Alexander Wieland, BSc

IPv6 over Bluetooth Low Energy on Android OS

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor
Ass.Prof. Dr.tech. Carlo Alberto Boano

Advisor
Dipl.Ing. Michael Spörk

Institute of Technical Informatics

Graz, August 2020

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

Date	Signature

Alexander Wieland, BSc

IPv6 over Bluetooth Low Energy on Android OS

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Informatik

eingereicht an der

Technischen Universität Graz

Beurteilender
Ass.Prof. Dr.tech Carlo Alberto Boano

Betreuer
Dipl.Ing. Michael Spörk

Institut für Technische Informatik

Graz, August 2020

EIDESSTATTLICHE ERKLÄRUNG

Abstract

Bluetooth Low Energy (BLE) is the energy-efficient evolution of Bluetooth. This revised technology is used in a wide range of application areas (e.g., connected health, smart homes, logistics and automation) as well as in a large number of consumer electronic devices. Most of these applications and devices are often connected not only to each other, but also to other systems or networks through the Internet, forming the so called "Internet of Things" (IoT). Since the application requirements and employed hardware platforms are largely diverse, not all devices forming the IoT use the same communication standards: this makes it difficult to preserve the Internet's end-to-end principle when developing IoT systems.

Thus, the need for a standardized connection between a Bluetooth Low Energy device and the Internet arose in the past decade, leading to the development of the RFC Standard 7668 for transmitting IPv6 packages through Bluetooth Low Energy. This standard allows the sending and forwarding of IPv6 packets within a BLE network and beyond, although a border router is still necessary to connect the BLE network to the Internet. For this task, smartphones and tablets are suitable, because they are able to connect to the Internet through 3G / LTE or WiFi, and most of them also have an integrated BLE chip. Unfortunately, to date, no smartphone operating system provides support for IPv6 over BLE. The contribution of this thesis is the design and implementation of an IPv6-over-BLE communication stack compliant to the RFC Standard 7668. The stack is interoperable with the Bluetooth stack of Android OS, the most widespread mobile operating system for smartphones and tablets; and it is usable within a normal Android application. In designing and implementing the IPv6-over-BLE stack, a number of challenges had to be tackled, including Android's restricted access to the BLE stack, the need to handle several IPv6-over-BLE connections in parallel, as well as the required multithreading, and the different ways of IPv6-supported Internet connections.

Furthermore, this thesis entails experiments that prove that the Android implementation is interoperable with existing standard-compliant solutions, that the implemented IPv6-over-BLE stack is working in parallel with the existing Bluetooth stack of Android, and that routing and border routing functionality is also possible with this implementation. The simple use of the app and the widespread use of Android devices could further accelerate the adoption of IPv6-over-BLE communication within the Internet of Things.

Kurzfassung

Bluetooth Low Energy (BLE) ist die energieeffiziente Evolution von Bluetooth. Diese überarbeitete Technologie wird in einer Vielzahl von Anwendungsbereichen (e.g., vernetzte Medizin, Smart Homes, Logistik und Automatisierung) und einer großen Anzahl von Unterhaltungselektronik verwendet. Die meisten dieser Anwendungen und Geräte sind nicht oder nicht nur miteinander verbunden, sondern auch mit anderen Systemen oder Netzwerken über das Internet. Diese Geräte und Netzwerke bilden das "Internet der Dinge". Da die Anzahl der verschiedenen Geräte sehr groß ist, verwenden nicht alle Geräte dieselben Kommunikationsstandards: das macht es schwierig das End-zu-End Prinzip des Internets bei der Entwicklung von Systemen im Internet der Dinge zu gewährleisten. Aus diesem Grund stieg im letzten Jahrzehnt der Bedarf für eine standardisierte Verbindung zwischen den Bluetooth Low Energy Geräten und dem Internet, was zu der Entwicklung des RFC 7668 Standard für die Übertragung von IPv6 Paketen über Bluetooth Low Energy führte. Dieser Standard erlaubt das Senden und Weiterleiten von IPv6 Paketen innerhalb eines BLE Netzwerks und darüber hinaus, obwohl ein Router weiterhin erforderlich ist, um das BLE Netzwerk mit dem Internet zu verbinden.

Für diese Aufgabe sind Smartphones und Tablets gut geeignet, da sie über 3G / LTE oder WiFi eine Verbindung zum Internet herstellen können, und die meisten von diesen Geräten auch über einen integrierten BLE Chip verfügen. Bis heute unterstützt leider kein mobiles Betriebssystem IPv6 über BLE.

Diese Masterarbeit enthält das Design und die Implementierung eines IPv6-über-BLE Kommunikationsstacks, der dem RFC 7668 Standard entspricht. Dieser Kommunikationsstack ist mit dem existierenden Bluetooth Kommunikationsstack des Android Betriebssystems kompatibel, welches das am meisten verbreitete mobile Betriebssystem für Smartphones und Tablets ist. Dieser Stack ist mit einer normalen Android Applikation verwendbar, welche leicht zu installieren und zu verwenden ist. Die größten Herausforderungen bei der Implementierung waren der eingeschränkte Zugriff auf den BLE Stack des Android Betriebssystems, die Unterstützung von mehreren IPv6 über BLE Verbindungen gleichzeitig, das Multithreading, und die unterschiedlichen Möglichkeiten einer IPv6-unterstützenden Internetverbindung. Darüber hinaus enthält diese Arbeit Tests welche zeigen, dass die Implementierung mit vorhandenen standardkonformen Lösungen kompatibel ist, dass der implementierte IPv6-über-BLE Kommunikationsstack parallel zum bereits vorhandenen Bluetooth Stack von Android funkioniert, und dass Routing und Border Routing Funkionen in dieser Implementierung ebenfalls enthalten sind und funktionieren. Durch die einfache Verwendung der App und der weiten Verbreitung von Android Geräten könnte dies die Verwendung der IPv6-über-BLE Kommunikation innerhalb des Internet der Dinge weiter forcieren.

Acknowledgment

First I would like to thank my supervisor Carlo Alberto Boano for his support and feedback during my work on this Master Thesis. Additionally I would like to thank Michael Spörk for his help and advice during this thesis.

I would like to thank my partner Sabine for her support and understanding. I would also like to thank my familiy, especially my parents, for their support and understanding, and for making me who I am today. Finally, I thank my friends and workmates for motivating and encouraging me.

Graz, August 2020 Alexander Wieland

Danksagung

Ich möchte vor allem meinem Betreuer Carlo Alberto Boano für seine gute Zusammenarbeit und Unterstützung danken. Zudem danke ich auch Michael Spörk für seine Hilfe und seinen Rat während dieser Arbeit.

Ich möchte mich auch bei meiner Partnerin Sabine für ihre Unterstützung und ihr Verständnis bedanken. Ebenso danke ich meiner Familie, vor allem meinen Eltern, dass sie mich immer unterstützt und gefördert haben, jederzeit beigestanden sind, und mich zu dem gemacht haben der ich heute bin. Zuletzt danke ich noch meinen Freunden und Arbeitskollegen, welche mich motiviert und bestärkt haben.

Graz, im August 2020 Alexander Wieland

Contents

List of Figures

List of Tables

Abbreviations

6CO 6LoWPAN Context Option

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks

AFH Adaptive Frequency Hopping

AOSP Android Open Source Project

API Application Programming Interface

ARO Address Registration Option

ATT Attribute Protocol

BDA Bluetooth Device Address

BLE Bluetooth Low Energy

BR Border Router

CI Connection Interval

CID Channel ID

CoC Connection-oriented Channel

CPU Central Processing Uni

DAC Destination Address Compression

DAM Destination Address Mode

DHCPv6 Dynamic Host Configuration Protocol version 6

DK Development Kit

DNS Domain Name System

ECN Explicit Congestion Notification

GAP Generic Access Profile

GATT	Generic Attribute Profile
GFSK	Gaussian Frequency Shift Keying
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HCI	Host Controller Interface
ICMPv6	Internet Control Message Protocol version 6
IDE	Intelligent Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IID	Interface Identifier
IoT	Internet of Things
IP	Internet Protocol
IPHC	Internet Protocol Header Compression
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
ISP	Internet Service Provider
JNI	Java Native Interface
L2CAP	Logical Link Control and Adaptations Protocol
LAN	Local Area Network
LE	Low Energy
LL	Link Layer
LTE	Long Term Evolution
MPS	Maximum Payload Size
MTU	Maximum Transmission Unit

NA	Neighbor Advertisement
NHC	Next Header Compression
NS	Neighbor Solicitation
OS	Operating System
PHY	Physical Layer
PIO	Prefix Information Option
PRR	Packet Reception Rate
PSM	Protocol Service Multiplexer
RA	Router Advertisement
RAM	Random Access Memory
RS	Router Solicitation
RTT	Round Trip Time
SAC	Source Address Compression
SAM	Source Address Mode
SDK	Software Development Kit
SIG	Special Interest Group
SM	Security Manager
SoC	System on Chip
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
WPAN	Wireless Personal Area Network

Chapter 1

Introduction

The connection of everyday objects to the Internet of Things (IoT) has gained increasing popularity in the last few years. Over the years, more and more simple devices such as fridges, washing machines, watches, parking areas and parcels use wireless technologies to connect to the Internet. Once connected, these smart objects can talk to other devices on the Internet to improve their user's experience. The main advantages of smart objects are, among others, the saving of energy, the saving of effort and time for the user, and the increasing personalization of objects and applications. The automatic notification of available parking places when entering a car park, the automatic shutter adjustment based on sensor data collected by a weather station, the suitable training plan due to the data collected by a fitness bracelet, or the mechanical distribution of packets according to their RFID-chip are just a few examples.

To support a vast amount and variety of devices on the IoT, there are several requirements on the communication on the IoT.

First, the devices need to be interoperable with other devices on the Internet. To achieve such interoperability, standardized communication procotols, such as the Internet Protocol (IP), are used. Devices and systems of different producers and for different purposes have interfaces and protocols to fit their original demands. With the IoT, the requirements increased and a standardized communication is necessary to connect these devices and systems with each other [1].

Second, the communication methods used by these devices need to be reliable to satisfy the end-user or given safety requirements. Indeed, to guarantee the correct behaviour of devices and applications, especially when they concern critical behaviour like a pacemaker or a fire alarm, a reliable communication is critical.

Third, some smart objects are battery-powered and therefore operate on a limited energy budget. For these devices it is essential that the used communication technology is energy efficient, in order for the system to have a low power consumption. Smart objects usually do not process much data, but they rather only collect and transmit sensor readings. Therefore, the radio is the most power consuming component of a smart object, and it is essential that it is shut off as much as possible.

1.1 BLE and the Internet of Things

Bluetooth Low Energy (BLE) is a low-power wireless technology that meets all the requirements of smart objects. It is designed to be very energy-efficient during connection establishment and data exchange [2]. Additionally, BLE provides several communication parameters that allow to configure the communication and therefore support a wide range of applications like a wireless blood pressure wrist and the BLE Blood Pressure Profile (BLP), or the Cycling Speed and Cadence Profile (CSCP) on a cyclocomputer [3]. Since BLE is already integrated into most everyday devices like smartphones and tablets, all these devices can directly interact with BLE-based smart objects without the need for a gateway. One example is the Nuki Smart Lock [4], which opens the door automatically if eligible smartphones are in range. Another example is the shutterBox [5], which is a shutter accessible by a tablet or smartphone over BLE. To make BLE even better suited for its use in the IoT, the Internet Engineering Task Force (IETF) has released in 2015 the RFC 7668 standard [6], which specifies how IPv6 packets can be exchanged over BLE connections. This enables smart objects based on BLE to directly connect to the IoT with their own IPv6 address. BLE-based smart objects that support this RFC specification are able to connect to any standard-compliant IPv6-over-BLE border router, like the Raspberry Pi, to join the IoT. Support for IPv6 over BLE is also available in most operating systems for constrained embedded devices like Zephyr [7] and Contiki [8].

1.2 Problem Statement

A major problem, however, is that so far there exists no RFC 7668 compliant implementation of an IPv6-over-BLE border router for smartphones or tablets. Neither the new iPhone models nor any Windows phone have IPv6 over BLE implemented. Also the Android operating system does not have an IPv6-over-BLE implementation, since the operating system denies the access to the L2CAP layer of the BLE stack, which is mandatory to implement IPv6-over-BLE support.

The aim of using smartphones as IPv6-over-BLE border routers is to facilitate and simplify the use of IoT devices. Standardization means that the router can be used for all IoT devices which are also standardized.

The main challenge of adding IPv6-over-BLE support to the Android OS is the access to the BLE L2CAP layer, which is the basis to support IPv6-over-BLE communication in Android. All layers above can be implemented inside the Android application itself. Another goal is to make the application as encapsulated as possible, to ensure reusability. It should run on as many Android OS as possible, to simplify the use and to spread it, and thus to expedite further development. Beside that, the developed application must be interoperable with other RFC 7668 compliant implementations.

1.3 Contributions

This thesis discusses how IPv6 over BLE can be implemented on the Android operating system, and presents an implementation according to the RFC 7668 standard [6]. Following points are addressed in this work:

- Design of a portable implementation of IPv6 over BLE for the Android operating system;
- Implementation on the Android Open Source Project (AOSP) version 7;
- Implementation on the Android factory image version 8;
- Full interoperability with implementations according to RFC 7668 standard [6];
- Routing functionality within the BLE subnet;
- Border router functionality.

In addition to these contributions, this Master thesis includes an evaluation of the comparison between an existing RFC 7668 compliant border router implementation, and the versions implemented in this thesis. Specifically, the evaluation considers power consumption and latency, and also contains further analyses the use of different connection parameters, the simultaneous usage of several nodes and other BLE protocols, as well as the routing functionality inside the BLE subnet and to the Internet.

1.4 Limitations

The application for supporting IPv6-over-BLE communication has the following limitations:

- The implementation is developed and tested with two operating systems: the Android Open Source Project version 7 and the Android factory image version 8. Although the application itself is designed for being used in further operating systems, small adaptations need to made when using another OS besides these two, as discussed in Section 8.3.
- The Internet routing capabilities are dependent on the mobile radio provider or the Internet service provider. One of them must support IPv6 communication, and in case of WiFi also the WiFi router must support IPv6. If none of them supports it, an alternative would be the usage of an IPv6 tunnel application, provided by third party providers.
- The user interface from the application of this Thesis contains only basic functionality. Sufficient for these experiments, it supports device discovery, connection establishment, and IPv6 prefix setting. Further functionalities are not provided by the implementation presented in this thesis, but can be expanded as the application is designed to support enhancements.

1.5 Outline

The remaining part of this thesis is structured as follows. First, the necessary background knowledge is shown in Chapter 2, which includes the key features of Bluetooth Low Energy and the BLE-stack. It also shows the design of IPv6 over BLE, the structure of Android

OS, and the hardware employed in all experiments. Chapter 3 compares existing IPv6-over-BLE implementations and summarizes scientific papers addressing the same topic. Chapter 4 describes the design of the new Bluetooth stack and the necessary adaptations of the Bluetooth architecture inside the Android operating system. It also includes the design of the user interface, the description of the communication setup, and a listing of the design challenges. Chapter 5 presents the implementation details concerning the Bluetooth stack, the integration into Android OS, the user interface and thread handling inside the application, as well as the points to consider when porting it to other Android OS versions. The evaluation of the performance, supported features and limits of the implementation are described in Chapter 6. The conclusion of this thesis is presented in Chapter 7. Chapter 8 provides possible improvements and additions on the implementation supporting IPv6 over BLE on Android.

Chapter 2

Background

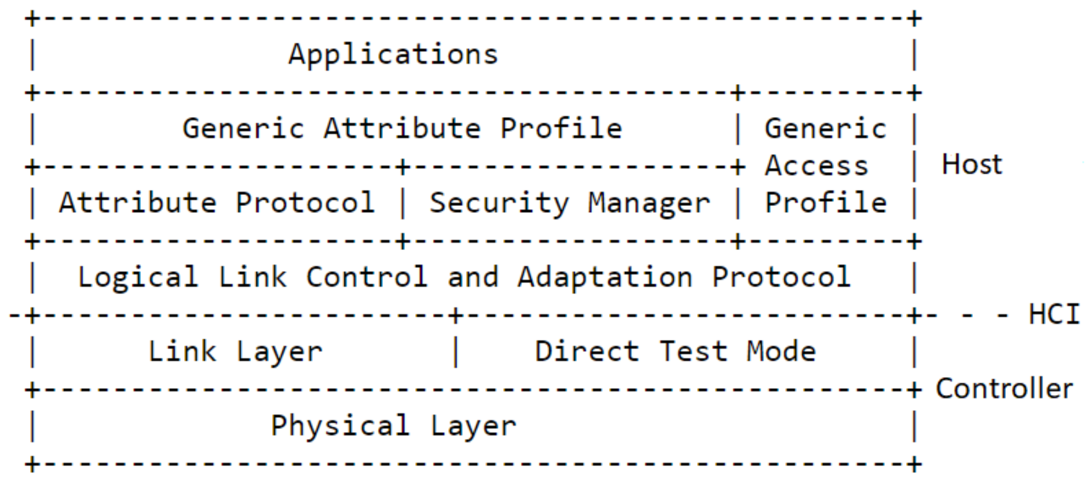
This chapter describes the main background aspects and their relation to this thesis. First, the radio technology Bluetooth Low Energy (i.e., the evolution of classic Bluetooth) and its stack are described. Second, details on the IPv6-over-BLE connection corresponding to the RFC 7668 standard [6], such as the topology, its adaptation of IPv6, and its stack are described. Third, both versions of Android OS used in this thesis, the Android Open Source Project, and the official Android factory image, are presented.

2.1 Bluetooth Low Energy

Bluetooth Low Energy (also referred to as Bluetooth LE or Bluetooth Smart) is a radio technology representing the energy-efficient evolution of Bluetooth. While the focus of classic Bluetooth was on high throughput, high range, and feature enhancements, the main design goal of BLE was its simplicity, energy-efficiency and low hardware cost, which makes it very suitable for applications that need a low data rate [3].

2.1.1 Fundamentals

The BLE radio operates in the unlicensed 2.4 GHz, industrial, scientific, and medical (ISM) band [9]. Because it is globally license free, several other wireless technologies, like WiFi, Bluetooth Basic Rate / Enhanced Rate (BR/EDR), or IEEE 802.15.4, also use these frequencies. In addition, this band is also interfered by other devices like microwave ovens or radars, leading to communication errors and packet loss. To avoid such interferences, BLE uses adaptive frequency-hopping (AFH). BLE uses only three radio channels for advertising and setting up the communication. Since there are just these three channels to scan, the connection between two devices can be set up much faster, compared with classic Bluetooth. Even if a connection is established between two BLE devices, the radio is mostly switched off. It is only switched on again if there are packets to transfer. So BLE does not necessarily poll between connected devices, such as classic Bluetooth is doing. Pure BLE chips are easier constructed than Bluetooth chips. They do not need such a wide frequency band and have a reduced dynamic memory because of reduced buffer sizes. It is also not necessary for all BLE chips to have a transmitter and receiver implemented, for some devices one of them is sufficient, depending on the task of the device, and it



```
 +-------------------------------------------------+
 |              Applications                       |
 +---------------------------------------+---------+
 |        Generic Attribute Profile      | Generic |
 +--------------------+------------------+ Access  | Host
 | Attribute Protocol | Security Manager | Profile |
 +--------------------+------------------+---------+
 |  Logical Link Control and Adaptation Protocol   |
-+-----------------------+-------------------------+- - - HCI
 |      Link Layer       |    Direct Test Mode     |
 +-------------------------------------------------+ Controller
 |              Physical Layer                     |
 +-------------------------------------------------+
```

Figure 2.1: BLE communication stack [6].

enables a saving in the silicon area. In addition to the short package size, shorter headers, a uniform packet format, and a simple protocol help reducing the memory usage.

2.1.2 BLE Stack

In this section the BLE stack in version 4.2 is described. The changes of the stack up to version 5 are described in the next section. Figure 2.1 shows the Bluetooth Low Energy protocol stack. The stack is split into two parts: a BLE controller and a BLE host. Both are connected via the Host Controller Interface (HCI).

