
BACHELORARBEIT

TOWARDS A DEPENDABLE IPv6-OVER-BLE BORDER ROUTER BASED ON LINUX

durchgeführt am
Institut für Technische Informatik
Graz University of Technology, Austria

von
Christoph Schmidt

Betreuer:
Dipl.-Ing. Michael Spörk, BSc

Begutachter:
Ass. Prof. Dr. Carlo Alberto Boano

Graz, 24. November 2019

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Bachelorarbeit identisch.

Graz, am

(Unterschrift)

Abstract

Today, *Wi-Fi*, *Bluetooth* and similar technologies have been emerged to a standard of many devices like smartphones or PCs. The *Internet of Things* (IoT) describes a network in which not only smartphones and PCs can communicate. In fact, a wide variety of devices should get connected with each other, be it smart light bulbs or kitchen devices.

The fields of application for the *IoT* are broad and range from *Smart Home* applications right up to *Smart Health*.

The idea is to connect all these smart devices using an energy-efficient connection like *Bluetooth Low Energy* (BLE) by just one *BLE Border Router* while each device gets its own IP-Address. Although BLE is implemented in nearly every device and there has been a lot of talk about the Internet of Things in the last couple of years the actual adoption of this technology in consumer products has been relatively slow. Yet, there are only few attempts to do this on a user-friendly level. The contribution of this thesis is to provide an easy way of setting up such a Border Router, even for less sophisticated users, which can help to distribute the IPv6-over-BLE technology to mass. For this reason, the implementation of a stable and configurable *IPv6-over-BLE Border Router* may be an interesting use case.

Kurzfassung

Funktechnologien wie *WLAN*, *Bluetooth* und Co. gehören zur Grundausstattung zahlreicher Geräte wie Smartphones, PCs und dergleichen.

Doch längst sind das nicht die einzigen Gegenstände: Im *Internet der Dinge*, das ein Netzwerk beschreibt, in welchem unterschiedlichste Geräte untereinander kommunizieren können, sind bereits Leuchtmittel ebenso integriert wie Küchengeräte. Dabei sind die Anwendungsfelder umfassend und breit gefächert. Angefangen bei *Smart Home*-Anwendungen und *Home Automation* bis hin zur *Smart Health*.

Die Idee ist es, alle *Smart Devices* über die energiesparende Verbindung *Bluetooth LE* (BLE) mit nur einem Endgerät oder Gateway zu verbinden, wobei jedes dieser Geräte eine eigene IP-Adresse erhält. Obwohl BLE, der energiesparende Bruder von *Bluetooth*, bereits weite Verbreitung gefunden hat, sind die Bestrebungen zur Übertragung von IPv6-Paketen auf einen eher kleinen Anwenderbereich beschränkt. Bis jetzt gibt es wenige Versuche dies auf einer benutzerfreundlichen Ebene zu tun.

Eine einfache Applikation zum Aufsetzen eines benutzerfreundlichen *IPv6-over-BLE* Border Routers - auch für weniger versierte Benutzer verwendbar - könnte dazu beitragen, dass die Technologie *IPv6-over-BLE* größere Verbreitung findet. Aus diesem Grund stellt die Implementation eines stabilen und konfigurierbaren *IPv6-over-BLE* Border Routers einen interessanten Anwendungsfall dar.

Danksagung

An diese Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Anfertigung meiner Bachelorarbeit unterstützt und motiviert haben.

Zuallererst möchte ich mich bei Michael Spörk für seine hervorragende Betreuung meiner Arbeit bedanken, der auch den Anstoß zu diesem Thema lieferte. Michael gab mir die Möglichkeit, diese am *Institut für Technische Informatik* durchzuführen, unterstützte mich während der Umsetzung und Anfertigung maßgeblich und stand mit Geduld, Hilfsbereitschaft und nützlichen Vorschlägen stets zur Verfügung.

Auch gebührt meinen Eltern großer Dank, die mich mit ihrem Rückhalt und ihrer Unterstützung hierher geführt haben, wo ich heute stehe.

Graz, 24. November 2019

Christoph Schmidt

Inhaltsverzeichnis

1 Einleitung	7
1.1 Motivation	7
1.2 Problembeschreibung	8
1.3 Zielsetzung	9
1.4 Struktur der Arbeit	10
2 Technischer Hintergrund	11
2.1 Bluetooth Classic	11
2.2 Bluetooth Low Energy	12
2.2.1 Bluetooth Protocol Stack	12
2.2.2 Physical Layer	12
2.2.3 Link Layer	13
2.2.4 Host Controller Interface	15
2.3 IPv6 und Bluetooth LE	17
2.3.1 Internet Protokoll Version 6	17
2.3.2 IPv6 over BLE	17
2.4 Anwendungsszenario	20
2.5 Hardware	21
2.5.1 Raspberry Pi 3	21
2.5.2 Partical Xenon	21
2.5.3 Nordic Semiconductor nRF52840 Development Kit	21
2.5.4 Panasonic PAN 1762 Development-Kit (USB Adapter)	22
3 Bluetooth Kernelmodul	23
3.1 6LoWPAN Kernelmodul	23
3.2 Mehrfache Peer-Verbindung	24
3.2.1 Problembeschreibung	24
3.2.2 Lösungsansatz	25
3.3 Ansprechen mehrerer Host Controller	26
3.3.1 Wechseln der Host Controller	26
3.3.2 Implementation	26
3.3.3 Anwendung des <code>select</code> -Befehls	27
4 BLEd Daemon	29
4.1 Software-Lizenz und Code-Struktur	29
4.2 Voraussetzungen und Abhängigkeiten	29
4.3 Verwendung bestehender Software und Drittbibliotheken	30
4.4 Kompilation und Installation	31
4.5 Softwarearchitektur	31
4.5.1 „BLEd“ Daemon	32
4.5.2 Allgemeiner Ablauf	32
4.5.3 BLEHelper	34
4.5.4 Lesen der Konfiguration	38
4.5.5 HCICore	40
4.5.6 InteractionInterface	40
4.5.7 Sonstige Module	43

5	Evaluierung	45
5.1	IPv6 Konnektivität	45
5.1.1	Setup	45
5.1.2	Ergebnis	47
5.2	Auswahl mehrerer Bluetooth LE Dongles	48
5.2.1	Setup	48
5.2.2	Ergebnis	50
5.3	Testen des BLE Deamons	51
5.3.1	Setup	51
5.3.2	Ergebnis	51
5.4	Verhalten bei unterschiedlichen <i>Connection Parametern</i>	53
5.4.1	Setup	53
5.4.2	Ergebnis	54
6	Ergebnis	57
6.1	Weiterführende Arbeit	57

Abbildungsverzeichnis

2.1	Solution Areas (entnommen von [1])	11
2.2	Aufbau des BLE Protocol Stacks (adaptiert von [2, S. 144])	12
2.3	Link Layer Packet Format (adaptiert aus [3, S. 2691])	14
2.4	Beispiel eines Link Layer Packets im Advertising-Modus (inkl. Aufbau)	15
2.5	Aufbau eines HCI Event Packets (adaptiert aus [2, S. 199])	16
2.6	Exemplarisches HCI-Packet	16
2.7	BLE Protocol Stack mit IPv6 Unterstützung (adaptiert aus [4])	17
2.8	Topologie eines Sternenzwerts	19
2.9	Anwendungsszenario mit einem Sensor und einem Border-Router (adaptiert von [2, S. 373])	20
2.10	Raspberry Pi 3 (Bild aus [5])	21
2.11	Partical Xenon (Bild entnommen aus [6])	21
2.12	Nordic Semiconductor nRF52840 Development Kit	21
2.13	Panasonic PAN 1762 (Bild entnommen aus [7])	22
3.1	Grafische Darstellung des Verbindungsproblems mehrerer BLE-Geräte	24
4.1	Schema der BLEd-Applikation mit zwei BLEHelper-Instanzen	33
4.2	Flussdiagramm des BLEHelpers	34
4.3	Interaktion zwischen BLEHelper und HCICore	35
4.4	Abfolge der Befehlsabarbeitung am Beispiel des <code>connect</code> -Befehls	37
4.5	Anzeigen verbundener BLE-Nodes	41
4.6	Ändern der <i>Connection Parameter</i>	42
5.1	Schematische Aufbau des Testsetups zum Testen der IPv6 Konnektivität	45
5.2	Antwort beider BLE-Peers auf „pings“	46
5.3	Netzwerktopologie	48
5.4	Schematischer Aufbau des Testsetups	48
5.5	Ermitteln der verfügbaren <i>Host Controller Interfaces</i>	49
5.6	Senden eines „pings“ an den mit <code>hci0</code> verbundenen BLE-Peer	49
5.7	Senden eines „pings“ an den mit <code>hci1</code> verbundenen BLE-Peer	50
5.8	Verbundene BLE-Nodes, mit zwei Netzwerkinterfaces	50
5.9	Response Time zweier BLE-Peers mit gleichen Connection Parameter	54
5.10	Response Time zweier BLE-Peers mit unterschiedlichen Connection Parameter	54
5.11	Response Time zweier BLE-Peers mit gleichen und zueinander verschiedenen Connection Parameter	55

Abkürzungsverzeichnis

6LBR IPv6-over-BLE Border Router.

6LN IPv6-over-BLE Node.

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks.

ATT Attribute Protocol.

BLE Bluetooth Low Energy.

CRC Cyclic Reduncance Check.

DHCP Dynamic Host Configuration Protocol.

GAP Generic Access Profile.

GATT Generic Attribute Profile.

HCI Host Controller Interface.

ICMPv6 Internet Control Message Protocol version 6.

IEEE Institute of Electrical and Electronics Engineers.

IoT Internet of Things.

IP Internet Protocol.

IPSP Internet Protocol Support Profile.

IPSS Internet Protocol Support Service.

IPv4 Internet Protocol Version 4.

IPv6 Internet Protocol version 6.

ISM Industrial, Scientific and Medical Band.

L2CAP Logical Link Control and Adaptation Protocol.

LR-WPAN Low-Rate Wireless Personal Area Networks.

PDU Protocol Data Units.

RAM Random-Access Memory, Arbeitsspeicher.

RCF Request for Comments.

RCF Request for Comments.

RF Radio Frequency.

SIG Bluetooth Special Interest Group.

UART Universal Asynchronous Receiver Transmitter.

UDP User Datagram Protocol.

UUID Universally Unique Identifier.

WLAN Wireless Local Area Network.

1

Einleitung

Bereits im Jahre 1991 sprach der US-amerikanische Informatiker Mark Weiser in dem Artikel „The Computer for the 21st Century“ von einem System aus vernetzen Maschinen, welches er unter dem Begriff *Ubiquitous computing* (in etwa: „allgegenwärtiges Rechnen“) zusammenfasste. Solch ein System, so Weiser, wird aus Computern unterschiedlicher Größe bestehen, jeder für einen speziellen Aufgabenbereich bestimmt [8].

Mit der Entwicklung immer kleinerer Geräte, alle mit Drahtlosen Technologien wie RFID, Bluetooth oder 802.11 (WLAN) ausgestattet, hat sich zunehmend der Begriff *Internet of Things* (IoT, Internet der Dinge) entwickelt. So gehören mittlerweile Bluetooth und WLAN längst zur Grundausstattung jedes Smartphones und PCs. Das *Internet der Dinge* beschreibt ein Netzwerk, in welchem alle Geräte untereinander kommunizieren können, wobei jedes dieser Geräte einem bestimmten Zweck zuordenbar ist [9].

Dabei sind die Anwendungsfelder umfassend und breit gefächert, angefangen bei *Smart Home*-Anwendungen und *Home Automation* bis hin zur *Smart Health*.

Auch wenn der, vor allem in den letzten Jahren sehr häufig verwendete und das Computerzeitalter prägende Begriff nicht neu ist und trotz der Tatsache, dass sich auch große Technologieunternehmen wie Apple, Google oder Phillips aktiv an der Entwicklung in diesem Bereich beteiligen, so läuft die Integration doch eher langsam ab. Mehr noch könnte man das System als „Internet of my Things“ [10, vgl. Øvrebekk] bezeichnen. Hersteller verwenden unterschiedliche Technologien und diese „Smart Devices“ benötigen verschiedene Endgeräte, um sich mit dem Internet verbinden zu können. Viele Produkte sind daher auf spezielle *Hubs* angewiesen, um eine gemeinsame Schnittstelle zum Internet zu ermöglichen. Erst durch *Hubs* werden diese Geräte „smart“.

1.1 Motivation

Extrem energiesparende Technologien sind besonders im IoT- und Smartwear-Bereich interessant. Der 2010 eingeführte Standard *Bluetooth Low Energy (BLE)* (bzw. als *Bluetooth LE* bekannt und unter anderem als *Bluetooth Smart* vermarktet) hat dabei aber auch in medizinischen Umgebungen (*Smart Health*), beispielsweise bei Geräten zur Überwachung der Vitalparameter (Blutdruck, Herzfrequenz, etc.), Einzug gefunden [11].

Eine zentrale Anforderung an diese Technologien für diesen Bereich ist der Energieverbrauch, unabhängig davon, ob der Anwendungsbereich in der Consumer-Elektronik liegt, oder in der Medizintechnik.

Diese Eigenschaft - *low power consumption* genannt - also der geringe Verbrauch mit kleinen *duty cycles*¹ und genügend hohe Datenraten, stellt dabei einen wichtigen Faktor dar, um Geräte im Dauerbetrieb nahe am menschlichen Körper nutzen und betreiben zu können [12].

¹ Beschreibt das Verhältnis eines Signals zwischen dem aktiven Zustand und dem inaktivem

Wird von einem solchen Setup gefordert, dass das Betreiben von komplexen Netzwerken möglich ist, wird man vor allem im Bereich der Consumer-Elektronik bei verschiedenen, bereits etablierten Standards fündig. Seitens der Hersteller werden hier jedoch unterschiedliche Ansätze verfolgt. Um ein Beispiel zu nennen: Neben omnipräsenten Technologien wie *WLAN*, setzt der bereits genannte niederländische Hersteller Phillips bei seinen Smart-Home Lampen der *Hue*-Serie auf den *ZigBee Light Link*-Standard [13]. Zigbee (802.15.4) wurde für Netzwerke konzipiert, welche auf geringen Energieverbrauch ausgelegt sind. Gegenüber BLE findet sich Zigbee aber wenig bis gar nicht in Geräten der Endverbraucher wieder. Obwohl es bereits eine relativ breite Auswahl an energiesparenden Optionen gibt, welche für solche Zwecke geeignet scheinen, haben die Technologien gemein, dass eine Vielzahl der Produkte - wie diese Smart-Lampen - an Endgeräte gebunden sind. Um die oftmals proprietären Protokolle zu IP-Paketen verarbeiten zu können, werden Hubs benötigt, welche dann simultan als *Router* oder *Gateway* fungieren.

Warum IPv6 over Bluetooth LE?

Die Idee ist also, alle Geräte über eine energiesparende Verbindung mit nur einem Endgerät oder Gateway zu verbinden, wobei jedes dieser Geräte eine eigene IP-Adresse erhält. Pakete müssen nicht mehr von proprietären Formaten in IP(v4) oder IPv6-Pakete umgewandelt werden.

Mit *IPv6* wird auch das Problem des relativ begrenzten Adressraums von *IPv4* gelöst. Mit 2^{128} theoretisch möglichen Adressen bietet der Nachfolger genügend Platz für solche Geräte.

Zudem wurde BLE um die Möglichkeit ergänzt, ebendiese IP-Pakete zu übertragen [4]. Dies wurde im *Internet Protocol Support Profile (IPSP)* definiert. *IPSP* ist das Bluetooth-Profil unterhalb des *6LoWPAN*-Layers, welches die IPv6-Konnektivität über BLE zur Verfügung stellt (Abschnitt 2.3.2).

Neu ist auch, dass seit Bluetooth 4.0 der Standard auch Punkt-zu-Mehrpunkt-Verbindungen (*Mesh*) [14] unterstützt. Die *Mesh*-Fähigkeit von Bluetooth erlaubt insbesondere im *Smart Home* Bereich große Flexibilität [15].

In herkömmlichen Netzwerken sind meist eigene Geräte, z. B. *Repeater* für das Weitersenden und Verstärken eines Signals zuständig. In *Mesh*-Netzwerken hingegen wird das Signal von den Knoten aufgespannt und an den nächsten, verstärkt weitergereicht. Dadurch wird ein relativ gleich starkes Netz aufgespannt, das eine hohe Ausfallsicherheit bietet und konstante Datenübertragungsraten ermöglicht [16].

Eine einfache Applikation zum Aufsetzen eines benutzerfreundlichen *IPv6-over-BLE* Border Routers - auch für unerfahrene Benutzer verwendbar - könnte dazu beitragen, dass die Technologie *IPv6-over-BLE* größere Verbreitung findet. Aus diesem Grund stellt die Implementation eines stabilen und konfigurierbaren **IPv6-over-BLE Border Router (6LBR)** einen interessanten Anwendungsfall dar.

1.2 Problembeschreibung

Das Bluetooth Protokoll unterstützt bereits seit Version 4.0 Bluetooth Low Energy. Mit dem bereits 2015 eingeführten Standard *RCF 7668* [4] (*IPv6 over BLUETOOTH(R) Low Energy*) wurde eine Möglichkeit definiert, wie IPv6-Pakete über ein BLE Verbindung via *Low-Power Wireless Personal Area Network (6LoWPAN)* transportiert werden können.

Mit zunehmender Vernetzung hat die BLE-Technologie in nahezu jedem kommerziell erhältlichen Produkt (angefangen von Smartphones bis hin zu Küchengeräten und Kühlschränken) Einzug gehalten [17]. Obwohl dementsprechend Möglichkeit geboten wäre, diese Geräte mittels

Bluetooth-Verbindungen direkt mit dem Internet zu verbinden, so gibt es bis jetzt wenige Versuche dies auf einer benutzerfreundlichen Ebene zu tun.

Das Problem ist daher nicht neu, vielmehr gibt es bereits Methoden, das Vorhaben mehr oder weniger schnell zu realisieren. Die aktuell dafür vorgesehene Linux-Implementierung ist jedoch veraltet und nicht zufriedenstellend. Sie bietet nur geringfügige Interaktionsmöglichkeiten, unterstützt nur ein verbundenes Gerät und bietet keine benutzerfreundliche Parametrierung.

Ausgehend von einem *Raspberry Pi 3* mit *Raspbian GNU/Linux 9 (stretch)* in der Kernelversion *4.14.95-v7* verfolgt diese Arbeit das Ziel, die bestehende Bluetooth 6LoWPAN-Entwicklung zu adaptieren und einen quelloffenen, einfach anzuwendenden und zuverlässigen Applikation-Prototypen für den *IPv6-over-BLE* Support auf Linux zu implementieren. Dabei soll es möglich sein, diese als Zusatzapplikation in Form eines Daemons später auf jedem beliebigen Linux-basiertem Betriebssystem nachinstallieren zu können. Das Verhalten soll dabei an jene von Wifi-Routern angelehnt sein; das heißt die Anwendung soll sich bei Betrieb um Verbindungsaufbau und Verwaltung der verbundenen (BLE-)Geräte („BLE-Nodes“ oder 6LN) automatisch und ohne Benutzerinteraktion kümmern.

Es wird ein IPv6-over-BLE Border Router entwickelt, aufbauend auf der im aktuellen Linux-Kernel bereits verfügbaren Implementation des Bluetooth Low Energy Kernelmoduls (`bluetooth_6lowPAN`). Dabei sollen sowohl Daemon-Applikation als auch das verfügbare Kernelmodul in einen betriebsfertigen Zustand gebracht werden. Das Modul selbst befindet sich in der Entwicklung, liegt in Version *0.1* vor und weist dementsprechend noch einige Schwächen auf.

1.3 Zielsetzung

Diese Arbeit befasst sich in erster Linie mit der Entwicklung eines zuverlässigen IPv6-over-BLE Supports für Linux-basierte Betriebssysteme. Dieser Prozess ist grundsätzlich in zwei Schritte gegliedert. Im ersten Schritt wird die zugehörige und bereits bestehende Entwicklung des 6LoWPAN-Kernelmoduls erweitert. Der zweite Schritt befasst sich mit der Implementierung des Border Routers. Die Abfolgen werden soweit automatisiert, dass auf Linux (respektive dem *Raspberry Pi 3*) die nachfolgend genannten Anforderungen erfüllt und die daraus resultierenden Aufgaben automatisch ausführbar sind:

- Die 6LoWPAN Implementation inkl. dem 6LoWPAN-Kernelmodul aus dem `Debug Filesystem` auf ein Produktiv-Dateisystem zu bringen.
- Die fehlende Unterstützung des 6LoWPAN-Kernelmoduls zu korrigieren, IPv6-Pakete in einem Netzwerk mit mehreren, gleichzeitig verbundenen Geräten senden zu können.
- Eine Unterstützung mehrerer Bluetooth-Dongles am *Raspberry Pi* zu schaffen. Dabei soll es möglich sein, eine Auswahl zu treffen, welches BLE-Gerät welchem Dongle (bzw. Bluetooth Controller) zugewiesen wird.
- Die Implementation einer vollautomatischen Applikation in Form eines Daemons. Dabei soll der Benutzer die Möglichkeit haben, diesen auch während der Laufzeit konfigurieren zu können.

Für die Funktionalitäten des *Daemons* sind folgende Ziele festgelegt:

- Die Grundfunktionalität soll dabei einem *WiFi-Router* ähneln, wobei kontinuierlich nach neuen oder bereits bekannten *IPv6-over-BLE*-fähigen Geräten gesucht werden soll. Die theoretischen Grundlagen dahinter sind in Abschnitt 2.2 erläutert.

