Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Experimental evaluation of Bluetooth Low Energy Mesh networks

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Raül Caldera Sànchez

ADVISOR: Carles Gómez Montenegro

DATE: September, 1st 2018

CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

LIST OF ACRONYMS

IoT: Internet of Things

RFID: Radio Frequency Identification

NFC: Near Field Communication

BLE: Bluetooth Low Energy

RTT: Round Trip Time

SIG: Special Interest Group

BER: Bit Error Rate

SOC: System On a Chip

HCI: Host-Controller Interface

ISM: Industrial, Scientific and Medical

GFSK: Gaussian Frequency Shift Keying

TDMA: Time Division Multiple Access

MD: More Data

CRC: Cyclic Redundancy Check

ACK: Acknowledgement

L2CAP: Logical Link Control and Application Protocol

ATT: Attribute Protocol

GATT: Generic Attribute Profile

SMP: Security Manager Protocol

CCM: Cipher Block Chaining-Message Authentication Code

AES: Advanced Encryption Standard

GAP: Generic Access Profile

AD: Advertising Data

TTL: Time To Live

PDU: Processing Data Unit

UUID: Universally Unique Identifier

SAR: Segmentation and Reassembly

ECDH: Elliptic Curve Diffie-Helman

USB: Universal Serial Bus

LED: Light Emitting Diode

SDK: Software Development Kit

LE LL: Low Energy Link Layer

PB-ADV: Provisioning over Bearer Advertising

MAC: Media Access Control

RSSI: Received Signal Strength Indicator

1. INTRODUCTION

1.1. Motivation

In the latest years, Internet of Things (IoT) has been a revolution, spanning ambits such as smart cities, healthcare, agriculture… Normally, IoT networks consist in a great number of interconnected devices. Such devices are usually energy constrained. Therefore, technologies that allow low energy consumption are the most suitable for IoT applications. Some of these technologies are Radio Frequency Identification (RFID), Near Field Communication (NFC), those comprised in the IEEE 802.15.4 standard, or, concerning our case, Bluetooth Low Energy (BLE).

BLE has been one of the most successful technologies in the IoT field. Because of this, Bluetooth Special Interest Group (SIG) has decided to extend it in order to support mesh topologies. The Bluetooth SIG has recently developed a suite of specifications for BLE with support for mesh topologies called Bluetooth Mesh.

Bluetooth Mesh is still a very young technology, published just a year ago from the writing of this document. Therefore, since it may still have not reached its full use and potential in real life scenarios, it is a good time to analyse some of its performance parameters, in order to try to determine its future impact. This is the purpose of this master thesis.

1.2. Objectives of the work

One of the goals of this thesis is to introduce the basic concepts of Bluetooth Mesh and BLE (upon which Bluetooth Mesh is built), and to carry out experiments regarding performance parameters such as latency, Round Time Trip (RTT) delays, and energy consumption.

A very important concept in a Bluetooth Mesh network is *friendship*, in which a Low Power node, usually a less autonomous and simpler device, relies on another node known as friend, so that the latter can receive and save the former's packets. This allows the Low Power node to wake up periodically to request its packets, and be put to sleep the rest of the time, and thus saving a lot of battery, and substantially increasing its lifetime.

Therefore, one of the goals of this project is to create a current consumption model for a Low Power device. Several experiments regarding current consumption and battery lifetime for this kind of devices have been carried out.

1.3. Organization of the document

This document is organized in two main chapters: Chapter 2 introduces the BLE [1] and Bluetooth Mesh main concepts [2][3][4][5], regarding their characteristics, protocol stacks, network topologies, and so on. Chapter 3 is the body of this work, in which several experiments related to Bluetooth Mesh performance are carried out.

The first experiments try to measure some latency and RTT values, for different scenarios with a varying number of devices, and thus a different number of hops. Then, a series of experiments try to generate a model for the current consumption and battery lifetime of a Low Power device.

Finally, in Chapter 4, some conclusions are given from the results of this work, as well as some ethical considerations about the impact of this work and the technologies used. Finally, some future work directions are provided.

2. BLUETOOTH LOW ENERGY AND BLUETOOTH MESH: OVERVIEW

This chapter presents an overview on BLE and Bluetooth Mesh. In it, several aspects of these technologies are explained, such as their protocol stacks, network topology, security mechanisms, and overall characteristics. As already mentioned, BLE is overviewed before Bluetooth Mesh, since they share many commonalities. The main purpose of this chapter is to introduce their main concepts in order to have a better understanding of the experiments carried out in Chapter 3.

2.1. Bluetooth Low Energy

Bluetooth Mesh operates upon BLE. Therefore, before delving on the former, which is the main subject of this work, an overview on the BLE specification will be done in this section.

BLE is a single-hop, star topology wireless technology developed by the Bluetooth SIG as part of the Bluetooth 4.0 specification. Its purpose is to provide a reduced power consumption for several applications such as healthcare or home automation.

This section gives a quick view of BLE main characteristics, its protocol stack, and security methods.

2.1.1. BLE Protocol Stack

The BLE Protocol Stack (**Fig. 2.1**) is divided in two main parts: The Controller and the Host. The former is in charge of the lower layers (physical and link layers) and is usually implemented in Systems On a Chip (SOCs) with radio capabilities, while the second comprises the upper layers and is implemented in application processors.

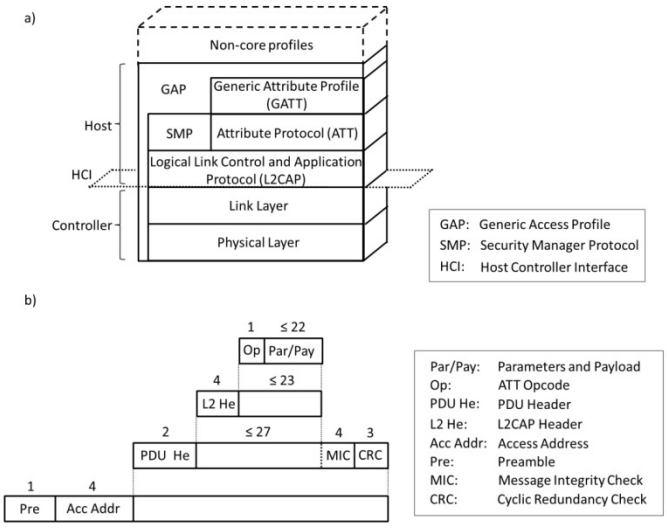


Fig. 2.1. BLE protocol stack [1]

Communication between Host and Controller is carried out by the Host-Controller Interface (HCI). On top of the Host, non-core profiles that provide application layer functionalities not defined by the Bluetooth specification may be run.

2.1.1.1. Physical Layer

BLE operates in the 2.4 GHz Industrial, Scientific & Medical (ISM) band (**Fig. 2.2**). It consists of 40 channels, with 2 MHz channel spacing. 37 of these channels are data channels, where bidirectional communication between connected devices takes place, and the remaining 3 are advertising channels, and are used for device discovery, connection establishment and broadcast transmission.