Physical Layer

The physical layer is responsible for sending and receiving data. As already mentioned, the radio chip can contain both receiver and transmitter, or only one of them, depending on the purpose of the device. It operates in the 2.4 GHz ISM band and uses 40 radio channels that have a spacing of 2 MHz, whereby three of these channels are dedicated advertising channels. BLE devices have a range of 30 to 100 meters and a data rate of 1 Mbps, BLE v5.0 achieves even 2 Mbps [10]. The physical layer uses Gaussian Frequency Shift Keying (GFSK) for modulating / demodulating the signals [3].

Link Layer (LL)

At the beginning of a BLE communication, the BLE devices can have two different roles: advertiser or scanner. The advertiser can broadcast BLE advertising packets on the three advertising channels. These packets are receivable by any listening device in close proximity. BLE devices that listen for these advertising packets are in the scanning state and thus called scanners. The link layer states are represented in Figure 2.2. The time slots in which the packets are sent are called advertising events, which are non-overlapping and equally spaced out over time. At the beginning of every advertising event, an advertiser sends an advertising packet on one or multiple advertising channels. This communication is happening unidirectionally from advertiser to scanner, and is called connection-less.

If these two devices want to communicate bidirectionally, they need to establish a connection. Therefore, one device acts as initiator and is at the initiating state. It can also receive advertising packets and can respond with a connection request to initiate a connection with the advertiser. This works only if the advertising message entailed that the advertiser supports connections. During connection establishment, the initiating device sends all necessary connection parameters in the connection request packet, including the used data channel map and information about connection event timing. At the link layer, the connection interval and the latency are set by the master. The connection interval is the time between the events during which the radio is not active. The latency is the number of events the slave can skip in order to conserve energy. A short interval means a higher data rate, but also a higher power consumption [11].

If the connection is established, the master device is the previous initiator, and the slave is the previous advertiser, and both exchange data over this connection-based mode [3]. The time slot at which the master and the slave exchange data is called connection event. At each of these events, the master sends a packet to the slave, and the slave has to respond. After this exchange, the master and the slave can send further data packets until the end of the connection event. This flow control and packet acknowledgement guarantees the correct order of packets and retransmission of packet losses [12]. The used data channel of a connection event is calculated by the Adaptive Frequency Hopping algorithm of the BLE specification. This algorithm chooses one of the 37 available data channels [3]. The purpose of AFH is to mitigate interference, because of the general use of the same 2.4 GHz ISM band. During connection establishment, master and slave also synchronize the change of the data channels, which happens for every connection event. The map for the common change of the channels is called hop sequence, and it is unique for every connection. Since one channel is only used for a small timeframe and the hop sequence changes for every connection, the interference is reduced to a minimum.

Host Controller Interface (HCI)

The Host Controller Interface (HCI) provides standardized communication between the host and controller devices. It ensures that host devices may use BLE controllers from different manufacturers. The controller implements all BLE buffer management and communication timing functionality. The BLE host simply needs to add packets to the BLE controller via the HCI. This simplifies the development on the host. Four different kinds of HCI packages exist: command packages, event packages, synchronous data packages, and asynchronous data packages [14]. Commands are used to send orders from the host to the controller. The HCI events are used to notify the host if an event occurs at the controller, which can be a response to an earlier command. Errors are also sent by HCI event packages. The data packets can only be sent from host to controller or vice versa if a connection has been established. The host controller interface is optional. In case that HCI is not implemented in the system, the L2CAP layer may interact directly with the link layer [3].

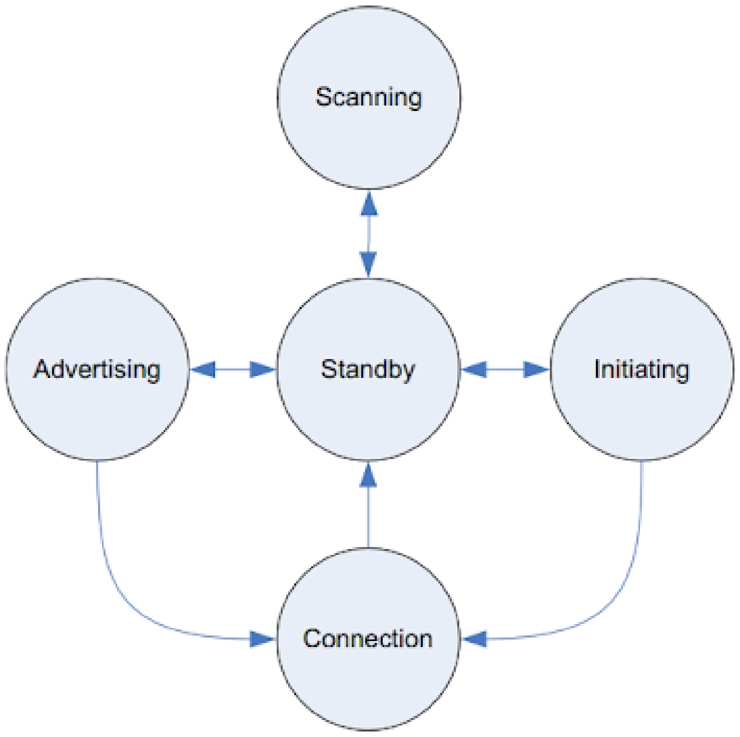


Figure 2.2: The link layer states of Bluetooth Low Energy taken from [13].

Logical Link Control and Adaptation Protocol Layer (L2CAP)

The Logical Link Control and Adaptation Protocol layer (L2CAP) provides protocol multiplexing and packet fragmentation and reassembly to the upper stack layers of the host. It is responsible for multiplexing and demultiplexing of channels over the shared logical link. It can also act as an interface between the upper layers and the controller layers, in case that the HCI is not implemented [3]. L2CAP contains five modes: basic L2CAP mode, flow control mode, retransmission mode, enhanced retransmission mode, and streaming mode. While only the basic L2CAP mode is used in BLE, the other ones are used in classic Bluetooth. In the flow control mode, the receiver is able to limit the number of incoming packets by setting the credit-value. For the use of IPv6 over BLE, it is required to support the flow control mode in BLE. Using the flow control mode, the requesting device (scanner) sends the credit-based connection request to the responding device (advertiser), which sends a credit-based connection response as answer. In the configuration process, the two devices negotiate the parameters of the L2CAP connection-oriented channel in credit-based flow control mode. These parameters are the low energy protocol service multiplexer (PSM), the channel ID (CID) of the source and the destination, the maximum transmission unit (MTU), the maximum payload size (MPS), and the initial credit value. The PSM is a port number that belongs to the protocol using the connection-oriented channel. The CID for the connection-oriented channels are dynamically assigned, where for BLE only the credit-based flow control mode for connection-oriented channels is possible [15]. The MTU describes the payload size of the L2CAP frame that will be exchanged

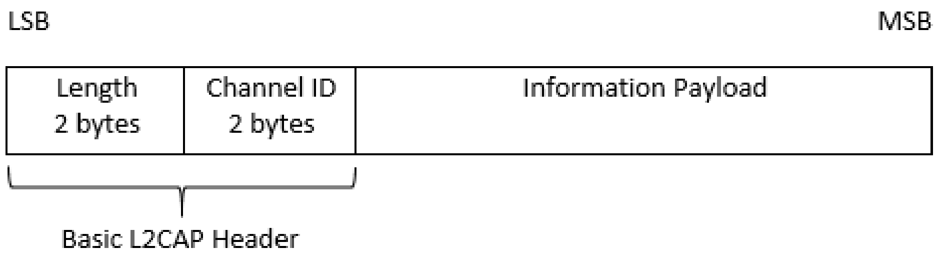


Figure 2.3: L2CAP packet format (adapted from [3]).

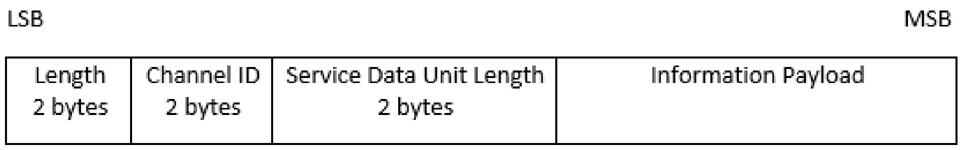


Figure 2.4: L2CAP flow-control packet format (adapted from [3]).

with the upper layers. The MPS describes the whole L2CAP frame, including L2CAP protocol information fields. The initial credit value states the number of LE frames that the advertiser can transmit. The advertiser can send a LE flow control credit packet to increase the credit counter. If the buffer is big enough for handling more packets, the flow control credit packet increases the credit counter according to the content of the credit packet. If the advertiser tries to send more LE frames than the credit counter states, the scanner will close the channel.

Security Manager Layer (SM)

The security manager layer (SM) contains the processes for authentication, pairing and encryption in BLE. It is needed if a connection requests security and is responsible for generating as well as securely storing cryptographic keys used for communication. Furthermore, the SM layer is responsible for generating the devices' random device address and for resolving random addresses from other BLE devices [3].

Attribute Protocol Layer (ATT)

The attribute protocol layer (ATT) implements a client / server protocol that allows BLE devices to exchange information about attributes. Every BLE device can be client or server, independent of being master or slave. The server has attributes that can be discovered, read, and written from the client. An attribute can be any data and has describing information about this data. The server can also notify the client about its attributes [3].

Generic Attribute Profile Layer (GATT)

The generic attribute profile layer (GATT) provides services to communicate between two connected devices. It defines a framework with services and characteristics, which uses the ATT attributes. Besides this, it defines how to read, write, and notify these attributes. The attributes can be grouped into services and be used hierarchically by the GATT layer [3].

Generic Access Profile Layer (GAP)

That layer defines the main functionalities all BLE devices have in common. It is mandatory for all devices and ensures interoperability between devices from different vendors. This entails general services regarding device modes, device discovery, device connection, and security. A BLE device can take one of four GAP roles: broadcaster, observer, peripheral, or central. In the broadcast role, a device sends advertising events periodically. In the observer role, a device receives the advertising packets from a broadcast device. In the Peripheral role, a device accepts a connection request and is therefore a slave. In the central role, a device initiates a connection establishment and is therefore a master [3].

2.1.3 BLE version 5

Bluetooth Low Energy version 5 has some improvements compared to version 4.2 [16]. First of all, the speed has been doubled. Second, the range has increased by 4 times compared to the previous version. Third, the data capacity of broadcast messages is 8 times higher than before. BLE version 5 is backwards compatible, but only with the features of the previous BLE version, so with the lower speed, range, and data capacity. This version also has two new physical layers compared to version 4. Each physical layer supports different capabilities. The first physical layer, called 1M PHY, is the same as used in version 4. The second one, called 2M PHY, has the capability of doubling the speed and supports higher data rates than the 1M PHY. The third physical layer, called Coded PHY, supports a much longer range thanks to additional symbol redundancy when encoding bits. However, not all features of version 5 can be used in parallel, since only one of the physical layer can be used at once.

Bluetooth version 5.1 was released in 2019 [17] and contains small improvements compared to version 5.0. It contains the feature to determine the direction of the Bluetooth signal, by using Angle of Arrival measurements, which allow for a more accurate location determination. In addition, BLE v5.1 makes pairing of devices faster by improving advertising, and enhances GATT caching for better avoiding disruptions.

Bluetooth version 5.2 was released in 2020 [18]. It contains a new feature to support LE audio in BLE devices through isochronous channels, which enables the communication of time bound data. In addition, version 5.2 supports low energy power control and enhanced attribute protocol.

2.2 6LoWPAN

6LoWPAN takes his name from 6Lo (IPv6 over networks of resource constrained nodes) and WPAN (Wireless Personal Area Networks). 6LoWPAN is a communication protocol designed to support IPv6 communication through wireless data transmission between constrained and low energy devices. It is specified in the RFC 4944 standard [19] released in 2007. This standard was developed for communication through IEEE 802.15.4, and was adapted for communication through Bluetooth Low Energy in the standard RFC 7668 [6], which was released in 2015.

2.2.1 6LoWPAN for IEEE 802.15.4

The 6LoWPAN adaptation layer has been specifically designed for IEEE 802.15.4 [19]. It contains functionalities like the IPv6 header compression [20], fragmentation and defragmentation of packets, address configuration, neighbor discovery [21] on low power wireless networks, and packet delivery in mesh networks.

Besides the star topology, the standard supports also mesh routing, which means that the features must also be adapted in terms of mesh routing. Neighbor discovery optimizations for 6LoWPAN are described in the RFC 6775 standard [21]. The latter defines neighbor discovery optimizations for low power wireless personal area networks including IPv6 neighbor discovery, addressing mechanisms, and duplicate address detection. These optimizations concern both star and mesh toplogies.

Since IPv6 needs a higher MTU than IEEE 802.15.4 supports, it is necessary to split the packets into smaller fragments that are individually sent and reassembled at the receiver. For the star topology, it is easy to transmit frames and reassemble them. However, there are two options for mesh routing: completely reassembling a message on each node before forwarding, or forwarding each frame separately.

The features of the 6LoWPAN adaptation layer for IEEE 802.15.4, which are also used for IPv6 over Bluetooth low energy (header compression, address configuration, neighbor discovery), are included in the standard of IPv6 over Bluetooth low energy [6] and are described in Section 2.3.2.

2.3 IPv6 over BLE

As already discussed, BLE fits all the requirements for smart objects that connect to the IoT. To allow seamless integration of BLE devices into the IoT, the RFC 7668 standard [6] specifies the exchange of IPv6 packets over a Bluetooth Low Energy connection. The main parts of this standard are the new protocol stack, the link model, the address configuration, and the header compression.

2.3.1 Topology

Initially, BLE only had support for star networks (see Figure 2.5) consisting of a central master device and one or multiple slave devices. At the end of 2017, however, the Bluetooth Special Interest Group (SIG) introduced the BLE mesh specification [22]. Because the specification of IPv6 over BLE only supports the star topology of BLE, we will focus on

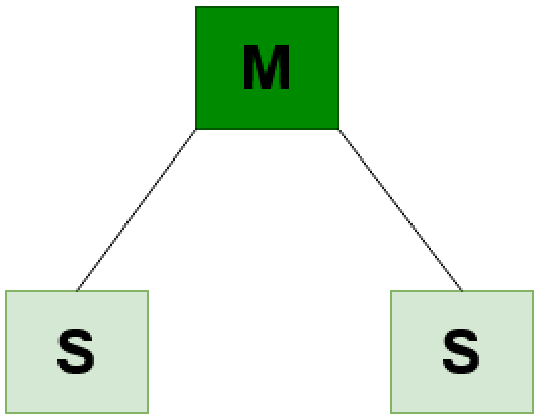


Figure 2.5: Piconet with one master (M) and two slaves (S).

this topology during this work. An IPv6-over-BLE network only supports star topology with an IPv6-over-BLE border router (the BLE master) as a central, and one or multiple IPv6-over-BLE nodes (BLE slaves), as shown in Figure 2.6. The BLE mesh topology is not supported in IPv6 over BLE.

The IPv6 neighbour discovery has to consider only functions concerning star topology. The nodes do not have to register their link-local addresses, but still need to register their non-link-local addresses on the border router. To this end, they send a Neighbour Solicitation (NS) message which contains the Address Registration Option (ARO). Then they get a response from the BR with the Neighbour Advertisement (NA) [6]. Each node device has only a single BLE connection to its border router. When nodes in the same subnet want to communicate, they need to route their messages over the central border router.

2.3.2 6LoWPAN for BLE

The 6LoWPAN adaptation layer was created for IEEE 802.15.4 [19] and it is the ancestor of IPv6 over BLE [6]. 6LoWPAN for BLE builds on top of that standard but it only takes on a few of the features of 6LoWPAN for IEEE 802.15.4, because IPv6 over BLE only supports star topology, and parts of the functionality is already covered by other layers in the BLE stack. The header compression and the address configuration are taken over. The neighbor discovery is partly taken over, since IPv6 over BLE does not support mesh topology. Packet fragmentation is not necessary, because this feature is already covered by the L2CAP layer of BLE.

The 6LoWPAN layer for BLE communication [6] is responsible for the compression of the IPv6 header [20], since the fragmentation gets executed on the L2CAP layer. The compression is standardized in RFC 6282 [20], and results in a header with a size of 2, 12 or 20 bytes depending on the network location of the target [23].

The IPv6 header has a size of 40 bytes. The first field contains the IP version with the size of 4 bits. It can be omitted entirely, since the version is already fixed. The next field

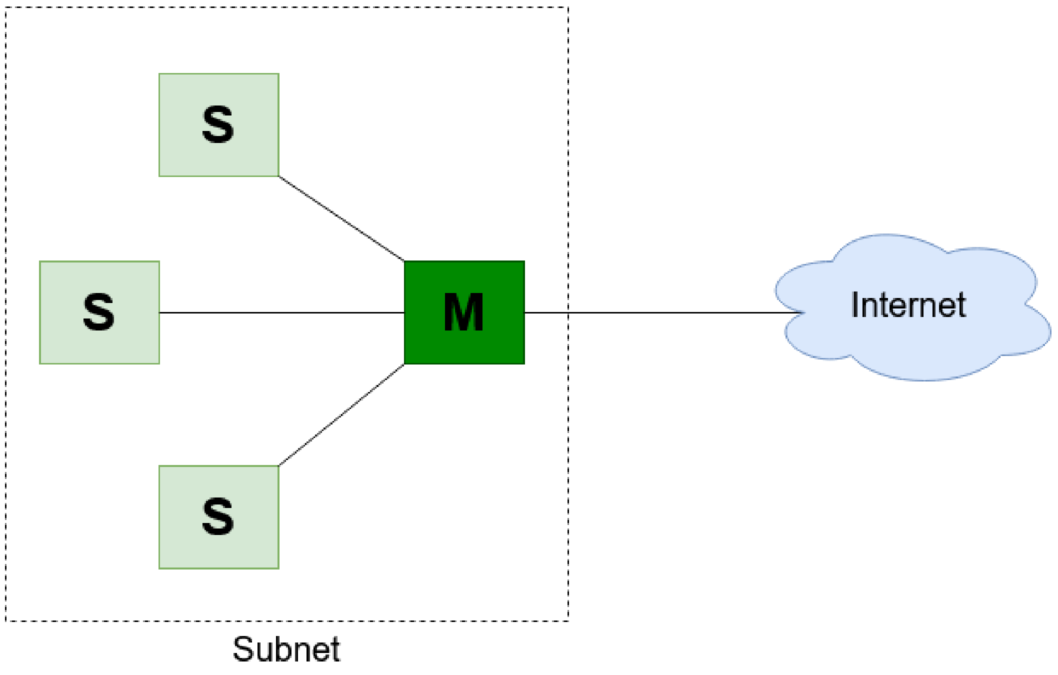


Figure 2.6: IPv6-over-BLE network connected to the Internet (adapted from [6]).

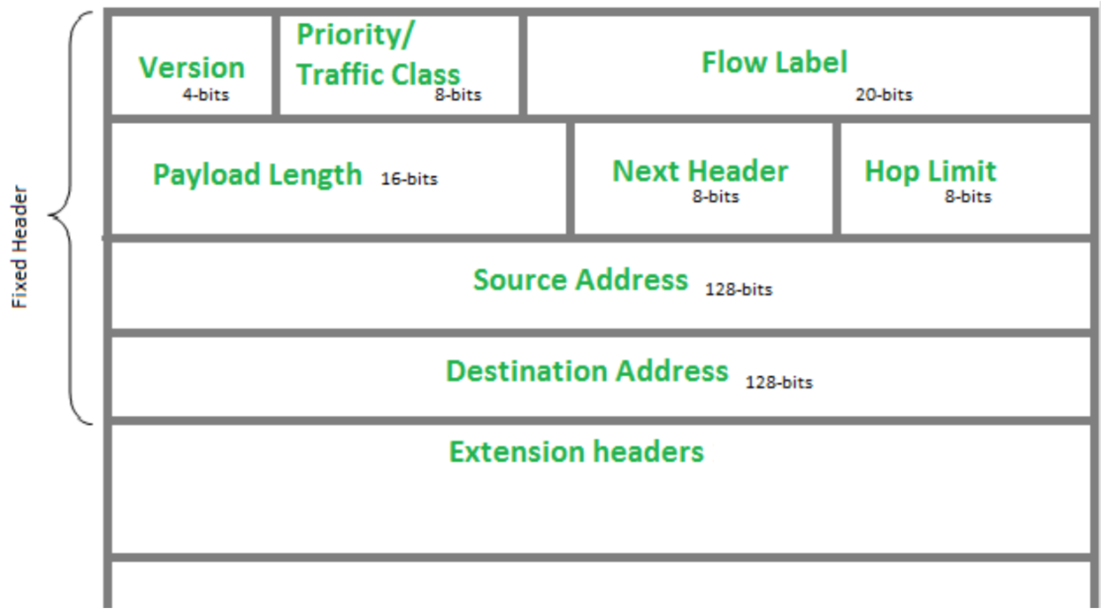


Figure 2.7: IPv6 header with fixed header fields (adapted from [24]).