Wird ein gültiges² Gerät erkannt, so soll dieses in weiterer Folge mit dem BLE Border Router verbunden werden.

- Der Benutzer soll wahlweise per Kommandozeile oder per Konfigurationsdatei die BLE-Geräte und den Daemon konfigurieren können. Mögliches Szenarium wäre das Setzen der *Link-Layer Connection Parameter* [3, S.1348] (Siehe „Erweiterte Funktionalitäten“).
- Wichtige Events sollen persistent gespeichert werden, um sie auch später abrufen zu können. Dazu zählen beispielsweise das Mitloggen von Verbindungsaufläufen oder Trennen der BLE-Geräte.

Nachfolgend sind **erweiterte Funktionalitäten** des Border Routers aufgelistet, welche im Rahmen der Arbeit zusätzlich zu den oben genannten Grundfunktionalitäten implementiert wurden:

- Die Unterstützung von mehreren *Host Controller Interfaces* (HCIs) (respektive BLE-Dongles).
- Die *Connection Parameters* eines BLE-Geräts sollen durch den Benutzer wahlweise per Konfigurationsdatei oder durch direkte Interaktion mit dem Daemon gesetzt werden können.

1.4 Struktur der Arbeit

Das nachfolgende Kapitel 2 befasst sich mit den technischen Grundlagen. Dabei wird einerseits auf die Funktionsweise von Bluetooth Low Energy, andererseits auf die von *IPv6-over-BLE*, eingegangen. Hierbei werden theoretische Grundlagen und verwendete Begriffe näher erklärt. In diesem Zusammenhang wird insbesondere auf die im Rahmen dieser Arbeit wichtigen Kapitel der Bluetooth Spezifikation Bezug genommen.

In Kapitel 3 wird die Adaptierung des 6loWPAN Kernelmoduls erläutert. Dabei werden die abgeänderten und adaptierten Codeteile genauer beleuchtet. Das Kapitel 4 geht auf die Implementierung des Daemons ein, dessen Evaluierung und die prototypische Anwendung wird in Kapitel 5 beschrieben. Die Ergebnisse finden sich zusammengefasst in Kapitel 6 wieder. An diesem Punkt wird auch auf Ausblicke und Erweiterungsmöglichkeiten eingegangen.

² Das Gerät unterstützt IPv6-over BLE

2

Technischer Hintergrund

In diesem Kapitel wird neben der theoretischen Betrachtung der einzelnen Funktionalitäten speziell auf die Bluetooth LE- und IPv6-Technologie eingegangen.

Das Ziel ist es, ausgewählte Themen der Technologie *Bluetooth Low Energy* vorzustellen, wobei in Grundzügen die Funktionsweisen beschrieben werden. Besonders Aspekte, welche für die Implementierung des Border-Routers wichtig sind, werden in diesem Zusammenhang herausgearbeitet.

2.1 Bluetooth Classic

Bluetooth wurde ursprünglich zur drahtlosen Überbrückung von Telefonie- und Computerwelten [18], sowie zur Ablöse von den nur auf Sicht funktionierenden Infrarotverbindungen, entwickelt [19]. 1998 wurde aus einem fünf Firmen umfassenden Konsortium die *Bluetooth Special Interest Group (SIG)* geformt, welche 1999 den ersten Standard Bluetooth 1.0 [20] hervorbrachte. Mittlerweile ist Bluetooth als Grundausstattung in nahezu allen tragbaren Geräten, wie Smartphones, Computern, Laptops und Tablet-PCs, oder Peripheriegeräten wie Tastaturen und Mäusen zu finden. Selbst in Geräten wie Küchenwaagen wird mittlerweile die Funktechnologie eingesetzt. Dabei sind nicht nur einfache Punkt-zu-Punkt Anwendungen möglich. Neben Protokollen zur Übertragung von Audio-Dateien und Daten bietet Bluetooth auch die Möglichkeit, ganze Netzwerke mit mehreren Bluetooth-Geräten zu erstellen [1].

Netzwerktopologie und BLE

In den ersten Versionen von Bluetooth wurde nur eine *Point-to-Point* Verbindung unterstützt. Das heißt, zwei Knoten (Nodes) konnten nur direkt miteinander verbunden werden.

Mit der Adaption der *Bluetooth Core Spezifikation Version 4.0* im Jahr 2010 wurde zudem Bluetooth Low Energy eingeführt [20]. Neben dieser energiesparenden Variante, die jedoch nicht mit dem klassischen Bluetooth kompatibel ist, gibt es seitdem für Bluetooth-Anwendungen die Möglichkeit komplexerer Netzwerke (siehe Abbildung 2.1).

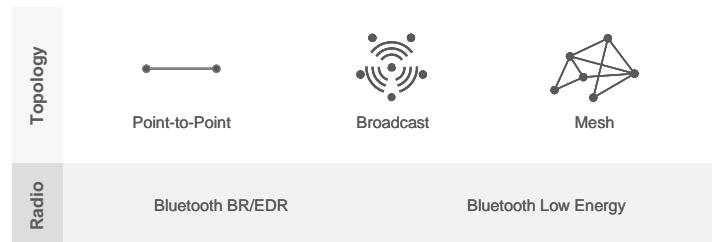


Abbildung 2.1: Solution Areas (entnommen von [1])

Zusätzlich zum *Broadcasting* wird seitdem die Option geboten, ein System als *Punkt-zu-Mehrpunkt*-Netzwerk zu betreiben [14].

2.2 Bluetooth Low Energy

Bluetooth Low Energy hat gegenüber dem klassischen Bluetooth mehrere entscheidende Vorteile, weshalb diese Technologie besonders für den *Wearable-* und *IoT*-Bereich interessant ist.

Dazu zählen neben der Einfachheit und der Verbreitung im Besonderen die Energieeffizienz: BLE-Geräte verbringen dabei die meiste Zeit in einem schlafenden Zustand [21], was diese Technologie energiesparend und daher für diesen Bereich attraktiv macht. Gegenüber dem herkömmlichen Bluetooth Standard wird ein vereinfachter *Link-Layer* verwendet. Zudem sind durch den BLE-Standard nur drei Kanäle für das *Advertising* spezifiziert [3, S. 2689]. Mit diesen speziellen Kanälen ist es möglich, eine Verbindung zwischen zwei Bluetooth LE Geräten in weniger als drei Millisekunden herzustellen [17]. Anstatt daher alle Kanäle, wie dies beim klassischen Bluetooth der Fall ist, zu scannen, reicht es drei zu überprüfen, um festzustellen, ob sich Geräte in der Nähe befinden.

Nachfolgend sind einige Grundlagen behandelt, welche im Rahmen dieser Arbeit relevant erscheinen.

2.2.1 Bluetooth Protocol Stack

In Abbildung 2.2 ist der *Protocol Stack* des BLE Protokolls dargestellt. Die folgenden Abschnitte werden sich näher mit den Schichten des Stacks befassen. Konkreter wird auf den physikalischen Layer (*Physical Layer*), den Link-Layer (*Link-Layer*) und auf das Host-Controller Interface (*HCI*) eingegangen.

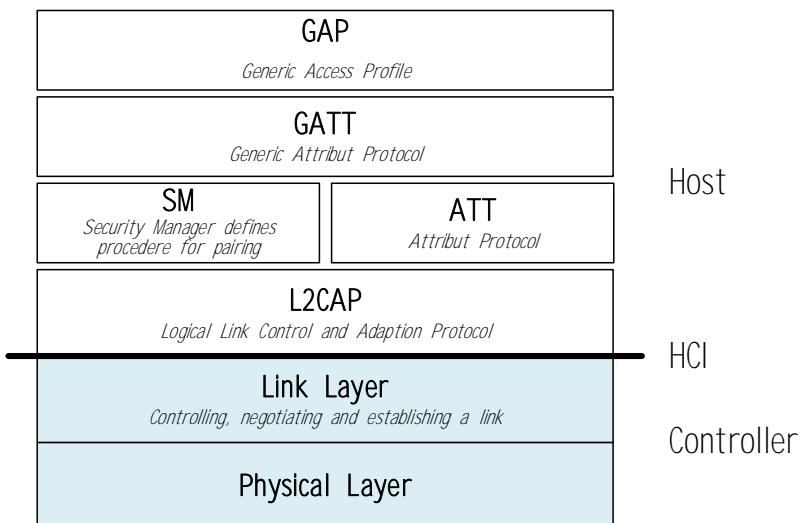


Abbildung 2.2: Aufbau des BLE Protocol Stacks (adaptiert von [2, S. 144])

2.2.2 Physical Layer

BLE arbeitet im nicht lizenzierten 2.4 GHz ISM Band mit 40, im Abstand von 2MHz liegenden, (physikalischen) RF³-Kanälen. BLE teilt sich damit mit WLAN dasselbe Frequenzband.

Drei dieser Kanäle, die „*primary advertising physical channel*“ sind für das *Advertising* (in etwa: das Werben, Ankündigen) reserviert und dienen zur Bekanntgabe eines in der Nähe befindlichen Geräts.

³ RF: Radio Frequency Kanäle, welche zum Übertragen verwendet werden

Die restlichen 37 Kanäle („secondary advertising physical channel“) werden für die Übertragung von Daten (*Data Transmission*) genutzt, wenn zwischen den Geräten bereits eine Verbindung besteht [3, 22, S. 2689].

LE Controller selbst können eine der drei Funktionen besitzen [2]:

Transmitter Beispielsweise eine Fernbedienung, die nur senden, aber nicht empfangen muss

Receiver Zum Beispiel das zugehörige Fernsehgerät, welches nicht mit der Fernbedienung kommunizieren muss

Transmitter und Receiver Unterstützt das Senden und Empfangen

Central und Peripheral Devices

Ein BLE Gerät kann zudem eine der beiden folgenden Rollen besitzen. Welche der Rollen eingenommen wird, entscheidet sich beim Verbindungsauftbau.

Central Device Das sind typischerweise Computer oder Smartphones

Peripheral Device Üblicherweise Sensoren oder dergleichen

Zudem wird zwischen zwei Typen von Daten unterschieden, welche von solchen Geräten gesendet werden können: *Advertising Packets* und *Scan Response Data*.

2.2.3 Link Layer

Der Link Layer ist für das Verwalten der Verbindung verantwortlich. Der anschließende Abschnitt stellt den Link Layer kurz vor, befasst sich jedoch näher mit den *Advertising Events*.

BLE Advertising Events

Unabhängig, ob es sich um ein BLE-Gerät handelt, welches nur Beacons⁴ überträgt, oder eine Dauerverbindung mit dem Host aufbauen will, startet jedes Gerät im *Advertising Mode* [22]. Damit BLE-Geräte andere BLE-Geräte erkennen ist es notwendig, dass derartige *Advertising Packets* permanent von *Peripheral Device* gesendet werden [23].

Für einen bidirektionalen Datenaustausch sucht ein Master nach BLE-Geräten (Peripherals), welche eine Verbindung aufbauen wollen und initiiert diese bei einem entsprechenden *Advertising Packet*. Ist eine Verbindung erfolgreich hergestellt, so befinden sich beide in einem gewissen Zustand, dem *connected link layer state*. Derjenige, der die Verbindung initiiert, nimmt die Rolle des *Masters*, derjenige, der die *Advertising Packet* versendet hat, die des *Slaves* ein.

Mit Hilfe von *Advertising Packets* gibt ein BLE-Gerät zudem bekannt, welche Absichten es hat.

⁴ Ein Beacon kann beispielsweise Ortsangaben, Temperaturdaten oder ähnliches sein

Bluetooth Link Layer Packet

Ein einzelnes Paket (Abbildung 2.3) kann sowohl für *Advertising* als auch für die Datenübertragung verwendet werden [22]. Diese Pakete bestehen aus vier obligatorischen Feldern:

- *Präambel*
- *Access Address*
- *Protocol Data Units (PDU)*
- *Cyclic Redundancy Check (CRC)*

Darüber hinaus besteht ein weiteres optionales Feld.

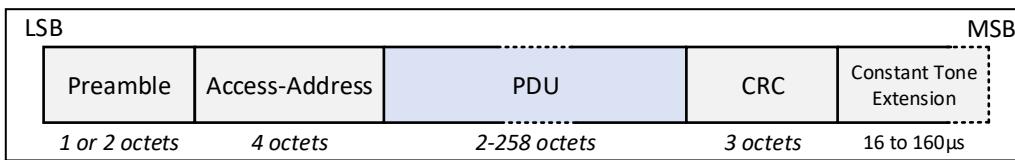


Abbildung 2.3: Link Layer Packet Format (adaptiert aus [3, S. 2691])

Wichtig hierbei ist das PDU-Feld, da mit diesem Feld bestimmt werden kann, ob es sich um ein *Advertising Package* oder ein *Data Package* handelt [22].

BLE Nodes nutzen bestimmte *BLE Advertising Informationen* um ihre IPv6-over-BLE Funktionalität bekannt zu geben. Diese Informationen finden sich in der *PDU* als *Universally Unique Identifier (UUID)* wieder.

Um eine IPv6-Funktionalität festzustellen, kann der BLE Border Router gezielt nach diesen *BLE Advertising Informationen* suchen und initiiert anschließend eine Verbindung mit den BLE-Nodes.

Die in diesem Fall relevante Information ist der *Internet Protocol Support Service (IPSS)* mit der UUID 0x1820. Ein beispielhaftes *Link Layer Packet* [3, S. 2691], welches die gesuchte UUID 0x1820 enthält, ist nachfolgend in Abbildung 2.4 dargestellt. Dabei wurde die *Präambel* ausgelassen. Das Paket ist im *LittleEndian*-Format dargestellt.

1. **Access Address** 0x8E89BED6: Die Access-Adresse 0x8E89BED6 ist ein standardmäßiger 4-Byte großer hexadezimaler Wert und dient der Identifikation von BLE-Geräten, wenn andere LE Geräte denselben physikalischen Kanal benutzen [2].
2. **Packet Header** 0x0D60: Definiert (unter anderem) den PDU Type [3].
3. **Advertising Address** 0xD874EA0BF1ED: Enthält die Adresse des sendenden BLE-Geräts.
4. **Advertising Data**: Das Datenfeld. Dieses enthält ein Feld für eine „Flag“, sowie die gesuchte UUID 0x1820.
5. **CRC**: 24 bit CRC-Wert aller PDU-Bits [3, S. 2695].

Damit die Daten vom Controller zum Host gelangen und die oberen Schichten (L2CAP, ATT, GATT und GAP; siehe Abbildung 2.2) mit den darunter liegenden Schichten kommunizieren können, gibt es das *Host Controller Interface (HCI)*.

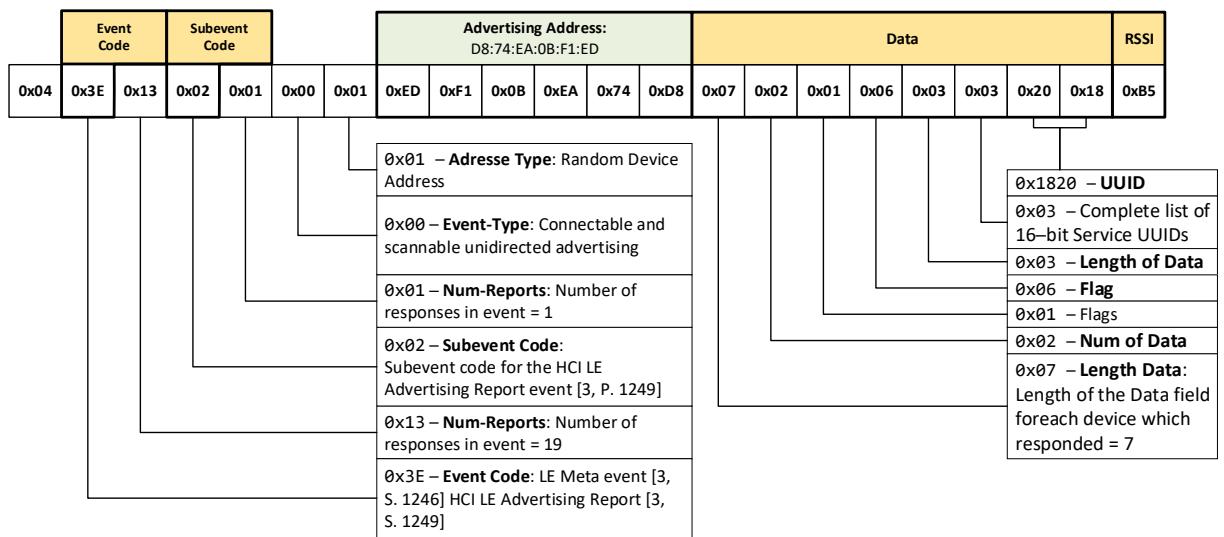


Abbildung 2.4: Beispiel eines Link Layer Packets im Advertising-Modus (inkl. Aufbau)

2.2.4 Host Controller Interface

In der Regel sind der Bluetooth-Controller und das, den Controller bedienende Gerät getrennt. Das Gerät (der Host) ist in diesem Fall beispielsweise der Prozessor des Smartphones oder eines PCs. Die Anbindung des Bluetooth-Controllers kann unter anderem über USB erfolgen, wie dies bei einem handelsüblichen USB-Bluetooth-Adapter der Fall ist.

Zur Kommunikation zwischen dem Host und Bluetooth-Controller wird das *Host Controller Interface* (HCI) verwendet, welches eine Schnittstelle zwischen diesem Host und dem Bluetooth Controller darstellt. Diese Schnittstelle kann in unterschiedlicher Form realisiert sein, z. B. UART, USB oder Ähnlichem. Die Kommunikation über das HCI zwischen dem Host und dem Controller findet in Form von *Command-* bzw. *Event-Packets* (siehe Abschnitt 2.2.4) statt.

Der schemenhafte Aufbau eines solchen *Event-Packets* ist auf Seite 16, Abbildung 2.5, ersichtlich.

HCI Commands und Events

Die Kommunikation zwischen dem Host und dem Controller basiert auf Paketen. Grundsätzlich unterscheidet man zwischen den folgenden Typen [2]:

1. **HCI Command Packets:** Zum Senden von Kommandos und Befehlen zwischen dem Host und dem Controller.
Beispiel: Zurücksetzen (Reset) des Controllers durch den Host.
2. **HCI Event Packets:** Zur Benachrichtigung des Hosts durch den Controller im Falle eines eintretenden Events.
Beispiel: Kommando zum Verbindungsauftbau oder Antwort auf einen bereits gesendeten Befehl.
3. **HCI ACL Data Packets:** Datenaustausch zwischen dem Host und dem Controller nach Verbindungsauftbau.

Im Rahmen dieser Arbeit sind für die Funktionalität des BLE Border-Routers insbesondere die *HCI Event Packets* interessant. Details zur Implementation finden sich im Abschnitt 4.5.3

HCI Event Packets

Die nachfolgende Abbildung 2.5 zeigt den schemenhaften Aufbau eines HCI *Event-Packages*.

Das Paket beginnt mit dem in der Grafik nicht eingezeichneten Indikator 0x04. Diesem Oktett folgt ein Event-Code. Für das *LE Meta Event* ist das der hexadezimale Wert 0x3E.

Alle auftretenden Events sind im Falle von BLE in diesem gemeinsamen *LE Meta Event* (0x3E) gekapselt, wobei ein zugehöriger Subcode die genauere Bezeichnung des *Meta Events* enthält [2].

Event Code 0x3E	Parameter Total Length	Sub Event Code	Sub Event Parameter 0	Sub Event Parameter ...
--------------------	---------------------------	-------------------	-----------------------------	-------------------------------

Abbildung 2.5: Aufbau eines HCI Event Packets (adaptiert aus [2, S. 199])

Die folgende Abbildung 2.6 zeigt ein aufgeschlüsseltes HCI Event Paket.

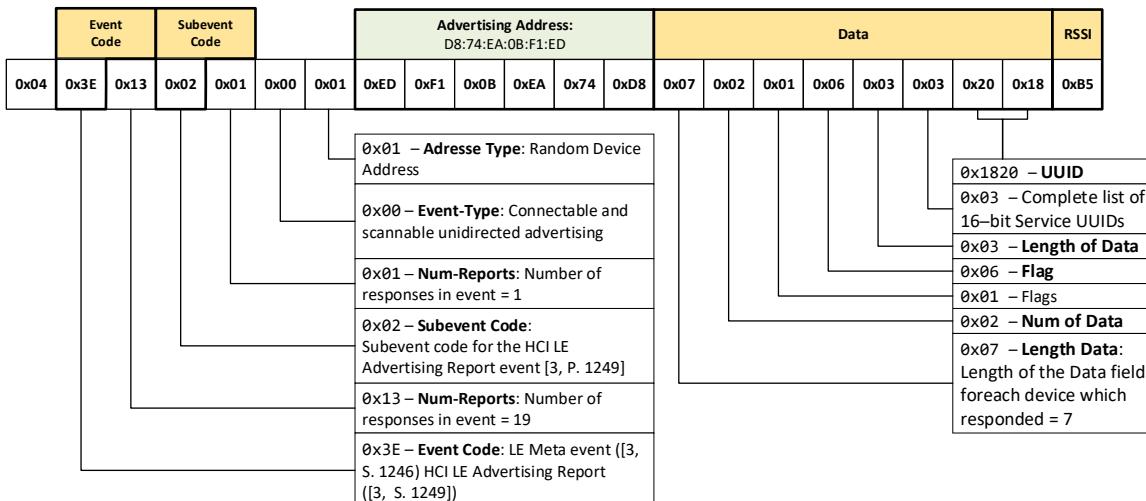


Abbildung 2.6: Exemplarisches HCI-Packet

Eine komplette Liste an Events findet sich in der *Bluetooth Core Specifications Version 5.1, Vol 2, Part E* [3, S. 1156].