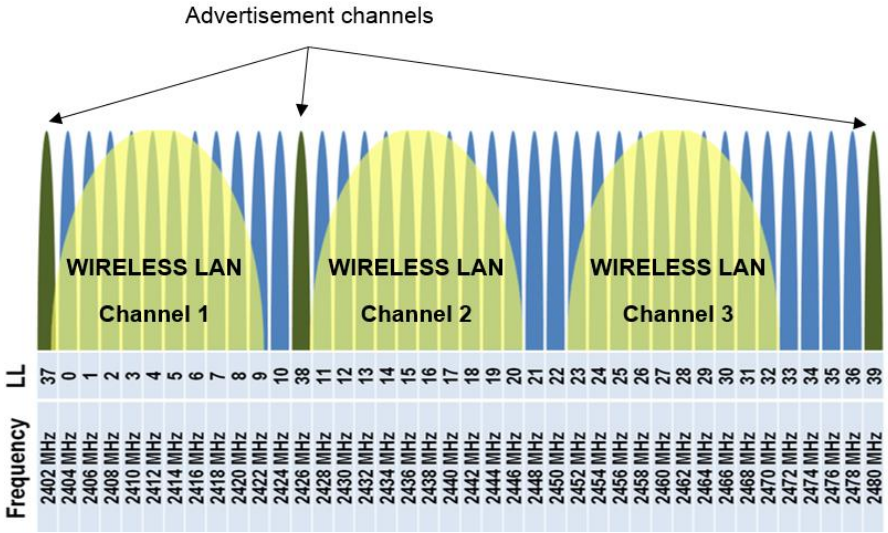


Fig. 2.2. BLE advertising channels in the 2.4 GHz ISM band

BLE incorporates an adaptive frequency hopping algorithm, by which the channel for data transmission is changed in each connection event. Gaussian Frequency Shift Keying (GFSK) modulation scheme is used, and the data rate at this layer is 1 Mbit/s.

2.1.1.2. Link Layer

Broadcast data transmission structure has two main types of devices. The first one is the advertiser, which sends data broadcast in advertising packets using the advertising channels, and the second is the scanner, which purpose is to receive such data. Every transmission takes place in intervals of time called advertising events. The advertising channels are used sequentially.

On the other hand, bidirectional data communication follows a different scheme. First, the advertiser, or slave, announces itself through the advertising channels, and the other device, called the initiator or master, listens them. The initiator can transmit a *Connection Request* message to the advertiser to make a connection. Once the connection is made, packets will be identified by a random 32-bit access code.

The master can manage many simultaneous connections with several slaves, and thus forming a piconet that follows a star topology. Slaves are in sleep mode by default. Master coordinates the medium access following a (Time Division Multiple Access) TDMA scheme, and also controls the frequency hopping algorithm.

When the connection is created, the packets are transmitted during non-overlapping time units called connection events. Each connection event uses a certain frequency. The connection event always starts with the master sending a packet, and the slave must respond.

Connection may end when there is nothing else to send, which is controlled by the More Data (MD) bit, or when two consecutive erroneous packets are sent, which is controlled by a 24-bit Cyclic Redundancy Check (CRC).

The time between connections is called *connInterval*, which can take a range of values from 7.5 ms to 4 seconds. Another parameter, called *connSlaveLatency*, sets the number of connection events during which the slave is not required to listen, and can take values from 0 to 499 (**Fig. 2.3**). The amount of time set by the *connInterval* and *connSlaveLatency* parameters should not exceed the time set by the parameter *connSupervisionTimeout*, since it would cause a supervision timeout. The latter parameter can take a range of values from 100 ms to 32 seconds.

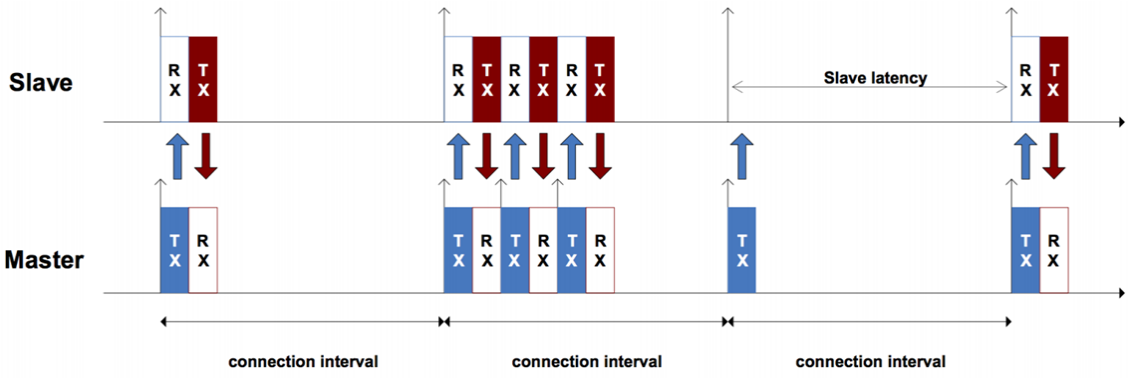


Fig. 2.3. connInterval and connSlaveLatency timing sequence

Link Layer uses a cumulative Acknowledgement (ACK) mechanism for flow-control.

2.1.1.3. The Logical Link Control and Application Protocol (L2CAP)

The main goal of the Logical Link Control and Application Protocol (L2CAP) is to multiplex the data of the upper layer protocols. The data of these services is handled in a best-effort approach. Since upper layer protocols provide data units that fit into the L2CAP maximum payload size, segmentation and reassembly are not used.

2.1.1.4. The Attribute Protocol (ATT)

This protocol works on top of L2CAP, and defines the roles of server and client. It also establishes data structures called attributes. Attributes are stored in the server, and the client can send requests to access them.

Servers can send the attributes with two types of messages: *notifications*, that do not need confirmation by the client, and *indications*, which do require the client to send confirmation.

2.1.1.5. The Generic Attribute Profile (GATT)

The GATT defines a framework that uses the ATT for discovery of services and the exchange of characteristics between devices. A characteristic is a set of data which includes a value and properties, and they are stored in attributes.

2.1.1.6. The Security Manager Protocol (SMP)

BLE offers security at the link and ATT layers. The link layer supports encryption by implementing Cipher Block Chaining-Message Authentication Code (CCM) algorithm, and authentication with a 12-byte signature computed with a 128-bit Advanced Encryption Standard (AES) block cipher, with a counter included to prevent replay attacks. BLE privacy feature mechanism also allows to use private addresses and change them periodically, to avoid tracking of the device.

BLE supports different levels of pairing and key sharing, which message exchange is managed by the SMP protocol. However, the supported pairing methods do not protect against passive eavesdropping.

2.1.1.7. The Generic Access Profile (GAP)

GAP defines four main roles for devices: the broadcaster, which broadcasts data via the advertising channels and cannot make connections with other devices, the observer, which receives the data transmitted by the broadcaster, the central (master) and the peripheral (slave). It also defines modes and procedures for the discovery of devices and services, security and management of connections.

Because of its topology, BLE has some limitations when it comes to coverage, and this can bring shortcomings in applications such as home automation. In the next section, Bluetooth Mesh, an overcoming technology which may be a solution for such shortcomings, will be overviewed.

