskutiert werden.

Ziel ist es, den Protokollentwurf ausschließlich auf Ebene des 6LoWPAN-Adaptionslayers zu realisieren. Zusätzlich zu den Anforderungen in der Aufgabenstellung ist diese Entscheidung dem Umstand geschuldet, dass der Zugriff auf die Funktionen des Adaptionslayers an mehreren Stellen des Quellcodes der darüberliegenden Protokollschicht erfolgt. Abgesehen von den Schwierigkeiten bei der Realisierung würde das Anpassen der Codestellen außerhalb des 6LoWPAN-Adaptionslayers schwer nachvollziehbar sein und zu Unübersichtlichkeiten führen.

In den folgenden Abschnitten werden die Modifikationen der Quellcodedatei „sicslowpan.c“ beschrieben und Bezug auf die Quellcodeanalyse genommen. Der vollständige, modifizierte Quellcode ist im Anhang (CD) zu finden.

5.2.1 Implementieren der neuen Dispatch-Bytes und Header

Im Rahmen der Realisierung des Protokollentwurfs werden drei neue Dispatch-Bytes und Header implementiert. Der Entwurf schlägt noch ein viertes Dispatch-Byte für die Signalisierung einer Netzwerkauslastung durch den Acknowledgement-Header (RFRAG-ACK) vor (siehe Abschnitt 4.1.3). Diese Funktion wird allerdings nicht umgesetzt und dieser Header daher nicht übernommen.

Die neuen Dispatch-Bytes wurden auf die gleiche Art und Weise wie die zuvor durch die Entwickler definierten 6LoWPAN-Typen integriert. Zum Zwecke der Übersicht erfolgt dies allerdings nicht in der Headerdatei „sicslowpan.h“, sondern, wie sämtliche Definitionen für den Entwurf, in der Quellcodedatei „sicslowpan.c“. Der Quellcodeausschnitt 5.5 zeigt diese Präprozessoranweisungen.

```
/* Eigene Dispatch-Bytes        */
#define SICSLOWPAN_DISPATCH_RFRAG        0xe8 /* 11   101000 = 232   */
#define SICSLOWPAN_DISPATCH_RFRAG_AR     0xe9 /* 11   101001 = 233   */
#define SICSLOWPAN_DISPATCH_RFRAG_ACK    0xea /* 11   10101x = 234   */
/*                              */
```

Listing 5.5: Definition der Dispatch-Bytes für den Protokollentwurf

Für das Erzeugen und die Analyse der bisher durch den 6LoWPAN-Adaptionslayer bereitgestellten Header werden über den Präprozessor Konstanten für den Zugriff auf spezifische Elemente des Rime-Buffers definiert. So ist es bspw. möglich, das Datagramm-Tag aus dem Buffer auszulesen oder es dort hineinzuschreiben. Da es sich um einen Integerwert mit 16 Bit handelt, werden dafür die Präprozessormakros `GET16(ptr,index)` und `SET16(ptr,index,value)` eingesetzt. Für die Header des Protokollentwurfs (siehe 4.1.3) werden zum Zwecke des Zugriffs auf den Rime-Buffer eigene Konstanten definiert, die in dem Quellcodeauszug 5.6 zu sehen sind.

```
/* Eigene Pointer auf den Rime-Buffer */
#define RIME_RFRAG_DISPATCH          0
#define RIME_RFRAG_OFFSET            1
#define RIME_RFRAG_TAG               2
#define RIME_RFRAG_SEQUENCE_SIZE     4
#define RIME_RFRAG_ACK_DISPATCH      0
#define RIME_RFRAG_ACK_TAG           1
#define RIME_RFRAG_ACK_BMP_HIGH      3
#define RIME_RFRAG_ACK_BMP_LOW       5
/*                                  */
```

Listing 5.6: Definition der Pointer auf den Rime-Buffer für den Protokollentwurf

Für die Verarbeitung der Header während des Sendens der Fragmente und innerhalb des Prozesses des Zusammenfügens der Fragmente zu einem Datagramm sind nicht nur Angaben über den Headeraufbau und die Position der verschiedenen Informationen innerhalb des Headers nötig. Für die Berechnung der bereits verarbeiteten Bytes eines Datagramms und für den Zugriff auf die Position der Nutzdaten wird die Länge der verschiedenen Header-Typen benötigt. Der Quellcodeausschnitt 5.7 zeigt die für den Protokollentwurf definierten Header-Längen.

```
/* Eigene Definitionen für die Headerlaenge */
#define SICSLOWPAN_RFRAG_HDR_LEN                6      /*one byte*/
#define SICSLOWPAN_RFRAG_AR_HDR_LEN             6
#define SICSLOWPAN_RFRAG_ACK_HDR_LEN            7
/*                                          */
```

Listing 5.7: Definition der Headerlängen für den Protokollentwurf

5.2.2 Modifikationen der Funktion `output()`

Diese Funktion ist, wie im Abschnitt 5.1.5 beschrieben, für das Senden der Datagramme unter Verwendung des 6LoWPAN-Adaptionslayers verantwortlich. Bei der Realisierung des Protokollentwurfs wurde möglichst viel des originalen Quellcodes wiederverwendet und der Programmierstil der Entwickler des Betriebssystems Contiki beibehalten.

In einem ersten Schritt wird überprüft, ob bereits ein Datagramm unter Verwendung des Protokollentwurfs übertragen wird. Falls ja, wird die Funktion sofort wieder verlassen und dem aufrufenden Programmcode ein Fehler signalisiert. Das erneute Senden eines Datagramms wird erst möglich, wenn der Retransmissionprozess abgebrochen oder erfolgreich beendet wurde. Handelt es sich bei der Ebene 2-Zieladresse um keine Broadcastadresse oder soll die Übertragung unter Einsatz des Protokollentwurfs durchgeführt werden, wird das gesamte Datagramm in den zusätzlich definierten 6LoWPAN-Retransmission-Buffer kopiert. In diesem Zusammenhang werden auch Informationen wie das Datagramm-Tag, die Paketlänge und die Zieladresse zwischengespeichert. Das Kopieren des gesamten Datagramms ist notwendig, da die Reihenfolge, mit der die Fragmente am Zielknoten eintreffen, willkürlich sein kann. Darüber hinaus ist es nicht vorhersagbar, welches der Fragmente im Falle eines Übertragungsfehlers erneut gesendet werden muss.

Der Quellcodeabschnitt des Betriebssystems bis zu der Entscheidung, ob eine Fragmentierung des Datagramms notwendig ist, blieb unverändert. Hier erfolgt nun erneut eine Prüfung, ob der Protokollentwurf zu verwenden ist oder ob es sich bei der Ebene 2-Zieladresse um keine Broadcastadresse handelt. Führt diese Prüfung zu einem negativen Resultat, wird der Quellcode der Origi-

nalimplementierung ausgeführt. Anderenfalls wird der neue Programmabschnitt abgearbeitet, der den Entwurf realisiert.

Dieser Programmabschnitt ist mit der Originalimplementierung vergleichbar. Die benötigten Header werden auf ähnliche Art und Weise erzeugt. Für den Zugriff auf den Rime-Buffer und die Berechnung der bereits verarbeiteten Bytes des Datagramms werden jedoch die neu eingeführten Konstanten verwendet (siehe Abschnitt 5.2.1). Die neue Position des *datagram_offset* und der *datagram_size* innerhalb der Header sowie das Verwenden einer Sequenznummer für die Fragmente zum Zweck der Neuanforderung machten einige Veränderungen der Berechnungen nötig.

Mit dem Senden eines Fragments wird die im Rahmen des Protokollentwurfs neu eingeführte Sequenznummer um eins erhöht. Für das künstliche Erzeugen eines fehlerhaften Rahmens/Fragmentes kann über ein Präprozessormakro zusätzlicher Programmcode eingefügt werden, der im Abschnitt 5.2.5 näher erläutert wird. Diese Option wurde auch beim Senden der Fragmente unter Verwendung der Originalimplementierung eingefügt.

Nach dem Senden des letzten Fragments wird die Anzahl der übertragenen Fragmente festgehalten und der neu eingeführte Prozess, der für die Retransmission der Fragmente notwendig ist, gestartet. Dieser Schritt wird nur durchgeführt, wenn eine Bestätigung der Fragmente erwünscht ist. Die Umsetzung der Retransmissions und Acknowledgements wird im Abschnitt 5.2.4 detailiert beschrieben. Wie bei der Originalimplementierung wird im Anschluss das *datagram_tag* um eins erhöht und die Kontrolle an die aufrufende Funktion zurückgegeben.

5.2.3 Modifikationen der Funktion `input()`

Diese Funktion ist für den Empfang der IP-Pakete und falls erforderlich, für das Zusammensetzen der einzelnen Fragmente zu einem vollständigen Datagramm verantwortlich. Wie auch bei der Funktion für das Senden eines Datagramms, wurde der ursprüngliche Programmierstil der Contiki-Entwickler beibehalten und um die Möglichkeit, den zu implementierenden Protokollentwurf korrekt auswerten zu können, erweitert.

Der in Abschnitt 5.1.5 beschriebene Schritt der Dispatch-Byte-Analyse wurde für die neuen Header des Protokollentwurfs erweitert. Um die *datagram_size*, das *datagram_offset* und die neu eingeführte Sequenznummer ermitteln zu können, mussten die entsprechenden Berechnungen leicht angepasst werden.

Für den Fall, dass ein Bestätigungsrahmen des Protokollentwurfs empfangen wurde, existiert nun ein Programmabschnitt, der seine Informationen auswertet. Das Datagramm-Tag wird ermittelt und mit dem zusammenzusetzenden Datagramm verglichen. Handelt es sich um einen erwarteten Bestätigungsrahmen, wird das darin enthaltene Bestätigungsbitmuster (32 Bit) ausgewertet und die vom Kommunikationspartner bestätigten Fragmente in zwei dafür vorgesehenen Variablen (je 16 Bit) gespeichert. Auf Grundlage dieses Bitmusters wird eine mögliche Retransmission durchgeführt.

Da es sich um einen prototypisch implementierten Protokollentwurf handelt, wird der dort vorgestellte Komprimierungsalgorithmus (siehe Abschnitt 4.1.3) für das Bestätigungsbitmuster nicht implementiert. Dieser Komprimierungsalgorithmus würde zu einem Mehraufwand an Rechenleistung und Arbeitsspeicher führen. Die Ressourcen an Arbeitsspeicher wurden durch die Einführung eines separaten Retransmission-Buffers bereits ausgeschöpft.

Der Vorschlag des Protokollentwurfs beschreibt eine Kodierung des komprimierten Bestätigungsbitmusters für bis zu 28 Fragmente – der Entwurf selbst erfordert jedoch eine Fragmentierung in bis zu 32 Fragmente. Der Entwurf weist darauf hin, dass der Komprimierungsalgorithmus für zukünftige Erweiterungen angepasst werden kann. Für die Bestätigung der empfangenen Fragmente wird im Entwurf der neu eingeführte Acknowledgement-Header RFRAG-ACK verwendet. Ein solcher Datenrahmen enthält allerdings keine Nutzinformationen. Wurde das Bestätigungsbitmuster ausgewertet, wird der zuvor gestartete Prozesse für das Senden möglicher Retransmissions informiert. Im Anschluss wird die Kontrolle an das Betriebssystem zurückgegeben.

Das Initiieren eines Reassembly-Vorgangs wurde um das Zurücksetzen des Bestätigungsmusters, das in einem Bestätigungsrahmen übertragen wird, erweitert. Zusätzlich wird der Prozess für das Senden einer Bestätigung, falls gewünscht, gestartet. Die Entscheidung, ob der Kommunikationspartner eine Bestätigung erwartet, wird in der prototypischen Implementierung des Protokollentwurfs an dieser Stelle getroffen.

Der weitere Programmablauf folgt weitestgehend dem der Originalimplementierung, nur unter Berücksichtigung der neuen Header. In dem Abschnitt für das Kopieren der Nutzdaten wurde eine Funktionalität realisiert, die im Falle des Empfangs eines Fragments basierend nach dem Protokollentwurf die enthaltene Sequenznummer auswertet. Die in den Fragmenten enthaltene Sequenznummer wird in einem zweiten Bestätigungsbitmuster durch das Setzen des entsprechenden Bits vermerkt. Diese Information wird ähnlich wie bei einem empfangenen Bestätigungsrahmen in zwei Variablen (16 Bit) gespeichert. Dieses Bitmuster entspricht der in einem Bestätigungsrahmen übertragenen Bitmap.

Wurden sämtliche Fragmente eines Datagramms empfangen und wird ihre Bestätigung durch den Sender erwartet, wird der Prozess, der für das Senden der Bestätigungsrahmen (siehe Abschnitt 5.2.4) verantwortlich ist, informiert. Wie bei der Originalimplementierung wird im Anschluss der μIP-Stack über das eingetroffene IP-Paket informiert.

5.2.4 Realisierung der Acknowledgements und der Retransmissions

Für die Umsetzung der Funktionalitäten Acknowledgement und Retransmission eventuell verlorengegangener Pakete wurden verschiedene Möglichkeiten in Betracht gezogen und testweise implementiert. Die Schwierigkeit bei der Implementierung ergibt sich aus dem Umstand, dass es innerhalb des Quellcodes in der Datei „sicslowpan.c“ nicht möglich ist, die Kontrolle über die gesendeten und empfangenen Datagramme zu behalten und auf eintreffende Bestätigungsrahmen reagieren zu können. Der Versuch, ausstehende Events durch Aufrufen der Funktion `int process_run (void)` abzuarbeiten und anschließend den eigenen Programmcode auszuführen, war nur auf dem AVR RZ RAVEN Board lauffähig. Auf dem AVR RZ USB Stick führte diese Lösung zu einem Stillstand des Systems. Eine Lösung, die ausschließlich auf „Protothreads“ basierte, wurde aufgrund der fehlenden, einfachen Integrationsmöglichkeit in das bestehende Betriebssystem verworfen. Realisiert wurden die beiden Funktionen durch das Starten zweier entkoppelter Prozesse unter Einsatz der durch die Contiki-Entwickler bereitgestellten Bibliothek „process.h“.

Der sogenannte „Protothread“ ist ein Mechanismus, der innerhalb des Betriebssystems Contiki eingesetzt wird, um die parallele Programmierung in einer ereignisgesteuerten Umgebung zu realisieren. „Protothreads“ belegen auf den eingesetzten Systemen wenige Ressourcen an Arbeitsspeicher und kommen ohne eigenen Stack aus. Dadurch ist das Verwenden lokaler Variablen nur einge-

schränkt möglich. Die Protothreads wurden von Adam Dunkels[6] entwickelt, eine Dokumentation[7] wurde auf seinem Internetauftritt bereitgestellt. Realisiert werden die Protohreads über Präprozessormakros, hinter denen sich spezifische Codesegmente verbergen. Der C-Quellcode hinter diesen Makros entspricht einer Switch-Anweisung, über welche die ereignisorientierte Steuerung der einzelnen Threads realisiert wird.

In Contiki werden Prozesse[8] über einen einzelnen Protothread umgesetzt. Dabei wird eine spezielle Implementierung mit veränderten Präprozessormakros verwendet. Dadurch wird es Prozessen möglich gemacht, auf eintreffende Ereignisse zu warten. Prozesse können in zwei Betriebsmodi ausgeführt werden, kooperativ und bevorzugt. Der bevorzugte Kontext wird jedoch nur in den Interrupt-Handlern der Gerätetreiber und für zeitkritische Echtzeitanwendungen eingesetzt.

Prozesse des Betriebssystems Contiki können auf zwei Arten von Ereignissen reagieren, auf synchrone und asynchrone. Asynchrone Ereignisse werden dabei in die Prozesswarteschlange des Betriebssystemkerns eingereiht, wohingegen synchrone Ereignisse den entsprechenden Prozess direkt aufrufen. Ein besonderes Ereignis ist das Poll-Request, welches nur bei Prozessen im bevorzugten Modus eintreten kann. Hierbei wird der zugehörige Prozess so schnell wie möglich informiert.

Der Prozess für das erneute Senden eventuell verloren gegangener Fragmente wird am Ende der Funktion `static uint8_t output(uip_lladdr_t *localdest)` gestartet und beim Eintreffen eines Bestätigungsrahmens oder beim Ablauf des Retransmission-Timers informiert. Die Zeit bis zum Auslaufen dieses Timers wird in der Konstanten `SICSLOWPAN_SRC_TIMER` definiert und ist mit 375 ms voreingestellt. Wurde der Prozessstart initiiert, wird das Senden weiterer Datagramme verhindert. Innerhalb des Prozesses wird der Retransmission-Timer neu initialisiert. Wird der Prozess abgearbeitet, erfolgt eine Prüfung, welche der gesendeten Fragmente bereits durch den Empfänger bestätigt wurden. Unbestätigte Fragmente werden nun erneut übertragen. Wurde die in der Definition `SICSLOWPAN_MAX_RETRANSMIT` (voreingestellt ist der Wert 5) festgelegte Anzahl an neuen Übertragungsversuchen erreicht oder sämtliche Fragmente quittiert, beendet sich der Prozess selbst. Das Senden neuer Datagramme ist nun wieder möglich.

In der Funktion `static void input(void)` wird der Prozess für das Senden der Bestätigungen mit dem Empfang des ersten Fragments eines Datagramms gestartet. Wurden sämtliche Fragmente eines IP-Paketes empfangen oder läuft der definierte Acknowledgement-Timer aus, wird der Prozess informiert. Die Zeit bis zum Auslaufen des Timers wird durch die Präprozessordefinition `SICSLOWPAN_ACK_TIMER` angegeben und ist mit 187 ms voreingestellt. Innerhalb des Prozesses wird der Bestätigungsrahmen RFRAG-ACK auf Grundlage der bereits empfangenen Datagrammfragmente erzeugt und an den Sender übertragen. Solange eine Bestätigung gefordert wird und die Anzahl der gesendeten Bestätigungrahmen kleiner ist als der Wert in der Definition `SICSLOWPAN_MAX_ACKNOWLEDGEMENT` (voreingestellt ist der Wert 10), ruht der Prozess bis zum nächsten Aufruf. Ist diese Bedingung nicht mehr erfüllt, wird der Prozess beendet.

Die Einstellungen für den Retransmission-Timer und den Acknowledgement-Timer wurden so gewählt, dass die Übertragung eines Datagramms in möglichst kurzer Zeit durchgeführt werden kann. Dabei wurde experimentell der Wert bestimmt, bei denen das Betriebssystem Contiki noch fehlerfrei arbeitete. Der Wert für die Anzahl der Retransmissionversuche wurde empirisch bestimmt, im Hinblick auf den Umstand, dass höhere Protokollschichten ihrerseits Daten erneut

[6] http://www.dunkels.com/adam/ (Abrufdatum 26.9.2012)

[7] http://dunkels.com/adam/download/pt-1.4-refman.pdf (Abrufdatum 26.9.2012)

[8] http://www.sics.se/contiki/wiki/index.php/Processes (Abrufdatum 26.9.2012)

senden könnten. Darüber hinaus sollten die Bewertungsmessungen des Kapitels 6 in einem akzeptablen Zeitrahmen durchführbar sein. Eine Datagrammübertragung wird daher nach 5 Versuchen oder ungefähr 2 s abgebrochen. Die Anzahl der gesendeten Bestätigungen wird durch die Formel

$$\frac{\left(\frac{1000ms}{2^2}+\frac{1000ms}{2^3}\right)\cdot SICSLOWPAN\_MAX\_RETRANSMIT}{\left(\frac{1000ms}{2^3}+\frac{1000ms}{2^4}\right)}$$ angenähert und ergibt den Wert 10.

Diese Formel ist aus der Berechnung der Werte für die Initialisierung der verwendeten Timer innerhalb des Quellcodes abgeleitet. Der geklammerte Term über dem Bruchstrich entspricht der Ablaufzeit des Retransmission-Timers (in ms) und der geklammerte Term unterhalb des Bruchstrichs der Ablaufzeit des Acknowledgement-Timers (in ms). Der Quellcodeausschnitt 5.8 verdeutlicht die entsprechenden Definitionen durch den Präprozessor.

```
// Zeit zwischen zwei Acknowledgementframes
#define SICSLOWPAN_ACK_TIMER (CLOCK_SECOND>>3)+(CLOCK_SECOND>>4)
// Zeit zwischen zwei Retransmissions
#define SICSLOWPAN_SRC_TIMER (CLOCK_SECOND>>2)+(CLOCK_SECOND>>3)
```

Listing 5.8: Definitionen für die Initialisierung der Timer

5.2.5 Funktionalitäten für die Konfiguration des Protokollentwurfs und für die Analyse

Für die Konfiguration der Implementierung und für die Bewertung des Protokollentwurfs wurden mehrere Definitionen in der Quellcodedatei „sicslowpan.c“, dargestellt im Quellcodeausschnitt 5.9, eingeführt. Darüber werden die in diesem Abschnitt beschriebenen Funktionen realisiert.

```
/* Konfigurationsdefinitionen */
#define ACK_MODE                2
#define RANDOM                  0x007f
#define DEBUGDRAFT              0
#define GENERATEERRORS          0
/*                             */
```

Listing 5.9: Definitionen für die Konfiguration des Protokollentwurfs

Über die Definition `ACK_MODE` kann dabei festgelegt werden, ob die Originalimplementierung (Wert 0), der Protokollentwurf ohne Bestätigungsanforderung (Wert 1) oder der Protokollentwurf mit Bestätigungsanforderung (Wert 2) verwendet werden soll. Im Rahmen dieser Abschlussarbeit sind nur die Modi 0 und 2 relevant. Die Konstante `DEBUGDRAFT` wird verwendet, um ein spezifisches Debugging zu gewährleisten. Dieser Schritt wurde nötig, da sich die Wünsche an eine Debuggingausgabe im Rahmen der Protokollimplementierung von denen der Contiki-Entwickler unterscheiden. Der derzeitige Stand der Implementierung ermöglicht die Ausgabe von Informationen im Rahmen einer Retransmission. Der dafür nötige Programmcode wird über ein Präprozessormakro hinzugefügt. Jeder zugewiesene Wert ungleich Null realisiert dabei das Debugging - auf dem AVR RZ USB Stick wird die Ausgabe allerdings unterdrückt.

Die Definition `GENERATEERRORS` erzeugt künstlich Fehler in den Fragmenten. Realisiert wird diese Funktion durch eine einfache Bedingung, dargestellt im Quellcodeausschnitt 5.10. Dabei wird eine Zufallszahl, basierend auf der Summe der Systemzeit in Clock-Ticks und der Sequenznummer, UND-verknüpft mit der in der Definition `RANDOM` gespeicherten Bitmap, erzeugt. Am

Beispiel des hexadezimalen Wertes für RANDOM von 0x007f wird deutlich, dass im Durchschnitt nach 127 korrekt gesendeten Fragmenten eine fehlerhafte Übertragung durch die Nichtausführung der Funktion send_packet(&destination) erzwungen wird.

```
#if GENERATEERRORS
                    if((clock_time()+sequence_ack)&RANDOM){
#endif
                     send_packet(&dest);
#if GENERATEERRORS
                    }
#endif
```

Listing 5.10: Quellcodeausschnitt für die Generierung von Übertragungsfehlern

Auch wenn die Funktion eine zufällige Komponente enthält, kann dieser Durchschnittswert bei genügend hoher Anzahl von gesendeten Fragmenten angenommen werden. Die im Kapitel 6 durchgeführten Messungen im Rahmen der Bewertung der Protokollimplementierung können diese Fehlergenerierungsfunktion anstelle der Entfernung zwischen den Stationen als Parameter verwenden. Die Filtermaske, sofern sinnvoll gewählt, spiegelt dabei die Anzahl der fehlerfrei übertragenen Datenrahmen wieder - oder im Umkehrschluss, die Anzahl der generierten Fehler. Wird der Definition GENERATEERRORS der Wert 0 zugewiesen, werden keine Fehler generiert. Der Wert 1 erzeugt fehlerhafte Fragmente, der Wert 2 wendet den Mechanismus auch auf die Bestätigungsrahmen an.

5.3 Testszenarien für den Vergleich der Implementierungen

Nach erfolgreicher Implementierung des Protokollentwurfs sieht es die Aufgabenstellung vor, die beiden Implementierungen miteinander zu vergleichen. Hierfür ist es angedacht, Parameter wie die maximale Übertragungsreichweite, die Übertragungszeit, die Paketverlustrate und den Datendurchsatz zu bestimmen.

Für das Bestimmen der maximalen Übertragungsrate wäre es denkbar, verschiedene Tests in definierten Entfernungen für beide Protokollimplementierungen durchzuführen und die Ergebnisse gegenüberzustellen. Im Vorfeld sollen Messungen zeigen, ob dieses Verfahren nachvollziehbare und zuverlässige Resultate liefert. Diese Messungen werden im Rahmen der Realisierung des Konzeptes durchgeführt. Sollten die Resultate nicht überzeugen, wären zwei alternative Ansätze denkbar. Zum einen wäre als Parameter für die Messungen die Empfangsfeldstärke heranzuziehen. Das durch die Entwickler bereitgestellte Anwendungsbeispiel „Webserver-ipv6-raven“ bietet hier die Möglichkeit, die Received Signal Strength Indication (RSSI) als Indikator für die Empfangsfeldstärke auszuwerten und die Nachvollziehbarkeit der Messergebnisse zu gewährleisten. Der zweite Lösungsansatz wäre, im Rahmen der Implementierung eine Funktion zu realisieren, die definierte fehlerhafte Datenrahmen erzeugt oder eine bestimmbare Anzahl von Rahmen nicht überträgt. Beide Ansätze wären vorstellbar, um verlorengegangene Fragmente durch die zunehmende Entfernung zwischen Sender und Empfänger zu simulieren.

Für das Bestimmen von Übertragungszeit, Paketverlustrate und Datendurchsatz in Abhängigkeit von der Entfernung muss eine Datenübertragung zu dem AVR RZ RAVEN realisiert und ausgewertet werden. Eine einfache und auf allen Plattformen verfügbare Möglichkeit stellt die Ping-Applikation basierend auf dem Protokoll ICMPv6 dar. Durch die Angabe spezifischer Parameter können die Anzahl der übertragenen Datenbytes, die Anzahl der gesendeten Echo-Requests und

das Zeitverhalten beeinflusst werden. Die Ping-Anwendung wertet empfangene Echo-Replys aus und liefert als Ergebnis relevante Angaben über die minimale, maximale und durchschnittliche Round Trip Time (RTT) sowie über die Paketverlustrate.

Anstatt das Protokoll ICMPv6 einzusetzen kann der Datenaustausch auch auf Ebene der Anwendungsschicht erfolgen. Um das Verhalten und die Leistungsfähigkeit von Webseiten und Webapplikationen messen zu können, wurde das Werkzeug Apache JMeter™[9] entwickelt. Da Apache JMeter™ auf der Programmiersprache Java basiert, ist diese Anwendung an keine Hardwareplattform gebunden.