Der Subevent-Code *LE Advertising Report Event* ist in der Bluetooth Core Specification Version 5.1 [3] auf Seite 1249 näher beschrieben.

Befindet sich ein BLE Gerät in Reichweite eines Empfängers, kann durch gezieltes Überprüfen der *BLE Advertising Informationen* nach dem *Service Data*-Feld gesucht werden. Das Feld wird über die Generic Attribute Profile (GATT) festgelegt und beschreibt Merkmale und Verhalten von BLE Geräten [3]. In diesem Fall wird nach der *IP Support Service UUID* 0x1820 gefiltert.

2.3 IPv6 und Bluetooth LE

Die Grundidee des Vorhabens ist es, jedem Gerät eine individuelle IP-Adresse zuordnen zu können. Dazu wurde in der Spezifikation *RCF 7668* [4] die Verwendung von IPv6-Paketen zur Übertragung festgelegt.

Wohingegen IPv4 mit seinem Adressbereich bereits jetzt an die Grenzen gelangt, bietet IPv6 gegenüber seinem Vorgänger eine weitaus höhere Anzahl an Individualadressen. In dem nachfolgenden Kapitel wird der IP-Standard Version 6 in Grundzügen vorgestellt.

2.3.1 Internet Protokoll Version 6

IPv6 ist der Nachfolger des im Jahr 1981 eingeführten Standards IPv4 [24]. In den nächsten Abschnitten sollen einige Vorteile von IPv6 gegenüber IPv4 aufgelistet werden [25]:

1. Erweiterter Adressraum

Gegenüber IPv4 wurde der Adressraum von 32 bit auf 128 bit erhöht. Damit bietet dieser im Vergleich zum Vorgänger mit knapp 4.3 Milliarden (2^{32}) Adressen einen weitaus größeren Adressbereich von 2^{128} theoretisch möglichen Adressen.

2. Autokonfiguration

Mit IPv6 wurde die *Stateless autoconfiguration* eingeführt, welche das automatische Konfigurieren der Interfaces ermöglicht, ohne einen DHCP Server verwenden zu müssen. IPv6-Adressen werden in diesem Fall nicht zentral vergeben und gespeichert, sondern durch den Host selbst erzeugt. Das ermöglicht eine einfache Vergabe von IP-Adressen [26]. Die *Stateless autoconfiguration* wurde ursprünglich durch Dokument *RCF 2462* [27] spezifiziert.

3. Einfacher Header

Gegenüber dem Vorgängerformat ist der *Packet Header* mit einer fixen Länge von 40 Bytes einfacher [28]. Das erlaubt eine schnellere Verarbeitung der Pakete als es bei IPv4 der Fall ist.

Für das schnell wachsende *Internet of Things (IoT)* liegen die Vorteile von IPv6 damit klar auf der Hand.

2.3.2 IPv6 over BLE

Mit Version 4.1 der Bluetooth Spezifikation wurde ein erweiterter *Internet of Things*-Support eingeführt. Durch den bereits im Jahr 2015 veröffentlichten Standard *RCF 7668* (IPv6 over BLUETOOTH(R) Low Energy) [4] wurde eine Möglichkeit definiert, wie IPv6-Pakete über ein BLE Verbindung übertragen werden. In Abbildung 2.7 ist der *Bluetooth Protocol Stack* dargestellt.

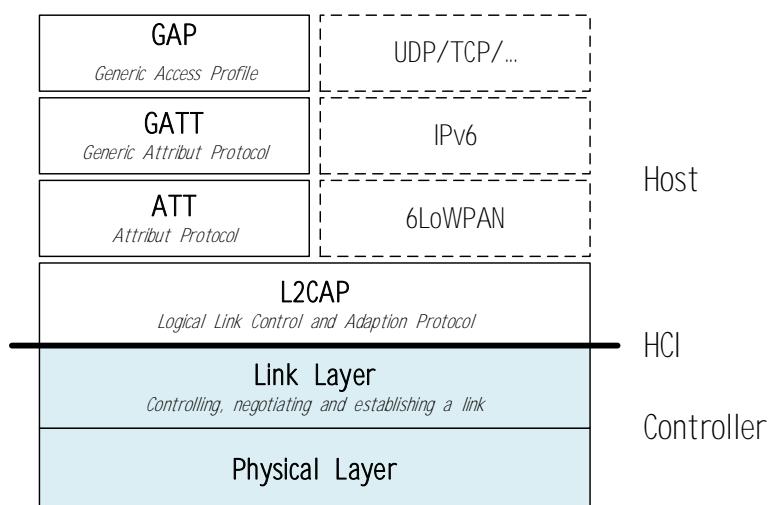


Abbildung 2.7: BLE Protocol Stack mit IPv6 Unterstützung (adaptiert aus [4])

Im Folgenden werden einige Begrifflichkeiten und Standards hierzu vorgestellt.

RFC 7668 IPv6 over BLUETOOTH(R) Low Energy [4]

Spezifiziert wie die IPv6-over-BLE Kommunikation über BLE-Verbindungen stattfindet, wobei hier die Übertragungstechnik von Low-power Wireless Personal Area Network (6LoWPAN) genutzt wird. Ein *6LoWPAN for Bluetooth LE-Layer* wird im Bluetooth Communication Stacks oberhalb der BLE L2CAP Protokollsicht (Abb. 2.7) eingefügt [4](3.1).

RFC 6282 Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks [29]

Die relativ großen IPv6- und UDP-Header müssen im besten Fall auf einige wenige Bytes reduziert werden, um IPv6-Pakete über BLE übertragen zu können.

RFC 6775 Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) [30]

Das Dokument beschreibt den Ansatz der Neighbor Discovery in 6LoWPAN-Topologien, einschließlich der Mesh-Topologie. Da BLE erst seit neueren Versionen Mesh-Netzwerke unterstützt, werden nur die Aspekte für Sternnetzwerke (Abbildung 2.8) berücksichtigt.

RFC 4944 Transmission of IPv6 Packets over IEEE 802.15.4 Networks [31]

Beschreibt dabei die Übertragungsprotokolle von IPv6-Paketen über den Standard *IEEE 802.15.4 Low-Rate Wireless Personal Area Networks (LR-WPAN)*. Die BLE Verbindung weist ähnliche Eigenschaften wie IEEE 802.15.4 auf und viele der hierin definierten Mechanismen können auf die Übertragung von IPv6 auf BLE-Verbindungen angewendet werden.

BLE benötigt einen adaptierten 6LoWPAN Layer oberhalb der L2CAP-Schicht.

Abbildung 2.7 zeigt den IPv6-Stack parallel zum GATT-Stack. Der GATT Stack wird benötigt, um eine gültige Node anhand des *Internet Protocol Support Service* erkennen zu können [4].

6LoWPAN

Das Akronym *6LoWPAN* steht für *IPv6 over low-power wireless personal area networks* [32]. Wie aus dem Namen hervorgeht, handelt es sich hierbei um *Personal Area Networks*, welche zur Kommunikation zwischen Geräten wie Smartphones, Keyboards oder Ähnlichem eingesetzt werden. Die Reichweite ist daher auf einige Meter beschränkt [2, S. 3]. Das Präfix „Lo“ für *Low Power* macht deutlich, dass die Anwendung besonders für den Bereich der Funkübertragung mit niedrigem Energieverbrauch ausgelegt wurde.

Netzwerktopologie

Entsprechend dem Dokument RCF 7668 [4] folgt das IPv6-over-BLE-Netzwerk der in Abbildung 2.8 dargestellten Sternkopologie. Der IPv6-over-BLE Border Router (6LBR) nimmt damit den Platz der *Central Role* ein, die übrigen IPv6-over-BLE Nodes (6LNs) implementieren die *Peripheral Nodes*.

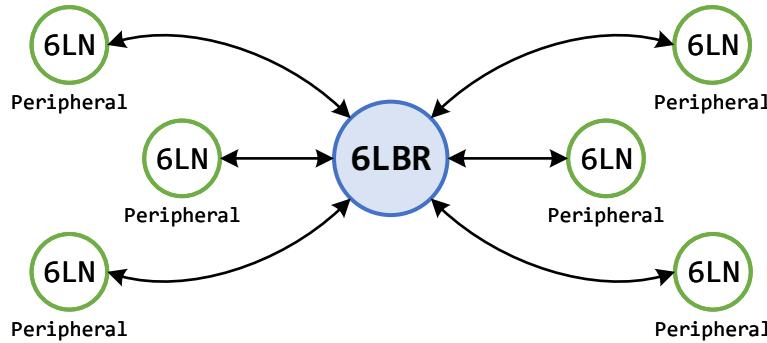


Abbildung 2.8: Topologie eines Sternenzwerts

Ein typisches Anwendungsszenario entsprechend dem Aufbau nach Abbildung 2.8 findet sich in Abschnitt 2.4 auf Seite 20.

Wie bereits auf Seite 18 erwähnt, wurden für IPv6-over-BLE schon bekannte Implementierungen und Spezifikationen genutzt. Dazu zählen die Dokumente *RFC 4944* [31], *RFC 6282* [29] und *RFC 6775* [30], welche für *IEEE 802.15.4* spezifiziert wurden.

Bluetooth 4.0 unterstützt nur Netzwerke der Sternkopologie (Abbildung 2.8). Seit Bluetooth Version 4.1 ist es aber auch möglich, neben mehreren *Peripheral-Devices* auch mit mehreren *Central-Devices* in einem Netzwerk eine Verbindungen untereinander herzustellen (sogenannte BLE-Mesh Netzwerke).

Das ursprüngliche Dokument RCF 7668 [4] berücksichtigt diese Art der Netzwerktopologie noch nicht und beschreibt daher nur Netzwerke, welche dieser Sternkopologie folgen. Für eine Kommunikation zwischen mehreren BLE-Geräten muss daher die IP-Routing Funktionalität genutzt werden.

Der Nachfolger von RCF 7668 ist noch nicht vollständig ausgearbeitet. Das zugehörige Dokument ist als Entwurf unter dem Namen *IPv6 Mesh over BLUETOOTH(R) Low Energy using IPSP* [33] veröffentlicht.

Das heißt im Umkehrschluss, dass *Peripheral Devices* untereinander nicht direkt miteinander sprechen können. *6LN-to-6LN* Kommunikationen über die Link-Local-Adressen sind in diesem Fall nicht möglich und müssen über den gemeinsamen Border Router ablaufen [4]. In diesem Fall hat jede BLE Node (*Peripherals*) eine eigene, direkte Verbindung zum Router [17].

Local Link Adresse

Jede der Nodes erhält eine 128-Bit IPv6 (Local-Link-)Adresse, welche auf Basis der privaten 48-Bit-Hardware Adresse (BLE-MAC-Adresse) der Nodes generiert wird [2, S.374].

Dabei wird aus der Geräteadresse der BLE-Node 00:11:22:33:44:55 die Local-Link-Adresse FE80::211:22FF:FE33:4455 (Nullen können weggelassen werden) erzeugt.

2.4 Anwendungsszenario

Die nachstehende Abbildung 2.9 zeigt ein mögliches Anwendungsszenario: Ein Netzwerkelement (die BLE Node) unterstützt den *Internet Protocol Support Service (IPSS)* und sendet oder empfängt IPv6-Pakete über BLE. Die Node, beispielsweise ein Sensor oder Handheld-Gerät, erhält eine eigene IPv6-Adresse und kann somit direkt mit Geräten im *Internet of Things* sprechen.

Der IPv6-over-BLE Border Router (6LBR) sendet oder empfängt entsprechende IPv6-Pakete über BLE. Fakultativ kann der Border Router in weiterer Folge mit dem Internet verbunden werden - dieser stellt somit eine Internetverbindung über die BLE-Verbindung den Nodes zur Verfügung.

Resultierend daraus, dass die Daten bereits als IPv6-Paket vorliegen, findet die Verarbeitung der Pakete durch den Router mit minimalem Aufwand statt.

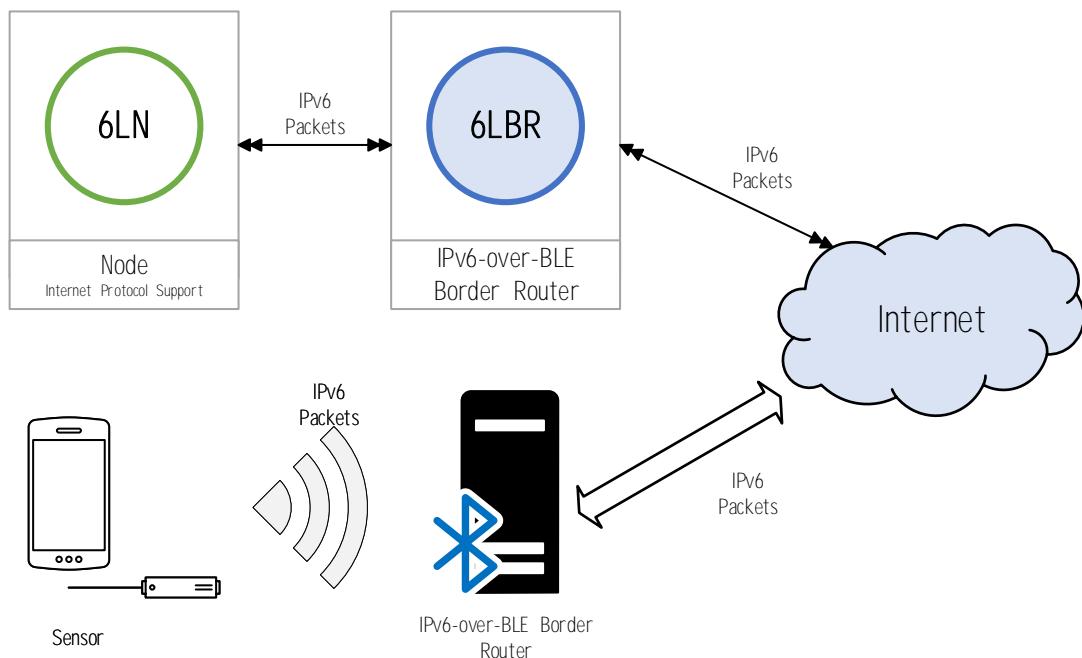


Abbildung 2.9: Anwendungsszenario mit einem Sensor und einem Border-Router (adaptiert von [2, S. 373])

2.5 Hardware

Dieser Abschnitt befasst sich mit der Hardware, welche im Rahmen der Arbeit zur Entwicklung und für Evaluierungszwecke benutzt wurde.

2.5.1 Raspberry Pi 3

Die Daemon-Applikation wurde für den *Raspberry Pi 3* entwickelt, welcher somit als *Border-Router* fungiert.

Der Raspberry Pi 3 ist die dritte Generation des *Single-Board* Computers mit einem *64-bit 1.2GHz Quad Core, 1 GB Arbeitsspeicher* und wird mit *WLAN* und *Bluetooth Low Energy(BLE)* [34] ausgeliefert. Die Verwendung eines Dongles ist nicht zwingend erforderlich. Als Stromversorgung wird ein 5V Netzteil mit 2.5A verwendet.

Als Betriebssystem wird *Raspbian OS* verwendet, eine auf Debian basierende Linux-Distribution für die Raspberry Pi Familie. In der Version *Raspbian OS 4.19* sind bereits ein Großteil der notwendigen Module und der Software mit enthalten.

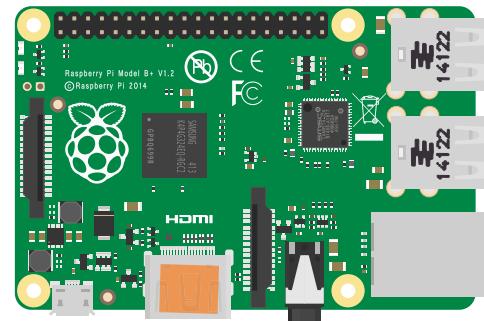


Abbildung 2.10: Raspberry Pi 3 (Bild aus [5])

2.5.2 Partical Xenon

Der *Particle Xenon* ist ein IoT-Development-Kit, basierend auf einem *Nordic nRF52840* mit einem 32-bit, *64Mhz ARM Cortex-M4F* Prozessor, *256KB RAM*, *IEEE 802.15.4* und *Bluetooth 5* [35]. Als Betriebssystem wird *Zephyr OS*⁵ verwendet, welches offiziell für diesen Typ unterstützt wird.



Abbildung 2.11: Partical Xenon (Bild entnommen aus [6])

2.5.3 Nordic

Semiconductor nRF52840 Development Kit

Das nRF52840 Development Kit ist ein Entwicklungsboard, das Bluetooth 5, Bluetooth Mesh, Thread, ZigBee, 802.15.4, ANT und proprietäre 2,4 GHz Anwendungen auf dem nRF52840 SoC⁶ unterstützt. Basierend auf einem ARM Cortex M4 bietet das Board ein USB Interface zum Debuggen und Programmieren [36].

Wie beim Partical Xenon kommt bei diesem Development-Board *Zephyr OS* zum Einsatz.

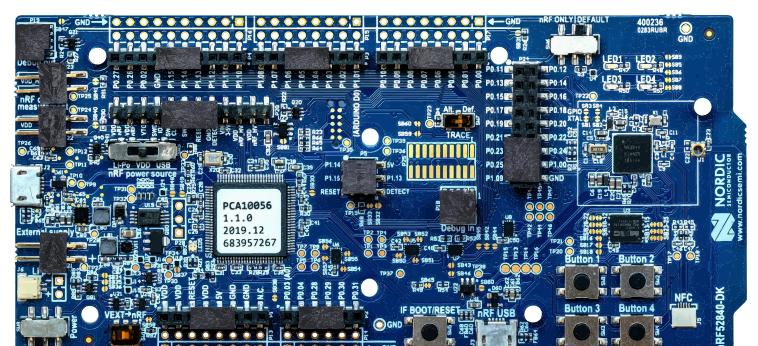


Abbildung 2.12: Nordic Semiconductor nRF52840 Development Kit

⁵ <https://www.zephyrproject.org/>

⁶ System on Chip

2.5.4 Panasonic PAN 1762 Development-Kit (USB Adapter)

Das PAN1762 Bluetooth LE Modul basiert auf dem *Toshiba TC35680* Einzelchip-Controller, ausgestattet mit einem ARM Cortex-M0. Der Flash-Speicher umfasst 128 kB, der RAM 144 kB. Das Modul enthält Standard-Profile der Bluetooth Special-Interest-Group [37].

Das Evaluierungskit besteht aus einem USB-Dongle der zum Betrieb, zur Entwicklung und zum Debuggen nutzbar ist. Das Kit bietet zudem die Möglichkeit, Sensoren über die Break-Out-Stifteleisten anzuschließen [38].



Abbildung 2.13: Panasonic PAN 1762 (Bild entnommen aus [7])
Möglichkeit, Sensoren über die Break-Out-Stifteleisten anzuschließen [38].

3

Bluetooth Kernelmodul

Gegenstand dieses Abschnittes ist die Anpassung des *bluetooth_6lowpan* Kernelmoduls. Hierbei wird insbesondere auf die notwendigen Adaptierungen des Quellcodes eingegangen.

3.1 6LoWPAN Kernelmodul

Ein Kernelmodul ist eine Software, die vom Betriebssystem während der Laufzeit in den Kern geladen werden kann. Damit bieten Kernelmodule die Möglichkeit, den Kernel zu erweitern. Prominente Beispiele sind Hardwaretreiber, wie beispielsweise jene für WLAN-Karten [39].

Der Border-Router setzt das aktivierte *bluetooth_6lowpan* Kernelmodul voraus. Dieses Modul erweitert das Betriebssystem um den BLE-Router Support und um die Fähigkeit der IPv6-Übertragung via BLE. Der Quellcode hierzu findet sich im Original im Sourcecode-Repository des Linux-Kernels⁷.

Die entsprechend dieser Arbeit angepasste Version wird im Projekt-Repository⁸ unter dem Pfad `linux_4_19/net/bluetooth/6lowpan.c` zur Verfügung gestellt.

Als Basis wurde eine frühe Version *0.1* verwendet, welche sich noch auf dem `debugfs` befindet. Das `debugfs` ist ein virtuelles Dateisystem des Linux-Kernels und hat, anders als beispielsweise das `procfs` keine Regeln, welche Informationen auf diesem abgelegt werden. Zudem ist es auch nicht als stabile ABI⁹ zwischen Kernel- und Userspace konzipiert [40]. Vielmehr sollte es dem Entwickler dazu dienen, (Debug-)Informationen der Kernelmodule in den Userspace zu exportieren.

Im Rahmen dieser Arbeit wurde das Kernelmodul leicht modifiziert. Dazu wurde im ersten Schritt das Dateisystem geändert. Das Kernelmodul soll vom `debugfs` auf ein Dateisystem geändert werden, welches für den Produktiv-Betrieb konzipiert ist.

Am Quellcode selbst sind ebenfalls Änderungen vorgenommen worden, die in den folgenden Abschnitten näher erläutert werden. Neben der Ausbesserung eines Fehlers im Quellcode des Moduls, der verhindert, dass mehrere verbundene BLE-Geräte in einem BLE-IP-Netzwerk erreichbar sind, wurde das Modul auch um die Funktion des Auswählens der Host Controller (`select`-Kommando, Abschnitt 3.3) erweitert.

