Swapnil Kadam, B. Eng

IPv6 over Connection-less BLE

MASTER'S THESIS

to achieve the University Degree of

Diplom-Ingenieur

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dr. Carlo Alberto Boano

Advisor

Dipl.-Ing. BSc. Michael Spörk

Institute of Technical Informatics

Graz, April 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.

Date	Signature

Abstract

The Internet of Things (IoT) is a growing topic and a lot of intensive research is going on to provide energy-efficient wireless communication. Bluetooth Low Energy (BLE) is a prominent candidate among all low-power wireless communication protocols to create IoT applications, as it promises significant advantages of high energy-efficiency and high reliability. Although the RFC 7668 standard allows BLE devices to exchange IPv6 packets over connection-based BLE communication, BLE devices need to maintain a BLE connection by exchanging a number of keep-alive packets. Continuously exchanging such keep-alive messages may not be suitable for low duty-cycle power-constrained devices, as it may drain their batteries. Therefore, we propose IPv6 over connection-less BLE to make low-duty cycle devices more energy efficient.

The main contribution of this thesis is the design of an IPv6 over connection-less BLE communication stack suitable for the architecture of the Zephyr OS (an operating system for resource-constrained IoT devices) and its hardware-independent implementation. Moreover, the evaluation performed in this thesis shows that the connection-based communication provides 100 % reliability, while 96% of reliability can be achieved with connection-less BLE while reducing the energy consumption up to 65 % when transmitting 100 bytes of packet every 5 minutes. Moreover, the evaluation also confirms that IPv6 over connection-less BLE requires almost 30 % less packet transmission time and 20 % less energy compared to terminating and re-establishing a BLE connection when needed.

Kurzfassung

Das Internet der Dinge (Internet of Things, IoT) ist ein wachsendes Forschungsthema, in dem es unter anderem wichtig is energieeffiziente drahtlose Kommunikation zu gewährleisten. Bluetooth Low Energy (BLE) ist ein herausragender Kandidat unter allen drahtlosen Kommunikationsprotokollen mit geringem Stromverbrauch für die Erstellung von IoT-Anwendungen, da es erhebliche Vorteile einer hohen Energieeffizienz und einer hohen Zuverlässigkeit verspricht. Obwohl der RFC 7668-Standard es BLE-Geräten ermöglicht, IPv6-Pakete über verbindungsbasierte BLE-Kommunikation auszutauschen, müssen BLE-Geräte eine BLE-Verbindung aufrechterhalten, indem sie eine Reihe von Keep-Alive-Paketen austauschen. Das kontinuierliche Austauschen solcher Keep-Alive-Nachrichten ist möglicherweise nicht für Geräte mit niedrigem Leistungszyklus geeignet, da dadurch die Batterien schnell entladen werden können. Daher schlagen wir IPv6 über verbindungsloses BLE vor, um Geräte mit niedrigem Arbeitszyklus energieeffizienter zu machen.

Der Hauptbeitrag dieser Arbeit ist der Entwurf eines IPv6 über einen verbindungslosen BLE-Kommunikationsstapel, der für die Architektur des Zephyr-Betriebssystems (ein Betriebssystems für ressourcenbeschränkte IoT-Geräte) und dessen hardwareunabhängige Implementierung geeignet ist. Darüber hinaus zeigt die in dieser Arbeit durchgeführte Evaluierung, dass die verbindungsbasierte Kommunikation eine 100 % ige Zuverlässigkeit bietet, während 96 % der Zuverlässigkeit mit einer verbindungslosen BLE erreicht werden kann. Gleichzeitig wird der Energieverbrauch um bis zu 65 % reduziert, wenn alle 5 Minuten 100 Byte Paket übertragen werden. Darüber hinaus bestätigt die Auswertung auch, dass IPv6 über verbindungslose BLE fast 30 % weniger Paketübertragungszeit und 20 % weniger Energie erfordert als das Beenden und Wiederherstellen einer BLE-Verbindung bei Bedarf.

Acknowledgements

First of all, I would like to express my deepest thanks and gratitude to my supervisor **Ass. Prof. Boano Carlo Alberto**, *Institute of Technical Informatics, Graz University of Technology* for suggesting the topic of this thesis and his kind support.

It is great pleasure to acknowledge my deepest thanks and express my extreme sincere appreciation to my advisor **Dipl.-Ing. Michael Spörk** *Institute of Technical Informatics, Graz University of Technology* for his constant guidance and kind endless help, without which this work would not have been possible.

Last of all, I would like to thank my family and friends for their unceasing encouragement and support.

Contents

List of Figures

Abbreviations

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks

Adv Advertising

ATT Attribute Protocol

BLE Bluetooth Low Energy

CB Connection-based

CL Connection-less

Ext Extended

GAP Generic Access Profile

GATT Generic Attribute Profile

HCI Host Controller Interface

IETF Internet Engineering Task Force

IoT Internet of Things

IP Internet Protocol

IPHC IP Header Compression

IPSS Internet Protocol Support Service

List of Figures

IPv6 Internet Protocol version 6

ISM Industrial, Scientific and Medical

L2CAP Logical Link Control and Adaptation Protocol

Leg Legacy

LL Link Layer

MTU Maximum Transmission Unit

NA Neighbor Advertisement

NS Neighbor Solicitation

RS Router Solicitation

RFC RFC Request for Comments

SM Security Manager

1 Introduction

The Internet of Things (IoT) is helping the current society to become a smarter one by allowing everyday objects to communicate with each other. The advent of emerging low-power wireless technologies such as IEEE 802.11b (WiFi), ANT+, ZigBee, and Bluetooth Low Energy - all using the 2.4 GHz industrial, scientific and medical (ISM) radio band increased the development of the IoT applications even further [**1**]. Although most of the devices using these low-power wireless technologies are very constrained in resources, some applications require these devices to run for several years on constrained energy sources like coin cell batteries [**16**].

To connect these resource-constrained embedded devices to the Internet, in the last few years, the Internet Engineering Task Force (IETF) group standardized the exchange of Internet Protocol version 6 (IPv6) packets over low-power wireless link layer technologies like IEEE 802.15.4 [**24**] and BLE [**18**]. These standards use the Low-Power Wireless Personal Area Networks (6LoWPAN) adaptation layer in its network stack to exchange IPv6 packets.

However, previous studies have proven that, among several wireless competing technologies using the 2.4 GHz ISM band, Bluetooth Low Energy (BLE) is the most energy-efficient short-range wireless communication protocol [**1**]. Furthermore, BLE also offers several advantages as its hardware is already widely adopted in plenty of electronic consumer devices [**16**] such as mobile phones, tablets, and computers, and therefore provides easy connectivity among BLE devices and allows direct interaction with users. The new standard of BLE named BLE 5 introduces some interesting new features which could be used to make IPv6 devices even more energy-efficient and to extend the range of IPv6 over BLE networks by coding a payload at 500 Kbps or 125 kbps.

1.1 Problem Statement

BLE has two communication modes: connection-based BLE, where two devices use a BLE connection to bi-directionally exchange data, and connectionless BLE, where broadcasts are used to transmit data uni-directionally. As stated in the RFC 7668 - IPv6 over Bluetooth Low Energy standard, IPv6 over BLE devices use connection-based communication, where they need to establish a BLE connection to exchange IPv6 packets [**18**]. A BLE connection, however, needs to be maintained and therefore requires both BLE devices to periodically exchange keep-alive packets [**6**]. For very low-duty cycle applications, these mandatory keep-alive messages, however, may lead to unnecessary energy consumption that is needed to maintain the BLE connection.

Consider, for example, an application of a temperature sensing device that monitors room temperature. In this scenario, the temperature sensing device needs to transmit 100 bytes of IPv6 data periodically every 5 mins to other BLE router devices that sends the temperature data to the Internet. This application scenario can be achieved by establishing a BLE connection between two devices to send IPv6 data. However, according to BLE specification, the BLE devices need to exchange data or keep-alive packet at least every 32 sec in order to maintain the BLE connection. If the keep-alive packet is not exchanged within this timeout, the connection is considered to be lost.

Even if we use the most power-efficient configuration of a BLE connection (e.g., at least one keep-alive packet is exchanged within 16 sec [**6**]) on a BLE device, the device turns on the radio for around 38 times within 5 mins of application interval to maintain a BLE connection. The length of each keep-alive packet is 10 bytes (1 byte of Preamble, 4 bytes of Access address, 2 bytes of header, and 3 bytes of CRC). In order to send a message of 100 bytes application data every 5 mins, the device exchanges 38 keep-alive packets (10 bytes/packet) in a total of 380 bytes just to maintain a BLE connection. Therefore, for very low duty-cycle power-constrained devices, connection-based communication may not be power efficient, as it incurs an overhead of maintaining the BLE connection. As per the discussed example, the benefit of using connection-less over connection-based BLE can be up

to 70%, which motivates the investigation of using connection-less BLE to transmit IPv6 data.

However, currently, IPv6 is built solely on top of connection-based BLE, where the fragmentation and reassembly of an IPv6 packet are handled at the L2CAP layer of BLE. Since the L2CAP layer is only available in connection-based communication (see Section 4.2), the fragmentation and reassembly of an IPv6 packet need to be handled in the design of IPv6 over connection-less BLE. The other challenge is to transmit 1280 bytes of IPv6 packet over connection-less BLE. That's because, even though legacy advertising (advertising feature from prior BLE standards 4.0, 4.1, 4.2 and also exist in BLE 5, where advertiser broadcasts packets to its nearby devices and scanner on the other end scans for those packets.) with connection-less mode doesn't incur the overhead of establishing and maintaining a connection, it is not suitable for larger data packets as the maximum data it can broadcast in an Adv. packet is limited to 31 bytes.

Moreover, to transmit an IPv6 packet over BLE, devices need to configure the network interface by exchanging all the necessary information (network prefix, context information and Internet parameters (Hop limit and Link MTU)). In IPv6 over connection-based BLE communication, it is performed after establishing a link-layer and L2CAP connection by neighbor discovery procedure. However, in connection-less BLE communication, link-layer and L2CAP connections are not established. Therefore, it is necessary to use connection-based BLE to get the information required to configure a network interface. Hence the switching between connection-based and connection-less BLE needs to be handled in the design of IPv6 over connection-less BLE.

1.2 Thesis Contribution

We, therefore, aim to implement IPv6 over connection-less BLE. The connection-less approach doesn't need to exchange keep-alive packets or establish a connection and exchange information every single time to set up the network interface, which may lead to saving a reasonable amount of energy. We propose IPv6 over connection-less BLE and perform a detailed experimental

study by implementing it into the Zephyr OS on the Nordic Semiconductor nRF52 platform and measure its reliability and power consumption on real devices. Towards this goal, we make the following contributions:

- We discuss in detail how IPv6 over connection-less BLE can be used to broadcast large IPv6 packets and show how constrained devices with low radio duty cycle can use it to reduce their overall power consumption.
- We present the design and implementation of IPv6 over connection-less BLE.
- We show how devices can dynamically switch from connection-less to connection-based BLE and vice versa at runtime, i.e. to update the IPv6 prefix and context information.
- We perform an evaluation of the energy consumption of the IPv6 over connection-less BLE communication stack implemented in Zephyr OS, and compare the energy efficiency of the IPv6 over connection-based and IPv6 over connection-less communication. Finally, we present the reliability of IPv6 over connection-less BLE and compare it with the reliability of IPv6 over connection-based communication and show that although connection-based BLE provides 100 % reliability, almost 96 % of reliability can be achieved with connection-less BLE while reducing a power consumption of a BLE radio of a node by up to 65 % when transmitting 100 bytes of IPv6 packet / 5 mins.

1.3 Outline

The remainder of the thesis is organized as follows. Chapter 2 presents BLE 5 and its newly added features. Moreover, it summarizes the advantage of using Extended Advertising (Ext. Adv.) for IPv6 over connection-less BLE. It also discusses the IPv6 over BLE communication stack and explains the Zephyr OS and hardware used in this thesis. Moreover, it focuses on existing

research studies related to this thesis and compares existing BLE communication stacks to select the suitable BLE stack for IPv6 over connection-less BLE design. Chapter 3 compares the connection-less vs connection-based BLE by performing a theoretical analysis. Chapter 4 addresses the requirement and the design challenges of IPv6 over connection-less BLE. It describes the detailed design of the IPv6 over connection-less BLE communication stack developed in this thesis and Chapter 5 explains its implementation along with the challenges faced. The performance evaluation of the implemented IPv6 over connection-less BLE communication stack is performed in Chapter 6. Chapter 7 concludes the results and gives a summary of possible improvements that can be performed to improve the performance and suggests further research directions on IPv6 over connection-less BLE.

2 Background

This chapter introduces the reader to Bluetooth Low Energy (BLE) 5 and other necessary information to understand the content of this thesis. Specifically, Section 2.1 introduces BLE 5 as a newer version of the BLE standard and its newly added key features. Besides, it also provides information about the BLE communication stack. Section 2.2 describes the IPv6 over BLE network stack. Moreover, it also explains its 6LoWPAN adaptation layer, header compression and network topologies used in the IPv6 over BLE network. Section 2.3 delivers a summary of the Zephyr real-time operating system and its network stack used for the implementation of the design presented in this thesis. Moreover, the embedded platform used for the implementation and the evaluation is presented in Section 2.4. Section 2.5 explains the existing research work that is related to this thesis and lists existing BLE stack implementations for resource-constrained devices that can be considered while implementing this thesis work.

2.1 Bluetooth Low Energy (BLE)

This section introduces Bluetooth Low Energy 5 to a reader with its history, key features and communication stack.

2.1.1 Bluetooth

These days, we are in regular contact with Bluetooth technology as it has become a part of our mobile experience [2]. It also includes wireless headphones, portable speakers, wireless keyboards, mouse, for exchanging data over the air. It all started back in 1998 with the idea of establishing

short-range radio links to exchange data among handsets without using the tangle of wires [2]. At that time, Ericsson took an initiative and encouraged other companies to form the Bluetooth Special Interest Group recognized as SIG to collaborate on the technology and roll out the specifications for the first Bluetooth standard, Bluetooth Version 1.0. It was offering around 700Kbps of throughput but in realistic conditions, it was usually notably less [2]. The following iterations were focused on increasing bandwidth, adding Adaptive Frequency Hopping (AFH) and Bluetooth found its way into cellphones, speakers, computers, printers, and many electronic devices.

2.1.1.1 BLE

Another breakthrough in Bluetooth technology, version 4.0, introduced compact wireless power-constrained devices by allowing them to run on small capacity batteries for a longer time [2]. This new low-power Bluetooth is called Bluetooth Low Energy (BLE) and also remarketed as Bluetooth Smart in 2010 [16]. Following the success of version 4.0, SIG started considering BLE for IoT applications and provided support to the IPv6 communication in the later version 4.1. The data-carrying capacity of BLE data packets was limited to 31 bytes which was then increased by almost ten times by allowing data packets to exchange up to 251 bytes of data in BLE 4.2 [2]. This version added many new features that improved the speed of data transmission and made it more reliable with new privacy features. The new version of BLE, BLE 5.0, introduced in 2016 also offered some interesting new features that expand the capabilities of BLE devices.

2.1.1.2 BLE-5 and its key features

More Physical (PHY) Modes:

BLE 5 introduces new radio physical layer (PHY) called 'LE 2M' which allows an increase in communication speed by doubling the bandwidth from 1Mbps to 2Mbps [2]. The time required to transmit and receive BLE packets also decreases since it supports the data rate of 2M symbols per second. This means the same data sent in BLE 5 will now take half the time

compared to prior standards at the price of lower link-layer reliability [4]. Another significant benefit of having a 2Mbps data rate is that the radio will turn on for a shorter period in transmitting the data, contributing an additional significant advantage of reduced power consumption which may lead to extended battery life [4]. Another notable change of BLE 5 is a significant improvement in the communication range. BLE 5 introduces a new radio PHY named 'LE Coded', that offers an extended communication range up to 400% of the prior range [2]. Even in the worst case, it should achieve a range of 200 meters outdoors and about 40 meters indoors. In BLE, for both 1 Mbps and 2 Mbps connection modes, the payloads and packet headers are un-coded. However, in LE Coded connection mode, these are coded and it supports 500 and 125 kbps data rates. The lower the data rate the longer is the range that can be achieved: with a 500 kbps data rate it is possible to obtain nearly twice the range of standard BLE, whereas, with 125 kbps four times range can be reached [4]. As these coded packets are approximately 2 to 8 times larger in size, the time required to transmit the same data may increase, which may also lead to higher power consumption [4].

Extended Broadcast Capacity:

BLE 5 provides a new communication mode, called Extended Advertising (Ext. Adv.) [4]. It allows BLE devices to broadcast up to 254 bytes of payload in Adv. packets compared to 31 bytes in prior standards. This allows an application to transmit larger data to other BLE devices without even establishing a BLE connection. In addition to increase the maximum packet length, BLE-5 proposes the use of data channels named "secondary advertising channels" for broadcasting up to 254 bytes of Adv. packets.

Extending the communication range by a factor of four, increasing the speed, reducing energy consumption, and extending the data-carrying capacity of broadcast packets, should allow wider adoption of the standard in consumer devices, including the emerging IoT [4]. As a result, it could be likely that BLE 5 eventually becomes one of the most preferable wireless standards used in many IoT applications.

2.1.2 BLE-5 Communication Stack

The BLE protocol stack consists of three major components: an application, a BLE controller and a BLE host (see Figure 2.1). An application and a BLE host are always a part of a single chip. However, they can be implemented with a controller on a single chip as well. The application (App) is the highest block of the stack, and it represents the direct interface with the user. The host implements the functionalities of the upper layers, while the controller implements the link layer (LL) and physical layer (PHY) [13]. The communication between host and controller takes place via the standardized Host Controller Interface (HCI) [6].

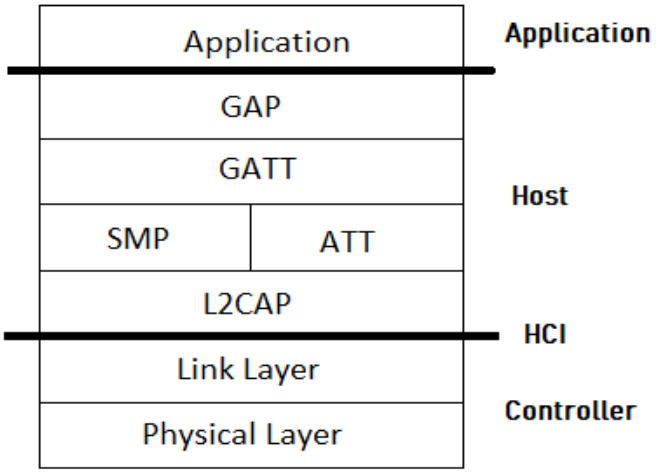

Figure 2.1: BLE communication stack (adapted from [13]).

- The host includes the following layers:
  - Generic Access Profile (GAP)
  - Generic Attribute Profile (GATT)
  - Logical Link Control and Adaptation Protocol (L2CAP)
  - Attribute Protocol (ATT)
  - Security Manager Protocol (SMP)
- The controller includes the following layers:

- Link Layer (LL)
- Physical Layer (PHY)

Physical Layer (PHY):

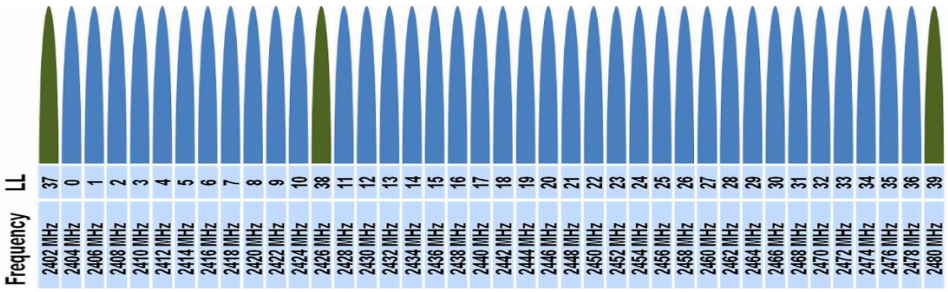

Figure 2.2: Distribution of 40 BLE channels and a location of 37 data channels (0-36) and 3 advertising channels (37-39) in the 2.4GHz ISM spectrum band [7].

BLE operates in the globally unlicensed 2.4 GHz Industrial, Scientific and Medical (ISM) band, the same one used by Wi-Fi and ZigBee. To be precise, the BLE radio band covers from 2.4 GHz to 2.48GHz. As can be seen in Figure 2.2, the band is divided into 40 BLE radio channels. Three of these channels (37, 38 and 39) are dedicated for advertising., while the remaining 37 are allotted to exchange data packets over BLE communication. To avoid external interference in the ISM radio band with other wireless communications, BLE implements an Adaptive Frequency Hopping (AFH) for its connections. Using the AFH, BLE devices that share a connection periodically change the BLE radio channel used for communication to mitigate interference and multipath fading [5]. Two variants have been added to the BLE 5 standard. The first one is LE 2M PHY which offers double the symbol rate compared to the 1 Mbps BLE standard (see Section 2.1.1.2). Both the LE 1M and LE 2M PHYs are part of LE Uncoded PHY as they do not support error correction coding stage. However, another variant supports two coding schemes S=8 and S=2 (S is the symbols per bit) which is called as LE coded PHY [4]. Moreover, BLE 5 also allows to use the channel 0-36 as secondary Adv. channels. This allows BLE devices to broadcast longer Adv. packets to other non connected BLE devices.

Link Layer (LL):

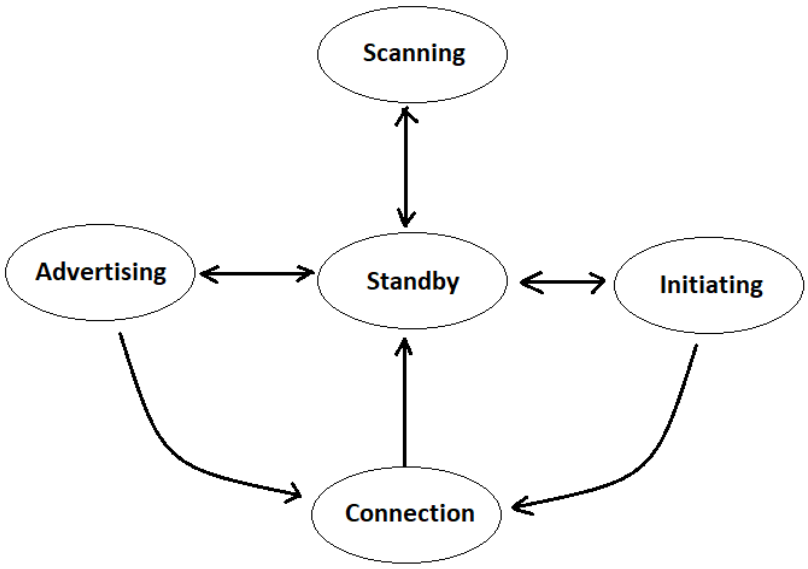

Figure 2.3: Link layer state machine (adapted from [17]).

The Link Layer (LL) is the software (SW) part that directly interfaces with the PHY. The LL schedules packet transmission and reception based on parameters provided by the host. It also ensures the transmission of data and the time synchronization between two consecutive packets from different channels. It defines various roles a device can play, i.e., advertiser, scanner, master, and slave.

The LL defines a state machine as shown in Figure 2.3:

- Standby: The device does not exchange packets with other devices [17].
- Advertising: The device acts as an advertiser and broadcasts advertising packets on Adv. channels [17].
- Scanning: The device acts as a scanner and scans for packets sent by an advertiser on Adv. channels [17].
- Initiating: The device responds to the Adv. packets and initiates the connection [17].