In Apache JMeter™ können sogenannte Testpläne erstellt werden, die das Senden einer Reihe von HTTP-Requests an den im AVR RZ RAVEN Board implementierten „Webserver-ipv6-raven“ ermöglichen. Vergleichbar mit dem Ping-Befehl kann die minimale, maximale und durchschnittliche Zeitdauer, die für das Ausführen der Anfrage benötigt wird, ermittelt werden. Darüber hinaus können die Messreihen weitere Parameter der Übertragung, die eine Aussage über die Performance des Webservers liefern, graphisch aufbereitet dargestellt werden. Für diese Testpläne stehen verschiedene Logik-, Auswerte- und Steuerelemente wie bspw. Listener, Sampler, Timer und Graphen zur Verfügung. Eine detaillierte Anleitung[10] für das Erstellen eines Testplanes wurde durch die Entwickler auf der Internetseite des Softwareprojektes bereitgestellt.

Auf Grundlage beschriebener Testmethoden wird eine definierte Anzahl von Messungen durchgeführt. Wird hier eine genügend große Anzahl von Messungen ausgewertet, kann eine repräsentative Aussage für das Übertragungsverhalten gewonnen werden. Um die Dauer der Messungen in einem akzeptablen Rahmen halten zu können, werden für den Einsatz des Protokolls ICMPv6 1000 Wiederholungen und für die Webanfrage unter Verwendung des Werkzeuges Apache JMeter™ 100 Wiederholungen veranschlagt.

5.4 Messaufbau für Bewertung und Analyse

Der in Abbildung 5.6 dargestellte Messaufbau wird für die Analyse mit dem Werkzeug Wireshark (Abschnitt 6.1) und die Bewertung des Protokollentwurfs (Abschnitte 6.2 und 6.3) verwendet. Neben dem AVR RZ RAVEN Board und zwei AVR RZ USB Sticks (Jackdaw) werden die zwei in den Abschnitten 4.2.1 und 6.2.1 vorgestellten Computerplattformen AT5IONT-I und D945GSEJT eingesetzt. Der auf dem Computersystem AT5IONT-I installierte AVR RZ USB Stick wird dabei im Sniffermodus betrieben und ist für die Wiresharkanalyse der übertragenen Informationen des 6LoWPAN-Adaptionslayers und des Standards IEEE 802.15.4 gedacht. Für die Analyse der darüber liegenden Ebenen des OSI-Referenzmodells wird eine separate Wiresharkinstallation auf dem Gerät D945GSEJT unter Einsatz des zweiten AVR RZ USB Sticks verwendet.

Die im Rahmen der Protokollbewertung einzusetzenden Programme „Ping6“ und „JMeter“ werden auf dem Gerät D945GSEJT ausgeführt. Für die softwaretechnische Realisierung der linklokalen Router Advertisements und zum Zwecke der Vergabe einer globalen IPv6-Adresse wurde auf dem Gerät D945GSEJT der Router Advertisement Daemon (radvd) installiert. Das Listing 5.11 zeigt die für den Daemon verwendete Konfiguration. Das Listing 5.12 dokumentiert das eingesetzte Shell-Skript, um den Daemon zu starten und dem über den AVR RZ USB Stick erreichbaren Netzwerk ein IPv6-Adresspräfix zuzuweisen. Das AVR RZ RAVEN Board ist über die Adresse `fd00:142::11:22ff:fe33:4455` und der auf dem Gerät D945GSEJT installierte AVR RZ USB Stick über die Adresse `fd00:142::1` ansprechbar.

[9] http://jmeter.apache.org/ (Abrufdatum 15.9.2012)

[10] http://jmeter.apache.org/usermanual/build-web-test-plan.html (Abrufdatum 15.9.2012)

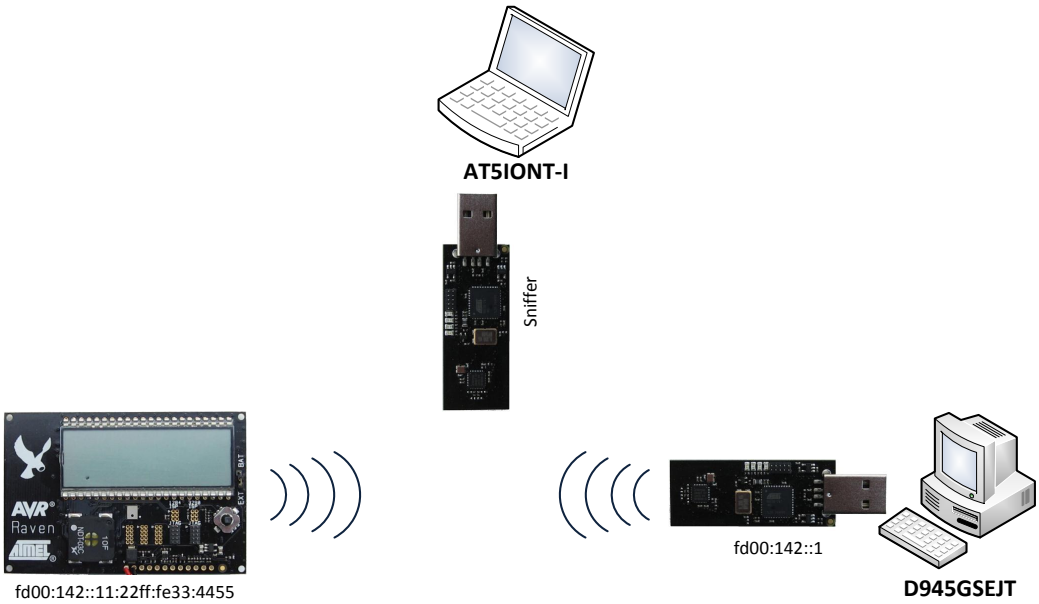


Abbildung 5.6: Messaufbau für die Bewertung und Analyse des Protokollentwurfs

```
interface usb0
{
    AdvSendAdvert on;
    AdvLinkMTU 1280;
    AdvCurHopLimit 128;
    AdvReachableTime 360000;
    MinRtrAdvInterval 100;
    MaxRtrAdvInterval 150;
    AdvDefaultLifetime 200;
    prefix fd00:142::/64
    {
        AdvOnLink on;
        AdvAutonomous on;
        AdvPreferredLifetime 4294967295;
        AdvValidLifetime 4294967295;
    };
};
```

Listing 5.11: Inhalt der Konfigurationsdatei „radvd.conf“

```
sudo ip -6 address add fd00:142::1/64 scope link dev usb0
sudo radvd start
```

Listing 5.12: Shellskript „jackdaw.sh“ für die IPv6-Adresskonfiguration

Kapitel 6

Bewertung und Diskussion

6.1 Analyse mit Wireshark

Die Analyse des Netzwerkverkehrs zwischen dem AVR RZ RAVEN Board und dem AVR RZ USB Stick gemäß dem Messaufbau in Abschnitt 5.4 wurde mit dem Netzwerkanalysewerkzeug Wireshark durchgeführt. Dabei wurde auf den beiden Stationen die 6LoWPAN-Originalimplementierung des Betriebssystems Contiki in der Version 2.5 und der Protokollentwurf „LoWPAN fragment Forwarding and Recovery“ von Pascal Thubert und Jonathan W. Hui in den Festspeicher programmiert und die gewonnenen Ergebnisse gegenübergestellt.

6.1.1 Fragmentierter Datenverkehr - Originalimplementierung

Die Tabelle 6.1 zeigt die Zusammenfassung des Wireshark-Mitschnittes der originalen 6LoWPAN-Implementierung. Da es sich um ein bekanntes Protokoll handelt, ist das Werkzeug in der Lage, die 6LoWPAN-Header zu dekodieren. Hier wurde ein Echo-Request mit der Länge von 512 Bytes vom AVR RZ USB Stick an das AVR RZ RAVEN Board gesendet. Das ursprüngliche Datagramm wurde durch den μIP-Stack im AVR RZ USB Stick in die 6LoWPAN-Pakete 1 bis 6 zerlegt. Nach erfolgreichem Reassembly im AVR RZ RAVEN Board und Auswerten des Datagramms und der darin enthaltenen ICMPv6-Nachricht zeigen die Pakete 7 bis 12 die fragmentierte ICMPv6-Echo-Reply-Antwort, die an den AVR RZ RAVEN USB Stick gesendet wurde. Das Paket 13 zeigt die ICMPv6-Echo-Reply-Nachricht nach dem Reassembly-Prozess. Die Pakete ab Nummer 14 gehören bereits zu einem neuen abgesetzten ICMPv6-Echo-Request mit der Sequenznummer 2.

In der Abbildung 6.1 wurde beispielhaft das erste 6LoWPAN-Fragment des ICMPv6-Echo-Requests mit der Sequenznummer 1 detailliert dargestellt. Der markierte Bereich des Hexdumps zeigt anschaulich den Beginn des Fragmentierungs-Headers für ein initiales Fragment wie in Abschnitt 2.2.7 erläutert. Die Bits der ersten beiden Bytes können entsprechend ihrer Bedeutung dem Dispatch-Byte und der Datagrammgröße zugeordnet werden. Die Bits 11000_2 kodieren das initiale Fragment und die Bits 01000110000_2 die Datagrammgröße von 560. Die nächsten beiden Byte-Positionen ergeben dem Headeraufbau zufolge das Datagram-Tag von 30.

6.1.2 Fragmentierter Datenverkehr - Protokollentwurf

Die Tabelle 6.2 zeigt die Zusammenfassung des Wireshark-Mitschnittes für den Protokollentwurf. Für Wireshark handelt es sich bei dem Entwurf um ein unbekanntes Protokoll. Daher wird der Netzwerkverkehr dem Standard IEEE 802.15.4 zugeordnet. Wie bei der originalen Implementierung wurde an das AVR RZ RAVEN Board ein ICMPv6-Echo-Request mit einer Länge von 512 Bytes an Nutzdaten gesendet. Die dargestellten IEEE 802.15.4-Rahmen 2 bis 8 verdeutlichen, dass das Datagramm wieder in sechs Teilen übertragen wird. Die Rahmen 9 bis 14 gehören zu

Tabelle 6.1: Wiresharkmitschnitt Echo-Request (512 Bytes) - Originalimplementierung

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	6LoWPAN	133	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
2	0.006977000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	6LoWPAN	138	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
3	0.015980000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	6LoWPAN	138	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
4	0.023970000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	6LoWPAN	138	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
5	0.032969000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	6LoWPAN	138	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
6	0.039965000	fd00:142::1	fd00:142::11:22ff:fe33:4455	ICMPv6	122	Echo (ping) request id=0x0729, seq=1
7	0.051966000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	6LoWPAN	133	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
8	0.057962000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	6LoWPAN	138	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
9	0.065963000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	6LoWPAN	138	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
10	0.072960000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	6LoWPAN	138	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
11	0.078965000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	6LoWPAN	138	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
12	0.083960000	fd00:142::11:22ff:fe33:4455	fd00:142::1	ICMPv6	122	Echo (ping) reply id=0x0729, seq=1
13	0.086965000	fd00:142::11:22ff:fe33:4455	fd00:142::1	ICMPv6	574	Echo (ping) reply id=0x0729, seq=1
14	2.997.711.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	6LoWPAN	133	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
15	3.003.710.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	6LoWPAN	138	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
16	3.011.693.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	6LoWPAN	138	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
17	3.019.719.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	6LoWPAN	138	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
18	3.028.697.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	6LoWPAN	138	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
19	3.035.705.000	fd00:142::1	fd00:142::11:22ff:fe33:4455	ICMPv6	122	Echo (ping) request id=0x0729, seq=2

Tabelle 6.2: Wiresharkmitschnitt Echo-Request (512 Bytes) - Protokollentwurf

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::12:13ff:fe14:1516	ff02::1	ICMPv6	107	Router Advertisement
2	0.000831000	fe80::12:13ff:fe14:1516	ff02::1	ICMPv6	126	Router Advertisement
3	3.930.624.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	135	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
4	3.935.608.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
5	3.942.611.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
6	3.949.619.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
7	3.957.607.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
8	3.965.623.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	123	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
9	3.978.618.000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	IEEE 802.15.4	135	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
10	3.985.628.000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	IEEE 802.15.4	139	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
11	3.993.616.000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	IEEE 802.15.4	139	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
12	3.999.616.000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	IEEE 802.15.4	139	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
13	4.006.598.000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	IEEE 802.15.4	139	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
14	4.012.600.000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	IEEE 802.15.4	123	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
15	4.015.645.000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	IEEE 802.15.4	44	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
16	4.020.622.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	44	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
17	6.931.349.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	135	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
18	6.938.329.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
19	6.947.326.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16

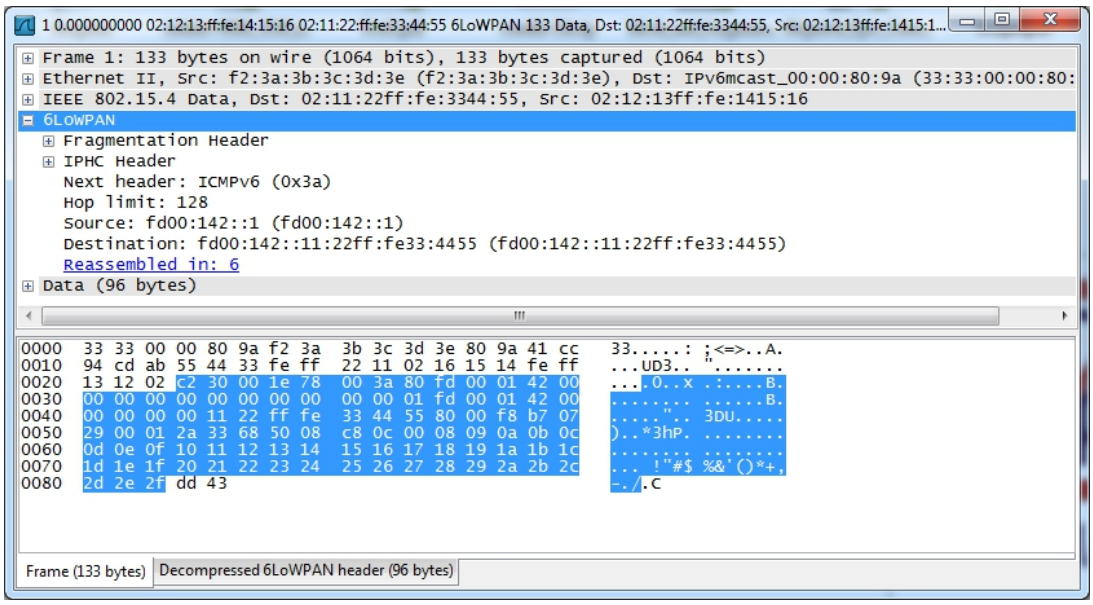


Abbildung 6.1: Wiresharkmitschnitt, 1. 6LoWPAN-Fragment - Originalimplementierung

der ICMPv6-Echo-Reply-Nachricht, mit der das AVR RZ RAVEN Board die Anfrage des AVR RZ RAVEN USB Sticks beantwortet. Die Rahmen 15 und 16 zeigen die Bestätigungsrahmen, die zwischen Board und Stick ausgetauscht werden, um die gesendeten ICMPv6-Nachrichten zu quittieren. Ab Rahmen 17 wird bereits das fragmentierte Datagramm des folgenden ICMPv6-Requests dargestellt.

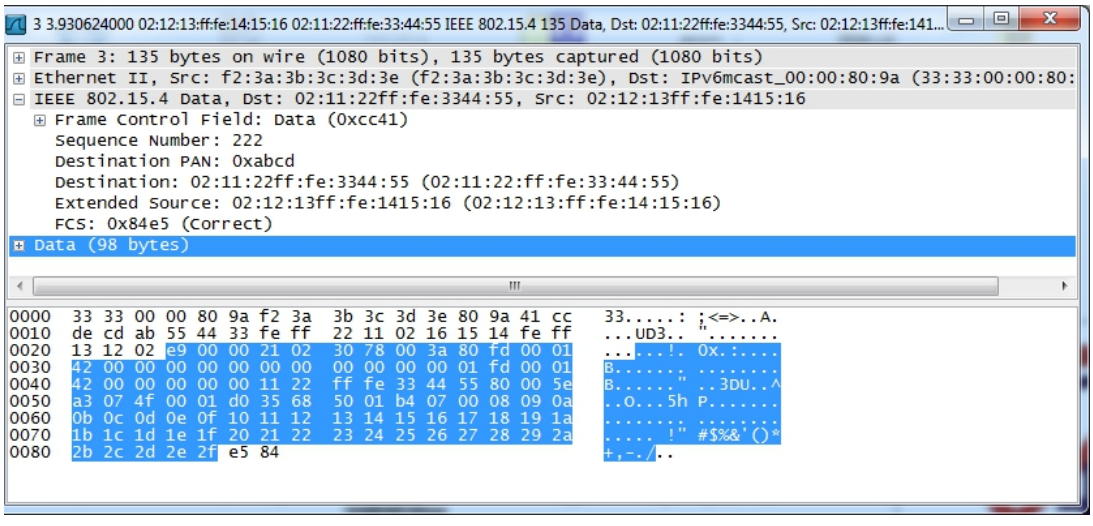


Abbildung 6.2: Wiresharkmitschnitt, 1. Fragment (RFRAG-AR) - Protokollentwurf

In diesem Beispiel hätte man das sofortige Senden der Bestätigung für die durch das AVR RZ RAVEN Board empfangene ICMPv6-Echo-Request-Nachricht erwartet. Der für das Senden zuständige Prozess reiht sich jedoch in die Prozesswarteschlange des Betriebssystems ein. Das Senden der ICMPv6-Echo-Reply-Nachricht erfolgt in dieser Implementierung zuerst. Ein Versuch,

das umgehende Senden der Bestätigungen zu realisieren führte dazu, dass ein Teil des μIP-Stacks nicht mehr korrekt arbeitete. Daher wurde dieses etwas unerwartete zeitliche Verhalten in Kauf genommen.

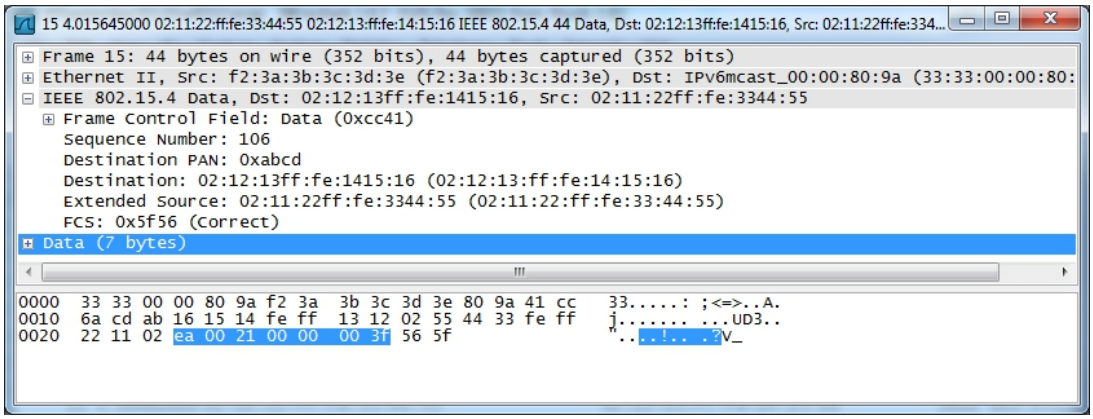


Abbildung 6.3: Wiresharkmitschnitt, Bestätigungsrahmen (RFRAG-ACK) - Protokollentwurf

In der Abbildung 6.2 ist das erste gesendete Fragment des ICMPv6-Echo-Requests nach dem Protokollentwurf im Detail zu sehen. In den Nutzdaten (markierter Bereich im Hexdump) ist der Headeraufbau des Protokollentwurfs, beschrieben in Abschnitt 4.1.3, erkennbar. Das Dispatch-Byte von 0xE9 kennzeichnet ein Recoverable Fragment, für das eine Bestätigung verlangt wird (RFRAG-AR). Ihm folgen die Bytes für das Offset mit dem Wert 0 und für das Datagramm-Tag mit dem Wert 33. In den letzten beiden Bytes des Headers werden die Sequenznummer (00000_2 für den Wert 0) und die Länge des Datagramms (01000110000_2 für den Wert 560) kodiert.

Die Abbildung 6.3 dokumentiert den genauen Aufbau des Bestätigungsrahmens (RFRAG-ACK), den das AVR RZ RAVEN Board an den AVR RZ RAVEN Stick für das empfangene Datagramm 33 sendet. Die markierten IEEE 802.15.4-Nutzdaten zeigen die Position des RFRAG-ACK-Headers. Eingeleitet wird der Header mit dem Dispatch-Byte 0xEA, gefolgt von den zwei Bytes, die für das Datagramm-Tag mit dem Wert 33 stehen. Die darauffolgenden 4 Bytes bilden die übertragene, unkomprimierte Bestätigungsbitmap. Der hexadezimale Wert 0x3F bestätigt alle der 6 übertragene Fragmente in einem einzigen Rahmen. Die letzten beiden unmarkierten Bytes bilden die sogenannte FCS und gehören nicht mehr zum RFRAG-ACK-Header.

Die Abbildung 6.4 zeigt ein auf Grundlage des Wiresharkmitschnitts in Tabelle 6.2 erstelltes Sequenzdiagramm. Anhand dieses Diagramms wird die Funktionsweise des Protokollentwurfs noch einmal verdeutlicht. Der Zeitstempel 3,931 entspricht dabei dem Rahmen 3 in der Tabelle 6.2. Die Router Advertisement-Nachrichten wurden bei der Erstellung des Diagramms ausgeblendet. Der Rahmens 3 wird vom AVR RZ RAVEN Stick zum AVR RZ RAVEN Board übertragen. Durch den verwendeten Header RFRAG-AR wird eine Bestätigung verlangt. Dieser Header-Typ wird für sämtliche Fragmente des zu übertragenen Datagramms verwendet. Mit dem Empfang des Rahmens 3 wird im AVR RZ RAVEN Board der Acknowledgement-Timer gestartet. Sollte das Datagramm nicht innerhalb der voreingestellten 187 ms komplett empfangen worden sein, wird durch den Acknowledgement-Prozess eine Bestätigung für die bisher empfangenen Fragmente an den Stick versandt.

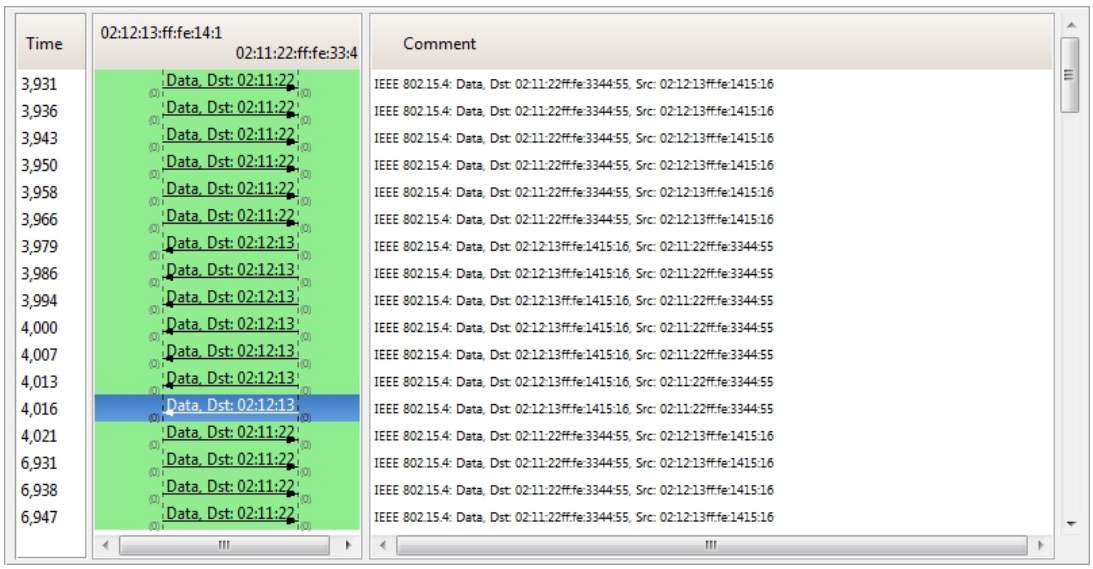


Abbildung 6.4: Wiresharkmitschnitt, Sequenzdiagramm - Protokollentwurf

Wurden sämtliche 6 Fragmente des Datagramms übertragen (Zeitstempel 3,966), wird in dem AVR RZ RAVEN Stick der Retransmission-Timer gestartet. Sollten innerhalb der voreingestellten 375 ms nicht sämtliche Fragmente bestätigt worden sein, überträgt der Retransmission-Prozess des AVR RZ RAVEN Sticks die unquittierten Fragmente erneut – bis zu fünf Mal.

In dem dargestellten Beispiel wird mit dem Empfang des 6. Fragments durch das AVR RZ RAVEN Board das Senden des Bestätigungsrahmens RFRAG-ACK initiiert. Solange der Bestätigungsrahmen nicht vom Stick empfangen oder der Retransmission-Prozess abgebrochen wurde, ist das Senden neuer Datagramme durch den Stick nicht möglich. Das asynchrone Ereignis, mit dem der Acknowledgement-Prozess über das Senden einer Bestätigung informiert werden soll, wird durch den Contiki-Prozessplaner verzögert. Es wird zuvor das Datagramm und die darin enthaltene ICMPv6-Echo-Request-Nachricht ausgewertet und mit der Übertragung der ICMPv6-Echo-Reply-Nachricht vom AVR RZ RAVEN Board an den AVR RZ RAVEN Stick begonnen. Dieses Verhalten ist unerwünscht. Es wird wieder eine Bestätigung durch die Übertragung des Headers RFRAG-AR angefordert. Bei der Übertragung der ICMPv6-Echo-Reply-Nachricht laufen in den Stationen die gleichen Vorgänge ab wie bei der Übertragung der ICMPv6-Echo-Request-Nachricht. Nur sind die Rollen von Stick und Board vertauscht.

Bei Zeitstempel 4,016 (markierter Bereich) wird, durch den Prozessplaner verzögert, der Bestätigungsrahmen für das Datagramm der ICMPv6-Echo-Request-Nachricht vom AVR RZ RAVEN Board an den AVR RZ RAVEN Stick übertagen. Mit dem Empfang der Bestätigung ist nun das Senden neuer Datagramme möglich. Im Gegenzug wird bei Zeitstempel 4,021 die Bestätigung für die ICMPv6-Echo-Replay-Nachricht vom Stick an das Board gesendet. Das Programm „Ping6“ wertet im Anschluss die benötigte Paketumlaufzeit aus und gibt sie mit weiteren Informationen in der Konsole aus. Bei Zeitstempel 6,931 ist im Sequenzdiagramm das Senden des ersten Fragments der nächsten ICMPv6-Echo-Request-Nachricht vom AVR RZ RAVEN Stick an das AVR RZ RAVEN Board zu erkennen.