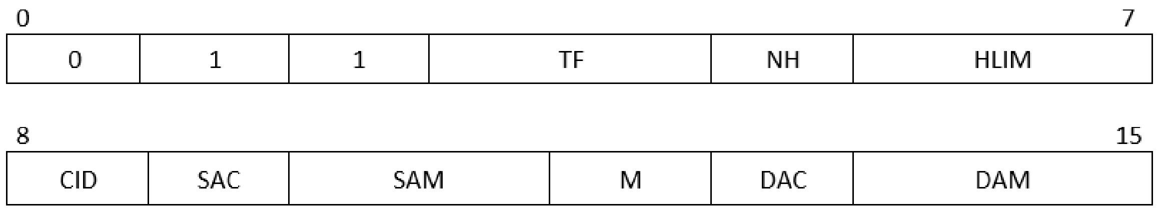


Figure 2.8: Compressed IPv6 base header (adapted from [20]).

is one byte long and contains the traffic class with 6 bits, and 2 bits Explicit Congestion Notification (ECN) priority value. This field is either fully elided, partially elided (1 or 3 bytes carried in-line), or fully contained (4 bytes) in the in-line fields. The field describing the payload length with size of 2 bytes is also entirely omitted. The next field is 1 byte long and specifies the type of the next header. Either this field is carried fully in-line, or this field is compressed and the next header is encoded using the next header compression (NHC). After that, there is the hop limit field, also with length of 1 byte. It is either fully carried in-line or compressed, and the value is included inside the base format of the header compression. The remaining two fields contain the source and the destination addresses, each with a size of 16 bytes. The addresses can either be completely omitted, partially omitted (8, 6 or 2 bytes carried in-line), or completely included in-line.

The source address of a transmission from a node to the border router has to be fully elided, also transmissions with a non-link-local source address that are registered to the border router if it is the latest address of the node for the indicated prefix. The source address of a transmission from a border router to a node has to be fully elided, if it is the link-local address based on the border routers BDA. Also the source prefix or address has to be elided if a compression context has been set up. The destination address of a transmission from a BR to a node has to be also fully elided, if it is the link-local address based on the nodes BDA, or if the latest registered address matches with the destination address. To elide addresses, the header compression flags SAC, SAM, DAC and DAM are used.

The resulting compressed IP header is shown in Figure 2.9. It consists of the base format, which is either 2 or 3 bytes long, the appended in-line IPv6 header fields, the next header compression, appended in-line next header fields, and the payload.

The first three bits of the base format are a dispatch. Then 2 bits follow for traffic class and flow label. The next bit belongs to the next header field, and the last two bits of the first byte describe the hop limit. The second byte starts with one bit describing if a context identifier extension field is attached, which is following the two bytes of the base format of the IPv6 compressed header. The next bit belongs to the source address compression, followed by two bits for the source address mode. Then one bit for the multicast compression, followed by one bit for the destination address compression are used. The last two bits belong to destination address mode.

The next header compression [20] also consists of a header compression encoding and

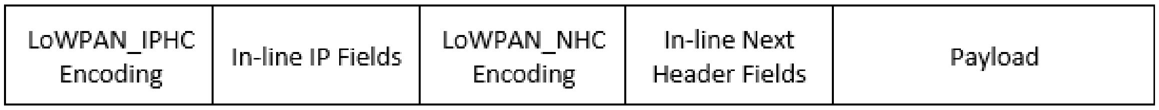


Figure 2.9: Full compressed IPv6 message with 6LoWPAN IPHC and NHC configuration (adapted from [20]).

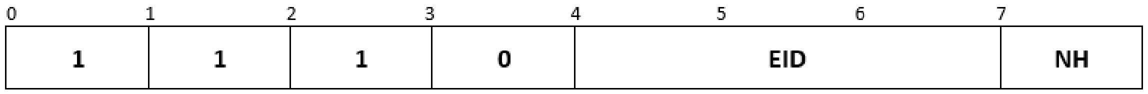


Figure 2.10: IPv6 extension header encoding (adapted from [20]).

in-line fields. The next header compression encoding consists of a single byte (Figure 2.10). The first four bits are a dispatch, and the following three bits are an identifier for the following IPv6 extension header. The last bit indicates if the following header uses next header compression as well.

2.3.3 IPv6 adaptation

To use IPv6 packets on BLE devices, several adaptations need to be performed. First, a link-local IPv6-over-BLE address is generated based on the devices' BLE address. The link-local addresses are used to establish the connection. Nodes and border routers should use a private Bluetooth Device Address (BDA), and thus should periodically change the BDA to keep the addresses private, as already mentioned at the link layer in Section 2.1.2. From the BDA a 64-bit Interface Identifier (IID) is formed. Link-local addresses should stay the same for one L2CAP channel. So changing device addresses does not influence communication, as long as the L2CAP channel stays the same [6]. Addresses that are non-link-local should not embed the BDA in the IID. For generating the IID, methods like cryptographic generated addresses, privacy extensions, or DHCPv6 should be used instead. If the BDA is private, it can be used in the IID.

Second, IPv6 over BLE compresses IP headers [20]. In addition, also the device addresses can be compressed using the star topology. Further details on the IPv6 header compression and the included compression of device addresses, both provided by 6LoWPAN, is described in Section 2.2.

2.3.4 IPv6 over BLE Stack

To support IPv6 over BLE, only the layers of the BLE host need to be adapted. The BLE controller stays unchanged. The normal BLE stack can be seen in Figure 2.1, and the new stack supporting IPv6 can be seen in Figure 4.1. The L2CAP layer needs to be used in credit-based flow control mode. This mode checks if the communication partner can receive the packets and then sends all fragmented packets consecutively in the right order. The credit number states the amount of packets a device can receive. This mode is used to control the flow of the LE-frames, so that the IPv6 packets are fully transferred. IPv6 packets may be bigger than the BLE package size: IPv6 alone has a header length of 40 bytes, UDP has a header of 8 bytes. In BLE specification v4.0 and v4.1 the data field at the L2CAP layer has a size of 27 bytes, but in v4.2 and above it increases to 251 bytes [25]. Therefore, a fragmentation and reassembly of the IPv6 packets is necessary. This will be taken care by L2CAP frames to its already integrated features. L2CAP enables higher layer protocols to send bigger data packets, which get split and sent to the controller in appropriate sizes dependent on the buffers. After reception, the L2CAP layer reassembles the fragmented data, reconstructs the data packet, and notifies the upper layers [3].

2.4 Android Operating System

There are several operating systems that would be suitable to implement IPv6 over BLE on smartphones, like iOS, Windows Phone, and Android. The focus of this thesis is on Android because it has a market share [26] of 75.0 % worldwide and provides its operating system not only for smartphones, but also for tablets. These are good reasons to implement the requested solution for Android devices.

Because of the restrictions of a normal Android application, a wide intervention to the lower layers of the Android operating system is not possible with a normal Android app. Especially the access to the Bluetooth LE L2CAP layer, which is an important part of the implementation of IPv6-over-BLE communication, is denied by the OS.

However, the Android Open Source Project (AOSP) [27] enables to make changes at every layer of the operating system. Thus it is possible to implement the IPv6 communication in the necessary layers of the Bluetooth Low Energy stack, and to add the remaining functionality in the app itself. Other advantages of using the AOSP is the free availability of this operating system and the suitability for many devices, both smartphones and tablets.

If the L2CAP layer does not have to be changed because the required functionality is already supported in the layer, and made accessible through the interface, it is possible to build the remaining implementation into the app itself. This also simplifies the usage, as only the Android application has to be installed.

The factory image of Android 8 supports the L2CAP connection establishment of a connection-oriented channel in credit-based flow control mode. However, the interface is not publicly available, which normally excludes its use. Nevertheless, through the workaround with Java reflection, this functionality can still be used. This means that the Android app developed in this thesis can also work with all smartphones and tablets that use Android 8. This considerably increases the proportion of devices that could immediately support IPv6-over-BLE communication. In addition, the effort required when using a factory image from Android is significantly lower than when using the Android Open Source project.

Figure 2.11: LG Nexus 5X smartphone with Android OS [28].

2.5 Employed Hardware

The hardware used for the implementation of IPv6 over BLE and for the evaluation experiments are the LG Nexus 5X, the Raspberry Pi 4 Model B, the Samsung Galaxy Tab S3 tablet as border router, as well as the Nordic Semiconductor nRF52840-DK development kit as node device.

LG Nexus 5X

The LG Nexus 5X is a smartphone with Android OS. It is a Google device, which means that it is distributed by Google, and was produces by a contract manufacturer [28]. It is designed for Android OS and has some advantages compared to non Google devices, like preinstalled software and additional Google support. The used operating system is Android 8.0. Since it is a Google device, it is easy to install a version of the Android Open Source Project on it, which is the reason why it was chosen for this thesis. It has a Qualcomm Snapdragon 808 64 bit System on Chip (SoC), which contains 6 CPU cores (2 x 1.8 GHz, 4 x 1.44 GHz), an Adreno 418 graphical unit, two LPDDR3 memory controller, and a LTE modem [29]. The RAM size is 2 GB, and for WiFi and Bluetooth it contains a Qualcomm QCA6174A chip which supports Bluetooth version 4.2 [30].

Because this smartphone can handle both AOSP 7 and Android 8, it is ideal to compare the same implementation running on both OS.

Figure 2.12: Samsung Galaxy Tab S3 tablet with Android OS [31].

Samsung Galaxy Tab S3

The Samsung Galaxy Tab S3 [31] is a tablet with Android OS. The OS version on it is Android 8.0. It has a Qualcomm Snapdragon 820 64 bit SoC. It contains 4 CPU cores (2 x 1.6 GHz and 2 x 2.15 GHz), an Adreno 530 graphical unit, an LPDDR4 memory controller, and an LTE and WiFi modem [32]. It contains 4 GB RAM and the Qualcomm QCA6174A chip, the same as in the LG Nexus 5X, supports Bluetooth version 4.2 and WiFi [30].

Nordic Semiconductor nRF52840-DK

The nRF52840 development kit (DK) from Nordic Semiconductor is a low-power SoC. It contains a 32 bit ARM Cortex M4 CPU with 64 MHz, 256 kB RAM and 1MB flash memory. It also contains support for Bluetooth 5, which is completely backwards compatible. Therefore, the nRF5280-DK as node device is interoperable with router devices that support Bluetooth 4.2 [33]. The used operating system is Zephyr [7], which is an open source real time OS developed by the Linux foundation for the Internet of Things. It contains a full IPv6-over-BLE stack, and supports both node and router functionality.

Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B device is used as border router and is treated as reference concerning the IPv6-over-BLE implementation [34]. It contains a Broadcom BCM 2711

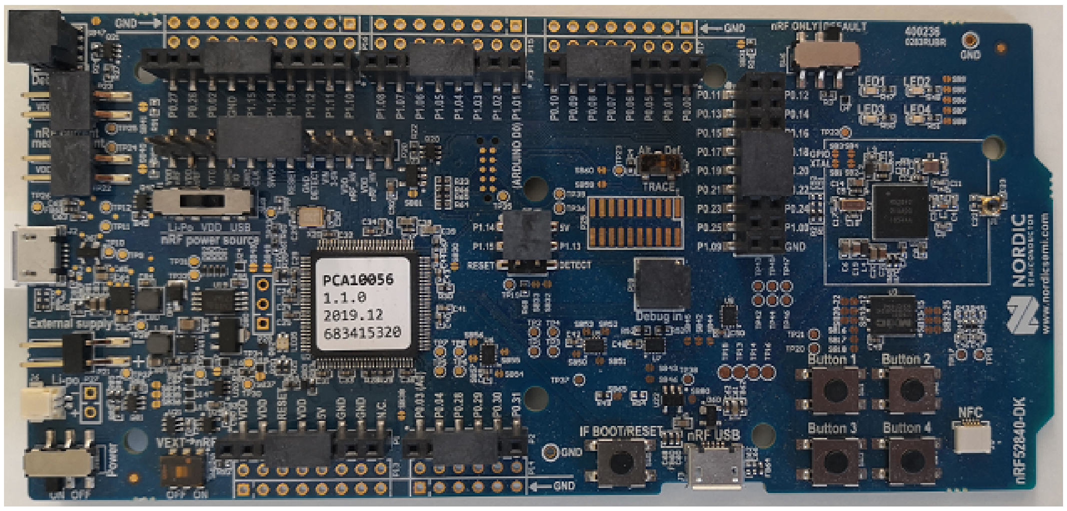


Figure 2.13: Nordic Semiconductor nRF52840-DK [33].

Figure 2.14: Raspberry Pi 4 Model B with Raspbian OS [34].

SoC, which includes an ARM Cortex-A72 CPU with 4 cores (4 x 1.5 GHz). It has 4 GB LPDDR4 RAM, and the Broadcom BCM54213PE module contains support for Bluetooth version 5. The operating system version used in this thesis is Raspbian 10, which includes an existing IPv6-over-BLE stack, and supports both node and router functionality.

Chapter 3

Related Work

This chapter lists related work to this thesis. It shows already existing solutions of IPv6 over BLE on other operating systems. It also presents studies related to this thesis.

3.1 Existing 6LoWPAN Border Router

The border router functionality is already implemented in some operating systems. However, most of these operating systems are designed for constrained devices, which are used as sensor nodes inside the IoT. Examples are Zephyr [7], Contiki [35], or Android Things [36], where mainly the node functionality is used. In addition, the devices which are using these OS are usually not well suited as border routers. The major reasons for that are the restricted connectivity, the constrained power supply, and the low processing power because of the energy-efficient design.

The Zephyr OS is an open source real time operating system. It is developed by the Linux foundation for the use inside the Internet of Things. It entails support for IEEE 802.15.4, classic Bluetooth, and Bluetooth Low Energy, which includes also a full IPv6-over-BLE stack. It supports multiple architectures and is intended for devices with small memory size and constrained processing power, like system controls or sensor nodes.

Contiki is an open source operating system for constrained devices, and is well suited for use in the Internet of Things [35]. It is Internet capable, and contains a full 6LoWPAN stack for both IEEE 802.15.4 and BLE. Although it contains both support for node and router role, the functionality of using it as border router is not that often used because of the constrained power supply of the used devices.

Android Things is an Android-based embedded OS [36]. The target systems are low-power memory-constrained devices intended for the IoT. It supports both Bluetooth Low Energy and WiFi, and it further entails a LoWPAN API for both. However, the last offical version with new features and bug fixes was released 2018: after that, only updates with security patches were released. Also, no newer versions are announced, so I assume it has been discontinued.

Other OS, where the IPv6-over-BLE implementation is also available, are better suited as border router devices. The most usable implementation is the one of Linux for the

Bluetooth stack "BlueZ" [37]. It contains full functionality for node devices, as well as for router devices.

One of the most popular versions of Linux is Ubuntu, which is usually used on notebooks. Another widespread Linux based operating system, which also contains the IPv6-over-BLE implementation, is Raspbian, the OS for RaspberryPi devices. These two operating systems, combined with the device types, have almost the same practical relevance as the IPv6-over-BLE implementation at a smartphone or tablet.

Table 3.1 summarizes the properties of the software, as well as the properties of the corresponding hardware, and the usability of the IPv6-over-BLE features.

	Ubuntu / Notebook	Raspbian / Raspberry Pi
Node functionality	supported	supported
Router functionality	supported	supported
Border router functionality	supported	supported
Effort of Set-Up	low	low
Portability of device	medium	medium
Interoperable devices	high	medium
Supported BLE versions	4.0 / 4.1 / 4.2 / 5.0	4.0 / 4.1 / 4.2 / 5.0

Table 3.1: Properties of the IPv6-over-BLE implementation in Linux.

The node functionality is available for both devices. In terms of the IoT, the node functionality means not only the role inside the network, but also the providing of sensor data. In terms of sensors, both notebooks and Raspberry Pi devices do not have much offer, but additional sensors can be easily added via USB.

The router and border router functionalities are available at both devices and the resources of the devices are better suited for these purposes. The energy supply, the processors, and the RAM are designed for a high data processing rate. The energy supply also supports a long range and high radio duty cycle of Bluetooth and WiFi. Different to smartphones and tablets, notebooks and Raspberry Pi devices also contain LAN ports. Due to the high processing and connectivity capabilities, both devices are well suited as border routers.

The effort of setting up the devices as nodes or routers is low, because both operating systems already contain the Bluetooth 6LoWPAN module and do not need to be installed later. When the devices contain an appropriate BLE hardware chip, the node and router functionality can be used immediately.

The portability of the devices is a relevant factor, since sensor nodes can be placed at hardly accessible locations. The notebook with the Ubuntu OS is only partially suitable to act as mobile border router. The advantages are the compactness of the device, which includes screen, keyboard, and touchpad. Another advantage is that the energy supply lasts for several hours, maybe even for days with the right battery, and the use of a reserve battery is also possible. Because of these characteristics, the rating of Ubuntu running on a notebook leads to medium portability. For high portability, a device must also be easier to transport than a notebook and have a battery that lasts at least a few days.

The Raspberry Pi can be either well or not that well suited. Although the device itself is very small, if additional devices like a monitor, keyboard, and a mouse are included to be operable, the equipment is too extensive. Also, the power supply is normally not constructed for a mobile use. However, with a battery as power supply, and if the Raspberry Pi is used as headless border router, it is still very suitable. These conditions result in the assessment "medium portability".

The number of interoperable devices is very high for the Ubuntu version, since the requirements are very low. Nearly every notebook is suitable for this operating system. However, to use IPv6 over BLE, the device needs to support Bluetooth Low Energy. The number of devices which are suitable for using Raspbian OS is quite smaller. Nevertheless, out of all existing Raspberry Pi devices, everyone is suitable to support IPv6 over BLE. The main disadvantage of older Raspberry Pi versions (e.g., 1 and 2), is that they do not have an integrated Bluetooth or WiFi chip. For these, an external BLE and WiFi adapter is necessary.

Ubuntu supports all BLE versions to enable IPv6-over-BLE communication, although the versions 4.0 and 4.1 are less efficient due to smaller packet size. There exists a lot of notebooks that support every version of BLE, which is one of the main advantages of the nearly everywhere compatible Ubuntu OS. Raspbian also supports all BLE versions for the IPv6-over-BLE implementation, but it is partially bounded to the integrated BLE chips inside the Raspberry Pi devices. On the Raspberry Pi 3, the Bluetooth versions 4.1 and 4.2 are integrated, depending on the model version, and beginning with the Raspberry Pi 4 the Bluetooth version 5.0 is integrated. But, of course, it is possible to use an external BLE adapter, which exists for every available BLE version.

3.2 Related Research Studies

One of the first scientific papers concerning IPv6 over BLE is about adapting the Bluetooth stack of Linux. Wang et al. [38] implemented the first prototype system to enable the transmission of IPv6 packets over BLE, with changes in BlueZ, the official Bluetooth stack of Linux. They have evaluated the optimized communication that means use of context-based IPv6 header compression, which reduces the number of link layer packets to be transmitted. This reduction results in lower latency and lower power consumption, so overall it improves the transmission efficiency.

The experiment executed by Yoon et al. [39] shows that it is possible to use personal healthcare devices to send emergency data with IPv6 packets over BLE. The limitations are the standardless communication of the healthcare devices, and the difficulty of ensuring emergency data transmission due to possibly unpaired smartphones. To handle these limitations, they are using IPv6-over-BLE communication, which includes the use of IPv6 header compression, and the transfer of the emergency data by using the advertising process.

Since the network performance of BLE has a high potential, Spörk et al. [40] designed and implemented an IPv6-over-BLE stack with enhanced control over energy usage and timeliness, called BLEach. This stack contains new BLE functionality: adaptive radio duty cycling, IPv6-over-BLE traffic priorization and multiplexing, and indirect link-quality monitoring. It was the first open-source stack with full support for IPv6 over BLE, and the experiments proved that BLEach is interoperable with standard compliant stacks, as well as portable, modular, and energy efficient.