- Connection: The devices communicate with each other by acting one as a master and another as a slave. The Master defines the timing of transmission [17].

Host Controller Interface (HCI):

The HCI can operate over different physical communication protocols like SPI, UART or USB [6]. The communication between the host and the controller is achieved with standardized commands and events. The host sends standardized commands defined by the HCI to the controller and can expect the events in return as a response from the controller. Due to the standardization, the different BLE hosts and controllers from different vendors can also communicate with each other [17].

Logical Link Control and Adaptation Protocol (L2CAP):

The L2CAP receives the application data from upper layers and encapsulates them into the standard BLE data packet, which can be transmitted on data channels during connection. It also performs fragmentation and recombination when a large amount of data needs to be exchanged [17].

Attribute Protocol (ATT):

The ATT layer allows a device to discover and set attributes of another (remote) BLE device. These attributes can be device name, device model number, sensor values [12]. It is a peer to peer protocol between client and server. Attribute client and attribute server communicate with each other on a dedicated L2CAP channel. The server provides a set of attributes that can be read and set by the client [6]. The client and server role is independent of the master or slave role of the device.

Generic Attribute Profile (GATT):

The Generic Attribute (GATT) uses globally unique 128-bit universally unique identifier (UUID) to identify profiles, services and characteristics exposed by the connected remote device [6]. A remote device can expose, for example heart rate monitor service and its corresponding characteristics like heart rate and unit. For efficiency, shortened 16-bit UUID is also supported and there are many 16-bit UUIDs defined by the Bluetooth SIG for defined GATT profiles. GATT uses ATT as a transport mechanism to exchange the data of the services and characteristics.

Security Manager (SM):

The Security Manager deals with encryption and authentication of data packets using 128-bit AES cipher block. The privacy of the BLE devices is also improved by the security manager by providing an additional method for generating a random device address.

Generic Access Profile (GAP):

The Generic Access Profile is the highest layer in the BLE host component of the stack. It defines the procedures used for device discovery, connection setup, service discovery and security. GAP defines four device roles: Broadcaster, Observer, Peripheral, and Central [6]. The device can operate in one or more GAP roles at the same time if it is supported by the link layer of the used controller. A broadcaster periodically transmits data on Adv. channels using Adv. events. An observer scans for these Adv. packets on the Adv. channel. A central device acts as a master and initiates connections to several peripherals. The peripheral accepts a connection and act as a slave device.

2.1.3 BLE-5 Communication

The BLE 5 supports two modes of communication: connection-based and connection-less. To transfer the data over connection-based communication, devices need to establish a connection by exchanging a number of

connection-less packets before transmitting the actual data. However, maintaining a connection also incurs overhead for resource-constrained devices with a low duty cycle (e.g., sending a packet every 5 minutes as we show in Section 6.1). The connection-less mode can be used to broadcast a data unidirectionally and can also be used to establish a connection in connection-based mode.

2.1.3.1 Legacy Advertising:

In the connection-less mode we can use two different advertising specifications: Legacy Advertising and Extended Advertising. As BLE 5 is backward compatible, it also supports the Adv. packet structure and Adv. mode of the BLE 4.x, called Legacy Advertising (Leg. Adv.). The connection-less communication can be achieved over Leg. Adv. A legacy advertising PDU can advertise up to 31 bytes of payload on Adv. channels [6]. In Leg. Adv. mode, BLE devices can act either as an advertiser or a scanner.

Advertiser:

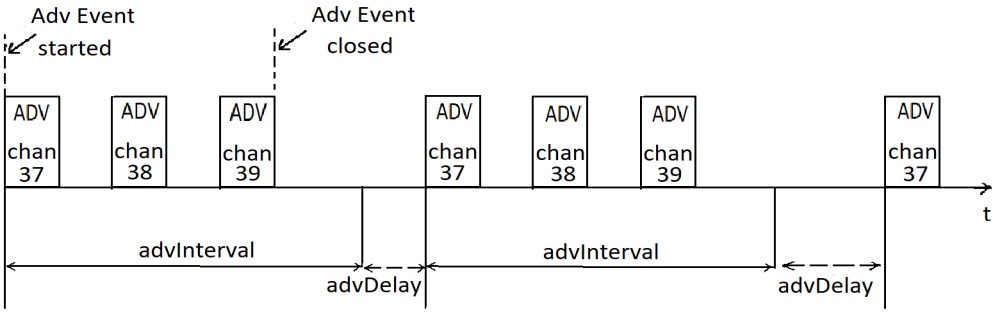

Figure 2.4: Legacy Advertising on an Advertiser (adapted from [6]).

In the advertising state, the link layer broadcasts the Adv. PDUs on one or all three Adv. channels in each Adv. event. Fewer Adv. channels can be selected in order to save power. However, advertising on more channels

increases the possibility of collecting the packet on the scanner devices. The data sent during an Adv. event on these Adv. channels carries the same information in its protocol data units (PUDs). The time between two consecutive Adv. events can be configured by the user application and is defined by the *advertising Interval (advInterval)* parameter [**6**]. The *advInterval* is in the range of 20 ms to 10 sec and can be set with an integer multiple of 0.625 ms. As seen in Figure 2.4, the actual interval between two Adv. events is extended with the *advDelay* value. It is a pseudo-random value with a range of 0 ms to 10 ms generated by the LL and added at the end of each Adv. event to avoid a collision with other BLE devices [**6**].

Scanner:

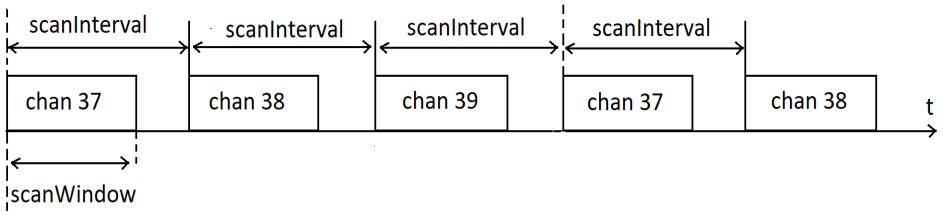

Figure 2.5: Scanning state with *scanInterval* and *scanWindow* parameters using all three Adv. channels (adapted from [**6**]).

A device needs to enter into a scanning state to receive the Adv. packets transmitted by an Advertiser. While scanning, the link layer listens on one or up to 3 Adv. channels. The duration of scanning on a single Adv. channel is defined as ”scan window”, and the interval between the start of two consecutive scan windows is set with the *scanInterval* parameter [**6**]. If the *scanWindow* and the *scanInterval* parameters are set to the same value by the host, the LL should scan continuously [**6**]. The scanning process is shown in Figure 2.5.

The scanner supports two type of scanning, defined by the host: passive and active. When in passive scanning, the LL only receives packets and is not permitted to send any packet. In active scanning, the LL listens for Adv. PDUs and, depending on the Adv. PDU type, it may request an advertiser

to send additional information in order to establish a connection by sending a scan request [6].

Legacy Advertising Packet Format:

The packet transmitted on Adv. channels in the Adv. mode, uses the Adv. channel PDU. As we have been discussing connection-less communication over Leg. Adv., we will particularly focus on Adv. channel PDU.

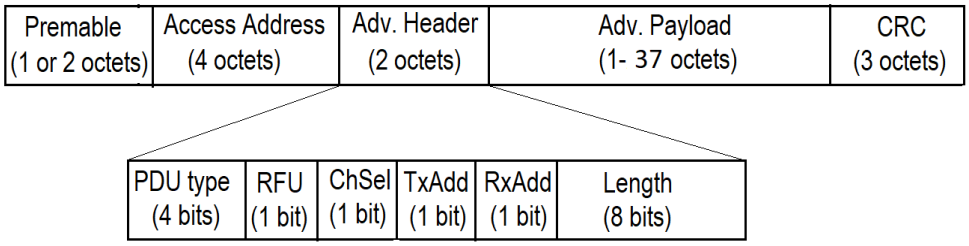

Figure 2.6: Link layer packet format for legacy advertising (adapted from [6]).

The *ChSel* (channel selection), *TxAdd* (transmitter address), and *RxAdd* (receiver address) fields of the Adv. header from the link layer Adv. channel PDU are specific to the different PDU type defined by Adv. modes [6]. *PDU type* 0010b indicates Non-Connectable and Non-Scannable Undirected (ADV_NONCONN_IND) PDU used for connection-less transmission. The length of the Adv. Payload is defined in the *length* field of the adv. header and is valid in the range of 1 to 37 octets. Legacy packets are not allowed to transmit more than 37 bytes as per the BLE standard.

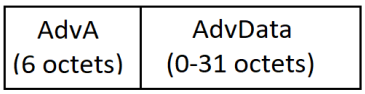

Figure 2.7: ADV_NONCONN_IND_PDU (adapted from [6]).

The ADV_NONCONN_IND PDU is used by the Advertiser in connection-less communication mode to broadcast Adv. packets on the Adv. channels. On broadcasting this PDUs, an advertiser does not expect any scan request

from the scanner in order to establish a connection. This PDU is particularly used for connection-less advertisement and shown in Figure 2.7. It is used in the Adv.Payload field of the advertising channel PDU shown in Figure 2.6. The *TxAdd* field in the Adv.Header (see Figure 2.6) indicates whether the advertiser's address is public (TxAdd = 0) or random (TxAdd = 1) and sets the corresponding address in the *AdvA* field of the ADV_NONCONN_IND illustrated in Figure 2.7. The length of the actual data can be calculated as:

$$\text{AdvData} = \text{Length from Adv.Header} - 6 \tag{2.1}$$

The limited payload length of Leg. Adv. can be a problem for some applications. For example, the limited payload size of 37 bytes is too small to carry all the characters in most URLs and another important data for some applications [**14**]. Fortunately, BLE 5 helps these applications by introducing Ext. Adv.

2.1.3.2 Extended Advertising:

Since the specification of BLE 5, another connection-less mode, called Extended Advertising is specified. A device using Ext. Adv. is able to transmit a longer data packets and may use other BLE radio channels for advertising than channels 37, 38 and 39. The channels used for Leg. Adv. are called primary Adv. channels, while the remaining channels are also used for advertising in Ext. Adv. called secondary Adv. channels [**6**].

Advertiser:

Ext. Adv. supports up to 254 bytes of user data in Adv. packets and it can be achieved by first advertising the Extended Advertising (ADV_EXT_IND) PDU on the primary Adv. channel, that points to an Auxiliary Advertising (AUX_ADV_IND) PDU on a secondary Adv. channel. AUX_ADV_IND PDU is the one who carries up to 254 bytes of actual user data. Ext. Adv. can be observed in Figure 2.8. An advertising event begins by sending Adv. PDUs on a primary Adv. channel with the first used Adv. channel index and ends with the last used Adv. channel index. The time between Adv. events is specified by the Adv. interval which can be set in the range of 20

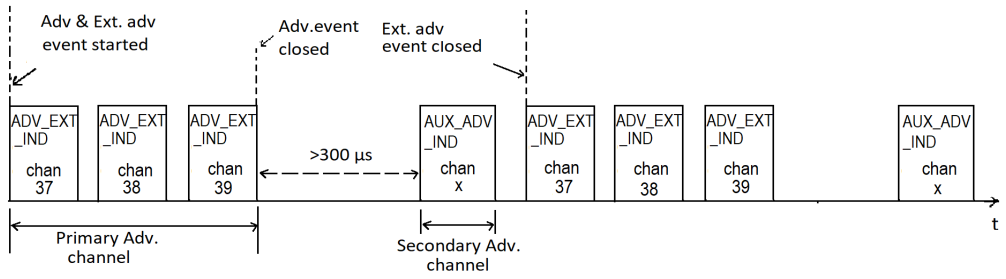

Figure 2.8: Extended Advertsing on an Advertiser (adapted from [6]).

ms to 10 sec. An Ext. Adv. event starts with an Adv. event and ends with an AUX_ADV_IND PDU if the user data is limited to 254 bytes. If the device needs to transmit more than 254 bytes of data, BLE 5 Ext. Adv. supports AUX_CHAIN_IND_PDUs and can be seen in Figure 2.9. This allows devices to broadcast a large amount of data in its Adv. events.

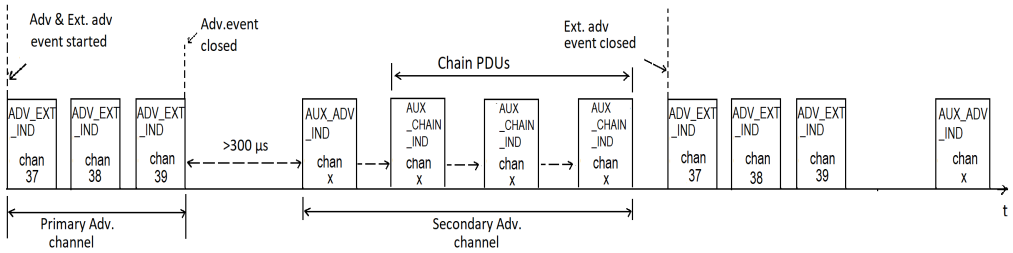

Figure 2.9: Exteneded Advertising with AUX_CHAIN_IND_PDUs on an Advertiser (adapted from [6]).

Data with a length of more than 254 bytes is fragmented into several packets. The fragmented data is transmitted on a secondary Adv. channel using AUX_CHAIN_IND PDUs following the AUX_ADV_IND PDU. AUX_ADV_IND PDU points to the AUX_CHAIN_IND PDU and AUX_CHAIN_IND PDU points to the next AUX_CHAIN_IND PDU and it forms a chain hence called as Chain PDUs.

Ext. Adv. also allows skipping the number of Adv. events before advertising AUX_ADV_IND. As seen in Figure 2.10, all the PDUs from both the Adv.

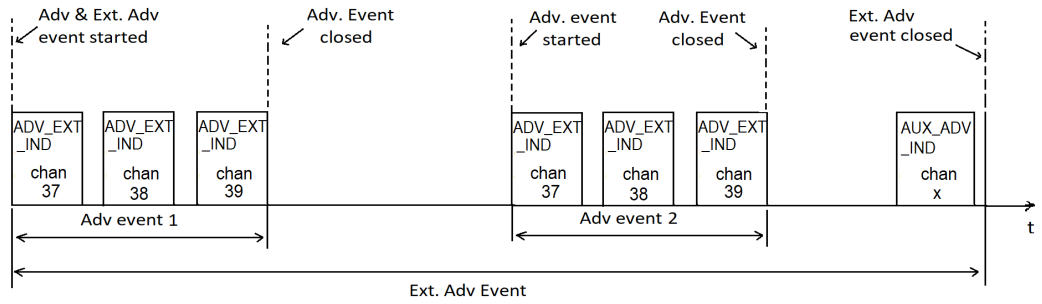

Figure 2.10: Skip number of advertising events before transmitting AUX_ADV_IND PDU (adapted from [6]).

events point to the same AUX_ADV_IND PDU. Ext. Adv. event skips one Adv. event (e.g., Adv event 1 as shown in Figure 2.10) before transmitting the AUX_ADV_IND PDU. The maximum number of events that can be skipped are 255 [6].

Scanner:

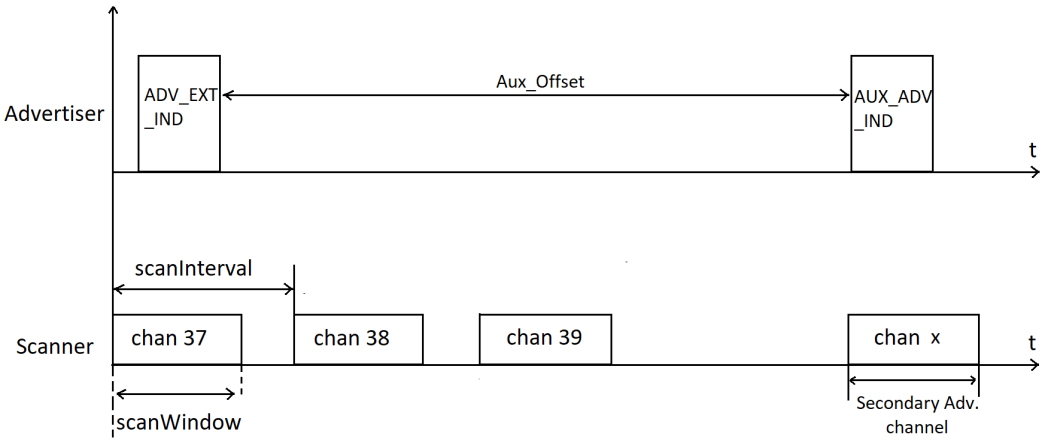

Figure 2.11: Extended Scanning with Aux packet (adapted from [6]).

In each scanWindow, the link layer scans on a primary Adv. channel index for Adv. packets and always scans all the primary Adv. channel indices

before repeating the same channels. The LL can scan continuously for a complete interval if *scanWindow* and *scanInterval* parameters are set to the same value. As shown in Figure 2.11, on receiving a PDU with the AuxPtr field present, the scanner should also listen to the auxiliary PDU it points to [6]. To achieve that, the scanner scans on a specified secondary Adv. channel on a particular time span pointed by the ADV_EXT_IND PDU.

Extended Advertising Packet Format:

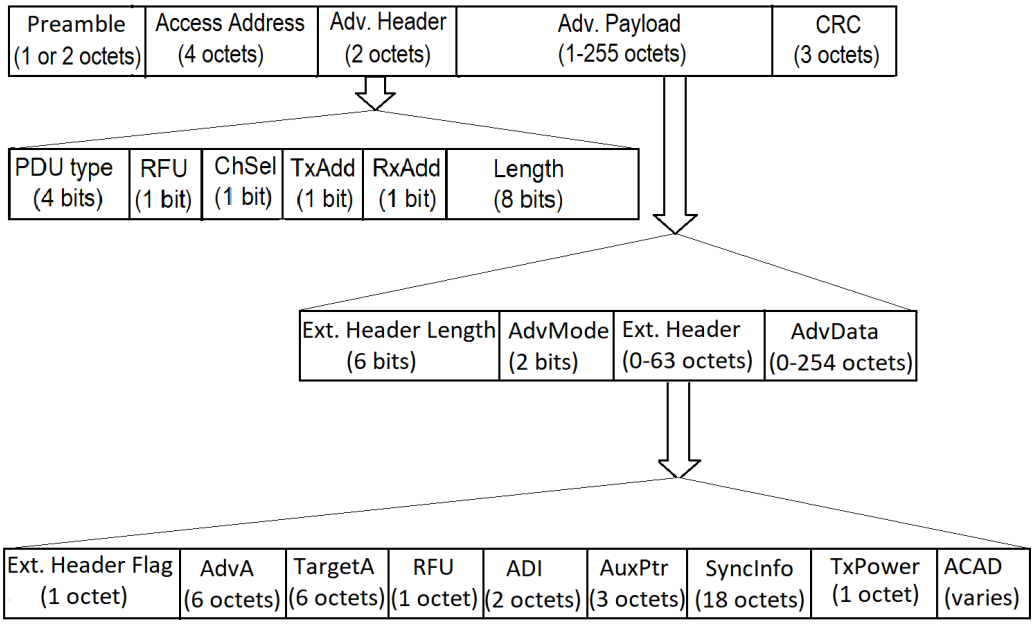

Figure 2.12: Extended Advertising channel PDU (adapted from [6]).

As BLE 5 supports several Adv. and data PDUs, Ext. Adv. PDU can be differentiated from other PDUs with PDU type 0111b. All other fields in the header remain the same as in the Adv. channel PDU header of Leg. Adv. (see Figure 2.6).

The Ext. Header Length indicates the length of the Ext. Header field and the value varies in the range of 0 to 63 octets [6]. The AdvMode field indicates

the mode of the advertisement and the value of the AdvMode field shall be set to oob for a Non-connectable Non-scannable mode to not receive any response from the Scanner. AdvData may contain advertising data from the host. However, the amount of actual data we can transmit is dependent on the number of fields present in the Ext. Header and can be calculated as:

$$\text{AdvData} = \text{PDU Adv. Len} - (\text{Ext. header Len} + 1) \tag{2.2}$$

The order of the Ext. Header fields present in the Ext. Header is the same as the flags present in Ext. Header flag (i.e., the AdvA field is first if present, then the TargetA field if present, etc.)[6].

The AdvA field sets the advertiser's device address and the TargetA field sets the scanner's or initiator's device address to which the Adv. packet is directed [6].

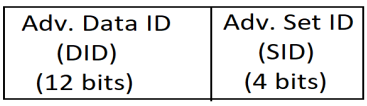

Figure 2.13: AdvDataInfo Field (adapted from [6]).

Adv. sets are used to transmit different Adv. events with different advertisement parameters (e.g., advertising PDU type, interval, PHY) simultaneously. These different Adv. sets transmitted by a single device are identified by Adv. Set ID (SID) [6]. The Adv. Data ID (DID) field of AdvDataInfo (ADI) (see Figure 2.13) is set to different IDs, depending on the data set in the payload field [6]. Different DIDs are set for different data.

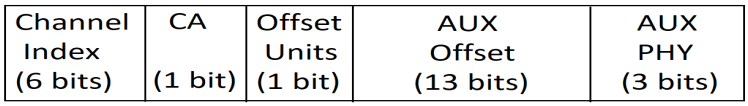

Figure 2.14: AuxPtr Field (adapted from [6]).

The ADV_EXT_IND PDU header contains an AuxPtr field in its Ext. Header, and carries the information about AUX_ADV_IND PDU advertised on a secondary channel. As shown in Figure 2.14, AuxPtr contains a time

offset (AUX Offset) from this primary Adv. packet and a secondary Adv. channel (Channel Index) on which AUX_ADV_IND PDU will be transmitted. Basically, the Scanner of this advertisement will be able to know exactly when and where to listen for the AUX_ADV_IND PDU. The Aux Offset is set at least the length of the ADV_EXT_IND packet plus Minimum AUX Frame Space (T_MAFS) which is 300 µs [**6**]. The Aux PHY field is set to the PHY among LE 1M, LE 2M and LE coded used to transmit the auxiliary packet.

2.1.3.3 Benefits of Extended Advertising:

As discussed earlier, the maximum data length in Leg. Adv. is limited to 31 bytes. To transmit longer than 31 bytes of data with legacy advertisement, the scan response (SCAN_RSP) PDU can be used, where an advertiser can transmit additional up to 31 bytes of user data in its payload. To transmit this additional SCAN_RSP PDU, a scanner must send a scan request (SCAN_REQ) PDU to the advertiser to request additional information about the advertiser. However, to transmit more than a total of 62 bytes of user data a device needs to establish a BLE connection. However, with an Ext. Adv., the user can send more than 31 bytes of data on the Adv. channels without even establishing a connection, which can potentially be more energy efficient for the same amount of data when no bi-directional data exchange is necessary.