In der aktuellen Kernelversion *4.14.95-v7* wird *Raspian OS* nicht mit einem vorkompilierten *bluetooth_6lowpan* Kernelmodul ausgeliefert. In diesem Fall muss das Kernelmodul zusätzlich kompiliert und installiert werden. Die Vorgangsweise wird in Kapitel 5 unter Abschnitt 5.1.1 beschrieben.

⁷ <https://github.com/raspberrypi/linux>

⁸ <https://github.com/agentsmith29/ipv6-over-ble-borderrouter.git>

⁹ Application Binary Interface, Schnittstelle zwischen Computerprogrammen

3.2 Mehrfache Peer-Verbindung

Entsprechend der Bluetooth-Spezifikation werden neben Punkt-zu-Punkt-Verbindungen auch Sternnetzwerke unterstützt (Abschnitt 2.3.2).

Das im Dokument RCF 7668 [4] beschriebene Szenario geht von einer solchen Sterntopologie (Kapitel 2.3.2 Abbildung 2.8) aus. Im Zuge dieser Arbeit findet daher die in dem Dokument beschriebene Netzwerktopologie ebenfalls Anwendung.

Demnach sind mehrere gleichzeitig verbundene BLE-Geräte möglich, wobei das Raspberry Pi als *Central Device* bzw. Border-Router agiert.

Die Implementation des *bluetooth_6lowpan* Kernelmoduls weist jedoch in Version 0.1 (zum Zeitpunkt des Verfassens die aktuelle, veröffentlichte Version), einen Fehler auf. Damit war es nicht möglich, ein beliebiges BLE-Gerät in einem Netzwerk zu erreichen, wenn gleichzeitig zwei oder mehrere BLE-Geräte mit dem Border-Router verbunden sind.

Trotz der scheinbar validen Verbindung reagieren die Nodes aber beispielsweise nicht auf *ICMPv6*-Pakete („*pings*“), wie sie das gleichnamige Kommandozeilenprogramm *ping6* sendet. Das Problem ist in Abbildung 3.1 grafisch dargelegt.

Ein entsprechender Test (beschrieben in Abschnitt 5.1) mit mehr als einem verbundenen BLE-Peer, schlägt bei Verwendung des unangepassten Kernelmoduls in Version 0.1, fehl.

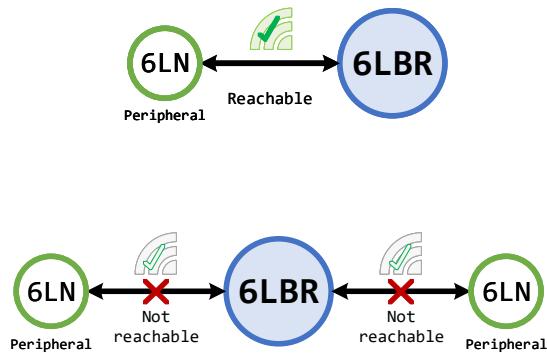


Abbildung 3.1: Grafische Darstellung des Verbindungsproblems mehrerer BLE-Geräte

3.2.1 Problembeschreibung

Wie bereits erläutert ist es nicht möglich, verbundene Geräte zu erreichen, wenn die Anzahl (> 1) an Geräten überschritten wird (Abbildung 3.1). Das Problem findet sich in der aktuellen Implementation des Bluetooth Kernelmoduls.

Das in Abschnitt 2.3 genannte Dokument RCF4944 [31] beschreibt dabei die Übertragung von IPv6-Paketen über 802.15.4-Netzwerke. Der 802.15.4-Standard unterstützt entsprechend seiner Spezifikation auch Multi-Hop-Netzwerke.

In der Datei `linux_4_19/net/bluetooth/6lowpan.c` ist die Funktion

```
static inline struct lowpan_peer *peer_lookup_dst
```

für das Ermitteln der Zieladresse der Datenpakete verantwortlich. Darin findet sich auch der Fehler, der verhindert, dass in einem BLE-Netzwerk mehr als ein verbundenes Gerät erreichbar ist. Die Implementation der Funktion geht in diesem Fall davon aus, dass bei nur einem verbundenen Gerät, eben nur dieses Gerät Pakete empfangen kann [41]. Im Codelisting 3.1 ist dieser Fall dargestellt.

```
181 if (count == 1) {
182     rcu_read_lock();
183     peer = list_first_or_null_rcu(&dev->peers, struct lowpan_peer,
184                                 list);
185     rcu_read_unlock();
186     return peer;
187 }
```

Script 3.1: Ermitteln der Zieladresse bei nur einem Peer

Zwischen Router und BLE-Node ist eine direkte Verbindung hergestellt, die Pakete werden über deren Lokal-Link Verbindung gesendet.

Sind jedoch mehrere Geräte in einem Netzwerk, wird überprüft, ob ein Paket eine bestimmte Adresse erreichen kann. Dieses Verhalten ist für 802.15.4 Netzwerke gültig, um mit komplexeren Multi-Hop-Netzwerken, wie sie von 802.15.4 unterstützt werden, umgehen zu können. In BLE-Netzwerken die der Stern topologie folgen, existiert, wie in Abschnitt 2.3.2 beschrieben, nur eine direkte Verbindung (über einen eigenen Link) mit dem Router. Gibt es keine Routes und wurde in den Paket-Daten auch kein Gateway angegeben, so führt dies dazu, dass die entsprechende Logik (Script 3.2) fälschlicherweise davon ausgeht, dass das Ziel nicht erreicht werden kann.

```
192 |     if (ipv6_addr_any(nexthop))
193 |         return NULL;
```

Script 3.2: Fehlerhafte Entscheidung, wenn mehr als ein Peer verbunden ist

Wird also weder eine Route noch ein Gateway gefunden, gibt die Funktion NULL zurück (Code-listing 3.2).

3.2.2 Lösungsansatz

Eine funktionierende Fehlerbehebung („Patch“) ist bereits im Linux Kernel Version 5 verfügbar. Dabei wurde für das Modul die verfügbare Änderung [42] und die von Sasha Levin [43] veröffentlichte, herangezogen.

```
185 static inline struct lowpan_peer *peer_lookup_dst(struct lowpan_btle_dev *dev,
186                                                 struct in6_addr *daddr,
187                                                 struct sk_buff *skb)
188 {
...
208     if (!rt) {
209         if (ipv6_addr_any(&lowpan_cb(skb)->gw)) {
210             /* There is neither route nor gateway,
211              * probably the destination is a direct peer.
212             */
213             nexthop = daddr;
214         } else {
215             /* There is a known gateway
216             */
217             nexthop = &lowpan_cb(skb)->gw;
218         }
219     } else {
220         nexthop = rt6_nexthop(rt, daddr);
...
226     memcpy(&lowpan_cb(skb)->gw, nexthop, sizeof(struct in6_addr));
227 }
...
246     return NULL;
247 }
```

Script 3.3: Ausbesserung der Funktion `peer_lookup_dst`

In der original vorliegenden Funktion wird bei mehr als nur einem verbundenen BLE-Gerät nicht mehr nach dem korrekten Peer gesucht. Mit der Änderung (Zeilen 213 bis 227) wird auch der Fall berücksichtigt, bei dem mehr als ein BLE-Gerät verbunden ist. Es wird somit die korrekte Adresse gefunden und das inkorrekte Routing-Verhalten des Border-Routers behoben.

3.3 Ansprechen mehrerer Host Controller

An den Border Router wird zusätzlich die Anforderung der Skalierbarkeit gestellt. Es soll möglich sein, mehrere *Host Controller* anzusprechen und zu verwalten. Das heißt, der Host Controller soll *gewechselt* werden können.

3.3.1 Wechseln der Host Controller

Unter *Wechseln* wird das dezidierte Ansprechen des Controllers verstanden: Eine gemeinsame Kontroll-Datei (`6lowpan_control`) soll zur Konfiguration aller verfügbaren Controller dienen. In der ursprünglichen Version des Kernelmoduls war dies nicht möglich. Hier wurde bei mehr als zwei Host Controller Interfaces jenes mit der höheren HCI-Nummer angesprochen.

Da das Verhalten der ursprünglichen Implementierung den Anforderungen der Arbeit nicht gerecht wird, ist nachfolgend eine Möglichkeit beschrieben, wie ein Wechsel der Host-Controller implementiert werden kann. Mittels dem, im Folgenden näher erläuterten `select`-Kommando, soll es möglich sein festzulegen, für welchen Controller nachfolgende Kommandos bestimmt sind. Eine Befehlskette wird solange an den ausgewählten Controller weitergeleitet, bis dieser mittels `select` erneut gewechselt wird.

3.3.2 Implementation

Um das Wechseln der Host Controller zu ermöglichen, müssen an folgenden aufgelisteten Dateien Modifikationen vorgenommen werden. Weniger relevanter Code ist in den nachfolgenden Quellcodeauszügen aus Gründen der Übersichtlichkeit weggelassen. Diese Teile sind mit [...] gekennzeichnet.

- `linux_4_19/net/bluetooth/hci_conn.c`
- `linux_4_19/net/bluetooth/6lowpan.c`

In diesen Dateien wird die Logik für das `select`-Kommando implementiert. Die Anwendung des Befehls soll dabei ähnlich den bereits verfügbaren `connect`- und `disconnect`-Befehlen erfolgen. Zur Steuerung wird ebenfalls die Kernel-Control-Datei `6lowpan_control` verwendet. Die Syntax zum Wechseln ist daher analog zu den beiden obig genannten Kommandos.

Für die Interpretation des Befehls muss in der Datei `linux/net/bluetooth/6lowpan.c` folgendes implementiert werden:

```
1196 static ssize_t lowpan_control_write(struct file *fp, const char __user *user_buffer,
1197                                     size_t count, loff_t *position)
1198 {
1199 ...
1279     if (memcmp(buf, "select ", 5) == 0) {
1280         buf[buf_size] = '\0';
1281
1282         bdaddr_t hci_dev_addr_tmp;
1283         hci_dev_addr = &hci_dev_addr_tmp;
1284         sscanf(&buf[7], "%hhx:%hhx:%hhx:%hhx:%hhx:%hhx",
1285             &hci_dev_addr->b[5], &hci_dev_addr->b[4], &hci_dev_addr->b[3],
1286             &hci_dev_addr->b[2], &hci_dev_addr->b[1], &hci_dev_addr->b[0]);
1287
1288         printk(KERN_INFO "Selected hci: %pMR", hci_dev_addr);
1289         set_default_hci(hci_dev_addr);
1290     }
1291     return count;
1292 }
```

In der Datei `linux_4_19/net/bluetooth/hci_conn.c` wird zudem eine neue Funktion `set_default_hci` erstellt, die das Setzen der gewünschten Host Controller Adresse erlaubt.

```

634  /* This is a function for selecting the default hci device
635  /* This is for storing the default hci device address */
636  uint8_t set_default_hci_device[8];
637
638  int set_default_hci(bdaddr_t *def_hci_address) {
639      printk(KERN_INFO "hci_conn: Set default hci: %pMR\n", def_hci_address);
640      int i = 0;
641      for(i = 0; i <= 5; i++) {
642          set_default_hci_device[i] = def_hci_address->b[i];
643      }
644      printk(KERN_INFO "Using default hci: %pMR\n", set_default_hci_device);
645      return 0;
646  }
647 EXPORT_SYMBOL(set_default_hci);

```

Dabei ist `EXPORT_SYMBOL` wichtig, da erst dadurch das Symbol `set_default_hci` für dynamisch geladene Kernelmodule verfügbar gemacht wird. Anschließend muss geprüft werden, ob die verfügbaren Host Controller Adressen mit der gewollten abgleichbar sind. Dies erfolgt in der Funktion `hci_get_route` (Zeile 686 - 700). Gibt es eine Adresse, welche zur der, mit `set_default_hci` gesetzten passt, wird dieser Host Controller gewählt (Zeile 696).

```

636  struct hci_dev *hci_get_route(bdaddr_t *dst, bdaddr_t *src, uint8_t src_type)
637  {
...
686      if (bacmp(&d->bdaddr, dst)) {
687          hdev = d;
688          // Here we check if the selected device is in the list
689          int i;
690          int match = 1;
691          for(i = 0; i < sizeof(d->bdaddr); ++i) {
692              if(d->bdaddr.b[i] != set_default_hci_device[i])
693                  match = 0; break;
694          }
695
696          if(match == 1) {
697              printk(KERN_INFO "Match %pMR", &d->bdaddr);
698              break;
699          }
700      }
...
708      return hdev;
709  }

```

Wird „select“ in die Datei geschrieben, so ruft `lowpan_control_write` die Funktion `set_default_hci_device` auf und in der Funktion `hci_get_route` wird das Gerät gewählt.

3.3.3 Anwendung des select-Befehls

Der Befehl wird, wie die bereits implementierten Befehle `connect` und `disconnect`, durch das Schreiben in die Kontrolldatei aufgerufen.

```
echo "select XX:XX:XX:XX:XX:XX" > /proc/ble/6lowpan_control
```


4

BLED Daemon

Dieses Kapitel befasst sich mit der Implementierung der Daemon-Applikation. Zudem werden die verwendeten, frei erhältlichen Programmbibliotheken und andere frei erhältliche Codes bzw. Codeteile, vorgestellt. Dem Leser wird der Aufbau der Applikation nähergebracht und auf Umsetzung, aber auch auf deren Verbesserungsmöglichkeiten eingegangen.

Dabei richtet sich der Fokus insbesondere auf die Implementierung zur Interaktion zwischen dem 6LoWPAN Kernelmodul und dem Ansprechen der *Host Controller Interfaces*, sowie auf die Automatisierungsschritte.

Im übrigen wird auf Eigenheiten, Interaktionsmöglichkeiten, aber auch auf Implementationen der im Kapitel 2 beschriebenen Punkte, eingegangen.

4.1 Software-Lizenz und Code-Struktur

Die Arbeit selbst wird als Open-Source Projekt unter der Lizenz *GNU General Public License 3*¹⁰ (GPL 3.0) veröffentlicht und kann frei verwendet werden.

Das Projekt wird auf der Software-Entwicklungsplattform *Github* zu Verfügung gestellt:

<https://github.com/agentsmith29/ipv6-over-ble-borderrouter>

Darin enthalten sind jegliche Dateien, auf die in dieser Arbeit referenziert werden.

Die Applikation mit dem Namen BLED ist als Daemon, das heißt in Form eines Unix Hintergrundprogramms, entwickelt worden. Eine direkte Interaktion ist nicht vorgesehen und in diesem Sinne auch nicht ohne weiteres möglich. Üblicherweise finden Interaktionen auf anderem Wege, nämlich indirekt statt (z. B. via Sockets¹¹ oder Pipes). Mit Interaktionsmöglichkeiten befasst sich der Abschnitt 4.5.6 ab Seite 41 näher. Im Abschnitt 5.3 findet sich die dazugehörige Evaluierung.

4.2 Voraussetzungen und Abhängigkeiten

Für den Betrieb des Daemons sind folgende Voraussetzungen spezifiziert:

- *Raspberry Pi 3* mit *Raspbian GNU/Linux 9 (stretch)* in der Kernelversion *4.14.95-v7* oder neuer. Das 6LoWPAN Kernelmodul muss über die *raspi-config* gegebenenfalls aktiviert werden.
- Das 6LoWPAN Kernelmodul in der Version 0.2.3b. Die notwendigen Quelldateien sind dem Projekt (Abschnitt 4.1) beigelegt. Entsprechende Änderungen werden in Kapitel 3 näher beschrieben.
- Der Quellcode liegt in *C++* vor, als Build-System wird *CMake* verwendet. Zur Übersetzung werden die folgenden Command-Line-Tools empfohlen:

¹⁰ <https://www.gnu.org/licenses/gpl-3.0>

¹¹ Kommunikationsendpunkt bzw. bidirektionale Schnittstelle zur Interprozess- oder Netzwerk-Kommunikation

- CMake ab Version 2.8. *CMake* ist ein Programmierwerkzeug zum Erstellen und Packen von Software.
 - Ein aktueller Compiler wie `gcc` oder `g++`. Auf den meisten Linux-Distributionen ist dieser bereits vorinstalliert.
 - Alternativ ist ein entsprechender *Cross-Compiler* für den *Raspberry Pi* dem Projekt ebenfalls beigelegt und kann zur Kompilation verwendet werden.
- In der aktuellen Version 1.1.0b des BLEd Daemon muss das Programm `bluetoothctl` installiert sein.

4.3 Verwendung bestehender Software und Drittbibliotheken

Der *BLEd-Daemon* baut in Teilen auf bestehende, frei verfügbare Software auf. Dabei wurden einige Grundfunktionalitäten für den `HCICore` aus dem offiziellen Bluetooth Protocol Stack *BlueZ*[44] entnommen und weiterverwendet.

- Für den Daemon selbst wird die, in *BlueZ* [44] enthaltene Library `lib bluetooth` kompiliert, die für die Grundfunktionalitäten hinsichtlich der Kommunikation zwischen dem Daemon und den Host-Controllern verantwortlich ist. Teile aus dem Programm `hcitool`, dieses ist auch eigenständig laufend, sind für diese Zwecke angepasst und für den Daemon entsprechend abgeändert implementiert worden. Verweise der verwendeten Code-Teile finden sich im Quellcode des Projekts (siehe Abschnitt 4.1) und sind speziell gekennzeichnet.
- Für das Verarbeiten von Konfigurationsdateien wird die unter der *GNU Lesser General Public License v2.1* veröffentlichte, frei erhältliche Bibliothek `libconfig`¹² [45] verwendet.
- Die von Google entwickelte, frei erhältliche Bibliothek *Protocol Buffers*¹³ [46] dient zur Serialisierung von Datenstrukturen und eignet sich daher hervorragend zum Austausch, sowie dem Schreiben und Lesen von strukturierten Daten über *Datenstreams* wie Dateien oder Sockets. Die Implementation dieser Bibliothek wurde nicht im Rahmen der Grundfunktionalitäten vorgenommen, sondern stellt eine Erweiterung des Daemons dar. Der Daemon bietet dementsprechend die Möglichkeit, andere Applikationen wie eine HTML-basierte dynamische Konfigurationswebsite anzubinden. Dabei könnte der Austausch zur Interprozesskommunikation über Sockets via *protobuf*-Messages erfolgen. Für das Serialisieren und Deserialisieren der Daten kann zur Gänze die *protobuf*-Bibliothek verwendet werden.

¹² <https://github.com/hyperrealm/libconfig>

¹³ <https://github.com/protocolbuffers/protobuf>

4.4 Kompilation und Installation

Das Projekt wurde für die Debian-basierte Linux-Distribution *Raspbian OS* entwickelt und ist auch auf dieser getestet worden (Abschnitt 5.3). Vorab muss die Daemon-Software kompiliert werden. Für die Inbetriebnahme und Installation der Software sind `sudo`-Berechtigungen erforderlich.

Kompilieren Das Projekt kann mittels der beiliegenden CMake-Datei kompiliert werden. Mittels `CPack` wird anschließend direkt ein installierbares Debian-Package erstellt. Vorab muss die Bibliothek *protobuf*¹⁴(siehe Abschnitt 4.3) installiert sein. Sollte die Bibliothek fehlen, führt dies unweigerlich zum Abbruch des Kompilierprozesses.

Die zugehörige CMake-Datei findet sich unter `./BLEd/CMakeLists.txt`. Folgende Befehlsabfolge ist aufzurufen:

1. `mkdir build && cd build`
2. `cmake ..`
3. `cmake -build .`
4. `cpack`

Waren die Schritte erfolgreich, wird ein Debian-Package mit dem Namen `BLEd.deb` erstellt.

Installation Nach dem erfolgreichen Kompilieren kann die Installation mittels dem Paketemanager `apt` erfolgen. Dabei wird der Befehl `sudo apt-get install -f BLEd.deb` aufgerufen.

Ausführen der Applikation Der Daemon wird über den Systemmanager `systemd` gestartet. Das Starten erfolgt dabei mittels `systemctl`-Befehl: `sudo systemctl start BLEd.service` und das Stoppen über: `sudo systemctl stop BLEd.service`

Alternativ kann die Software auch mittels bereitgestellten Shell-Skripts kompiliert und installiert werden. Die Skriptdatei `./buildRPi.sh` dient zur Kompilation und mittels des Skripts `./installDaemon.sh` wird die Installation am System vorgenommen. Diese genannten Dateien werden im Projektrepository im Ordner `./BLEd` zur Verfügung gestellt.

4.5 Softwarearchitektur

Um den Daemon als performant laufende Applikation betreiben zu können, sind die gesamten Module in der objektorientierten Programmiersprache *C++* verfasst. Das Projekt baut zum Teil auf dem Linux Bluetooth Stack *BlueZ* [44] auf, welcher als *C*-Code vorliegt. Daher gestaltet sich die Portierung der Teile des BlueZ-Quellcodes von *C*-Code zu *C++* als relativ einfach und erfordert nur geringfügigen Aufwand. Zudem war der hohe Bekanntheitsgrad und die daraus resultierende weite Verbreitung dieser Programmiersprache ausschlaggebend für die Wahl.

Ein wesentlicher und bei der Entwicklung stark gewichteter Punkt stellte die Skalierbarkeit dar. Damit soll die Möglichkeit offen gehalten werden, mehrere *Host-Controller* durch nur eine Daemon-Applikation verwalten zu können. Hierzu wurde das `select`-Kommando im Bluetooth Kernelmodul implementiert (Abschnitt 3.3).