2.1.2. BLE performance characteristics

Bluetooth Low Energy experiences a trade-off between energy consumption, piconet size, and throughput. Such characteristics have to be defined according to the specific requirements of each application, and they mainly depend on how the connectivity events between the nodes are configured.

Theoretical results have determined, for a coin cell battery powered BLE device, a lifetime that ranges from 2 days to approximately 14 years. Furthermore, the

estimated number of simultaneous slaves per master ranges from 2 to 5917. The minimum latency to obtain a sensor reading from the master is 676 µs. Performance of BLE communications will be affected by the Bit Error Rate (BER) introduced [1].

2.2. Bluetooth Mesh

As seen in last subsections, BLE was originally designed to only enable star topology networks. This limits BLE applicability in many IoT domains, since star topology networks may not ensure connectivity for all devices.

The Bluetooth Mesh is a specification developed by the Bluetooth SIG. It uses the lowest layers defined in the Bluetooth 4.0 Specification, used in BLE (**Fig. 2.4**). Its aim is to exploit the BLE potential in cases where the coverage area is larger than a single radio coverage range.

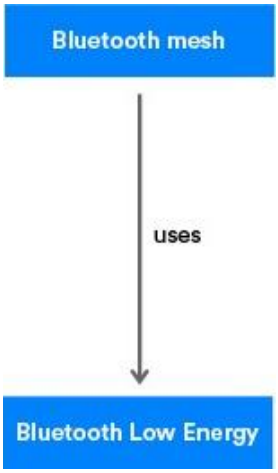


Fig. 2.4. BLE and Bluetooth Mesh relationship [5]

This section gives a quick view of Bluetooth Mesh main characteristics, its protocol stack, and security methods, and applications.

2.2.1. Bluetooth Mesh Architecture

Bluetooth Mesh follows a layered architecture, as defined in the Mesh Profile (**Fig. 2.5**). This layered architecture consists of the BLE Core Specification, The Bearer layer, the Network layer, the Lower Transport layer, the Upper Transport layer, the Access layer, the Foundation Model layer, and the Model layer.

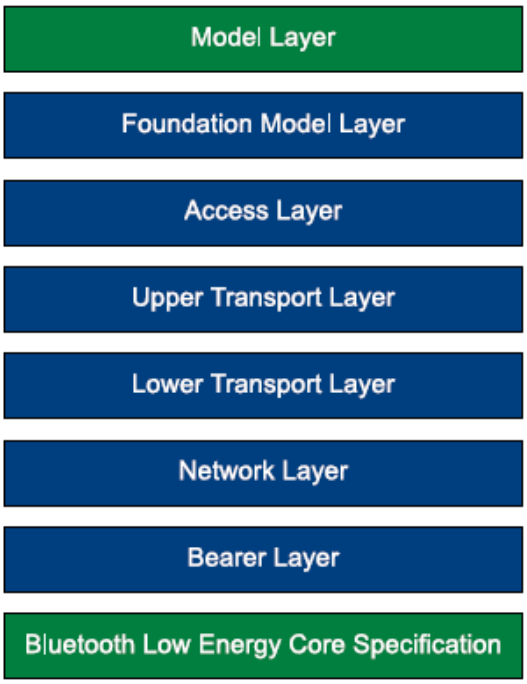


Fig. 2.5. Mesh system architecture [2]

The Model layer defines models that standardize operations in mesh user scenarios. Further explanation of models is found in Chapter 2.2.6.

The Foundation Model layer is in charge of defining the states, messages, and models required to configure the mesh network.

The Access layer defines how higher layer applications can use the upper transport layer and the format of application data. It is also responsible to define and control the application data encryption and decryption that is performed in the upper transport layer.

The Upper Transport layer is responsible for encryption, decryption and authentication of application data. It provides confidentiality of access messages. It also establishes how control messages are used to manage the upper transport layer between nodes.

The Lower Transport layer controls the segmentation and reassembly of upper transport layer messages into multiple Lower Transport Processing Data Unit (PDUs), to deliver them to other nodes.

The Network layer defines the addressing of transport messages towards one or more elements. Elements are defined later in Chapter 2.2.6. It also defines the network message format that allows Transport PDUs to be transported by the bearer layer. This layer decides whether to relay messages, maintain them for further processing, or reject them. It is also in charge of network message encryption and authentication.

The Bearer Layer defines how network messages are transported between nodes. There are two types, the advertising bearer and the GATT bearer.

2.2.2. Protocol Stack and Network Topology

In the previous subsections, the several layers of Bluetooth Mesh have been seen. However, since mesh messages are generally contained inside the payload of BLE advertisement packets, Bluetooth Mesh physical and link layers are the same than those of BLE. Therefore, it is possible to compare both BLE and Mesh Protocol Stacks (**Table 2.1**).

However, Bluetooth Mesh presents a different layer organisation on the Host part of the protocol stack, and therefore, the upper layers are incompatible with each other.

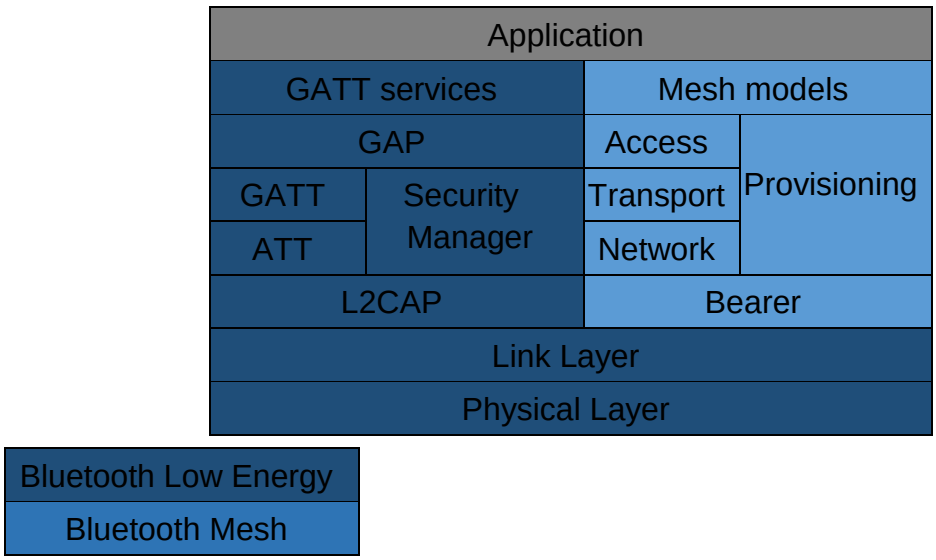


Table 2.1. BLE and Bluetooth Mesh protocol layers [3]

Unlike BLE, Bluetooth Mesh is a broadcast-based network protocol, where every device can communicate to any other device within radio range. Moreover, a device can send a message to other devices outside its range by means of controlled flooding whereby network nodes relay such message towards the destination. Devices can also drop in and out of the network at any given time.

2.2.2.1. Mesh transport

Bluetooth Mesh uses the BLE advertiser and scanner roles. These devices communicate by sending advertisement packets, which are picked up by mesh devices and handled like other BLE advertisement packets. A unique Advertising Data (AD) type in the mesh packet payload is added.