For certain applications (e.g., temperature sensing device that monitors room temperature and sends 100 bytes of IPv6 packet every 5 min to the router BLE device that sends the temperature data to the Internet.), it is also important to maximize the battery life of the power-constrained devices on the receiving end. That can be achieved by enabling devices to sleep for a longer interval without dropping important advertised data. Figure 2.15 shows an example of how Ext. Adv. with event skip parameter is advantageous for low duty cycle devices. If a receiver failed to receive the Adv. packet of the first Adv. event due to its sleeping period, it gets another opportunity to receive an Adv. packet from another Adv. event point to the same Aux packet. With the information received by the Adv. packet, the

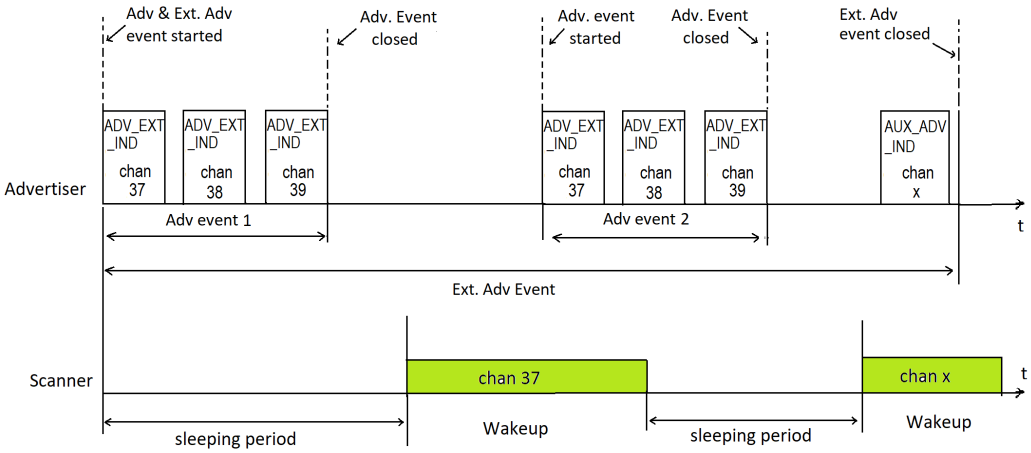

Figure 2.15: Event skip example with scanning (adapted from [6]).

scanner knows when to wake up again to collect Aux packet with original data.

2.1.3.4 Extended Advertising in BLE Stack:

As described in Section 2.1.3.2, Ext. Adv. allows BLE devices to broadcast a large amount of data in its Adv. packets over connection-less communication. BLE devices in Ext. Adv. act either as an advertiser or a scanner and the design of the Ext. Adv. in BLE stack is also based on these two roles of operation. The Ext. Adv. is enabled, disabled, and configured with the help of commands and events that are exchanged between the host and the controller part of the stack using the Host Controller Interface (HCI).

Advertiser

The procedure of how the host can start Ext. Adv. is shown in Figure 2.16. The host can start Ext. Adv. by issuing *HCI LE Set Extended Advertising Parameters* command to the controller to set the Adv. parameters. On receiving a command from the host, the controller generates the command complete event and acknowledges the host that such command was successfully executed. If the controller does not support a command, it shall return the

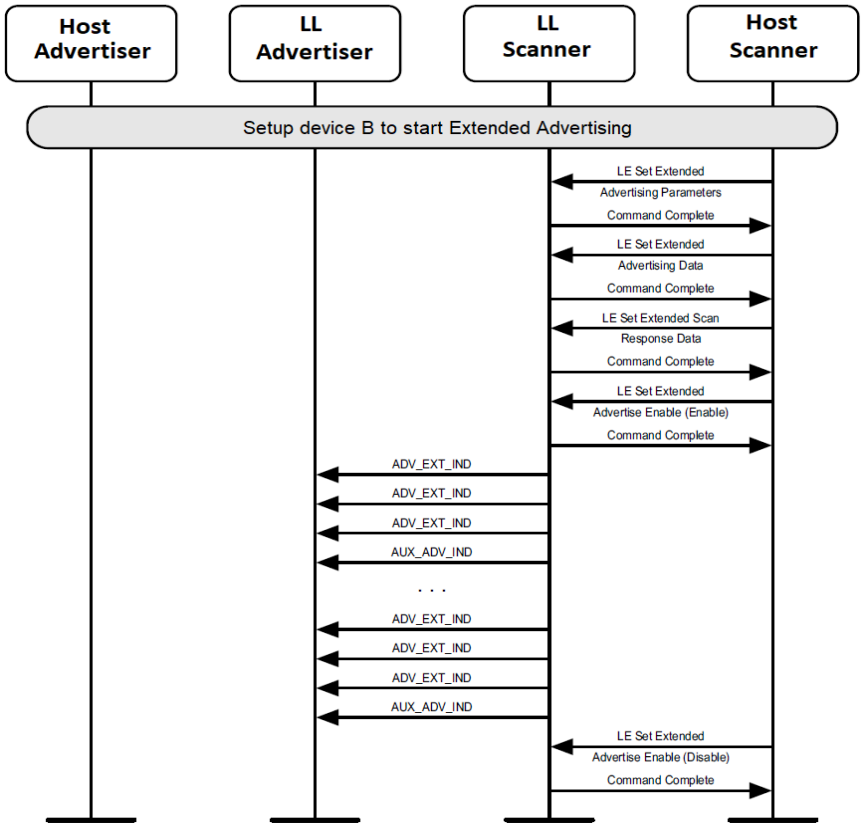

Figure 2.16: HCI commands and events for enabling Extended advertising [6].

respective error code. Furthermore, the host issues *HCI LE Set Extended Advertising Data* command in order to set the data used in advertising PDUs that have a data field. Although ADV_EXT_IND PDU does not carry any user data, the data set by this command is broadcasted in the payload field of AUX_ADV_IND PDU and AUX_CHAIN_IND PDUs. *HCI LE Set Extended Scan Response Data* command is issued on receiving the command complete event for the latest command. It is used to provide scan response data used in scanning response PDUs. For connection-less advertisement, this command can be skipped, as the advertiser does not expect any scanning request after the advertisement. The last but most important command to enable the advertisement is *HCI LE Set Extended Advertising Enable*. The controller only starts an advertising event when it receives a command to

enable it. The same command is used to disable an advertisement.

Scanner:

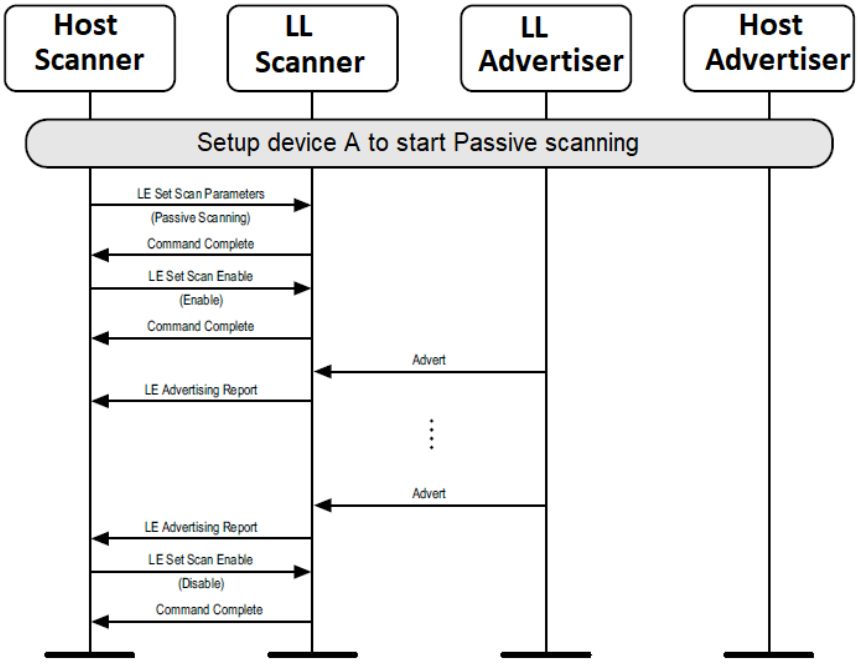

Figure 2.17: HCI commands and events for enabling scanning [6].

Figure 2.17 shows how the host on a BLE device can use the HCI interface to enable Extended Scanning. When entering into scanning state, the host needs to send the *HCI LE Set Extended Scan Parameters* command to the controller. The command is used to set the parameters of scanning state. Scan_window and scan_interval parameters define how long and how frequently a controller should scan. On receiving the command complete event from the controller, another important command the host needs to send is *HCI LE Set Extended Scan Enable* to enable scanning. After successfully switching a radio into the scanning state, the scanner expect advertising packets on the advertising channels. The advertising packet from other devices trigger an event called *LE Advertising Report* in the controller in order to notify the host about the received data. The termination of the

scanning state can be achieved by setting the value of the *Enable* parameter of the *HCI LE Set Extended Scan Enable* command to the "Scanning disabled" value.

2.2 IPv6 over BLE

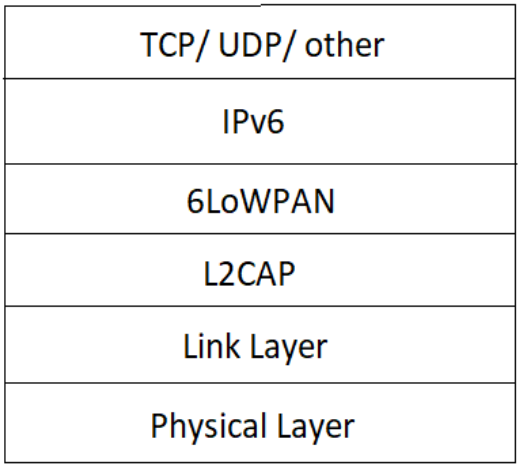

Figure 2.18: BLE stack for IPv6 over BLE (adapted from [17]).

Even though Bluetooth was originally introduced to allow communication among small devices, computers, and mobile phones, with the release of RFC 7668 [**18**], Bluetooth devices can have independent Internet access and the ability to exchange IPv6 packet over the Internet.

IPv6 is the state-of-the-art protocol for communication with a large number of Internet-connected devices since it provides a larger address space and allows each device to have its own IP address. Although BLE was not designed to provide IPv6 connectivity, the best way to transport IPv6 packets over BLE connection is by using IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) [**16**].

2.2.1 6LoWPAN over BLE

The 6LoWPAN is a special layer designed to provide IPv6 connectivity to the power-constrained devices. It was initially proposed to exchange IPv6 packets over IEEE 802.15.4 links [**24**]. Later recognizing the importance of IPv6 connectivity for other wireless technologies, the Internet Engineering Task Force (IETF) proposed a standard RFC 7668 [**17**] that enables IPv6 communication over the BLE standard by using a revisited 6LoWPAN layer. This adaptation layer is located in between the BLE L2CAP and IPv6 layer of the network stack, as shown in Fig.2.18.

The 6LoWPAN layer of BLE handles the encapsulation of IPv6 packets. At the time in which the RFC 7667 was released, the maximum transmission unit (MTU) of the BLE logical link was 23 bytes, which was to transmit the 40 bytes of complete IPv6 header and the 8 bytes of UDP header. To improve the performance of BLE devices with respect to IPv6 connectivity, it was needed to implement header compression. The 6LoWPAN layer of BLE performs the compression and decompression of IPv6 as well as UDP headers. Due to the limited payload size of the logical link of BLE, fragmentation, and reassembly of the 1280 bytes or larger IPv6 packets is performed by the L2CAP layer and not by the 6LoWPAN layer like in other 6LoWPAN standards.

2.2.2 Network Topology

IPv6 over BLE uses a connection-based approach. The peripheral devices communicate with a central device with a separate link and form a star topology or one subnet. In a subnet, peripherals act as nodes (6LNs) and the central device acts as a border router (6LBR). The figure 2.19 shows a typical instance where the BLE subnet is connected to the Internet. 6LN devices communicate through the 6LBR to the other 6LN device or to the Internet [**17**]. The border router forwards the IPv6 packets between 6LN and the Internet.

In some cases, the BLE subnet is not connected to the Internet and becomes a private network. In that case, the 6LBR acts as a central and rout the IPv6

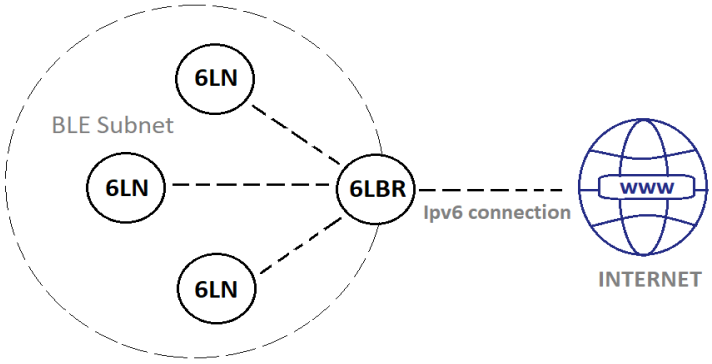

Figure 2.19: BLE subnet with Internet connectivity (adapted from [17]).

packets among the 6LN, which act as Nodes of the subnet, as shown in Figure 2.20.

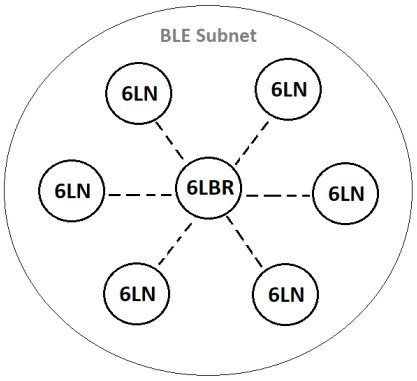

Figure 2.20: BLE subnet without Internet connectivity (adapted from [17]).

2.2.3 Stateless Address Autoconfiguration

The BLE IPv6 link-local addresses of both 6LN and 6LBR are generated based on the 48-bit Bluetooth device addresses. The link-local address of the BLE devices is formed with the 64-bit Interface Identifier (IID) which is formed by inserting oxFF and oxFE in the middle of the 48-bit device

address and the prefix fe80::/64, as described in RFC7136 [**19**] and RFC 4291 standards [**20**].

2.2.4 Header Compression

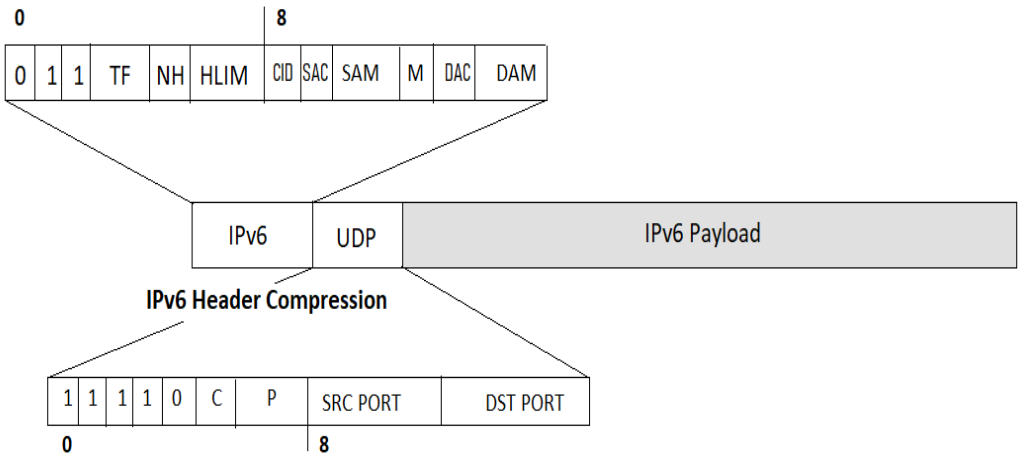

Figure 2.21: Compressed IPv6 and UDP header (adapted from [**14**]).

As mentioned in Section 2.2.1, IPv6 and UDP Header compression is the most significant function of the 6LoWPAN layer. The RFC 6282 [**21**] standard has described the rules for compression of IP headers. The main idea behind header compression is that IPv6 header of BLE packets uses some common and redundant values which can be compressed and either fully or partially elided and can be restored on receiving end [**14**]. For example, 128-bit source and destination IPv6 address can be replaced by a 4-bit context identifier (CID) and recovered with the help of shared contexts. Compressed format of IPv6 and UDP header is shown in Figure 2.21.

The LOWPAN_IPHC Encoding format defined in RFC 6282 [**21**] standard is used to compress the IPv6 header. The encoding can be 2 or 3 bytes depending on the context. The unelided fields of the IPv6 header are appended after the LOWPAN_IPHC in compressed or partially elided format. These 2 bytes of base encoding include TraffIc class and Flow Label (TF), Next Header (NH), Hop Limit (HLIM), Context Identifier Extension (CID), Source

Address Compression (SAC), Source Address Mode (SAM), Multicast Compression (M), Destination Address Compression (DAC) and Destination Address Mode (DAM). The UDP header is also compressed using LOWPAN_IPHC Encoding into 2 bytes. The compressed UDP header contains Checksum (C), Port (P), and source port and destination port.

2.3 The Zephyr Operating System

The Zephyr OS is a real-time open-source operating system hosted by the Linux Foundation and used for low cost, connected, and resource-constrained IoT devices [22]. This operating system is written in C and assembly language and runs on platforms with as few as 8kB of memory. It is optimized for power-constrained and small memory footprint devices. In terms of connectivity, it supports many wireless communication protocols such as Bluetooth Low Energy, Wi-Fi, NFC and IEEE 802.15.4 while providing full IP network stack with HTTP, UDP and TCP support. It also supports low-power IoT communication standards like 6LoWPAN, CoAP, and MQTT.

2.3.1 Data Sending (TX):

The process of sending IPv6 data over connection-based BLE using the Zephyr network stack can be observed in Figure 2.22:

1. The network application opens the BSD socket and uses it while sending the data. The application data is fed into the buffer and prepared to be sent to kernel space.
2. If the socket is open for a UDP connection, the UDP header is initially added in front of the application data in the network protocol layer. The header of the network protocol completely depends on the socket type. Furthermore, the IP header is added to the UDP/TCP packet.

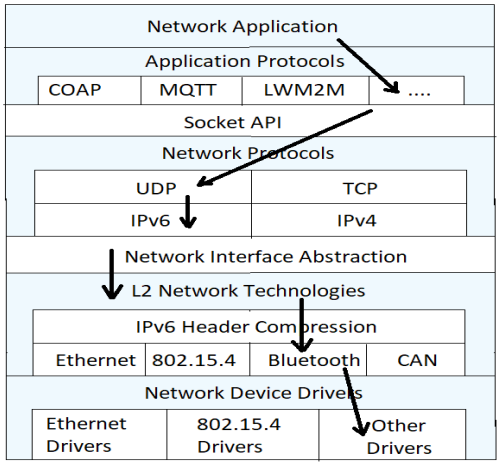

Figure 2.22: Network data flow while sending a Network packet in the Zephyr Network stack (adapted from [22]).

3. In order to send the IP packet over L2 network technologies, the interface needs to be set up and enabled. The network abstraction layer checks the interface and, if it is set up, it passes the packet to the lower layer.
4. The packet is checked by a dedicated L2 layer and adds its header to the packet and sends the packet to the network device driver layer.
5. The network device driver layer sends the packet on its physical channel.

2.3.2 Data Receiving (RX):

Whenever a device receives an IPv6 packet over connection-based BLE, the received network packet is processed in the network stack in the following manner:

1. The device driver layer receives the network packet. This layer assigns a RX buffer for the received packet and passes it to the correct L2 network technologies layer.
2. The dedicated L2 technology from the L2 network technologies layer stripes the header of the L2 layer and passes it to the upper layer.

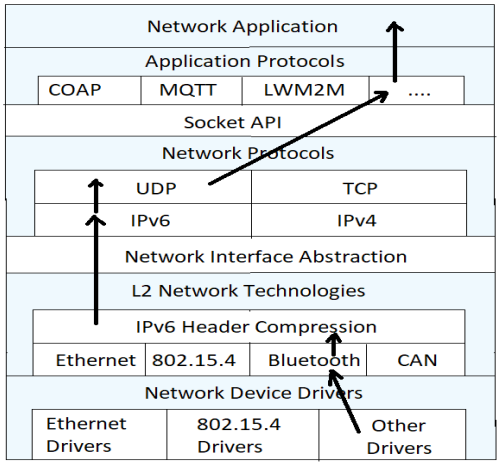

Figure 2.23: Network data flow while receiving a Network packet in the Zephyr Network stack (adapted from [22]).

3. If the packet is IP-based, the Network protocol layer checks if it is a valid IP packet by calculating the checksum and by checking the fields of the IP header. If this is the case, then IP and UDP headers are stripped from the packet.
4. It finds the active socket to which the received packet belongs, and passes the data to the application layer.
5. The application layer receives the data and processes it.

2.4 Hardware

Nordic Semiconductor nRF52:

The nRF52 DK is a popular development kit for ultra-low power operations that supports Bluetooth 5, Bluetooth mesh, ANT, NFC, and 2.4 GHz wireless protocols [34]. The kit can be powered up by USB or external sources. It can also be powered by a CR2032 battery for on-field testing. The application runs on the ARM® Cortex®-M4 32-bit processor with 64 MHz clock speed. It also offers 512 kB of flash and 64 KB of RAM. Nordic semiconductor offers a so-called SoftDevice with a full BLE communication stack. This SoftDevice

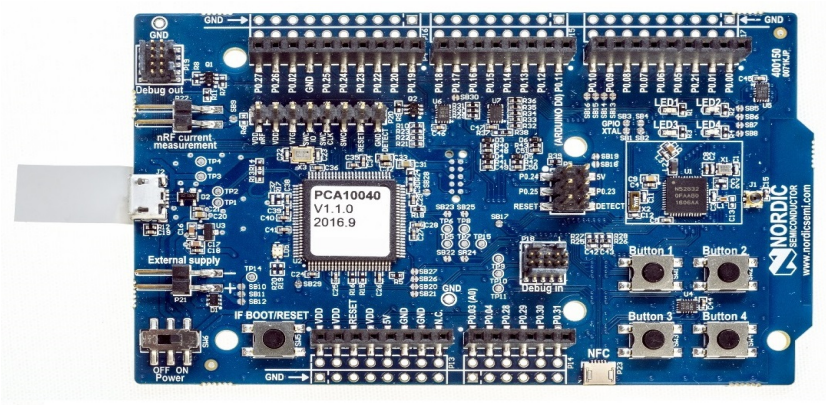

Figure 2.24: Nordic Semiconductor nRF52 development kit [23].

is easily available on the Internet but its source code is restricted. The significant advantage of this SoC is that it is supported in many open-source operating systems for resource-constrained devices like Zephyr RTOS, that allows full access to the inner working of the BLE controller. Using Zephyr RTOS and the nRF52 platform, we are able to have full control over the BLE radio and can implement our approach.

2.5 Related Work

This section explains the existing scientific studies that are related to this thesis (Section 2.5.1). It also summarises existing BLE stack implementations for resource-constrained devices that can be considered while implementing this thesis work (Section 2.5.2).

2.5.1 Research Studies

Yoon et al. [25] used the connection-less BLE to transfer emergency data of Healthcare device using IPv6 packets over IPv6 over BLE (6BLE) platform. The design uses connection-based and connection-less communication to transmit emergency data to the emergency center. The authors focus on

a critical medical situation when the patient's paired smartphone is not connected, the emergency data should reach the emergency center at any cost. Hence, the authors suggest the use of the Advertising process to transmit emergency data to the non-paired BLE devices. Since BLE advertising packets support only up to 31 bytes in the payload, fragmentation and segmentation have to be performed to transmit at least 66 bytes size of emergency data including IPv6 compressed header. To transmit 66 bytes of emergency data through advertising packets, the authors have defined a fragmentation header that contains fragmentation dispatch, length and fragmentation identifier that is set using 6LoWPAN. When the critical situation arises, the healthcare device starts advertising and sends fragments using LE_update_Adv_data HCI command. The scanner on the other end gathers the fragments using LE_Adv_reports events and generates full IPv6 packet.