6.1.3 Fragmentierter Datenverkehr - Retransmission (Protokollentwurf)

Die Tabelle 6.3 enthält einen Wirehark-Mitschnitt, der eine beispielhafte Retransmission dokumentiert. Hier wurde der ICMPv6-Netzwerkverkehr zwischen dem AVR RZ RAVEN Stick und dem AVR RZ RAVEN Board bei einer hohen Anzahl von fehlerhaften Fragmenten/Rahmen festgehalten. Erzeugt wurde die ICMPv6-Nachricht mit dem Programm „Ping6“ bei einem 1200 Byte langen Echo-Request. Die aufgezeichneten Rahmen 119 bis 124 sind von besonderem Interesse, da dort die Arbeitsweise des Protokollentwurfs sehr anschaulich nachvollzogen werden kann.

Bei dem Rahmen 119, dargestellt in Abbildung 6.5, handelt es sich einen Bestätigungsrahmen (RFRAG-ACK), der vom Board an den Stick übertragen wurde. Das 1248 Byte große IPv6-Datagramm mit dem Tag 0x02FD wurde in 13 Fragmente aufgeteilt – die Bestätigungsbitmap weist auf den Nichtempfang der Fragmente mit der Sequenznummer 0 und 12 hin.

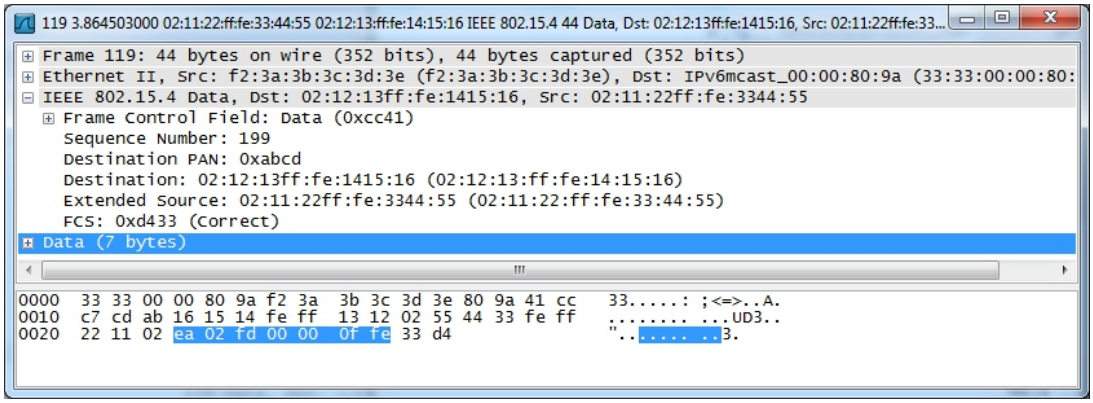


Abbildung 6.5: Übertragener 802.15.4-Rahmen Nr.119 (Protokollentwurf RFRAG-ACK)

Die Abbildungen 6.6 und 6.7 zeigen den zweifachen Retransmissionversuch des Fragments (RFRAG-AR) mit der Sequenznummer 0 für das IPv6-Datagramm mit dem Tag 0x02FD. Zwischen den beiden Sendeversuchen konnte der Empfang des Fragments durch das AVR RZ RAVEN Board nicht bestätigt werden. Die geforderte Bestätigung für das Fragment mit der Sequenznummer 0 enthält der an den AVR RZ RAVEN USB Stick gesendete Rahmen 124, dargestellt in der Abbildung 6.8. Die darin enthaltene unkomprimierte Bestätigungsbitmap bestätigt das Fragment durch das gesetzte niederwertigste Bit.

Die Rahmen 121 und 122 veranschaulichen eine bestätigte Retransmission für das Fragment mit der Sequenznummer 3 des IPv6-Datagramms mit dem Tag 0x02BF, die nicht graphisch dokumentiert wurde.

Die Abbildung 6.9 zeigt ein erstelltes Sequenzdiagramm, das auf den Daten der Tabelle 6.3 beruht. Der interessante Rahmen 119 bei Zeitstempel 3,865 wurde markiert. Die in den Abbildungen 6.5 bis 6.9 dargestellten und zuvor beschriebenen Beispiele für Retransmissions und Bestätigungen können anhand dieses Diagramms noch einmal nachvollzogen werden.

Tabelle 6.3: Wiresharkmitschnitt Echo-Request (1200 Bytes) - Retransmission (Protokollentwurf)

No.	Time	Source	Destination	Protocol	Length	Info
110	3.689.465.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
111	3.697.490.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
112	3.706.461.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
113	3.716.483.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
114	3.725.471.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
115	3.733.472.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
116	3.741.462.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
117	3.750.458.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
118	3.759.465.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
119	3.864.503.000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	IEEE 802.15.4	44	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
120	4.234.571.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	135	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
121	4.245.557.000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	IEEE 802.15.4	139	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
122	4.422.573.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	44	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
123	4.794.630.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	135	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
124	4.981.651.000	02:11:22:ff:fe:33:44:55	02:12:13:ff:fe:14:15:16	IEEE 802.15.4	44	Data, Dst: 02:12:13ff:fe:1415:16, Src: 02:11:22ff:fe:3344:55
125	5.006.644.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	135	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
126	5.014.652.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
127	5.022.617.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
128	5.030.681.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16
129	5.038.621.000	02:12:13:ff:fe:14:15:16	02:11:22:ff:fe:33:44:55	IEEE 802.15.4	139	Data, Dst: 02:11:22ff:fe:3344:55, Src: 02:12:13ff:fe:1415:16

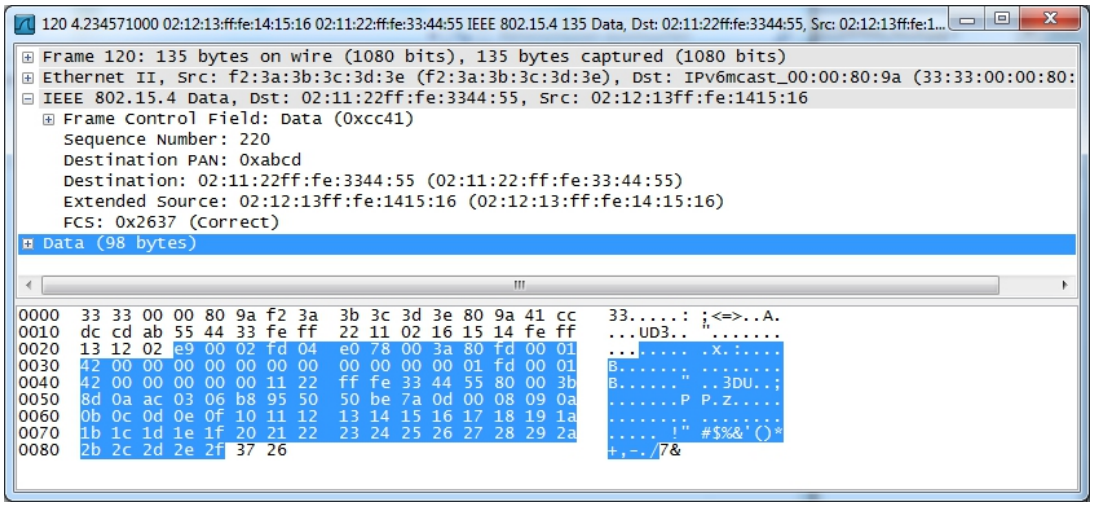


Abbildung 6.6: Übertragener 802.15.4-Rahmen Nr.120 (Protokollentwurf RFRAG-AR)

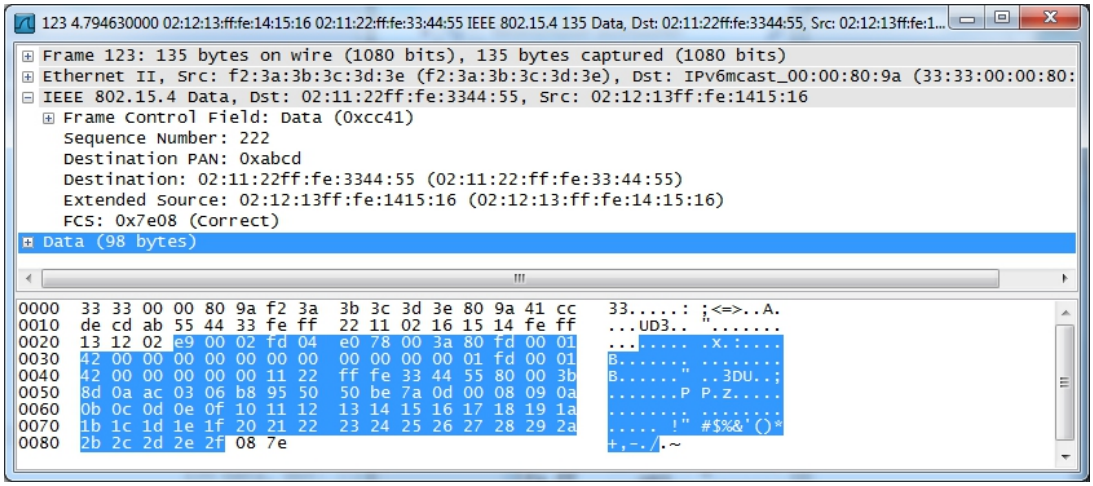


Abbildung 6.7: Übertragener 802.15.4-Rahmen Nr.123 (Protokollentwurf RFRAG-AR)

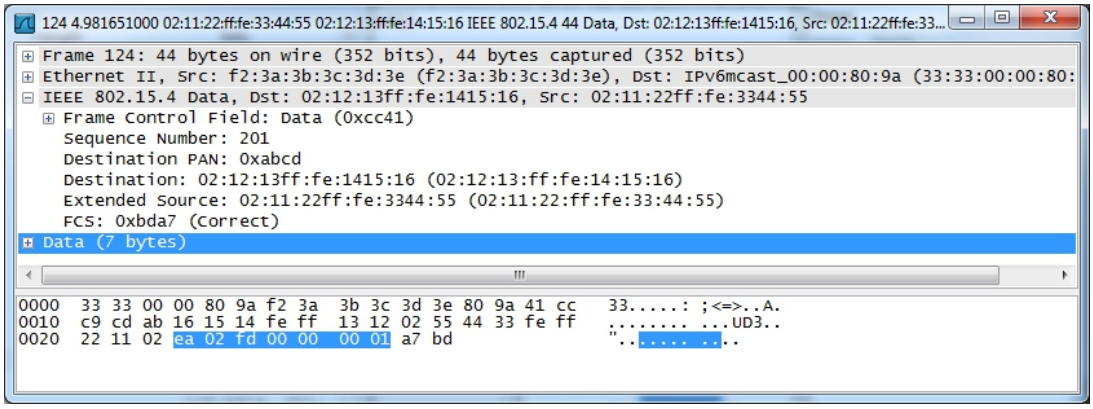


Abbildung 6.8: Übertragener 802.15.4-Rahmen Nr.124 (Protokollentwurf RFRAG-ACK)

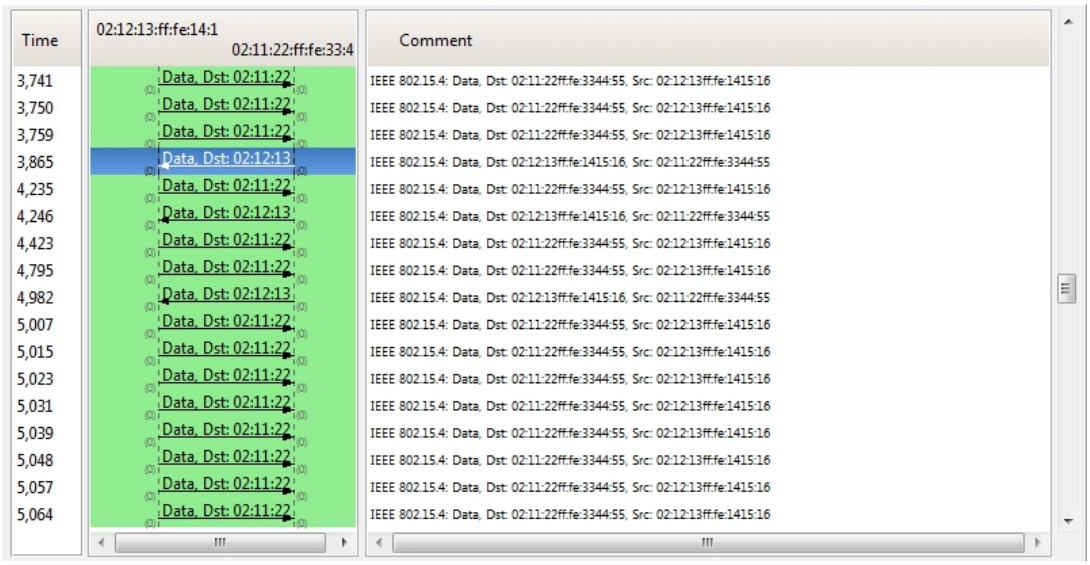


Abbildung 6.9: Wiresharkmitschnitt, Sequenzdiagramm - Retransmission (Protokollentwurf)

6.2 Messergebnisse unter Einsatz des Programms „Ping6“

6.2.1 Die Parameter Übertragungsreichweite und Paketlänge

Im Abschnitt 5.3 wurden einige Szenarien für den Vergleich des Protokollentwurfs mit der Contiki-Originalimplementierung beschrieben. Eine dieser Möglichkeiten ist die Verwendung des Protokolls ICMPv6 und seiner Anwendung in der Applikation „Ping6“. Dieses Programm wird nun für das Bestimmen der Parameter Übertragungszeit, Paketverluste und Retransmissionrate herangezogen. Dabei werden diese Parameter in Abhängigkeit von der Entfernung zwischen den Stationen und der Länge des Echo-Requests in Bytes ermittelt.

Es wurden Probemessungen mit variierendem Abstand zwischen den Stationen durchgeführt. Die Messungen in der verfügbaren Testumgebung führten dabei zu unbefriedigenden Ergebnissen. Der Einfluss der Entfernung zwischen den Stationen war schwer reproduzierbar und messbare Retransmissionraten traten nur in einem kleinen Bereich auf, kurz vor dem Verbindungsabbruch. Eine künstliche Dämpfung/Abschirmung durch metallene Gegenstände schuf keine Abhilfe. So wurde stellvertretend für die Übertragungsreichweite die in Abschnitt 5.2.5 beschriebene Fehlergenerierungsfunktionalität des modifizierten Quellcodes herangezogen.

Für eine erste Messreihe wurde der Befehl „Ping6“ auf den Betriebssystemen Windows 7 Professional x64 und auf der Linuxdistribution Debian „Squeeze“ (32 Bit) eingesetzt. Unzuverlässigkeiten in Form von nicht nachvollziehbaren Paketverlusten unter der Windowsplattform, die das Messergebnis verfälschten, führten zu der Entscheidung, sämtliche Messreihen unter der Linuxdistribution durchzuführen. Hierfür wurde eine eigens für die Vergleichsmessungen bereitgestellte Hardwareplattform Intel® Desktop Board D945GSEJT mit dem Prozessor Intel® Atom™ N270 verwendet.

Eine erste Messung mit verschiedenen Paketlängen zeigte ein problematisches Verhalten des Befehls „Ping6“ auf. Nach einer Retransmission eintreffende Datagramme haben eine deutlich höhere Paketumlaufzeit als fehlerfrei übertragene. Die Anwendung berücksichtigt diesen Umstand jedoch nicht und fährt mit dem Senden der Echo-Requests fort. Dadurch werden augenscheinlich

Pakete als verloren angezeigt, die ihr Ziel, wenn auch verzögert, erreichen würden. Die Dokumentation[1] des Befehls schlägt die Option „-W“ vor, um die Wartezeit auf ein Echo-Reply anzupassen. Das Verwenden dieser Option führte allerdings nicht zum gewünschten Erfolg.

```
ping6 -s 1200 -c 1000 -i 3 fd00:142::11:22ff:fe33:4455 > Text.txt
```

Abbildung 6.10: Ping6-Befehl mit Parametern für die durchzuführenden Messungen

Alternativ wurde eine Testmessung mit dem Parameter „–i“durchgeführt, der ein Senden der Echo-Requests im angegebenen Sekundenintervall erzwingt. Diese Messung lieferte ein repräsentatives Ergebnis. Sämtliche Messungen wurden dementsprechend mit dem in Abbildung 6.10 dargestellten und parametrisierten Ping6-Befehl durchgeführt. In diesem Beispiel wird ein 1200 Byte großes Echo-Request-Paket 1000 Mal an die angegebene IPv6-Adresse gesendet und die Ausgabe in die Datei „Text.txt“ umgeleitet. Um die Messung in einer vertretbaren Zeit durchführen zu können, wurde ein Intervall von 3 Sekunden gewählt.

6.2.2 Paketverlustrate

Die Paketverlustrate wurde für den Protokollentwurf und für die Originalimplementierung mit dem Befehl „Ping6“ der Linuxdistribution Debian „Squeeze“ (32 Bit) durchgeführt. Dabei wurden die Paketgröße und die Anzahl der generierten Fehler variiert. Jede Messung mit ihren spezifischen Parametern wurde 1000 Mal durchgeführt, um ein möglichst genaues Ergebnis zu erhalten. Die Tabellen 6.4 und 6.5 dokumentieren die Ergebnisse für beide Implementierungen. Die Zeile „ICMPv6“ enthält dabei die verwendeten Werte für den Parameter „-s“ des Pingbefehls, der die Anzahl der Bytes des übertragenen Echo-Requests angibt. Die Spalte „Rahmen“ enthält den Durchschnittswert der fehlerfrei übertragenen Fragmente/Datenrahmen, basierend auf der in Abschnitt 5.2.5 beschriebenen Fehlergenerierungsfunktion. Alternativ können die angegebenen Werte als Maß für die erzeugten fehlerhaften Fragmente interpretiert werden. Eine „15“ bedeutet hier, dass durchschnittlich 15 Fragmente fehlerfrei übertragen werden, oder dass jedes 16. Fragment fehlerbehaftet ist. Diese Parameter werden für sämtliche unter Einsatz des Befehls „Ping6“ durchgeführten Messungen verwendet.

Tabelle 6.4: Paketverlustrate in Prozent in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die **Retransmissionimplementierung**

ICMPv6 / Rahmen	128 B	256 B	384 B	512 B	768 B	1024 B	1200 B
15	01	15	26	30	53	57	71
31	00	03	03	07	18	30	42
63	00	01	03	01	03	09	09
127	00	00	00	01	01	03	03
255	00	00	00	00	00	00	01
511	00	00	00	00	00	01	00

Die Messwerte des Protokollentwurfs, aber auch die der Originalimplementierung zeigen, dass die Paketverluste mit steigender Anzahl an fehlerhaften Fragmenten ansteigen. Dieses Verhalten wird für gewöhnlich erwartet. Innerhalb einer Messreihe für eine bestimmte Anzahl generierter

[1] http://man-wiki.net/index.php/8:ping6 (Abrufdatum 27.9.2012)

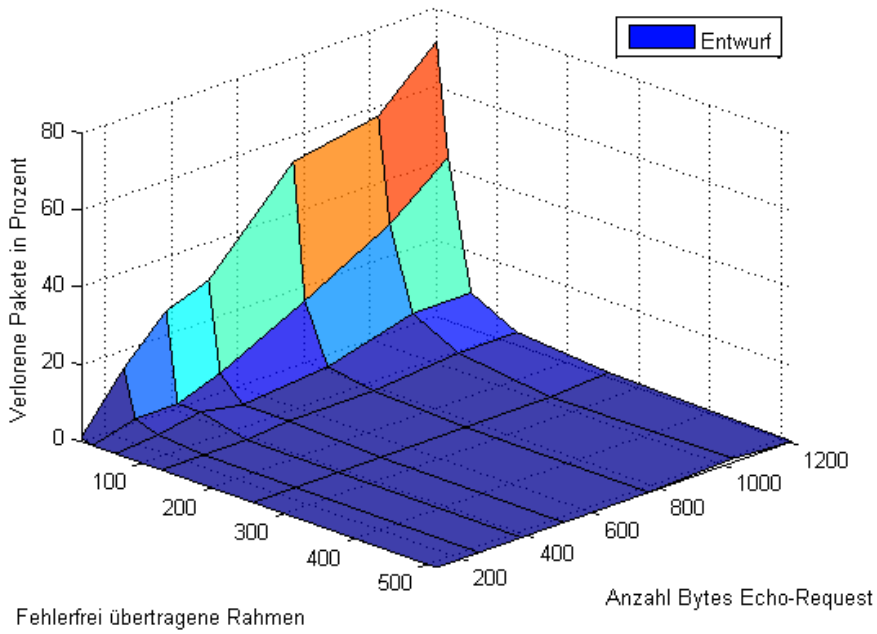


Abbildung 6.11: Paketverlustrate in Prozent in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die Retransmissionimplementierung

Fehlerfragmente wird der Einfluss der Länge eines Datagramms deutlich. Die Wahrscheinlichkeit, dass ein IP-Paket aufgrund eines fehlerhaften Fragments verworfen wird, steigt mit seiner Länge. Die Messreihe der Originalimplementierung mit einem Durchschnittswert von 15 fehlerfrei übertragenen Fragmenten zeigt bei einer hohen Anzahl Bytes (ab 512) in einem Echo-Request, dass nahezu alle IP-Pakete verloren gehen. Hierbei gilt es zu bedenken, dass auf beiden Übertragungsrichtungen fehlerhafte Fragmente erzeugt werden.

Tabelle 6.5: Paketverlustrate in Prozent in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die **Originalimplementierung**

ICMPv6 / **Rahmen**	**128 B**	**256 B**	**384 B**	**512 B**	**768 B**	**1024 B**	**1200 B**
15	29	45	69	88	93	94	94
31	11	19	20	29	47	73	71
63	06	11	15	16	25	32	36
127	06	06	07	12	12	18	17
255	01	03	03	05	06	10	08
511	00	01	01	05	03	04	04

Ein direkter Vergleich zeigt den Vorteil des implementierten Protokollentwurfs. Insbesondere bei Echo-Requests mit wenigen Fehlern konnten nahezu alle Paketverluste verhindert werden. Selbst bei hoher Anzahl von fehlerhaften Fragmenten und mittleren Datagrammlängen kann der Entwurf überzeugen. Der Fall des Echo-Requests mit einer Länge von 1200 Bytes zeigt allerdings die Grenzen seiner Leistungsfähigkeit auf. Die hohe Fehlerrate, die nun vermehrt während einer

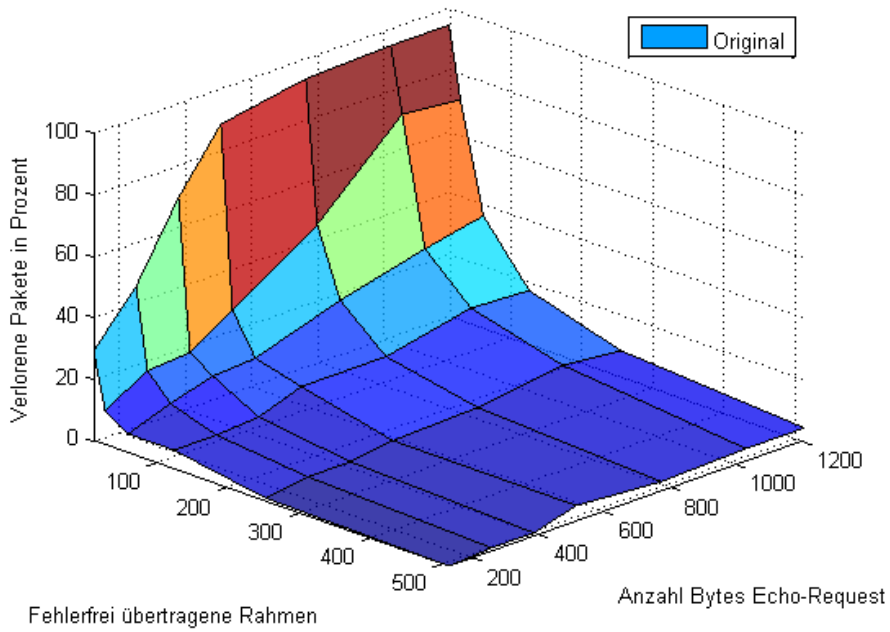


Abbildung 6.12: Paketverlustrate in Prozent in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die Originalimplementierung

Retransmission auftritt, führt zu Abbrüchen aufgrund der limitierten Anzahl von neuen Übertragungsversuchen.

Die Messwerte der beiden Tabellen 6.4 und 6.5 wurden mit der Software MATLAB graphisch in einem dreidimensionalen Diagramm aufbereitet. Die Abbildungen 6.11 und 6.12 zeigen diese Diagramme für den Entwurf und die Originalimplementierung. In der Abbildung 6.13 wurden die Graphen der beiden Implementierungen in einem Diagramm dargestellt. Dieser Versuch einer Gegenüberstellung beider Protokolle zeigt den Vorteil des Entwurfs unter dem Gesichtspunkt der Paketverluste. Die Fläche des Protokollentwurfs liegt beinahe im gesamten dargestellten Bereich unter jener der Originalimplementierung. Nur bei der geringsten Anzahl an fehlerhaften Fragmenten und sehr kurzen Datagrammen (128 Bytes Länge für das Echo-Request) ist kein Unterschied zu erkennen.

6.2.3 Retransmissionrate

Der Anteil der Dank des Protokollentwurfs erfolgreich übertragenen Pakete ist auf direktem Weg unter Verwendung des Programms „Ping6“ nicht messbar. Durch die hohe Anzahl der durchgeführten Messungen im Rahmen der Paketverlustrate kann die Differenz der Tabellen 6.4 und 6.5 als gute Näherung für die Retransmissionrate angenommen werden. Die Tabelle 6.6 dokumentiert die Retransmissionrate der IP-Pakete in Prozent, die Grafik 6.17 die dreidimensionale Darstellung dieser Werte.