Bluetooth Mesh has the particularity that the advertisement payload changes on every transmission. Every Bluetooth Mesh advertisement is transmitted only once for every device, and if there is no traffic in the mesh, the devices stay silent.

2.2.2.2. Relays

Relays are devices that forward messages towards their destination, and thus expanding the range of the network (**Fig. 2.6**). Any mesh device may be configured as a relay. The number of hops a message can do in a network is limited by the Time To Live (TTL) value, which will decrement with every hop. Such TTL value can be up to 126, and is decremented by each time the message is received and relayed by one device.

Relays use the technique of flooding, by which the message is broadcast to all the neighbours. This brings high probability that the message will be properly delivered, without needing routing mechanisms, which are not supported by the Mesh Profile Specification, nor information about the network topology. To avoid forwarding the same message twice by a relay, all mesh devices have a message cache to know which packets the relay has already forwarded.

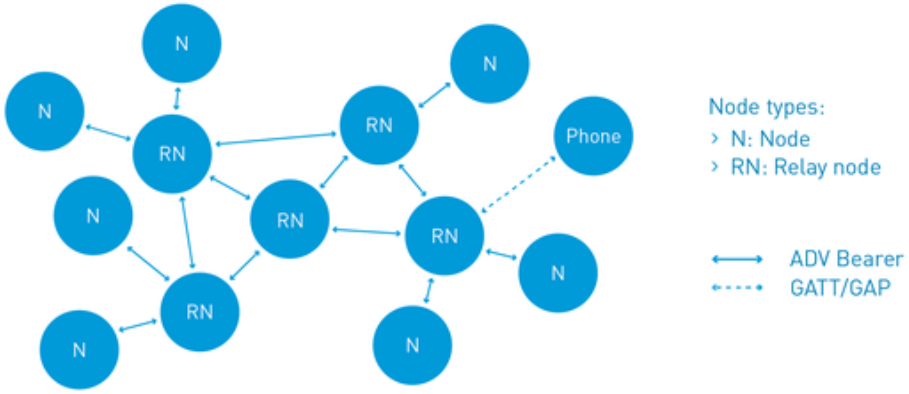


Fig. 2.6. Relay nodes in a Bluetooth Mesh network [5]

Despite the aforementioned advantages of flooding, one drawback is that this mechanism can cause a lot of redundant traffic, which may affect the throughput and the reliability of the network. One way to reduce this undesired effect is to limit the number of relays. The number of relays brings a trade-off between message redundancy and network reliability, and therefore, such number must be tuned according to each network own characteristics such as layout, traffic to support and reliability requirements.

2.2.2.3. Low Power nodes and friendship

In order to enable broadcast-based communication, the radio of the devices must be constantly listening, causing higher power consumption than in BLE devices. This may be a problem for Low Power devices, which cannot keep up with such consumption, and there is also trouble if they cannot listen and process mesh messages as well as to receive the security updates. If the Low Power Node does not receive such messages, it may not operate correctly, and could even be dropped out of the network.

To overcome this, Bluetooth Mesh includes a *friendship* feature, by which the Low Power node establishes a relationship with a neighbouring mesh device that is not as energy-constrained. Friendship is first initiated by the Low Power node. The Friend device will cache messages in its Friend Queue, and will forward them to the Low Power device at regular time intervals. The Friend device will also deliver security updates. This allows the Low Power device to save energy and to listen only at certain established times.

Such nodes must be within a single hop of each other and in the same subnet. A Friend node may be friends with multiple Low Power nodes, while a Low Power node can only be friends with a single Friend node.

2.2.2.3. GATT proxy

The aim of the GATT proxy protocol is to enable support for BLE devices that do not support mesh packets. BLE devices establish a GATT connection with the GATT bearer that will allow them to participate in the mesh network.

2.2.2.4. Mesh gateway

A mesh gateway is a node that is able to translate messages between a mesh network and a non-Bluetooth Technology. A node may be able to send and receive messages through a mesh gateway.

2.2.3. Addressing

Bluetooth Mesh defines three types of addresses (**Fig. 2.7**). The first type are unicast addresses, which are unique for every device. A device joining the network is provided a range of unicast addresses. These addresses cannot be changed, and are used in order to send a message directly to the device. A mesh network can bear up to 32767 unicast addresses.

The second type are group addresses, which allow to identify a group of devices at once with a single address. Groups may have many devices, and a device can be at different groups simultaneously. There can be a maximum of 16127 group addresses in a mesh network.

The last type are virtual addresses. These are multicast addresses and have a large address space (128-bit), so they are Universally Unique Identifier (UUIDs). These addresses do not have to be tracked by a network configuration device. Therefore, addresses can be generated ad-hoc between devices.

There is also a special value to represent an unassigned address that is not used in messages.

Values	Address Type
0b0000000000000000	Unassigned Address
0b0xxxxxxxxxxxxxx (excluding 0b0000000000000000)	Unicast Address
0b10xxxxxxxxxxxxx	Virtual Group
0b11xxxxxxxxxxxxx	Group Address

Fig. 2.7. types of addressing in Bluetooth Mesh [2]

2.2.4. Provisioning

When a device is still not a member of the mesh network, it is identified as an unprovisioned device.

When a device connects to a network, it is assigned its unicast address, a network key and a device key. Such device is known as node. The provisioning is done by a trusted device called *Provisioner*. The Provisioner knows all the addresses of the devices in the network. An unprovisioned device cannot send or receive mesh messages, but it can advertise its presence to a provisioner, which can invite it to join the mesh network, after it has been authenticated.

Once a new device has been provisioned, the provisioner uses the device's device key to establish a secure channel to configure it. It now can send or receive mesh messages, and is managed by a Configuration Client, to configure how the node communicates with other nodes. The Configuration Client may also remove the node from the mesh network.

It is important to highlight the difference between device and node. A single device may support several instances of a node, by being provisioned more than once to different mesh networks. Each instance of a mesh network is determined by addresses and a device key, which are obtained during provisioning.

2.2.4.1. Adding devices to a mesh network

As already explained, devices are added into the mesh network by a provisioner, and they become nodes. Provisioning of devices can be done by using either an advertising bearer or a point-to-point GATT-based bearer. The former is implemented by all devices. The latter allows devices that do not support provisioning over an advertising bearer natively to be provisioners.

To allow assistance in provisioning several devices, a device has an attention timer that can be set by a provisioner. Then a provisioner may send a single message to a device to cause it to identify itself for a certain period of time, and after that, the device stops doing it.

The protocol to run over these two bearers is a derivative of the SMP. SMP requires a reliable bearer, and this is something that cannot be guaranteed by the advertising provisioning bearer, because many devices may have a very

limited user interface. Therefore, the new protocol is designed to enable reliable delivery of messages regardless of the bearer.

Apart from this, many devices may be unable to advertise or to comprehend mesh messages. To enable communication, the specification enables the use of GATT connectivity for all existing devices.

2.2.5. States, bound states, and messages

A state is a value representing a condition of an element. An element exposing a state is known as server, while an element accessing a state is known as client. When a state has more than two values, it is known as a composite state.

Bound states are states that are related to each other, in a way that a change in one results in a change in the other. Bound states may be from different models in one or more elements (see 2.2.6 for details on models and elements).