The evaluation of this work has been performed with 102 bytes of UDP packet. Since the Adv. payload size is limited to 31 bytes, excluding the fragmentation header advertising packet offers only 24 bytes for the user data. Although the design supports IPv6 packets over the advertising process, it would only be suitable for small IPv6 packets. Hence, this work does not adhere to the minimum MTU size of 1280 bytes mandated by IPv6. The authors also suggests the use of a smartphone gateway application, which converts the BLE packet to the IPv6 packet, but it does not provide end-to-end IP connectivity between two devices.

Spörk [**16**] presented the design and implementation of an IPv6 over BLE communication stack using Contiki OS. The implemented stack is open source, which allows others to use the code and do further research. The design of Spörk's IPv6 over BLE communication stack introduces an additional BLE-HAL layer. The implemented BLE-HAL layer supports CC2650 BLE hardware and every hardware should provide a hardware-specific implementation of BLE-HAL to support IPv6 over BLE stack. Furthermore, Spörk also indicates the possibility of a more energy-efficient IP stack than the existing stack implementation by suggesting the use of advertising packets to exchange IPv6 packet between network devices as future work.

In contrast to these existing researches, we used the Ext. Adv. feature of BLE 5 to transmit large IPv6 packets over connection-less communication. It uses

Adv. PDUs to transmit these IPv6 packets to the non-connected BLE devices. As we discussed earlier, Yoon et al. implemented a new fragmentation header to transmit IPv6 packets using Adv. PDUs. With our approach, the fragmentation and the reassembly of an IPv6 packet is handled by the Ext. Adv. itself. Since each fragmented PDU carries up to 254 bytes of user data, devices need to transmit less packets to transmit the 1280 bytes of IPv6 packet compared with Yoon et al's. approach. This may also lead to more energy-efficient and more reliable connection-less IPv6 communication.

2.5.2 BLE Stacks

This section summarizes the most relevant implementation of BLE communication stack for resource-constrained BLE devices.

BLE stack	Open Source	IPv6	BLE-5	REF
Contiki OS - BLE stack	YES	YES	NO	[27]
Nordic Semiconductor - BLE stack	NO	YES	YES	[22]
Texas Instruments - BLE stack	NO	NO	YES	[29]
Apache Mynewt - NimBLE	YES	NO	YES	[30]
Zephyr RTOS - BLE stack	YES	YES	YES	[22]

2.5.2.1 Contiki OS BLE Stack

Contiki is an open-source operating system for resource-constrained IoT devices. It is designed and developed to provide standardized low-power wireless communication to the resource-constrained devices [27]. Contiki supports Internet protocols such as IPv4, IPv6, and 6LoWPAN. Although Michael Spörk has provided support to the open-source IPv6 over BLE through his thesis [16], at the time of implementation Contiki was not supporting any BLE-5 supported hardware. Furthermore, Contiki does not support the exchange of IPv6 packets over connection-less BLE.

2.5.2.2 Nordic Semiconductors BLE Stack

Nordic Semiconductors supports the IPv6 over BLE 6LoWPAN adaptation layer as defined in the RFC7668 standard draft [28]. Nordic Semiconductors also provides support to BLE 5 features such as increased broadcast capacity, long-range mode and 2 mbps data rate in its software development kits (SDK). The application layer interacts with the API provided by SDK to use the communication stack known as a SoftDevice. Nordic SoftDevices are basically pre-compiled binary with a Bluetooth communication stack without run time dependencies: hence, developers can not change the functionality or add additional features to the communication stack. This choice keeps the development cycle stable and prevents the developer from introducing a bug during the development. The SDK, however, is a proprietary, closed service binary that does not allow fine-grained control over the BLE radio (e.g., provide the control over used channels for Ext. Adv.) such control is necessary in order to create an effective IPv6 over connection-less BLE solution.

2.5.2.3 Texas Instruments BLE Stack

TI SimpleLink BLE5-Stack Software Development Kit offers the functionality of BLE5 with the CC2640R2F platform [29]. The controller, host, and application part of the stack are implemented on the CC2640R2F as a single-chip solution. The application and BLE stack is based on Texas Instrument RTOS know as TI-RTOS. Though the BLE stack supports Ext. Adv. feature to transmit a large amount of data in its Adv. packets, it doesn't support IPv6 over BLE communication. Additionally, the stack is not open-source.

2.5.2.4 Apache Mynewt NimBLE

Apache Mynewt offers NimBLE, known to be the world's first open-source BLE stack with both host and controller implementations and is compliant with BLE 5 specifications [30]. It also includes HCI implementation, which allows devices to use the different controller and host parts from different vendors. Though NimBLE supports Ext. Adv., 2Msym/s PHY for

higher throughput, Coded PHY for LE Long Range features of BLE 5, it doesn't support IPv6 over BLE communication. The latest Apache Mynewt 1.7.0, Apache NimBLE 1.2.0 released (August 4, 2019) doesn't support IPv6 communication nor the expected release date is available.

2.5.2.5 Zephyr RTOS BLE Stack

As explained in Section 2.3, Zephyr is an open-source operating system designed for low-powered resource-constrained IoT devices. It offers a full IP network stack while supporting several link-layer technologies such as Bluetooth Low Energy, Wi-Fi, NFC and IEEE 802.15.4. Besides, it is fully compliant with RFC 7668 standard that allows BLE devices to exchange IPv6 packets with other IP supported BLE devices. Furthermore, though the BLE stack of the Zephyr RTOS is BLE 5 compliant and provides support to BLE 5 supported hardware, the Ext. Adv. feature of BLE 5 is not implemented in the existing BLE stack.

However, the open-source implementation of the entire BLE stack gives full control over the radio, which allows us to implement Ext. Adv. on the controller as well as the host part of the BLE stack. Moreover, since the network stack is already compliant with RFC 7668 standard, the open-source implementation gives us an opportunity to implement an IPv6 over connection-less solution using existing implementation.

3 Connection-less vs Connection-based BLE

There are two communication modes supported by BLE 5, one is connection-based and the other one is connection-less. How they can be used to provide IPv6 connectivity to IoT devices is presented in this chapter. Section 3.1 explains the connection-based communication while Section 3.2 explains connection-less communication with all the necessary formulas required to perform theoretical analysis. Section 3.3 compares the connection-based BLE vs connection-less BLE in detail in terms of their radio-on time.

3.1 Connection-based BLE

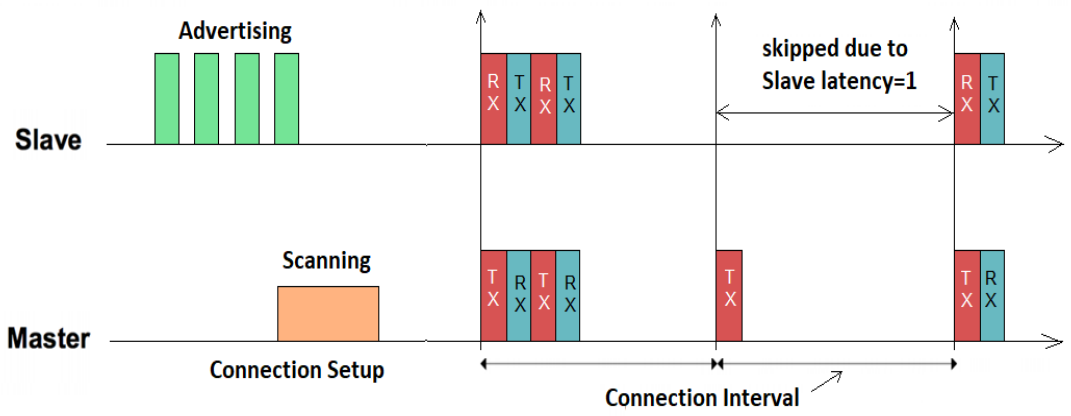

Figure 3.1: Connection-based BLE communication (adapted from [37]).

In connection-based BLE, bi-directional data transfer occurs between a master and a slave. As shown in Figure 3.1, the connection is set up between both devices in the connection setup phase by exchanging packets on primary Adv. channels. When the BLE connection is successfully established, application data is exchanged over the data channels in connection events. The timing of connection event is specified by the *Connection_Interval* parameter. In each connection event, the master transmits the first packet and slave respond to that. However, a slave can skip the number of connection events by using the *Slave_Latency* parameter. It means, the slave doesn't wake up to receive the packet from the master and does not transmit any packet in that connection event in order to conserve energy. The maximum *Slave_Latency* depends on the *Connection_Interval* and *Supervision_Timeout* parameter. If no packet is exchanged during maximum *Supervision_Timeout* of 32 sec, the connection is considered to be lost **[6]**.

We are interested in the radio-on time required to transmit D bytes from slave to master using the connection-based BLE mode.

The total Radio-on time of slave device is

$$t_{on} = t_{data} + t_k, \tag{3.1}$$

where t_{data} is the time needed to transmit the actual data packets, while t_k is the radio-on time of the slave required to exchange empty packets with the master.

The number of connection events between exchanging two subsequent IPv6 packets is calculated as:

$$N_{CE} = \left\lceil \frac{t_a}{Connection\_Interval} \right\rceil, \tag{3.2}$$

where t_a is the time between two IPv6 packets (application interval) **[37]**, while *Connection_Interval* is the BLE parameter that decides the timing of each connection event as discussed above.

The number of connection events required to transmit application data is

$$N_{CE\_data} = \left\lceil \frac{D}{F} \right\rceil, \tag{3.3}$$

where D is the data in bytes, F is the capacity of connection event [37].

The slave device can skip a number of connection events using *Slave_Latency*. The *Slave_Latency* can be configured and dynamically changed, but needs to adhere to:

$$Slave\_Latency = ((Supervision\_Timeout)/(connection\_Interval \cdot 2)) - 1, \tag{3.4}$$

as shown in [6].

where $Supervision\_Timeout$ is the timeout for connection as discussed above.

The total number of connection events within an application interval t_a when the slave exchanges keep-alive packets with a master is calculated as:

$$N_{slave\_CE} = \left\lceil \frac{N_{CE} - N_{CE\_data}}{Slave\_Latency + 1} \right\rceil, \tag{3.5}$$

In a connection event, when a slave wakes up to exchange the keep-alive packet, it exchanges two keep-alive packets with a master. One is from master to slave and the other one is from slave to master. Hence, the total number of keep-alive packets exchanged within an application interval is calculated as:

$$N_{slave} = N_{slave\_CE} \cdot 2, \tag{3.6}$$

The total number of bytes exchanged within an application interval to exchange keep-alive packets is given by

$$N_{slave\_bytes} = N_{slave} \cdot S_k \tag{3.7}$$

where S_k is the size of a keep-alive (empty) packet which is a link-layer packet without data. The keep-alive packet consists of a 1-byte preamble, 4-byte access address, a 2-byte link-layer header, and a 3-byte CRC, in total $S_k = 10$ bytes.

The total radio-on time required to exchange keep-alive packets in an application interval (t_a) can be calculated by

$$t_k = \frac{N_{slave\_bytes} \cdot 8}{data\,rate}, \tag{3.8}$$

where the multiplication of 8 converts a number of bytes into a number of bits. While *data rate* can be 1 Mbps for 1M PHY channel.

The radio-on time to transmit D bytes from slave to master depends on the number of D bytes and the number of packets required to transmit it. The maximum data-carrying capacity P of a data packet in BLE 5 is 251 bytes. Since we are considering low duty-cycled applications, we assume the connection interval long enough to send the whole IPv6 packet in one connection event. The total number of bytes required to send D bytes including link-layer header is given by

$$S_{data} = \begin{cases} \lceil \frac{D}{P} \rceil \cdot P + \lceil \frac{D}{P} \rceil \cdot S_k, & \text{if } D \bmod P = 0 \\ \lfloor \frac{D}{P} \rfloor \cdot P + \lfloor \frac{D}{P} \rfloor \cdot S_k + (D \bmod P) + S_k, & \text{otherwise} \end{cases} \tag{3.9}$$

as shown in [37].

The total radio-on required to transmit S_{data} in an application interval is calculated as:

$$t_{data} = \frac{8 \cdot S_{data}}{data\,rate}. \tag{3.10}$$

3.2 Connection-less BLE

In connection-less communication, an advertiser transmits packets on the Adv. channels while the scanner scans for these packets on those channels. As we discussed in Section 1.1, connection-based communication may not be energy efficient for power-constrained low duty-cycled (eg., sending a packet every 15 minutes) devices, as it incurs an overhead of establishing and especially maintaining the connection. However, Leg. Adv. of connection-less mode does not incur an overhead of establishing and maintaining a

connection, but it is not suitable for larger data packets, as the maximum data to be broadcasted using Adv. packet is limited to 31 bytes. Therefore, we considered using an Ext. Adv. mode of connection-less BLE to remove this limitation, as discussed in Section 2.1.3.3.

The radio-on time of an advertiser to transmit actual user data using Ext. Adv. is given by

$$t_{on} = t_{primary} \cdot N_{primary\_chan} + t_{secondary} \tag{3.11}$$

where $t_{primary}$ is the amount of time spent to transmit one EXT_IND_PDU on the primary Adv. channel. $N_{primary\_chan}$ is the number of primary Adv. channels used to transmit EXT_IND_PDU packets. While $t_{secondary}$ is the time spent to transmit actual user data using AUX_IND and CHAIN_IND PDUs on secondary Adv. channels. The maximum user data or payload P that can be transmitted using AUX_IND_PDU and CHAIN_IND_PDU is 248 bytes. The radio-on time required to transmit EXT_IND_PDU on primary Adv. channel is calculated as:

$$t_{primary} = \frac{S_{primary} \cdot 8}{data\ rate}, \tag{3.12}$$

where $s_{primary}$ is the size of EXT_IND_PDU, which consists of 1-byte Preamble, 4-byte Access Address, 2-byte Adv. Header, 13-byte Ext. Header (1-byte Ext. Header Len and AdvMode, 1-byte Ext. Header Flag, 6-byte AdvA, 2-byte ADI, 3-byte AuxPtr) and 3 bytes CRC, in total $S_{primary} = 23$ bytes for 1Mbps data rate.

The size of link-layer header overhead S_{header} in AUX_IND and CHAIN_IND PDU is 1-byte Preamble, 4-byte Access Address, 2-byte Adv. Header, 8-byte Ext. . Header (1-byte Ext. Header Len and AdvMode, 1-byte Ext. Header Flag, 2-byte ADI, 3-byte AuxPtr) and 3-bytes CRC, in total $S_{header} = 17$ bytes. The size of the complete data packet is computed as follows

$$S_{data} = \begin{cases} \lceil \frac{D}{P} \rceil \cdot P + \lceil \frac{D}{P} \rceil \cdot S_{header}, & \text{if } D \bmod P = 0 \\ \lfloor \frac{D}{P} \rfloor \cdot P + \lfloor \frac{D}{P} \rfloor \cdot S_{header} + (D \bmod P) + S_{header}, & \text{otherwise} \end{cases} \tag{3.13}$$

The total radio-on time to send the actual user data on secondary Adv. channels using AUX_IND and CHAIN_IND PDU is given by

$$t_{secondary} = \frac{S_{data} \cdot 8}{data\,rate} \tag{3.14}$$

3.3 Comparison

Here we compare the connection-based and connection-less communication based on the number of bytes and the radio-on time required for temperature sensing device to transmit 100 bytes of IPv6 data periodically every 5 mins to other BLE router devices that sends the temperature data to the Internet.

The size of a packet in connection-based BLE consists of a 1-byte Preamble, 4-byte Access Address, 2-byte Adv. Header, 100-byte data, 4-byte MIC, 3-byte CRC, in total $S_{data} = 114$ bytes. While the number of bytes required to exchange keep-alive packets for 5 min is 380 bytes and it is obtained from equations (3.2)-(3.7). Therefore, to transmit 100 bytes of sensor data and maintain a connection for 5 min, the slave device exchanges 494 bytes.

Compared to connection-based BLE, the size of AUX_ADV_IND PDU in connection-less BLE consists of 17-byte S_{header} and 100-byte data, in total $S_{data} = 117$ bytes. While the size of ADV_EXT_IND_PDU is $s_{primary}$ which is 23 bytes.

Therefore, connection-less BLE with CL 1 configuration (ADV_EXT_IND_PDU is transmitted on only one primary advertising channel (channel 37)) requires 140 bytes while with CL 3 configuration (ADV_EXT_IND_PDU is transmitted on all three primary advertising channels (channel 37, 38, 39)) requires 186 bytes in total to transmit 100 bytes of sensor data.

Table 3.1: Comparison of the radio time required for sending data of 100 bytes using connection-based vs connection-less BLE.

	connection-based			**connection-less**	
	Data	**Keep-alive packet**	**Total**	**CL 1**	**CL 3**
Total bytes transmitted	114	380	494	140	186
Radio-on time (µs)	912	3040	3952	1120	1488

The table 3.1 shows that, the number of bytes and the radio-on time required to send 100 bytes of user data in connection-based BLE is 71 % higher than the connection-less BLE. Moreover, around 76 % of radio-on time in connection-based BLE is required to exchange keep-alive packets just to maintain a BLE connection, while only 24 % of radio-on time is needed to send actual user data in this scenario. This analysis shows that the connection-less BLE using only one primary Adv. channel (CL 1) requires 71 % less bytes and radio time while using all three primary Adv. channels (CL 3) requires 62 % less bytes and radio time than the connection-based BLE when transmitting 100 bytes data/ 5 mins.

The number of packet transmissions required to transmit a successful packet to achieve high reliability using connection-less BLE with CL 1 and CL 3 configuration is calculated as:

$$p = Probability\,of\,successful\,transmisson\,of\,packet. \tag{3.15}$$

$$P\,(N\,packets\,are\,unsuccessful) = (1-p)^N \tag{3.16}$$

$$P\,(At\,least\,1\,successful \mid N\,transmissions) = 1-(1-p)^N \tag{3.17}$$

where N is the number of transmissions for a successful packet transmission.

The values used for reliability of connection-less BLE with CL 1 and CL 3 configuration for single-packet transmission in table 3.2 are obtained from the experiment performed to evaluate the reliability of connection-based and connection-less BLE in Section 6.3. While the reliability for different number of packet transmission (N) is computed using equation 3.17.

Table 3.2: Number of packet transmissions required to achieve high reliability at the cost of radio-on time using connection-less BLE.

Packet transmissions (N)	CL 1			CL 3		
	Reliability	Radio-on time (µs)	Radio time saving compared to CB BLE	Reliability	Radio-on time (µs)	Radio time saving compared to CB BLE
1	54 %	1120	71 %	96 %	1488	62 %
2	79 %	2240	43 %	99.75 %	2976	24 %
3	90 %	3360	15 %	99.98 %	4464	-11 %
4	96 %	4480	-13 %	-	-	-

The table 3.2 shows that, the 90 % of reliability can be achieved with CL 1 configuration by sending packet 3 times while spending 15 % less radio-on time compared to connection-based BLE. Moreover, almost 62 % of radio-on time can be saved by providing 96 % of reliability when transmitting 100 bytes packet/ 5 mins.

4 IPv6 over Connection-less BLE

This chapter presents the design of the IPv6 over connection-less BLE communication stack. Section 4.1 and Section 4.2 explains the requirements and design challenges of IPv6 over connection-less BLE stack. Section 4.3 and 4.4 describe the features and limitations of the implemented communication stack. Section 4.5 discusses the design of a new L2 layer designed to transmit IPv6 packets over connection-less BLE communication.

4.1 Requirements

This section defines the requirements that needs to be fulfilled in the design of IPv6 over connection-less BLE.

IPv6 Compliant:

To make the IPv6 over connection-less BLE communication stack IPv6 compliant, devices must handle at least 1280 bytes of an IPv6 packet over connection-less BLE to support the Maximum Transmission Unit (MTU) of IPv6 protocol [**8**].

Handle fragmentation and reassembly of IPv6 packets:

In IPv6 over connection-based BLE, the fragmentation and reassembly of large IPv6 packets are performed in the L2CAP layer of the communication stack, as described in [**18**]. The L2CAP layer uses a logical link that runs over a single physical link to exchange data between two devices [**6**]. However, in connection-less BLE, the logical link can not be created as the physical connection is not established between two devices. As a result, the L2CAP

layer does not play any role in a connection-less communication and hence it is not available in the connection-less BLE. Therefore, it is required to handle the fragmentation and reassembly of the IPv6 packet in connection-less communication.

Dynamic switching between communication modes:

In some applications, devices may require to switch from connection-less to connection-based BLE communication to update routing information and application configuration. For example, an application of a temperature sensing device that monitors room temperature. In this scenario, the temperature sensor needs to transmit 100 bytes of IPv6 data periodically every 5 mins to other BLE router to send the temperature data to the Internet. Therefore, the temperature sensor may require to re-establishes an IPv6 over connection-based BLE connection to briefly check for IPv6 routing changes (e.g., every 60 mins) to update routing information and application configuration. Hence, the switching between IPv6 over connection-less BLE and connection-based BLE needs to be handled in the communication stack.

4.2 Design Challenges

In the design of IPv6 over connection-less BLE stack, a challenge that needs to be overcome is that we need to be able to transmit upto 1280 bytes of data in order to support the MTU allowed by IPv6 [8]. Since the maximum data-carrying capacity of Leg. Adv. packet is limited to 31 bytes, this mode is not suitable for IPv6 packets as an uncompressed IPv6 header itself is 40 bytes. Although Ext. Adv. feature of BLE 5 allows devices to broadcast up to 254 bytes of payload in its Adv. packet, the IPv6 packet needs to be fragmented into small chunks to transmit 1280 bytes of IPv6 packets. Unfortunately, as discussed in Section 4.1, the BLE L2CAP layer is not present in the connection-less BLE to handle the fragmentation and reassembly of IPv6 packets, the challenge to handle the fragmentation and reassembly of an IPv6 packet in connection-less BLE needs to be solved in the design of IPv6 over connection-less BLE.

To send an IPv6 packet, devices need to exchange the necessary information (network prefix, context information and configuration) that is required to configure a network interface. This information is exchanged using a network discovery procedure. In connection-based BLE, network discovery is performed on the top of a link-layer and L2CAP connection. However, in connection-less BLE, since link-layer and L2CAP connections are not established, neighbor discovery can not be performed. Therefore, initially, it is required to perform a neighbor discovery using connection-based BLE and later switch to connection-less BLE. The main challenge while designing the IPv6 over connection-less BLE is to dynamically switch from connection-based to connection-less BLE. Since in connection-based IPv6 communication, all the data packets are IPv6 packets and as the Internet protocol provides end to end connectivity, the user data sent by the application layer of the node devices is read by the application layer of the router. In order to request the router to switch to connection-less communication, the IPv6 packet with special data would have been sent to the router but it would have only read by the application layer. This makes the switching from connection-based to connection-less BLE application layer dependent. In this case, the handshaking needs to be implemented in the application layer, which makes the application implementation more complex. Another possibility could be to use the Link layer control PDU. It is nothing but the Data channel PDU that is used to control the link-layer connection [**6**]. The LL_TERMINATE_IND control PDU allows the other devices to know the reason for the termination of the connection. We can use this opportunity to inform the router to switch to connection-less communication. On receiving LL_TERMINATE_IND at the link layer, the device generates an event that informs the Host the reason for the disconnection.

4.3 Features

4.3.1 Open Source Implementation

Since the implementation of this thesis is performed in the open-source Zephyr OS, the thesis work will be available under the 3-clause BSD license,

which allows full open-source usage. The availability of open-source "IPv6 over connection-less BLE" implementation may open the possibility to perform experimental studies on the top of IPv6 over connection-less BLE by adding additional features or performing improvements in the implemented communication stack.