Die Applikation ist auf Multithreading ausgelegt, die einzelnen Teile laufen dabei grundsätzlich

¹⁴ <https://github.com/protocolbuffers/protobuf/blob/master/src/README.md>

asynchron ab. Das Suchen nach BLE-Nodes, die Benutzerinteraktion und das Ansprechen der Host Controller erfolgt dabei in eigenen, asynchron ablaufenden Threads. Der Datenaustausch zwischen den Threads wird durch Buffer realisiert. Damit bleibt der Daemon trotz ständigem „pollings“, das heißt dem permanenten Suchen nach *Advertising Packets* (Details dazu im Abschnitt 2.2) und dem parallelen Verarbeiten von Benutzereingaben, reaktiv.

4.5.1 „BLEd“ Daemon

Die Applikation teilt sich in drei Hauptteile: Den **BLEHelper**, den **HCICore** und das **InteractionInterface**, worauf in diesem Abschnitt näher eingegangen werden soll. Die Gliederung erfolgt hierbei nach folgendem Schema:

- Aufgaben, die sich auf die Basisinteraktion mit den Host Controllern bzw. der BLE-Nodes (das Wechseln der Controller, Verbinden und Trennen von Nodes, etc.), aber auch auf das umfassende Verwalten von Verbindungslisten und das Verarbeiten von Benutzerbefehlen beschränken, werden durch eine Helper-Klasse, genannt **BLEHelper**, erledigt.
- Alle Aufgaben die das Ansprechen der Host Controller (wie das Auslesen von Verbindungsinformationen), das Suchen nach neuen BLE-Geräten, das Senden von *Connection-Parameter*, das Akquirieren von Verbindungsstati verbundener Geräte, zum Teil auch das Auslesen der Informationen der Host Controller selbst betreffen, werden durch den **HCICore** bzw. dem **HCIInterface** erledigt.
- Das Entgegennehmen von Befehlen, die der Benutzer an den Daemon senden möchte, wird durch das **InteractionInterface** erledigt.

Auf den nachfolgenden Seiten wird das Verhalten der Applikation beschrieben. Dazu wird zunächst der Ablauf in allgemeiner Art und Weise dargestellt, um ein Bild des Prozesses zu erhalten. Die einzelnen Software-Teile werden dann im Laufe des Kapitels vorgestellt. Auf Grund des Umfangs werden dem Leser nur wichtige Gesichtspunkte der Applikation erläutert.

4.5.2 Allgemeiner Ablauf

Der Daemon wurde als Multithreading-Applikation ausgelegt. Zunächst wird die Konfigurationsdatei (Abschnitt 4.5.4) geladen. Diese enthält Informationen zu den einzelnen BLE-Nodes und zur Konfiguration des Daemons. Da der BLEd-Daemon grundsätzlich ohne Interaktion des Benutzers starten soll, können wichtige Einstellungen in Konfigurationsdateien angegeben werden. Ein Schema dieser Datei findet sich in Abschnitt 4.5.4.

Nachdem die in der Konfigurationsdatei gesetzten Parameter übernommen wurden, wird pro *Host Controller* eine Instanz eines **BLEHelpers** (Helpers) erzeugt. Diese Instanz stellt das Hauptprogramm dar und übernimmt alle weiteren Schritte im Ablauf (Abschnitt 4.5.3).

Je Helper-Instanz wird ein Thread des **HCICore** (Abschnitt 4.5.5) erzeugt, welcher permanent nach neuen (gültigen) BLE-Nodes sucht. Wird eine Node gefunden, wird die Information dem Helper mitgeteilt. Dieser ist dann unter Zuhilfenahme des **NodeHandler** für die Verwaltung der Nodes verantwortlich.

Schema der Applikation

Der schematische Aufbau der BLEd-Applikation ist zum leichteren Verständnis in Abbildung 4.1 dargestellt. In diesem Schema wurde von zwei angeschlossenen Bluetooth-Controllern ausgegangen. Diese sind durch die beiden Host Controller `hci0` und `hci1` repräsentiert. Für jeden am Host verfügbaren Host Controller wird eine neue Instanz des `BLEHelpers` angelegt. Er kommuniziert mit dem `HCICore` und dem `InteractionInterface`, nicht jedoch mit anderen `BLEHelpers`.

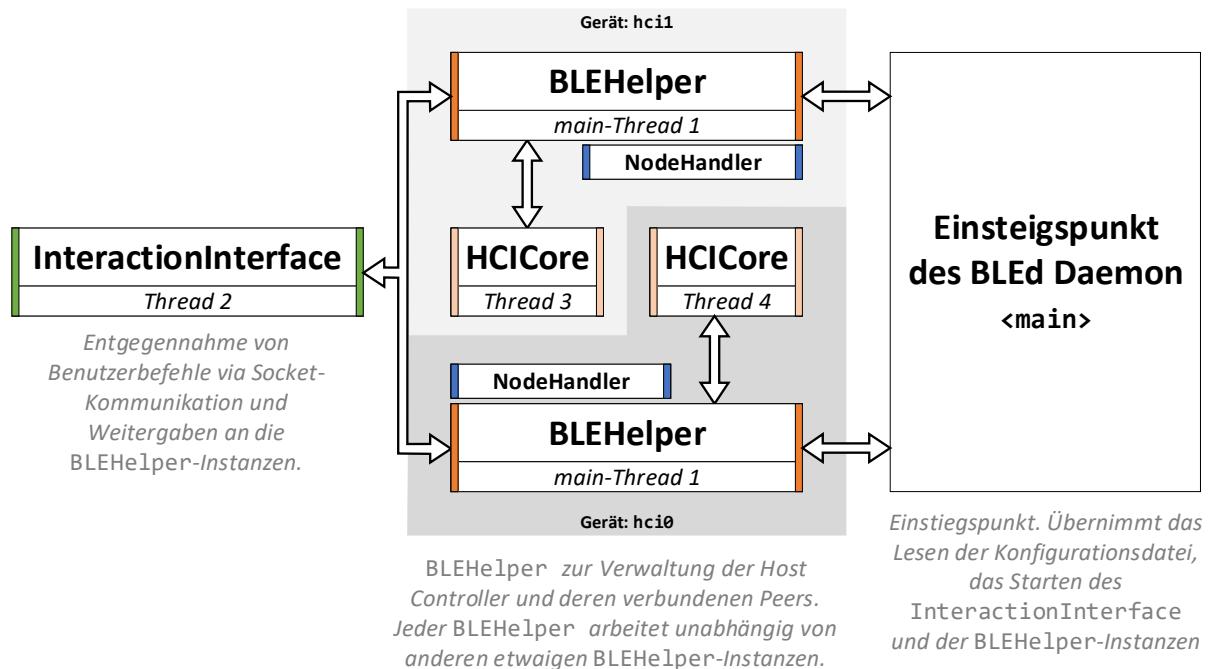


Abbildung 4.1: Schema der BLEd-Applikation mit zwei BLEHelper-Instanzen

Betriebsmodi

Der BLEd-Daemon unterstützt zwei grundsätzliche Verbindungsmethoden der BLE-Nodes:

- Automatisches Verbinden
- Manuelles Verbinden

In der Konfigurationsdatei kann einer Node ein bestimmter Host Controller als Standard-Gerät zugewiesen werden, wobei dessen Hardwareadresse als eindeutige Identifizierung dient. Dazu wird in der Konfigurationsdatei die gewünschte Nummer des HCIs eingetragen. Wird nun eine Node mit entsprechender Hardware Adresse gefunden, wird versucht, diese mit dem entsprechenden BLE Controller zu verbinden. Wird keine Information (in der Konfigurationsdatei) gefunden, die dem Host Controller eine Node zuweist, so wird in weiterer Folge ein vorgegebener Standard-Host Controller gewählt (Abschnitt 4.5.4).

Ist mindestens ein BLE-Gerät mit dem Border Router verbunden, wird periodisch (im Abstand von ≤ 1 s) dessen Verbindungsstatus überprüft.

4.5.3 BLEHelper

`./BLEd/src/BLEHelper.h`

Die BLEHelper Klasse dient der Verwaltung verbundener Nodes. Der Aufgabenbereich umfasst das Verbinden oder Trennen der Verbindung. Dabei interagiert der Helper mit dem HCICore. Nachfolgend ist in Abbildung 4.2 das Flussdiagramm des BLEHelpers dargestellt.

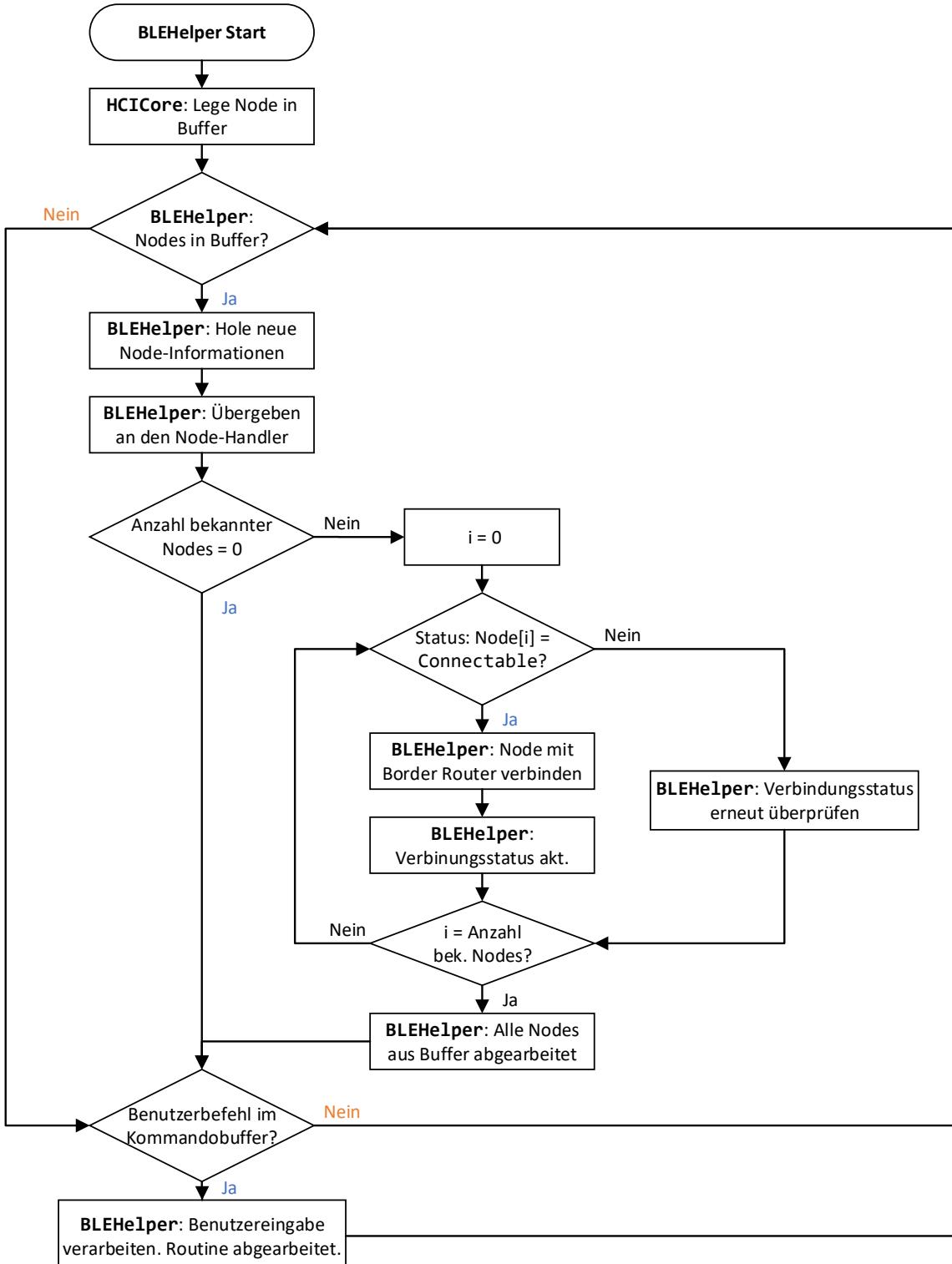


Abbildung 4.2: Flussdiagramm des BLEHelpers

Sind mehrere Host Controller verfügbar, so wird pro Controller eine Instanz des Helpers angelegt. Diesem Umstand entsprechend ist die Anzahl der Helper-Instanzen gleich der Anzahl der zu verwaltenden Host Controller.

Eine **BLEHelper**-Instanz startet eine Instanz des **HCICores**, wobei mithilfe dieser Instanz nach verfügbaren BLE-Geräten gesucht wird.

Wird ein BLE-Gerät durch die **HCICore**-Suche erkannt, wird ein Verbindungsversuch initiiert. Anschließend werden die Verbindungsinformationen des verbundenen Geräts aktualisiert. Dabei findet auch gleichzeitig eine Aktualisierung aller anderen, etwaig verbundenen BLE-Geräte statt.

Im letzten Schritt werden noch die Benutzereingaben verarbeitet. Die Kommunikation zwischen dem **InteractionInterface** und den **BLEHelpers** findet asynchron statt. Um Benutzereingaben „transportieren“ zu können, dient die Klasse **ICommand** als „Kapsel“.

Vom **InteractionInterface** werden diese „**ICommand**-Kapseln“ angelegt, die dabei die Daten aus der Benutzereingabe enthalten, und in einen gemeinsamen Buffer gelegt. Auf diesen Buffer können alle **BLEHelpers**-Instanzen zugreifen und die **ICommands** lesen und verarbeiten.

Verbinden von BLE-Nodes

Eine Helper-Instanz übernimmt anfänglich auch alle notwendigen Initialisierungen des HCIs und startet auch den **HCICore**. Das heißt, **BLEHelper** und **HCICore** werden simultan initialisiert.

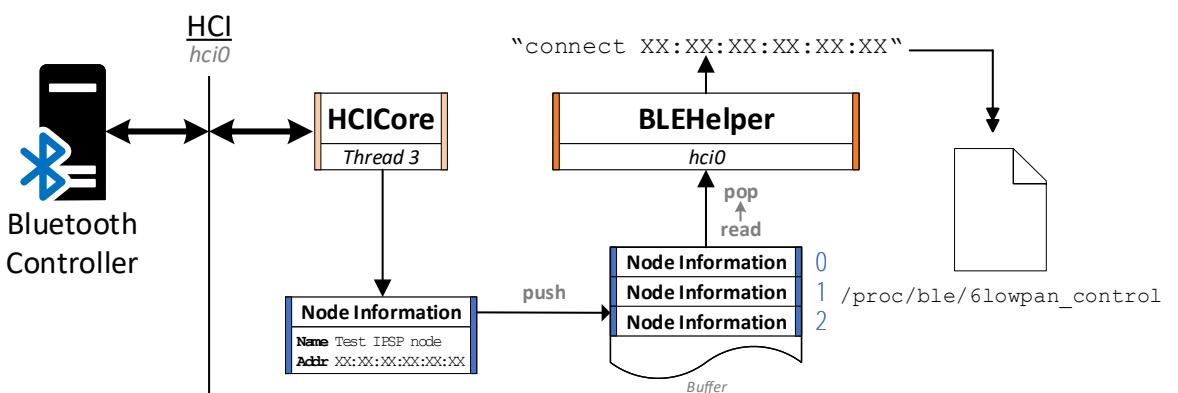


Abbildung 4.3: Interaktion zwischen **BLEHelper** und **HCICore**

Im ersten Schritt wird durch den Helper überprüft, ob der **HCICore** neue Nodes gemeldet hat, die verbunden werden können. Da Helper und die (**HCICore**-)Suche asynchron ablaufen, erfolgt die Kommunikation zwischen den beiden Klassen ebenfalls über Buffer (Abb. 4.3). Wird durch die Suche ein BLE-Gerät gefunden, das in seinem *Advertising Packet* die UUID für die Unterstützung des *Internet Protocol Support Service 0x1820* enthält, wird der **BLEHelper** durch den **HCICore** (über diese Buffer) darüber benachrichtigt, dass es BLE-Geräte mit IPv6-Unterstützung gibt, die eine Verbindung mit dem Border Router (6LBR) initiieren wollen.

Die Verbindungsinformation in diesem Buffer enthält neben der Bluetooth Adresse des BLE-Geräts auch ein Feld für dessens aktuellen Verbindungsstatus. Dieses Feld hat jedoch rein informativen Charakter und dient nur zur Verwaltung durch den Daemon. Standardmäßig hat ein neues, noch nicht mit dem Router verbundenes BLE-Gerät den Status **CONNECTABLE**.

Diese neue, noch nicht verbundene BLE-Node wird zu einer Verwaltungsklasse (dem `NodesHandler`) hinzugefügt. Ähnlich einer Liste sind darin alle *verbundenen* und *vormals verbundenen/nicht mehr verbundenen* Geräte enthalten. Zusätzlich werden Informationen der einzelnen Nodes wie der *Handle*, *Name*, etc. darin abgelegt.

Verbundene Geräte in der Liste haben den Status `CONNECTED` und vormals verbundene `DISCONNECTED`. Geräte, die zwar verfügbar sind, aber nicht verbunden werden sollen, sind mit dem Status `DISABLED` gekennzeichnet. Geräte die gefunden, aber noch nicht verbunden wurden, haben den Status `CONNECTABLE`.

Die Informationen der Nodes werden pro Schleifendurchlauf aktualisiert. Das gilt für jede, in der Liste befindliche Node.

Findet der Helper eine neue BLE-Node in der Liste, für die noch kein Verbindungsversuch unternommen wurde (diese befindet sich somit noch auf dem Status `CONNECTABLE`), so wird ein Verbindungsevent angestoßen.

Dazu kommuniziert der Daemon über die im Kapitel 3 erwähnte Datei `6lowpan_control` mit dem 6LoWPAN Kernelmodul (Abb. 4.3). Hierfür wird der entsprechende Befehl in die 6LoWPAN-Kontrolldatei `/proc/ble/6lowpan_control` geschrieben. Prinzipiell unterstützte Befehle sind `connect` und `disconnect`, wobei letzterer nicht fehlerfrei funktioniert.

In der Version *0.2.3b* ist zudem das Auswählen des gewünschten HCI-Geräts möglich, welches essentiell für den Betrieb des Daemons ist. Zu diesem Zweck wurde im Rahmen dieser Arbeit der `select`-Befehl implementiert, der in der oben genannten Version vom 6LoWPAN Kernelmodul als validier Befehl akzeptiert wird. Eine nähere Beschreibung zu dem `select`-Befehl findet sich in Abschnitt 3.3.

Durch den Daemon wird vor dem Verbinden, entsprechend der Angabe in der Konfigurationsdatei, der Host-Controller gewechselt, wofür der `select`-Befehl in folgender Form in die 6LoWPAN-Kontrolldatei geschrieben wird:

```
„select XX:XX:XX:XX:XX:XX“ > /proc/ble/6lowpan_control  
XX:XX:XX:XX:XX:XX steht dabei für die Adresse des Host Controllers.
```

Das Verbinden der BLE-Node erfolgt anschließend durch das Kommando:

```
„connect XX:XX:XX:XX:XX:XX“ > /proc/ble/6lowpan_control  
XX:XX:XX:XX:XX:XX steht dabei für die Adresse der BLE-Node.
```

Ist eine gültige Verbindung zustande gekommen und besteht ein Link zwischen dem BLE-Gerät und dem Border Router, so werden die Verbindungsinformationen aktualisiert. Diese Informationen werden unter Zuhilfenahme des `HCICore` direkt über das Host Controller Interface ausgelesen. Dazu zählen unter anderem *Handle* oder *State*.

Erst bei einem gültigen *Handle* (d. h. $\neq 0$), wird der Status der BLE-Node von `CONNECTABLE` auf `CONNECTED` gesetzt. Damit gilt die BLE-Node als vollständig mit dem Border Router verbunden.

Verarbeiten der Benutzereingaben

Die Helper-Instanz übernimmt neben den vorhin genannten Funktionen auch die des Verarbeitens von Befehlen. Diese werden von einem Benutzer über eine Socket-Verbindung dem Daemon mitgeteilt. Die Entgegennahme der Eingabe erfolgt dabei asynchron, sodass kein Blockieren stattfindet. Es ist jedoch möglich, die Eingabe gezielt zu blockieren, bis der **BLEHelper** diese abgearbeitet hat. Prinzipiell findet diese Option jedoch nur dann Anwendung, wenn dem Benutzer eine Meldung zurückgegeben werden soll. Bei Befehlen ohne Rückgabe bleibt die Eingabemaske reaktiv und es können auch vor Verarbeitung alter Befehle neue gesendet werden.

Über das **InteractionInterface** (Abschnitt 4.5.6) werden die vom Benutzer eingegebenen Befehle verarbeitet und in **ICommands** gekapselt.