IPv6 over BLE mainly supports star topology, but the latest Bluetooth specifications also describe functionalities enabling enhanced topologies as well, so the Internet draft [41] describes the possibility of IPv6 mesh topology over BLE using Internet Protocol Support Profile (IPSP).

This draft assumes a mesh network with BLE link layer connections between neighbouring devices that support IPv6 over BLE, since a peripheral can connect to more than one central, and one device can be both a peripheral and a central device. In addition, IPSP supports that a device can implement both roles at the same time. The devices that must forward the packets need to implement node and router roles at the same time, while leaf-nodes only need the node role. Besides the protocol stack, the draft also entails descriptions of the subnet and link model, as well as security considerations.

Chapter 4

Creating an IPv6-over-BLE Stack for Android OS

This chapter describes the creation of an IPv6-over-BLE stack for Android OS. Section 4.1 describes the IPv6-over-BLE stack in general, including all layers and the connection establishment. Section 4.2 lists the requirements of an ideal implementation of an IPv6-over-BLE stack. Section 4.3 shows the integration into Android, describing the current Bluetooth architecture in Android and the structure of the new IPv6-over-BLE stack for Android. Lastly, Section 4.4 discusses the challenges of the integration of the new stack into Android.

4.1 IPv6 over BLE Stack

4.1.1 Communication Stack

The Bluetooth low energy stack has already been described in Section 2.1, and is shown in Figure 4.1. Besides the normal Bluetooth low energy stack, another stack in parallel enables IPv6-over-BLE support, which is located above the L2CAP layer. Until this point, both stacks share the same layers, the physical layer, the link layer, and the L2CAP layer. The exchange of IPv6 packets over a Bluetooth Low Energy connection is specified by the RFC 7668 standard [6].

In Figure 4.1, the layers that already exist but may have to be adjusted are colored in orange. The layers which have to be newly created are colored in red. The unchanged layers are splitted into controller colored blue, and host colored purple.

As already described in Section 2.3, the L2CAP layer needs to be used in credit-based flow control mode. This mode checks if the communication partner can receive new packets and then sends all fragmented packets consecutively in the right order. The credit number states the amount of packets a device can receive. This mode is used to control the flow of the LE-frames, so that the IPv6 packets are fully transferred. IPv6 packets may be bigger than the BLE package size. IPv6 has a header length of 40 bytes, and UDP has also 8 bytes. In BLE specification v4.0 and v4.1 the data field at the L2CAP layer has a size of 27 bytes, but in v4.2 and above it increases to 251 bytes [25]. In order to support the mandatory minimum IP packet length of at least 1280 bytes, a fragmentation

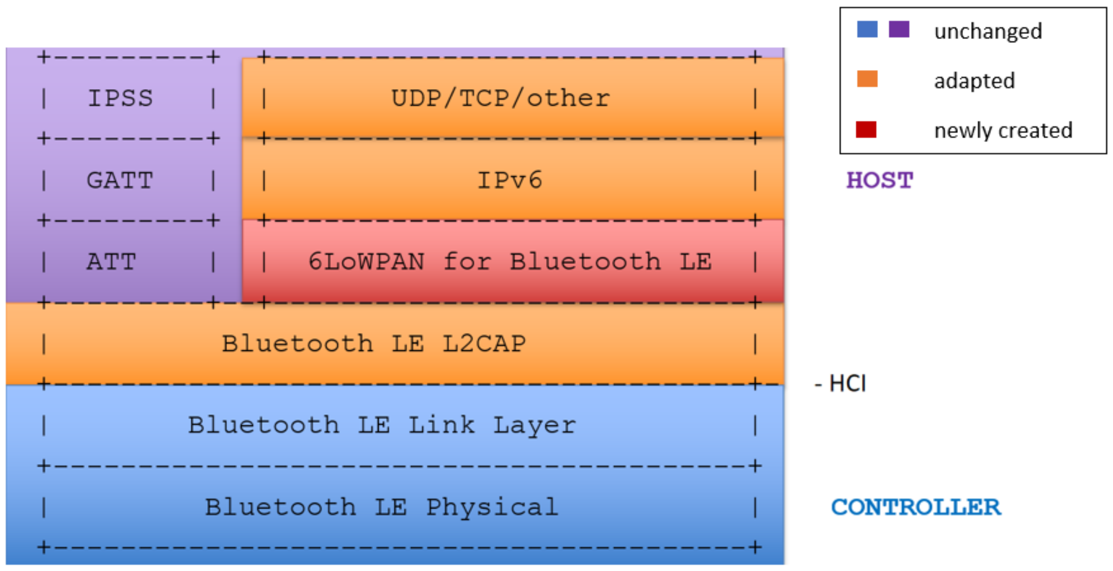


Figure 4.1: BLE stack for IPv6 and GATT (adapted from [6]).

and reassemble of the IPv6 packets is necessary. This task will be carried out by the already integrated features of the L2CAP layer. This enables higher layer protocols to send bigger data packets, which get split and sent to the controller in appropriate sizes dependent on the buffers. After reception, the L2CAP layer reassembles the fragmented data, reconstructs the message, and notifies the upper layers [3].

Above the L2CAP layer stands the 6LoWPAN for Bluetooth low energy layer. It contains the compression and decompression of the IP header and the address configuration. If the operating system does not already support IPv6 over BLE, this layer has to be created completely from scratch. In case the OS already supports IPv6 over IEEE 802.15.4, the existing 6LoWPAN layer can be adapted for use with BLE.

The header compression results in a header size of 2, 12, or 20 bytes, instead of the full 40 bytes of the IPv6 header. Information is either compressed, or partly / fully elided. The needed data can be restored by the receiver from internal informations, or reconstructed by context information. The detailed description of the header compression and the address configuration is shown in Section 2.3. Besides the IPv6 header compression, it contains also the compression of the next header of the message, which includes a compression of the UDP header. The whole compression format is specified in the RFC 6282 standard [20].

For address configuration, first a link local IPv6-over-BLE address is generated based on the devices' BLE address and a standard prefix. This address is used to establish the connection. After that, to generate and register the public IPv6 addresses, the border router distributes a shared IPv6 prefix. This prefix can be taken from the Internet service provider, or can be set by the user. By using this prefix, all nodes register their new public IPv6 address at the border router and are accessible by that.

Above the 6LoWPAN layer are the IPv6 layer and the UDP or other transport layer protocols. A lot of operating systems already entail IPv6 and UDP, so they can either be used instantly, or must be adapted. IPv6 also entails the neighbor discovery functionality, and this layer is the highest layer that needs to be supported. Possible layers above (i.e., UDP, TCP) are not mandatory and may be required depending on the application.

4.1.2 Communication Setup

The setup of the communication between the border router and the node device is divided into several steps. It is initiated from the node by sending BLE advertisement packets to the environment, and the router device selects the node and starts the connection establishment.

After the node has sent the advertisements, it waits for the router to establish the connection. If the node wants to connect to a border router, it has to offer the Internet Protocol Support Service [42] as GATT service. This service is used to recognize the support of IPv6-over-BLE connections. The node has to include the following connection parameters within his advertising packets:

- Flags (discoverability and availability of Bluetooth modes);
- Transmission Power (the transmission power level of outgoing packets);
- Device name;
- Service UUIDs (list of UUIDs of the provided services, Internet Protocol Support Service (IPSS) is one of them);
- Connection Interval (minimum and maximum value of the preferred connection interval).

The router receives the advertising packets and can derive from its properties that the node wants to connect to a border router for obtaining access to the Internet. After checking for duplicate link-local addresses, the router sends the connection request at the BLE link layer. This connection request entails the connection parameters and by receiving it from the node, the link layer connection is established.

After that, the connection setup goes up to the L2CAP layer. Within the L2CAP connection, the IPv6 packets get fragmented, sent, and reassembled. At this layer, the router has to start by sending an L2CAP connection request. The IPSS declares that the connection-oriented channel with credit-based flow control mode has to be used within the L2CAP connection. Therefore, the connection request contains also an initial number of L2CAP credits. Credits are a number of L2CAP packets that the device accepts inside the L2CAP channel. After the transmission of one L2CAP packet, the credit number is decremented and, if it reaches zero, no more packets will be accepted. New credits can be allocated with the L2CAP flow control credit packet. The request from the border router gets answered with an L2CAP connection response from the node. The L2CAP channel is then created successfully.

To discover neighbors, IPv6 over BLE uses the neighbor discovery from IPv6, optimized for 6LoWPAN and standardized in [21]. Because IPv6 over BLE does not support mesh networking, only the star topology has to be supported by the neighbor discovery. Since

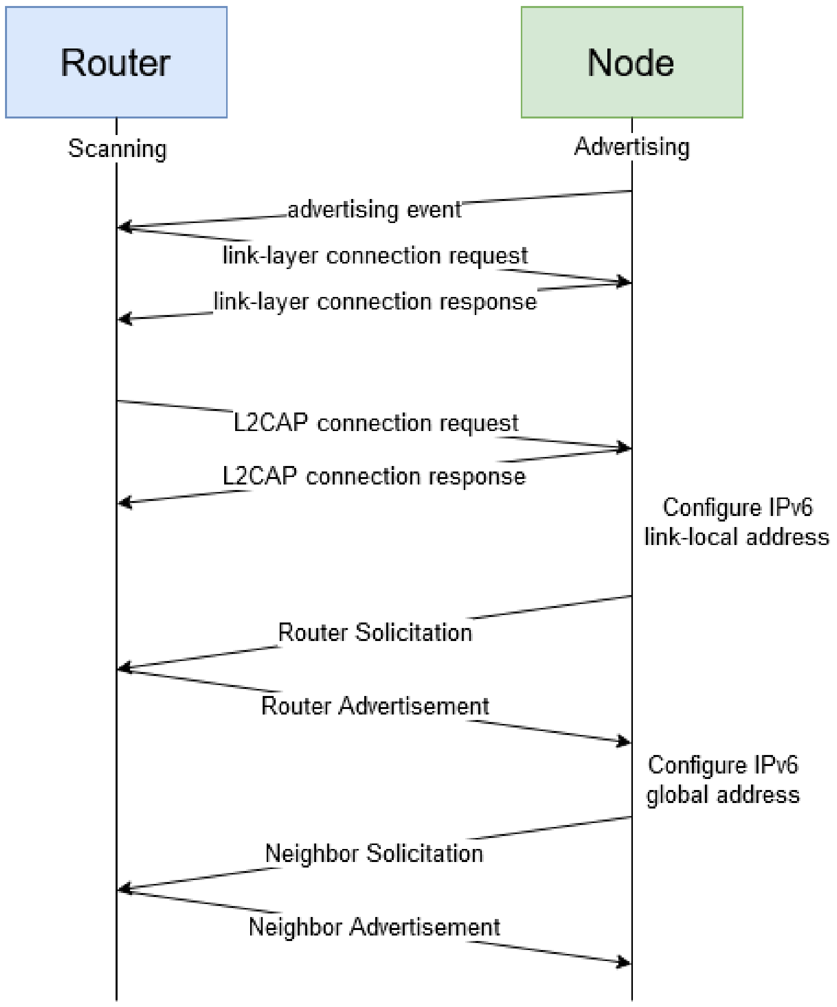


Figure 4.2: Steps of an IPv6-over-BLE connection setup between border router and node.

the L2CAP channel is already established, IPv6 packets can already get sent through this channel. The node sends the router solicitation (RS) message, which is an ICMPv6 message, based on IPv6, to the all-routers multicast address, if a connection to a router has not yet been established. The node waits for the according response. The router sends as a response a router advertisement message. The RA contains also the IPv6 prefix of the subnet with the prefix information option (PIO), and context informations for the IPv6 header compression with the 6LoWPAN context option (6CO).

With the prefix from the RA, the node can create the global address and register it at the border router. This registration is made with a neighbor solicitation (NS) message, with the address registration option (ARO). If the node has also another global address, it can also add it by an additional ARO option inside the NS message. After receiving the neighbor solicitation, the border router checks the global addresses for duplicates. If no duplicate error occurs, the border router acknowledges the successful address registration with the neighbor advertisement (NA) message.

4.2 General Requirements

This section describes the general requirements of an ideal implementation of IPv6 over BLE into an operating system.

- **Portability**

  An ideal implementation of IPv6 over BLE supports all devices with suitable hardware and the same operating system for which the implementation was developed, and it should be compatible with newer OS versions. Optionally, an implementation can be made portable to be used with different operating systems. The implementation should have an API that is usable by all applications running on the same operating system.

- **Scalability**

  An implementation should support as many IPv6-over-BLE connections in parallel as possible, to make the implementation usable by use cases that necessitate of parallel connections. Additionally, the implementation should support the simultaneous usage of both the normal BLE stack and the IPv6-over-BLE stack.

- **Efficiency**

  To guarantee efficiency and the resource availability for other applications in the OS, the IPv6-over-BLE implementation should consume little energy and little memory.

- **Interoperability**

  The implementation has to be conform to the RFC 7668 standard [6]. This mainly includes connection establishment and compression formats, and guarantees interoperability with other devices that have a standard-compliant implementation.

- **Usability**

  The implementation should be easily addable to the operating system, in case it is not already integrated in the OS at the time of installation. It should be installable without any specific access rights or changes inside the OS.

To support generality of the IPv6-over-BLE implementation, and small effort for setting up, the best solution is to create an own Android application. The application should be easy to install without having to root the smartphone. Other properties (i.e., scalability, lightweight, standard-compliance) can be supported through the implementation of this standalone Android application (i.e., supporting several IPv6-over-BLE connections in parallel, low energy and memory consumption, IPv6-over-BLE communication compliant to RFC 7668 standard).

4.3 Integration Into Android

Based on the requirements on an IPv6-over-BLE stack, we discuss how such a stack can be implemented in the Android OS. Since the L2CAP layer of Android does not support credit-based flow control mode so far, the adaptation of this layer is the first change that needs to be made on the Bluetooth stack.

Above the L2CAP layer stands the new 6LoWPAN for Bluetooth low energy layer. The 6LoWPAN layer has to be created completely from scratch, since the provided functionality is nowhere available at Android. This layer contains the compression and decompression of the IP header and the address configuration. IPv6 and the protocols above are already supported by Android. It is either possible to use the functionality implemented in Android, or to use other libraries offering these functions. In this thesis, we use the basic functionality of Android. In case more specialized or sophisticated IPv6 functionality is needed, existing Android libraries may be used.

4.3.1 Bluetooth Architecture in Android

To support IPv6 messages over BLE communications, the BLE connection needs to support credit-based flow control mode. To add this mode to the Bluetooth stack, access to the L2CAP layer is required. Android versions from 1 until 4 are using the same Bluetooth stack as Linux is using, the BlueZ [43]. With BlueZ, the access to the L2CAP layer from the Android application layer is possible.

So the L2CAP connection settings can be changed to use a credit-based flow control mode. However, starting from version 5, Android is using a new Bluetooth stack, called Bluedroid. Android is supporting a stricter safety policy in each version, so the access to the L2CAP layer of the Bluetooth stack gets denied from enabling the new Bluedroid stack from version 5. To enable IPv6 over BLE, the L2CAP layer itself and the interfaces to access the layer functions have to get changed.

The communication between the Android Application and the Bluetooth Stack is divided into several steps. It crosses several layers of the software stack of Android, which can be seen in Figure 4.3, and uses different interfaces to communicate between these layers.

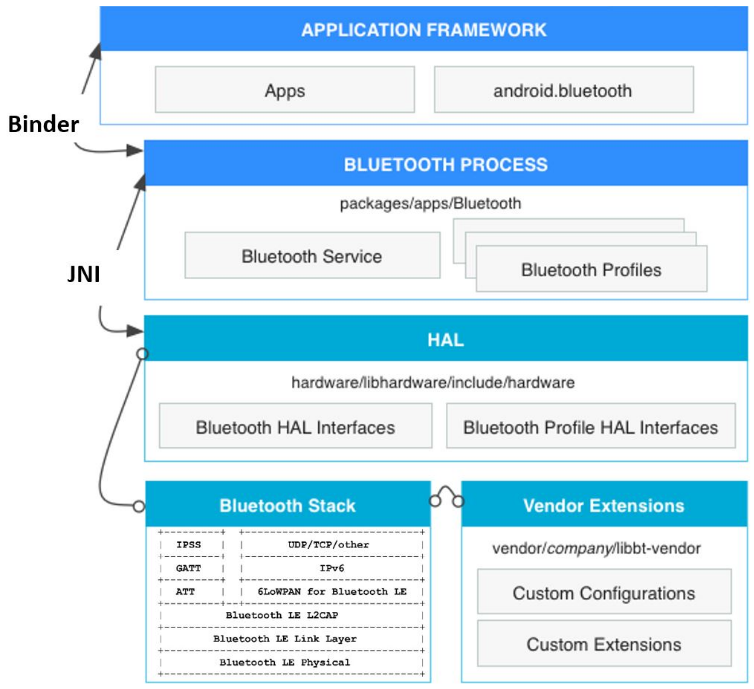


Figure 4.3: Bluetooth architecture of Android (adapted from [44]).

Vendor Extensions

On the bottom of the Android Bluetooth stack is the vendor extension layer. This layer is optional and can be used by manufacturers to implement vendor-specific functionality, extensions, or configuration for the BLE device. Manufacturers of Android devices can change some configurations, make extensions, or reimplement some functions for a better use of the product's own properties. To support the modifications of the manufacturers, the Android OS has an own location for these modifications with access to necessary subsystems of the OS, which includes also the Bluetooth communication. An example for a vendor extension is a so-called attention command for a Bluetooth Headset. This command causes the Headset to interact with the smartphone, which means that, for example, the battery level of the headset can be read out in order to show it at the OS [45].

Bluetooth Stack

The Bluetooth Stack is located as second lowest layer in the Android Bluetooth architecture, above the optional "vendor extensions" layer. The existing Bluetooth stack of Android entails an implementation for Classic Bluetooth and BLE. For our purpose we use Bluetooth version 4.2 [46]. It is 250% faster and enables an up to ten times higher packet capacity than version 4.1 [47]. The existing BLE stack does not support connection-oriented channels with credit-based flow control mode. It also does not entail a 6LoWPAN layer, which needs to be newly created. The stack is fully implemented with the programming language C++. The stack is called from upper layers through the BLE hardware abstraction layer, which also contains an interface to make the C++ functions available from Java.

BLE Hardware Abstraction Layer (HAL)

The BLE HAL is located between the Bluetooth stack and the Bluetooth process. It contains the interfaces of the Bluetooth stack (Bluetooth HAL interfaces) and of the Bluetooth profiles (Bluetooth profile HAL interfaces) inside the Bluetooth process layer. The interfaces of the profiles get called by the Bluetooth stack, which calls into the Bluetooth process layer through JNI. The interfaces of the stack get called by the Bluetooth services and Bluetooth profiles through JNI.

Java Native Interface (JNI)

While the Android Bluetooth process is implemented in Java, the HAL is implemented in C and C++. In order for these components to communicate, the Java Native Interface (JNI) is used. JNI is necessary if other programming languages besides Java are used and they are not provided by the standard Java class library. It is a clearly defined interface and platform independent. It enables to call functions and can be called by other languages. It is used to make calls into the HAL, to make calls from the HAL into the Bluetooth process, and it receives callbacks from the HAL if special Bluetooth operations happen, for example when devices are discovered.

Bluetooth Process

The Bluetooth process is located between the application framework and the HAL layer. It communicates with the upper layer through Binder, and with the lower layer through JNI. It consists of the Bluetooth service and the Bluetooth profiles.

The Bluetooth service communicates with the HAL through JNI and calls functions from the Android framework. Since the service is a separate process, it has to communicate with the app through an Android Binder, a component of the Android OS that provides inter process communication (described below). For our purposes, we adjust the already existing Bluetooth service. It gets created by the application and runs in the background, even if the app is closed. It is also possible to start a service when the Android OS boots, however, this is not necessary for our stack, because the IPv6-over-BLE connections can only be established through the App, and therefore also the service gets initiated with it.

The service notifies the user about updates to the Bluetooth connection, such as connection establishment and cancellation. The main part of the service is to forward all received IPv6 packets from the Bluetooth devices to the Internet and vice versa. This can be achieved by using the already existing implementation of the IPv6 layer in Android. The service just needs to use it in combination with the existing Internet connection and pass the IPv6 packets through the network interface.

All available BLE profiles like GATT or GAP are fully implemented in the original Bluetooth stack. But IPv6 over BLE does not use BLE profiles and therefore no profile is designated for this kind of application. The IPv6 module (which includes the ICMPv6 protocol) on top of the IPv6-over-BLE stack is sufficient.