4.3.2 Toggle Communication Modes

Though this thesis provides support to unidirectional IPv6 packets over connection-less BLE, it also allows devices to toggle between connection-based and connection-less IPv6 communication. The toggling between connection-based and connection-less IPv6 BLE communication can be performed in the application layer. The devices can switch to either of these states as per the requirement of the application.

4.4 Limitations

The communication stack implemented in this thesis has the following limitations:

4.4.1 Specific to BLE 5

Since the Adv. PDU of BLE version 4.2 and more former versions support up to 31 bytes of user data in its payload, we used the Ext. Adv. feature from BLE 5 to transmit large IPv6 packets over connection-less communication.

4.4.2 Limited to Zephyr OS

Since the IPv6 over connection-less BLE is the newly proposed approach through this thesis, currently it is only implemented in open-source Zephyr OS. However, as the implementation is based on the HCI, it can also be implemented in other HCI supported BLE network stacks. Currently, both

the transmitter and the receiver of the application must use this newly implemented BLE stack to achieve connection-less IPv6 communication.

4.5 Design

This section presents the design of the IPv6 over connection-less BLE communication stack of the Zephyr OS. Section 4.5.1 discusses the changes in the existing Zephyr OS network stack to achieve IPv6 over connection-less BLE. Section 4.5.2 describes the design of IPv6 over connection-less BLE communication stack and the necessary steps to achieve the connection-less communication between a node and a router in detail.

4.5.1 Communication Stack

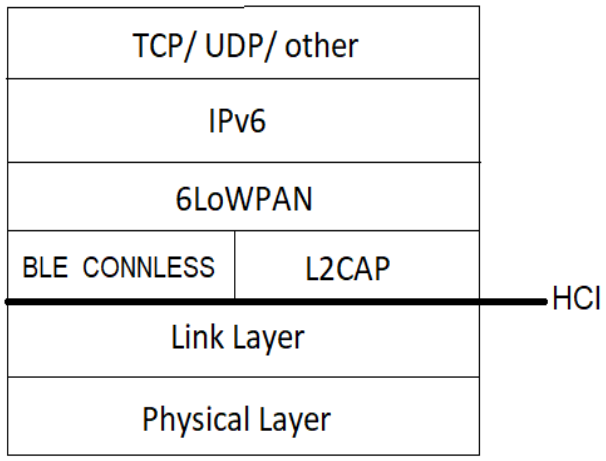

Figure 4.1: BLE network stack with IPv6 over connection-less BLE support.

Figure 4.1 shows the communication stack of connection-based IPv6 over BLE (right). This figure also includes the new BLE_CONNLESS layer (left) that is required to support IPv6 over connection-less BLE.

As discussed in Section 4.1, although the fragmentation and reassembly of large IPv6 packets are handled in the L2CAP layer in the IPv6 over connection-based BLE, it is a challenge to handle this in the connection-less BLE due to unavailability of the L2CAP layer in connection-less BLE communication. Therefore, I have designed the BLE_CONNLESS layer that handles fragmentation and reassembly of IPv6 packets in connection-less BLE.

The operation of the BLE_CONNLESS layer is role-specific. If the device is configured as a node, the BLE_CONNLESS layer receives IPv6 packets from the 6LoWPAN layer and splits this IPv6 packet into one or multiple fragments and send these to a controller with the help of `set Ext.Adv.data` HCI command. This IPv6 packet is transmitted to the router using Ext. Adv. with one or multiple Ext. Adv. packets. According to the BLE specification [**6**], the BLE radio is able to receive Ext. Adv. packets with a maximum payload length of 254 bytes. Therefore, the router receives a complete IPv6 packet using one or multiple Ext. Adv. packets. If the device is configured as a router, the reassembly of these received fragments is handled at the BLE_CONNLESS layer to send the complete IPv6 packet to upper layers. The fragmentation and reassembly of the IPv6 packet in connection-less BLE will be discussed in detail in Section 4.5.2.3.

As specified in RFC 4944 [**24**], the 6LoWPAN layer also offers the fragmentation mechanism for IPv6 packets. The 6LoWPAN layer splits the datagram into multiple fragments if it doesn't fit into the single link-layer frame. Each fragment carries an extra 4 or 5 bytes as a fragment header along with the fragment data. With the 6LoWPAN fragmentation mechanism, devices need to send an additional 4 to 5 bytes of fragment header with the user data in each Adv. packet. Therefore, to avoid the transmission of extra bytes in each Adv. packet the fragmentation mechanism provided by the Ext. Adv. and used in BLE_CONNLESS layer is considered instead of the 6LoWPAN fragmentation mechanism.

Moreover, the BLE_CONNLESS layer also handles the operations performed during the toggling of IPv6 communication between connection-based and connection-less BLE. It configures the controller to transmit and receive the IPv6 packets over connection-less communication and passes the received IPv6 packet to the lower and upper layer respectively. It configures the Ext.

Adv. and updates advertising data of Ext. Adv. with the new IPv6 packet. We will discuss the detail operation of BLE_CONNLESS layer in Section 4.5.2.2 and 4.5.2.3.

This new layer is fully compatible with the architecture of the existing Zephyr OS network stack and can be seen in Figure 4.2. In order to add support to the BLE_CONNLESS layer in the communication stack, users need to set the CONFIG_NET_L2_BTLESS flag in the configuration file of the application.

4.5.1.1 Overview of Zephyr Network Stack with BLE CONNLESS Layer

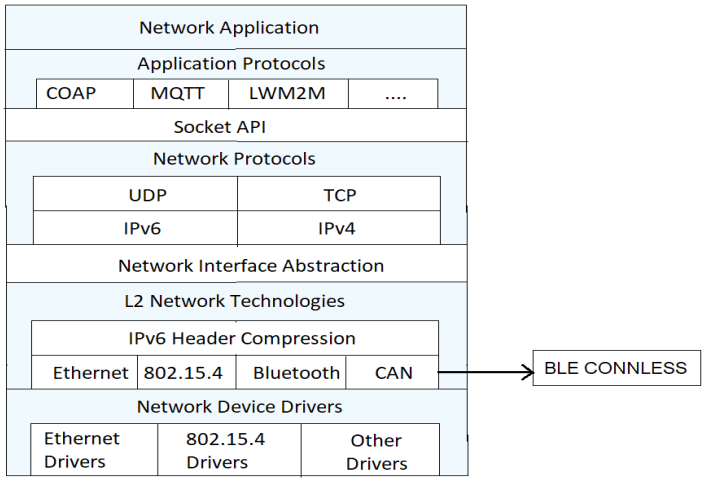

Figure 4.2: Zephyr OS Network Stack (adapted from [22]).

- **Network Application:** The network application opens a network connection, sends/receives data, and closes a connection by either using provided application protocols like MQTT, CoAP, LWM2M or accessing the BSD socket API directly [22].
- **Network Protocols:** UDP, TCP, IPv6, IPv4, ICMPv6, and ICMPv4 are part of the core network protocols which are used by Application protocols or BSD sockets to send or receive data [22].

- **Network Interface Abstraction:** The system can have a number of network interfaces and the basic functionality of all the network interfaces is provided in this layer [22].
- **L2 Network Technologies:** This layer supports several link-layer technologies like Ethernet, IEEE 802.15.4, CAN, Bluetooth, and newly added BLE_CONNLESS. Including BLE_CONNLESS some of these technologies also support IPv6 header compression [22].
- **Network Device Drivers:** The device drivers handle physical communication like sending and receiving packets [22].

4.5.2 Communication:

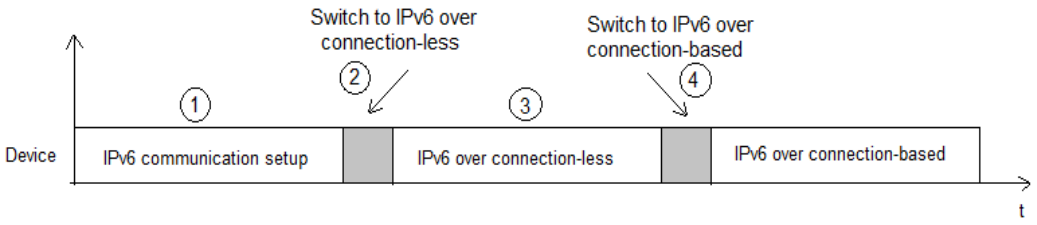

Figure 4.3: Step 1, 2 and 3 are mandatory to follow sequentially to achieve the IPv6 over connection-less communication. Initially, IPv6 communication is set up (step 1), then the transition is performed from IPv6 over connection-based to connection-less communication (step 2), later, IPv6 over connection-less communication is handled (step 3). If a device required to switch to connection-based BLE, that is accomplished by performing step 4.

The design of IPv6 over connection-less communication presented in this thesis can be observed in Figure 4.3. However, IPv6 over connection-less BLE can be achieved by performing step 1 (IPv6 communication setup), step 2 (switch to IPv6 over connection-less BLE) and step 3 (IPv6 over connection-less BLE). It is mandatory to follow steps 1, 2 and 3 sequentially to achieve an IPv6 over connection-less BLE.

For example, a node can be a thermostat that monitors the room temperature and sends 100 bytes of IPv6 packet to the router to send the sensor data to the Internet. In IPv6 communication setup step, initially, a BLE connection is established between a thermostat and a router by exchanging BLE

packets over the Adv. channels. Following the standardized BLE communication setup between both the devices, neighbour discovery is performed to achieve IPv6 communication (see Section 4.5.2.1). Furthermore, since all the necessary information is exchanged to perform IPv6 communication and devices are able to bi-directionally exchange IPv6 packets over the connection-based communication, the transition needs to be performed from IPv6 over connection-based to IPv6 over connection-less communication. This switch is handled in the switch to IPv6 over connection-less communication step (step 2). In this step, the thermostat requests the router to switch to connection-less communication by sending LL_TERMINATE_IND PDU with its disconnect reason field (see Section 4.5.2.2). On receiving a custom disconnect reason code, router perfoms all required steps (see Section 4.5.2.2) and both the devices switch to connection-less communication to conserve energy. Furthermore, in IPv6 over connection-less communication step, thermostat transmits the IPv6 packets over Adv. channels with the help of Ext. Adv. feature of BLE 5 and router scans for these IPv6 packets over those Adv. channels (see Section 4.5.2.3). Moreover, as discussed in Section 4.1, if the application demands to switch to connection-based communication to update IPv6 information, network prefix, and context information, the transition needs to be performed. In switch to IPv6 over connection-based communication (e.g. to bi-directionally exchange data or to update routing information) step, thermostat device stops transmitting IPv6 packets over Adv. channels rather it starts advertising connectable Adv. PDUs to re-establish communication with a router (see Section 4.5.2.4). Furthermore, both the devices establish a link-layer and the L2CAP connection and start exchanging IPv6 packets over connection-based communication.

4.5.2.1 IPv6 Communication Setup:

The IPv6 communication setup between a thermostat and a router (6LBR) can be observed in Figure 4.4. Initially, the thermostat acts as an Advertiser and starts transmitting advertising BLE packets on primary Adv. channels to all the neighboring BLE devices. By sending a connection request to the thermostat, the router establishes a link-layer connection with thermostat. Further, the router takes initiative to establish an L2CAP connection between both the devices. Once the L2CAP connection is established between

thermostat and router, IPv6 communication is possible and neighbor discovery can be performed. Therefore, as next step, the thermostat sends a Router Solicitation (RS) message to the all-routers multicast address and waits for Router Advertisement (RA) message from the router device that carries vital information such as IPv6 prefix and contexts that will be used in the IPv6 header compression of 6LoWPAN [**16**]. A thermostat registers its non-link-local address with the router using Neighbor Solicitation (NS) and waits for the successful acknowledgment from the router. At this point, both devices can exchange IPv6 packets over connection-based communication.

Link-Layer Connection

The thermostat acts as an Advertiser when it is not connected to any other BLE router and starts broadcasting connectable Adv. packets on the primary Adv. channel. As per the Internet Protocol Support Profile (IPSP) introduced by Bluetooth SIG and the IETF in order to enable IPv6 over Bluetooth LE, the thermostat must implement the GATT server role and an IP Support Service (IPSS) while the router must implement a GATT client role to discover a thermostat using IPSS. When a router receives the Adv. packet transmitted by a thermostat that includes a service universally unique identifier (UUID) field of advertising data (AD) with IP Support Service UUID, it sends a connection request to that thermostat. After receiving a connection request from the router, the link-layer connection is established between two devices. However, the router also ensures that no two devices with the same link-local address are connected at the same time, as it knows the link-local address of all connected devices.

BLE L2CAP Connection

Once the link-layer connection is established between a thermostat and a router, the router takes the initiative to establish an L2CAP connection by sending a connection request to the thermostat. On receiving the L2CAP connection request, the thermostat opens the L2CAP channel which is a logical link between these two devices to exchange IPv6 packets over connection-based communication mode. A thermostat informs the successful opening of the L2CAP channel by sending an L2CAP connection response to the

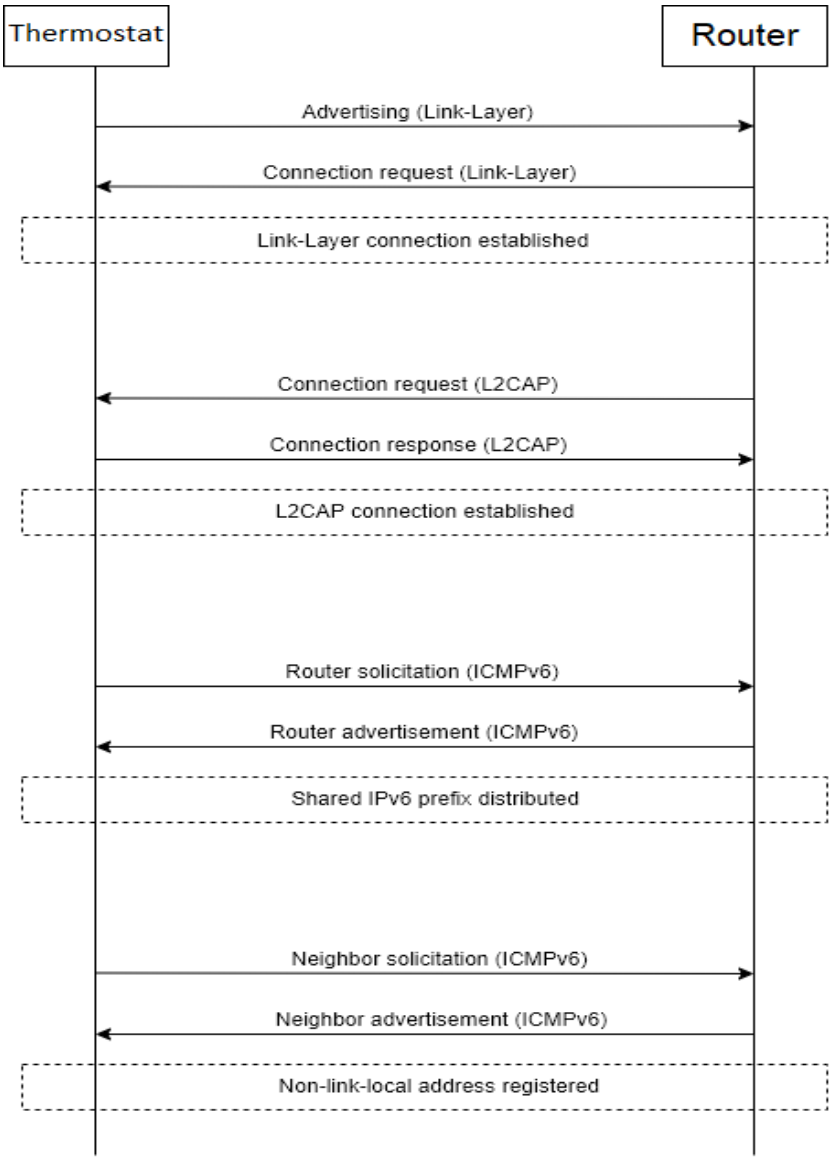

Figure 4.4: The steps followed and the messages exchanged between a thermostat and a router to configure both the devices to exchange IPv6 packets. Initially, the BLE link-layer connection is established, second, a BLE L2CAP connection is created, then the IPv6 prefix is exchanged and later thermostats non-link-local address is registered at the router (adapted from [**16**]).

thermostat. At this point, both the devices are connected and can exchange data packets over the L2CAP channel.

The Maximum Transmission Unit (MTU) of L2CAP channels over Bluetooth LE is at least 23 octets but may carry up to 251 bytes with LE Data Length Extension feature. Hence to transmit the IPv6 packet of 1280 bytes or larger, fragmentation and reassembly are provided by the L2CAP layer.

Router Solicitation

After the L2CAP connection is established between the thermostat and the router, Neighbor discovery is performed and it is presented in RFC 6775 [35]. At this point, a thermostat doesn't know a network prefix or the router that will provide a network prefix to routing the packets over the internet. Hence a thermostat sends a Router Solicitations (RS) which is an Internet Control Message Protocol (ICMP) message with its link-layer address as a Source link-layer

Address Option (SLLAO) to the all-routers multicast address and waits for a response from the router. On receiving RS from a thermostat, a router replies with Router Advertisement (RA) message to the address mentioned in the SLLAO field of the RS, with parameters such as current hop limit, IPv6 prefix information, context information that is used in the header compression of 6LoWPAN and flags for how devices generate IPv6 addresses. When the thermostat receives an RA, it creates a global IPv6 address which is called as a Non-link-local IPv6 address and in the following step attempts to register with a router.

Neighbor Solicitation

In order to register the created global IPv6 address with a router, a thermostat sends a neighbor solicitation (NS) message with an Address Registration Option (ARO) and SLLAO. The NS is sent with the created IPv6 address as a source address. A router verifies received IPv6 address for duplicates with its neighbor cache. The router maps the received IPv6 addresses with their link-layer address to perform duplicate address detection(DAD). If the received non-link local address doesn't exist in the neighbor cache, it is registered with the router as it is not a duplicate. If the received non-link

local address does exist, its mapped link-local address is checked with the one received with the SLLAO field of the NS message. If they are the same, it is re-registered or error is returned. The status of the registration of the Non-link layer address is acknowledged by sending a Neighbor Advertisement (NA) message to the thermostat with a status field.

Now, devices are set to exchange IPv6 packets with each other.

4.5.2.2 Switching to IPv6 over Connection-less BLE

Once the IPv6 connection is established and the devices can exchange IPv6 packets as shown in Figure 4.5, a thermostat may inform the router to switch to connection-less communication by issuing the `Disconnect` command from the application layer to the controller in order to disconnect the BLE link-layer connecton between the thermostat and the router.

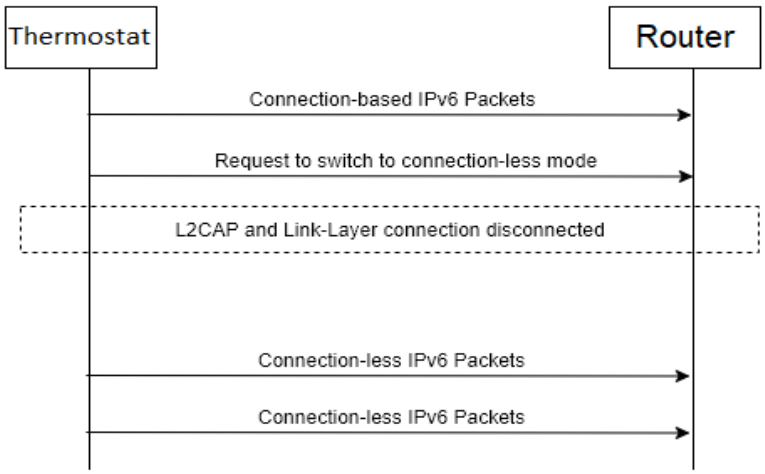

Figure 4.5: The steps followed to switch the IPv6 communication between a thermostat and a router from connection-based to connection-less.

After successfully receiving the command from the host, the controller sends the command status to the host and transmits LL_TERMINATE_IND PDU

on a LE connection to the router. The LL_TERMINATE_IND PDU contains one significant field named ErrorCode, that informs the router why the connection is being terminated. As per the BLE specification version 5, 0x33 ErrorCode is reserved for future use, therefore, we use this code to indicate a switch to connection-less mode. After receiving an acknowledgment from the router, the Disconnection complete event is generated on both the devices. The link-layer of the thermostat sends this event with "Connection Terminated by Local Host (0x16)" as a Reason event parameter to the host. On the other hand, the link layer of the router acknowledges host by Disconnection complete event with the reason for disconnection provided by LL_TERMINATE_IND PDU.

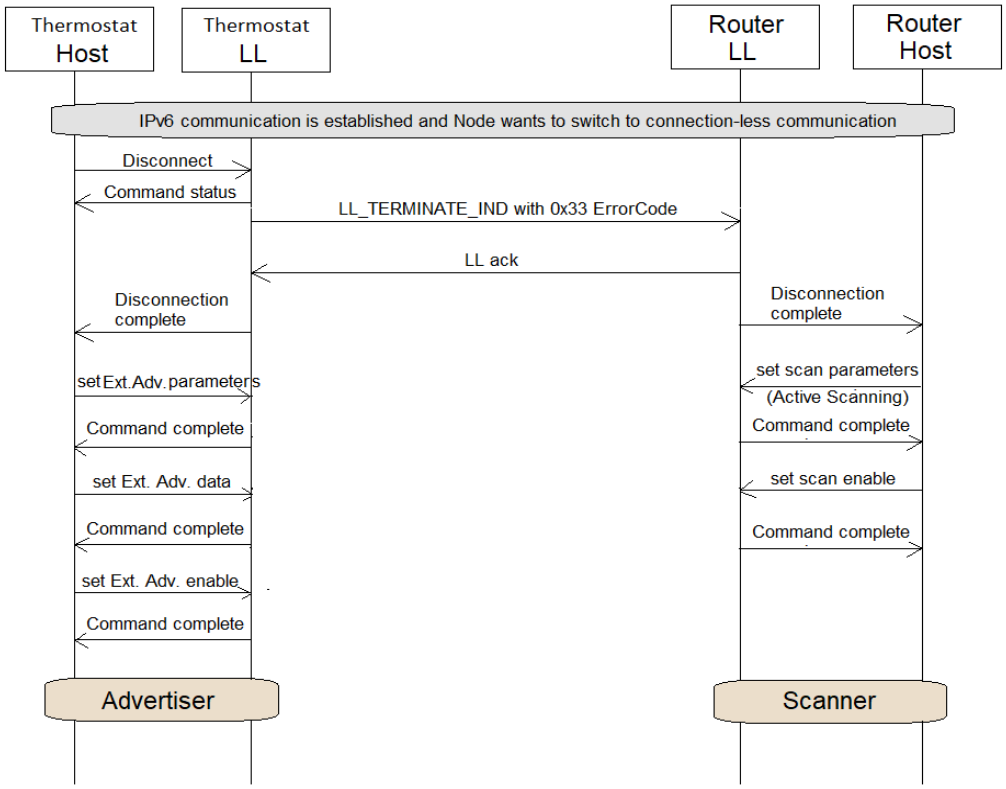

Figure 4.6: The handshaking procedure used to switch the IPv6 communication between thermostat and router to connection-less communication (adapted from [6]).

Consequently, both the devices break the link-Layer and L2CAP connection

between them while keeping the network interface up. The network interface is a link between network device drivers and the upper part of the network stack which ties them together [22]. It can be observed in Figure 4.2 between network protocols and L2 network technologies layers. During the network connection, all the data is sent and received via a network interface.