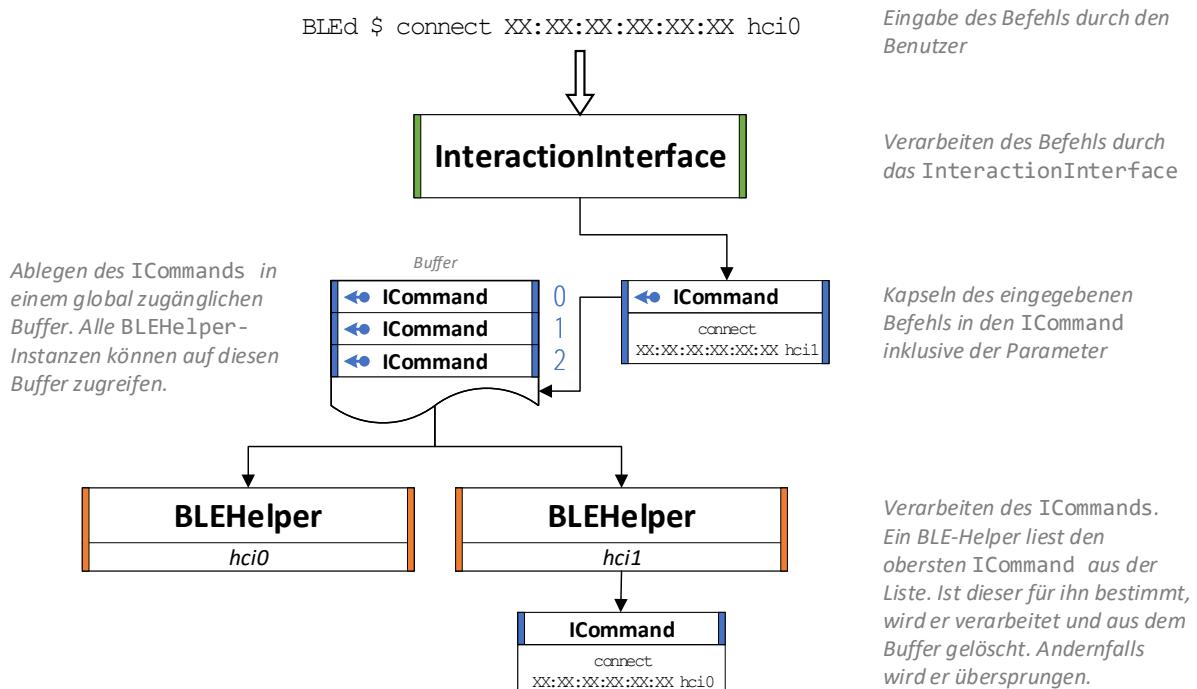


Abbildung 4.4: Abfolge der Befehlsabarbeitung am Beispiel des `connect`-Befehls

Als vereinfachte Darstellung findet sich in Abbildung 4.4 das Schema zur Abarbeitung eines Befehls. Nachdem der Benutzer seine Eingabe getätigt hat, das heißt der Befehl „abgeschickt“ wurde, wird eine Klasse des **ICommand** zur Kapselung dieses Benutzerbefehls erstellt. Diese Instanz enthält übergebene Informationen zu dem Befehl und dessen Parameter.

Diese „Befehls-Kapsel“ wird in weiterer Folge in einen, für alle **BLEHelper** zugänglichen Buffer gelegt. Dies ist notwendig, weil von vornherein nicht zwangsläufig festgelegt sein muss, für welche Helper-Instanz der Befehl bestimmt ist. Daher bietet die **ICommand**-Klasse das Setzen eines Attributs, um zu differenzieren, ob Befehle direkt an **BLEHelper**-Instanzen adressiert, oder als allgemeine Befehle in den Buffer zur Verarbeitung gelegt werden sollen.

Handelt es sich um einen „allgemeinen“ Befehl, werden diese von allen **BLEHelper** zumindest auf Verarbeitbarkeit überprüft. Die Abarbeitung durch die Helper-Instanzen erfolgt nach dem *First-Come-First-Serve* Prinzip.

Bei dieser Art wird ein Befehl aus dem Buffer geholt und durch den Helper überprüft, ob dieser für ihn (den Helper) bestimmt ist. Ist der Befehl durch den **BLEHelper** verarbeitbar, wird er interpretiert und aus dem Buffer gelöscht. Damit steht der Befehl nicht mehr für andere Helper-Instanzen zur Abarbeitung bereit.

Kann der Befehl durch keine Helper-Instanzen abgearbeitet werden oder war er nicht für diese

Instanz bestimmt, so verbleibt der Befehl im Buffer bis alle Helper den Befehl mindestens einmal durchlaufen haben. Nach erfolglosem Durchlaufen aller Instanzen wird der `ICommand` ebenfalls permanent aus dem Buffer gelöscht.

Da jedem Host-Controller eine Helper-Instanz zugewiesen ist, können Befehle ebenso direkt an eine bestimmte Helper-Instanz adressiert werden. Dies ist am Beispiel des `connect`-Befehls in Abbildung 4.4 schematisch dargestellt. Dabei erfolgt die Identifizierung über die Nummer des zugehörigen Host Controllers (zum Beispiel `hci0` oder `hci1`).

Der Befehl wird nur von der korrespondierenden Instanz berücksichtigt und auch nur von dieser verarbeitet. Andere Helper ignorieren den Befehl. Allgemeinen wird der Befehl aus dem Buffer gelöscht, wenn dieser nicht interpretiert werden kann oder fehlerhafte Informationen enthält.

Eine Liste an Befehlen findet sich im Abschnitt 4.5.6.

4.5.4 Lesen der Konfiguration

`./BLEd/src/ConfigHandler.h`

Direkt nach dem Start des Daemons wird die Konfigurationsdatei gelesen, wobei dieser Schritt nicht obligatorisch ist. Wird eine fehlerhafte Konfiguration erkannt oder keine Konfigurationsdatei gefunden, überspringt der Daemon diesen Schritt. In diesem Fall wird eine Warnung in die Log-Datei geschrieben und als Standard-HCI `hci0` gewählt.

Wird jedoch eine Konfigurationsdatei verwendet, so muss eine gültige Konfiguration auf jeden Fall folgende Felder enthalten:

```
1  version = "1.0";
2
3  hci_config = {
4      hci_devices = ["hci0", "hci1"];
5      default_hci_device = "hci0";
6      autoconnect = true;
7      socket_port = 11111;
8  };
9
10 node_config = {
11     nodes = ( { bd_address = "FE:8D:E4:E3:19:69";
12                 bd_address_type = "public";
13                 target_hci = 0;
14                 conn_params = "" ; },
15
16         { bd_address = "D8:74:EA:0B:F1:ED";
17             bd_address_type = "public";
18             target_hci = 1;
19             conn_params = "connection_parameter_1" ; }
20     );
21 }
22
23 connection_parameter_1 = {
24     conn_interval_min = 200;
25     conn_interval_max = 200;
26     conn_latency = 2;
27     supervision_timeout = 3000;
28     minimum_ce_length = 400;
29     maximum_ce_length = 400;
30 }
```

Script 4.1: Beispiel Konfigurationsdatei

Konfigurationsfelder

Die Konfigurationsfelder umfassen folgende Einträge.

- Das Feld `hci_config` enthält Einstellungsparameter, die den *Daemon* betreffen. Dazu zählen die Einträge:

`hci_devices` Ein Array welches die *Host Controller* enthält. Alle HCIs, welche in diesem Feld angegeben sind werden bei Start des Daemons initialisiert.

`default_hci_device` Setzen des Standard-*Host Controllers*. BLE-Nodes, zu welchen keine gültigen Einstellungsparameter (unter `node_config`) gefunden werden, werden automatisch mit diesem *Host Controller* verbunden, sofern die Option `autoconnect` gesetzt wurde.

`autoconnect` Gibt an, ob BLE-Nodes mit IPv6-over-BLE Funktionalität automatisch mit dem Daemon verbunden werden sollen. Ist diese Option auf `false` gesetzt, muss das Verbinden manuell erfolgen (Abschnitt 4.5.6, `connect`-Befehl).

`socket_port` Port für den Socket zur Kommunikation über das `InteractionInterface`.

- Das Feld `node_config` beschreibt Konfigurationen für BLE-Geräte. Dabei ist die Hardwaredresse eines BLE-Geräts der eindeutige Schlüssel zur Identifikation. In diesem Fall sind folgende Parameter möglich:

`bd_address` Der eindeutige Wert zur Identifikation der BLE-Node erfolgt über die Hardwaredresse.

`bd_address_type` Typ der Hardwaredresse

`target_hci` Sind mehrere Host Controller vorhanden, kann mit dem Feld `target_hci` angegeben werden, mit welchem BLE Controller die BLE-Node verbunden werden soll.

`conn_params` Kann einen Verweis zu einem `connection_parameter`-Konfigurationsfeld (im Beispiel `connection_parameter_1`) enthalten. Das Feld, auf das verwiesen wird, enthält dabei die Werte für die *Link-Layer Connection Parameter*, mit welchen die BLE-Node initialisiert werden soll.

Dazu kann ein neues Konfigurationsfeld mit beliebigen Namen, jedoch den festgelegten Werten `conn_interval_min`, `conn_interval_max`, `conn_latency`, `supervision_timeout`, `minimum_ce_length` und `maximum_ce_length` angelegt werden. Der gewählte Name des Felds wird als Wert gesetzt.

Soll eine BLE-Node nicht mit initialen *Connection Parameter* verbunden werden, muss dieses Feld `conn_params` in der `node_config` dennoch enthalten sein. Der Wert ist jedoch optional und kann daher leer gelassen werden. In diesem Fall sind zwei doppelte Anführungszeichen ohne Inhalt anzugeben (Codelisting 4.1, Zeile 14).

4.5.5 HCICore

`./BLEd/src/HCICore/HCIInterface.h`

Die Interaktion mit den HCI-Geräten sowie die Suche nach *Advertising Packets* erfolgt durch den **HCICore**, eine auf dem Linux Bluetooth Stack BlueZ [44] basierende Klasse. Diese stellt eine im eigenen Thread ablaufende Instanz für jedes HCI-Gerät bereit.

Entsprechende Codeteile die adaptiert und verändert wurden, sind im Quellcode des Projekts (Abschnitt 4.1) kenntlich gemacht. Nachfolgend werden die wichtigsten Funktionen dieser Klasse aufgelistet:

permanentScan(...) Startet die Suche nach verfügbaren BLE-Nodes (6LN). Werden neue Nodes mit der UUID *IP Support Service* (0x1820) gefunden, werden sie in einen Buffer geschrieben. Die **BLEHelper** Instanz arbeitet den Buffer ab und versucht eine Verbindung zwischen der Node und dem IPv6-over-BLE Border Router (6LBR) herzustellen.

sendCommand(...) Das Übertragen von LE Commands (als hexadezimale Werte) an die Nodes.

updateStateOfNode(...) Aktualisiert den Verbindungsstatus einer Nodes.

acquireDeviceAddress(...) Gibt die Hardwareadresse des Host Controllers zurück.

Da es sich bei **HCICore** um eine Adaption des Tools aus dem Bluetooth Protocol Stack *BlueZ* handelt, ähnelt die Funktionsweise dem Command-Line-Tool **hcitool**. Zur Kommunikation mit anderen Klassen sind des Weiteren folgende Funktionen implementiert:

updateStateOfNode(...) Liest die Informationen zu einer gegeben BLE-Node aus. Dazu zählen unter anderem *Handle*, *Type*, *State* und *Link Mode*.

getNextDiscoveredNodes(...) Gibt die nächste in einem Buffer befindliche BLE-Node Information zurück.

4.5.6 InteractionInterface

`./BLEd/src/InteractionInterface.h`

Wie bereits in der Einleitung beschrieben, bieten Daemon-Applikationen nur die Möglichkeit der indirekten Interaktion. Da eine der Hauptfunktionen darin besteht, den Daemon auch im laufenden Betrieb konfigurieren zu können, ist eine Kommunikationsmöglichkeit implementiert. Wie bei bekannten Routern sollen neben Grundfunktionalitäten, beispielsweise dem Verbinden und dem Trennen von BLE-Nodes, weitere spezifischere Einstellungsmöglichkeiten geboten werden. Ein Anwendungsfall ist das Ändern von *Connection Parametern*. Zusätzlich gibt es die Möglichkeit, die Nodes im laufenden Betrieb einem (sofern verfügbaren) anderen Host Controller zuzuweisen oder deren Verbindung gänzlich zu trennen.

Die Aufgaben zur Entgegennahme von Befehlen wird durch die Klasse **InteractionInterface** erledigt. Eine Instanz öffnet einen Kommunikationsendpunkt, die Kommunikation selbst erfolgt über Sockets. Dabei werden mehrere simultane Socket-Verbindungen unterstützt. Der Port wird in der Konfigurationsdatei angegeben (Abschnitt 4.5.4) und kann vom Benutzer frei gewählt werden.

Zur Interaktion mit dem Daemon können Command-Line-Tools, wie die Netzwerk-Utilities zur TCP-Kommunikation *netcat* und *telnet*, genutzt werden.

Vorteil: Über den Socket kann ein bidirektonaler Datenaustausch stattfinden, sodass spezifischer mit Meldungen auf Benutzereingaben reagiert werden kann.

Sockets eignen sich daher hervorragend zum Austausch von Daten zwischen verschiedenen Anwenderprogrammen über das Netzwerk oder lokal. Eine mögliche Erweiterung der Socket-Applikation wäre eine webbasierte Konfigurationsseite, die eine grafische Einstellungsmöglichkeit des Daemons bieten könnte. Dazu wurde bereits durch die Integration der Bibliothek *protobuf* [46] die Fähigkeit implementiert, Daten des Daemon zu serialisieren (siehe Kapitel 5).

Benutzerbefehle

Vom Daemon werden die nachfolgend aufgelisteten, interaktiven Befehle unterstützt:

connect <bd-address> <hci-dev> Ist der Daemon so konfiguriert, dass Verbindungen nicht automatisch aufgebaut werden, so können BLE-Nodes mittels des Kommandos **connect <bd-address> <hci-dev>** manuell verbunden werden. Dabei wird vorausgesetzt, dass die gegebene BLE-Node via Advertising Events mitteilt, dass sie den *Internet Protocol Support Service* unterstützt und somit durch die HCICore-Suche als gültige BLE-Node erkannt wird. Waren Nodes bereits verbunden und wurde die Verbindung (z. B. durch einen vorhergehenden **disconnect**-Befehl) getrennt, können diese erneut mit dem Daemon verbunden werden, wenn sie vom **HCICore** als gültig erkannt werden.

disconnect <bd-address> Ist eine BLE-Node mit dem Router verbunden, trennt dieser Befehl die Verbindung. Zudem wird ein Vermerk erstellt („Blacklisting“), der die BLE-Node erst durch Aufruf des **connect**-Befehls wieder mit dem Router verbindet.

remap <bd-address> <target-hci> Der **remap**-Befehl ändert die Zuordnung einer BLE-Node mit der Adresse *bd-address* zu einem Host Controller mit der Nummer *target-hci*. Dementsprechend ist die Verwendung dieses Befehls nur bei mehreren Host Controllern sinnvoll. *target-hci* gibt dabei den gewünschten Controller an. Der **Remap**-Befehl verhält sich dabei wie ein **disconnect**- mit anschließendem **connect**-Befehl, wobei der Host-Controller für diese BLE-Node zuerst geändert und anschließend die Node mit diesem verbunden wird.

list <hci-device> Der **list**-Befehl gibt eine Liste von verbundenen Nodes eines Host Controllers aus (Abbildung 4.5). Dabei werden nur die mit dem angegebenen Controller *hci-device* verbundenen Nodes berücksichtigt. Grundsätzlich werden Informationen zur *Hardwareadresse* die BLE-Node, dem *Handle* und dem *Status* ausgegeben.

Address	Name	Handle	State
EC:A1:6C:F6:17:E0	Test IPSP node	64	1
FE:8D:E4:E3:19:69	Test IPSP node	65	1

Abbildung 4.5: Anzeigen verbundener BLE-Nodes

con-update <hci-device> Um die *Connection Parameter* einer BLE-Node ändern zu können, stellt der **con-update**-Befehl ein Interface zu Verfügung. *hci-device* gibt dabei an, welcher Host-Controller angesprochen werden kann. Dabei wird ein Assistent gestartet, welcher als Eingabe die Connection-Parameter erwartet. Die gewünschte BLE-Node wird über deren *Handle* ausgewählt. Der *Handle* einer BLE-Node kann über das **list**-Kommando ausgegeben werden. Die Einheit der Eingaben erfolgt je nach Kontext entweder in ms (Millisekunden) oder als hexadezimaler Wert.

```
pi@raspberrypi:~ $ netcat localhost 11111
BLEd Daemon v 1.0.11(beta)
BLEd $ con-update 0
Handle: 64
Connection Interval Min (ms): 100
Connection Interval Max (ms): 100
Conn Latency (hex): 2
Supervision Timeout (ms): 3000
Minimum CE Length (hex): 200
Maximum CE Length (hex): 200
BLEd $
```

Abbildung 4.6: Ändern der Connection Parameter

Die Spezifikation für *LE Connection Update Command* [3, S. 1348] schreibt folgende Parameter vor:

- **Connection_Handle** Eingabe als dezimaler Wert. Der Handle kann über den Befehl **list** ausgegeben werden.
- **Conn_Interval_Min**: Der Minimalwert für das *Connection Interval*. Der Wert muss kleiner oder gleich dem Maximalwert des *Connection Intervals* sein.

$$N = \frac{\text{Input}}{1.25\text{ms}} \quad \text{Input} \hat{=} \text{Werteeingabe in [ms]}$$

Wertebereich: 0x0006 bis 0x0C80 bzw. 7.5 ms bis 4s.

- **Conn_Interval_Max**: Der Maximalwert für das *Connection Interval*. Der Wert muss größer oder gleich dem Minimalwert des *Connection Intervals* sein.

$$N = \frac{\text{Input}}{1.25\text{ms}} \quad \text{Input} \hat{=} \text{Werteeingabe in [ms]}$$

Wertebereich: 0x0006 bis 0x0C80 bzw. 7.5 ms bis 4s.

- **Conn_Latency**: Werteeingabe als hexadezimaler Wert im Wertebereich. Die Latenz des Slaves (der BLE-Nodes). Umso geringer dieser Wert gewählt wird umso schneller können Daten übertragen werden. Wertebereich: 0x0000 bis 0x01F3
- **Supervision_Timeout**: Supervision Timeout für den LE Link. Bestimmt das Timeout, ab wann ein Link nach dem letzten Datenaustausch als unterbrochen gilt.

$$N = \frac{\text{Input}}{10\text{ms}} \quad \text{Input} \hat{=} \text{Werteeingabe in [ms]}$$

Wertebereich: 0x000A bis 0x0C80 bzw. 100 ms bis 32s .

- **Minimum_CE_Length:** Informationsparameter über die minimale Länge eines *Connection Events*.

$$N = \frac{\text{Input}}{0.625\text{ms}} \quad \text{Input} \hat{=} \text{Werteeingabe in [ms]}$$

Wertebereich: 0x0000 bis 0xFFFF.

- **Maximum_CE_Length:** Informationsparameter über die maximale Länge eines *Connection Events*.

$$N = \frac{\text{Input}}{0.625\text{ms}} \quad \text{Input} \hat{=} \text{Werteeingabe in [ms]}$$

Die Interaktion ist möglich, sobald die Klasse `InteractionInterface` instanziert wurde. Weitere Initialisierungen der Klasse sind nicht notwendig.

4.5.7 Sonstige Module

Funktionale Teile, die durch verschiedene Klassen verwendet werden, sind in der Datei `./BLEd/src/UtilityClass.h` zusammengefasst.

Logger

`./BLEd/src/Logger.h`

Das Schreiben in eine Log-Datei wird durch die Klasse `Logger` übernommen. Dabei erfolgt der Zugriff über die Funktion `write(...)`. Es kann zwischen sechs Log-Level differenziert werden: `FATAL_MSG`, `ERROR_MSG`, `WARNING_MSG`, `SYSTEM_MSG`, `INFO_MSG`, `DEBUG_MSG`

5

Evaluierung

Dieser Abschnitt befasst sich mit der Evaluierung der Arbeit.

5.1 IPv6 Konnektivität

Der Abschnitt 3.2 in Kapitel 3 befasst sich mit der Problematik mehrerer verbundener BLE-Geräte, wobei auch eine Problemlösung (Abschnitt 3.2.2) angeboten wird. Diese wird nachfolgend evaluiert.

Der Test umfasst damit die Verifizierung der IPv6 Konnektivität. Das heißt, es wird überprüft, ob mehrere BLE-Geräte im Netzwerk erreicht werden können. Dazu sollen *ICMPv6* Pakete (*Pings*) an BLE-Nodes gesendet werden. Bei korrekter Funktionsweise antworten alle Nodes auf Pakete.

5.1.1 Setup

Der Test wurde auf einem *Raspberry Pi 3* mit *Raspbian GNU/Linux 9 (stretch)* in der Kernelversion *4.14.95-v7*, welcher als *IPv6-over-BLE* Border Router (6LBR) fungiert, durchgeführt. Als Bluetooth-Controller wurde der interne Bluetooth Adapter des Raspberry Pi verwendet. Als *IPv6-over-BLE* Node (6LN) wurden die Development-Boards *Particle Xenon* (Kapitel 2.5.3) genutzt. Ziel war es, beide Boards zeitgleich mit dem Border Router zu verbinden (Abbildung 5.1). Beim Aufsetzen des Raspberry Pi als Border Router, hat sich der Autor großteils an die entsprechende Anleitung von Nordic Semiconductors [47] gehalten. Diese ist nachfolgend dargelegt.

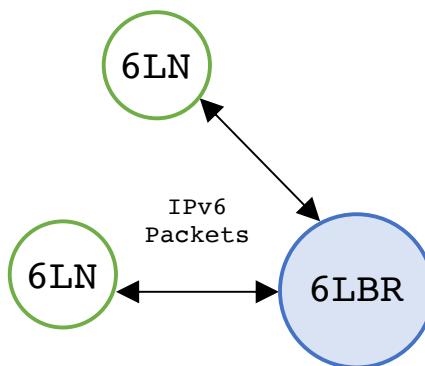


Abbildung 5.1: Schematische Aufbau des Testsetups zum Testen der IPv6 Konnektivität

Die in Kapitel 3 beschriebenen Änderungen am Bluetooth Kernelmodul (*6lowpan.c*) wurden dabei bereits angewandt. Zum Aktivieren und Kompilieren des Kernelmoduls wurden die Kommandozeilen-Scripts *./buildKernel.sh* und *./copy2sd.sh* verwendet, die den Projektdateien beiliegen.