As already mentioned, the Java Native Interface is used for the communication between components that are implemented using different programming languages. To make calls into the HAL layer and receive callbacks back from the HAL, the Bluetooth Service and the Bluetooth Profiles are using JNI. They are also getting called by the Bluetooth Profile HAL interfaces through JNI. For IPv6 over BLE only the original Bluetooth service uses JNI, because the existing profiles are not used and no new profiles are created.

Binder

The communication between the application framework and the Bluetooth process takes place through a Binder. The latter is a component of the Android OS that provides inter-process communication. It enables different processes to execute procedures in other processes. One needs to distinguish between client and server process. The server provides the methods which can be called by the client. But with parameters and return values, data can be passed in both directions. To make calls from the framework to the Bluetooth service and vice versa, it is also necessary to use a Binder for these communication tasks.

Application Framework

The highest layer contains the Android application and the framework that entails the function calls for the whole Bluetooth communication process. The Framework only calls functions from the underlying layers and processes data which can then be used by the application, and vice versa. The applications themselves are the top layer and the interface to the users. They process, forward, and graphically prepare data. User applications, such as the IPv6-over-BLE application, are installed on this layer only, and access to the lower layers is restricted.

4.3.2 Structure of IPv6-over-BLE stack

The best structured integration of new layers would be to place them in the same location as the original Bluetooth stack. This would require changes at nearly all layers of the whole Bluetooth architecture. However, access to these layers is restricted, and normal application developers do not have access to it. To make the according changes, full access to the whole Bluetooth architecture is required.

Besides the official Android factory images, where the source code is restricted, there exists the Android open source project, where the whole operating system is accessible and changeable (described in Section 2.4). This project is also officially offered by Android and contains nearly all functions of the same factory images, but it is open for regular developers to make changes and add additional functions. However, a big disadvantage of this changeable operating system is that, after every change to the system, the whole OS needs to be flashed to the smartphone (or tablet), which resets the whole device. This process is very time-consuming, especially when different devices other than the official Google ones are used. Thereby, implementing the functionality of this thesis over the entire Bluetooth architecture results in a high effort of setting up and using the implementation. To avoid this effort and enable an easier usage (and thereby the higher spread of the functionality), the use of an Android application alone to support IPv6 over BLE would be preferable. This would significantly reduce the effort by simply installing the Android application from the store or a third party source, without the need of resetting or rooting the device.

To make the whole functionality work from the application layer, the Bluetooth L2CAP layer must be accessible from the application itself. Since the only missing functionality of the L2CAP layer is the credit-based flow control mode, it is best to implement this mode directly on the layer, and add according interfaces to the remaining layers of the Bluetooth architecture. By doing this, the application layer does not need to have access to L2CAP: instead, it can simply use the providing interface of the application framework. Further functions of the IPv6-over-BLE communication, starting with the 6LoWPAN layer, are then easily implementable in the application.

Another reason for designing the whole 6LoWPAN layer and above layer functionality inside the Android application is that, however, in Android 8 there is a solution for L2CAP credit-based flow control mode. This solution is also not accessible from the application layer. With a workaround, it is possible to use this function, if the developer knows where it is and how to use it. The workaround is the so called „Java reflection“, which enables the access of not public members and functions.

Java reflection is a feature inside the Java programming language, which enables access to classes, interfaces, functions and members, which are declared private [48]. The purpose of this feature is that a program can examine itself, also for executing software tests where some members must be accessed and changed this tool can be used. For the IPv6-over-BLE implementation this feature is being misused.

4.4 Challenges

This section lists the challenges of integrating an IPv6-over-BLE stack in Android. A detailed description of how these challenges were tackled is provided in Chapter 5.

1. **No support of L2CAP connection-oriented channel in credit-based flow control mode in Android**

   The first challenge in creating the new IPv6-over-BLE communication stack in Android is that BLE at Android has no support for L2CAP credit-based flow control mode. Since the access to this layer is restricted for Android applications, this is the main hurdle for using only an Android application for the whole IPv6-over-BLE implementation. In order to add this mode to the L2CAP layer and make it accessible via the application, almost all layers within the Android Bluetooth architecture must be adapted and supplemented. This causes a lot more effort, because there are not only different programming languages involved, but also technologies to interoperate between these languages, layers, and processes, all of which have to be adjusted. To access all layers of the Bluetooth architecture, the so called Android open source project is used, where the developer has access to the whole operating system. After changing, the whole OS has to be flashed on a device. For the version 7 of the AOSP, which is comparable with the official version 7 of Android, an L2CAP credit-based flow control mode implementation has been made open source by a developer. This implementation fulfils the raw purpose of this L2CAP mode and can be used: only small adaptations have to be made to access it from the Android application.

2. **Access to L2CAP layer interface denied**

   The next challenge was that the access to the existing L2CAP credit-based flow control mode implementation was denied, because the function inside the application framework was not public. There are two ways to solve this. The first way would be to change the declaration of the corresponding function within the application framework from private to public. However, with this solution, a normal application developer would have to import the new application framework to build the application code. The second way would be to use the private function with Java reflection. The building of the application code works instantly, and the usage of the function works too. If the application gets installed on another operating system, for example the official Android 7, the Java reflection will throw an error because the required function does not exist there.

   In the official Android 8 operating system also an implementation of the L2CAP credit-based flow control mode exists. However, this implementation has the same problem as the implementation in AOSP 7: the function in the application framework is declared private. The solution of changing the application framework does not work for official Android versions, since the access to the application framework is denied. Therefore, only the workaround with Java reflection works for using the same application in Android 8.

3. **Internet access not allowed from main thread**

   The third challenge is an Android restriction that does not allow the main thread to access the Internet. In case of the border router functionality to forward packets from and to the Internet, this restriction is a problem, but it can be solved easily: creating a new thread to connect to the Internet.

4. **Handling several IPv6-overBLE connections at the same time**

   Challenge number four is the handling of several connections in parallel. The 6LoWPAN and above layers are implemented inside the Android application, which means the handling of the connection and the processing and forwarding of messages and data is also placed inside it. To guarantee the fastest processing of messages and handling of connections, each IPv6-over-BLE connection gets opened inside an own thread. This guarantees no timing and processing problems when several devices are connected simultaneous, and it implicitly solves the challenge of restricted Internet access from the main thread.

5. **IPv6-supported Internet connection**

   The last challenge can be the IPv6 connection from the border router device to the Internet. This is actually not a problem of the design of the application, but rather of the Internet service provider or the cellular network. When WiFi is used, both the Internet service provider and the Internet router must support IPv6. If LTE is used for Internet access, some adjustments of the LTE properties of the Android device have to be made. Also the mobile radio provider must support IPv6. If there is no support for IPv6, there are also some ways to still enable IPv6 communication, for example by using an IPv6 tunnel.

Table 4.1 summarizes the challenges and lists the possible solutions for two different Android versions: Android Open Source Project 7 (AOSP 7) and Android 8.

Challenge	Possible Solution
1) Missing L2CAP CoC support in Android	**AOSP 7**: Offers full access to BLE stack **Android 8**: Hidden L2CAP CoC support
2) Access to L2CAP interface denied	**AOSP 7**: Make interface public, or use Java Reflection to access interface **Android 8**: Use Java Reflection to access interface
3) Internet access not allowed from main thread	Create new thread to connect to Internet
4) Support several IPv6-over-BLE connections simultaneously	Create new thread for each connection
5) IPv6-supported Internet connection	Use appropriate ISP or mobile radio provider which supports IPv6, or use IPv6 tunnel

Table 4.1: Challenges of the IPv6-over-BLE integration into Android.

Chapter 5

Implementation

This chapter describes the implementation and all steps that are necessary to enable an IPv6-over-BLE communication on an Android-based smartphone, as discussed in Chapter 4. Section 5.1 lists the features of the IPv6-over-BLE implementation. Section 5.2 shows the implemented changes to the Bluetooth stack, both with and without direct changes inside the Android OS. Section 5.3.2 discusses the Android application itself, which contains the interface to the user and controls the IPv6-over-BLE connection. Section 5.4 describes the challenges that were showing up during all parts of the implementation. Finally, Section 5.5 presents the possibility to use this implementation on other Android OS versions, in addition to the two presented in this thesis.

5.1 Features

This section lists the features of the IPv6-over-BLE implementation in Android.

- Re-use of existing Android components

  The existing Bluetooth stack of Android does not need to get recreated from scratch, only up from the L2CAP layer. This layer, and all layers below, can be shared. This includes the device discovery, connection establishment, and the packet fragmentation. Also the whole existing Bluetooth stack can be used in parallel by GATT-based applications.

- Standard - compliant support for IPv6 over BLE

  The application is designed to support the standard features of IPv6 over BLE. This includes the connection establishment, the router functionality, routing, and routing to the Internet. The support of this functionality includes header compression and packet fragmentation.

- Compatible with existing Bluetooth functionality

  The implementation of this thesis is compliant to the standard RFC 7668 [6]. This means it contains the functionality for setting up an IPv6-over-BLE connection and exchanging IPv6 packets through this connection. It is fully interoperable with compliant node devices and routers.

- No rooting of phone

  One approach to make the IPv6-over-BLE application runnable on the smartphone, is to make the necessary changes inside the Bluetooth stack of the Android operating system and flash this OS to the phone. The other approach is to use the Android 8 factory image, which supports greater access to the Bluetooth stack from the application layer, whereby the application works without changes inside the Bluetooth stack. For flashing the OS, the bootloader of the smartphone must get unlocked, which is a straightforward console command. After flashing, the application must get installed to use the IPv6-over-BLE functionality. For the second approach, only the application must get installed at a smartphone with Android 8 on it.

5.2 Communication Stack

This section lists the changes and additions to the Bluetooth stack in general, as well as the changes specific to each version of Android.

5.2.1 Bluetooth Stack

As described in Section 4.1.1, the physical layer and the link layer can be simply reused. The L2CAP layer must support the credit-based flow control mode. This mode is implemented in Android, which means also the L2CAP layer can be reused. The next layer is 6LoWPAN for BLE, which has to be created from scratch.

Since the fragmentation and reassembly of packets gets executed on the L2CAP layer, only the compression and decompression of the IPv6 header must be implemented for the 6LoWPAN layer. The header compression is described in detail in Section 2.2.

6LoWPAN receives the reassembled packets from the L2CAP socket. The Bluetooth socket creation and connection setup is described in Section 5.2.2 below. The first step is to parse the first two bytes, they contain the IPHC base encoding. If an additional context encoding is appended, a third byte needs to be parsed. Because of the information gained from the base encoding, all IPv6 header fields can be restored. Either the field is carried fully in-line directly behind the base header, the field is partially elided so that only a part of it is carried in-line, or the field is fully elided. After the IPv6 header fields are decompressed, the next header fields can be decompressed and processed. In case of ICMPv6, the next header is not compressed. The handled UDP header is compressed and, therefore, decompressed and parsed accordingly. The current implementation of 6LoWPAN only supports UDP header compression, which is the most popular transport protocol for 6LoWPAN applications. Adding support for other transport protocols and extension header fields is part of future work. No further message types are required for the experiments in this thesis, but can be enhanced easily inside the decompress and parsing methods because they are designed for extensions. The remaining part of the message can be considered as payload. No further decompression is needed, and further processing is part of the according layers.

5.2.2 Integration In Android

To integrate the new Bluetooth stack into Android, the binding between the existing, the reused layers, and the new layers is important. In this case, the binding element is the L2CAP layer: this layer and all layers below can be reused. The access to this layer is different between Android Open Source Project version 7 and Android 8.

Android Open Source Project version 7

As already mentioned, the only necessary change in the L2CAP layer is the activation of the connection-oriented channel in credit-based flow control mode. As mentioned above, standard Android 7 does not support this feature of L2CAP. The Android Open Source Project, however, implements this support [15]. Nevertheless, the implementation is hidden per default and therefore not accessible from the Android app. There are two possibilities to make the L2CAP connection-oriented channel usable.

The first possibility is to change the accessibility of the L2CAP function and change it to public, so that it is accessible from Android applications. This function creates an L2CAP server socket, which entails the connection-oriented channel in credit-based flow control mode. However, making this function accessible, which is a change in the interface between Android applications and application framework, results also in additional effort while programming the application itself. To be able to compile the programmed application with the use of this new function, an Android SDK must be used, which includes this function. Since the official SDK for Android 7 does not include this function, a new advanced SDK must be generated from the modified Android Open Source Project. This is then integrated into the development environment and ensures that the application can compile using the L2CAP Connection Oriented Channel (CoC) socket function and works on the smartphone with AOSP 7.

The second possibility is to use Java Reflection to access the non public function of the L2CAP layer. By using this, the Android Open Source Project can be flashed onto the device directly without any changes. This reduces the effort enormously, not only because the interface does not have to be changed, but also because the effort of generating and including the changed SDK is completely eliminated. To use Java reflection, the method name and parameters of the hidden function must be known, because the function is not supported by the SDK, and therefore not known by the IDE when programming the application.

Regardless if the first or the second option gets used, when calling the function in the Android application, a Bluetooth socket with the parametrization of an L2CAP connection-oriented channel in credit-based flow control mode is returned. This Bluetooth socket gets used to enable a connection to a suitable Bluetooth device, which has to accept the connection request. A hexadecimal value must also be passed to the function, which specifies the protocol to be supported. For this implementation, this is the value 0x20023. It results from the L2CAP mask for LE Connection Oriented Channels 0x20000, and from the value for the Internet Protocol Support Profile 0x0023 [49]. IPSP defines the communication between nodes and routers with IPv6 over BLE. It helps to find compatible devices which also support this kind of communication. The protocol service multiplexer

(PSM) uses this value to distinguish between the protocols. This allows the IPSP data to be processed in the application.

When scanning for available BLE devices in the environment, also the supported Bluetooth profiles are visible. So in this case, it is possible to filter devices which support IPv6-over-BLE communication, because the IPSP protocol is included in the advertisements. But it is not absolutely necessary to filter the available devices according to that, because if they do not support it, the connection establishment fails anyway.

Android 8

In the official factory image of Android 8, the functionality of establishing an L2CAP connection-oriented channel in credit-based flow control mode is implemented, but it is also not accessible to the normal Android application. Since the source code of the factory images is not accessible, this functionality cannot be made visible to the Android application.

The hidden interface can be accessed using Java reflection, just like AOSP 7, although the function names differ. In AOSP 7 it is named `createL2capSocket`, in Android 8 it is called `createInsecureL2capSocket`. The function parameter is the same, i.e., the hexadezimal value of the IPSP protocol. This means that the same Android application can be used like in the Android Open Source Project, and only the function name has to get exchanged. In Android applications it is possible to check for API levels. Every Android version and subversion has its own API level. It is not possible to check whether it is an official factory image, or a version of the open source project, but it is possible to generally check for the API level. This means it is possible to differ Android version 7 and Android version 8, and call the corresponding L2CAP function for each Android version. At the official factory image of Android 7 this would still not work, but, with this, it is possible to support these different OS versions with just one Android application.

Comparison

Comparing the two used operating systems shows various advantages of the respective versions. The official factory image of Android version 8 supports more manufacturers and devices, and the effort of enabling IPv6-over-BLE communication is quite smaller than when using AOSP 7. With a device supporting Android 8, the user just needs to install the developed application, provided the device supports at least BLE version 4.0. There is no need to root the device or to install a modified OS. With AOSP 7, the effort involved in downloading, changing, building, and flashing the OS is considerably greater. However, this OS version offers a possibility that is not feasible with Android 8. Since the entire source code of the operating system is accessible, also additional connection parameters for the L2CAP CoC are changeable: the maximum payload size and the credit size. At the implementation at AOSP 7 and Android 8, these parameters cannot be changed via the interface, and the default values are very low, at 27 bytes MPS and credit size 1.

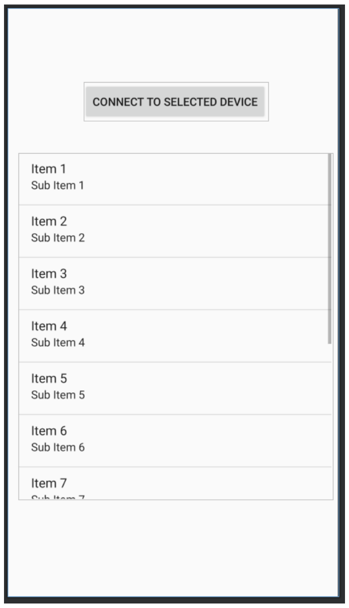


Figure 5.1: Design preview of the Android application with only necessary features, a scrollable list which gets updated only once at the beginning, and a button to connect to the selected device.

5.3 Android Application

5.3.1 Design

The main tasks of the Android application are the interface to the user and the control of connections. The connection handling includes the listing of available devices, the initiating of the connection establishment, and the automatic connection setup. It also informs the user about the connection state, and the user can decide how established connections should be used, or terminate the connections. The application enables also routing between connected devices, and routing to the Internet. This thesis presents an app that implements all necessary features to support IPv6 over BLE. Due to the seamless integration of all IPv6-over-BLE components into Android, other developers can easily use these components to create more sophisticated apps.

Necessary features (Figure 5.1):

- List of devices: After scanning the Bluetooth signals, the application should generate a list of available Bluetooth devices in the proximity. Either the application generates a list with all devices, and the items which support IPSS are denoted separately, or a list containing only the devices which support IPSS. All listed items are touchable, to select the device to which the user wants to connect with.

- Connect button: After selecting one device, the user has to push the connect button to start the connection establishment. Inside this application, only the IPv6-over-BLE connection is valid. If the connection setup fails, the whole connection so far gets aborted.

Optional features (Figure 5.2):

- Refresh button: After starting the application, it automatically searches for Bluetooth devices in the proximity. The search duration is about a few seconds. After that, the resulted device list stays constant and does not change anymore. The refresh button starts a new search for Bluetooth devices, adds new discovered devices and removes devices that are no longer available.
- Disconnect button: After selecting a device, the user has to push the disconnect button to terminate the connection to this device. Without this button, either the node has to terminate the connection, or the whole application must be closed.
- Highlight selected device: After listing the available devices, the user can select one of them to which the device should connect. By touching the item from the list, this device gets the current selected one. To show the user that the selection was successful, and to which device a connection will be established, the item is marked in color. Before manually starting the connection setup, the user can change his choice by just touching another item from the list.
- Highlight connection status: To inform the user if the connection is either not established, currently established, or the device is already connected. Only IPv6-over-BLE connections are supported.
- Prefix text field: The used prefix for public IPv6 addresses can be directly set by the user. For the testcases a hardcoded value would be sufficient, but it is more user friendly and sustainable to set the prefix directly at the user interface.
- Enabling Bluetooth reminder: If the application starts and Bluetooth is not yet activated, this is a possible error source, and besides that, also annoying. To avoid that, a simple dialog should occur after starting the application, if Bluetooth is not activated. It simply offers the Bluetooth activation and avoids that the user have to do it manually.

To test the IPv6-over-BLE implementation, also the UDP echo server application has to be included. The echo server application mirrors received UDP packets, which are received by the application, and are addressed to this device. If the address belongs to another device, the router functionality should forward the packets to the real target, if a connection to it is established by the router. Otherwise, an error message will occur notifying that the target is not known. The echo server functionality is independent from the user interface, since it is only automatically used if the node device is using the corresponding UDP echo client application. If the node sends other IPv6 packets, such as an ICMPv6 echo request, the router processes it normally and answers with an according ICMPv6 echo response.

5.3.2 Implementation

This section presents the Android application, more precisely how the user interface and the connection control have been implemented. The usage of the L2CAP layer and the implementation of the above layers are already described above in Section 5.2.1 and 5.2.2.

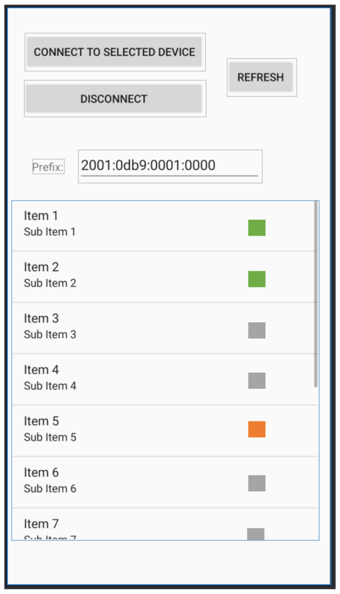


Figure 5.2: Design preview of the Android application with optional features besides the necessary ones. The GUI contains a refresh button, a button to disconnect from the selected device if it is connected, a field for entering IPv6 prefix, and a colored status information field for the connection: grey means not connected, orange means connection setup in progress, and green means IPv6-over-BLE connection enabled.