Furthermore, on the thermostat, since the network interface is up, the application data is encapsulated into the IPv6 packet and passed to the BLE_CONNLESS layer to transmit it over BLE link-layer technology. The BLE_CONNLESS layer plays a vital role in switching and handling both connection-based and connection-less IPv6 communication. Since the L2CAP connection is disconnected, as we discussed earlier, the BLE_CONNLESS layer uses the Ext. Adv. feature to transmit the IPv6 packet received from the upper layers to the router. On receiving IPv6 packet from the upper layers, BLE_CONNLESS layer sets the Ext. Adv. parameters on the controller using `set Ext.Adv.parameters` HCI command. Moreover, the received IPv6 packet is provided as user data by `set Ext.Adv.data` command to the controller, which later transmitted using Ext. Adv. over the secondary Adv. channels. The very next step is to enable Ext. Adv., which is done by using `set Ext.Adv.enable` command. As a result, thermostat acts as an Advertiser and starts advertising IPv6 packet over connection-less communication.

However, on the other end, after the termination of the link-layer and L2CAP connection, BLE_CONNLESS layer on the router issues `set scan parameter` command to the controller to configure the active scanning. On receiving command complete event from the controller, BLE_CONNLESS layer enables active scanning according to the parameters provided by `set scan parameter` command. Besides, the router acts as a Scanner and starts expecting IPv6 packets on advertising channels.

4.5.2.3 IPv6 over Connection-less BLE

As we have seen in Section 4.5.2.2, in IPv6 over connection-less BLE, the thermostat acts as an Advertiser and starts transmitting IPv6 packets on the Adv. channels using Extended Advertising, while the router acts as a Scanner and scans for those IPv6 packets on the Adv. channels. Figure 4.7 shows how the IPv6 packet is passed through the communication stack in order to

be transmitted on the Adv. channel in connection-less IPv6 communication. The IP and the 6LoWPAN layer perform the same operation on application data as in connection-based communication and encapsulate and compress the IPv6 packet with the 6LoWPAN header compression mechanism. The compressed IPv6 packet is transferred to the BLE_CONNLESS layer to perform the further operations. As we discussed earlier, the data needs to be passed to the controller from BLE_CONNLESS layer in order to be transmitted using Extended Adverting.

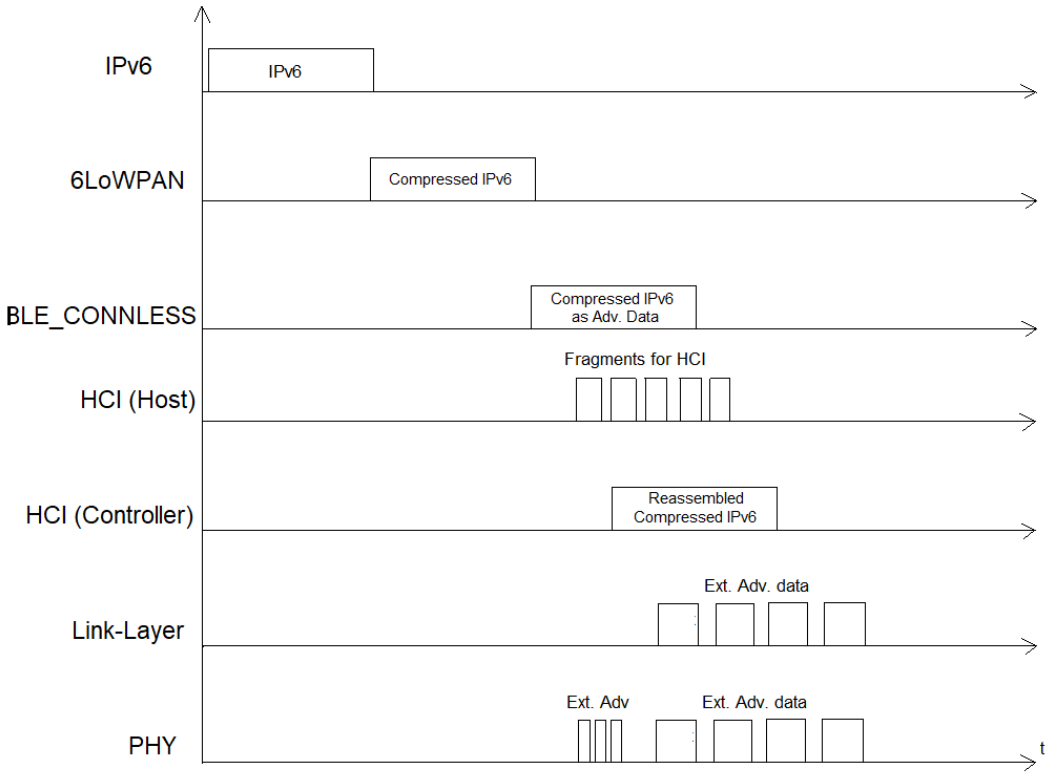

Figure 4.7: The data flow while sending the IPv6 packet through IPv6 over connection-less communication stack.

In BLE stack, the data is transferred from the host to the controller with the help of the HCI. The maximum data-carrying capacity of each HCI packet excluding the HCI header is limited to 255 bytes [**6**]. To make the IPv6 over connection-less communication stack IPv6 compliant, the host should pass

Table 4.1: Fragmentation of data performed in HCI_LE_Set_Extended_Advertising_Data command [6].

Value	*ParameterDescription*
0x00	Intermediate fragment of fragmented extended advertising data
0x01	First fragment of fragmented extended advertising data
0x02	Last fragment of fragmented extended advertising data
0x03	Complete extended advertising data

at least 1280 bytes of IPv6 packet to the controller. This can be achieved by fragmenting the complete IPv6 packet into 255 bytes of small fragments. The `set Ext.Adv.data` HCI command is used by the BLE_CONNLESS layer to pass the data to the controller using HCI packets with respective fragment sequence values. This fragment sequence can be observed in Table 4.1. The controller, on the other end, receives these packets and reassembles the complete IPv6 packet according to their fragment sequence values.

Furthermore, since the AUX_ADV_IND and AUX_CHAIN_IND PDUs can carry up to 254 bytes of user data in its payload (as discussed in Section 2.1.3.2), the fragmentation needs to be performed over IPv6 packet to pass the complete IPv6 packet using Ext. Adv. to the router . The data-carrying capacity of each packet varies depending on the extended header field of AUX_ADV_IND and AUX_CHAIN_IND PDU. As discussed in Section 2.1.3.2, the fragmentation of the IPv6 packet is taken care of by the Ext. Adv. feature at the link layer. In Ext. Adv., the IPv6 packet is treated as user data and transmitted using AUX_ADV_IND and AUX_CHAIN_IND PDU over the secondary Adv. channels. The AUX_ADV_IND and AUX_CHAIN_IND PDUs contain the AuxPtr field that points to the next AUX_CHAIN_IND PDU carrying the next fragment of the IPv6 packet. Since the last AUX_CHAIN_IND PDU does not contain the AuxPtr field as no more packet is transmitted, it helps the router to recognize the last fragment or AUX_CHAIN_IND PDU to perform reassembly of a complete IPv6 packet.

However, since the Extended Scanning is enabled on the router, it scans for the Ext. Adv. packets on primary Adv. channels. On receiving Ext. Adv. PDU on the primary Adv. channel, the scanner scans for the AUX_ADV_IND and

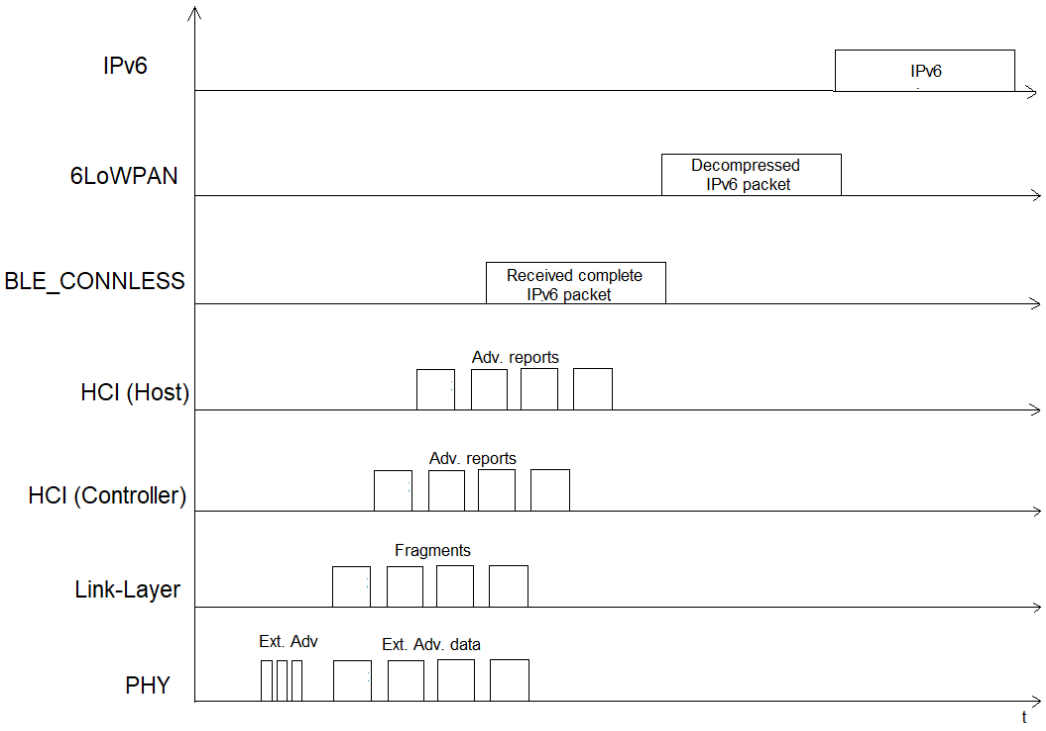

Figure 4.8: The data flow of received IPv6 packet in IPv6 over connection-less communication stack.

AUX_CHAIN_IND PDUs on a secondary Adv. channel pointed by the Ext. Adv. PDU in order to reassemble the complete IPv6 packet. These received AUX_ADV_IND and AUX_CHAIN_IND PDUs packets are passed to the host part of the HCI interface with the help of Adv. reports. The host part of the HCI receives these Adv. reports and passess to BLE_CONNLESS layer. BLE_CONNLESS layer reassembles these packets into one complete IPv6 packet and passes it to the upper layer to decompressed the IPv6 header. Once the decompressed IPv6 packet is ready, it is transferred to the upper layers to pass the actual user data to the application layer.

Channel Management

As we have discussed in Section 3.1, in connection-based communication, both devices establish a connection and start exchanging data packets over

the data channels (0-36), while in connection-less communication, the user data is exchanged over the Adv. channels. As discussed in Section 2.1.3.2, the Ext. Adv. uses secondary Adv. channels (0-36) to transmit the large packets over the connection-less communication, therefore it is required to perform the channel selection for the packets transmitted on the secondary Adv. channels. The used channel of Ext. Adv. can be specified using the AuxPtr field of the extended header of ADV_EXT_ IND, AUX_ADV_IND, and AUX_CHAIN_IND PDUs (see Figure 2.14). This field indicates the used secondary Adv. channel of the upcoming PDU of the same Extended Adv. event. To avoid the interference on the secondary Adv. channels, random channel selection is performed periodically at the start of each Ext. Adv. event, it means packets from the different Ext. Adv. events are transmitted on different secondary Adv. channels while all the packets from same Ext. Adv. event are transmitted on the same channel.

Moreover, the number of primary Adv. channels used for transmitting Extended Advertising PDUs can be controlled by the Primary_Advertising_Channel_Map parameter of `set Ext.Adv.parameter` command mentioned in table 5.1. This parameter allows an Advertiser to transmit Ext. Adv. PDU on only one primary Adv. channel instead of all three channels, which limits the number of packets transmitted on the primary Adv. channels. It may help to reduce the energy consumption of Advertiser even further as the less number of packets are transmitted.

Handling of New Application Data:

As the data from the application layer may change over time, it must be handled in the IPv6 over connection-less communication stack. BLE_CONNLESS layer starts the Ext. Adv. with a new IPv6 packet for only one Ext. Adv. interval. The interval between two IPv6 packets is entirely handled by the application layer which is suitable for the non-periodic applications.

4.5.2.4 Switching to IPv6 over Connection-based BLE

If the thermostat is willing to switch to connection-based mode (e.g., to update IPv6 prefixes or check for updates), the host issues the `set`

`Ext.Adv.enable` command with disable parameter to stop Ext. Adv. in case of Normal duty cycle applications. To establish the link-layer and the L2CAP connection between both the thermostat and the router, a thermostat starts a legacy advertisement with connectable Adv. PDUs as discussed in Section 4.5.2.1. A router sends the connection request if the Advertising packet received from a thermostat contains IP supported service UUID and establishes a link-layer connection between these two devices. Once the link-layer connection is established, a router initiates the L2CAP connection and receives an acknowledgement from the thermostat on the successful creation of a Logical channel.

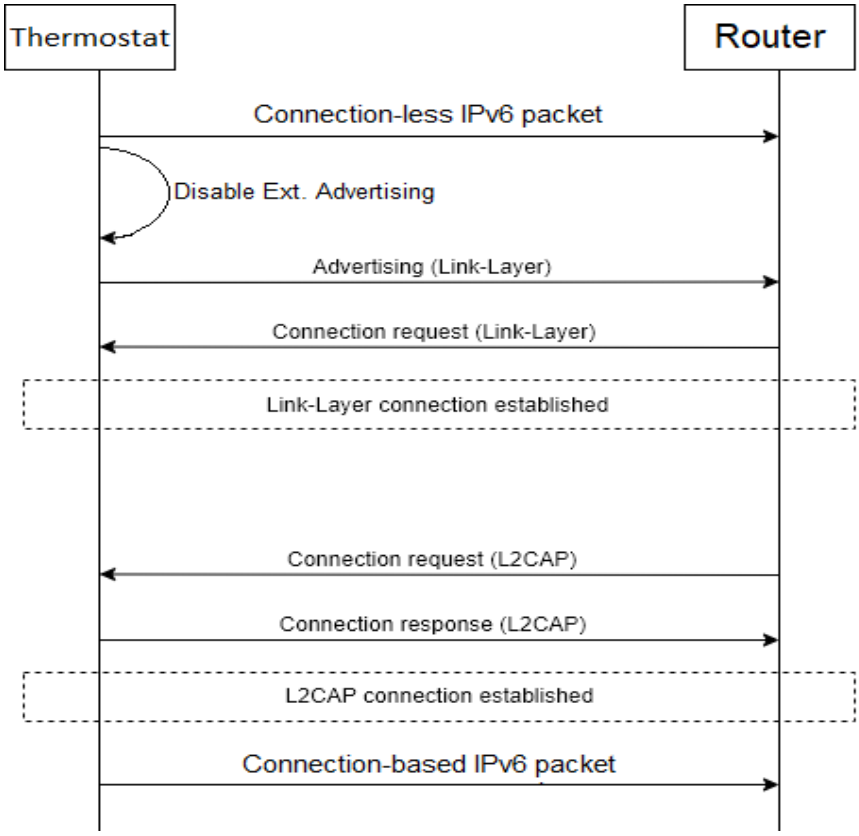

Figure 4.9: The steps followed to switch the connection-less IPv6 communication to connection-based communication. Initially, Extended Advertising is disabled. Secondly the BLE link-layer connection is established and later, a BLE L2CAP connection is created (adapted from [18]).

Moreover, the address used on the thermostat can be a random static address or the private address. The random static address is either burned into the device during the manufacturing process or generated on every power cycle. On the other hand, the devices that want to remain private use a private address that changes periodically. In case of a random static address, since the address doesn't change during runtime, the network interface is kept up while switching the IPv6 communication from connection-based to connection-less. As a result, the network data required to transmit the IPv6 packet is retained. Consequently, the thermostat can start exchanging IPv6 packets with a router over the L2CAP channel. Besides, if the thermostat uses a private address or device wants to perform neighbour discovery to update the network prefix, context information and configuration, the network interface is brought down while switching the communication from connection-less to connection-based. The thermostat and the router follow the same steps as explained in Section 4.5.2.1 to establish IPv6 connection-based communication.

5 Implementation

This chapter presents the implementation of the IPv6 over connection-less BLE communication stack in the Zephyr OS. Section 5.1.1 discusses the changes performed in the existing Zephyr OS BLE stack to support the Ext. Adv. feature. Section 5.1.2 describes the implementation of a new BLE_CONNLESS layer in the Zephyr OS network stack to support the connection-less IPv6 communication over BLE. Section 5.1.3 gives an overview of the challenges faced during the implementation.

5.1 Communication Stack

The implementation is based on version 1.13.0 of Zephyr OS (https://docs.zephyrproject.org/1.13.0/). It is an open-source operating system for resource-constrained IoT devices provided by the Zephyr Project's group. The source code is open source under the 3-clause BSD license and can be found at (https://github.com/IPv6-over-connectionless-BLE/zephyr).

5.1.1 Extended Advertising

The implementation of Ext. Adv. is divided into two parts: Advertiser and Scanner. First, we will discuss the Advertiser followed by the Scanner.

5.1.1.1 Advertiser

The Extended Advertising feature has been implemented on a Host as well as a Controller part of the Advertiser.

Host: The Host component of the stack is implemented in `../subsys/bluetooth/host/hci_core.c`.
All the commands required to activate the Extended Advertising namely:

- *HCI_LE_Set_Extended_Advertising_Parameters*,
- *HCI_LE_Set_Extended_Advertising_Data*,
- *HCI_LE_Set_ Extended_Advertising_Enable*

are implemented in the Host with the `bt_le_ext_adv_start`, `set_ext_ad`, `set_ext_advertise_enable` functions respectively.

Table 5.1: Parameters set by the *HCI_LE_Set_Extended_Advertising_Parameters* command [6].

Parameter-name	*Functionality*
Advertising_Event_Properties,	Legacy/ Extended Advertising event
Primary_Advertising_Interval_Min,	Minimum interval time of an event
Primary_Advertising_Interval_Max,	Maximum interval time of an event
Primary_Advertising_Channel_Map,	Selection of primary advertising channels
Own_Address_Type,	Public/ private own device address
Peer_Address_Type,	Public/ private target device address
Peer_Address,	Target device address
Advertising_Tx_Power,	Selection of radio power
Primary_Advertising_PHY,	LE 1M Primary adv PHY
Secondary_Advertising_Max_Skip,	No. of Adv event skip before AUX_ADV_IND packet is sent
Secondary_Advertising_PHY,	LE 1M Secondary advertisement PHY
Advertising_SID,	Value of the Advertising SID subfield in the ADI field of the PDU
Scan_Request_Notification_Enable,	Not supported.

As defined in the Host Controller Interface, commands send by the host are received by the controller. The very first command sent by the host is *HCI_LE_Set_Extended_Advertising_Parameters* and that is handled by `bt_le_ext_adv_start` function of the `hci_core.c` file. It sets all the parameters required for configuring Extended Advertising and the parameters set by this command are described in Table 5.1. The command *HCI_LE_Set_Extended_Advertising_Data* performs a significant role by allowing the host to transmit a large amount of data to the controller. As we

discussed in Section 4.5.2.3, the HCI packet used to send a command to the controller can carry up to 255 bytes of data excluding the HCI command packet header. It means that the maximum amount of application data that can be sent to the controller using an HCI packet is limited to 255 bytes. In the case of 1280 bytes of the IPv6 packet, the complete packet must be sent to the controller. Therefore, the fragmentation of the complete packet requires to be performed and hence handled in `set_ext_ad` function. The larger packet is fragmented into small chunks of data and each fragment is sent to the controller using *HCI_LE_Set_Extended_Advertising_Data* command. The command is sent with the operation parameter mentioned in Table 5.2 that informs the controller about the fragment sequence. If the data is larger than 255 bytes, the *HCI_LE_Set_Extended_Advertising_Data* command is sent to the controller several times with a suitable operation parameter until the whole data is sent by the host.

Controller: The controller component of the stack is implemented in `../subsys/bluetooth/controller/ll_sw/ll_adv.c` and `../subsys/bluetooth/controller/ll_sw/ctrl.c`. These files contain all the implementation related to packet encapsulation, time synchronization between packet on the primary-secondary Adv. channel, event skip and Adv. events.

The commands received from a host are executed on a controller and the status of the commands is sent to a host using HCI events. The Ext. Adv. in the controller is handled by the ticker timer. The ticker implementation is specific to BLE controller scheduling and it is nothing but a soft real time radio/resource scheduling. It basically schedules events to access the radio. The controller uses two ticker timer: one for Adv. events and another for Ext. Adv. events. The timer used for Adv. events works in a periodic manner and calls *radio_event_adv_event* function (which loads the information about AUX_ADV_IND PDU in AuxPtr field of an ADV_EXT_IND PDU) with a fixed interval provided by the host. Packets broadcasted by the radio with the help of this event are sent on the primary Adv. channel. Similarly, another timer also works in a periodic manner, but the interval between two events is dependent on the max_skip parameter provided by host with *HCI_LE_Set_Extended_Advertising_Parameters* command. It defines the number of Adv. events is skipped before transmitting AUX_ADV_IND PDU on sec-

Table 5.2: Parameters set by the *HCI_LE_Set_Extended_Advertising_Data* command [6].

Parameter-name	*Functionality*
Operation,	Handles fragmentation of data
Fragment_Preference,	Should controller perform fragmentation
Advertising_Data_Length,	The number of octets in the Advertising Data
Advertising_Data,	Advertising data

ondary Adv. channels. The AUX_ADV_IND and AUX_CHAIN_IND PDUs are transmitted in Ext. Adv. events. If the user data provided by the host using *HCI_LE_Set_Extended_Advertising_Data* is more than one AUX_ADV_IND PDU can hold, the next AUX_CHAIN_IND PDU is scheduled by the Controller. The duration between these two packets is set to 300us as a Minimum AUX Frame Space (T_MAFS), this allows the radio to go to Rx or sleep mode in between. Besides, the count of AUX_CHAIN_PDU is depended on the size of the user data. Since the maximum amount of data that can fix in one AUX packet with an Auxptr field of the extended header is around 246 bytes, fragmentation of the data is performed on a controller to send the complete user data in one Ext. Adv. event.

5.1.1.2 Scanner

Host: As described in Section 2.1.3.4, the commands used to start an Extended Scanning are implemented in the host component with `../subsys/bluetooth/host/hci_core.c`.
All the required commands:

- *HCI_LE_Set_Extended_Scan_Parameters*,
- *HCI_LE_Set_Extended_Scan_Enable*,
- *HCI_LE_Set_ Extended_Advertising_Enable*

are implemented with the `bt_le_scan_start` and `set_le_scan_enable` function respectively.

Table 5.3: Parameters set by the *HCI_LE_Set_Extended_Scan_Parameters* command [6].

Parameter-name	*Functionality*
Own_Address_Type,	Public/ Random device Address
Scanning_Filter_Policy,	Accepts all advertising packets
canning_PHYs,	Scan advertisements on the LE 1M PHY
Scan_Type,	Passive/ Active scanning
Scan_Interval,	Interval between start of the subsequent scan on the primary advertising channel.
Scan_Window,	Duration of the scan on the primary advertising channel.

Controller: The controller component and all the events are implemented in `../subsys/bluetooth/controller/ll_sw/ll_adv.c` and `../subsys/bluetooth/controller/ll_sw/ctrl.c`.

These files contain all the code related to Adv. report event, extraction of data from received packets and time synchronization between Ext. Adv. packets.