Messages are the way of communication in a mesh network. Messages operate on states. For each state, there is a defined set of messages. A client may use them to request a value of a state or to change it. A server supports each message, and it may also transmit unsolicited messages that carry information about states or changing states.

A message has an opcode, associated parameters and an associated behaviour. The total size of the message is determined by the transport layer, which may use a Segmentation And Reassembly (SAR) mechanism. The goal is to fit messages in a single segment.

The transport layer is able to transport up to 32 segments with its SAR mechanism, with a maximum message size of 384 octets. SAR does not impose extra overhead on the access layer payload per segment.

Messages are defined as acknowledged or unacknowledged. Acknowledged messages require a response.

2.2.5.1. Message publication and subscription

Bluetooth Mesh uses a publication-subscribe structure for messages, in which nodes that generate messages publish them to a unicast, group or virtual address, and nodes that are interested in receiving such messages will subscribe to these addresses.

Each instance of a model can subscribe to one or more group or virtual addresses. A node can have several subscriptions per instance of a model's element. Multiple subscription addresses allow a node to respond to messages published to different groups.

2.2.6. Models and Elements

A model is a specific service in a device. It defines a set of states, as well as messages to act on these states. A single device can have several models. Each model carries out different tasks like device configuration, sensor readings, etc.

In order to allow and standardize communication between different devices, the Mesh Profile Specification defines an access layer, whose purpose is to route mesh messages between the various models in a device. The specification also defines three types of models: server models, client models and control models. A single device may include server, client and control models.

Server models are composed of one or more states, that can be in one or more elements. They define a set of mandatory messages that can be transmitted or received, and the behaviour of the element when it transmits or receives such messages.

Client models define a set of mandatory and optional messages that a client uses to request, change or consume corresponding server states. The client models do not have states.

Control models may contain both client and server features, in order to communicate with both other client or server models. They may also contain control logic, which is a set of rules to coordinate the interactions between other models that the control model connects to.

Models belong in elements. Every device can have one or more elements. Each element has its own unicast address, and therefore it is a differentiated (virtual) entity in the mesh network. When elements receive messages, these messages are handled by a model instance (**Fig. 2.8**). Each node has at least one element, which is known as primary element, but may have others. The number and structure of elements within a node is static throughout the lifetime of the node inside the network. In case of change, the node must be re-provisioned.

Each element has a GATT Bluetooth Namespace Descriptor value that helps identify which part of the node the element represents.

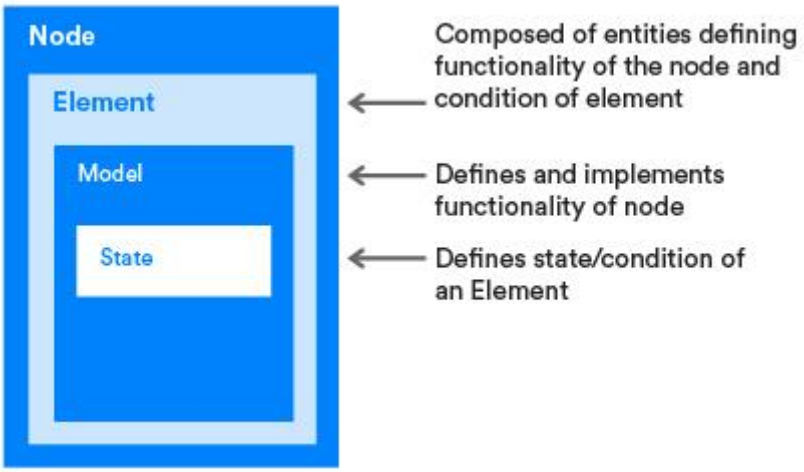


Fig. 2.8. Element, model and states structure in a device [5]

To allow uniqueness in message resolving, only one model instance per element can implement a handler for a specific message opcode. If a device has multiple instances of the same model, each instance must be assigned to a different element. If two models implement handlers for the same message, these models must also be in different elements.

Extended models are models that are made up of other models, which can be in multiple elements. These kinds of models allow complex behaviour with the advantage of minimizing message and state duplication. Otherwise, root models are models that are self-contained.

As already explained, a public and subscribe system is used for communication between models. Every model may subscribe to a set of group and virtual addresses. Then the model will only handle messages that are published to one of the addresses it has subscribed to, or the containing element's unicast address. A model may also maintain a publish address to publish messages.

Coming up next, a real example to see how elements and models work is presented. Considering a node acting as a light bulb [5], may have this structure:

- Node: Light bulb
    - Element: Primary element.
        - Model: Functionality of node.
            - State 1: On/Off
            - State 2: Brightness from 0 to 10

2.2.7. Network configuration

Network configuration is carried out by a central network configurator. A device must be configured by a provisioner, and every device must implement a mandatory Configuration Server model in their first element, whose purpose is to configure the rest of its models.

Provisioner also has a Configuration Client model, and uses its instance to give the new device a set of application keys and addresses. It makes sure no duplicate unicast addresses are allocated.

2.2.8. Security

In this subsection the several security measures that Bluetooth Mesh employs to prevent third-party interference and monitoring will be overviewed. These measures comprise authentication, message encryption, privacy, replay protection, and obfuscation.

2.2.8.1. Authentication

As part of the provisioning process, authentication lets the user confirm that the device being added to the network is authenticated. The Mesh Profile Specification defines several out-of-band authentication methods, such as light blinking, output and input of passphrases, key pre-sharing…

Provisioning is secured with an Elliptic Curve Diffie-Helman (ECDH) public key cryptography.

2.2.8.2. Message encryption

Bluetooth Mesh supports two layers of encryption. Such encryption uses AES-CCM with 128-bit keys.

The lowest layer is called *network encryption*. It protects the messages in the mesh network from being readable by devices that do not form the network. The encryption is done with a network encryption key. Each subnet of the network has its own network key. A network can have up to 4096 subnets. Different keys for different subnets allow to effectively divide the network, since a mesh relay only forwards messages that are encrypted with a known network key. A node may belong to more than one subnet by knowing several network keys. When the device is provisioned, it is added to one subnet and it may be later added to more subnets using the Configuration Model.

The upper layer of encryption is the *transport encryption*. It limits what devices can do within a network by encrypting the application payload with an application or device key. This allows that some features are only allowed to certain devices in the network.

Application keys separate access rights to the applications in the network, while device keys are used to manage devices in the network. Each device has a unique device key, only known by the provisioner and the own device. It is used when configuring the device and for setting other device-specific parameters.

Both encryption layers include a message integrity check value used to validate that the message was encrypted with the right encryption keys.

2.2.8.3. Privacy

Privacy key allows encryption of source address and message sequence number in every mesh message.

2.2.8.4. Replay protection

To avoid replaying of previous messages, every device includes a unique pair of sequence number and source address in the messages. The device receiving the message stores the sequence number and ensures that it is the expected one.

2.2.8.5. Obfuscation

Obfuscation is a privacy mechanism that utilizes AES to encrypt the source address, sequence numbers and other header information using a privacy key, in order to make tracking of nodes more difficult.