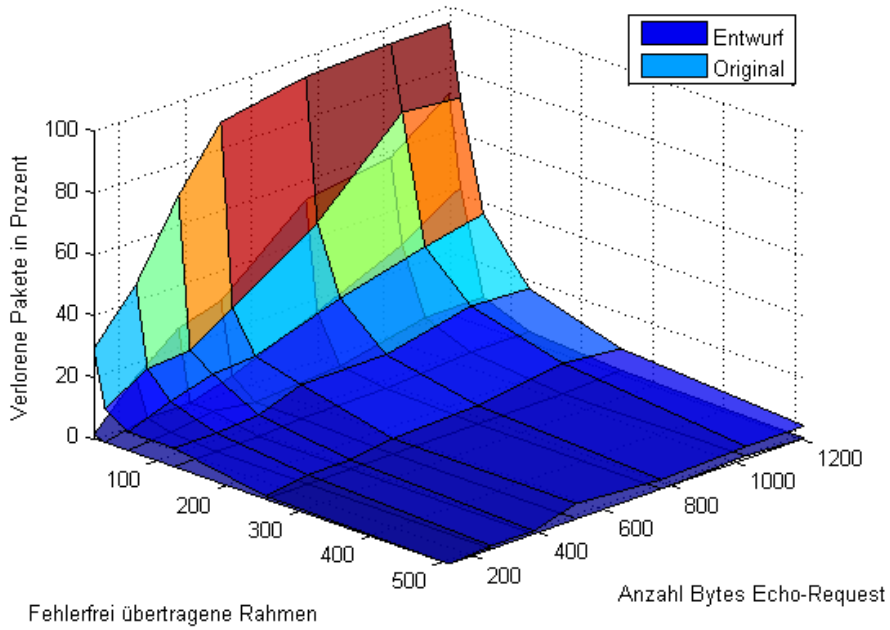


Abbildung 6.13: Paketverlustrate in Prozent in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler - Gegenüberstellung der Implementierungen

Tabelle 6.6: Erneut übertragene Pakete in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die **Retransmissionimplementierung**

ICMPv6 / **Rahmen**	**128 B**	**256 B**	**384 B**	**512 B**	**768 B**	**1024 B**	**1200 B**
15	28	30	43	58	40	37	23
31	11	16	17	22	29	43	29
63	6	10	12	15	22	23	27
127	6	6	7	11	11	15	14
255	1	3	3	5	6	10	7
511	0	1	1	5	3	3	4

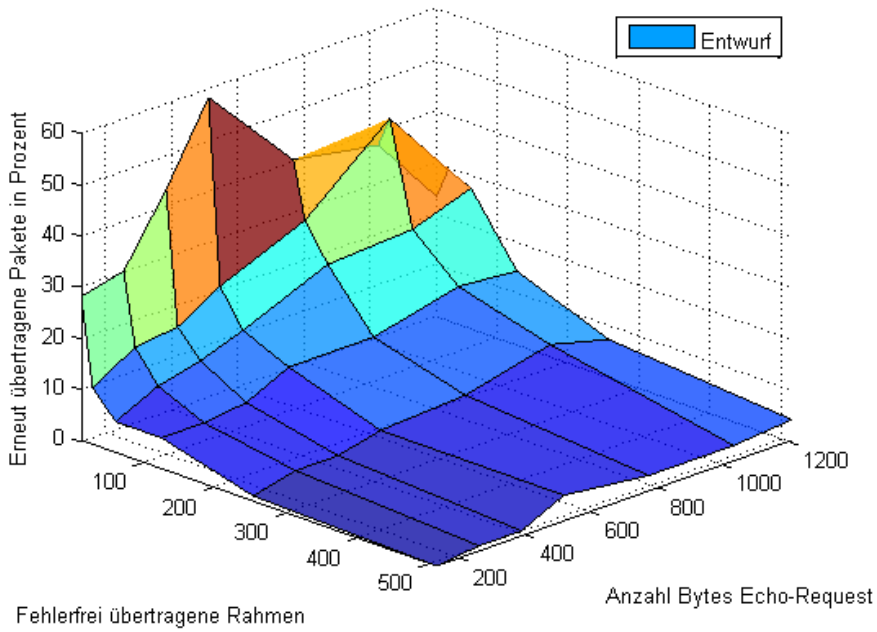


Abbildung 6.14: Erneut übertragene Pakete in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die Retransmissionimplementierung

Im Zusammenhang mit den Messungen aus Abschnitt 6.2.2 können hohe Werte in dem Diagramm als besonders erfolgreicher Einsatz des Protokollentwurfs interpretiert werden. Hier sind insbesondere die Längen von 384 bis 768 Bytes für den Echo-Request bei generierten Fehlern in jedem 16. Paket hervorzuheben. Bei Längen von 1200 Bytes ist der Protokollentwurf weniger effizient, da hier die begrenzte Anzahl von Retransmissionversuchen zum Tragen kommt. Während sämtlicher Bewertungsmessungen wurde ein Wert von 5 für die Anzahl der erneuten Sendeversuche und ein Wert von 10 für die Anzahl der gesendeten Bestätigungsrahmen verwendet.

6.2.4 Paketumlaufzeit

Die statistischen Daten, die durch das Programm „Ping6“ nach einer Messung geliefert werden, beinhalten unter anderem mehrere Angaben zu der Paketumlaufzeit der abgesetzten ICMPv6-Pakete. Das Programm berechnet die minimale sowie die maximale Paketumlaufzeit einer Messung. Darüber hinaus wird als Ergebnis der arithmetische Mittelwert sämtlicher Umlaufzeiten und die dazugehörige Moving Standard Deviation (MSTD) angegeben. Die MSTD ist eine statistische Auswertung, die eine gute Näherung für die Standardabweichung angibt. Sie lässt eine Aussage darüber zu, wie die Messwerte um den Durchschnittswert herum verteilt sind.

Für die Bewertung der Paketumlaufzeiten der Originalimplementierung und des Protokollentwurfs wurde das arithmetische Mittel, auch Durchschnitt genannt, als aussagekräftiger Messwert herangezogen. In den Tabellen 6.7 und 6.16 wurden die Paketumlaufzeiten für den Protokollentwurf und die Originalimplementierung in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler dokumentiert. Die dreidimensionalen Graphen dieser Messreihen sind in den Abbildungen 6.15 und 6.16 zu sehen.

Tabelle 6.7: Mittlere Paketumlaufzeit in ms in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die **Retransmissionimplementierung**

ICMPv6 / Rahmen	**128 B**	**256 B**	**384 B**	**512 B**	**768 B**	**1024 B**	**1200 B**
15	430	670	732	829	1170	1410	1497
31	202	321	340	472	779	916	1024
63	131	222	270	280	413	576	666
127	81	144	183	201	309	399	450
255	64	134	134	160	248	323	375
511	64	110	110	132	202	303	294

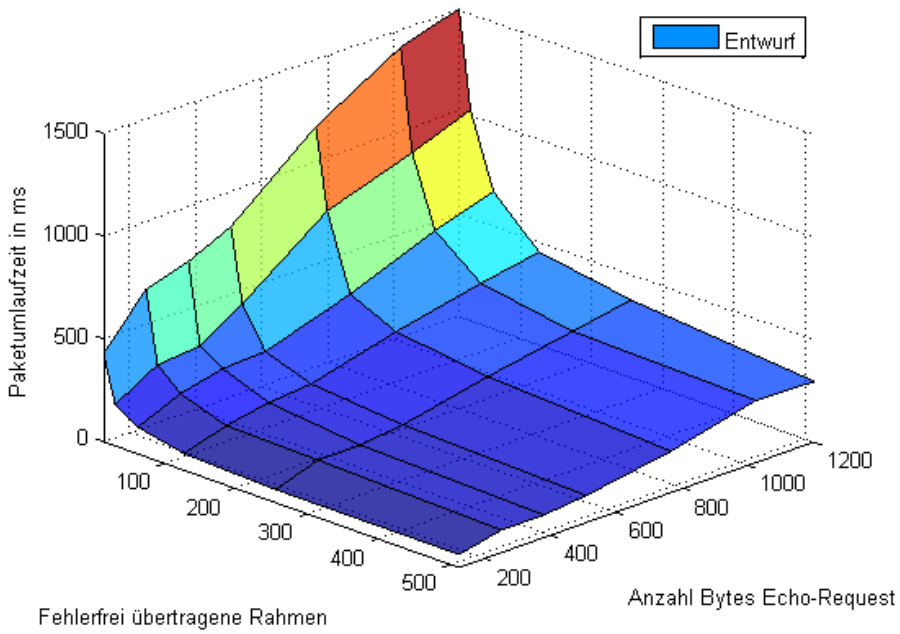


Abbildung 6.15: Mittlere Paketumlaufzeit in ms in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die Retransmissionimplementierung

Für den Protokollentwurf ist ein Ansteigen der Paketumlaufzeiten mit wachsender Länge der Datagramme zu erkennen. Dabei wächst die Paketumlaufzeit innerhalb einer Messreihe mit spezifischer Anzahl an erzeugten fehlerhaften Fragmenten relativ konstant. Ebenso steigt die Paketumlaufzeit mit einer Erhöhung der Anzahl der fehlerhaften Fragmente an – hier ist allerdings ein exponentieller Charakter zu erkennen. Die steigende Anzahl nötiger Retransmissions bei hoher Anzahl an fehlerhaften Fragmenten führt also zu einem exponentiellen Anwachsen der Paketumlaufzeit. Begrenzt wird diese nur durch die Limitierung der neuen Sendeversuche.

Tabelle 6.8: Mittlere Paketumlaufzeit in ms in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die **Originalimplementierung**

ICMPv6 / Rahmen	128 B	256 B	384 B	512 B	768 B	1024 B	1200 B
15	41	66	85	105	149	195	220
31	41	66	85	105	150	194	218
63	41	66	85	105	149	193	217
127	41	66	85	105	149	193	217
255	41	66	85	105	149	193	217
511	41	66	85	105	149	193	217

Bei dem Graphen der Originalimplementierung ist das konstante Anwachsen der Paketumlaufzeit in Abhängigkeit von der Anzahl der Bytes in einem Echo-Request noch deutlicher zu erkennen als bei dem Protokollentwurf. Der Verlauf entspricht nahezu exakt dem einer linearen Funktion. Da in der Originalimplementierung keine Retransmissions durchgefürt werden, ist mit wachsender Anzahl von fehlerhaften Fragmenten/Rahmen kein Ansteigen der Paketumlaufzeit zu beobachten. Die fehlerhaften Pakete werden einfach verworfen und gehen nicht in die Statistik der Paketumlaufzeit ein.

Eine Gegenüberstellung der Graphen der beiden Implementierungen in einem Diagramm in Abbildung 6.17 zeigt, dass der Vorteil einer reduzierten Paketverlustrate mit einer um bis zu sieben Mal höheren durchschnittlichen Paketumlaufzeit bei hoher Anzahl an fehlerhaften Fragmenten erkauft wird. Das wiederholte Übertragen dieser Fragmente kann unter Umständen zu einer Auslastung des Netzwerkes führen. Ein Ereignis, dessen Eintreten man mit dem Protokollversuch zu verhindern versucht. Sehr lange Paketumlaufzeiten führen darüber hinaus zu Retransmissionversuchen auf höheren Ebenen des OSI-Referenzmodells.

Die in dem prototypisch implementierten Entwurf verwendete Anzahl an neuen Sendeversuchen, gesendeten Bestätigungen und Timereinstellungen scheint für die im Rahmen dieser Abschlussarbeit durchgeführten Messungen praktikabel zu sein. Messungen zwischen mehrere Hops entfernten Endknoten in einem LR-WPAN mit realistischen Dimensionen würden hier weitere Erkenntnisse liefern.

6.2.5 Mehrfach übertragene Datagramme

Während der Messungen für die Bewertung des Entwurfs unter Verwendung des Befehls „Ping6“ der Linuxdistribution Debian ist das in Abbildung 6.18 aufgetretene Phänomen zu beobachten. Für das gesendete Echo-Request mit der Sequenznummer 64 wurde das Echo-Reply innerhalb der erwarteten Paketumlaufzeit und anschließend ein verzögertes Duplikat empfangen. Eine Analyse des in Abbildung 6.19 dargestellten Debuggingmitschnittes deutet die Ursache für die Übertragung des zweiten Echo-Replys an. Der String „srcACK-bmp“ bezeichnet hierbei das Bestätigungsbitmuster, das die sendende Station speichert und wird hexadezimal angegeben. Die beiden 16 Bit-

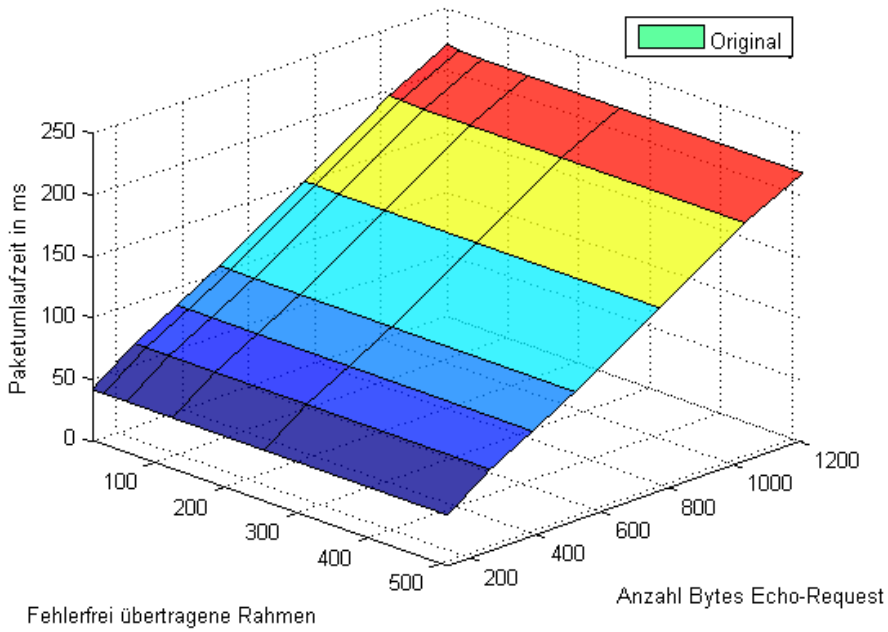


Abbildung 6.16: Mittlere Paketumlaufzeit in ms in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die Originalimplementierung

Werte enthalten den Wert null – offensichtlich wurde von der Gegenseite kein einziges Fragment empfangen.

Die Erklärung für das dargestellte Verhalten ist leicht nachvollziehbar. Das vom AVR RZ RAVEN Board gesendete Datagramm wurde vom AVR RZ USB Stick fehlerfrei empfangen und an die höhere Protokollschicht übergeben. Bei der Übertragung des Bestätigungsrahmens vom AVR RZ USB Stick zum AVR RZ RAVEN Board ist jedoch ein Fehler aufgetreten. Mit Ablauf des Timers für die Retransmissions im AVR RZ RAVEN Board wurde das Datagramm mit all seinen Fragmenten erneut an den AVR RZ USB Stick übertragen. Für das zweite Paket wurde der durch den AVR RZ USB Stick gesendete Acknowledgement-Rahmen empfangen und das Datagramm konnte so bestätigt werden. Die Applikation „Ping6“ erkannte dieses Duplikat anhand der doppelt aufgetretenen Sequenznummer und protokollierte es.

In der Tabelle 6.9 sind sämtliche bei den mit dem Programm „Ping6“ durchgeführten Messungen aufgetretenen Duplikate erfasst worden. Die Messungen wurden wie gewohnt mit steigender Anzahl der fehlerhaften Fragmente und Erhöhung der Länge des Echo-Requests durchgeführt. Diese gesammelten Messdaten wurden mit der Mathematiksoftware MATLAB aufbereitet und als dreidimensionaler Graph in der Abbildung 6.20 dargestellt.

Die Interpretation dieser Messwerte gestaltet sich in der Art schwierig, dass speziell bei längeren Datagrammen und sinkender Anzahl von fehlerhaften Fragmenten keine eindeutigen Tendenzen erkennbar sind. Allgemein kann jedoch angenommen werden, dass die Zahl der Duplikate mit der Anzahl der Fehler in den Fragmenten ansteigt. Insbesondere bei Echo-Requests mit wenigen Bytes Länge steigt das Auftreten von mehrfach gesendeten Datagrammen sprunghaft an. Die Ursache

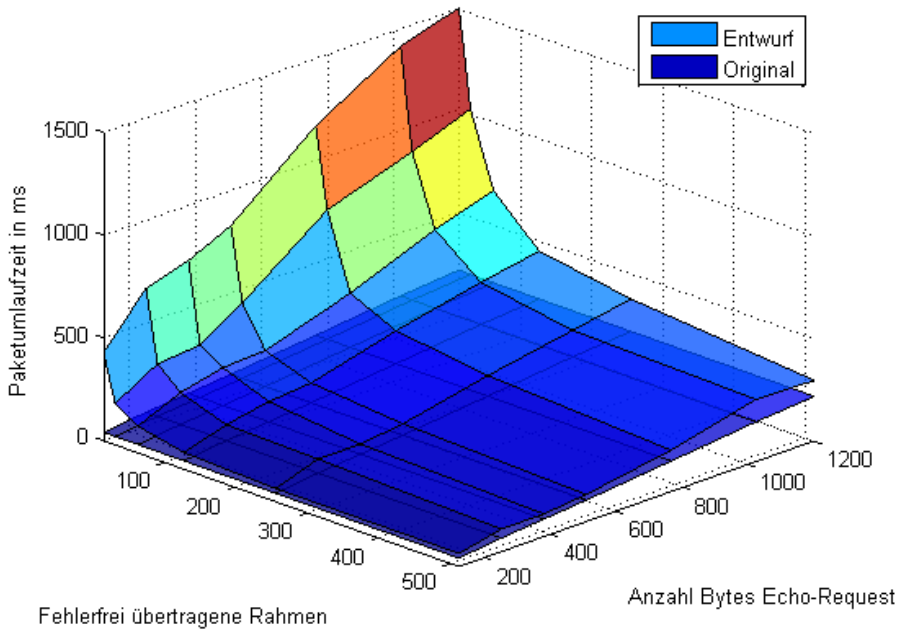


Abbildung 6.17: Mittlere Paketumlaufzeit in ms in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler - Gegenüberstellung der Implementierungen

```
1208 bytes from fd00:142::11:22ff:fe33:4455: icmp_seq=64 ttl=128 time=211 ms
1208 bytes from fd00:142::11:22ff:fe33:4455: icmp_seq=64 ttl=128 time=831 ms (DUP!)
1208 bytes from fd00:142::11:22ff:fe33:4455: icmp_seq=65 ttl=128 time=219 ms
1208 bytes from fd00:142::11:22ff:fe33:4455: icmp_seq=66 ttl=128 time=219 ms
1208 bytes from fd00:142::11:22ff:fe33:4455: icmp_seq=67 ttl=128 time=701 ms
1208 bytes from fd00:142::11:22ff:fe33:4455: icmp_seq=68 ttl=128 time=679 ms
```

Abbildung 6.18: Doppelt empfangenes Datagramm für Echo-Request (Seq 64)

liegt hier in der wachsenden Wahrscheinlichkeit, dass ein Bestätigungsrahmen von einem Fehler betroffen ist und ein Datagramm unnötigerweise erneut gesendet wird.

Tabelle 6.9: Übertragene Duplikate in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die **Retransmissionimplementierung**

ICMPv6 / Rahmen	128 B	256 B	384 B	512 B	768 B	1024 B	1200 B
15	166	73	35	25	29	28	29
31	15	39	15	26	31	16	20
63	08	10	10	08	16	19	16
127	09	04	03	03	04	02	02
255	02	02	08	05	08	04	02
511	01	21	28	01	00	01	04

```
retransmit fragment 1 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 2 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 3 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 4 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 5 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 6 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 7 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 8 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 9 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 10 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 11 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 12 of 13, datagram 63, srcACK-bmp 0:0
retransmit fragment 13 of 13, datagram 63, srcACK-bmp 0:0
received acknowledgement 0:3ff datagram 63
retransmit fragment 11 of 13, datagram 63, srcACK-bmp 0:3ff
retransmit fragment 12 of 13, datagram 63, srcACK-bmp 0:3ff
retransmit fragment 13 of 13, datagram 63, srcACK-bmp 0:3ff
received acknowledgement 0:1c00 datagram 63
received acknowledgement 0:1fff datagram 64
received acknowledgement 0:1fff datagram 65
received acknowledgement 0:1fff datagram 66
```

Abbildung 6.19: Debugging der Retransmission für Echo-Request (Seq 64)

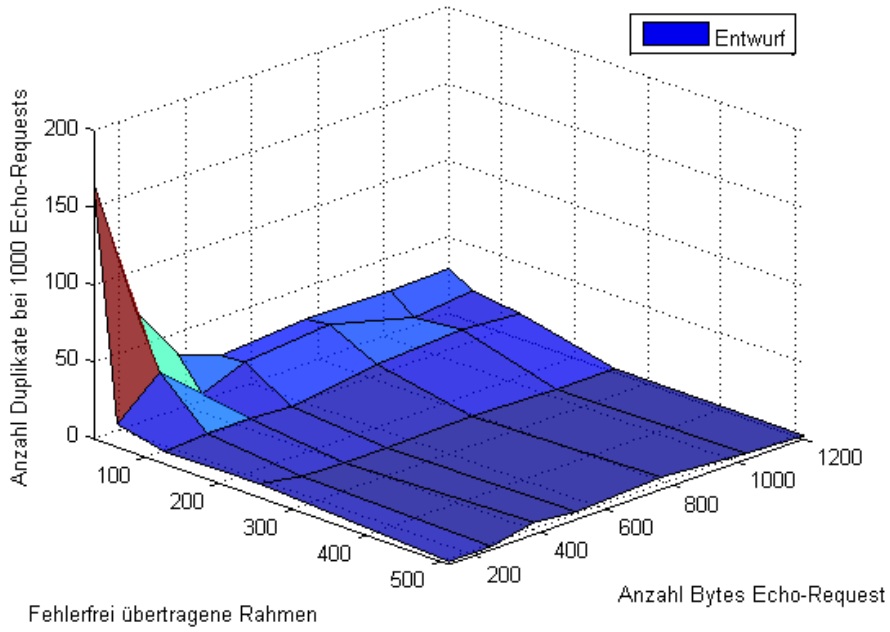


Abbildung 6.20: Übertragene Duplikate in Abhängigkeit von der Länge des Echo-Requests und der Anzahl der generierten Fehler für die Retransmissionimplementierung

6.3 Messergebnisse unter Einsatz des Programms „JMeter“

Das Programm „JMeter“ wurde für die Bestimmung der Parameter Datenrate in kB/s und Durchsatz in Seitenaufrufen/min verwendet. Darüber hinaus wurden die Parameter „durchschnittliche Latenz“ und „Latenz bei Marke 90 Prozent“ erfasst, da sie eine gute Vergleichsmöglichkeit zu den Paketumlaufzeitmessungen mit dem Programm „Ping6“ bieten. Die Marke 90 Prozent bedeutet für die Messung, dass 90 Prozent aller ermittelten Latenzwerte unter dieser Marke liegen. Die Messungen mit dem Programm „JMeter“ wurden auf der gleichen Hardwareplattform durchgeführt wie jene mit dem Programm „Ping6“. Dabei wurde die von der Linuxdistribution Debian unterstützte Version 2.3.4 der Applikation eingesetzt.

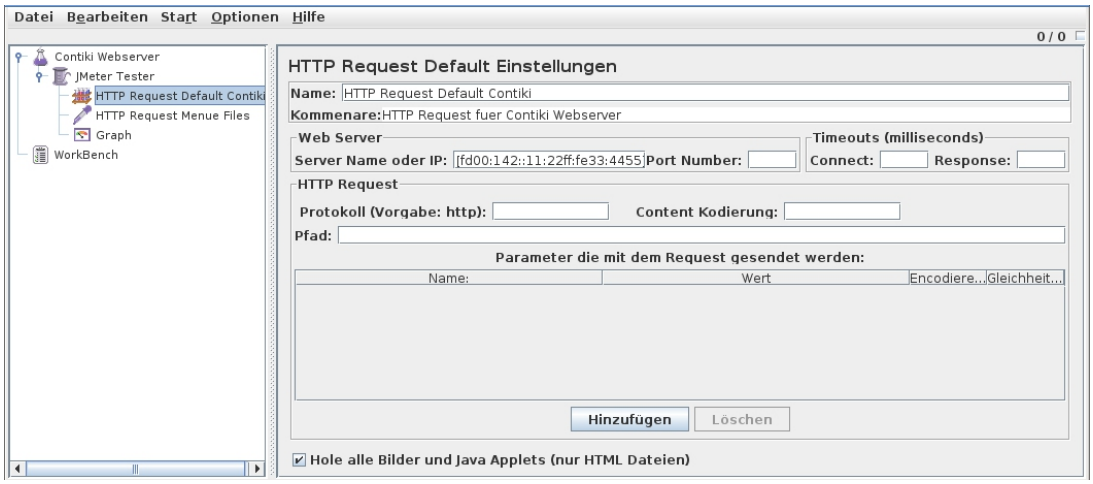


Abbildung 6.21: Das für die Messung konfigurierte Programm „JMeter“

```
fd00:142::1: /files.shtml
number of fragments: 2
number of fragments: 2
number of fragments: 10
number of fragments: 2
number of fragments: 9
fd00:142::1: /style.css
number of fragments: 2
number of fragments: 2
number of fragments: 7
number of fragments: 14
number of fragments: 10
```

Abbildung 6.22: Fragmentierung der Datagramme der aufgerufenen Webseite

Innerhalb der Messungen wurde über einen HTTP-Get-Request die Seite „/files.shtml“ des Contiki-IPv6-Webservers aufgerufen. Dieses Beispielprogramm wurde für das AVR RZ RAVEN Board kompiliert und in dessen Festspeicher programmiert. Für die durchzuführenden Messungen wurde die Anzahl der fehlerhaften Fragmente wie bei der Bewertung mit dem Programm „Ping6“

variiert. Die Abbildung 6.22 verdeutlicht, wie stark die einzelnen Datagramme während der Übertragung des Inhaltes der aufgerufenen Webseite fragmentiert wurden. Bei dem Aufruf werden sowohl kurze als auch lange Datagramme über eine TCP-Verbindung übertragen. Für die durchgeführten Messungen ist daher ein repräsentatives Ergebnis zu erwarten. Die Debuggingausgabe der Abbildung 6.22 wurde, ebenso wie die der Abbildung 6.19, speziell für diese Messung erzeugt und danach aus dem Quellcode entfernt.

„JMeter“ wurde so konfiguriert, dass jeder Seitenaufruf mit einer bestimmten Anzahl an fehlerhaften Fragmenten 100 Mal durchgeführt wird – für beide Implementierungen. Die Konfigurationsdatei für das Programm „JMeter“ wurde dem Anhang (CD) dieser Abschlussarbeit hinzugefügt. Die Ergebnisse dieser Messungen wurden in den Tabellen 6.10 und 6.11 dokumentiert. Die Fehler in den Datagrammen wurden erneut unter Einsatz der in Abschnitt 5.2.5 beschriebenen Fehlergenerierungsfunktion erzeugt. Die Angabe im Tabellenkopf gibt also den Durchschnittswert der fehlerfrei übertragenen Fragmente/Rahmen an. Vorausgesetzt wird dabei, dass eine hohe Anzahl von Fragmenten übertragen wird. Der Inhalt der Tabellen wurde unter Verwendung der Software MATLAB graphisch aufbereitet und in zweidimensionalen Diagrammen (Abbildungen 6.23 und 6.24) dargestellt. Dabei werden die Implementierungen für jeden Messparameter gegenübergestellt.

Die Abbildung 6.23 zeigt den Durchsatz in Webseitenaufrufen/min und die Datenrate in kB/s für den Protokollentwurf und die Originalimplementierung. Der prototypisch implementierte Protokollentwurf weist dabei Performanceeinbußen bei beiden Parametern auf, wenn die Anzahl von durchschnittlich 127 fehlerfrei übertragenen Fragmenten nicht unterschritten wird. Das Übertragen der Bestätigungsrahmen und verlorengegangener Fragmente stellt daher einen nicht zu vernachlässigenden Faktor dar.