On receiving an advertising packet at the physical layer, the radio triggers an interrupt to the upper layer to examine the packet. If the PDU type of the received PDU is 0x07 (Ext. Adv. PDU), the function called by this interrupt extracts and stores the significant information such as channel index, Aux offset, offset unit from the AuxPtr field of the EXT_ADV_IND PDU into global variables. The saved information is used to grab the AUX_ADV_IND PDU on a secondary Adv. channel. The function of an Adv. report event encapsulates the received data with the HCI packet format in order to forward the data to the host. The RSSI value of the received packet is appended to the data before delivering it to the host. Since the Advertiser transmits multiple AUX packets such as AUX_ADV_IND and AUX_CHAIN_IND PDUs on secondary Adv. channels, the Scanner reassembles complete user data by collecting these PDUs from secondary Adv. channels.

5.1.2 BLE_CONNLESS Layer

The implementation of the BLE_CONNLESS layer that is used in this thesis can be found in the source file `../subsys /net /ip /l2/ ble_conless.c`. The existing implementation of the Bluetooth L2 layer has been used to establish an IPv6 connection between a node and a router. The BLE_CONNLESS L2 layer can operate in two different modes: node or router.

Node:

As explained in Section 4.5.2.1, once the IPv6 connection between a node and a router is established, a node device transmits IPv6 packets to the router over the L2CAP communication channel. To switch the communication from connection-based to connection-less, a network management procedure handler is implemented and it is registered using a NET_MGMT_REGISTER_REQUEST_HANDLER macro of Zephyr. The registered switch procedure handler is invoked through *net_mgmt()* API from the application layer at the time of switching. The switch handler initially checks if the device is connected to a router over an L2CAP communication channel. If both devices are connected, switch handler calls `bt_hci_disconnect` function from `../subsys/bluetooth/ host/conn.c` that sends `HCI_Disconnect` command with the custom 0x33 reason code as explained in Section 4.5.2.2 to the controller. On receiving connection disconnect reason 0x33, a router sends L2CAP_DisconnectReq command to the node and as a response receives L2CAP_DisconnectRsp. As a result, the L2CAP connection between these two devices is disconnected and subsequently, the Link layer connection is also disconnected. On termination of L2CAP and Link layer connection, a node receives the Disconnect Complete event from a controller and sets the state of the connection to disconnected.

The BLE_CONNLESS L2 layer is designed to not expose the lower link-layer and device drivers of Bluetooth to the higher IP stack. It is made possible using network interface structure that is declared in `..include/net/net_if.h`. The initialization of BLE_CONNLESS L2 layer is performed using NET_L2_INIT macro that registers net_bt_send, net_bt_recv L2 Layer handling functions with upper layers *send()*, *recv()* API respectively.

- **net_bt_send**: Since the network interface is up, the application layer sends the user data to the network layer to prepare the IPv6 packet and pass it to the lower layers. This function is called by the IP core stack to send a prepared IPv6 packet. The function calls *net_6lo_compress()* API provided by `..subsys/net/ip/6lo.h` to perform 6LoWPAN header compression. The resulting compressed IPv6 packet is sent to the Link-layer with the *bt_le_ext_adv_start()* API. It starts Ext. Adv. as explained in Section 5.1.1.1 and acts as an Advertiser. The function returns with either NET_OK if everything is as expected or NET_DROP if something wrong happens.
- **net_bt_recv**: In connection-less IPv6 communication a node doesn't receive any packet from a router. This function is only valid on a router side, hence will be explained in a router part.

Router:

On receiving a LL_TERMINATE_IND PDU with 0x33 error code from a node to switch the communication from connection-based to connection-less (as explained in Section 4.5.2.2), the disconnected() connection callback function is invoked. If the connection terminate error code is equal to our predefined 0x33, it disconnects the connection while keeping the network interface up. If the connection terminate reason is other than 0x33, the network interface is brought down.

Once the Link layer and a L2CAP connection between a node and a router is disconnected, a router starts active scanning by calling *bt_le_scan_start()* API from `../subsys/bluetooth/host/hci_core.c` with *device_found()* as a callback function. The *device_found()* callback function is called on receiving a complete Ext. Adv. packet from the Lower layers. The function checks if the network interface is up, to perform further operation or it drops the packet. The received IPv6 packet is inserted into struct net_pkt and put into the head of the network buffer using *net_pkt_frag_insert()*. This buffer is passed to the network stack via *net_recv_data()*. The network stack passes the buffer to the L2 layer's *net_bt_recv()* callback function which is registered with recv() function. The *net_bt_recv()* function calls *net_6lo_uncompress()* to uncompress the 6LoWPAN header before passing the complete IPv6 packet to the upper layer. The *net_bt_recv()* function returns NET_CONTINUE if

the packet is successfully uncompressed and the network stack should then handle it or NET_DROP in case of an incorrect or incomplete packet.

5.1.3 Implementation Challenges

The biggest challenge while implementing the IPv6 over connection-less BLE communication stack was that the controller part of the Zephyr BLE stack uses ticker timer to handle the Adv. events. As discussed earlier in Section 2.1.3.2, an Ext. Adv. feature implemented on an Advertiser works with two Adv. events, one Adv. event to transmit the packets on primary Adv. channels and another one is an Ext. Adv. event to transmit packets on secondary Adv. channels. In the existing implementation, the functionality of an Adv. event is achieved with a ticker timer that schedules the Adv. event periodically. The existing implementation of the ticker is complex and there is no reference documentation available publicly [**36**]. The ticker can only be used with a valid ticker id. The Advertising role offers two ticker ids, one is to start the Adv. event and another one to stop the Advertising role. To start the Ext. Adv. event and provide synchronization with the Adv. event, it is required to use another ticker timer. Since the ticker ids allocated for the Advertising role are already used, it was a challenge to use another ticker timer to start the Extended Adv. event. The use of existing ticker ids is done in such a way that the ticker id required to stop the Advertising role is also used for Ext. Adv. events and the synchronization between both Adv. and Ext. Adv. events is achieved.

6 Evaluation

To evaluate the performance of the IPv6 over connection-less BLE implemented in this thesis, we start by measuring the power consumption of a node device using different communication parameters and compare it with the IPv6 over connection-based communication in Section 6.1 and 6.2. In Section 6.3, we examine the reliability of IPv6 over connection-less BLE over different distances. Section 6.4 evaluates the switching between connection-less and connection-based communication modes.

6.1 Comparison of Connection-based vs Connection-less IPv6 over BLE

First of all, we will have a look at the comparison of IPv6 over connection-based vs IPv6 over connection-less BLE communication. In connection-based communication, the router and the node device establish a standard BLE connection and exchange all the data required to transmit IPv6 packets. Furthermore, all IPv6 packets are transmitted over the L2CAP channel according to RFC 7668 [**18**]. In connection-based mode, the BLE connection is never terminated.

Instead, in IPv6 over connection-less communication, the IPv6 packets with actual application data are transmitted over the connection-less communication using Extended Advertising as proposed in this thesis. Since devices do not exchange keep-alive packets in connection-less communication, the node device transmits fewer packets compared to connection-based BLE communication. In this section, we will compare the power consumption of a node device using either IPv6 over connection-based or IPv6 over connection-less communication. For this experiment, we assume the router

to be wall-powered, i.e. the router does not have a constrained power supply.

6.1.1 Experimental Setup

To evaluate IPv6 over connection-less BLE communication, we use two nRF52 development kits from Nordic Semiconductor, one as a node and one as a router. The power consumption of a node device is measured using the Low Voltage Power Monitor FTA22D from Monsoon Solutions Inc.[**15**] as a power monitoring device.

The application used on the node devices is a modified version of the echo-client sample application of Zephyr (zephyr/samples/net/echo_client.c), that was adopted to transmit UDP packets unidirectionally. On the router, the modified version of the echo-server sample (zephyr/samples/net/echo_server.c) is used to achieve communication between node and router. To support the 1280 bytes of IPv6 packet length, the CONFIG_NET_BUF_DATA_SIZE parameter is set to 1280 in the application configuration file. Without this change, the maximum IPv6 packet size of 1280 bytes would not have been achieved in both the communication modes. Since the switching between the IPv6 over connection-less BLE and IPv6 over connection-based BLE is implemented in (Zephyr/subsys/net/ip/l2/ble_connless.c), the CONFIG_NET_L2_BT_CONNLESS flag is set in the configuration file to include it during the compilation.

In connection-based BLE communication, we use the most energy-efficient settings that are still valid according to BLE 5 specifications. The **Supervision_Timeout** is set to 32 sec, which is the maximum achievable connection timeout [**6**]. However, we set the **Slave_Latency** to 3 and the **Connection_Interval** parameter to 3.99 sec (if it is set to its maximum limit of 4 sec, it does not fullfill the requirement of having **Supervision_Timeout** greater than ((1 + Slave_Latency) * Connection_Interval * 2) as mentioned in [**6**]).

In connection-less communication, the **Primary_Advertising_Interval** parameter value is set to 100 ms. In this case, the Ext. Adv. is activated for only one Adv. event of 100 ms, i.e the IPv6 data is transmitted only once.

This allows the application to transmit non-periodic IPv6 packets. The interval between two IPv6 packets is denoted as **application interval**. The **Primary_Advertising_Channel_Map** (see Table 5.1) parameter is set to channel 37 to achieve higher energy efficiency as with this advertiser uses only one primary Adv. channel (channel 37) for advertising. The scanner on the other end is configured for active scanning with active as a **Scan_Type** (see Table 5.3) and 30 ms as a **Scan_Interval** and **Scan_Window**, i.e. continuous scanning. All the three primary Adv. channels (37, 38 and 39) are used for scanning.

6.1.2 Result

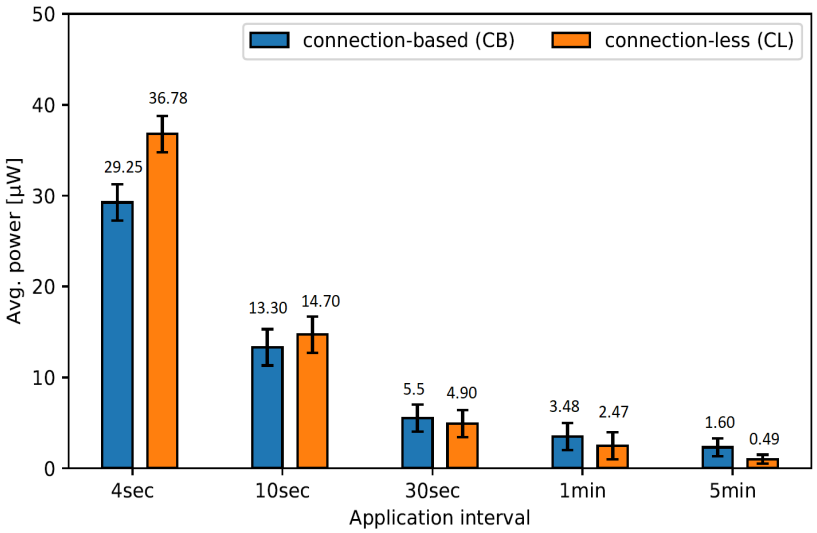

Figure 6.1: Average power consumption of the BLE radio of a node device (excluding the base power consumption of the device) when sending 100 bytes of IPv6 packets with different application intervals over connection-based and connection-less BLE communication.

Initially, we measured the base power consumption of the nRF52 board with the Zephyr OS in deep-sleep mode (all the peripheral on board are disabled) by using the sample program + (../zephyr/samples/boards/nrf52/power_mgr). The average power consumption with deep-sleep mode we observed during the measurement was

781 uW. The issues [32] and [33] on the git-hub repository of Zephyr OS indicate that at the time of Zephyr 1.13.0 released, there was no comprehensive power management solution in Zephyr OS which causes the higher base power consumption for the nRF52 board. Additionally, in [32], the user claims that with nRF52x boards burned into Nordic SDK draw 2uA current in an idle state while with Zephyr, the current is always about 290 uA. Therefore, we subtract this base power consumption of the nRF52 board from the measurements and focus on the power consumption of the BLE radio through the remainder of this thesis.

Figure 6.1 shows the average power consumption of the BLE radio of a node device in both connection-based and connection-less IPv6 communication when sending 100 bytes of IPv6 packet with different application intervals. The power measurement was started on the 1st IPv6 packet was sent and stopped after sending the 100th IPv6 packet.

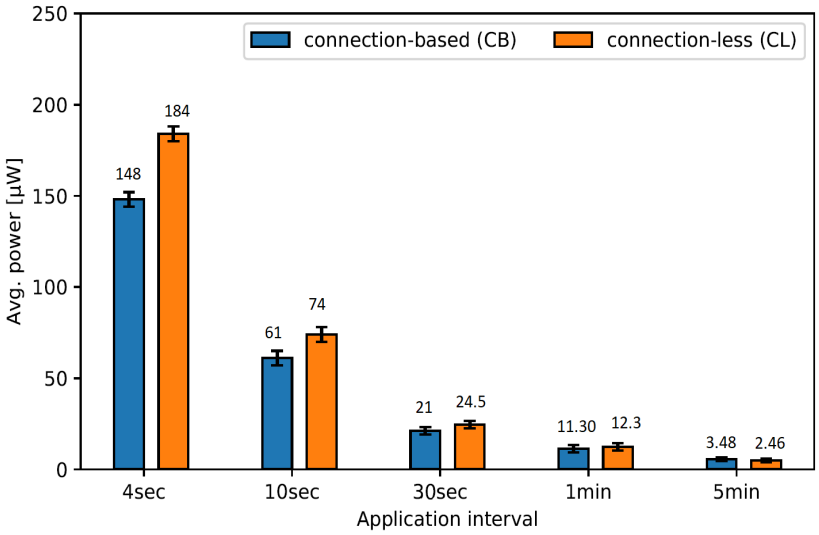

Figure 6.2: Average power consumption of the BLE radio of a node device (excluding the base power consumption of the device) when sending 500 bytes of IPv6 packets with different application intervals over connection-based and connection-less BLE communication.

Figure 6.1 shows that the average power consumption of active radio of high duty-cycle applications (e.g., 4-sec application interval) in connection-less

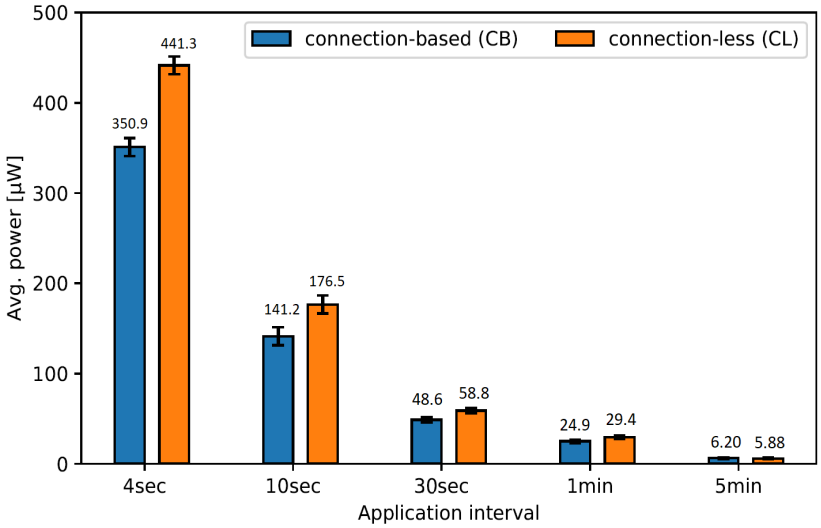

Figure 6.3: Average power consumption of the BLE radio of a node device (excluding the base power consumption of the device) when sending 1200 bytes of IPv6 packets with different application intervals over connection-based and connection-less BLE communication.

communication is around 20 % higher than the connection-based communication. Moreover, Figure 6.2 and 6.3 also show the same behaviour of a node device with different packet size.

The reason for the higher power consumption in high-duty cycle applications could be that in connection-less communication a node device needs to transmit extra packets on primary Adv. channels to point to the AUX_ADV_IND PDU for each IPv6 packet (see Section 2.1.3.2). As we were expecting, for low duty cycle applications the power consumption of connection-less is lower than the connection-based communication.

Moreover, from Figure 6.1, 6.2 and 6.3 we can observe that the average power consumption of active radio of low duty-cycle applications (e.g., 5-min application interval) is around 70 % higher in connection-based BLE compared with connection-less BLE communication. The interval between two IPv6 packets has an effect on a power consumption of active radio of a node device in connection-based BLE. As predicted by our models in Section 3.3, the reason for higher power consumption is that in connection-based

Node	Radio Active (µs)	References
ADV_EXT_IND	184	Eq. 3.13
AUX_ADV_IND	936	Eq. 3.15
Total Radio-on time of node = 1120		

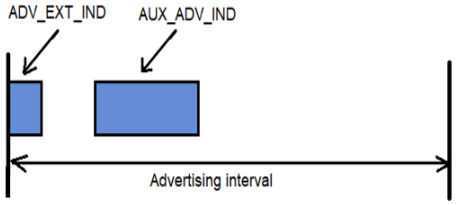

Figure 6.4: Radio-on time of a node device to transmit 100 bytes of IPv6 packet in connection-less communication (adapted from [38]).

Master Tx	Radio Active (µs)	Slave Tx	References
Empty Packet	80		Eq. 3.8
	912	Data channel PDU	Eq. 3.10
Empty Packet	80		Eq. 3.8
Empty Packet	80		Eq. 3.8
Empty Packet	80		Eq. 3.8
Empty Packet	80		Eq. 3.8
	80	Empty Packet	Eq. 3.8
Total Radio-on time of slave = 1152			

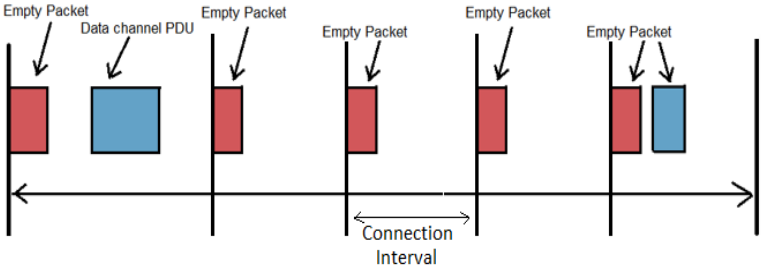

Figure 6.5: Radio on time of a router and node device to transmit 100 bytes of IPv6 packet in connection-based communication for 5 connection intervals with slave latency 3 (adapted from [38]).

BLE the low duty-cycle node device exchanges more number of keep-alive packets with router to maintain a BLE connection.

The active radio time of ADV_EXT_IND PDU from Figure 6.4 is calculated using equation 3.12 while AUX_ADV_IND PDU is calculated using equation 3.13 and 3.14 respectively. However, the active radio time of Data channel PDU from Figure 6.5 is calculated using equation 3.9 and 3.10 while the Empty packet is calculated using equation 3.8.

The theoretical analysis from Section 3.3 shows that the radio-on time to transmit 100 bytes of IPv6 packet in connection-less BLE is 1120 us (calculated using 3.11) for a 5 min of application interval, while in connection-based BLE it is 3952 us (calculated using 3.1). It means, the radio-on time in connection-based BLE is 71 % more than connection-less BLE while sending 100 bytes of IPv6 packet every 5 mins. Besides, the experimental data from Figure 6.1 shows that the average power consumption of active radio is 0.49 uW for connection-less BLE while 1.60 uW for connection-based BLE when sending 100 bytes of IPv6 packet every 5 mins. In this case, connection-based BLE required 69 % more power than the connection-less BLE. Therefore, we can say that the theoretical analysis and experimental data confirm the same behaviour of a node device for the low duty-cycle application.

Furthermore, Figure 6.1, 6.2 and 6.3 shows that the size of the IPv6 payload contributes to the power consumption of a BLE radio. The longer the IPv6 packet the higher the power consumption of a BLE radio.

6.2 Connection-less Communication vs. Re-establishing a Connection

As we discussed, Section 1.1 indicates the possibility to re-establish a standardized BLE connection for each IPv6 packet to minimize the energy consumed by the keep-alive packets during connection-based communication. We compare the time and the energy required to transmit an IPv6 packet in connection-less communication with re-establishing a BLE connection approach.

6.2.1 Experimental Setup

The experiment setup used to compare the connection-less communication vs re-establishing a connection approach is similar to Section 6.1. The Monsoon power measurement tool is used to measure the power required to establish a BLE connection and transmit the IPv6 packet, while the time is measured with the help of kernel clock APIs provided by the Zephyr OS [31]. In connection-based communication, the node device initially transmits connectable Adv. PDUs and after establishing a connection, it transmits the actual IPv6 packet. Hence, the time is measured from the 1st connectable Adv. PDU to the end of actual IPv6 packet transmission. On the other hand, in connection-less communication, time measurement is started when the application issued data to lower layers and stopped at the end of the successful IPv6 packet transmission. Initially, we transmitted 10 IPv6 packets with packet sequence numbers from a node to the router. The number of packet transmissions requied to receive the first successful IPv6 packet on a router is measured using the packet sequence number of the received packet. As we measured the number of packet transmissions required for successful packet transmission, we measured the time and energy required to perform the same number of transmissions on a node device. This experiment measures the time and energy required for the successful transmission of 100 bytes of an IPv6 packet (taking reliability into account) from the node to the router.

During the experiment, the **Connection_Interval** parameter is set to 7.5 ms to achieve a fast data transfer. The **slave_latency** is set to 0, that means slave skips no connection events, while the **Supervision_Timeout** is set to 100 ms. The **advertising interval** of a node device is varied from 100 ms to 500 ms (see Figure 6.6 and 6.7) in both the communication modes. The router, on the other hand, uses 30 ms as a **Scanning_Interval** and **Scanning_Window** and scans on all primary Adv. channels in scanning mode. The experiment is performed with two different **Primary_Advertising_channel_map** parameter values for connection-less BLE. One with channel 37 (CL 1) and another with all three primary Adv. channels (CL 3).

6.2.2 Result

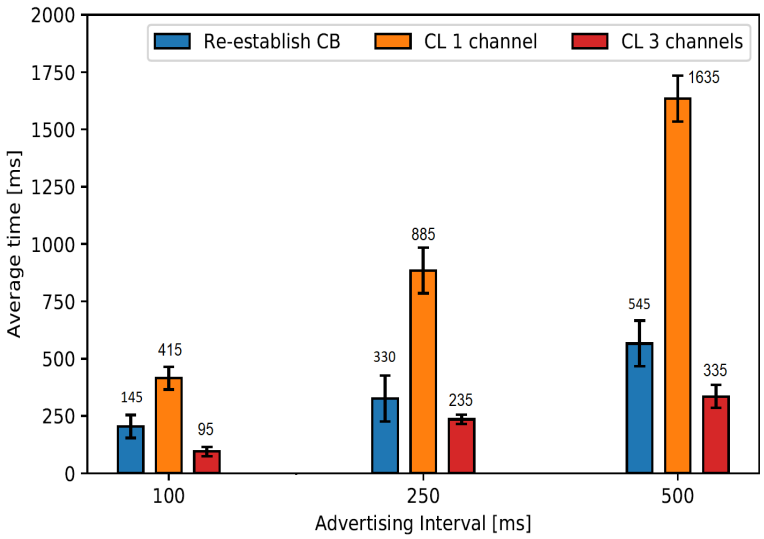

Figure 6.6: Time required to transmit 100 bytes of IPv6 packet with connection-less communication and re-establishing a connection approach from node to router.

Figure 6.6 shows the time required to transmit an IPv6 packet using different advertising intervals, over connection-less BLE and re-establishing a connection approach. According to the experiment, the time required to transmit an IPv6 packet by reconnecting approach is higher than the connection-less BLE. The reason behind the higher time could be an advertising interval of

the node and the scanning interval of the router. The higher the advertising interval, the longer the re-connecting time of both the devices as well as devices also perform neighbor discovery before exchanging actual IPv6 packets. In connection-less BLE, the time required to transmit the same size of the IPv6 packet with CL 1 is higher compared with CL 3 configuration. It is due to the more number of IPv6 packets are required to send for a successful reception at the router with CL 1 configuration.