To support AOSP 7 and Android 8 with the exact same application, only the connection setup needs to differ between Android versions. The layers above L2CAP, the user interface, the connection handling, threading, routing, and Internet routing are working equally for both OS versions.

To be able to support all functions, the application needs several permissions [50]. First of all, the BLUETOOTH permission, which allows the application to connect to paired Bluetooth devices, is needed. Furthermore, the permission BLUETOOTH ADMIN allows the application to discover and pair with Bluetooth devices. Additional to Bluetooth also the ACCESS COARSE LOCATION permission is used, because of the possible approximate location determination with Bluetooth. Also ACCESS NETWORK STATE is used to access information about networks, like connection state in Bluetooth. Finally, the INTERNET permission to access the Internet is necessary.

The main thread of the application creates the user interface and checks if Bluetooth is switched on. If it is not, the user is asked via a dialog to activate it. The user interface is constructed with one page and basic functionality to show the feasibility of an IPv6-over-BLE application on Android. The interface contains a list of available BLE devices in close proximity. This list is automatically filtered and only shows devices that offer the IPSP protocol, which means that they support IPv6 over BLE. When scanning available BLE devices in the proximity, the advertising data contains UUIDs of the supported Bluetooth GATT services of the advertising devices, including IPSS.

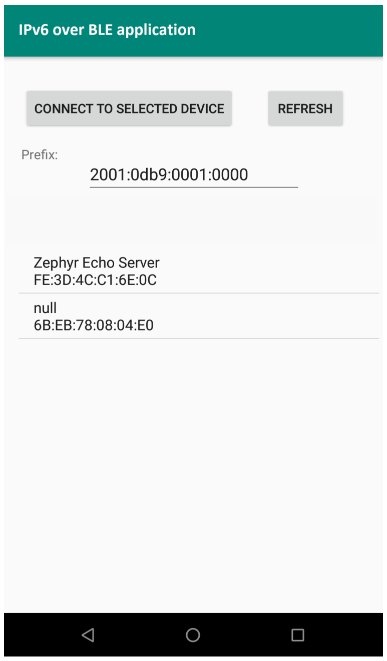
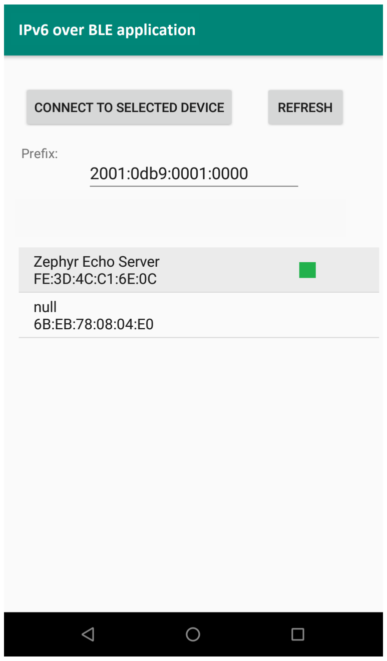


Figure 5.3: Screenshot of the app's user interface with two devices in proximity.

If the list contains more items than the screen can show, it becomes scrollable. The items of the list are selectable: each item shows the device name and Bluetooth device address, with the currently selected item highlighted in color. If the connection to a device is established, a green icon occurs besides the according item inside the list. The next important part of the interface is the Connect button. Pushing this button initiates a connection setup between the remote device and the smartphone. The only other button in the interface is the Refresh button. The list of available BLE devices in the proximity is generated after the launch of the application. The first proximity scanning is executed right after the launch. After that, the scan for BLE devices, and therefore the update of the device list, gets only executed when the Refresh button is pressed. The last part of the interface is the prefix field, which contains the IPv6 prefix that can be entered manually by the user.

The user interface hence contains basic functionality for establishing IPv6-over-BLE connections and handling of data received through it. As mentioned in Section 5.3, there are necessary features and optional features. The necessary features that have been implemented are the list of available devices and the connect button. The interactions regarding the connections themselves are basic. The connection establishment can be initiated, and a successful establishment results in showing a green status icon. No further actions on the connections are necessary. If the connection terminates, the green icon inside the device list disappears. These features are sufficient for the experiments.

When initiating the user interface, the callback functions of the buttons get assigned. The callback function of the Connect button does not only initiate an IPv6-over-BLE connection setup, but it also creates a new thread where this connection initiation takes place. This thread gets then inserted into the routing table, which is a list of all connected



Prefix:
2001:0db9:0001:0000

Zephyr Echo Server
FE:3D:4C:C1:6E:0C 🟩

null
6B:EB:78:08:04:E0

Figure 5.4: Screenshot of the app's user interface with two devices in proximity, one is selected, and the connection is established.

devices with the according addresses and threads.

This implementation also supports IPv6 routing functionality, both inside the BLE subnet and between the BLE network and the Internet. After restoring the destination address of incoming packets, the application needs to differ between messages targeting the router device itself, or messages with a different destination, using the smartphone as router. If the target is inside the BLE subnet, the routing table is used to forward the message through the right Bluetooth socket to the associated Bluetooth device. But if the target address is not inside the BLE network, the smartphone has to forward the message to the Internet. Provided there is an active Internet connection that is IPv6 capable, the message will be forwarded accordingly. Otherwise an error message is sent back.

To ensure the functionality required for the experiments, an UDP echo client program is implemented using the UDP layer. This program is only used if an UDP packet with the target address of the smartphone is received. Then the received packet gets answered with a message targeting the sender of this received packet with the same payload. This program is implemented encapsulated, which means it can easily be replaced by another application.

5.4 Implementation Challenges

This section discusses the challenges that were faced while implementing the designed application. Challenges which already came up when creating the IPv6-over-BLE stack for Android in Section 4.4 are not entailed. The main challenges that emerged are:

- Lack of Documentation of Android Open Source Project;

- Configuration of the BLE connection parameters;

Android Open Source Project

Using the Android Open Source Project means using the source code of the whole operating system of Android. Since the documentation of the operating system itself and the documentation of the code is very crude, this includes a lot of read-in and testing. During research and read-in into the code, it became clear that an implementation of an L2CAP CoC for the AOSP had already been developed by an open source developer. However, this implementation is not accessible from the application layer, so it is not directly usable from an Android application. There were two possible solutions for that problem: make this existing implementation public or use Java reflection. Making it public results in changes of the interface inside the framework, and therefore inside the software development kit. For official Android application development within an intelligent development environment (IDE), the official application framework for the supported Android version is used to ease the development and to build the code. So if the framework gets changed, the IDE also have to include the new changed SDK for developing and building the code according to the new version. By comparing both possibilities, Java reflection has more advantages.

The research showed that the official factory image of Android version 8 also contains an L2CAP CoC implementation, which is also hidden from the application layer. Same as AOSP 7, the hidden interface can be accessed using Java reflection. So after Android versions 5, 6, and 7, which were not supporting this functionality to enable IPv6 over BLE, AOSP version 7 and the official factory image version 8 were suitable to implement IPv6-over-BLE support.

BLE Connection Parameters

There are several parameters that influence the energy-efficiency and timeliness of the whole connection. When creating the BLE connection, certain parameters can be set, and also when setting up the L2CAP CoC connection. The connection interval can be set by the node device during the BLE connection establishment.

When setting up the L2CAP CoC the credit size and maximum payload size values are set by the router device. But the interface for creating the L2CAP Bluetooth socket does not take these parameters to be set at connection establishment, instead always the standard values of credit size 1 and MPS 27 bytes are used, which are improvable considering the efficiency. In Android version 8 these values are not changeable, since the access to the L2CAP layer implementation is restricted. But in the Android Open Source Project full access to all layers is given, which means that the values are changeable and the connection can be made more efficient.

5.5 Portability

This section provides an overview on how to use this implementation of the IPv6-over-BLE communication on other versions of the Android operating system.

This implementation is working for the Bluetooth architectures of AOSP version 7 and the factory image of Android 8. The application itself works for both versions, and does not need to be adapted to work for one or the other. It checks the Android version and uses the according connection setup with credit-based flow control mode. The application should continue to run for newer versions of Android, because newer versions support applications of older API versions.

The application can be divided into several parts: the user interface, the thread handling for the connections, the establishment of an L2CAP CoC in credit-based flow control mode, the header compression, the tasks of layers above 6LoWPAN like neighbor discovery, router discovery, and the functionality of layers above IPv6 like the UDP echo server application. The L2CAP connection establishment is important for portability, and it must be differentiated for each Android version. For the newer Android versions, i.e., version 9 and version 10, the L2CAP function call would have to be adapted to establish the connection. Either this function is implemented in the new versions and is also officially supported, or it is implemented but not public accessible like in version 8. If this function is not implemented in the newer versions, only the use of the Android Open Source Project for these Android versions remains. Either the implementation of version 7 or 8 is reused in the AOSP, or it has been newly implemented by other open source developers. If this is not the case, it is possible to implement this functionality from scratch in the newer versions in the AOSP.

No matter which of these options are used, the application itself only needs to be adapted to the function call for the L2CAP connection. The other parts of the application mentioned above should then be able to be used again without any problems.

Chapter 6

Evaluation

This chapter contains the evaluation of the IPv6-over-BLE implementation in Android 7 Open Source Project, as well as in Android 8. It is subdivided into several experiments, where the first one, Section 6.1, contains the comparison between router devices and OS versions. One node, for which the round trip time and the average power consumption are measured, is connected to the router device. In addition, Section 6.2 analyses the effect of the parameters maximum payload size and credits on the round trip time. Section 6.3 describes the experiment with several nodes connected simultaneous. Section 6.4 describes the Routing between two nodes inside the BLE subnet. Further measurements are made in Section 6.5 at a Long-Time Experiment to check energy and memory behaviour, at using another GATT based application in parallel in Section 6.6, and at connecting a node to the Internet by using the Android device as a Border Router in Section 6.7.

6.1 Energy Consumption and Timeliness

This connection measurements contain one node device (slave) and one router device (master). The programs running on these devices are UDP echo server (router) and UDP echo client (node). The echo server application listens for incoming UDP packets with a UDP message size of 8 or 512 bytes and sends them back. The echo client application generates packets and sends them, waits for the packets to be sent back, and verifies if the data is the same.

6.1.1 Experimental Target

The targets of these tests are the successful connection establishment and data transmission, and the comparison between the Android versions, different Android devices, and the existing IPv6-over-BLE solution of Linux. The comparison parameters are the average power consumption, the round trip time, and the packet reception rate.

6.1.2 Experimental Setup

The setup used for this experiment is shown in Figure 6.1. It contains the node device Nordic Semiconductor nRF52840-DK with BLE version 5.0 and has the UDP echo client application on it. The operating system on the node device is Zephyr. As router device,

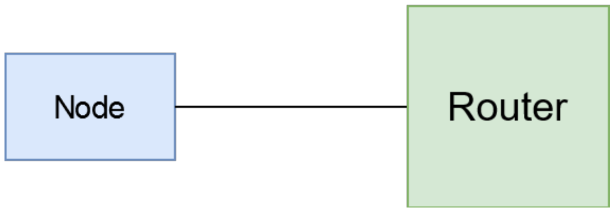


Figure 6.1: Network topology for the connection measurement.

an LG Nexus 5X smartphone with a Bluetooth 4.2 chip is used, which contains the UDP echo server application as part of the Android application.

The average power consumption gets measured on the nRF52840-DK with a Monsoon Power Monitor [51]. To get an accurate measurement, all unnecessary features, such as logging, are deactivated. The round trip time gets also measured at the node device, the latency between sending and receiving UDP packets gets calculated inside the UDP echo client application.

As described in Section 5.2.2, in the AOSP 7 implementation, the parameters MPS and credits can be increased, the default values are 23 bytes maximum payload size and credit size 1. With the AOSP 7 implementation the standard values (MPS 23, credits 1), as well as the modified values (MPS 123, credits 10) gets compared. For the Android 8 version of the implementation, only the standard values can be used. As additional Android device, also the Samsung Galaxy Tab S3 tablet with a Bluetooth 4.2 chip and Android 8 on it gets added to the test set. The maximum link layer packet length of the smartphone LG Nexus 5X, of the tablet Samsung Galaxy Tab S3, and of the Nordic Semiconductors nRF52840-DK sensor node is 251 bytes. The maximum link layer packet length of the Raspberry Pi 4 Model B is 27 bytes. Baseline values for the results from the Android devices are the measurement results of the IPv6-over-BLE implementation in Linux, executed on a Raspberry Pi 4 Model B with a Bluetooth 5 chip.

6.1.3 Result and Analysis

Figure 6.3 shows the measured average power consumption of the nRF52 node in milliwatt. The different bar colors show the different BLE devices used as border router. The node is connected to one border router at a time (see Figure 6.2). The different payload length (PL) and connection interval (CI) settings are shown on the x-axis. The figure shows the average power consumption values over two minutes for every device version and combination of packet length / connection interval. Each iteration was repeated three times.

As the values show, a shorter connection interval consumes more power than a longer connection interval, since it results in a longer active time of the BLE radio in total. Regarding the packet length, a shorter packet length consumes less power at the experiments with the Rasperry Pi, and the AOSP 7 with enhanced connection parameters as border routers. These two devices have higher MPS and credit values, which results into a higher data transmission rate, and therefore a higher power consumption. In opposite, the Nexus 5X devices with AOSP 7, Android 8, and the Samsung tablet with Android 8 consume less

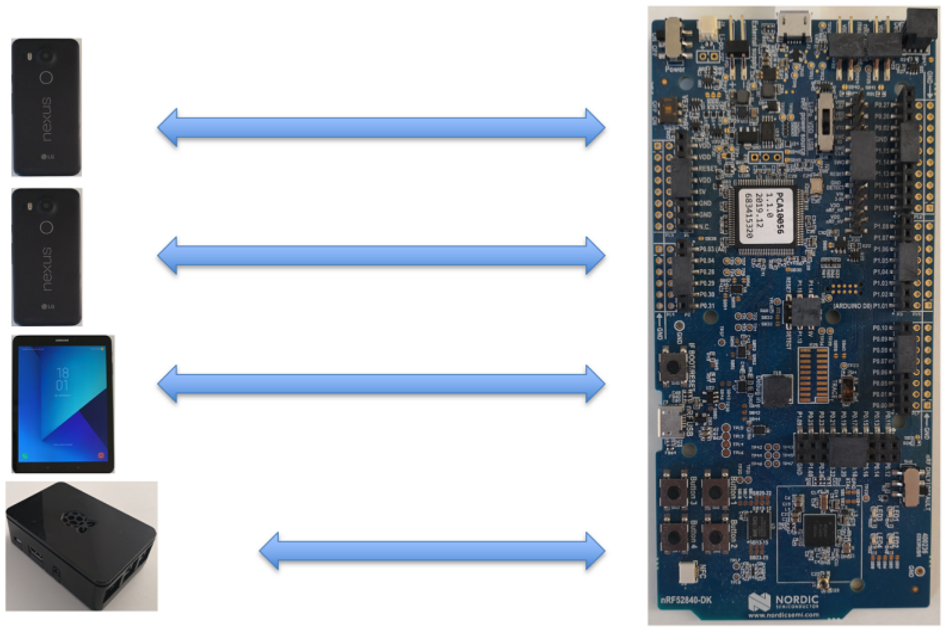


Figure 6.2: Connection test setup with Nordic Semiconductors nRF52840-DK as node device. As router device the LG Nexus 5X, Samsung Galaxy Tab S3 tablet, and the Raspberry Pi 4 Model B are used. For each experimental session only one router is connected to the node device.

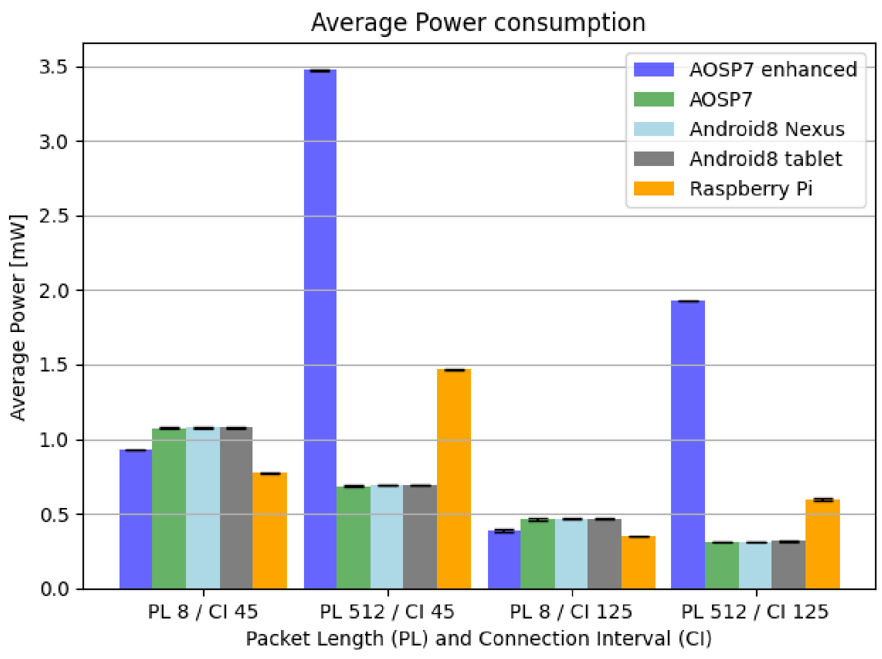


Figure 6.3: Average Power Consumption of the node device connected to the according router device.

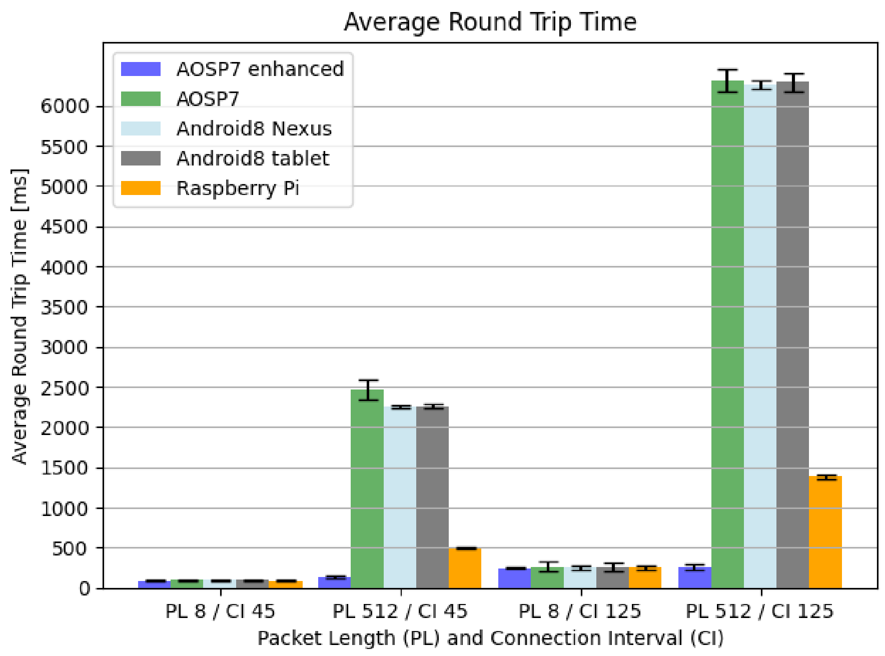


Figure 6.4: Round trip time of the UDP messages sent by the node device to the according router device.

power when transmitting longer packets. The reason for that are the low MPS and credit values. When transmitting packets bigger than the maximum payload size, the fragmentation needs a credit amount higher than 1. When using the flow control credit command to increase credits, it gets send in the next connection event, and the next fragmented packet is sent in the connection event after it. This results in a significant longer round trip time, but also decreases the power consumption.

Figure 6.4 shows the measured round trip time in milliseconds for the same experintal setting. Combinations of packet length / connection interval, and devices with OS are the same as in Figure 6.3.

In contrast to the power consumption, the RTT is shorter at smaller packet lengths, and also shorter at smaller connection intervals, since longer active time increases the throughput of data. So the minimum RTT occurs at small packet length and short connection interval, and the maximum RTT occurs at big packet length and long connection interval.