Tabelle 6.10: **Retransmissionimplementierung**

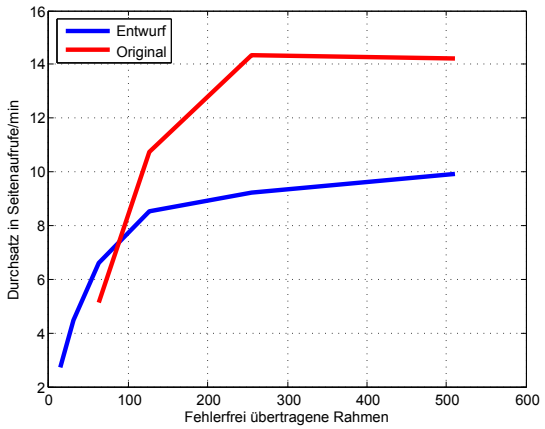
Rahmen / **Messreihe**	**15**	**31**	**63**	**127**	**255**	**511**
Durchsatz in Aufrufe/min	2,721	4,480	6,601	8,515	9,227	9,920
Datenrate in kB/s	0,167	0,275	0,406	0,524	0,567	0,610
Latenz ⌀ in ms	22048	13391	9087	7043	6499	6045
Latenz (90%) in ms	33311	20526	15805	10999	9792	9480

Tabelle 6.11: **Originalimplementierung**

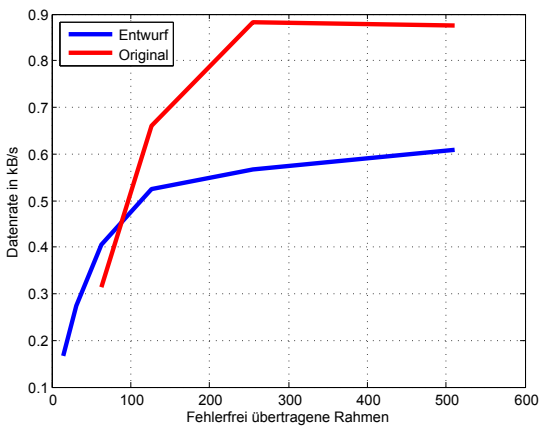
Rahmen / **Messreihe**	**15**	**31**	**63**	**127**	**255**	**511**
Durchsatz in Aufrufe/min	—	—	5,125	10,744	14,358	14,237
Datenrate in kB/s	—	—	0,315	0,661	0,883	0,875
Latenz ⌀ in ms	—	—	11659	5581	4175	4211
Latenz (90%) in ms	—	—	18496	11496	8308	8121

Der Bereich zwischen durchschnittlich 63 und 127 fehlerfrei übertragenen Fragmenten zeigt sehr anschaulich die Grenze der Leistungsfähigkeit der Originalimplementierung. Der Bereich wurde aufgrund der großen Abstände des Parameters „fehlerfrei übertragene Rahmen“ in dem Diagramm linear interpoliert dargestellt. Bei hoher Anzahl von fehlerhaften Fragmenten bietet der Protokollentwurf einen Vorteil gegenüber der Originalimplementierung. Unterhalb der Grenze von durchschnittlich 63 fehlerfrei übertragenen Fragmenten war bei der Originalimplementierung keine Messung mehr durchführbar. Die Messungen wurden daher ohne ein Ergebnis abgebrochen.

Die Abbildung 6.24 zeigt die Diagramme der gemessenen Parameter „durchschnittliche Latenz“ und die „Latenz bei der Marke 90 Prozent“. Für durchschnittlich 127 oder mehr fehlerfrei übertragene Fragmente/Rahmen ist für den Protokollentwurf eine höhere Verzögerung beim Aufbau der Webseite zu beobachten. Wird die Anzahl der fehlerhaften Fragmente weiter erhöht, verschlechtern sich beide Parameter – der Protokollentwurf weist nun ein günstigeres Verhalten bezüglich der Latenz auf. Wie bereits bei der Interpretation der Abbildung 6.23 erwähnt, war die Messung für eine hohe Anzahl von fehlerhaften Fragmenten nicht mehr möglich und wurde abgebrochen. Der Protokollentwurf bietet hier den Vorteil, dass ein Zugriff auf die Webseite, wenn auch mit hohen Verzögerungszeiten, möglich ist. Der Anstieg der Verzögerung zeigt hier allerdings, wie auch bei der Messung mit dem Programm „Ping6“, dokumentiert in der Abbildung 6.15, ein exponentielles Verhalten. Ein Webseitenaufbau von durchschnittlich 20 Sekunden stellt die Grenze des Zumutbaren dar. Interessant wäre erneut eine Vergleichsmessung in einem LR-WPAN mit realistischen Dimensionen und mehrere Hops entfernten Stationen.

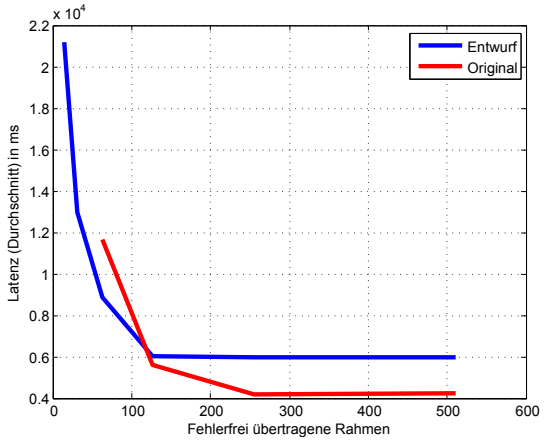


(a) Durchsatz in Seitenaufrufe/min

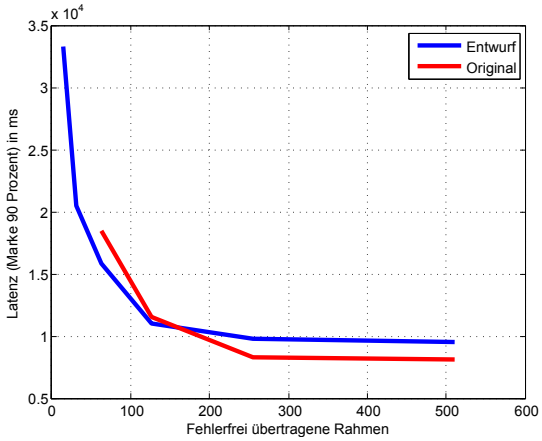


(b) Datenrate in kB/s

Abbildung 6.23: Durchsatz und Datenrate des Contiki-Webservers



(a) Latenz in ms



(b) Latenz in ms (Marke 90%)

Abbildung 6.24: Latenz des Contiki-Webservers in ms

6.4 Zusammenfassung der Ergebnisse und Interpretation

Die Messungen unter Einsatz der Programme „Ping6“ und „JMeter“ haben gezeigt, dass ein Großteil der Datagramme, die bei der Originalimplementierung verloren gehen würden, mit der Retransmissionimplementierung übertragen werden konnten. Die Ergebnisse der Messreihen zu den Parametern Paketumlaufzeit, Durchsatz, Datenrate und Latenz zeigten, dass mit teilweise erheblichen Verzögerungen bei der Übertragung zu rechnen ist. Besonders deutlich wurde dies bei Analyse der Kommunikation mit hohem Anteil fehlerhafter Fragmente/Rahmen. Hier konnte ein exponentieller zeitlicher Aufwand für die Übertragung, der nur durch die Anzahl der Sendeversuche begrenzt wurde, nachgewiesen werden.

Die zusätzlichen Verzögerungen sind allgemein auf die Übertragung des Bestätigungsrahmens zurückzuführen. Das wird besonders während einer Übertragung bei leicht gestörtem Übertragungskanal deutlich. Das Einreihen des Prozesses in die Prozesswarteschlange und die Zeit, die zwischen dem auslösenden Ereignis und dem tatsächlichen Aufruf des Prozesses vergeht, wirken sich nachteilig auf das Übertragungsverhalten aus. Die im Abschnitt 6.1 erläuterte ungewollte zeitliche Abfolge der übertragenen Rahmen ist auf diesen Ablauf zurückzuführen. Die Zeit für das Abarbeiten der Prozesse, die zuvor in die Prozesswarteschleife eingereiht wurden, muss daher bei einem ereignisbasierten Multithreadmechanismus berücksichtigt werden.

Verschlechtert sich die Übertragungsqualität auf dem Übertragungskanal, wächst die Übertragungszeit mit dem beschriebenen exponentiellen Charakter an. Die erneute Übertragung der nicht bestätigten Fragmente durch den Retransmission-Prozess und das Warten auf die Bestätigung des Kommunikationspartners sind durch den Contiki-Prozessplaner, der die Prozesse entsprechend der eintreffenden Ereignisse abarbeitet, schwer zu steuern.

Um dieses Verhalten positiv beeinflussen zu können, wären umfangreiche Messreihen mit alternativen Timereinstellungen bzw. Werten für die Anzahl der übertragenen Bestätigungsrahmen und Retransmissions denkbar. Die verwendeten Werte wurden innerhalb dieser Abschlussarbeit experimentell bestimmt - in einem begrenzten Zeitrahmen. Denkbar wäre auch das Übertragen mehrerer Bestätigungsrahmen hintereinander. Dieses Verhalten könnte von der Größe des Datagramms und von einer Schätzung der Qualität des Übertragungskanals abhängig gemacht werden.

Während der Implementierungsphase wurden ansatzweise Test durchgeführt, um das unerwünschte zeitliche Verhalten bei der Übertragung der Fragmente zu beeinflussen. Dabei wurde versucht, den Aufruf des Prozesses für die Bestätigungsrahmen durch das Absetzen eines synchronen Ereignisses zu beschleunigen – dabei wird die Prozesswarteschleife umgangen. Der Lösungsansatz führte zu diesem Zeitpunkt zu keinem Erfolg. Ein Aufgreifen dieser Idee und tiefergehende Analysen des Contiki-Quellcodes könnten jedoch eine Lösung für dieses Problem aufzeigen.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Fazit

Im Rahmen dieser Abschlussarbeit wurde der Protokollentwurf „LoWPAN fragment Forwarding and Recovery“ von Pascal Thubert und Jonathan W. Hui prototypisch implementiert. Dabei wurde der Fokus auf die Realisierung eigener Dispatch-Bytes und Header gelegt, die das Anfordern spezifischer, während der Übertragung verlorengegangener Datagramm-Fragmente ermöglichen. Der erfolgreich erstellte Protokollentwurf wurde auf dem Atmel AVR RZ RAVEN 2.4 GHz Wireless Evaluation Kit eingesetzt und durch die Analyse von umfangreichen Messreihen zu den Parametern Paketverlustrate, Paketumlaufzeit, Retransmissionrate und Datendurchsatz bewertet. Diese Messdaten wurden bei variierender Anzahl von fehlerhaften Rahmen und unterschiedlicher Länge der Datagramme erhoben.

Die Ergebnisse dieser Bewertung machen das Potential des Protokollentwurfs bezüglich der Betriebssicherheit und Zuverlässigkeit in LR-WPANs bei Einsatz des Protokolls IPv6 deutlich. Gleichzeitig wurden die Grenzen der Leistungsfähigkeit des Protokollentwurfs nachgewiesen. Das Bestätigen der übertragenen Fragmente und mögliche Retransmissions führen zu Performanceverlust und größeren Paketumlaufzeiten, sofern die drahtlose Übertragung leichten oder keinen Störeinflüssen ausgesetzt wird.

Der Netzwerkverkehr der 6LoWPAN-Implementierung des Betriebssystems Contiki 2.5 und der implementierte Protokollentwurf wurden mit dem Analysewerkzeug Wireshark aufgezeichnet und gegenübergestellt. Dadurch wurde die Funktionsfähigkeit des Protokollentwurfs am Beispiel einer Retransmission demonstriert, aber auch der eingeschränkte Einfluss auf den zeitlichen Ablauf der gesendeten Bestätigungsrahmen aufgrund der ereignisgesteuerten Prozessverarbeitung durch das Betriebssystem aufgezeigt.

7.2 Weiterführende Arbeiten

Innerhalb des Bearbeitungszeitraumes dieser Abschlussarbeit wurden Fragen aufgeworfen und Ideen entwickelt, die eine Antwort bzw. einen Lösungsansatz offen ließen. Daraus lassen sich Themen ableiten, die zu einer auf den Protokollentwurf aufbauenden Arbeit führen könnten. Zu diesem Zweck wurden in diesem Abschnitt einige Gedankengänge zusammengefasst.

Das Betriebssystem Contiki in der Version 2.5 unterstützt keinen Mesh-Header und somit ist das in Abschnitt 2.2.3 beschriebene Routingverfahren „Mesh-Under“ für den Protokollentwurf nicht anwendbar gewesen. Der Einsatz einer Routingstrategie auf Ebene der Vermittlungsschicht würde zwar den Pakettransport über mehrere Stationen hinweg ermöglichen, der Weg durch das WPAN

wäre aber durch das Routingprotokoll vorgegeben. Der Fall, dass die Datenrahmen ihre Zielstation in einem Netzwerk auf mehreren Wegen erreichen können, ist hier nicht gegeben. Trotzdem wäre die Bewertung des Protokollentwurfs in einem Netzwerk mit realistischen Dimensionen von Interesse, da so weitere Erkenntnisse über die Praxistauglichkeit gewonnen werden könnten. Im Rahmen des Protokollentwurfs (Abschnitt 4.1.5) wird ein alternativer „Route-Over“-Mechanismus auf Ebene der Vermittlungsschicht vorgeschlagen, der den Einsatz der neuen Dispatch-Bytes und Header ermöglicht. Die zugrunde liegende Idee liegt dort im Führen einer Liste, welche unter anderem die Datagramm-Tags für das korrekte Zuordnen der Bestätigungsrahmen zwischenspeichert. Daher könnte es Thema einer Abschlussarbeit sein, diesen Routingentwurf zu implementieren und ihn anhand vergleichbarer Messungen wie im Rahmen dieser Arbeit zu bewerten. Die Kommunikationspartner würden in diesem Fall mehrere Stationen auseinander liegen.

Die Verfügbarkeit des Atmel AVR RZ RAVEN 2.4 GHz Wireless Evaluation Kits war bereits während der Bearbeitungszeit dieser Arbeit eingeschränkt. Kurz vor dem Abgabetermin war diese Hardwareplattform nur noch über den Hersteller Atmel zu beziehen. Die Portierung auf eine alternative Plattform wäre eine naheliegende Konsequenz – auch mit dem Hintergrund, den Quellcode auf Wiederverwertbarkeit zu überprüfen.

Das verwendete Bestätigungsbitmuster wurde im Rahmen dieser Abschlussarbeit in ihrer unkomprimierten Form implementiert. Als Begründung wurden der erhöhte Implementierungsaufwand und die durch die Einführung eines separaten Retransmission-Buffers bereits erschöpften Ressourcen an Arbeitsspeicher genannt. Als weiterführende Arbeit wäre die Umsetzung des Vorschlages in Abschnitt 4.1.3 für ein komprimiertes Bestätigungsbitmuster denkbar.

Wie aus den Messungen in Kapitel 6 hervorging, erreicht der Protokollentwurf bei hoher Rate fehlerhaft übertragener Fragmente die Grenze der Leistungsfähigkeit. Ein möglicher Lösungsansatz, selbst bei einer derart eingeschränkten Verbindungsqualität akzeptable Resultate zu erzielen, wäre die Umsetzung eines adaptiven Protokollverhaltens. Die zu implementierende Lösung würde die Qualität des Übertragungskanals überwachen und ab einem definierten Schwellwert den Bestätigungsrahmen (RFRAG-ACK) mehr als einmal übertragen. Eine Gegenüberstellung der Implementierung mit einfacher Bestätigung der übertragenen Fragmente und des experimentellen Entwurfs mit bspw. zwei gesendeten Bestätigungen könnte mögliche Vorteile eines adaptiven Protokolls aufzeigen.

Abkürzungsverzeichnis

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
AP	Access Point
API	Application Programming Interface
APT	Advanced Packaging Tool
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
BPSK	Binary Phase Shift Keying
BSD	Berkeley Software Distribution
CAP	Contention Access Period
CCA	Clear Channel Assessment
CCITT	Comité Consultatif International Téléphonique et Télégraphique
CEA	Consumer Electronics Association
CFP	Contention-Free Period
CID	Context Identifier Extension
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSS	Chirp Spread Spectrum
DAC	Destination Address Compression
DAE	Destination Address Encoding
DAM	Destination Address Mode
DCI	Destination Context Identifier
DLL	Data Link Layer
DNS	Domain Name System
DSCP	Differentiated Services Code Point
DSSS	Direct Sequence Spread Spectrum
ECN	Explicit Congestion Notification
EDNS0	Extension Mechanisms for DNS
EEPROM	Electrically Erasable Programmable Read-Only Memory
EID	Extension Header ID
ETSI	European Telecommunications Standards Institute
EUI-64	Extended Unified Identifier
FCC	Federal Communications Commission
FCS	Frame Check Sequence
FFD	Full Function Device
FIB	Forwarding Information Base

GFSK . . . . . Gaussian Frequency Shift Keying
GTS . . . . . Guaranteed Time Slot

HAN . . . . . Home Area Network
HVAC . . . . . Heating, Ventilation and Air Conditioning

IANA . . . . . Internet Assigned Numbers Authority
IC . . . . . Integrated Circuit
ICMP . . . . . Internet Control Message Protocol
IDE . . . . . Integrated Development Environment
IEEE . . . . . Institute of Electrical and Electronics Engineers
IETF . . . . . Internet Engineering Task Force
IID . . . . . Interface Identifier
IP . . . . . Internet Protocol
ISDN . . . . . Integrated Services Digital Network
ISM . . . . . Industrial, Scientific and Medical
ISO . . . . . International Standards Organization
ISP . . . . . In-circuit Serial Programming

JTAG . . . . . Joint Test Action Group

LCD . . . . . Liquid Crystal Display
LED . . . . . Light-Emitting Diode
LLC . . . . . Logical Link Control
LoWPAN . . . . . Low-Power Wireless Personal Area Networks
LR-WPAN . . . . . Low-Rate Wireless Personal Area Network
lwIP . . . . . lightweight IP

MAC . . . . . Media Access Control
MCPS-SAP . . . . . MAC Common Part Sublayer Service Access Point
MIC . . . . . Message Integrity Check
MLME . . . . . MAC Sublayer Management Entity
MLME-SAP . . . . . MAC Sublayer Management Entity Service Access Point
MPDU . . . . . MAC Protocol Data Unit
MPSK . . . . . M-ary Phase Shift Keying
MSB . . . . . Most Significant Bit
MSDU . . . . . MAC Service Data Unit
MSS . . . . . Maximum Segment Size
MSTD . . . . . Moving Standard Deviation
MTU . . . . . Maximum Transmission Unit

NAS . . . . . Network Attached Storage
ND . . . . . Neighbor Discovery
NHC . . . . . Next Header Coding

O-QPSK . . . . . Offset Quadrature Phase Shift Keying
OSI . . . . . Open Systems Interconnection

PAN . . . . . Personal Area Network
PD-SAP . . . . . Physical Layer Data Service Access Point
PDA . . . . . Personal Digital Assistant
PDU . . . . . Protocol Data Unit
PHY . . . . . Physical Layer

PIB	PAN Information Base
PLME	Physical Layer Management Entity
PLME-SAP	Physical Layer Management Entity Service Access Point
PMTUD	Path MTU Detection
PPDU	Physical Layer Protocol Data Unit
PPP	Point-to-Point Protocol
PSDU	Physical Layer Service Data Unit
PSSS	Parallel Sequence Spread Spectrum
QoS	Quality of Service
QPSK	Quadrature Phase Shift Keying
RAM	Random-Access Memory
RFD	Reduced Function Device
RFID	Radio Frequency Identification
RIB	Routing Information Base
ROM	Read-Only Memory
RSSI	Received Signal Strength Indication
RTT	Round Trip Time
SAC	Source Address Compression
SACK	Selective Acknowledgment
SAE	Source Address Encoding
SAM	Source Address Mode
SAP	Service Access Point
SCI	Source Context Identifier
SDU	Service Data Unit
SNAP	Subnetwork Access Protocol
SoC	System-on-a-Chip
SOF	Start of Frame
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagramm Protocol
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
UWB	Ultra-Wideband
WLAN	Wireless Local Area Network
WPAN	Wireless Personal Area Network

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Arkko u. a.] ARKKO, Jari ; PERKINS, Charles ; JOHNSON, Dave: *Mobility Support in IPv6*. http://tools.ietf.org/html/rfc6275. – abgerufen am 8. August 2012

[Borman u. a.] BORMAN, David A. ; DEERING, Stephen E. ; HINDEN, Robert M.: *IPv6 Jumbograms*. http://tools.ietf.org/html/rfc2675. – abgerufen am 6. September 2012

[Crawford] CRAWFORD, M.: *Transmission of IPv6 Packets over Ethernet Networks*. http://tools.ietf.org/html/rfc2464. – abgerufen am 9. Oktoberi 2012

[Deering u. Hinden] DEERING, Stephen E. ; HINDEN, Robert M.: *Internet Protocol, Version 6 (IPv6) Specification*. http://tools.ietf.org/html/rfc2460. – abgerufen am 8. August 2012

[Dierks] DIERKS, Tim: *The Transport Layer Security (TLS) Protocol Version 1.2*. http://tools.ietf.org/html/rfc5246. – abgerufen am 2. September 2012

[Dunkels] DUNKELS, Adam: *dunkels07rime.pdf*. http://www.sics.se/~adam/dunkels07rime.pdf. – abgerufen am 9. Oktober 2012

[Floyd u. a.] FLOYD, Sally ; RAMAKRISHNAN, K. K. ; BLACK, David L.: *The Addition of Explicit Congestion Notification (ECN) to IP*. http://tools.ietf.org/html/rfc3168. – abgerufen am 6. August 2012

[Gutierrez u. a. 2011] GUTIERREZ, Jose A. ; WINKEL, Ludwig ; JR, Edgar H. C. ; JR, Raymond L. B.: *Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensors With IEEE 802.15.4*. 3. Aufl. Standards Information Network, 2011. – ISBN 073816285X

[Haberman u. Thaler] HABERMAN, Brian ; THALER, Dave: *Unicast-Prefix-based IPv6 Multicast Addresses*. http://tools.ietf.org/html/rfc3306. – abgerufen am 7. August 2012

[Heffner u. a.] HEFFNER, John W. ; CHANDLER, Ben ; MATHIS, Matt: *IPv4 Reassembly Errors at High Data Rates*. http://tools.ietf.org/html/rfc4963. – abgerufen am 5. September 2012

[Hui u. Thubert a] HUI, Jonathan ; THUBERT, Pascal: *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. http://tools.ietf.org/html/rfc6282. – abgerufen am 5. August 2012

[Hui u. Thubert b] HUI, Jonathan ; THUBERT, Pascal: *LoWPAN fragment Forwarding and Recovery*. http://tools.ietf.org/html/draft-thubert-6lowpan-simple-fragment-recovery-07. – abgerufen am 16. Juni 2012

[Kupris u. Sikora 2007] KUPRIS, Gerald ; SIKORA, Axel: *ZigBee : Datenfunk mit IEEE 802.15.4 und ZigBee*. Poing : Franzis, 2007. – ISBN 9783772341595 3772341594

[Kushalnagar u. a.] KUSHALNAGAR, Nandakishore ; MONTENEGRO, Gabriel ; CULLER, David E. ; HUI, Jonathan W.: *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. http://tools.ietf.org/html/rfc4944. – abgerufen am 2. Juli 2012

[McCann u. a.] MCCANN, Jack ; MOGUL, Jeffrey ; DEERING, Stephen E.: *Path MTU Discovery for IP version 6*. http://tools.ietf.org/html/rfc1981. – abgerufen am 5. September 2012

[Mogul u. Deering] MOGUL, J. C. ; DEERING, S. E.: *Path MTU discovery*. http://tools.ietf.org/html/rfc1191. – abgerufen am 5. September 2012

[Nichols u. a.] NICHOLS, Kathleen ; BLACK, David L. ; BLAKE, Steven ; BAKER, Fred: *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. http://tools.ietf.org/html/rfc2474. – abgerufen am 6. August 2012

[Postel a] POSTEL, J.: *Internet Protocol*. http://tools.ietf.org/html/rfc791. – abgerufen am 8. August 2012

[Postel b] POSTEL, J.: *User Datagram Protocol*. http://tools.ietf.org/html/rfc768. – abgerufen am 2. September 2012

[Postel u. Reynolds] POSTEL, J. ; REYNOLDS, J. K.: *Standard for the transmission of IP datagrams over IEEE 802 networks*. http://tools.ietf.org/html/rfc1042. – abgerufen am 9. Oktoberi 2012

[Sargent u. a.] SARGENT, Matt ; PAXSON, Vern ; ALLMAN, Mark ; CHU, Jerry: *Computing TCP\x27s Retransmission Timer*. http://tools.ietf.org/html/rfc6298. – abgerufen am 5. September 2012

[Savola u. Haberman] SAVOLA, Pekka ; HABERMAN, Brian: *Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address*. http://tools.ietf.org/html/rfc3956. – abgerufen am 7. August 2012

[Shelby u. a. 2009] SHELBY, Zach ; BORMANN, Carsten ; MULLIGAN, Geoff: *6LoWPAN : The Wireless Embedded Internet*. 1. Aufl. New York, NY : Wiley, J, 2009. – ISBN 978–0–470–74799–5

[Tanenbaum 2011] TANENBAUM, Andrew S.: *Computer networks*. 5. Aufl. Boston : Pearson, 2011. – ISBN 978–0–13–255317–9

[Vasseur u. Dunkels 2010] VASSEUR, Jean-Philippe ; DUNKELS, Adam: *Interconnecting Smart Objects with IP: The Next Internet*. 1. Aufl. Morgan Kaufmann, 2010. – ISBN 0123751659

[Vixie] VIXIE, Paul: *Extension Mechanisms for DNS (EDNS0)*. http://tools.ietf.org/html/rfc2671. – abgerufen am 7. August 2012

Anhang A

Anhang: Ergänzende Informationen

A.1 Die Bitübertragungsschicht des Funkstandards 802.15.4

A.1.1 Frequenzbänder

Der Standard IEEE 802.15.4 wurde so geschrieben, dass die entsprechenden Geräte in der Weise hergestellt werden können, dass sie in jedem von drei eigenen Frequenzbändern arbeiten können. Zwei dieser Bänder können nur in bestimmten geographischen Gebieten genutzt werden, aber eines der Bänder ist nahezu auf der gesamten Welt verfügbar.

In Europa veröffentlicht das European Telecommunications Standards Institute (ETSI) Empfehlungen, die von allen europäischen Regulierungsbehörden anerkannt werden, wobei jedes Servicegebiet unter eine individuelle nationale typgenehmigende Behörde fällt. Innerhalb des europäischen Servicegebietes wird ein gemeinschaftlich genutztes Frequenzband zwischen 868 MHz und 868,6 MHz bereitgestellt, welches einen einzelnen Kanal von niedriger Datenrate mit weniger als 1% des maximalen Übertragungsauslastungsgrades unterstützt.

In China wurden durch die chinesische Regierung kürzlich Bestimmungen erlassen, die es erlauben, einen lizenzfreien Frequenzbereich für drahtlose Sensornetzwerke zu schaffen, der zwischen 779 MHz und 787 MHz liegt. In gleicher Weise gab die japanische Regierung ein Frequenzband für die Nutzung solcher Dienste frei, doch befindet sich dieses im Bereich von 850 MHz bis 856 MHz.