Um das Kernelmodul zu kompilieren, sind folgende Schritte ausgeführt worden:

- Das Kernelmodul in der *config* aktivieren.

Dazu ist die Script-Datei `./kernelConfig.sh` aufzurufen. Im darauffolgend geöffneten Konfigurationsfenster unter **Networking support → Bluetooth subsystem support → Bluetooth Low Energy (LE) features** ist das Modul **Bluetooth 6LoWPAN support** zu aktivieren.

- Kompilieren des Kernels mit Hilfe der Skript-Datei `./buildKernel.sh`.
- Kopieren des Kernels auf die SD-Karte mit Hilfe der Skript-Datei `./copy2sd.sh <device>`. Der Parameter `<device>` ist der Gerätename der SD-Karte und kann mittels dem Kommandozeilenbefehl `lsblk` ermittelt werden.

Die verwendete Anleitung von Nordic Semiconductors [47] wurde leicht abgeändert:

- Laden des 6LoWPAN Moduls

```
sudo su                                # Log in as a root user.
modprobe bluetooth_6lowpan # Load 6LoWPAN module.
```

- Das 6LoWPAN Kernelmodul aktivieren

```
echo 1 > /proc/ble/6lowpan_enable
```

- Das Host Controller Interface resetten und nach verfügbaren BLE-Nodes suchen

```
hciconfig hci0 reset # Reset HCI device - in this case hci0 device.
hcitool lescan      # Read 00:11:22:33:44:55 address of the nRF5x device.
```

- Die gefundenen BLE-Nodes verbinden.

Die Syntax lautet `connect <BD_ADDR> <BD_ADDR_TYPE>`

```
echo "connect 00:11:22:33:44:55 2" > /proc/ble/6lowpan_control
```

Testen

Nachdem zwischen **6LBR** und **6LN** eine Link-Layer Verbindung hergestellt wurde, können IPv6-Pakete übertragen werden. Das Testen der Verbindung erfolgt über das Senden von *ICMPv6*-Paketen (*pings*). Dazu wurde das Kommandozeilenprogramm `ping6` verwendet. Mit diesem Programm können *ICMPv6 Echo Request*-Pakete an Netzwerk-Hosts gesendet werden.

`ping6` verlangt als Parameter neben der Zieladresse auch die Angabe des Netzwerkinterfaces. Dieser kann über den Kommandozeilenbefehl `ifconfig` ermittelt werden. Bei einem verfügbaren Host Controller erhält das gesuchte Interface den Namen `bt0`. Eine beispielhafte Ausgabe des `ifconfig`-Befehls mit zwei `bt`-Interfaces ist in Abbildung 5.8 gezeigt.

Die Verbindung wurde im ersten Schritt über die die Multicast-Adresse `ping6 -I bt0 ff02::1` erfolgreich verifiziert (Abb. 5.2).

```
pi@raspberrypi:~ $ ping6 -I bt0 -c3 ff02::1
PING ff02::1(ffff:fe00:ff00:0:0:0:0:1) from fe80::b827:ebff:fe40:9b06%bt0 bt0: 56 data bytes
64 bytes from fe80::b827:ebff:fe40:9b06%bt0: icmp_seq=1 ttl=64 time=0.152 ms
64 bytes from fe80::d874:ea:fe0b:f1ed%bt0: icmp_seq=1 ttl=64 time=329 ms (DUP!)
64 bytes from fe80::e256:67ff:feb1:edb%bt0: icmp_seq=1 ttl=64 time=359 ms (DUP!)
64 bytes from fe80::b827:ebff:fe40:9b06%bt0: icmp_seq=2 ttl=64 time=0.120 ms
64 bytes from fe80::d874:ea:fe0b:f1ed%bt0: icmp_seq=2 ttl=64 time=328 ms (DUP!)
64 bytes from fe80::e256:67ff:feb1:edb%bt0: icmp_seq=2 ttl=64 time=358 ms (DUP!)
64 bytes from fe80::b827:ebff:fe40:9b06%bt0: icmp_seq=3 ttl=64 time=0.111 ms

--- ff02::1 ping statistics ---
3 packets transmitted, 3 received, +4 duplicates, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.111/196.563/359.641/170.498 ms
pi@raspberrypi:~ $
```

Abbildung 5.2: Antwort beider BLE-Peers auf „pings“

Beide BLE-Nodes antworten nach Anwendung der in Kapitel 3.3.2 genannten Änderungen auf die *pings*. Durch das „pingen“ einer spezifischen Node über die Lokal-Link-Adresse mittels `ping6 -i bt0 fe80::211:22ff:fe33:4455` konnte das Ergebnis wiederholt verifiziert werden. Aus diesem Verhalten wurde geschlossen, dass die Verbindungen zwischen Router und den Nodes korrekt aufgebaut sind.

Für jedes *ICMPv6*-Paket wird eine Antwort durch das Board und den Host (der Border Router) gesendet. Eine Antwort, die mit **DUP!** markiert ist, ist daher korrekt.

Erweiterter Test

Im Zuge dieser Evaluierung wurde ein erweiterter Test durchgeführt, indem das Setup auf acht BLE-Nodes vom Typ Nordic Semiconductor nRF52840DK (Abschnitt 2.5.3) erweitert wurde. Wie bei dem Grund-Setup erfolgte die Verifizierung der Verbindungen auch in diesem Testfall über die Multicast-Adresse (`ping6 -I bt0 ff02::1`).

5.1.2 Ergebnis

Nach Anwendung der in Abschnitt 3.2 beschriebenen Änderung am Bluetooth-Kernelmodul, antworten auch mehrere BLE-Nodes im Netzwerk auf *ICMPv6*-Pakete. Aus dem Verhalten von acht simultan verbundenen Nodes wird geschlossen, dass das Routing des Border-Routers somit korrekt funktioniert und die Anforderungen auch für größere Netzwerke mit mehreren BLE-Peers erfüllt sind.

5.2 Auswahl mehrerer Bluetooth LE Dongles

Ziel war es, dem Border-Router die Möglichkeit zu geben, auch mit mehreren Host Controllern umgehen zu können (Abschnitt 3.3). Diese Funktion war bis zum Verfassen dieser Arbeit noch nicht implementiert und wird hier evaluiert.

5.2.1 Setup

Der Test aus 5.1.1 wurde erweitert. Statt der Particle Xenon Boards wurden zwei Development Boards *nRF52840DK* von Nordic Semiconductor verwendet.

Als zweiter, parallel betriebener Bluetooth-Controller wurde neben dem internen Bluetooth-Adapter der Test-Dongle Panasonic PAN1762 (Abschnitt 2.5.4) [7] genutzt. Die Anbindung des externen Dongles erfolgt über USB.

Netzwerktopologie

Der vorhin genannte Dongle von Panasonic muss vorab mittels folgendem Kommandozeilenbefehl initialisiert werden:

```
sudo hcitool attach /dev/ttyUSB<n> any 115200 noflow
```

/dev/ttyUSB<n>: n gibt die Nummer des USB-Geräts an.

In Abbildung 5.3 ist die prinzipielle Netzwerktechnologie dargestellt. Die beiden Host Controller Interfaces *hci0* und *hci1* befinden sich am selben Host.

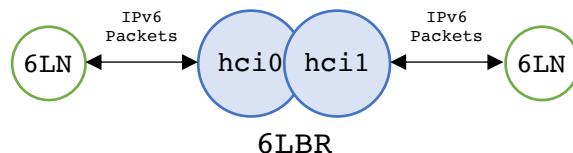


Abbildung 5.3: Netzwerktopologie

Abbildung 5.4 zeigt den Aufbau des Testsetups.

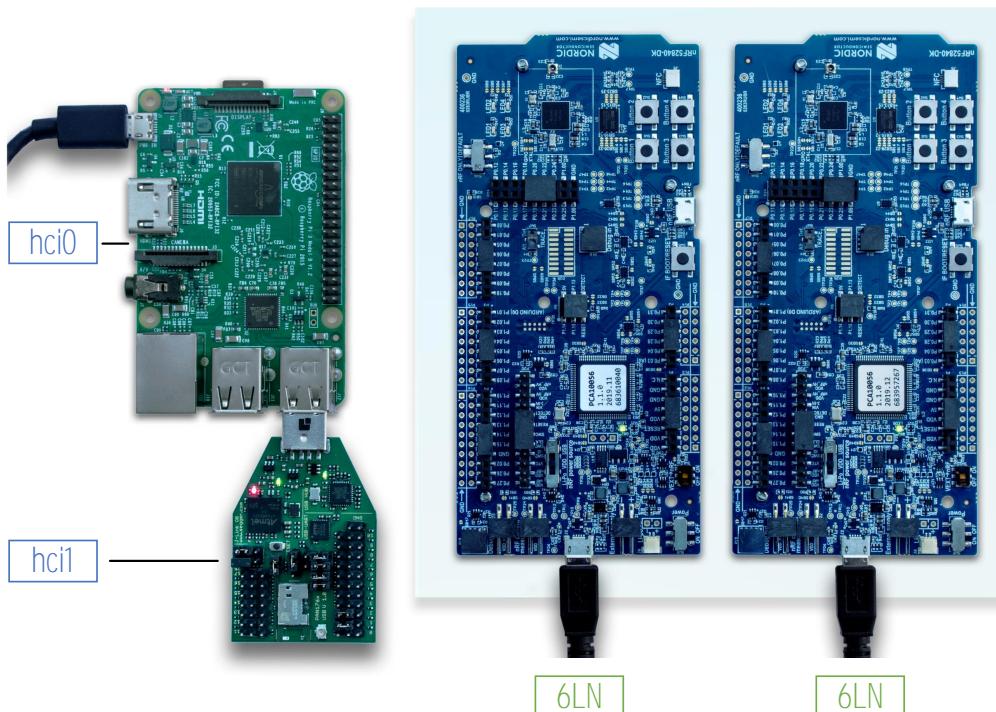


Abbildung 5.4: Schematischer Aufbau des Testsetups

Die Schritte zur Kompilation und Schritt 1 bis 2 zum Aufsetzen des Border Routers sind dabei dem Test aus dem vorangegangenen Abschnitt 5.1.1 zu entnehmen. Nach Durchführung der Schritte bis Punkt 2 aus Abschnitt 5.1.1 sind folgenden Befehle aufzurufen:

1. Die beiden Host Controller Interface resetten und nach verfügbaren BLE-Nodes suchen

```
hciconfig hci0 reset # Reset HCI device - in this case hci0 device.
hciconfig hci1 reset # Reset HCI device - in this case hci1 device.
hcitool lescan      # Read 00:AA:BB:XX:YY:ZZ address of the nRF5x device.
```

2. Das gewünscht HCI-Interface auswählen.

Dabei wird die HardwareAdresse des Host Controllers mit dem Befehl `sudo hciconfig` (Abbildung 5.5) ermittelt.

```
pi@raspberrypi:~ $ sudo hciconfig
hci1:  Type: Primary  Bus: UART
        BD Address: EC:21:E5:F5:2D:E9  ACL MTU: 251:8  SCO MTU: 0:0
        UP RUNNING
        RX bytes:238 acl:0 sco:0 events:17 errors:0
        TX bytes:99 acl:0 sco:0 commands:17 errors:0

hci0:  Type: Primary  Bus: UART
        BD Address: B8:27:EB:40:9B:06  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING
        RX bytes:230225 acl:75 sco:0 events:9403 errors:0
        TX bytes:12293 acl:104 sco:0 commands:322 errors:0

pi@raspberrypi:~ $
```

Abbildung 5.5: Ermitteln der verfügbaren Host Controller Interfaces

Um den Host Controller zu wechseln, wurde der in Abschnitt 3.3.2 implementierte `select`-Befehl verwendet.

```
select <HCI_DEVICE_ADDR>
```

<HCI_DEVICE_ADDR> ist dabei die Adresse des gewünschten Host Controllers.

Die Anwendung des Befehls erfolgte mittels folgenden Aufrufes:

```
echo "select B8:27:EB:40:9B:06" > /proc/ble/6lowpan_control
```

Ist mindestens eine Verbindung zwischen einem BLE-Peer und dem Host Controller hergestellt, wird ein neues (Bluetooth-)Netzwerkinterface (z. B mit dem Namen `bt0`) angelegt. Pro Host Controller Interface - mit mindestens einem verbundenen Peer - existiert ein Netzwerkinterface (Abbildung 5.8, in roter Markierung). Diese werden bei Vorhandensein andere Bluetooth-Netzwerkinterfaces aufsteigend nummeriert (`bt0`, `bt1`, ..., `btN`).

3. Die mittels `hcitool lescan` gefundenen BLE-Nodes verbinden

```
echo "connect 00:AA:BB:XX:YY:ZZ 2" > /proc/ble/6lowpan_control
```

Testen

Das Testen der Verbindung erfolgt analog zu Abschnitt 5.1.1.

```
pi@raspberrypi:~ $ ping6 -I bt0 -c2 fe80::d874:eaaff:fe0b:f1ed
PING fe80::d874:eaaff:fe0b:f1ed(fe80::d874:eaaff:fe0b:f1ed) from fe80::b827:ebff:fe40:9b06%bt0 bt0: 56 data bytes
64 bytes from fe80::d874:eaaff:fe0b:f1ed%bt0: icmp_seq=1 ttl=64 time=312 ms
64 bytes from fe80::d874:eaaff:fe0b:f1ed%bt0: icmp_seq=2 ttl=64 time=311 ms

--- fe80::d874:eaaff:fe0b:f1ed ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 311.997/312.461/312.926/0.726 ms
pi@raspberrypi:~ $
```

Abbildung 5.6: Senden eines „pings“ an den mit `hci0` verbundenen BLE-Peers

```

pi@raspberrypi:~ $ ping6 -I bt1 -c 2 fe80::e256:67ff:feb1:edb
PING fe80::e256:67ff:feb1:edb(fe80::e256:67ff:feb1:edb) from fe80::b827:ebff:fe40:9b06%bt1 bt1: 56 data bytes
64 bytes from fe80::e256:67ff:feb1:edb%bt1: icmp_seq=1 ttl=64 time=351 ms
64 bytes from fe80::e256:67ff:feb1:edb%bt1: icmp_seq=2 ttl=64 time=350 ms

--- fe80::e256:67ff:feb1:edb ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 350.023/350.553/351.083/0.530 ms
pi@raspberrypi:~ $

```

Abbildung 5.7: Senden eines „pings“ an den mit `hci1` verbundenen BLE-Peer

5.2.2 Ergebnis

Die Netzwerkinterfaces können mit `ifconfig` ermittelt werden. Das Testen der Verbindung selbst erfolgt ebenfalls über das Senden von *ICMPv6*-Paketen. Dabei haben beide BLE-Peers unabhängig voneinander auf die *pings* reagiert (Abbildung 5.7).

Die nachfolgende Abbildung 5.8 zeigt die beiden, mit unterschiedlichen Host Controllern verbundenen Boards und die beiden verschiedenen Netzwerkinterfaces.

Mit der erweiterten Implementation im Kernelmodul wurde das gewünschte Ergebnis erzielt.

```

pi@raspberrypi:~ $ sudo hcitool -i hci0 con
Connections:
< LE D8:74:EA:0B:F1:ED handle 64 state 1 lm MASTER
pi@raspberrypi:~ $

pi@raspberrypi:~ $ sudo hcitool -i hci1 con
Connections:
< LE E2:56:67:B1:ED:BB handle 1 state 1 lm MASTER
pi@raspberrypi:~ $

pi@raspberrypi:~ $ ifconfig -s
Iface      MTU   RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
bt0       1280      9     0     2 0        40      0     0     0 MRU
bt1       1280      2     0     4 0        20      0     0     0 MRU
eth0      1500   9081     0     4 0       7361      0     0     0 BMRU
lo        65536     59     0     0 0        59      0     0     0 LRU
wlan0     1500      0     0     0 0         0      0     0     0 BMU
pi@raspberrypi:~ $

[0] 0:bash*                               "raspberrypi" 18:12 17-Sep-19

```

Abbildung 5.8: Verbundene BLE-Nodes, mit zwei Netzwerkinterfaces

5.3 Testen des BLE Deamons

In diesem Abschnitt wird die Daemon-Applikation aus Kapitel 4 evaluiert. Die Testfälle umfassen dabei das

- automatische Verbinden
- Wechseln der Host Controller
- nachträgliche Verbinden der BLE-Peers bzw. Neuzuweisung an einen anderen Host Controller
- Trennen der Verbindung von BLE-Peers

Mit diesem Test soll gezeigt werden, dass mindestens acht BLE-Nodes und zwei Bluetooth-Dongles vollautomatisch verwaltet werden können.

5.3.1 Setup

Das Setup selbst besteht aus vier gemischten BLE-Nodes des Typs nRF52840DK und Particle Xenon. Am Border Router sind zwei Bluetooth Dongle aktiviert. Das Setup ist dabei gleich dem in 5.2 genannten. Neben dem internen Bluetooth Controller wurde per USB der externe Controller von Panasonic angeschlossen.

Test

Der Daemon wurde in Version *1.1.0(beta)* betrieben und getestet, zur initialen Konfiguration wurde die im Kapitel 4.5.4 beschriebene Konfigurationsdatei verwendet. Diese Datei ist ebenfalls dem Projekt-Repository beiliegend und unter `./BLEd/configs/bled_config.conf` zu finden.

Der Daemon wurde nach den Schritten in Abschnitt 4.4 kompiliert und als Debian-Paket installiert. Der Testzeitraum belief sich auf rund zwei Stunden. Dabei wurden in zufällig gewählten Zeitabständen manuell mehrere Socket-Verbindungen geöffnet und folgende Kommandos an den Daemon gesendet.

- `connect`
- `disconnect`
- `remap`
- `list`
- `con-update`

Die Verbindungen der Peers wurden mittels *pings* manuell getestet (siehe Anleitung 5.2), wobei erwartungsgemäß alle BLE-Nodes auf die Requests antworteten.

Die *Link-Layer Connection Parameter* wurden erfolgreich geändert. Auch hier konnten keine Anforderungsabweichungen festgestellt werden. Einzig in einem Fall führte die Aktualisierung zum Verbindungsverlust eines Peers, wobei das Problem jedoch auf Seiten des Peers lokalisiert wurde.

5.3.2 Ergebnis

Die Daemon-Applikation wurde in einem Zeitraum von ca. zwei Stunden betrieben. Dabei sind alle, zu diesem Zeitpunkt verfügbaren Befehle (Abschnitt 4.5.6), manuell getestet worden. Bei korrekter Anwendung werden alle Befehle fehlerfrei verarbeitet. Das `InteractionInterface` hat jedoch noch Probleme mit fehlerhaften Eingaben umzugehen. Ein falscher Befehl könnte unter Umständen zum Absturz führen. Der Daemon ist damit nicht mehr ansprechbar.

Das Aktualisieren der *Connection-Parameter* (Abschnitt 4.5.6) hingegen konnte problemlos durchgeführt werden und führte zum gewünschten Ergebnis. Die geänderten *Connection Parameter* konnten mittels der Antwortzeiten der *ping*-Requests verifiziert werden. Mit einem höheren *Connection Interval* ergibt sich eine höhere Antwortzeit.

Obwohl sich die Applikation im Beta-Stadium befindet, konnte während des Tests keine sonstigen Fehlfunktion in der Basis-Implementierung festgestellt werden. Das automatische Verbinden und Trennen der BLE-Peers erfolgten gemäß den Erwartungen. Auf Grundlage dieser Ergebnisse wurde der Daemon beurteilt, der damit die in Kapitel 1 genannten Spezifikationen erfüllt.

5.4 Verhalten bei unterschiedlichen Connection Parametern

Im folgenden Abschnitt wird das Verhalten von zwei Nodes, die mit dem selben HCI verbunden sind, evaluiert.

5.4.1 Setup

Zwei BLE-Peers des Types *Nordic nRF52840DK* sind mit dem Raspberry Pi verbunden. Als Bluetooth Controller wird der interne Controller des *Raspberry Pi* genutzt.

Das Verbindungsprozedere erfolgte entsprechend der Beschreibung aus 5.1.1.

Dem Gerät mit der Hardwareadresse E2:56:67:B1:ED:BB wurde der Handle 64, dem Gerät mit der Adresse D8:74:EA:0B:F1:ED der Handle 65 zugewiesen. Im Folgenden sind die Geräte mit den Namen *Peer 64* und *Peer 65* abgekürzt.

Die für die Tests verwendeten *Connection Parameter* sind in den nachfolgenden Tabellen, inkl. umgerechneter dezimaler und hexadezimale Werte, aufgelistet.