Independently of length of packets and connection intervals, the packet reception rate is 100% in every testcase.

In the implementation for AOSP 7 and Android 8, all sent packets are mirrored correctly by the router and received by the node. The UDP echo client program provided by Zephyr can be seen as RFC7668 compliant and fully functional, so the acceptance of the mirrored UDP messages is the additional proof of the successful connection setup, receiving and sending of packets, IP header compression, and next header compression. Therefore, the Android implementation is fully interoperable with other RFC 7668-conform implementations.

The same assumption can be made based on the similarities between the measurement results of this implementation and the IPv6-over-BLE implementation in the Raspberry Pi. In the testcases with packet length 8 bytes, all implementations are nearly the same with only a small dispersion of maximum 17 ms. The differences between these implementations are the credit-size of the packet transmissions, the MPS size, and the link-layer maximum packet size. But in the testcases with short packet length these differences are not time relevant. At longer packet lengths or longer connection intervals, the measured RTT is up to 5 times longer than the RTT at the Raspberry Pi. The only implementation where the values are lower is the modified AOSP 7 version. The explanation about the differences of the RTT values is the difference of the credits and the MPS values. With AOSP 7 the credit size was set to 10 and the MPS to 123 bytes. When using the Raspberry Pi, the credit size was set to 6 and the MPS was set to 230 bytes. Lower credit values result in additional requests for more credits, which means more messages need to be transmitted and more connection intervals are necessary. Lower MPS values or lower link layer packet lengths result in more packets to transmit: therefore, more connection intervals are necessary. The comparison between different MPS and credit values are described in Section 6.2.

Furthermore, it can be seen that the use of the same implementation on different devices (i.e., the Android 8 version on the Nexus 5X and on the Samsung Galaxy Tab S3 tablet) does not make a difference. Also using different OS with the same implementation (i.e., in the AOSP 7 or the Android 8 version) does not result in different RTT values.

In the average power consumption analysis the results are similar compared with the results of the RTT measurements. With short packet length the average power consumption is higher than the consumption with the Raspberry Pi, but with long packet length the average power consumption is lower than with the Raspberry Pi. The reason for that behaviour is the same: the difference with the credits and MPS values. The proof for that is again the better measurement results at the improved AOSP 7 version. The highest value was measured at the improved AOSP 7 version at 512 bytes packet length and 45 ms connection interval. Because of the high MPS value and the big credit size, more bytes can be transmitted compared with the other Android versions, i.e., more bytes are transmitted over the same time. Compared with the Raspberry Pi, the bigger value of the maximum link layer packet size, 251 instead of 27 bytes, enables a higher data transmission. This can also be seen in the round trip time values of the same packet length and connection interval of the improved AOSP 7 version and the Raspberry Pi, which is with 138,76 ms only a third of the RTT of the Raspberry Pi with 497,88 ms.

Possible differences between devices are also not noticeable, and the same between the Android OS versions with the same implementation.

6.2 Comparing MPS and Credit Parameters

In AOSP7 and Android 8 the default values of credits and MPS are set to 1 and 23, and they are not changeable in Android 8. However, by using the AOSP, as described in Section 5.2.2, these values are changeable.

6.2.1 Experimental Target

The target of this test is to show the differences in the combinations of the MPS and credit values, and displays which RTT and power consumption values result from the parameter changes.

6.2.2 Experimental Setup

The setup for this test is the same as for the test in Section 6.1, and can be seen in Figure 6.1 and 6.2. However, only the Nexus 5X smartphone is used, as it is possible as a Google device to flash the AOSP on it, and the changes of the MPS and credit values are only changeable at the AOSP. The used packet length is 512 bytes long, and the used connection interval measures 125 ms. The maximum link layer packet size of the Bluetooth low energy version of the LG Nexus 5X is 251 bytes.

6.2.3 Result and Analysis

The results for this experiment can be seen in Figure 6.5 and 6.6. When using a MPS of 23 bytes, the whole UDP packet with length of 512 bytes gets fragmented into a high number of fragments. Therefore, the power consumption is higher with credit size 10, because there are more fragments which gets transmitted in one connection event. When using credit size two, only two fragments are sent within one connection event.

When using a MPS of 123 bytes, the usage of credit size 2 results in even a higher power consumption than the credit size 10. The reason for this is the disadvantageous assignment of the fragmented packets to the connection events. Since the round trip time when using credit size 2 is almost the same as when using credit size 10, the assignment of fragments to the connection events is the same. Therefore, the reason for this increased power consumption when using credit size 2 is the additional flow control credit command. This command must be sent because the UDP packet length of 512 bytes when using a MPS of 123 bytes leads to more than 2 fragments.

When using a MPS of 256 or 512 bytes, the power consumption is nearly identical. The reason for that behaviour is that the fragmentation of the 512 bytes long UDP packet and the long connection intervals of 125 ms. This is to be expected, as the overall BLE radio time does not change significantly for different MPS and credit values. With higher credits and/or MPS, more bytes are sent during a connection event (lower round trip time), but the average power consumption does not change significantly. The spread amounts at most 0.16 mW, and is on average very close to each other, between 0.01 and 0.07 mW. Overall, the similarity of the values is as expected, and the differences are comparable low and therefore negligible.

Between the credit cases, 2 and 10, there are no significant differences in any testcase, neither at the round trip time, nor at the average power consumption in this scenario. But at the average round trip time there is one testcase that strikes: the one with MPS 23 bytes. Since 23 MPS is the original value, the only difference compared to the original configuration at AOSP 7 is the changed credit value, 2 or 10, instead of 1. As expected, the RTT value of the original configuration in AOSP 7 (with credit size 1 and 23 bytes MPS, testcase with PL 512 bytes and CI 125 ms on the right side in Figure 6.4) is about 15 times higher than the RTT value with the same MPS but with a higher amount of

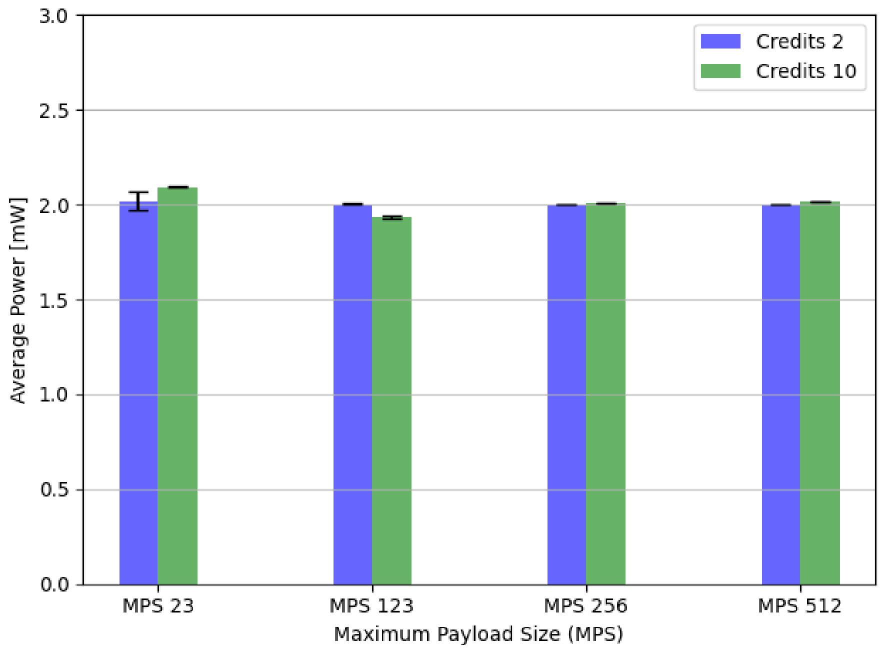


Figure 6.5: Average power ponsumption of the node device connected to the router, with different MPS and credit values. Results are measured in milliwatt.

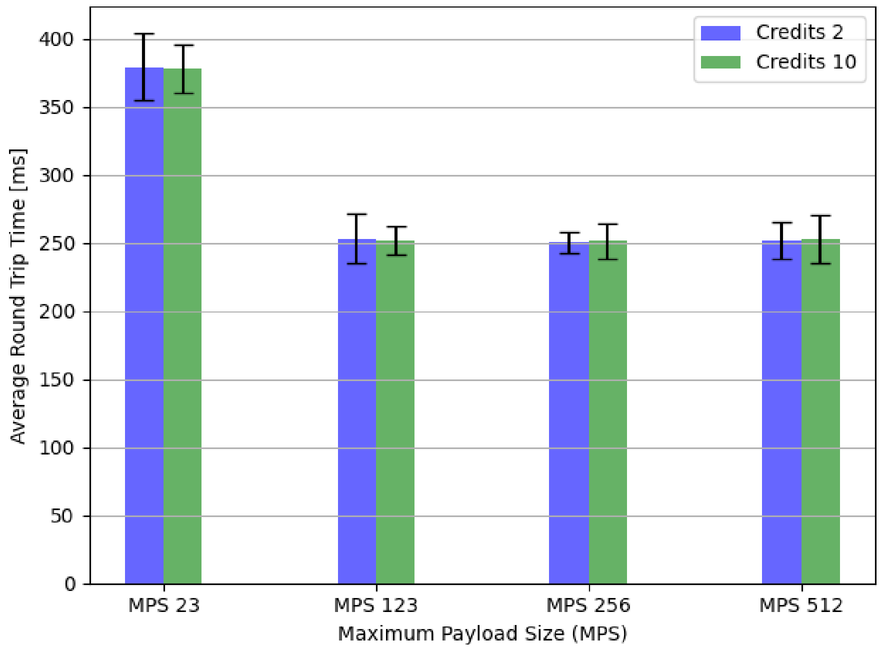


Figure 6.6: Round trip time of the UDP messages sent by the node device to the router, with different MPS and credit values. Results are measured as milliseconds.

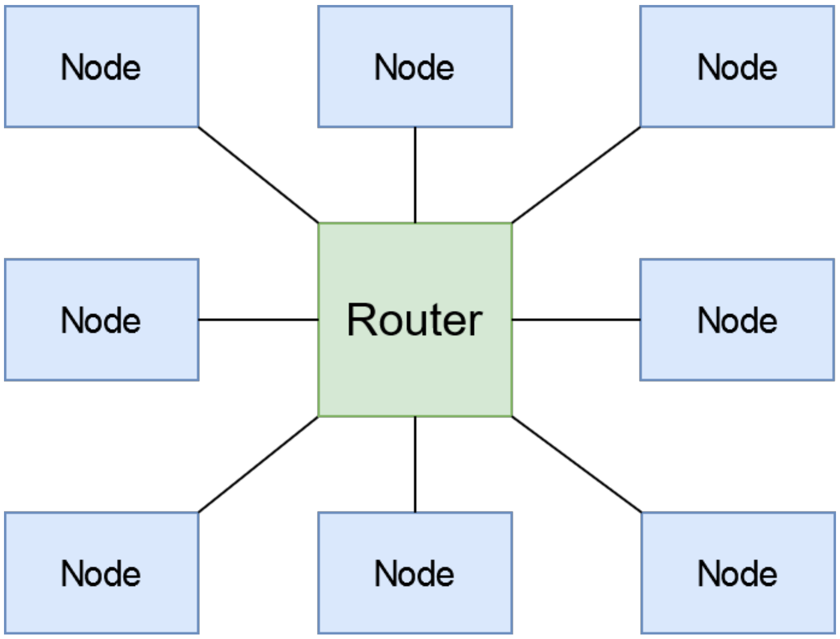


Figure 6.7: Network topology for parallel nodes usage measurements.

credits. So the more effective parameter change is the increasing of the credit size. The increasing of the maximum payload size also decreases the round trip time, but it is not that effective in this setting. With the higher MPS of 123 instead of 23 bytes, the RTT decreases about a third, but further increases are not noticeable at the round trip time values. This can be attributed to the long connection interval of 125 ms and the packet length of 512 bytes. A longer packet length, or a shorter connection interval would make a difference in the resulting RTT values with higher maximum payload sizes.

6.3 Scalability

This section describes the simultaneous connection of up to eight nodes to the Android device used as IPv6-over-BLE router. It shows the limitations of the Bluetooth radio and the routing implementation, and the possible disadvantages.

6.3.1 Experimental Target

The target of this test session is to prove that connecting several nodes simultaneously is possible, and to investigate how many devices can be connected. Additionally the workload of the Bluetooth radio and the application, the limitations of the Android device resources, as well as the effect on the BLE connections themselves are analysed.

6.3.2 Experimental Setup

Again, the LG Nexus 5X with OS Android 8 is used. The IPv6-over-BLE application is running on it, and the own version of the UDP echo server program. The node devices consist of the Nordic Semiconductors nRF52840 DK with the UDP echo client program on each of them. At each node device, the PRR is measured. To receive a baseline value, the first tests were run with only one connected node. After that, further tests were run with two, four, and eight nodes. For each node number, the runs were made three times.

6.3.3 Result and Analysis

The results of this test session can be seen in Table 6.1 and Figure 6.8. The connection with only one node device can be seen as baseline value for a better comparison with a higher number of node devices.

The main limitation for parallel connected devices comes from the Android Bluetooth Stack Bluedroid. Because of the latter, only seven concurrent active synchronous connections are possible. If the application tries to establish a BLE connection with an eight device, the most recent connection establishment does not succeed.

	1	2	4	8 (7)
PRR	100%	100%	100%	100%
Memory	61.6 - 62.4 MB	62.6 - 66.1 MB	63.4 - 65.3 MB	62.8 - 66.2 MB
Energy	15 - 25%	15 - 30%	30%	30 - 45%

Table 6.1: Results of the usage of several nodes in parallel connected to the same router inside one BLE subnet. Memory and energy were measured on the Android smartphone.

In relation to the round trip time, the values are only slightly higher if more than one node is connected. But between 2 and 7 nodes the is no latency increase noticeable. The PRR stays the same for every number of devices. Even with a large number of devices, the Bluetooth reliability remains the same which is to be expected, because simultaneous BLE connections of the same device do not affect each other. The energy consumption and memory usage are measured at the smartphone. These values indicate the efficiency of the implementation, and if it contains any memory leaks. The energy usage is measured by the Energy Profiler, provided by the Android Studio, which is the IDE for Android development. This profiler does not directly measure the energy consumptions: instead, it uses a model that estimates the energy consumption for each process running on the device.

The memory consumption on the smartphone is measured by the Memory Profiler, provided by the IDE Android Studio. The memory consumption on the smartphone only slightly increases with the increasing number of devices. The higher number of active BLE connections, and therefore the bigger routing table, is not that memory consuming.

The measured energy consumption belongs completely to the IPv6-over-BLE application, since no other processes are running, neither in the foreground, nor in the background. The increase in energy consumption can be attributed to the higher performance of the BLE radio. When communicating with more than one device at once, the active time of the bluetooth radio increases accordingly. Since the memory usage is only slightly higher, the power consumption of the memory and the computational power consumption are not the main reason for the increase.

The low energy and memory consumption do not influence the resource availability of

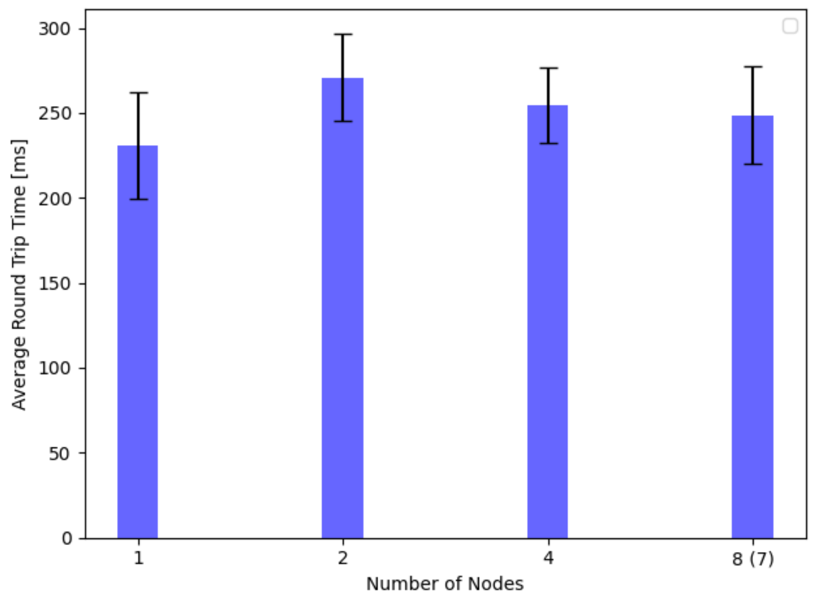


Figure 6.8: Round trip time of the UDP messages sent by the node devices to the router. At more than one device in parallel, the mean of all nodes is calculated together. Android only supports seven Bluetooth connections in parallel, connecting an eight device fails (shown as 8(7)). Results are measured as milliseconds.

the entire device so strongly that other applications on the Android device are affected, even with the maximum possible number of connected devices.

Summarising, the maximum number of parallel connected devices does not compromise the application, the Android OS, nor the IPv6-over-BLE connections.

6.4 Subnet Routing

This section shows the ability of routing packets inside the IPv6-over-BLE subnet, which includes the ability of being connected to at least two node devices simultaneously.

6.4.1 Experimental Target

The target of this section is to check the routing functionality of the implementation. It should be possible to parse the received messages, check the target address, and to forward the message correctly. This includes the correct setup and usage of the routing table inside the Android router implementation, and the handling of several BLE connections inside the same application.

6.4.2 Experimental Setup

As router device the LG Nexus 5X smartphone with Android 8 is used. The implementation is now used to forward received packets to a destination inside the BLE subnet. The

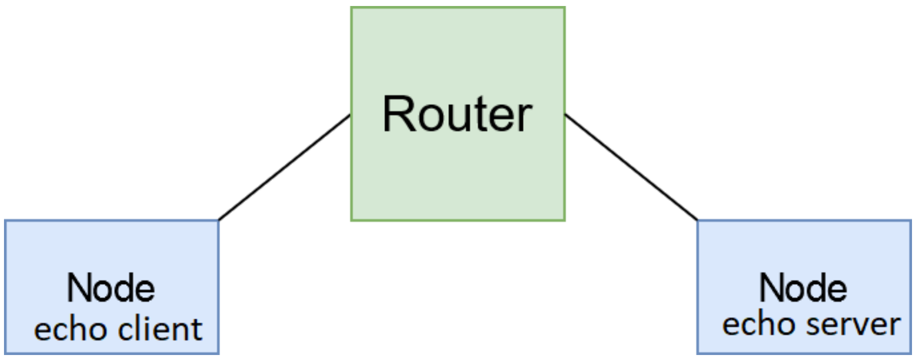


Figure 6.9: Network topology for the BLE subnet routing measurements.

node devices are the Nordic Semiconductors nRF52840 DK with the programs UDP echo client and UDP echo server.

The first node has the UDP echo client on it. It sends the UDP packets and checks the received packets if they are equal. This node is used for the RTT and PRR measurements. The target of the sent packets is the second node, to which the first node does not have a direct connection. To reach the second node, the first node has to send the packets to the router. The second node has the UDP echo server applicatin on it. It receives the packets sent by the first node and forwarded by the router, and sends them back on the same way. On this node also the PRR is measured, because this rate has to get measured for every BLE connection. As payload length and connection interval 512 bytes and 45 ms are used, respectively. These values show that the routing works also with packet fragmentation and a higher number of used connection intervals. The whole experiment is executed three times.

6.4.3 Result and Analysis

	Packet Reception Rate	Round Trip Time Average	RTT Stand. Deviation	Memory Usage	Energy Consumption
Android8 Nexus 5X	100%	4,958.59 ms	106.76 ms	62.5 - 64.3 MB of overall smartphone memory usage	15 - 25% of overall smartphone energy consumption

Table 6.2: Results of the Subnet Routing Test with RTT and PRR from the node device, memory usage and energy consumption from the router device.

Table 6.2 shows all measurements. The round trip time and the packet reception rate are measured at the node devices, the memory usage and the energy consumption are measured at the smartphone. The RTT results from the double way of the packet. Since the nodes are not connected to each other, the communication runs over the router. The gained RTT values confirm this route length of the packets.