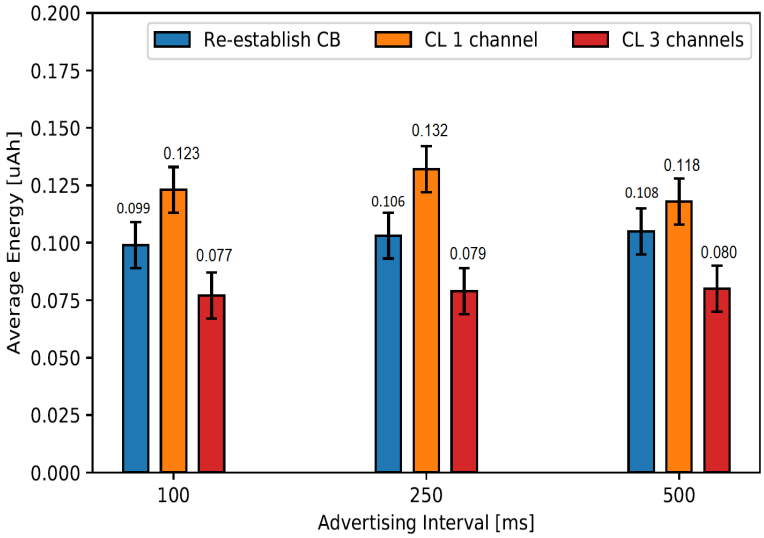

Figure 6.7: Energy consumption of a node device to transmit 100 bytes of IPv6 packet with connection-less and re-establishing a connection approach.

Figure 6.7 shows the energy required for BLE radio to transmit 100 bytes of an IPv6 packet using connection-less BLE and re-establishing a connection approach. In re-establishing a connection approach, the energy of BLE radio is measured for the time required to re-establish a standard BLE connection and transmit an actual IPv6 packet, while in connection-less communication, it is measured from the first transmission to the successful reception of the packet at the router. As we discussed in Section 6.1, the base power consumption of a device is significantly high, we subtracted the base energy consumption of a device from measurements.

We can observe from Figure 6.7, the energy consumption of a radio of a node device to transmit the same size of an IPv6 packet using connection-less BLE

with CL 3 configuration is lower than the re-establishing a connection approach. The reason behind the higher energy consumption of re-establishing a connection approach is that the devices transmit multiple connectable Adv. PDUs and exchange the number of packets for re-establishing a standard BLE connection before exchanging an actual IPv6 packet. However, in connection-less BLE, it depends on the number of packets are sent for a successful reception on a router. Moreover, the energy required to transmit an IPv6 packet using CL 1 configuration is higher than the CL 3 configuration due to the more number of packet transmission.

6.3 Reliability

This section compares the packet reception rate of the IPv6 over connection-less BLE communication stack to the existing IPv6 over connection-based communication stack in an office environment.

6.3.1 Experimental Setup

This experiment uses the same node and router setup as the power measurement setup used in Section 6.1. To measure reliability, 100 IPv6 packets are transmitted from a node device to the router over the distance of 1 meter and 10 meters. In connection-less communication, the packet reception rate of IPv6 packets is measured with two different **Primary_Advertising_channel_map** parameter values (CL 1 channel and CL 3 channel). The router scans on all the primary Adv. channels while the **Scan_Interval** and **Scan_Window** parameters of the router are set to 30 ms.

6.3.2 Result

The Figure 6.8 shows the packet reception rate of the connection-based communication and connection-less communication over the distance of 1 meter. According to the experiment, the IPv6 over connection-based communication stack has 100% of the packet reception rate, while connection-less

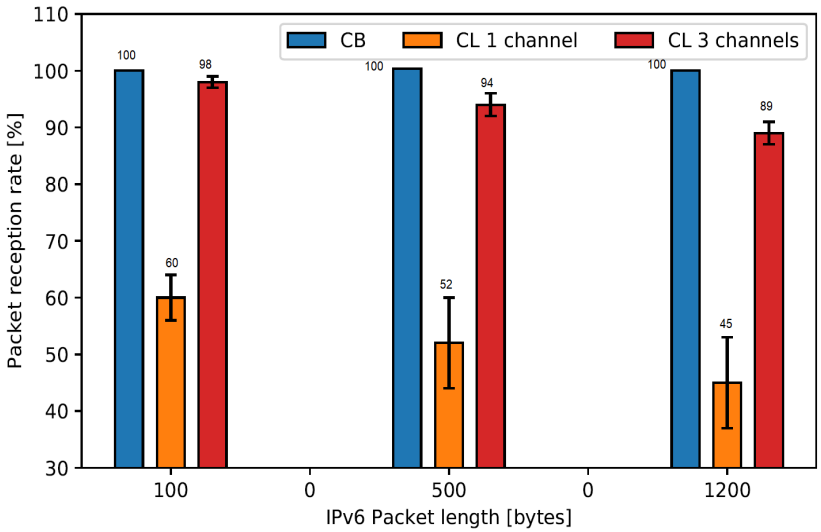

Figure 6.8: Packet reception rate for 1 meter distance between node and router with different size of IPv6 packets.

BLE is less reliable. This was expected since connection-based BLE uses autonomous re-transmissions and Adaptive Frequency Hopping.

As the distance between the node and the router increases, the strength of the advertised packet signal decreases and hence, due to the interference from other co-located wireless technology signals sharing the same 2.4 GHz ISM band, more packets get dropped. This can be seen in Figure 6.9.

Moreover, Figure 6.8 and 6.9 show that the packet reception rate of connection-less with CL 1 configuration is lower than CL 3 configuration and connection-based communication. However, the reliability of a CL 1 configuration depends on which channel the scanner is scanning on during the advertisement of an advertiser and hence, there is uncertainty. Therefore, the reliability varies from around 30 % to 60 % as the advertiser is advertising on a single primary Adv. channel. This experiment also shows that the IPv6 over connection-based BLE stack provides better reliability than the IPv6 over connection-less communication.

From Figure 6.9 and 6.8 we can see that the reliability of connection-less BLE with CL 1 configuration is 55 % when transmitting 100 bytes of IPv6 packets

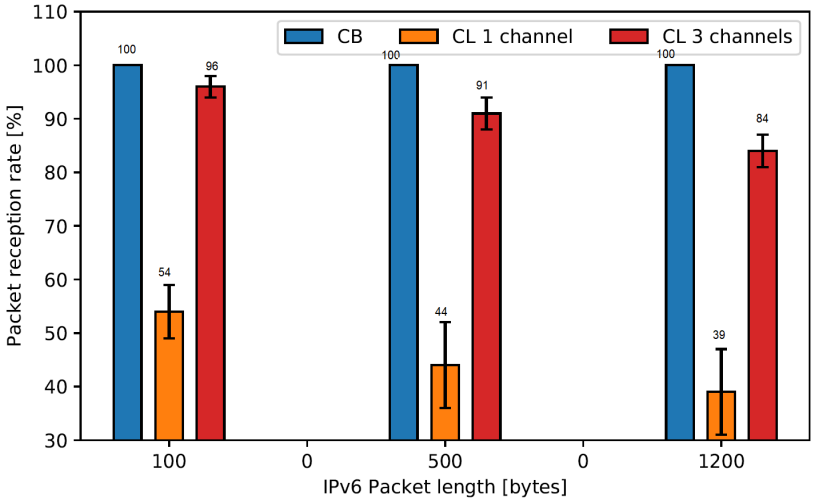

Figure 6.9: Packet reception rate for 10 meter distance between node and router with different size of IPv6 packets.

when transmitting 100 bytes of IPv6 packets, while with CL 3 configuration it is 96 %.

Although the motivation of this thesis is to minimize the energy consumption of low duty-cycle devices using connection-less BLE, it is less reliable compared to connection-based BLE as discussed above. Therefore, it is essential to provide high reliability also using connection-less BLE while keeping the power consumption low. The theoretical analysis from Section 3.3 and Table 3.2 show that if the packet transmission is performed thrice, a 90 % of reliability can be achieved using CL 1 configuration. While almost 96 % reliability can be achieved by transmitting the same IPv6 packet using CL 3 configuration in connection-less BLE. Therefore, we experiment to measure the average power consumed by the BLE radio of a node to achieve 90 % reliability in connection-based and connection-less BLE.

The experimental setup used to measure the average power consumed by a BLE radio of a node to achieve 90 % of reliability is similar to the experimental setup used in Section 6.1. In this experiment, we perform 3 packet transmissions when using CL 1 configuration and single packet transmissions when using CL 3 configuration in each application interval

to achieve 90 % of reliability (see Table 3.2). The power measurement was started on the 1st IPv6 packet was sent and stopped after sending the 100th IPv6 packet.

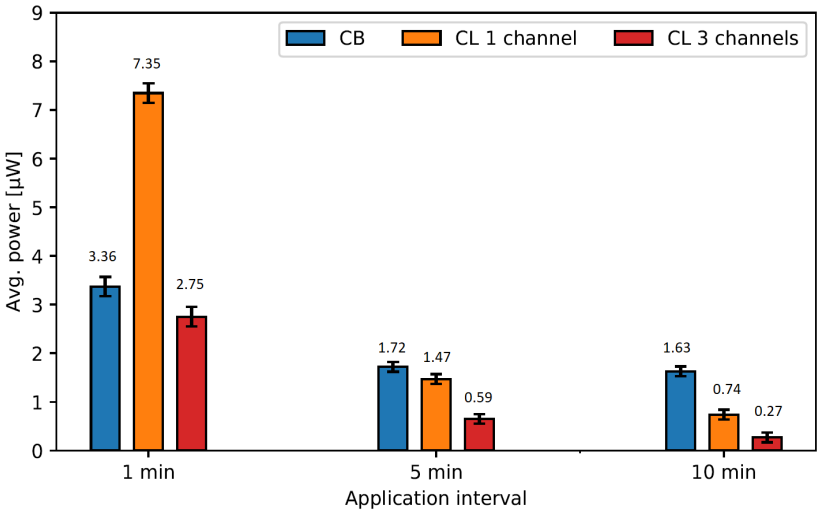

Figure 6.10: The energy consumption of the BLE radio of a node device in connection-based and connection-less BLE to achieve 90% reliability.

Figure 6.10 shows that the average power consumed by BLE radio to transmit an IPv6 packet in connection-based BLE is application interval dependent. Longer the interval the higher the power it requires. In connection-less BLE, the average power consumption of BLE radio depends on the number of packets are transmitted. From Figure 6.10 we can observe that connection-less BLE with CL 1 configuration consumed 13 % less average radio power while providing 90 % reliability. However, the average power consumed by a BLE radio in connection-less BLE with CL 3 configuration reduced by around 65 % while maintaining 96 % of reliability when sending 100 bytes of IPv6 packet/ 5 min.

6.4 Switching between Communication Modes

This section evaluates the BLE mode switching feature implemented in this thesis. The switching is performed from connection-based to connection-less

and vice versa. As we discussed in Section 4.1, the BLE devices establish a BLE connection to update routing information, network prefix, and configuration. For this experiment, devices switch to connection-based BLE every 1 hour and switch back to connection-less BLE once all necessary information is updated.

6.4.1 Experimental Setup

This experiment uses the same experimental setup used for power measurements in Section 6.2. In connection-based BLE mode, the **Connection_Interval** as 7.5 ms, **Slave_latency** as 0 and **Supervision_Timeout** as 100 ms is used. While in connection-less BLE mode, **Primary_advertising_Interval** is set to 20 ms where Ext. Adv. is activated for only one Adv. event of 20 ms. The application interval is set to 5 mins while the switching to connection-based BLE is performed every 60 min. As Section 6.3 confirmed that the connection-less BLE with CL 3 configuration is more reliable than CL 1 configuration, we used CL 3 configuration to transmit 100 bytes of IPv6 packet every 5 min. The experiment measures the average power consumed by a BLE radio as well as the transmission latency of a node. The experiment is performed for 4 hours and repeated 5 times.

6.4.2 Result

Figure 6.11 shows that the average power consumed by a BLE radio in connection-based BLE (switching to connection-based, exchanging information, and switching back to connection-less) is significantly high compared to connection-less BLE. The average time required to switch to connection-based BLE and exchange all the information is 85 ms. While the average time required to switch back to connection-less BLE from connection-based BLE is 22 ms.

The experiment also measures the transmission latency of a node, which can be observed in Figure 6.11. The unsuccessful transmission of IPv6 packets is indicated by a red cross on the x-axis of latency in Figure 6.11. The expected

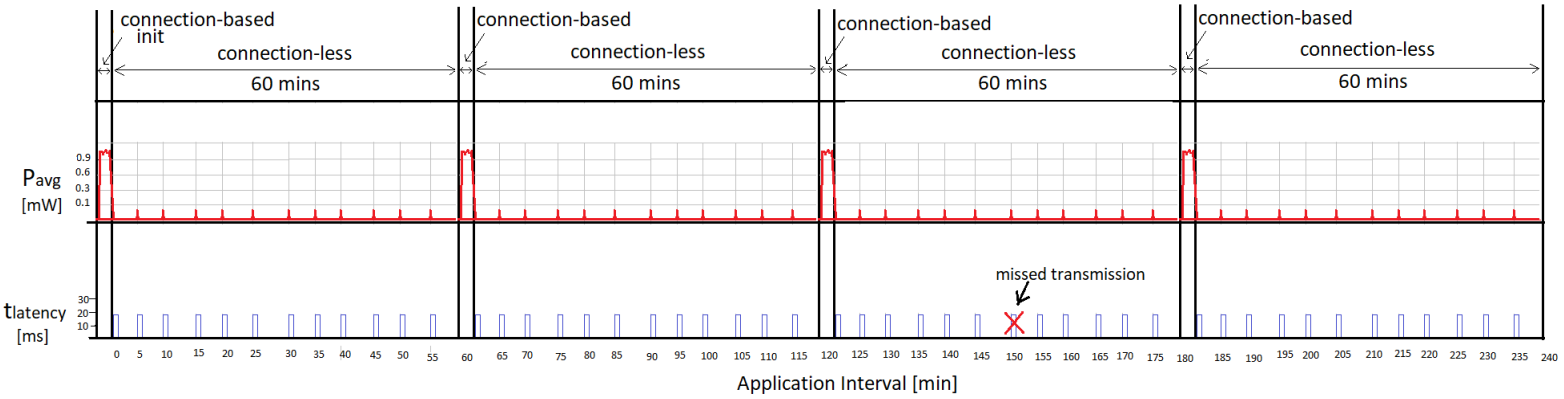

Figure 6.11: The avarage power consumption of BLE radio and transmission latency in connection-less BLE communication when the 100 bytes of IPv6 packet is transmitted every 5 mins. It also includes the switching between connection-based and connection-less BLE every 60 mins.

reliability of a node device was 96 % since we used connection-less BLE with CL 3 configuration. However, the experimental data shows 97 % reliability.

7 Conclusions and Future Work

This chapter concludes the thesis work and proposes future work that can be considered to enhance the performance of the IPv6 over connection-less BLE implementation.

7.1 Conclusions

This Master thesis compares the connection-less BLE over connection-based BLE for IPv6 communication and summarizes the benefits of connection-less IPv6 communication for low duty cycle applications. The major contribution of this thesis is the design of an IPv6 over connection-less BLE communication stack suitable for the architecture of Zephyr OS and its hardware-independent implementation.

The experimental studies performed in this thesis provides the first comparison of the IPv6 over connection-less BLE with the existing connection-based communication. The evaluation shows that the connection-less BLE (CL 3 channels) requires 30 % less radio-on time and 20 % less energy than the reconnect approach to transmit a successful IPv6 packet. The reliability experiment shows that the connection-based BLE has 100% reliability over the distance of 10 meters, while connection-less BLE (CL 3 channels) has around 96 % reliability for 100 bytes of packets. The experiment shows that the average power consumed by a BLE radio of a node can be reduced by 65 % by providing 96 % of reliability with connection-less BLE (CL 3 channels) when transmitting 100 bytes of IPv6 packet/5 mins.

7.2 Future Works

Energy Consumption:

As the implementation of the IPv6 over connection-less BLE communication stack is hardware-independent, it would be interesting to compare the energy consumption of nRF52 development kit with another BLE 5 supported hardware.

Apart from that, in IPv6 over connection-less mode, the router continuously scans for IPv6 packets over the Adv. channels and assumed to be wall-powered. It can be avoided by implementing synchronization between two devices which would allow the router to wake up and scan only on the time indicated by the node.

Security:

Currently, no security is provided over the connection-less communication. BLE connection-based communication allows devices to use different security levels to encrypt the communication to avoid man-in-the-middle attacks. As a future work, the security key used in connection-based communication could be reused in connection-less communication to encrypt the data.

Avoid Interference:

As we discussed in Section 4.5.2.3, the interference on the secondary Adv. channel is avoided using random channel selection. In future, the Channel Selection Algorithm #2 (CSA #2) can be implemented for connection-less communication as it is designed to avoid interference and multipath fading effects in connection-based communication and it makes difficult to trace the used channel of next event.

Different PHY:

The current implementation of IPv6 over connection-less BLE doesn't support LE coded PHY, since the LE coded is not implemented for Adv. mode in Zephyr OS BLE stack. Therefore, as a future work, energy consumption

and reliability can also be checked over LE coded PHY by providing support in BLE stack.

Broadcast multicast packets:

To transmit a multicast packet, devices need to establish a connection and transmit a multicast packet in unicast fashion. For example, if the Border Router (6LBR) is connected to multiple node (6LNs) devices and wants to transmit the multicast packet to all the connected 6LNs, it has to replicate the packet and transmit it as a unicast to each node [**18**]. Furthermore, if the 6LN needs to transmit the multicast packet to all the other 6LNs, it transmits such a packet as a unicast to the 6LBR, which later transmits it as a unicast to all the connected 6LNs. This is an overhead on 6LBR as it has to transmit several packets to achieve the multicast behavior of IPv6. Further study can be perform to use IPv6 over connection-less BLE to transmit multicast packet over BLE.

Bibliography

[1] Konstantin Mikhaylov, "Accelerated Connection Establishment (ACE) Mechanism for Bluetooth Low Energy",IEEE 2014 978-1-4799-4912-0/14

[2] Methods Mouser's technology and solution, Volume 1 issue 1

[3] http://www.summitdata.com/blog/whats-new-with-bluetooth-5-0.

[4] Mario Collotta, Giovanni Pau, Timothy Talty, IEEE and Ozan K. Tonguz, "Bluetooth 5: a concrete step forward towards the IoT" IEEE Communications Magazine · October 2017

[5] Jacopo Tosi, Fabrizio Taffoni, Marco Santacatterina, Roberto Sannino and Domenico Formica "Performance Evaluation of Bluetooth Low Energy: A Systematic Review" Published: MDPI journal. 13 December 2017,

[6] Bluetooth Core Specification v5.0

[7] https://www.electronicsweekly.com/news/design/communications/pros-cons-bluetooth-low-energy-2014-10/

[8] S. Deering, R. Hinden, RFC 2460 - Internet Protocol, Version 6 (IPv6) Specification. https://tools.ietf.org/html/rfc2460, December 1998.

[9] Haolin Wang, Minjun Xi, Jia Liu, Canfeng Chen, "Transmitting IPv6 Packets over Bluetooth Low Energy based on BlueZ" The 15th International Conference on Advanced Communications Technology-ICACT2013, 27 January 2013.

[10] M. Collotta and G. Pau, "A novel energy management approach for smart homes using Bluetooth low energy," IEEE J. Sel. Areas Commun., vol. 33, no. 12, pp. 2988–2996, Dec. 2015.

[11] Gaoyang Shan,Sun-young Im, and Byeong-hee Roh "Optimal AdvInterval for BLE Scanning in Different Number of BLE Devices Environment" 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 10-14 April 2016 .

[12] From cable replacement to the IoT Bluetooth 5 White Paper, Bluetooth 1MA1083e .

[13] Naresh Gupta. Inside Bluetooth Low Energy. Artech house, 2013.

[14] BLE Beacons for Internet of Things Applications: Survey, Challenges, and Opportunities. Kang Eun Jeon , James She, Perm Soonsawad, and Pai Chet Ng. IEEE Internet of Things journal , VOL. 5, NO. 2, April 2018

[15] https://www.msoon.com/

[16] Michael Spörk, "IPv6 over Bluetooth Low Energy using Contiki", master's thesis.

[17] BlueNRG-1, BlueNRG-2 BLE stack programming guidelines, PM0257 Programming manual

[18] J. Nieminen, T. Savolainen, M. Isomaki, Nokia, B. Patil, AT&T, Z. Shelby, Arm, and C. Gomez. RFC 7668 - IPv6 over BLUETOOTH(R) Low Energy. https: //tools.ietf.org/html/rfc7668, October 2015.

[19] B. Carpenter, S. Jiang, "RFC 7136 - Significance of IPv6 Interface Identifiers". https://tools.ietf.org/html/ rfc7136, February 2014.

[20] R. Hinden, S. Deering, "RFC 4291 - IP Version 6 Addressing Architecture". https://tools.ietf.org/html/rfc4291, February 2006.

[21] Ed. J. Hui, P. Thubert, "RFC 6282 - Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks". https://tools.ietf.org/html/rfc6282, September 2011.

[22] https://docs.zephyrproject.org/1.13.0/introduction/introducing_zephyr.html.

[23] https://www.nordicsemi.com.

[24] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler, RFC 4944 - Transmission of IPv6 Packets over IEEE 802.15.4 Networks. https://tools.ietf.org/html/rfc4944, September 2007.

[25] Wondeuk Yoon, Kiwoong Kwon, Minkeun Ha, and Daeyoung Kim. Transfer IPv6 Packets Over Bluetooth Low Energy with Ensuring Emergency Data Transmission.

[26] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in 29th Annual IEEE International Conference on Local Computer Networks, (Tampa, FL, USA, USA), pp. 455– 462, IEEE, Nov 2004. [Online]. Available: http://ieeexplore.ieee.org/ document/1367266/, Accessed: 2018-04-03.

[27] https://github.com/contiki-os/contiki

[28] https://www.nordicsemi.com/Software-and-Tools/Software/Bluetooth-Software

[29] Texas Instruments Incorporated. Texas Instruments - Bluetooth Low Energy Software Stack. http://www.ti.com/tool/ble-stack, 2016.

[30] https://mynewt.apache.org/latest/network/docs/index.html

[31] https://docs.zephyrproject.org/1.9.0/kernel/timing/clocks.html

[32] https://github.com/zephyrproject-rtos/zephyr/issues/12367

[33] https://github.com/zephyrproject-rtos/zephyr/issues/12150

[34] https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52-DK

[35] Z. Shelby, Ed. S. Chakrabarti E. Nordmark C. Bormann TZI "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)" https://tools.ietf.org/html/rfc6775 , November 2012

[36] https://lists.zephyrproject.org/g/devel/topic/22683500#4805

[37] Michael Spörk, Carlo Alberto Boano, Marco Zimmerling, Kay Römer, "BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices" SenSys '17, November 6–8, 2017, Delft, Netherlands. https://doi.org/10.1145/3131672.3131687

[38] Bogdan Tudosoiu, "Minimum Viable Device to support Internet of Things Realization" Lund, January 2014, http://startup-scaleup.eu/wp-content/uploads/2016/03/Feasibility-Study_small.pdf