Die Federal Communications Commission (FCC) ist die Regulierungsbehörde der Vereinigten Staaten von Amerika. Die FCC hat nur innerhalb der Vereinigten Staaten Amtsbefugnis, ihre Vorschriften werden allerdings als Modell für viele andere Nationen in Amerika und dem pazifischen Raum genutzt. So wie in Europa, ist ein gesondertes Band für Dienste verfügbar. Dieser als 915 MHz-Band bezeichnete Frequenzbereich erstreckt sich von 902 MHz bis 928 MHz. In diesem Frequenzband können mehrere IEEE 802.15.4-Kanäle mit niedriger Datenrate bereitgestellt werden.

Um die Kosten im Bereich des Produktdesigns, des Marketings und des Vertriebs zu senken und Anwendungen zu ermöglichen, die eine Erreichbarkeit zwischen verschiedenen behördlichen Gebieten erfordern, wäre es wünschenswert ein Frequenzband zu benutzen, welches weltweit verfügbar ist. Dieses Band sollte idealerweise lizenzfrei sein und genügend Bandbreite besitzen, um eine Vielzahl von Kanälen zu ermöglichen und hoch genug im Frequenzspektrum, so dass relativ effiziente Antennen möglich sind. Die Lage im Spektrum sollte aber niedrig genug sein, um die Verwendung von Ein-Chip-Lösungen unter Einsatz von günstigen IC-Herstellungsverfahren nicht zu verhindern.

Das Frequenzband, welches diesen Anforderungen am besten entsprach, ist das 2,4 GHz Industrial, Scientific and Medical (ISM)-Band. Es erstreckt sich von 2400 MHz bis 2483,5 MHz. Bis auf wenige Ausnahmen ist es weltweit verfügbar. Die Wellenlänge von 12,25 cm ermöglicht den effizienten Einsatz relativ kleiner Antennen. Die größere verfügbare Bandbreite ermöglicht es, mehrere Kanäle mit relativ hoher Datenrate bereitzustellen. Voneinander unabhängige Netzwerke mit gleichen Endgeräten können so parallel betrieben werden, ohne sich gegenseitig zu stören. [Gutierrez u. a., 2011, S. 44-45]

A.1.2 Datenraten

Aufgrund der physikalischen Eigenschaften der Frequenzbänder und den Bestimmungen in den Servicegebieten spezifiziert der Standard IEEE 802.15.4 unterschiedliche Datenraten und Modulationsverfahren. Diese durch den Standard definierten Bitübertragungsschichten werden dann in den dafür vorgesehenen Frequenzbändern genutzt.

Die vorgeschriebene Grunddatenrate im 868 MHz-Band ist 20 kbit/s, aber es darf optional eine höhere Datenrate von 100 kbit/s unter Verwendung des Offset Quadrature Phase Shift Keying (O-QPSK)-Modulationsverfahrens oder 250 kbit/s unter Einsatz des Parallel Sequence Spread Spectrum (PSSS)-Modulationsverfahrens implementiert werden.

In ähnlicher Weise ist die Grunddatenrate des 915 MHz-Bandes mit 40 kbit/s definiert. Die Implementierungen dürfen jedoch wahlweise eine Datenrate von 250 kbit/s unter Verwendung des O-QPSK- oder PSSS-Modulationsverfahrens anbieten.

Die 2,4 GHz Bitübertragungsschicht stellt Datenraten von 250 kbit/s unter Verwendung der Modulationsart O-QPSK bereit. Der Zusatz A ermöglicht auf der 2,4 GHz Bitübertragungsschicht unter Einsatz der Chirp Spread Spectrum (CSS)-Modulationstechnik Datenraten von 250 kbit/s und 1000 kbit/s. Die gleiche Erweiterung des Standards definiert drei Ultra-Wideband (UWB) Bitübertragungsschichten mit Datenraten von 110 kbit/s, 850 kbit/s 1700 kbit/s, 6,810 kbit/s und 27,2240 kbit/s.

Die Zusätze C und D beschreiben zwei weitere Bitübertragungsschichten im 780 MHz- und 950 MHz-Band. Entsprechende Geräte mit Unterstützung des 780 MHz-Bandes ermöglichen Datenraten von 250 kbit/s unter Verwendung der O-QPSK- bzw. M-ary Phase Shift Keying (MPSK)-Modulationsverfahren. Analog dazu werden im 950 MHz-Band Datenraten von 20 kbit/s und 100 kbit/s unter Einsatz der Modulationstechniken Binary Phase Shift Keying (BPSK) und Gaussian Frequency Shift Keying (GFSK) ermöglicht. [Gutierrez u. a., 2011, S. 47, 55]

Die Tabellen A.1 ff. fassen die Modulationsparameter und Datenraten noch einmal zusammen.

Einige Angaben in der Tabelle erfordern eine nähere Erläuterung. Die Spalte *Symbol Rate* beschreibt die Anzahl der Symbole, die pro Sekunde übertragen werden und steht in Beziehung zu der Bitrate. Das eingesetzte Modulationsverfahren bestimmt die Anzahl der Bits, die einem Symbol zugeordnet werden. Bei der Modulationsart BPSK entspricht die Symbolrate der Datenrate, da das Symbol die Wertigkeit von genau einem Bit aufweist.

Die Spalte *Chip Rate* enthält die Anzahl der übertragenen Chips pro Sekunde. Die Einheit steht im Zusammenhang mit dem Codemultiplexverfahren. Unter Zuhilfenahme von Spreiztechniken wird das Nutzsignal mit einer definierten Symbolrate über die ursprünglich benötigte Bandbreite hinaus aufgeweitet bzw. gespreizt. Beim DSSS-Verfahren wird das Datensignal mit einem orthogonalen Spreizsignal multipliziert oder XOR-verknüpft. Die Chiprate ist also höher als die Symbolrate.

Dieses Signal wird dann unter Verwendung von bspw. der BPSK- oder Quadrature Phase Shift Keying (QPSK)-Modulationstechnik auf eine Trägerfrequenz moduliert.

Tabelle A.1: Frequenzbänder und Modulationsparameter des Standards IEEE 802.15.4-2006, entnommen [Gutierrez u. a., 2011, S. 48]

Band	Frequency Band	Bit Rate	Symbol Rate	Modulation	Chip Rate
868 MHz	868-868.8 MHz	20 kb/s	20 ksymbols/s	Binary Phase Shift Keying (BPSK)	300 kchip/s
		100 kb/s	25 ksymbols/s	Offset Quadrature Phase Shift Keying (O-QPSK)	400 kchip/s
		250 kb/s	12.5 ksymbols/s	Parallel Sequence Spread Spectrum (PSSS)	400 kchip/s
915 MHz	902-928 MHz	40 kb/s	40 ksymbols/s	Binary Phase Shift Keying (BPSK)	600 kchip/s
		250 kb/s	62.5 ksymbols/s	Offset Quadrature Phase Shift Keying (O-QPSK)	1 Mchip/s
		250 kb/s	50 ksymbols/s	Parallel Sequence Spread Spectrum (PSSS)	1.6 Mchip/s
2.4 GHz	2.4-2.4835 GHz	250 kb/s	62.5 ksymbols/s	Offset Quadrature Phase Shift Keying (O-QPSK)	2 Mchip/s

A.1.3 Kanalzuweisung

Der Standard 802.15.4-2006 und seine Zusätze A, C und D definieren 16 Kanäle im 2450 MHz-Band, 30 Kanäle im 915 MHz-Band, 3 Kanäle im 868 MHz-Band, 14 überlappende CSS-Kanäle im 2450 MHz-Band, 16 Kanäle in drei UWB-Bändern (500 MHz und 3.1 GHz bis zu 10.6 GHz), 8 Kanäle im 780 MHz-Band und 22 Kanäle im 950 MHz-Band.

Der Standard 802.15.4 benutzt eine Kombination aus Kanalnummern und Kanalseiten, um die Übertragungsfrequenz zu bestimmen. Es gibt insgesamt 32 mögliche Seiten, aber nur sieben davon sind im aktuellen Standard definiert. Die restlichen Seiten sind für zukünftige Erweiterungen reserviert.

Jede Seite ist in 27 Kanäle unterteilt. Diese sind durchnummeriert von 0 bis 26. Den Seiten werden die Zahlen von 0 bis 31 zugeordnet. Die Seite 0 beispielsweise enthält einen Kanal im 868 MHz-Band, 10 Kanäle im 915 MHz-Band und 16 Kanäle im 2,4 GHz-Band. Die Tabelle A.1 verdeutlicht dies noch einmal. [Gutierrez u. a., 2011, S. 49]

A.1.4 Dienste auf der Bitübertragungsschicht

Die Bitübertragungsschicht stellt eine Schnittstelle zwischen dem Funkkanal und der MAC-Unterschicht bereit. Realisiert wird dieses Interface durch einen Dienst für den Datenaustausch und

Tabelle A.2: Frequenzbänder und Modulationsparameter des Standards IEEE 802.15.4, Zusatz A, entnommen [Gutierrez u. a., 2011, S. 48]

Band	Frequency Band	Bit Rate	Symbol Rate	Modulation
Ultra-Wide Band 1	250-750 MHz	110 kb/s	0.12 Msymbols/s	Burst Position Modulation (BPM) with Binary Phase Shift Keying (BPSK)
		850 kb/s	0.98 Msymbols/s	
		1700 kb/s	1.95 Msymbols/s	
		6810 kb/s	7.8 Msymbols/s	
		27240 kb/s	15.6/32.2 Msymbols/s	
Ultra-Wide Band 2	3.1-4.8 GHz	110 kb/s	0.12 Msymbols/s	
		850 kb/s	0.98 Msymbols/s	
		1700 kb/s	1.95 Msymbols/s	
		6810 kb/s	7.8 Msymbols/s	
		27240 kb/s	15.6/32.2 Msymbols/s	
Ultra-Wide Band 3	6.0-10.6 GHz	110 kb/s	0.12 Msymbols/s	
		850 kb/s	0.98 Msymbols/s	
		1700 kb/s	1.95 Msymbols/s	
		6810 kb/s	7.8 Msymbols/s	
		27240 kb/s	15.6/32.2 Msymbols/s	
2.4 GHz	2.4 -2.4835 GHz	250 kb/s	166.67 ksymbols/s	Chirp Spread Spectrum (CSS)
		1000 kb/s		

Tabelle A.3: Frequenzbänder und Modulationsparameter des Standards IEEE 802.15.4, Zusatz C, entnommen [Gutierrez u. a., 2011, S. 48]

Band	Frequency Band	Bit Rate	Symbol Rate	Modulation	Chip Rate
780 MHz	779-787 MHz	250 kb/s	62.5 ksymbols/s	Offset Quadrature Phase Shift Keying (O-QPSK)	1000 kchip/s
				M-ary Phase Shift Keying (MPSK)	

Tabelle A.4: Frequenzbänder und Modulationsparameter des Standards IEEE 802.15.4, Zusatz D, entnommen [Gutierrez u. a., 2011, S. 48]

Band	Frequency Band	Bit Rate	Symbol Rate	Modulation	Chip Rate
950 MHz	950-956 MHz	20 kb/s	20 ksymbols/s	Binary Phase Shift Keying (BPSK)	300 kchip/s
		100 kb/s	100 ksymbols/s	Gaussian Frequency Shift Keying (GFSK)	N/A

Tabelle A.5: Kanalseite 0 des Standards IEEE 802.15.4, entnommen [Gutierrez u. a., 2011, S. 49]

	Channel	**Center Frequency (MHz)**	**Availability**
868 MHz Band	0	868.3	Europe
915 MHz Band	1	906	America
	2	908	
	3	910	
	4	912	
	5	914	
	6	916	
	7	918	
	8	920	
	9	922	
	10	924	
2.4 GHz Band	11	2405	worldwide
	12	2410	
	13	2415	
	14	2420	
	15	2425	
	16	2430	
	17	2435	
	18	2440	
	19	2445	
	20	2450	
	21	2455	
	22	2460	
	23	2465	
	24	2470	
	25	2475	
	26	2480	

einem Dienst für das Management. Der Managementdienst wird auch Physical Layer Management Entity (PLME) genannt. Der Zugriff auf den Datendienst wird durch den Physical Layer Data Service Access Point (PD-SAP) realisiert. Die PLME stellt zu diesem Zweck den Physical Layer Management Entity Service Access Point (PLME-SAP) bereit.

Der Dienst für den Datenaustausch stellt der MAC-Unterschicht drei Dienstelemente zur Verfügung. Diese heißen PD-DATA.request, PD-DATA.confirm und PD-DATA.indication. Dieser Datendienst benötigt keine Bestätigung durch die MAC-Schicht auf der Gegenseite – daher wurde diese Grundfunktion nicht implementiert. Die Grafik A.1 verdeutlicht den Austausch von Datenpaketen am Beispiel eines Sequenzdiagrammes.

Der Managementdienst unterstützt Befehle zum Verwalten der Einstellungen für die Datenübertragung. Dies geschieht durch die Konfiguration der Attribute der PIB auf der Bitübertragungsschicht. Diese Parameter können über die Dienstelemente PLME-GET und PLME-SET ausgelesen und verändert werden (siehe Grafik A.2).

Für den Zugriff auf das drahtlose Übertragungsmedium existieren Funktionen für das Aktivieren und Deaktivieren des Sendeempfangsmoduls (siehe Grafik A.3). Darüber hinaus werden die Funktionen für die Mechanismen Energie Detektion und CCA bereitgestellt. [Gutierrez u. a., 2011, S. 73-75]

Die Tabelle A.6 fasst die Dienstelemente der PLME noch einmal zusammen. Die letzten vier Spalten geben dabei Auskunft über das Vorhandensein von Dienstelementen unter diesem speziellen Dienstzugangspunkt (Service Access Point (SAP)) der Bitübertragungsschicht. Die Zugangspunkte haben die Aufgabe, die Dienste einer Schicht des OSI-Referenzmodells der darüberliegenden Schicht zur Verfügung zu stellen und durch sie wird der Zugriff auf die Dienstelemente gewährt. Bei den Dienstelementen werden vier Arten unterschieden:

- *Request*: die Schicht N nutzt so einen Dienst der darunterliegenden Schicht N-1 und veranlasst das Ausführen einer Aufgabe.
- *Confirm*: die Schicht N nimmt die Antwort der Schicht N-1 nach einem *Request* entgegen.
- *Indication*: Die Schicht N-1 informiert die Schicht N über ein Ereignis.
- *Response*: Anwort der Schicht N an die Schicht N-1 nach einer *Indication*.

Da die Bitübertragungsschicht die unterste Schicht im OSI-Referenzmodell ist, entfallen die Zugriffspunkte für *Response* und *Indication*.

A.1.5 Clear Channel Assessment

Möchte man Pakete in Netzwerken mit deaktiviertem Beacon-Signal (unslotted Mode) übertragen, greift die MAC-Unterschicht vor dem Senden eines Datenrahmens oder MAC-Befehlsrahmens auf das CCA Dienstelement der Bitübertragungsschicht zu. Dieses Verfahren zur Prüfung der Verfügbarkeit des Übertragungsmediums wird auch in Netzen mit aktiviertem Beacon-Signal (slotted Mode) während der Zugriffskonfliktperiode durchgeführt.

Während der CCA-Vorgang durchgeführt wird, aktiviert die Bitübertragungsschicht den Empfänger, führt eine CCA-Messung durch und deaktiviert den Empfänger wieder. Im Anschluss an die CCA-Messung nutzt die Bitübertragungsschicht das PLME-CCA.confirm-Dienstelement, um anzuzeigen, ob der Übertragungskanal belegt ist oder nicht. [Gutierrez u. a., 2011, S. 75-76]

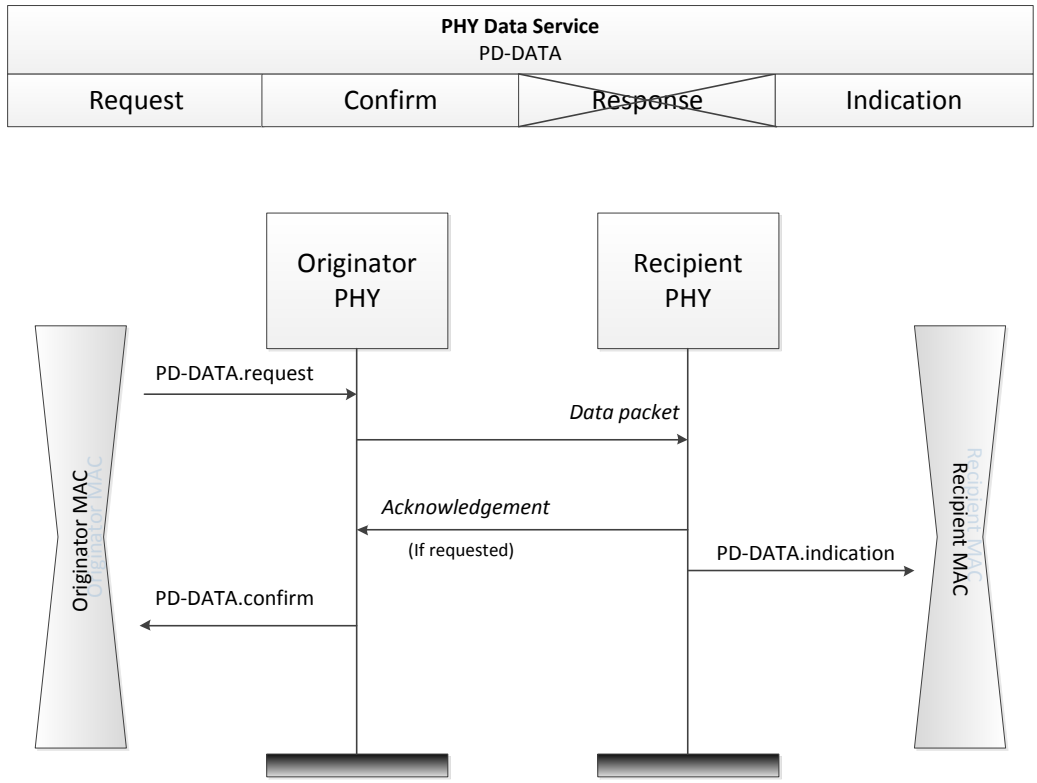


Abbildung A.1: Mechanismus für den Austausch von Datenpaketen (Sequenzdiagramm), abgezeichnet von [Gutierrez u. a., 2011, S. 74]

Tabelle A.6: Die Dienstelemente des Managementdienstes der Bitübertragungschicht, entnommen von [Gutierrez u. a., 2011, S. 74]

Primitive	**Category**	**Description**	Request	Confirm	Response	Indication
GET	Communication Settings	PHY PAN information base management	X	X		
SET			X	X		
SET-TRX-STATE	Radio Control	Enables/Disables radio system	X	X		
CCA	RF Energy Sensing	RF energy sensing: Clear Channel Assessment, Energy Detection	X	X		
ED			X	X		

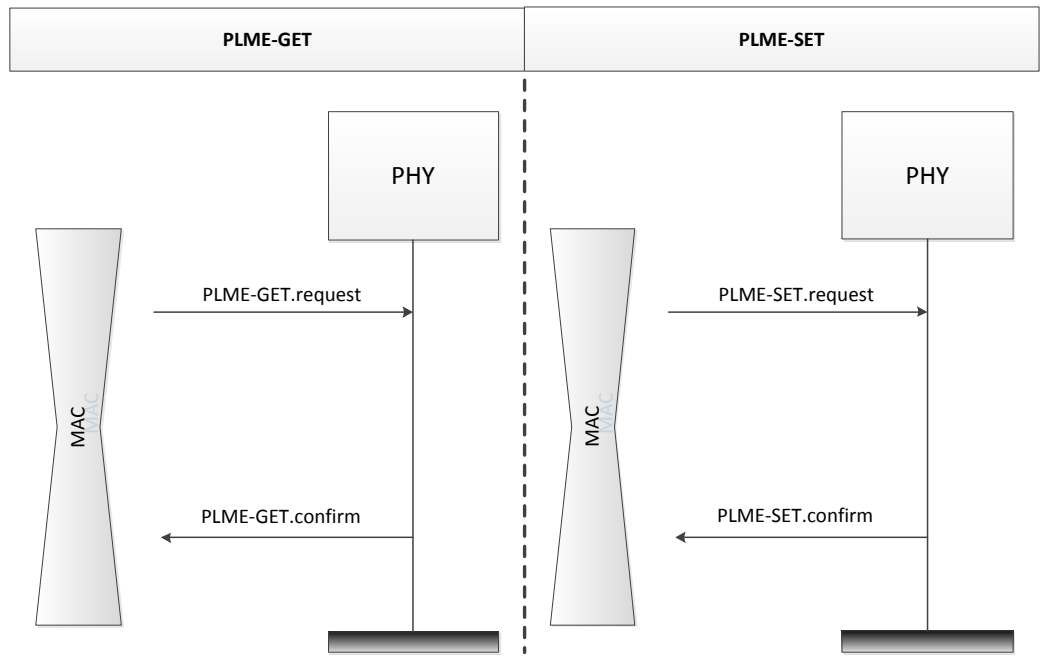


Abbildung A.2: Schreib- und Lesemechanismus der PIB auf der Bitübertragungsschicht (Sequenzdiagramm), abgezeichnet von [Gutierrez u. a., 2011, S. 75]

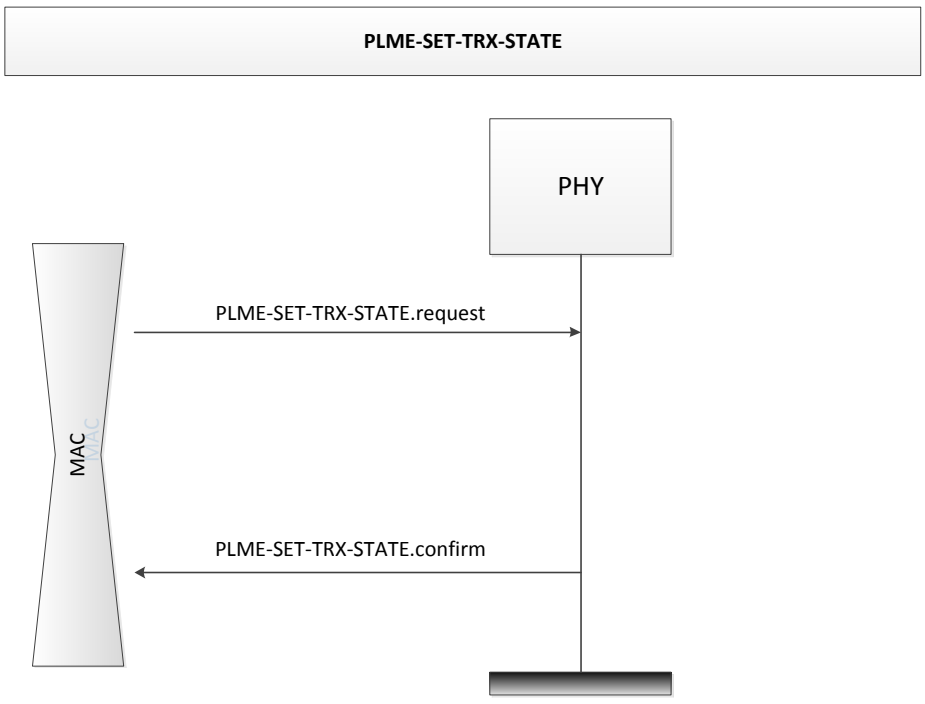


Abbildung A.3: Mechanismus für das Aktivieren und Deaktivieren des Sendeempfangsmoduls (Sequenzdiagramm), abgezeichnet von [Gutierrez u. a., 2011, S. 75]

Die Abbildung A.4 veranschaulicht den Ablauf noch einmal anhand eines Sequenzdiagrammes.

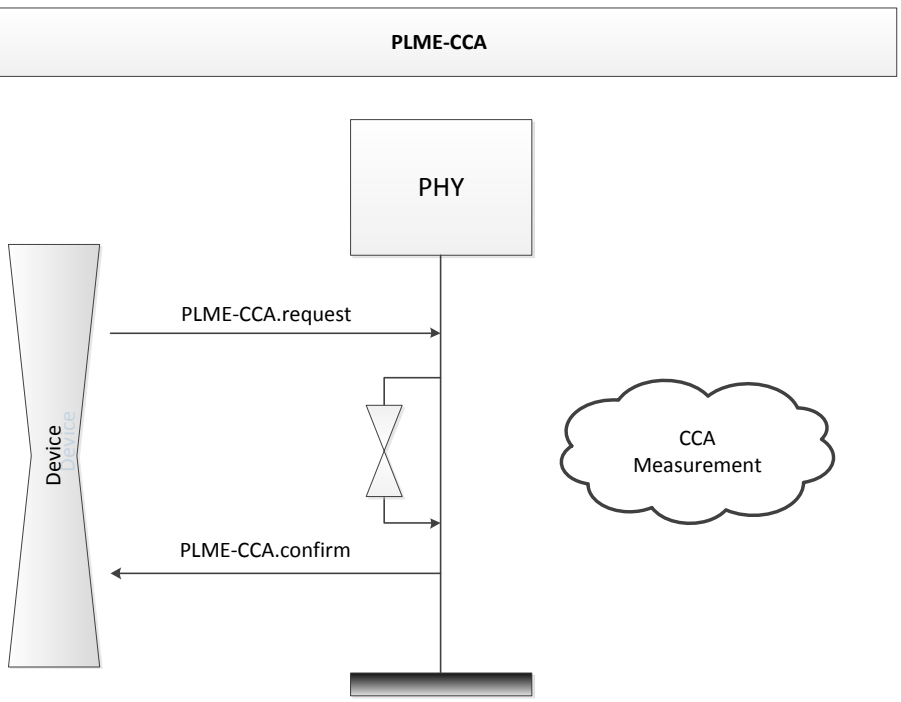


Abbildung A.4: Der CCA-Mechanismus für die Verfügbarkeitsprüfung des Übertragungskanals (Sequenzdiagramm), abgezeichnet von [Gutierrez u. a., 2011, S. 76]

A.1.6 Energie Detektion

Das PLME-ED-Dienstelement erlaubt es einem Gerät, eine Messung der Energie auf dem Übertragungskanal durchzuführen, auf dem es gerade arbeitet. Die Messung ähnelt jener des PLME-CCA-Dienstelements sehr, liefert aber ein Ergebnis mit einer höheren Auflösung. Es wird ein Energielevel im Bereich von 0 bis 255 zurückgegeben. Das Nutzen dieses Dienstelementes kann die Funktionalität der Netzwerkschicht verbessern. [Gutierrez u. a., 2011, S. 76]

Die Abbildung A.5 stellt den Vorgang noch einmal unter Zuhilfenahme eines Sequenzdiagrammes dar.

A.1.7 Rahmenaufbau

Die PDU der Bitübertragungsschicht des Standards 802.15.4, auch PPDU genannt, nimmt die nötigen Anpassungen an der Transceiverbaugruppe vor und enthält die Datenstrukturen der höheren Schichten des OSI-Referenzmodells. Die PPDU setzt sich aus drei Teilen zusammen: Dem Synchronisationsheader, dem Header der Bitübertragungsschicht und den Nutzdaten mit variabler Länge in Gestalt der Service Data Unit (SDU).