2.2.9. Applications

Bluetooth Mesh has greater power consumption than BLE, due to the fact that the radio has to be running constantly in active mesh devices. Therefore, in the case devices are battery-powered, their lifetime will be reduced. For these devices, there is the Low Power node role, as already mentioned. Therefore, Bluetooth Mesh targets simple control and monitoring applications, like sensor data gathering in home environments, etc …

Bluetooth Mesh can support up to 32767 devices in a network, with a maximum of 126 hops in diameter.

3. EXPERIMENTS

Now that BLE and Bluetooth Mesh technologies have been overviewed in Chapter 2, and there is a greater understanding of them, the main objective of this chapter is to carry out a number of experiments with the purpose to determine some Bluetooth Mesh characteristics such as latencies, RTTs, and current consumption and battery lifetimes of the devices, all in realistic environments.

3.1. Hardware and software platforms

In order to carry out these experiments, build Bluetooth Mesh networks that can send Bluetooth Mesh packets across them, and analyse communication, a series of hardware platforms have been used. Such hardware is from the Nordic manufacturer, specifically the Nordic nRF51 Development Kit [6] and the Nordic nRF51 Dongle [7](**Fig. 3.1**). Both devices can be programmed to behave like a node in a Bluetooth Mesh network, to carry out several roles, and act as a provisioner, a client, a server, a relay node, or others.

The Nordic nRF51 Development Kit is a device that can work both by powering through Universal Serial Bus (USB), through some external power supply pins, or autonomously by using a 3 V cell battery. The second one can work almost the same as the Development Kit, but can only be supplied power through USB; it is smaller, and does not have as many pins, nor Light Emitting Diodes (LED) li. Due to this, there may be some software that is not suited for the dongles. Specifically, these are the device models used in this project:

- 4 Nordic nRF51 Development Kits (nRF51422, PCA10028).
- 2 Nordic nRF51 Dongles (nRF51422, PCA10028).

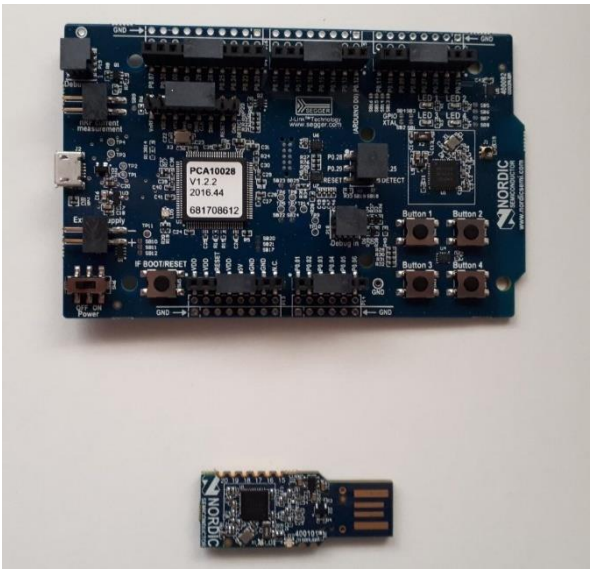


Fig. 3.1. NRF51422 DK (above), and NRF51422 Dongle (below)

Nordic also provides software environments for the nRF51 and nRF52 series, which are called Software Development Kits (SDKs). Such SDKs contain all the

Bluetooth software, structure and protocol for the devices to run them. SDKs contain a series of software that called examples, and they can be used to test the devices to do many tasks. In the experiments of this thesis, many of these examples are used.

SDKs come in many versions, both for BLE, and also a dedicated one for Bluetooth Mesh. For the experiments the following SDKs have been used:

- nRF5 SDK for Mesh v. 2.0.1 [8].
- nRF5 SDK v. 15.0.0: Necessary to build the nRF5 SDK for Mesh v. 2.0.1 [9].
- nRF5 SDK v. 11.0.0: Though not supporting Bluetooth Mesh, some of its examples have been used to carry out some of the experiments for this thesis [10].

To build the examples, and download them to the devices, two tools have been used: SEGGER Embedded Studio [11](**Fig. 3.2**) and Keil uVision 5 [12].

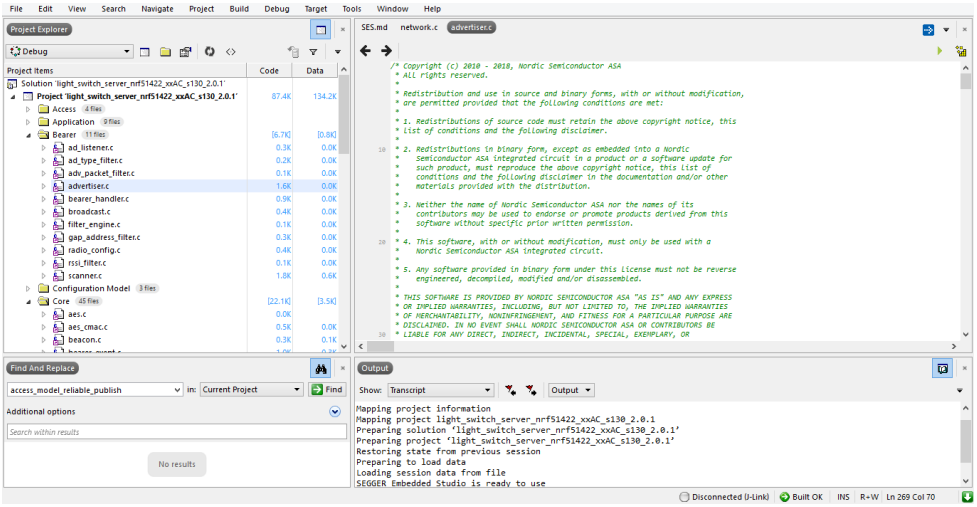


Fig. 3.2. SEGGER Embedded Studio interface

In many of the examples, it is necessary to use a sniffer to capture and analyse packets in order to do the appropriate measurements and obtain results. To do so, Nordic provides an application called nRF Sniffer [13], that allows a device to work as a sniffer. Version 2 of this software has been used, and programmed into one of the available dongles. The sniffer software used is Wireshark [14] (**Fig. 3.3**), which allows nRF Sniffer to be integrated into it.

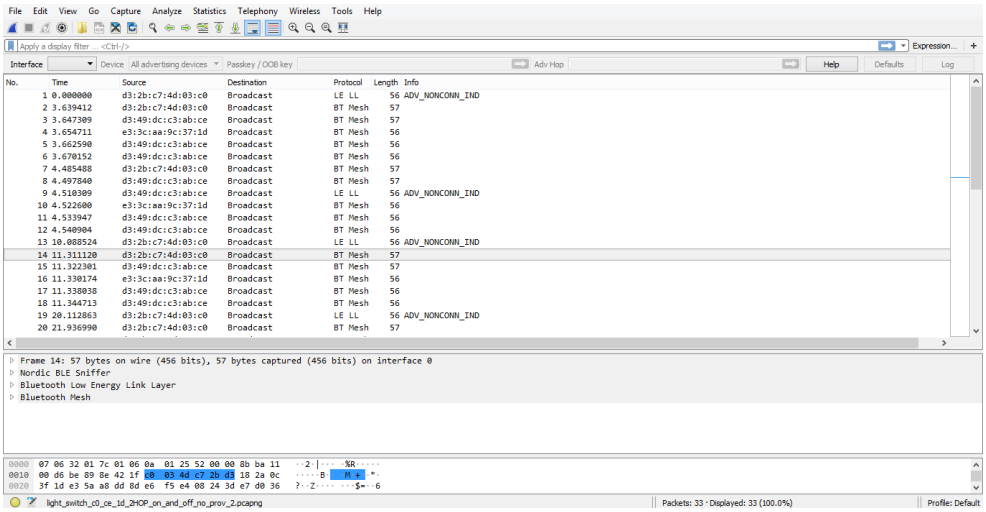


Fig. 3.3. Wireshark interface

The following sections explain the several experiments carried out in this project.