The overall PRR results from the own PRR of each node-router connection. Therefore, the overall PRR is 100%, because each sub-connection also does not have any packet loss. The memory usage stays constantly between 62.5 and 64.3 MB. In all tests, the implementation does not show any memory leak, nor needs so much memory that other applications running on the smartphone would be affected. The energy consumption at the smartphone stays for the routing application between 15 and 25%. This is the same as when two nodes are connected and the smartphone does not need to route packets between the connected devices. So the routing application does not consume more energy so that it would appear in the energy measurement of the Energy Profiler. This energy consumption value contains the data processing and the Bluetooth low energy connections. Of course, in the connection setup, the Bluetooth radio consumes more energy than in the connection state, which is a reason for the variation of 10%.

6.5 Long-Time Experiment

Some program behaviours may change after several hours at runtime, due to smartphone energy management, process handling, and Android application states. This experiment should check the application for possible errors that occur after several hours of runtime.

6.5.1 Experimental Target

The very long runtime generates different conditions at which the application runs. Prioritisation change of the application, timeout of the screen, or state change of the application service are possible conditions under which the application must behave constantly.

6.5.2 Experimental Setup

The setup used for this test is the same as in Section 6.1 and shown in Figure 6.1. It contains the node device Nordic Semiconductors nRF52840 DK with the UDP echo client application on it. As router device the LG Nexus 5X smartphone with Android 8 is used, with the the UDP echo server application on it. The packet reception rate and the round trip time are measured at the node device. The energy consumption and the memory usage are measured at the smartphone. The duration of the test is over 20 hours.

6.5.3 Result and Analysis

The results of this test are shown in Table 6.3 and 6.4. The memory usage has a variation of 3.4 MB, which implies that no unexpected behaviour occurs. Also the energy consumption stays between 10 and 25%, which is the normal range of the application connected to one IPv6-over-BLE node. Because of the memory and energy consumption, the application does not entail weak spots of the implementation, which would affect the performance or the battery time of the smartphone. The packet reception rate of 100% proves the reliability of the application and the BLE connection. So neither the standby state of the smartphone, the movement of the application to the background, or the parallel usage of other applications affect the IPv6-over-BLE application, or the BLE connection itself.

	Duration	Memory	Energy	PRR
Results	21 h	61.8 - 65.2 MB	10 - 25%	100%

Table 6.3: Results of the long time test of one node connected to the router with UDP echo client and UDP echo server programs running on them.

	AVG RTT/PRR	90%	95%	99%	MAX
RTT	235.19 ms	225.07 ms	281.18 ms	449,93 ms	449.98 ms
PRR	100%	100%	100%	100%	100%

Table 6.4: Analysis of the outliers of the measurement of the usage of one node connected to the router and sending UDP messages over a long time.

The outliers of this long time test are shown in Table 6.4. The unusual values account less than 10% of the whole measurement results. The outliers can be affected by possible small interferences, since the environment is not protected.

6.6 Parallel GATT Sample Usage

This section describes the parallel usage of a GATT-based application, while the IPv6-over-BLE application runs in the background. It compares the usage of the same connection setup with and without the parallel GATT application running.

6.6.1 Experimental Target

The target of this test is to check the possible interference of the use of the original BLE stack with the GATT layer, and the use of the new BLE stack with the IPv6 and 6LoWPAN layers. The IPv6-over-BLE connection should not have more restraints with the GATT device connected than if an additional IPv6-over-BLE device gets connected.

6.6.2 Experimental Setup

The setup for this test contains the Nordic Semiconductors nRF52840 DK with the UDP echo client program as the IPv6-over-BLE node device. The LG Nexus 5X smartphone with OS Android 8 version with the IPv6-over-BLE application acts as the router. For this experiment the Android 8 version was used because the application running on this OS is the easiest version for consumers to use. The packet length of the UDP data is 512 bytes, and the connection interval is 125 ms. Additionally, another Nordic Semiconductors nRF52840 DK with a GATT based heart rate monitor sample acts as the second node. After the connection setup and packet sending of the UDP echo programs, the IPv6-over-BLE application gets moved to the background, and a second BLE application gets open. This one connects to the heart rate monitor node and starts receiving the heart rate data. The packet reception rate and the round trip time are measured at the UDP echo client node. The successful connection setup and the successful receiving of the heart rate data are measured at the heart rate monitor application on the smartphone. The entire test is repeated three times.

6.6.3 Result and Analysis

Table 6.5 shows the base values without the GATT based application, and the test value with the GATT based application. It can be seen that the packet reception rate amounts 100% in both cases. The round trip time increases after the connection setup of the heart rate monitor device. But this increment does not belong to the usage of the GATT layer: it belongs to the additional BLE connection.

	AVG RTT	PRR
without GATT	6289,29 ms	100%
with GATT	6374,50 ms	100%

Table 6.5: Results of the average RTT and PRR with and without the usage of a GATT based heart rate monitor in parallel to the UDP echo server program.

The UDP echo client program records no packet loss nor additional latency because of the use of a GATT based application, compared to the same amount of connected IPv6-over-BLE devices. This concludes that the usage of another Bluetooth stack in parallel is not problematic for the device or the Bluetooth connections.

6.7 Internet Connectivity

This test shows the ability of the application to act as a border router and route IPv6 messages to and from the Internet.

6.7.1 Experimental Target

The target of this experiment is to test if the application can forward IPv6 messages from the BLE subnet to the Internet and vice versa. This means the application converts the router into a border router, and communication outside the BLE subnet is possible.

6.7.2 Experimental Setup

The IPv6-over-BLE application forwards a ping-request message from a node inside the BLE subnet to a server at the Internet, and forwards the response from the server back to the node. For this test, the Google server with the IPv6 address 2001:4860:4860::8888, and IPv4 address 8.8.8.8 is used.

For using LTE to send and receive IPv6 messages, the settings at the LTE access of the smartphone have to be changed. Also the mobile radio provider must offer this functionality. For using WiFi to send and receive IPv6 messages, the ISP has to offer that functionality, and the WiFi router must be able and adjusted accordingly to allow IPv6.

For this experiment, WiFi instead of LTE was used, due to additional effort for enabling a suitable provider and enabling the IPv6 connection through LTE. For the sake of simplicity, and since the focus of this thesis is not on the Internet connection, the routing takes place with WiFi, and with the conversion of the IPv6 address to IPv4. The address gets converted by requesting the DNS entry of the IPv6 address, and requesting the according IPv4 address of the resulted DNS entry. To support the general use of IPv6 addresses without a supporting provider, the use of an IPv6 tunnel is a good alternative. But to keep this experiment simple, a target address with an IPv4 address at the same domain name is used.

As node device the Nordic Semiconductor nRF52840 DK is used. As border router device the Nexus 5X with Android 8 is used. The connection to the Internet is established through WiFi. The node sends an ICMPv6 echo request message to the Google server. This message is sent via IPv6 over BLE to the border router, and gets forwarded via WiFi to the Google server. The ICMPv6 echo response from the server, received by the smartphone, gets forwarded via BLE to the node device. The only important measurement is the reception of the ICMPv6 echo response from the server at the node device.

6.7.3 Result and Analysis

This test measures only the successful routing of the IPv6 messages. The node receives the successful ping with IP address and used time.

	AVG RTT	Standard Deviation	PRR
Google Server	401,97 ms	32,25 ms	100%

Table 6.6: Results of pinging a server through the border router.

The connection of the node device to the server can be split into the Bluetooth Low Energy connection part from the node to the border router, and the Internet connection part from the border router to the server. The latency at the Internet connection varies much stronger, compared to the variation of the latency at the Bluetooth Low Energy connection. But the results prove that the connection was successful, and the routing of the IPv6-over-BLE application works.

Chapter 7

Conclusion

IPv6 over BLE provides direct Internet connectivity for smart constrained devices. Unfortunately, although smartphones have BLE functionality, no smartphone provides functionality of an IPv6-over-BLE border router. This thesis contributes the design and a functional solution for the IPv6 communication over BLE connections, using Android devices as border routers. It discusses the advantages of this new functionality, compares IPv6-over-BLE usage of various devices with each other, and compares the implementation with an existing solution.

The primary contribution of this thesis is the design and the implementation of an IPv6-over-BLE communication stack that is interoperable with the existing Android BLE stack, and that works with the operating systems Android Open Source Project version 7 and Android 8.

The experiments carried out in this thesis show the interoperability with existing solutions, compare the implementation running on different devices and different Android versions, and show the behaviour in different use cases like subnet routing, up to eight devices connected in parallel, or the border router functionality for Internet connectivity.

The first two experiments prove the interoperability with RFC 7668 compliant node devices. They also prove that the application is runnable at two different Android OS versions and on different devices. With the enhanced connection parameters in AOSP, the application can even be more efficient than the existing border router implementation on the Raspberry Pi with Linux. Further experiments show that several nodes can be connected simultaneously without relevant performance loss, even with the maximum number of multiple BLE connections on Android. Additional experiments prove the routing capabilities with subnet routing, as well as border routing capabilities with the Internet test. Finally, the long-time experiment as well as the experiment of using a GATT-based application in parallel to the IPv6-over-BLE application, show the robustness of the implementation in Android.

Summarizing, the designed, implemented, and tested application in this thesis is competitive with the current available IPv6-over-BLE solutions.

Chapter 8

Future Work

8.1 Node functionality

For this thesis only the router functionality of IPv6 over BLE has been implemented in Android. However, it is also possible to use the smartphone or tablet with the Android OS as node device. The data of the integrated sensors (e.g., accelerometer, gyroscope, light sensor, compass, barometer) of the device can be used and sent to a central router device, or also another hardware can be connected to the device which sends then the data to another router. The data can be sent periodically or in blocks if the Internet connection is interrupted.

The node functionality, and the functionality of connecting the device as router to another router, are standardized in the RFC 7668 standard [6]. For this functionality an Android application with a background service is necessary. This application needs permissions for accessing sensors and Internet. The IPv6-over-BLE connection functionality created during this thesis can be reused, with adaptation of the L2CAP CoC usage. Also the Internet connection, WiFi or LTE, must support IPv6, or an IPv6 tunnel is used.

8.2 User Interface

The user interface of the Android application is simply and offers only the minimum functionality and status information. There is a lot potential of extending the user interface to increase the user experience and to ease the usage. The status of the connection setup can be shown in detailed steps, such as router discovery and neighbour discovery. The status informations of the existing connections can be shown at an own info-screen for each connection.

Additionally, several devices could get selected and connected simultaneously, instead of selecting and connecting only one device after another. It can show, for example, the transfer rate, transferred data, and time of establishment. The available but not connected devices can either be shown in a list, or in a map which illustrates the approximated distance and direction of the available devices. User-defined filtering for searching for devices, especially when a lot of devices are available, would also be useful. Also the signal strength can be a parameter of the available devices. Existing Android applications regarding Bluetooth devices already have such maps of available devices implemented.

8.3 Android versions

The implementation provided by this thesis works for the Android open source project version 7 and the official system image of Android 8. Based on the spread and the ease of use, the implementation should primarily work for official system images of the Android OS. The current newer Android versions are 9 and 10.

To support newer Android versions, the most likely necessary adjustment for the implementation is the setup for establishing an L2CAP connection-oriented channel in credit-based flow control mode, because of interface changes in Android. The remaining part of this application can be reused with newer Android operating systems due to backward compatibility.

8.4 Further Operating Systems

The mobile OS Android is the most used one on the market. But it would make sense to implement the solution also for other mobile operating systems. The second most mobile OS is iOS from Apple, and it also does not contain an implementation for using IPv6 over BLE yet.

To add IPv6-over-BLE support to other operating systems, such as iOS, supporting L2CAP in credit-based flow control mode is critical. Reuse of the existing Android application for iOS is not possible, since including Java files in Swift (programming language for iOS applications) is not supported. However, the user interface and functionality of the iOS application for IPv6 over BLE can be based on the user interface of the existing Android application.

Bibliography

[1] Xia, Yang, Wang, Vinel, “Internet of Things,” *International Journal of Communication Systems*, 2012.

[2] C. C. Kevin Townsend, “Getting Started with Bluetooth Low Energy,” *O’Reilly Media Inc*, 2014.

[3] N. Gupta, *Inside Bluetooth Low Energy.* Artech house, 2013.

[4] “Nuki Smart Lock.” `https://nuki.io/`. Accessed: 2020-06-17.

[5] “Shutter Box.” Accessed: 2020-02-05.

[6] J. Nieminen, T. Savolainen, M. Isomaki, Nokia, B. Patil, AT and T, Z. Shelby, Arm, and C. Gomez, “RFC 7668 IPv6 over Bluetooth Low Energy,” *Internet Engineering Task Force (IETF)*, October 2015. `https://tools.ietf.org/html/rfc7668`.

[7] “Zephyr Project.” `https://docs.zephyrproject.org/`. Accessed: 2020-01-26.

[8] M. Spörk, “IPv6 over Bluetooth Low Energy using Contiki,” Master’s thesis, Graz University of Technology, 2016.

[9] “SIG Bluetooth. Radio Versions.” `https://www.bluetooth.com/bluetooth-technology/radio-versions`. Accessed: 2020-02-19.

[10] M. Afaneh, “Bluetooth 5 and BLE: Achieving maximum throughput and speed..” `https://www.novelbits.io/bluetooth-5-speed-maximum-throughput/`, September 2017. Accessed: 2020-01-10.

[11] “Connection parameters - Introduction to MBED BLE.” `https://os.mbed.com/docs/mbed-os/v6.2/apis/bluetooth.html`. Accessed: 2020-01-17.

[12] Carles Gomez, Joaquim Oller, Josep Paradells, “Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology,” *PubMed Central*, 2012.

[13] “Link Layer - How Bluetooth Low Energy works.” `https://medium.com/@zpcat/how-bluetooth-le-works-link-layer-b18475250259`. Accessed: 2020-06-10.

[14] “BLE5-Stack User’s Guide - Host Controller Interface (HCI).” `http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_35_00_33/docs/ble5stack/ble_user_guide/html/ble-stack/hci.html`. Accessed: 2020-03-05.

[15] "LE L2CAP Connection Oriented Channel at Android Open Source Project." `https://android.googlesource.com/platform/system/bt/+/6721232`. Accessed: 2020-03-26.

[16] "SIG Bluetooth. Bluetooth 5 Core Specification." `https://www.bluetooth.com/specifications/bluetooth-core-specification/`. Accessed: 2019-11-15.

[17] "Bluetooth Core Specification Version 5.1 Feature Overview." `https://www.bluetooth.com/bluetooth-resources/bluetooth-core-specification-v5-1-feature-overview/`. Accessed: 2020-08-26.

[18] "Bluetooth Core Specification Version 5.1 Feature Overview." `https://www.bluetooth.com/wp-content/uploads/2020/01/Bluetooth_5.2_Feature_Overview.pdf`. Accessed: 2020-08-26.

[19] G. Montenegro, Microsoft Corporation, N. Kushalnagar, Intel Corp, J. Hui, D. Culler, Arch Rock Corp, "RFC 4944 Transmission of IPv6 Packets over IEEE 802.15.4 Networks," *Internet Engineering Task Force (IETF)*, September 2007. `https://tools.ietf.org/html/rfc4944`.

[20] Ed. J. Hui, Arch Rock Corporation, P. Thubert, and Cisco, "RFC 6282 - Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," *Internet Engineering Task Force (IETF)*, September 2011. `https://tools.ietf.org/html/rfc6282`.

[21] Z. Shelby, Sensinode, S. Chakrabarti, Ericsson, E. Nordmark, Cisco Systems, C. Bormann, Universitaet Bremen TZI, "RFC 6775 - Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)," *Internet Engineering Task Force (IETF)*, November 2012. `https://tools.ietf.org/html/rfc6775`.

[22] "SIG Bluetooth. Mesh Model Specification." `https://www.bluetooth.com/specifications/mesh-specifications`. Accessed: 2019-11-16.

[23] Jonas Olsson, "6LoWPAN demystified," *Texas Instruments*, October 2014.

[24] "Internet Protocol version 6 (IPv6) Header." `https://tutorialspoint.dev/computer-science/computer-network-tutorials/computer-network-internet-protocol-version-6-ipv6-header`. Accessed: 2020-08-07.

[25] Gustavo Litovsky, "A look into Bluetooth v4.2 for Low Energy Products," *EDN Network*, May 2015.

[26] "Mobile operating systems' market share worldwide from January 2012 to December 2019." `https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/`. Accessed: 2020-03-10.

[27] "Android Open Source Project." `https://source.android.com/`. Accessed: 2019-10-10.

[28] "Google Store - Nexus 5X." `https://store.google.com/product/nexus_5x`. Accessed: 2019-09-21.

[29] "Qualcomm Snapdragon 808 Processor." `https://www.qualcomm.com/products/snapdragon-processors-808`. Accessed: 2019-12-09.

[30] "Qualcomm QCA6174a Wi-Fi/Bluetooth SoC." `https://www.qualcomm.com/products/qca6174a`. Accessed: 2019-12-09.

[31] "Samsung Electronics - Galaxy Tab S3." `https://www.samsung.com/at/tablets/galaxy-tab-s3-9-7-t825/SM-T825NZKAATO/`. Accessed: 2019-09-21.

[32] "Qualcomm Snapdragon 820 Mobile Platform." `https://www.qualcomm.com/products/snapdragon-820-mobile-platform`. Accessed: 2019-12-09.

[33] "Nordic Semiconductor - nRF52840 Bluetooth 5.2 SoC." `https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840`. Accessed: 2019-11-16.

[34] "Raspberry Pi Documentation." `https://www.raspberrypi.org/documentation/`. Accessed: 2020-02-06.

[35] "Contiki - The Open Source Operating System for the Internet of Things." `http://www.contiki-os.org/`. Accessed: 2020-05-28.

[36] "Android Things - Android Developers." `https://developer.android.com/things`. Accessed: 2020-05-28.

[37] "Linux Kernel Configuration Options." `https://core.docs.ubuntu.com/en/stacks/bluetooth/bluez/docs/reference/enablement/kernel-configuration-options`. Accessed: 2020-05-29.

[38] Haolin Wang, Minjun Xi, Jia Liu, Canfeng Chen, "Transmitting IPv6 packets over Bluetooth low energy based on BlueZ," *Nokia Research Center*, 2013.

[39] Wondeuk Yoon, Kiwoong Kwon, Minkeun Ha, and Daeyoung Kim, "Transfer IPv6 Packets Over Bluetooth Low Energy with Ensuring Emergency Data Transmission," *Korea Advanced Institute of Science and Technology*, 2016.

[40] Michael Spörk, Carlo Alberto Boano, Marco Zimmerling, Kay Römer, "BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices," *ACM Conference on Embedded Networked Sensor Systems*, 2017.

[41] C. Gomez, S. Darroudi, Univeritat Politecnica de Catalunya, T. Savolainen, DarkMAtter, M. Spoerk, Graz University of Technology, "IPv6 Mesh over Bluetooth Low Energy using IPSP," *6Lo Working Group*, 2019.

[42] SIG Bluetooth, "Internet Protocol Support Profile - Bluetooth Specification version 1.0.0." `https://www.bluetooth.com/specifications/bluetooth-core-specification/`, December 2014.

[43] "BlueZ - Official Linux Bluetooth protocol stack." `http://www.bluez.org/`. Accessed: 2020-04-25.

[44] "Bluetooth — Android Open Source Project." `https://source.android.com/devices/bluetooth`. Accessed: 2020-08-27.

[45] "Android Developers - Bluetooth Overview - Vendor-specific AT commands." `https://developer.android.com/guide/topics/connectivity/bluetooth`. Accessed: 2020-04-25.

[46] SIG Bluetooth, "Specification of the Bluetooth System - Covered Core Package version: 4.2." `https://www.bluetooth.com/specifications/bluetooth-core-specification/`, December 2013.

[47] SIG Bluetooth, "Specification of the Bluetooth System - Covered Core Package version: 4.1." `https://www.bluetooth.com/specifications/bluetooth-core-specification/`, December 2013.

[48] "Java Reflection." `https://www.oracle.com/technical-resources/articles/java/javareflection.html`. Accessed: 2020-08-27.

[49] SIG Bluetooth, "Logical Link Control for protocol/service multiplexers." `https://www.bluetooth.com/specifications/assigned-numbers/logical-link-control/`. Accessed: 2020-06-15.

[50] "Android Developers - Manifest Permissions." `https://developer.android.com/reference/android/Manifest.permission`. Accessed: 2020-05-18.

[51] "Monsoon Solutions." `https://www.msoon.com/`. Accessed: 2020-08-27.