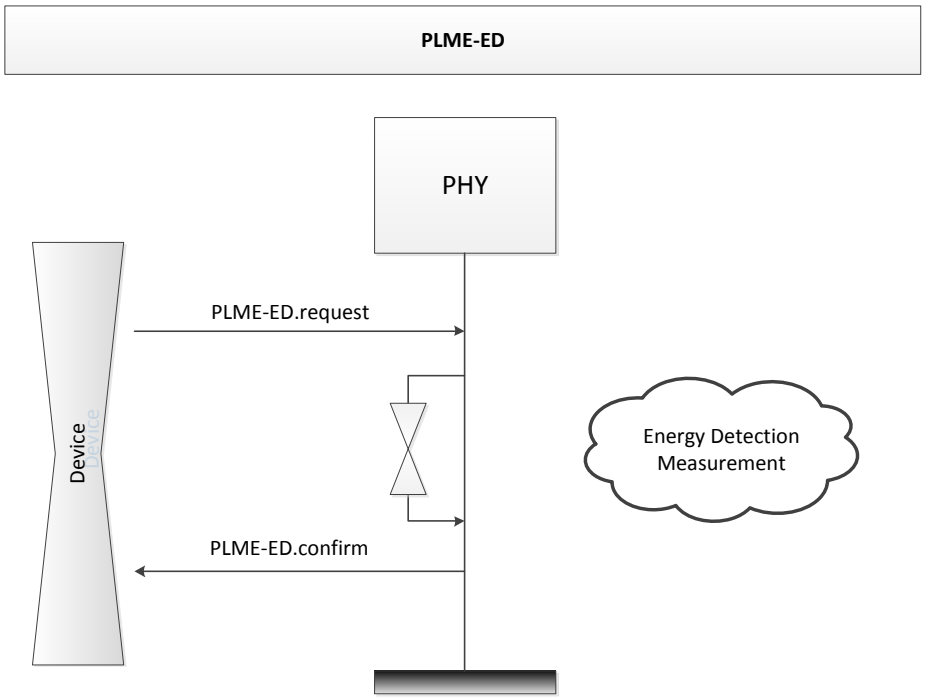


Abbildung A.5: Mechanismus für Messung der Energie auf einem Übertragungskanal (Sequenzdiagramm), abgezeichnet von [Gutierrez u. a., 2011, S. 77]

Der PPDU-Synchronisationsheader besteht aus zwei Datenfeldern, einer Präambel und einem Start of Frame (SOF)-Delimiter. Mit Ausnahme der PSSS- und der UWB-Bitübertragungsschicht besteht die Präambel aus 32 Bit. Alle Bits werden auf null gesetzt, es gilt jedoch zu bedenken, dass durch die Anwendung des DSSS-Verfahrens noch eine Multiplikation mit dem Spreizsignal erfolgt. Im Falle des optionalen PSSS-Modus ist die Präambel 40 Bits lang, wenn das Gerät im 886 MHz-Band arbeitet, und 30 Bit lang, wenn es im 915 MHz-Band eingesetzt wird. In beiden Fällen wird die Präambel aus der ersten Sequenz der entsprechenden Codetabelle gebildet. Das Präambelfeld dient der Synchronisation des Taktgebers im Empfänger auf die empfangenden Bits und Chips. Der SOF-Delimiter wird aus dem 8 Bit Muster 0x6f (11100101_2) gebildet und ermöglicht dem Empfänger den Paketanfang in einem Datenstrom zu ermitteln.

Der Header der Bitübertragungsschicht besteht aus einem einzelnen Byte, bei dem das Most Significant Bit (MSB) nicht genutzt wird und für spätere Erweiterungen reserviert ist. Die verbleibenden Bits geben die zu erwartende Rahmenlänge auf MAC-Ebene von maximal 127 Byte an. Rahmenlängen von 0 bis 4 Bytes und 6 bis 8 Bytes werden nicht verwendet. Bei Rahmen der Länge 5 handelt es sich um MPDU-Bestätigungsrahmen. Bei Rahmen mit Größen von 9 oder mehr Bytes handelt es sich um die MPDU-Nutzdaten, auch Payload genannt, für den Dienst der MAC-Protokoll-Schicht.

Die Nutzdaten der Bitübertragungsschicht bestehen aus einem einzigen Datenfeld, das Physical Layer Service Data Unit (PSDU) genannt wird. Die PSDU ist von variabler Länge und enthält, wie bereits erwähnt, die MPDU Nutzdaten welche die darüberliegende MAC-Schicht repräsentieren. [Gutierrez u. a., 2011, S. 77-78]

Die Grafik A.6 zeigt die Struktur der PPDU

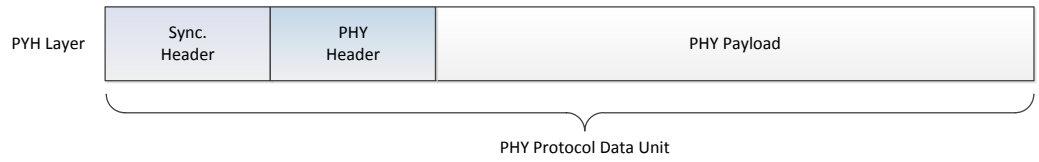


Abbildung A.6: PPDU-Struktur, abgezeichnet von [Gutierrez u. a., 2011, S. 78]

A.2 Contiki und μIP

A.2.1 Eingabeverarbeitung

Wenn der *Communication Device*-Treiber ein Paket aus dem Netzwerk empfängt, ruft er die μIP-Eingabeverarbeitungsfunktion auf, um das Paket an μIP zu übergeben. Der Programmcode von μIP, der für die Eingabeverarbeitung verantwortlich ist, analysiert den Paket-Header und bestimmt, ob die Anwendung aufgerufen werden soll. Ist das der Fall, übergibt μIP die Anwendungsdaten an die Anwendung.

Die μIP-Eingabeverarbeitung beginnt mit dem IP-Header. Der Ablauf der Eingabeverarbeitungsfunktionalität wird in der Abb. 13.4 gezeigt. Die Paketheader werden von oben nach unten analysiert, beginnend mit dem IP-Header. Zuerst wird das erste Byte des IP-Headers überprüft, um sicherzustellen, dass das eintreffende Paket ein IP-Paket ist und das IP-Protokoll mit der Version identisch ist, die μIP verarbeiten kann. μIP kann sowohl IPv4- als auch IPv6-Daten verarbeiten - aber nur eine Version zugleich.

Nachdem sichergestellt wurde, dass das Paket den richtigen IP-Header besitzt, überprüft μIP die Gültigkeit des IP-Headers. Die durch den IP-Header angegebene Länge wird mit der Länge des Paketes verglichen, welches von der darunterliegenden Schicht des OSI-Referenzmodells empfangen wurde. Ist die angegebene Länge im IP-Header länger als das Paket im Buffer, wird das Paket als missgebildet angesehen und verworfen. Ist das Paket im Buffer länger als durch den IP-Header angegeben, wird das Paket als gültig angenommen, scheint aber zusätzliche, nutzlose Daten am Paketende zu enthalten. μIP fährt mit dem Verarbeiten der Daten trotzdem fort. Für das Protokoll IP in der Version 4 wird das *Fragment*-Flag überprüft. Handelt es sich bei dem Paket um ein IP-Fragment, wird es in den Defragmentierungsbuffer kopiert. Wenn der Buffer nun ein komplettes IP-Paket enthält, wird das wieder zusammengesetzte IP-Paket als eingetroffenes IP-Paket angesehen und die Eingabeverarbeitung wird fortgesetzt. Für IPv6 wird das Zusammensetzen der IP-Fragmente als Teil der Verarbeitung der *Extension*-Header durchgeführt.

Als Nächstes werden Sender- und Empfängeradresse des Paketes überprüft. Pakete mit illegalen IP-Senderadressen werden verworfen. Beispiele hierfür sind Pakete, wo als Senderadresse eine Broadcast- oder Multicastadresse angegeben wurde. Pakete mit einer Empfängeradresse, die nicht zu einer der IP-Adressen der Station passt, werden verworfen oder an das *Packet-Forwarding*-Module von μIP übergeben. Das *Packet-Forwarding*-Modul von μIP kann das Paket nun an den *Next-Hop*-Nachbarn weiterleiten, abhängig von der Empfänger-IP-Adresse.

Pakete mit einer Empfänger-IP-Adresse, die zu einer der IP-Adressen der Station passen, werden weiterverarbeitet. Der Programmcode für die Verarbeitung unterscheidet sich hier zwischen IPv4 und IPv6. Für IPv4 wird die IP-Header Prüfsumme berechnet, um sicherzustellen, dass sie mit der empfangenen Prüfsumme identisch ist. Bei IPv6 ist keine Header-Prüfsumme definiert. Für IPv6-Pakete überprüft μIP das Vorhandensein beliebiger *Extension*-Header und verarbeitet diese. Wenn μIP Fehler in den *Extension*-Headern entdeckt, wird eine ICMPv6-Fehlermeldung generiert und an den Sender übertragen.

Für den Fall das μIP festgestellt hat, dass der IP-Header sowie die Länge des Paketes korrekt sind und die Empfängeradresse zu einer der Adressen der Station passt, wird das Paket an das richtige Protokoll der Transportschicht übergeben. μIP unterstützt drei dieser Protokolle: ICMP, UDP und TCP. ICMP ist genaugenommen kein Protokoll der Transportschicht, wird aber als solches in μIP implementiert!

Das Paket wird abhängig von dem Protokoll der höheren Schichten nun unterschiedlich weiterverarbeitet. Das Protokoll TCP ist hier das komplexeste Protokoll, das μIP implementiert. Die Verarbeitung des Protokolls UDP ist sehr einfach. Die Verarbeitung des Protokolls ICMP ist für IPv4 sehr einfach, jedoch etwas komplexer für IPv6. [Vasseur u. Dunkels, 2010, S. 169-171]

A.2.2 ICMP-Eingabeverarbeitung

Für IPv4 besteht die ICMP-Verarbeitung aus einer einzigen Funktionalität: auf eintreffende ICMP-Echo-Nachrichten zu reagieren. Falls eine Station eine ICMP-Echo-Nachricht empfängt, sendet es eine ICMP-Echo-Antwort. Die Antwort(Reply)-Nachricht enthält eine Kopie der Daten der ICMP-Echo-Nachricht. Solche ICMP-Echo-Nachrichten werden vom „Ping“-Programm, welches in den meisten Betriebssystemen vorhanden ist, versandt. Das Ping-Dienstprogramm versendet ICMP-Echo-Nachrichten für das Messen der RTT zu einer Station und um zu überprüfen, ob bestimmte Stationen erreichbar sind. Falls das Ping-Programm eine ICMP-Echo-Antwort empfängt, die zu einer ICMP-Echo-Nachricht passt, die zuvor gesendet wurde, wird die RTT auf dem Bildschirm ausgegeben.

Die ICMP-Verarbeitung für IPv6 ist komplizierter, da das Protokoll hier eine deutlich wichtigere Rolle spielt als in IPv4. Zusätzlich zur Echo-Funktionalität wird das Protokoll für das sogenannte ND, das *Router Discovery* und die *Duplicate Address Detection* sowie weiterer Mechanismen genutzt. μIP unterstützt *Neighbor Solicitation*-Nachrichten, *Neighbor Advertisement*-Nachrichten, *Router Solicitation*-Nachrichten und *Router Advertisement*-Nachrichten. In IPv6 werden die *Neighbor Solicitation*-Nachrichten auch für das Ausführen der *Duplicate Address Detection* genutzt, die von μIP unterstützt wird. [Vasseur u. Dunkels, 2010, S. 171]

A.2.3 UDP-Eingabeverarbeitung

Die UDP-Eingabeverarbeitung in μIP wurde einfach gehalten. Der Programmcode berechnet zuerst die UDP-Prüfsumme um sicherzustellen, dass das Paket gültig ist. Anschließend wird die UDP-Portnummer aus dem Paket verwendet, um die Anwendung zu ermitteln, an welche die Nutzdaten geliefert werden sollen. μIP führt eine Liste von Anwendungen und die entsprechenden UDP-Portnummern, welche sie verwenden. Eine Anwendung kann den sogenannten *Remote End Point* bestimmen, in dem sie die IP-Adresse und den UDP-Port des Kommunkationspartners (*Remote Peer*) angibt. Diese Angaben können auch weggelassen werden. Wird die *Remote*-IP-Adresse und der *Remote*-UDP-Port leer gelassen, nimmt die Anwendung sämtliche Pakete entgegen, die für ihren UDP-Port empfangen wurden. Ansonsten werden nur Pakete von der speziellen IP-Adress- und UDP-Port-Kombination akzeptiert. [Vasseur u. Dunkels, 2010, S. 171-172]

A.2.4 TCP-Eingabeverarbeitung

Die komplexeste Protokollimplementierung in μIP ist die für das Protokoll TCP. Dennoch wurde die Implementierung deutlich einfacher gehalten als TCP-Implementierungen in anderen IP-Stacks so wie bspw. dem BSD-UNIX-Stack. Ursache dafür ist die Tatsache, dass das μIP TCP-Protokoll nur die notwendigen Mechanismen implementiert, um die Anforderungen des Standards zu erfüllen und Interoperabilität zwischen zwei Systemen zu ermöglichen. Da μIP auf geringen Speicherverbrauch hin optimiert wurde und nicht auf hohe Performance, sieht die μIP TCP-Implementierung von der Realisierung zahlreicher performanceverbessernder Mechanismen, die in anderen IP-Stacks vorhanden sind, ab.

Die TCP-Verarbeitung beginnt mit dem Überprüfen der TCP-Prüfsumme. Sie wird auf Grundlage des TCP-Headers, den Anwendungsdaten und Teilen des IP-Headers berechnet. Anschließend wird die TCP-Portnummer des Paketes mit der Liste offener TCP-Verbindungen verglichen. Falls

das Paket zu keiner der offenen Verbindungen passt, überprüft μIP die Liste der „lauschenden“ (listening) TCP-Ports, aber nur für den Fall, dass in dem eingetroffenen Paket das *TCP SYN*-Flag gesetzt wurde. Andernfalls wird durch μIP ein *TCP RST*-Paket an den Sender verschickt.

Handelte es sich um ein *TCP SYN*-Paket für eine „lauschende“ Verbindung, kreiert μIP einen neuen Eintrag in die Tabelle der aktiven TCP-Verbindungen und trägt dort die korrekte TCP-Sequenznummer aus dem eingetroffenen *TCP SYN*-Paket ein. Sollte das *TCP SYN*-Paket die MSS-Option enthalten, merkt sich μIP die MSS, die es über diese Verbindungen übertragen kann. Als nächstes erzeugt μIP ein *TCP SYN*-Paket mit gesetztem *ACK*-Flag und schickt es zurück an den Kommunikationspartner.

War das Paket für eine aktive Verbindung bestimmt, stellt μIP sicher, dass die TCP-Sequenznummer des eingetroffenen Segmentes die ist, die μIP erwartet. Ist sie größer als erwartet, wird signalisiert, dass ein Paket verloren ging. In diesem Fall verwirft μIP das Paket in dem Wissen, dass es der Kommunikationspartner einige Zeit später erneut übertragen wird. Für μIP wäre es möglich, das Paket zwischen zu speichern, so dass es mit dem Eintreffen des fehlenden Paketes sofort verfügbar wäre. Dies würde allerdings Bufferspeicher voraussetzen, der in speicherbegrenzten Systemen, für die μIP ja entworfen wurde, nicht verfügbar ist.

Nach dem Überprüfen der TCP-Sequenznummer wird eine RTT-Schätzung durchgeführt. Der Grund hierfür ist das Bestimmen eines geeigneten Wertes für den Retransmissiontimer. Für TCP-Verbindungen mit einer langen RTT muss der Retransmissiontimer auf einen hohen Wert gesetzt werden. μIP schätzt die RTT durch das Vorhalten eines Zählers für jede TCP-Verbindung. Der Zähler wird während des zyklischen Verarbeitungsschrittes von μIP erhöht. Wenn ein TCP-Paket empfangen wurde, nutzt μIP den Wert des Zählers für die RTT-Schätzung. Zusätzlich wird ein Mittelwertfilter verwendet, um auftretende Spitzen bei der RTT-Schätzung der TCP-Verbindung zu glätten.

Als Nächstes werden unterschiedliche Aktionen in Abhängigkeit vom Status der TCP-Verbindung durchgeführt. Falls die TCP-Verbindung aufgebaut wurde, wird die Anwendung aufgerufen und übernimmt die Anwendungsdaten, die in dem eingetroffenen TCP-Paket vorhanden sind. Die Anwendung verarbeitet die Daten und hat die Möglichkeit, eine Antwort zu erzeugen, die μIP an den Kommunikationspartner zurücksendet. Falls sich die TCP-Verbindung, für die das eingetroffene Paket bestimmt war, im Zustand des Verbindungsauf- oder Abbaus befindet, wird in den entsprechenden Zustand gewechselt. Hier wird so wie in der Spezifikation beschrieben verfahren.

Falls eine TCP-Verbindung auf oder abgebaut wird, informiert μIP die Anwendung über dieses Ereignis durch einen Aufruf. Abhängig vom Zustand der TCP-Verbindung kann die Anwendung entscheiden, auf das Ereignis zu reagieren. Das Paket wird dann durch den Ausgabeverarbeitungsprogrammcode von μIP verarbeitet. [Vasseur u. Dunkels, 2010, S. 172-173]

A.2.5 Ausgabeverarbeitung

Die Ausgabeverarbeitung ist in μIP einfacher als die Eingabeverarbeitung. Die Ausgabeverarbeitung beginnt mit dem Aufrufen der Anwendung durch μIP. Eine durch μIP aufgerufene Anwendung hat die Möglichkeit, ein Datenpaket zu erzeugen. μIP fügt die nötigen Paketheader hinzu und übergibt es dem *Kommunikation Device*-Treiber für die Übertragung (siehe 13.5).

Bei TCP-Verbindungen kann die Anwendung Daten nicht unaufgefordert senden, sondern muss warten, bis sie von μIP aufgerufen wird. μIP ruft die Anwendung nicht nur auf, wenn neue Daten über die Verbindung eintreffen, sondern auch innerhalb der zyklischen Verarbeitungsphase. Das

gibt der Anwendung die Gelegenheit, Daten zu senden, auch wenn keine Daten über die Verbindung eintreffen. Anwendungen, die über das Protokoll UDP kommunizieren, können Daten zu jeder Zeit senden und müssen nicht warten, bis sie von μIP aufgerufen werden.

Die TCP-Ausgabeverarbeitung beginnt, wenn die Anwendung aufgerufen wurde. Das geschieht entweder als Teil der Eingabeverarbeitung oder während der zyklischen Verarbeitungsphase. Erfolgt der Aufruf während der Eingabeverarbeitung, liefert μIP entweder ein Paket an die Anwendung oder informiert diese, dass die Verbindung geöffnet oder geschlossen wurde. In jedem Fall kann die Anwendung Daten an den Kommunikationspartner senden. Falls ja, aktualisiert μIP den Verbindungsstatus für die TCP-Verbindung, stellt den Daten die nötigen Header voran und berechnet die benötigten Prüfsummen, bevor es das Paket versendet. Der Verbindungsstatus muss aktualisiert werden, da aufgrund der Anwendungsdaten die TCP-Sequenznummer der Verbindung erhöht wird. Die Anwendung kann die Verbindung auch schließen oder abbrechen und μIP wird dementsprechend reagieren.

Die UDP-Ausgabeverarbeitung kann entweder beginnen, wenn μIP eine Anwendung aufgrund eintreffender Daten aufruft, oder weil die Anwendung μIP direkt aufruft. μIP fügt den UDP-Header mit der benötigten UDP-Portnummer zu den Anwendungsdaten hinzu und berechnet die UDP-Prüfsumme bevor das Paket der IP-Schicht übergeben wird um den IP-Header hinzuzufügen. Die IP-Schicht fügt ihren Header hinzu und berechnet die IP-Prüfsumme. Das Paket wird anschließend dem *Communication Device*-Treiber übergeben und versandt. [Vasseur u. Dunkels, 2010, S. 173-174]

A.2.6 Zyklische Verarbeitungsphase

Das Ziel der zyklischen Verarbeitung von μIP ist das Aktualisieren der zeitbasierten Zähler, das Durchführen der Retransmissions und das Entfernen von inaktiven Verbindungen durch *timeout*. Die zyklische Verarbeitung wird typischerweise ein oder zwei Mal in der Sekunde aufgerufen, abhängig von der Konfiguration des Systems, auf dem μIP eingesetzt wird.

Die zyklische Verarbeitung beginnt mit dem Aktualisieren des Status des IP-Fragment-Reassemblybuffers. Falls ein Paket darauf wartet, wieder zusammengefügt zu werden, aktualisiert der Programmcode der zyklischen Verarbeitung den Counter, der das Alter des Paketes festhält. Falls das Paket älter als 30 Sekunden ist, wird es aus dem Reassemblybuffer entfernt.

Als Nächstes analysiert der Programmcode jede aktive TCP-Verbindung und prüft, ob zu übertragende Pakete vorhanden sind. Steht eine Retransmission aus und beinhaltet diese Anwendungsdaten, wird die entsprechende Anwendung durch μIP aufgerufen, damit diese die ursprünglich übertragenen Daten erneut erzeugt. Anwendungen oder eine durch μIP bereitgestellte Anwendungsbibliothek kann das Paket in einem externen Speicher für eine Retransmission vorhalten. In diesem Fall kann die Anwendung das gespeicherte Paket an μIP übergeben. Um Speicherplatz zu sparen, kann die Anwendung das Paket stattdessen neu erzeugen. Nach dem erneuten Bereitstellen der Daten durch die Anwendung erzeugt μIP die nötigen Header und führt die Retransmission aus. μIP verdoppelt den Retransmissiontimer im Rahmen der sogenannten exponentiellen *Back-Off*-Prozedur von TCP.

Der zyklische TCP-Programmcode prüft die Verbindungen zusätzlich auf Inaktivität (timeout) und schließt solche Verbindungen. Verbindungen die zu viele Retransmissions durchgeführt haben ohne eine Bestätigung zu empfangen sind ein Beispiel dafür und werden von μIP verworfen. [Vasseur u. Dunkels, 2010, S. 174]

A.3 Allgemeiner Aufbau von Sensorknoten

A.3.1 Sensoren und Aktoren

Sensorknoten interagieren mit ihrer Umwelt über Sensoren und Aktoren. Die Sensoren tasten physikalische Parameter ihrer Umwelt ab – die Aktoren werden verwendet, um die Umwelt zu beeinflussen. Ein typisches Beispiel für einen Aktor ist eine LED .

Die in einem Sensorknoten verbauten Sensoren und Aktoren können sehr einfach, aber auch sehr komplex sein. Ein Gerät, welches die Temperatur messen soll, benötigt nur einen einfachen Temperatursensor. Ein Sensor, der für die Überwachung eingesetzt werden soll oder für die Detektion von Personen, die z.B. über einen Zaun klettern, benötigen hingegen eine Vielzahl von Sensoren. Das können Ultraschallgeräte oder Kameras sein.

In der Regel besitzen Sensoren allerdings eine einfache Form und Funktion. Durch die limitierten Ressourcen des Mikrokontrollers werden hier Grenzen gesetzt. [Vasseur u. Dunkels, 2010, S. 123]

A.3.2 Die Spannungsversorgung

Wie jedes elektronische Gerät benötigt der Sensorknoten eine Spannungsquelle. Üblicherweise kommt hier eine Batterie zum Einsatz, es existieren jedoch auch alternative Konzepte wie Solarzellen, Piezoelektrizität (das Erzeugen einer elektrischen Spannung an Festkörpern, wenn sie elastisch verformt werden) oder durch Funkwellen übertragene Energie. Diese Form der Spannungsversorgung ist jedoch nicht die Regel und die „Smart Objects“ werden in diesem Fall nahe an der Energiequelle eingesetzt.

Die im Regelfall eingesetzten Batterien basieren auf unterschiedlichen Technologien und besitzen viele Formen und Größen. Der am häufigsten anzutreffende Batterietyp ist derzeit die Lithiumzelle. Durch den Einsatz von Hardwarekomponenten mit niedrigem Energiebedarf und intelligenter Software für das Energiemanagement kann ein Sensorknoten, der mit einer Lithiumzelle bestückt ist, einige Jahre arbeiten.

Wiederaufladbare Batterien, wie sie in Mobiltelefonen oder Notebooks eingesetzt werden, sind für „Smart Objects“ weniger geeignet. Im Gegensatz zu Mobiltelefonen oder Notebooks, die den Menschen als Hilfsmittel oder Werkzeug dienen, sollen Sensorknoten ohne menschliche Kontrolle bzw. Aufsicht arbeiten. Möglicherweise befinden sie sich an schwer zugänglichen Orten oder sind Teil größerer Geräte. Oft werden Sensorknoten, deren Spannungsquelle erschöpft ist, durch neuere Modelle ersetzt. Das spricht gegen den Einsatz wiederaufladbarer Batterien. Trotzdem ist ein Einsatz wiederaufladbarer Batterien in einem Sensorknoten denkbar, wenn der Wiederaufladungsprozess der Batterie ohne menschliches Eingreifen durchgeführt werden kann.

Auch wenn Einwegbatterien eine kostengünstige und praktikable Lösung sind, wirft ihr Einsatz auch Probleme auf. Sie sind schwer zu recyceln und tragen somit zur Umweltverschmutzung bei. Bei größeren und langfristigen Projekten, bei denen Sensorknoten zum Einsatz kommen, ist der Austausch „aufgebrauchter“ Geräte mit enormen Kosten verbunden. Darüber hinaus können Batterien aufgrund ungünstiger Einsatzbedingungen bei Feuchtigkeit oder durch das Auslaufen der Batterieflüssigkeit frühzeitig ausfallen. Aufgrund dieser Tatsachen wird an alternativen Möglichkeiten für die Spannungsversorgung von „Smart Objects“ geforscht.

Hier kommt eine Technik zum Einsatz, die als sogenanntes „Energy Scavenging“ bezeichnet wird. Hierbei wird es elektronischen Bauteilen ermöglicht ihren Strom aus der der sie umgebenden Umwelt zu gewinnen.

Der bekannteste Vertreter dieser Form der Energiegewinnung ist die Solarzelle, ein weiterer die Piezoelektrizität, bei der Bewegungen in elektrische Energie umgewandelt und für das Betreiben der Sensorknoten verwendet werden. Ein Beispiel hierfür sind die intelligenten Lichtschalter des Unternehmens EnOcean, die die gesamte benötigte Energie durch das Drücken des Schalters gewinnen.