3.2. Latency and RTT experiments

In this part of the project, several experiments have been carried out in order to determine one-way latency and RTT values for Bluetooth Mesh networks. In these experiments, the purpose is to calculate the time a packet needs to travel from a client device to a server device (one-way latency), or the time it needs to get a response back to the client (RTT), through a number of end-to-end hops. In most cases, the end server (receiver) is out of range for the client to transmit it directly. In this case, there are some devices in between that act as relays.

Two main experiments have been carried out: the first one aims to measure one-way latency for a two-hop scenario, while the second one measures RTT for a scenario with one, two and three hops, as well as predicting further RTT values for even more hops.

3.2.1. First experiment: One-way latency for a 2-hop scenario

This first experiments aims to measure one-way latency for a scenario with 3 devices: a client, a server acting as a relay and a remote server (**Fig. 3.4**).

- Client Media Access Control (MAC) address: e5:88:09:4c:36:6a
- Server 1 (Relay) MAC address: fb:27:12:37:46:d6
- Server 2 (Remote device) MAC address: d3:2b:c7:4d:03:c0

A sniffer is located close to Server 1. Client and Server 2 cannot see each other directly.

One-way latency is considered the time between sniffed outgoing packet from Server 1 (fb:27:12:37:46:d6) to Server 2 (d3:2b:c7:4d:03:c0) and sniffed incoming packet from client (e5:88:09:4c:36:6a) to Server 1 (fb:27:12:37:46:d6). Therefore, the objective was to analyse when the time between a sniffed packet from

e5:88:09:4c:36:6a and a sniffed packet from fb:27:12:37:46:d6. Propagation delay is considered negligible (since the sniffer is very close to server 1).

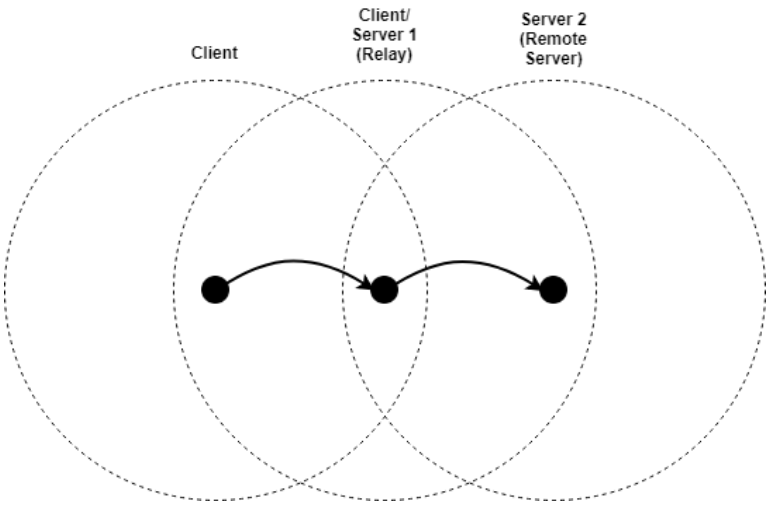


Fig. 3.4. Scenario for a 2-hop one-way latency calculation

The experiment is carried by using the *pb_remote* example. In this example there are three devices: the client and two servers. The purpose of the example is to remotely provision the most distant server by first provisioning the first server, and then provision the second server through the first one. The operations in the example are controlled in the form of commands that are given to the client, and then the devices act accordingly to these commands. The devices are arranged in a way that the client and the remote server cannot directly communicate with each other, but the first server is in the middle and can communicate with both, in order to ensure an end-to-end path in a forwarding scenario.

In this example, latency is measured by sniffing the packets in a location close to the first server (the relay). Closeness is to ensure that both packet reception and transmission by the Relay can be captured. It is then possible to measure the packet that passes between a sniffed incoming packet from the client to the relay server and a sniffed outgoing (presumably) forwarded packet from the relay server to the remote server (**Fig. 3.5**). This time can be considered to be the forwarding latency.

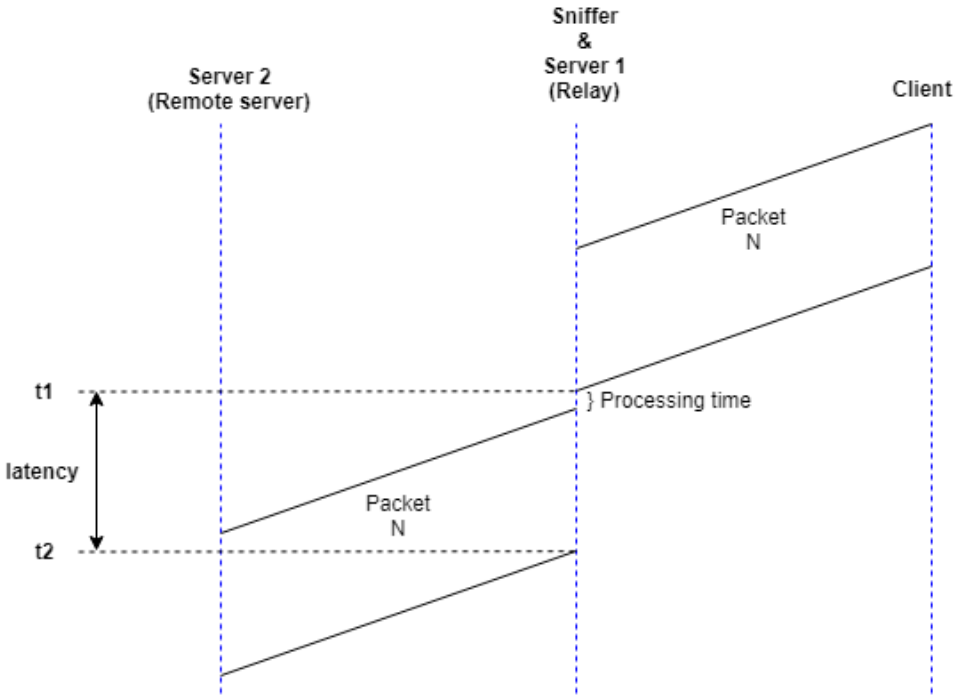


Fig. 3.5. Scheme of the expected situation to calculate one-way latency

The behaviour of the *pb_remote* example goes as follows:

1. Before the example runs, the three devices seem to advertise following the pattern described next. The two servers advertise every two seconds, and the client advertises every ten seconds. The packets are identified as “Mesh Beacon”, and the protocol is identified as Low Energy Link Layer (LE LL). This goes on until the first command is given to the client. The first command is the following: **1) Provision first available device with** Provisioning over Bearer Advertising **(PB-ADV)**.
2. When the first command is introduced by pressing the DK button, the client and the server exchange a series of packets identified by the sniffer as “PB-ADV”. It is at this moment that the provisioning between the client and the relay server is done. The first PB-ADV message is sent by the client upon command, and then the relay node seems to respond a certain number of PB-ADV messages back. At the same time, the second server keeps advertising.
3. When the second command is inputted at the client (**2) Set current client publish handle (corresponding to a known server)**), the client sends the first packets labelled as “Mesh Message” by the sniffer, and the protocol is identified as “BT Mesh”. These kinds of packets keep being exchanged between the client and the relay server. The remote server keeps sending advertising packets.
4. Upon inputting the third command at the client (**3) Start remote scanning**), the relay server starts sending PB-ADV packets, to which the remote server reacts by also sending this kind of packets. This is where the second provisioning is done. Meanwhile the client and the relay server keep sending “Mesh Message” packets. This behaviour seems to go on until the fourth command (**4) Cancel the remote scanning**) is inputted at the client.