	Input	Wert	
		dezimal	hexadezimal
Connection Interval Min	100 ms	80	0x50 0x00
Connection Interval Max	100 ms	80	0x50 0x00
Connection Latency	0x0000	0x0000	0x00 0x00
Supervision Timeout	1000 ms	100	0x64 0x00
Minimum CE Length	30 ms	48	0x30 0x00
Maximum CE Length	30 ms	48	0x30 0x00

Tabelle 5.1: Connection Parameter 1

	Input	Wert	
		dezimal	hexadezimal
Connection Interval Min	200 ms	160	0xA0 0x00
Connection Interval Max	200 ms	160	0xA0 0x00
Connection Latency	0x0000	0x0000	0x00 0x00
Supervision Timeout	1000 ms	100	0x64 0x00
Minimum CE Length	30 ms	48	0x30 0x00
Maximum CE Length	30 ms	48	0x30 0x00

Tabelle 5.2: Connection Parameter 2

	Input	Wert	
		dezimal	hexadezimal
Connection Interval Min	400 ms	320	0x40 0x00
Connection Interval Max	400 ms	320	0x40 0x00
Connection Latency	0x0000	0x0000	0x00 0x00
Supervision Timeout	1000 ms	100	0x64 0x00
Minimum CE Length	30 ms	48	0x30 0x00
Maximum CE Length	30 ms	48	0x30 0x00

Tabelle 5.3: Connection Parameter 4

5.4.2 Ergebnis

Abbildung 5.9 zeigt die statistische Auswertung der Antwortzeit der *ping*-Echo-Requests. Die rote Linie kennzeichnet den Median, die schwarzen Begrenzungen die Minimal- und Maximalwerte. Die roten Plus-Symbole zeigen Ausreißer im Datensatz. Insgesamt wurden 250 ICMP-Sequenzen gesendet.

Bei den gesetzten Parametern CP2 nach Tabelle 5.2 haben beide Peers im Durchschnitt in etwa dieselbe Antwortzeit. Die mittlere Antwortzeit von *Peer 64* liegt bei ca. 291 ms, die von *Peer 65* 293 ms.

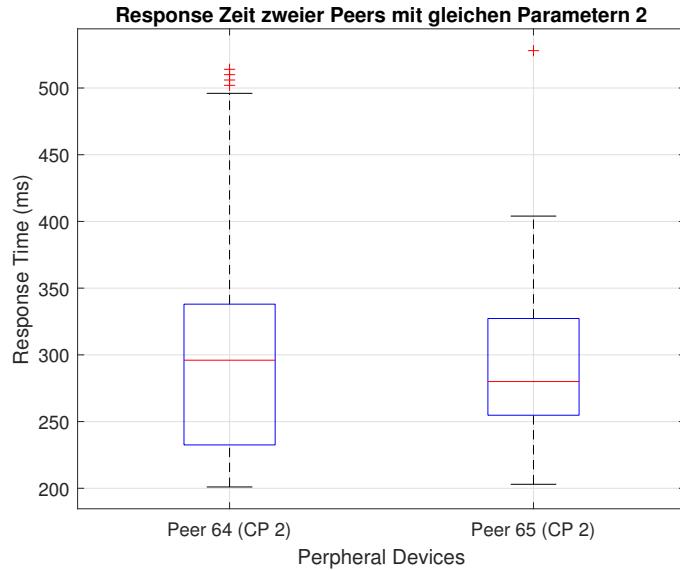


Abbildung 5.9: Response Time zweier BLE-Peers mit gleichen Connection Parameter

In Abbildung 5.10 ist die statistische Auswertung derselben Peers mit unterschiedlichen *Connection Parameter* dargestellt. Die *Connection Parameter* von *Peer 64* sind unverändert geblieben (CP 2, Tabelle 5.2), wohingegen das *Connection Interval* von *Peer 65* verdoppelt wurde (CP 4, Tabelle 5.3). Die durchschnittliche Antwortzeit von *Peer 64* liegt bei ca. 305 ms, die von *Peer 65* bei 612 ms. Die mittlere Antwortzeit von *Peer 65* ist ca. um das Doppelte gestiegen.

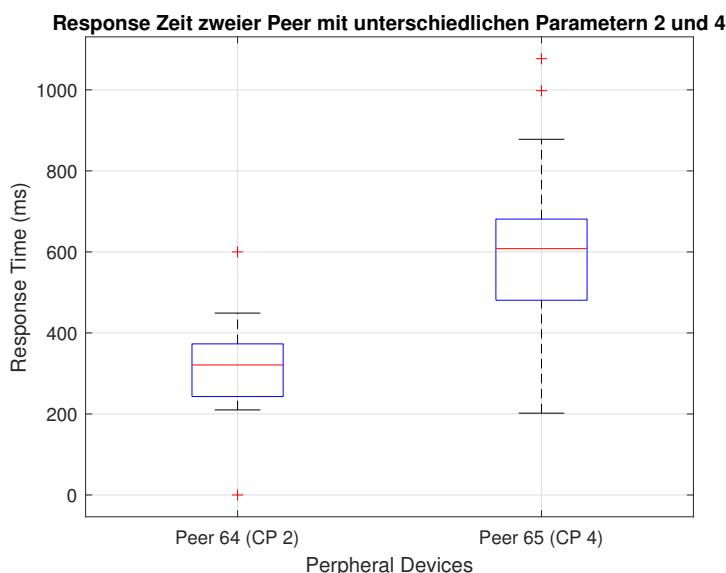


Abbildung 5.10: Response Time zweier BLE-Peers mit unterschiedlichen Connection Parameter

Im folgenden Plot 5.11 findet sich der Vergleich der Antwortzeit von *Peer 64* in unterschiedlichen Umgebungen. Dabei sind *Peer 64* und *Peer 65* zuerst mit gleichen und danach mit zueinander geänderten *Connection Parametern* getestet worden. Die *Connection-Parameter* von *Peer 64* sind in beiden Fällen unverändert geblieben, *Peer 65* wurde im zweiten Schritt geändert.

Die beiden linken Boxplots zeigen dabei *Peer 64* und *Peer 65* mit denselben *Connection Parametern* (CP 1, Tabelle 5.1), die beiden rechten *Peer 64* und *Peer 65* mit zueinander unterschiedlichen *Connection Parametern* (*Peer 64*: CP 1, Tabelle 5.1; *Peer 65*: CP 2, Tabelle 5.2). Der Mittelwert von *Peer 64* (wenn beide Peers dieselben Connection-Parameter haben), liegt bei 156 ms, der von *Peer 65* bei ca. 159 ms.

Werden die Connection Parameter von *Peer 65* auf CP 2 (Tabelle 5.2) gesetzt, steigt der Mittelwert der Antwortzeit von *Peer 65* auf ca. 289 ms. Der Mittelwert von *Peer 64* bleibt erwartungsgemäß im selben Bereich und liegt bei ca. 160 ms. Der Test wurde mit denselben Testparametern ein weiteres Mal wiederholt, wobei die Testergebnisse entsprechend gleich geblieben sind.

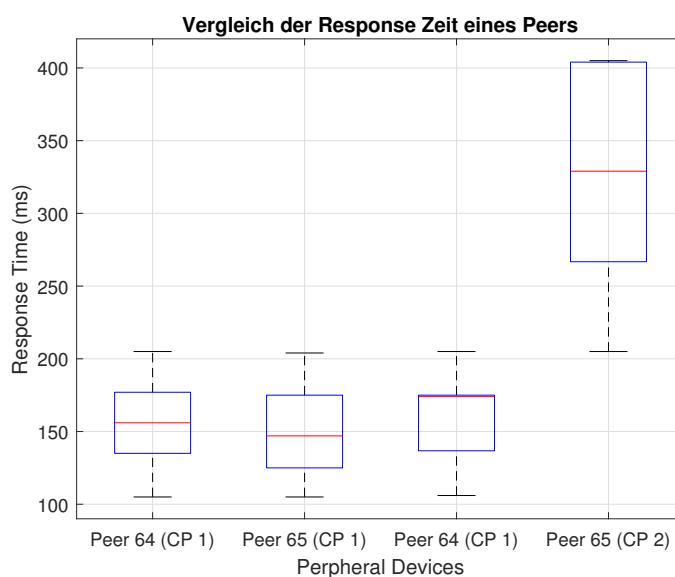


Abbildung 5.11: Response Time zweier BLE-Peers mit gleichen und zueinander verschiedenen Connection Parameter

6

Ergebnis

Das Ziel dieser Arbeit war es, einen IPv6-over-Bluetooth LE Border Router zu implementieren, wobei die Funktionsweise an die eines WLAN Routers angelehnt wurde. Dazu soll eine Applikation entwickelt werden, welche vollautomatisch und ohne Benutzerinteraktion auf einem Raspberry Pi 3 läuft. Die Aufgabenstellung sah vor, entsprechende Funktionalitäten zum automatischen Verbinden und Verwalten von BLE-Nodes zu implementieren, wobei auch mehrere Host Controller unterstützt werden sollten.

Dabei wird auf die bereits bestehenden Softwareteile des Bluetooth Kernelmoduls `bluetooth_6lowpan` aufgebaut. Die Schwierigkeit bestand darin, dieses Modul soweit anzupassen, dass es die in Kapitel 1 geforderten Punkte erfüllt. Dazu wurde das Modul durch entsprechende Funktionen erweitert und abgeändert.

Aus diesen Anforderungen wurde letztlich eine Lösung entwickelt, welche den grundlegenden Ansatz eines Routers implementiert, und die neben automatischem Verhalten dem Benutzer auch die Möglichkeit gibt, die BLE-Geräte selbst zu verwalten, indem auch die Zuordnungen der BLE-Peers zu Host Controllern geändert oder Verbindungen getrennt werden können.

Der Daemon wurde in der objektorientierten Sprache C++ programmiert und bietet somit eine gute Basis für zukünftige Weiterentwicklung. Zusätzlich wurden Schnittstellen implementiert, sodass es ohne großen Aufwand möglich ist, entsprechende Zusatzapplikationen, wie Webinterfaces oder Konfigurationsapplikationen für den BLEd-Daemon zu gestalten und anzubinden. Durch die modulare Weise ist es möglich, die nun bestehende Software zu erweitern und zu verbessern. Denkbar sind auch Sicherheitsfeatures wie Authentifizierungen. Eine Option wäre beispielsweise eine sichere Verbindung über Sockets via OpenSSL, welche implementiert werden kann.

6.1 Weiterführende Arbeit

1. Implementation eines Konfigurationsinterfaces in Form einer eigenständigen Applikation oder einer Website. Die derzeitige Implementation sieht kein Webinterface zur Konfiguration vor, doch würde sich die Verwendung eines solchen als nützlich erweisen. Dazu kann der unterstützte Befehlssatz aus Abschnitt 4.5.6 erweitert werden. Die Kommunikation kann hierzu über das bereits implementierte `InteractionInterface` erfolgen.
2. Erweitern des Befehlssatzes des Daemons. Aktuell unterstützt der Daemon nur einen Basisatz an Kommandos. Der Satz könnte mit entsprechenden Kommandos erweitert werden.

Zukünftige Benutzerbefehle

Folgende Befehle sind in der Entwicklung und noch nicht vollständig implementiert:

`exit` Beendet die aktuelle Verbindung mit dem `InteractionInterface` und schließt den Socket.

`shutdown` Beendet den Daemon.

`hci add <hci address>` Fügt einem weiteren Helper zur Verwaltung des angegebenen HCIs hinzu. Bei Aufruf dieses Befehls wird versucht, das angegebene Gerät zu initialisieren.

Sollten Befehle nicht verarbeitet worden sein, wird mit dem Initialisieren gewartet bis diese abgearbeitet wurden. Anschließend können BLE-Geräte mit den Standardbefehlen **connect** und **disconnect** oder **remap** diesem Host Controller zugewiesen werden.

hci remove <hci-address> Schließt den Helper, der für das Host Controller Interface mit der HardwareAdresse <hci-address> bestimmt ist. Verbundene BLE-Nodes verlieren gleichzeitig die Verbindung.

serialization <true|false> Ändert die Art der Ausgabe. Wird der Wert **true** angegeben, werden anstatt einer textbasierten Ausgabe nur mehr serialisierte Daten in Form von *protobuf*-Messages zurückgegeben. Das erleichtert die Anbindung an andere Applikationen.

Literaturverzeichnis

- [1] Bluetooth Special Interest Group. Solution Areas. <https://www.bluetooth.com/bluetooth-technology/solutions/>. Aufgerufen: 2019-08-06.
- [2] G. Naresh, *Inside Bluetooth Low Energy, Second Edition*, 2nd ed. Norwood: Artech House, 2016.
- [3] Bluetooth Special Interest Group. (2019, Jänner) Bluetooth Core Specification. <https://www.bluetooth.com/specifications/bluetooth-core-specification/>. Aufgerufen: 2019-07-07.
- [4] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby, and C. Gomez. (2015, Oktober) IPv6 over BLUETOOTH(R) Low Energy. <https://tools.ietf.org/html/rfc7668>. Aufgerufen: 2019-07-07.
- [5] Lucasbosch. (2014, Juli) Illustration eines Raspberry Pi B+. https://upload.wikimedia.org/wikipedia/commons/2/26/Raspberry_Pi_B%2B_illustration.svg. Aufgerufen: 2019-08-28.
- [6] Particle. Particle Xenon. <https://docs.particle.io/assets/images/xenon/xenon-top.png>. Aufgerufen: 2019-08-28.
- [7] Panasonic Corporation. PAN1762 Bluetooth Low Energy Module - Design Guide. https://eu.industrial.panasonic.com/sites/default/pidseu/files/downloads/files/wm_pan1762_design_guide_0.pdf. Aufgerufen: 2019-08-24.
- [8] M. Weiser. (1991, September) The Computer for the 21st Century. <https://www.lri.fr/~mbl/Stanford/CS477/papers/Weiser-SciAm.pdf>. Aufgerufen: 2019-07-09.
- [9] A. R. Chandan and V. D. Khairnar, “Bluetooth Low Energy (BLE) Crackdown Using IoT,” in *2018 International Conference on Inventive Research in Computing Applications (ICIR-CA)*, July 2018, pp. 1436–1441.
- [10] T. Øvrebekk. (2017, September) Why run IPV6 over Bluetooth low energy? <https://blog.nordicsemi.com/getconnected/why-run-ipv6-over-bluetooth-low-energy>. Aufgerufen: 2019-07-09.
- [11] H. Karvonen, M. Hämäläinen, J. Iinatti, and C. Pomalaza-Ráez, “Coexistence of wireless technologies in medical scenarios,” in *2017 European Conference on Networks and Communications (EuCNC)*, June 2017, pp. 1–5.
- [12] H. Karvonen, K. Mikhaylov, M. Hämäläinen, J. Iinatti, and C. Pomalaza-Ráez, “Interference of wireless technologies on BLE based WBANs in hospital scenarios,” in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Oct 2017, pp. 1–6.
- [13] J. Stern. (2012, Oktober) Philips Hue: The Light Bulb You Can Control With Your Phone. <https://abcnews.go.com/blogs/technology/2012/10/phipps-hue-the-light-bulb-you-can-control-with-your-phone/>. Aufgerufen: 2019-07-09.
- [14] H. Schubert. (2017, Juli) Bluetooth Mesh. <https://www.elektroniknet.de/elektronik/kommunikation/bluetooth-mesh-143929.html>. Aufgerufen: 2019-08-06.
- [15] Bluetooth SIG. Bluetooth mesh networking FAQs. <https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh/mesh-faq/>. Aufgerufen: 2019-11-04.

- [16] Jan Rähm. (2019, Oktober) Womit funkts das smarte Heim? <https://www.golem.de/news/funkstandards-womit-funkt-das-smarte-heim-1910-143420.html>. Aufgerufen: 2019-11-04.
- [17] M. Spörk, "IIPv6 over Bluetooth Low Energy using Contiki," Master's thesis, Technische Universität Graz, Oktober 2016.
- [18] D. Zivadinovic. (2018, Jänner) Vor 20 Jahren begann die Bluetooth-Ära. <https://www.heise.de/newsticker/meldung/Vor-20-Jahren-begann-die-Bluetooth-Aera-3938211.html>. Aufgerufen: 2019-08-06.
- [19] M. Kremp. (2012, Oktober) Was Sie über Bluetooth wissen müssen. <https://www.spiegel.de/netzwelt/gadgets/was-ist-bluetooth-a-860378.html>. Aufgerufen: 2019-08-06.
- [20] B. SIG. Our History. <https://www.bluetooth.com/about-us/our-history/>. Aufgerufen: 2019-08-06.
- [21] RS-Components. (2017, Jänner) Erforschung der geheimnisvollen Welt von Bluetooth 4.2 Low Energy (BLE). <https://www.rs-online.com/designspark/investigating-the-arcane-world-of-bluetooth-42-low-energy-ble-de>. Aufgerufen: 2019-08-06.
- [22] W. Warne. (2015, Februar) Bluetooth Low Energy – It starts with Advertising. <https://www.bluetooth.com/blog/bluetooth-low-energy-it-starts-with-advertising/>. Aufgerufen: 2019-07-08.
- [23] Mikroe. (2016, März) Bluetooth Low Energy - Part 1: Introduction To BLE. <https://www.mikroe.com/blog/bluetooth-low-energy-part-1-introduction-ble>. Aufgerufen: 2019-08-06.
- [24] University of Southern California. (1981, September) DARPA INTERNET PROGRAM - PROTOCOL SPECIFICATION. <https://tools.ietf.org/html/rfc791>. Aufgerufen: 2019-08-21.
- [25] H. Silvia, *IPv6 Essentials*, 2nd ed. Sebastopol: Ö'Reilly Media, Inc.", 2006.
- [26] IPv6.com. Stateless Auto Configuration. <https://www.ipv6.com/general/stateless-auto-configuration/>. Aufgerufen: 2019-08-21.
- [27] S. Thomson and T. Narten. (1998, Dezember) IPv6 Stateless Address Autoconfiguration. <https://tools.ietf.org/html/rfc2462>. Aufgerufen: 2019-08-21.
- [28] S. E. Deering and R. M. Hinden. (2017, July) Internet Protocol, Version 6 (IPv6) Specification. <https://tools.ietf.org/html/rfc8200>. Aufgerufen: 2019-08-21.
- [29] J. W. H. Ed. and P. Thubert. (2011, September) Compression Format for IPv6 Datagram over IEEE 802.15.4-Based Networks. <https://tools.ietf.org/html/rfc6282>. Aufgerufen: 2019-07-07.
- [30] Z. S. Ed., S. Chakrabarti, E. Nordmark, and C. Bormann. (2011, November) Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). <https://tools.ietf.org/html/rfc6775>. Aufgerufen: 2019-07-07.
- [31] G. Montenegro, N. Kushalnagar, J. W. Hui, and D. E. Culler. (2007, September) Transmission of IPv6 Packets over IEEE 802.15.4 Networks. <https://tools.ietf.org/html/rfc4944>. Aufgerufen: 2019-07-07.
- [32] Z. Shelby and C. Bormann. (2011, Mai) 6LoWPAN: The wireless embedded Internet - Part 1: Why 6LoWPAN? https://www.eetimes.com/document.asp?doc_id=1278794#. Aufgerufen: 2019-11-01.

- [33] C. Gomez, S. M. Darroudi, T. Savolainen, and M. Spoerk. (2019, März) IPv6 Mesh over BLUETOOTH(R) Low Energy using IPSP. <https://tools.ietf.org/id/draft-ietf-6lo-blemesh-05.html>. (work in progress) Aufgerufen: 2019-09-17.
- [34] Raspberry Pi Foundation, "Raspberry Pi 3 Model B, Spezifications," <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>, aufgerufen: 2019-08-04.
- [35] Particle, "XENON DATASHEET," <https://docs.particle.io/datasheets/mesh/xenon-datasheet/>, aufgerufen: 2019-08-04.
- [36] Nordic Semiconductor. nRF52840 DK Product Brief. <https://www.nordicsemi.com/-/media/Software-and-other-downloads/Product-Briefs/nRF52840-DK-product-brief.pdf>. Aufgerufen: 2019-08-26.
- [37] Mouser Electronic Inc. Panasonic PAN1762 Bluetooth® 5.0 Low Energy-Modul. <https://www.mouser.at/new/panasonic/panasonic-pan1762-ble-module/>. Aufgerufen: 2019-09-17.
- [38] ——. Panasonic PAN1762 Evaluierungskit. <https://www.mouser.at/new/panasonic/panasonic-pan1762-eval-kit/>. Aufgerufen: 2019-09-17.
- [39] ubuntuusers.de. Kernelmodule. <https://wiki.ubuntuusers.de/Kernelmodule/>. Aufgerufen: 2019-10-01.
- [40] J. Corbet, "IPv6 over BLUETOOTH(R) Low Energy," <https://www.kernel.org/doc/Documentation/filesystems/debugfs.txt>, 2009, aufgerufen: 2019-07-31.
- [41] J. Mayer, "Multiple peers with bluetooth_6lowpan," <https://www.spinics.net/lists/linux-bluetooth/msg78519.html>, Januar 2019, aufgerufen: 2019-08-04.
- [42] (2019, Juli) Bluetooth: 6lowpan: always check destination address. <https://github.com/raspberrypi/linux/commit/688d94fd0d10d9ebe611a445d85811894f8cf6c4>. Aufgerufen: 2019-10-05.
- [43] S. Levin, "[patch autosel 4.14 097/105] bluetooth: 6lowpan: search for destination address in all peers," <https://lkml.org/lkml/2019/7/15/1297>, 2009, aufgerufen: 2019-07-31.
- [44] BlueZ Project. Bluetooth protocol stack for Linux. <https://git.kernel.org/pub/scm/bluetooth/bluez.git>. Aufgerufen: 2019-04-25 DATE().
- [45] Hyperrealm. libconfig. <https://hyperrealm.github.io/libconfig/>. Aufgerufen: 2019-04-25.
- [46] Google Inc. Protocol Buffers. <https://developers.google.com/protocol-buffers/>. Aufgerufen: 2019-04-25.
- [47] Nordic Semiconductor. Connecting devices to the router - nRF5 IoT SDK v0.9.0. https://developer.nordicsemi.com/nRF5_IoT_SDK/doc/0.9.0/html/a00089.html. Aufgerufen: 2019-08-21.