Tabelle A.7: Verschiedene Spannungsversorgungen für Sensorknoten, Inhalt übernommen von [Vasseur u. Dunkels, 2010, S. 125]

Energiequelle	**typischer maximaler Strom (mA)**	**typische Ladung (mAh)**
CR2032 Knopfzelle	20	200
AA Alkali Batterie	20	3000
Solarzelle	40	unbegrenzt
HF Energie	25	unbegrenzt

Die in Funkwellen enthaltene Energie kann ebenso als Spannungsquelle genutzt werden. Ein sehr bekanntes Beispiel hierfür sind die sogenannten Radio Frequency Identification (RFID)-Tags, welche die in Funkwellen vorhandene Energie verwenden, um einen Funktransceiver für eine kurze Zeit mit Spannung zu versorgen. Ein sich in der Nähe befindendes RFID-Lesegerät sendet dabei die Funkwellen aus. Diese Art der Energieversorgung ist auch auf Sensorknoten anwendbar. Die Plattform WISP von Intel nutzt bspw. die Energie von sich in der Nähe befindenden RFID-Lesegeräten für die Spannungsversorgung des Sensorknotens. In der Tabelle A.7 werden verschiedene Energiequellen gegenübergestellt und miteinander verglichen. [Vasseur u. Dunkels, 2010, S. 123-125]

A.4 Anwendungsszenarien unter dem Standard IEEE 802.15.4

Der Standard IEEE 802.15.4 wurde für den Einsatz in einer Vielzahl von Anwendungsfällen entwickelt, bei denen eine einfache drahtlose Kommunikation über kurze Entfernungen mit begrenzter Leistung gewünscht und keinen besonderen Anforderungen an den Datendurchsatz gestellt werden. Diese Einsatzmöglichkeiten können in folgende Klassen eingeteilt werden:

- **Stick-On Sensor:** Dieser Anwendungsfall basiert auf drahtlosen Sensoren mit batteriegespeisten Transceivern, die völlig unabhängig arbeiten können. Der Fokus dieser Anwendung liegt auf der Überwachung und der Ferndiagnose.
- **Virtuelle Leitung:** Dabei handelt es sich um Überwachungs- und Steuerungsanwendungen, die nur über eine drahtlose Kommunikation realisiert werden können. Diese Technik kann an Orten eingesetzt werden, an denen das Verlegen einer Leitung zwecks drahtgebundener Kommunikation nicht möglich ist. Ein Beispiel hierfür wäre die Überwachung des Reifendrucks eines Fahrzeugs.
- **Drahtloser Knoten/Hub:** In diesem Einsatzfall wird einem drahtgebundenen Netzwerk eine drahtlose zentrale Bridge hinzugefügt. Dieser drahtlose Hub dient als Gateway zwischen dem drahtgebundenen Netzwerk und dem drahtlosen LR-WPAN. In vielen Fällen wird das drahtlose Hub-Netzwerk mit zwei Transceivern aufgebaut. Einer der Transceiver fungiert als Hub – bei dem anderen handelt es sich um ein LR-WPAN-Gerät bspw. in Gestalt eines Personal Digital Assistant (PDA).
- **Leitungsersatz:** Hier wird versucht, das Arbeiten mit Sensoren oder Peripheriegeräten durch das Entfernen der Kommunikationsleitungen und eine drahtlose Anbindung angenehmer zu gestalten. Für einige dieser Einsatzfälle wird bereits eine weitere WPAN-Technologie eingesetzt – der Standard IEEE 802.15.1 (Bluetooth). Der Standard 802.15.4 kann hier eine energiesparende und kostengünstige Alternative bieten. Der Leitungsersatz unterscheidet sich gegenüber dem Stick-On Sensor dadurch, dass eine permanent vorhandene Energiequelle oder wiederaufladbare Batterien eingesetzt werden.

Durch den Einsatz des Standards IEEE 802.15.4 auf Mikrokontrollerplattformen können eine Vielzahl von Anwendungen realisiert werden, von denen eine Auswahl in den folgenden Abschnitten näher untersucht wird. [Gutierrez u. a., 2011, S. 15]

A.4.1 Drahtlose Überwachungs- und Steuerungssensoren im industriellen und kommerziellen Bereich

Anwendungen, die den Standard IEEE 802.15.4 verwenden, sind größtenteils auf Überwachungsanlagen mit unkritischen Daten beschränkt, bei denen längere Laufzeiten vertretbar sind. Solche in der Industrie eingesetzten Überwachungsgeräte benötigen im Allgemeinen keinen hohen Datendurchsatz bzw. regelmäßiges Aktualisieren der Messwerte. Stattdessen wird der Schwerpunkt auf geringen Energieverbrauch und maximale Lebenszeit der batteriegespeisten Geräte gelegt.

Ein Einsatz in der Prozesssteuerung ist eher die Ausnahme. Auch wenn ein hoher Datendurchsatz eine untergeordnete Rolle spielt, sind feste Laufzeiten und eine gewisse Ausfallsicherheit von hoher Wichtigkeit. Für den speziellen Anwendungsfall der Prozesssteuerung fördert bzw. unterstützt die HART Communication Foundation den 802.15.4-basierten WirelessHART-Standard. Dieser Standard ist international auch als IEC 62591 bekannt.

Einen typischen Einsatzfall in der industriellen Automatisierungstechnik stellen drahtlose Zugangspunkte, die als Gateways für drahtgebundene Industrieprotokolle wie bspw. HART®, Fieldbus® oder PROFIBUS® fungieren, dar. Über diese Zugangspunkte ist eine Überwachung und die

Konfiguration der Parameter der Geräte eines drahtgebundenen Netzwerkes über einen PDA (siehe drahtloser Knoten) möglich. Alternativ kann ein Gerät über eine drahtlose Verbindung (siehe Leitungsersatz) an das Netzwerk angeschlossen werden. [Gutierrez u. a., 2011, S. 17]

A.4.2 Intelligente Energiesteuerung im Heimbereich

Im Rahmen der vorhergesagten Verdoppelung bis Verdreifachung des Energieverbrauchs bis zum Jahre 2050 hat das Energiemanagement von der Erzeugung bis hin zum Endverbraucher einen hohen Stellenwert in der Diskussion um Nachhaltigkeit eingenommen. Die Bezeichnung „intelligentes Netz“ steht für ein Konzept, das eine digitale Aufrüstung der elektrischen Übertagungswege und Verteilnetze beinhaltet. Man verfolgt hierbei das Ziel, ein besseres Netzmanagement zu Spitzenlastzeiten zu realisieren, den über regenerative Energien erzeugten Strom zu integrieren und den Verbrauchern mehr Kontrolle über den eigenen Energieverbrauch zu geben.

Zunächst stellte man sich das Lastmanagement so vor, dass es durch elektrische Hilfsmittel realisiert wird, die den Energieverbrauch beeinflussen. Eine Möglichkeit, größere Einrichtungen wie Heizung, Lüftung und Klimatechnik aus der Ferne zu steuern, wurde auch in Betracht gezogen.

Eine andere Technik zur Laststeuerung basiert auf dynamischer Preisgestaltung. Hier werden die Anlagen der Kunden in die Lage versetzt, den Strom zu Tageszeiten zu nutzen, wo ein niedriger Preis pro Kilowattstunde veranschlagt wird. Das ist in der Regel die Uhrzeit, zu der das Netz einen geringen Auslastungsgrad aufweist.

In der Vergangenheit war es eine große Herausforderung, die Vision des „intelligenten Netzes“ im Endkundensegment, speziell im Heimbereich umzusetzen. Häuser besitzen in der Regel keine standardisierte Infrastruktur, die einen Datenaustausch zwecks Realisierung der zuvor beschriebenen Ideen zulässt. Dieses Problem kann durch den Einsatz 802.15.4-basierter Netzwerke gemildert werden. Ein kostengünstiges Home Area Network (HAN) ist im Heimbereich leicht nachzurüsten und ermöglicht die benötigten Steuerungs- und Überwachungsfunktionen der Anlagen für ein erfolgreiches Energiemanagement. Die Steuerung kann hier durch den Endverbraucher oder eine entsprechende Einrichtung beim Energieversorger erfolgen. [Gutierrez u. a., 2011, S. 18-19]

A.4.3 Vernetzung und Automatisierung im Heimbereich

Der Endverbraucher- und Heimautomatisierungsmarkt bietet aufgrund seiner Größe ein gewaltiges Potential. LR-WPAN-Geräte basierend auf dem Standard IEEE 802.15.4 können Kabelanlagen im Heimbereich mit geringem Kostenaufwand ersetzen. Mögliche Einsatzgebiete wären hier die Endverbraucherelektronik, Peripheriegeräte für Personalcomputer, interaktive Spiele aber auch die Sicherheit im Heimbereich, die Lichtsteuerung und Klimaanlagen. Für die meisten dieser Anwendungsbeispiele existiert eine Industriegruppe, die den Einsatz einer kostengünstigen drahtlosen Lösung befürwortet. Beispiele für solche Gruppen sind: die Consumer Electronics Association (CEA), die Homeplug Allianz und die ZigBee Allianz. Von besonderem Interesse ist eine Studie, die von der ZigBee Allianz durchgeführt wurde. Hier wurden der Datendurchsatz und die Nachrichtenlaufzeit der Anwendungen im Bereich der Heimautomatisierung als akzeptable untere Grenze bzw. obere Grenze für LR-WPANs festgelegt. [Gutierrez u. a., 2011, S. 19]

Elektronik im Endverbrauchersegment

Zur Elektronik im Endkundensegment gehören Radioempfänger, Fernsehgeräte, Videorekorder, CD- und DVD-Player, Fernbedienungen und Haushaltsgeräte im Allgemeinen. Die Möglichkeit, eine Vielzahl von Geräten und Diensten mit einem einheitlichen Kontrollmechanismus steuern zu können, schafft viele Anwendungsfälle für den Standard IEEE 802.15.4

Mit der Annäherung des PC- und Endverbraucherelektronikmarktes ermöglicht es der Standard IEEE 802.15.4, Geräte aus beiden Segmenten mit einem vertrauten und intuitiv bedienbaren Gerät zu steuern – der Fernbedienung. Nun ist es nicht nur möglich, die Fernbedienung einer Gruppe von Geräten zuzuordnen, sondern auch unterschiedlichen Benutzern. Darüber hinaus kann so die Anzahl an Fernbedienungen in einem Haushalt reduziert werden. Die drahtlose Verbindung zwischen der Fernbedienung und den Multimedia-Geräten könnte für die Konfiguration der Lautstärke, des Equalizers und weiteren Einstellungen eines Homeentertainmentcenters verwendet werden. Aber auch Systeme wie das Thermostat und die Sicherheitsanlage könnten so konfiguriert werden. [Gutierrez u. a., 2011, S. 20]

Peripheriegeräte von Personalcomputern

Peripheriegeräte von Computersystemen wie drahtlose Mäuse, Tastaturen, Joysticks und PDAs stellen ein großes Einsatzpotential für LR-WPANs dar. Darüber hinaus versuchen die Computerhersteller und Softwareentwickler, die Schnittstellen eines PC's anwenderfreundlicher zu gestalten. Mit der Einführung von Fernbedienungen und PDAs wird diese Art der PC-Steuerung immer verbreiteter. Der Standard 802.15.4 ist besonders geeignet, um die energiesparenden und kostengünstigen Lösungen zu bieten, die dieser Anwendungsfall benötigt. [Gutierrez u. a., 2011, S. 20-21]

Automatisierung im Heimbereich

Der Heimautomatisierungssektor, der sich aus Heizung, Lüftung, Klimatechnik, Sicherheitstechnik, Beleuchtung und Steuerungsmechanismen für Fenster, Türen und Schlössern zusammensetzt, bietet enormes Potential für drahtlose Innovationen. Oft ist das Thermostat eines Hauses an Orten zu finden, wo man sich relativ selten aufhält (Korridore, Eingangsbereich etc.). Das führt oft dazu, dass sich die abgelesenen Temperaturwerte nicht mit der Temperatur decken, wie sie von den Bewohnern wahrgenommen wird. Demzufolge ist die die Steuerung der Umgebungstemperatur im Heimbereich oft uneffektiv.

Drahtlose Thermostate können im ganzen Haus verteilt und mit drahtlosen Lüftersteuerungen (Bereichssteuerung) des Heating, Ventilation and Air Conditioning (HVAC)-Systems verbunden werden, um die Temperatur in den Räumen individuell anzupassen. Die Rollläden könnten heruntergelassen werden, wenn der Fernseher während des Tages angeschaltet wird. Wanduhren o. ä. könnten mit einer Referenzuhr abgeglichen werden, so dass sie sich nach einem Stromausfall automatisch stellen würden. Die Sicherheitsanlage des Hauses könnte so einfach sein wie die sicherheitsorientierte Kfz-Zutrittssteuerung (Remote Keyless Entry) – ein Knopfdruck könnte alle Türen und Fenster verschließen. Drahtlose Rauchdetektoren und Glasbruchmelder könnten mit an das Haussicherheitssystem angeschlossen werden. Würden alle diese Anwendungsfälle mit dem Standard IEEE 802.15.4 realisiert werden, könnte eine einfache Fernbedienung für das Steuern dieser Funktionen, zusätzlich zu den elektronischen Endgeräten, verwendet werden.

Einige Beleuchtungsmittelhersteller haben angekündigt, IEEE 802.15.4-Transceiver in Glühbirnen bzw. den Vorschaltgeräten zu integrieren. Dadurch wäre es möglich, Leitungen einzusparen und die Leuchtmittel durch drahtlose Schalter, an einem beliebigen Ort platziert, ein- bzw. auszuschalten. [Gutierrez u. a., 2011, S. 21]

Sicherheit im Heimbereich

Vergleichbar mit der Heimautomatisierung können Sicherheitssensoren im gesamten Heimbereich verteilt angebracht werden. Typischerweise zählen Bewegungsmelder, Tür- und Fensterkontakte

und Leckmeldesysteme dazu. Wieder besteht der Wert des Einsatzes von LR-WPAN-Transceivern darin, dass sie im Gegensatz zu verdrahteten Sensoren leicht installiert werden können. Wanddurchbrüche und das Verlegen von Leitungen sind nicht notwendig.

Drahtlose Sensoren im Bereich der Heimsicherheit müssen sich zwei Herausforderungen stellen. Die Erste steht im Zusammenhang mit dem Energieverbrauch des Sensors. Für gewöhnlich werden solche Sensoren nicht an das Stromnetz angeschlossen, da sie in der Regel an Fenstern und Türen angebracht werden. Der Standard IEEE 802.15.4 begegnet dieser Herausforderung mit einem einfachen und dennoch effizienten Protokollstack. Darüber hinaus stellt die MAC-Definition den höheren Protokollschichten eine Schnittstelle bereit, über die sie die Funkschnittstelle aktivieren bzw. deaktivieren können.

Die zweite Herausforderung bezieht sich auf die Reichweite des drahtlosen Netzwerkes aufgrund der Ausbreitung des Funksignals. Die Sendeleistung der Transceiver des Standards IEEE 802.15.4 unterliegt den Beschränkungen der Regulierungsbehörde, wird aber auch durch die Notwendigkeit beschränkt, Energie zu sparen. Der Standard bietet hier die Möglichkeit, ein Peer-to–Peer-Protokoll einzusetzen (siehe 2.1.3), um ein Multihop-Netzwerk zu realisieren. Dadurch kann die Reichweite der drahtlosen Kommunikation erhöht werden. [Gutierrez u. a., 2011, S. 22]

Einsatz im Gesundheitswesen

Der Bereich Gesundheitswesen umfasst den Einsatz von Sensoren, Überwachungsanlagen und Diagnosegeräten. Der Bereich wird durch die Art der medizinischen Telemetrie unterteilt. Persönliche Gesundheitspflege beinhaltet Geräte wie Pedometer (Schrittzähler) und Pulsmesser, wie sie von Joggern und anderen Athleten verwendet werden. Diese Geräte erzeugen Daten, die zu einem Anzeigegerät übertragen werden müssen. Oft ist eine drahtlose Verbindung die einzige praktikable Lösung.

Ein weiterer Anwendungsfall ist das Überwachen des Gesundheitszustandes. Das tägliche Gewicht einer Person, seine Temperatur und andere Messwerte können über eine drahtlose Verbindung zwischen dem eigentlichen Messgerät und dem aufzeichnenden Computer bzw. PDA übertragen werden. Sind diese Daten im Internet verfügbar, kann professionelles Personal darauf zugreifen und den Gesundheitszustand aus der Ferne überwachen.

In den letzten Jahren ist die 802.15.4-Technologie ein Schlüsselelement der Altenpflege geworden. Diese Anwendung überwacht die täglichen Aktivitäten der Senioren aus der Ferne, um ihren Gesundheitszustand abschätzen zu können. Dadurch ist es älteren Hausbesitzern möglich, den Komfort des eigenen Heims zu genießen, während bei einem gesundheitlichen Notfall der Notdienst informiert wird. Unter der Vielzahl von eingesetzten Sensoren, ist der sogenannte „Panikknopf“ der bekannteste. Über diesen können die Senioren ein Notsignal absetzen, falls sie plötzlich handlungsunfähig werden sollten. [Gutierrez u. a., 2011, S. 22-23]

Einsatz im Spielesektor

Ein LR-WPAN bietet viele Einsatzmöglichkeiten im Spielesektor, insbesondere wenn die Kommunikation unter den Spielzeugen oder zwischen Spielzeug und PC stattfindet. Spielzeuge sind sehr kostenempfindlich. Der begrenzende Faktor einer rechenintensive Aufgabe wie die Spracherkennung oder Sprachsynthese in einem Spielzeug ist oft nicht die technische Realisierbarkeit, sondern dass sie mit möglichst geringen Hardwarekosten umgesetzt werden müssen. Durch das Nutzen einer drahtlosen Verbindung zwischen dem Spielzeug und einem sich in der Nähe befindenden PC können diese Kosten reduziert werden. In dem Spielzeug müssen nur die drahtlose

Verbindung und die benötigten Sensoren bzw. Aktoren realisiert werden. Solche auf den PC ausgeweiteten Spielesysteme bieten ein ausgeklügeltes Verhalten (bspw. bewegliche Roboter), nur begrenzt durch die Kapazitäten des Computersystems und der drahtlosen Verbindung.

Eine andere Möglichkeit innerhalb des Spielesektors ist das drahtlose Zusammenspielen zwischen Personen oder Personengruppen. Darüber hinaus könnten die Spielzeuge oder Spiele automatisch Updates durchführen, wenn sie sich innerhalb der Reichweite eines Computers befinden. Anschließen könnten sie in einem neuen Spielmodus oder mit einem neuen Feature weiterarbeiten – auch außerhalb der Reichweite des zuvor genutzten Computers.

Die Hersteller drahtloser Spielzeuge, die den Standard IEEE 802.15.4 einsetzen, könnten diese Spieleplattform zusätzlich zu neuen Spielen anbieten. Die Kosten und der Stromverbrauch eines drahtlosen Zusatzmodules wären gering genug - was es für die Anwender attraktiv machen würde.

Die Tabelle A.8 zeigt die Anforderungen der Anwendungen in den Bereichen Heimautomatisierung und Vernetzung. Die typische Datenrate der meisten Anwendungen liegt deutlich unter der maximal möglichen. [Gutierrez u. a., 2011, S. 23-24]

Tabelle A.8: Anforderungen an die Vernetzung und Automatisierung im Heimbereich , Inhalt übernommen von [Gutierrez u. a., 2011, S. 24]

Sector	**Maximal erforderliche Datenrate**	**Maximal zulässige Latenz**
Konsumerelektronik	3 kb/s	16,7 ms
PC-Peripherie	115,2 kb/s	16,7 ms
Heimautomatisierung	10 kb/s	100 ms
Gesundheitspflege	10 kb/s	30 ms
Spiele	115,2 kb/s	16,7 - 100 ms

A.4.4 Fahrzeugsensoren

Drahtlose Kommunikation wird heutzutage in Automobilen verwendet, mit dem Ziel, den Fahrkomfort und die Anzahl der angeboten Features zu erhöhen. Der erste Anwendungsfall für die drahtlose Technologie in einem Automobil war die sicherheitsorientierte Kfz-Zutrittssteuerung (Remote Keyless Entry). Ihm folgte das heutzutage noch aktuelle Anwendungsbeispiel, Hier werden die Kommunikationsleitungen der Telematikanwendungen im Kfz-Bereich ersetzt. Bei der Telematik handelt es sich um eine Informationsverknüpfung mehrere Datenverarbeitungssysteme.

Zurzeit wird in diesem Bereich hauptsächlich die Bluetooth-Technologie (IEEE 802.15.1) eingesetzt. Dabei wird der Fokus auf Telefonie und die Verknüpfung solcher Geräte mit der Boardelektronik gelegt.

Abgesehen von den Bereichen Sicherheit und Komfort spielen die Kosten für die verschiedenen Anwendungen bei Automobilen eine wichtige Rolle. Heutzutage sind Automobile mit einer gewaltigen Anzahl von Sensoren und Aktoren, verteilt über das gesamte Fahrzeug, ausgestattet. Durch den Einsatz dieser Geräte sind die Kosten für Installation, Fehlerdiagnose und Wartung enorm angestiegen. Darüber hinaus führt das Mehrgewicht zu einem spürbaren Mehrverbrauch an Kraftstoff.

Der optionale Einsatz von drahtlosen Technologien ermöglicht Flexibilität bei der Installation und bietet eine ausgereifte Alternative gegenüber drahtgebundenen Verbindungen. Eine Herausforde-

rung für dieses Anwendungsbeispiel ist es, die Kosten gering zu halten. Ursache hierfür sind die extremen Umweltbedingungen wie Witterung und Temperatur, unter denen diesen Geräte arbeiten sollen.

Ein typischer Anwendungsfall aus dem Bereich virtuelle Leitung wäre die Luftdrucküberwachung von Reifen. Dieses System ist aus vier Drucksensoren aufgebaut, die an jedem Reifen angebracht werden. Eine zentrale Station empfängt die Daten und wertet sie aus. Da die Sensoren auf den Reifen angebracht werden, ist der Einsatz von Stromversorgungskabeln und Kommunikationsleitungen nicht möglich. Die Sensoren werden daher batteriegespeist. Da es nicht praktikabel wäre, diese Batterien mit jedem Reifenwechsel zu ersetzen, sollten sie mindestens drei, vorzugsweise fünf Jahre im Einsatz sein. Daraus ergeben sich besondere Anforderungen an den Energieverbrauch der Sensoren. Ein ausgeklügeltes Energiemanagement schafft hierbei gute Voraussetzungen.

Die zu übertragenden Daten umfassen nur wenige Bits und die Kommunikation findet nur alle ein bis zehn Minuten statt. Nur bei plötzlichem Druckverlust wird die zentrale Station sofort benachrichtigt. Hier spielt der Energieverbrauch eine untergeordnete Rolle, da in den meisten Fällen der Reifen ohnehin ersetzt werden muss.

Die extremen Einsatzbedingungen und die metallische Struktur eines Automobils erschweren den Entwurf solcher auf Hochfrequenzübertragung basierender Lösungen. Der Einsatz der Zwei-Wege-Kommunikation (hier werden Informationen in beide Richtungen übertragen) hilft enorm, die Ausfallsicherheit solcher drahtlosen Verbindungen zu erhöhen. [Gutierrez u. a., 2011, S. 24-25]

A.4.5 Einsatz in der Landwirtschaft

Ein weiteres Anwendungsbeispiel für LR-WPANs ist eine Art der Bewirtschaftung von landwirtschaftlichen Nutzflächen, die als Precision Farming (deutsch Präzisionsackerbau) bezeichnet wird. Hierbei handelt es sich um eine umweltfreundliche Lösung, die Ertrag und Qualität unter Einsatz geringerer Kosten erhöht. Durch den herkömmlichen Einsatz der landwirtschaftlichen Maschinen und durch das schwer vorhersagbare Wetter sind der Ertrag und die Qualität einer Ernte ungewiss. Mit dem neuen Ansatz des Precision Farming wird die Landwirtschaft mehr informations- und softwareorientiert. Erreicht wird dies durch den Einsatz automatisierter, vernetzter und ferngesteuerter intelligenter Maschinen. Diese Anwendung setzt riesige vermaschte Netzwerke voraus, die aus tausenden von LR-WPAN-Geräten bestehen, die mit Sensoren verbunden sind. Diese Sensoren sammeln Informationen wie bspw. Bodenfeuchtigkeit, Stickstoffgehalt und pH-Wert. Wettersensoren, die Temperatur, Luftfeuchtigkeit, Niederschlag und Luftdruck messen, liefern dem Bauern weitere nützliche Informationen. Jeder Sensor überträgt die gewonnenen Daten zu dem für ihn zuständigen LR-WPAN-Gerät, welches diese Informationen an eine zentrale Station weiterleitet.

Um die gewonnenen Daten sinnvoll verarbeiten zu können, ist der Einsatz von Technologien erforderlich, die die genaue Lage der Sensoren in dem zu bewirtschaftenden Gebiet bestimmen. Die Summe dieser Informationen alarmiert den Bauern frühzeitig über mögliche Probleme, so dass er eingreifen und einen möglichen Ernteausfall abwenden kann.

Die Anforderungen an ein LR-WPAN sind sehr gering, da nur wenige Bits pro Tag an das Gerät übertragen werden und es sich um eine asynchrone Datenübertragung mit unkritischem Verhalten gegenüber der Latenz der Nachrichten handelt. Diese Faktoren begünstigen auch eine lange Lebenszeit der eingesetzten Batterien. Für diese Anwendung ist der Einsatz einer vermaschte Topologie erforderlich, demzufolge werden einige der Stationen für das Weiterleiten der Nachrichten eingesetzt. Voraussetzung ist weiterhin, dass es sich um ein selbstkonfigurierendes Netzwerk

handelt, da eine manuelle Konfiguration von Netzwerken dieser Dimension nicht praktikabel ist. [Gutierrez u. a., 2011, S. 25-26]

A.4.6 Weitere Anwendungsbeispiele

Ein besonderer Anwendungsfall aus dem Bereich des Endkundenmarktes ist ein Netzwerk innerhalb eines Klassenzimmers und der Einsatz eines Taschenrechners. Der Computer des Lehrers oder der PAN-Koordinator könnte Aufgaben und mathematische Probleme an jeden grafischen Taschenrechner oder Netzwerkgerät senden. Nach Fertigstellung könnten die Schüler ihre Lösungen auf den Computer des Lehrers hochladen. Dieses Netzwerk müsste nur eine kleine Anzahl von Geräten unterstützen und eine Peer-to-Peer-Kommunikation verhindern, um das Austauschen von Lösungen zu verhindern. Die Nutzdaten mit einer Größe von ungefähr 100 bis 500 Bytes müsste nur einige Male pro Schüler und innerhalb einer Stunde übertragen werden. Unter diesen Voraussetzungen würde die Lebensdauer der Batterien ein ganzes Semester betragen.

Ein weiteres, wichtiges Anwendungsbeispiel aus dem Bereich drahtloser Hub ist die Datenerfassung und Konfiguration aus der Ferne. Dadurch wären ein besserer Schutz vor Gefahren und eine angenehmere Arbeitsweise realisierbar. Durch den Einsatz kostengünstiger LR-WPAN-Stationen könnten mit entsprechend aufgerüsteten Geräten Informationen über eine drahtlose Verbindung unter Zuhilfenahme eines PDAs ausgetauscht werden. Eine Konfiguration oder ein Softwareupdate dieser Geräte wäre ebenso denkbar. [Gutierrez u. a., 2011, S. 26]

Anhang B

Anhang: CD mit Quellcodes, Messergebnissen und Abschlussarbeit (PDF)