5. When the fifth command (**5) Start remote provisioning**) is selected, the remote server also starts sending "Mesh Message" packets.

As a comment, packet lengths and number of packets between PB-ADV and Mesh Message packet interaction do not seem to follow any kind of logic that would indicate that packets are being forwarded. Moreover, the packets data is encrypted so it is impossible to recognize what is being transmitted. However, although the following results may not fully confirm to determine latency as such, it is considered that when selecting the fifth command, and the remote provisioning is selected, the two-hop transmission is carried out, and thus, the measured times are latency values indeed.

In order to calculate the latency, several of these transmissions have been analysed, and the median value of them all has been calculated (**Fig. 3.6**). These values correspond to the latency measurements after selecting the fifth command of the example, to which the remote provisioning is triggered, and thus creating the 2-hop one-way latency scenario.

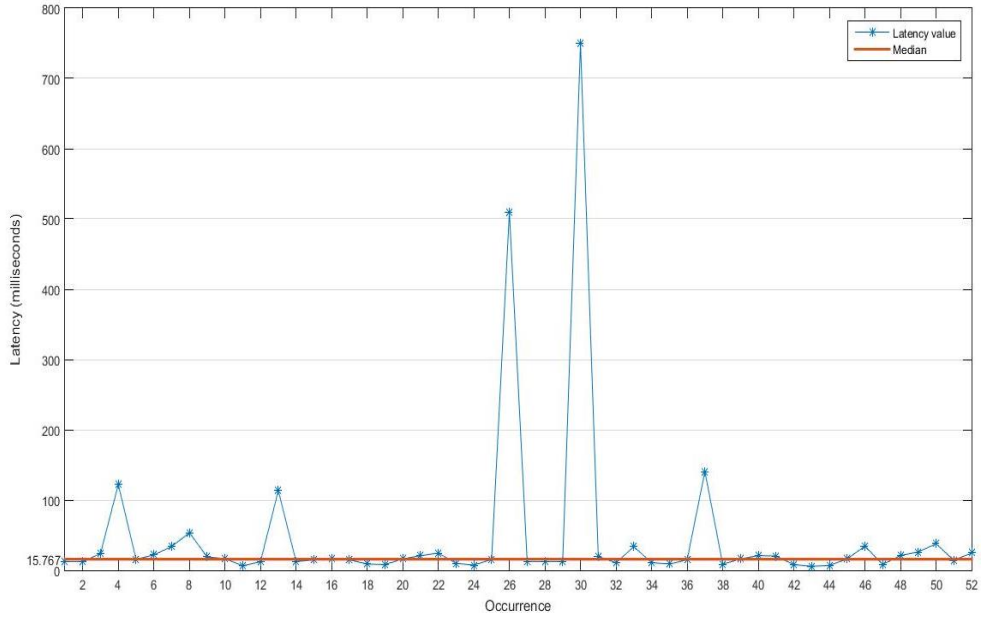


Fig. 3.6. Latency median value for several samples of the 2-hop latency scenario

As can be seen, there are some few occasions where for some reason the transmission takes much longer than usual. However, the median value showcases more properly which is the usual one-way latency value in this experiment. As shown in the figure, this median value is much lower and more according to the rest of the measurements. Therefore, the final one-way latency value for a transmission in a 2-hop scenario could be considered to be around the one of the median, that is 15.767 ms.

3.2.2. Second Experiment: RTT for different multihop scenarios

In this second experiment, the RTT for different multihop scenarios with different number of hops will be measured. In order to do this, another SDK example will be used, which is called *light_switch* example. The light switch example lets the user toggle LEDs on one or more server devices by pressing the buttons at the client device. The *light_switch* example is built upon the Simple On/Off model (**Fig. 3.7**), which allows bidirectional communication between devices, by switching things on or off by manipulating a single on/off state.

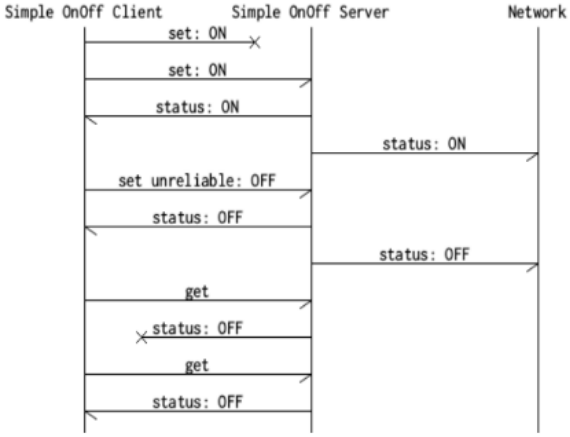


Fig. 3.7. Simple On/Off model behaviour

The complete behaviour of the *light_switch* example is as follows:

1. First of all, a provisioner device configures Button 1 on the client board to control the first server, Button 2 to control the second server, Button 3 to control the servers with Odd addresses, and Button 4 to control the servers with Even addresses. Once the device is done provisioning, it does not take part in the network and is excluded from the experiments.
2. After provisioning, the client can use its buttons to toggle the LEDs of the available servers, and thus sending packets to such server, which in return will send back packets in response, and thus, allowing to calculate the RTT by sniffing them. The packets sent when the buttons are pressed are labelled by the sniffer as “Mesh Message”, and the protocol is identified as “BT Mesh”. This kind of packets are the ones used to measure RTT. In between, there are also non-connectable advertising packets, labelled as “Mesh Beacon”, and the protocol is identified as LE LL.

The devices also have the capability of relaying the packets to distant servers which are not within the client’s reach, in order to be able to control the LED of servers that are further away. Therefore, this example is very useful to setup and evaluate multihop scenarios. By placing a sniffer close to the client, the packets can be captured, the hops of the scenario can be controlled and the RTT can be calculated (**Fig. 3.8**).

Note that it is not possible to trace the start of the packet transmission at the first hop, therefore such contribution to the RTT will not be included in the measurement.

In order to carry out this experiment, the transmission power of the devices has been reduced to - 16 dBm to ensure the devices can only reach their adjacent ones, and therefore to ensure the multihop scenario. To calculate the final RTT value for each case, the average of multiple different transmissions has been carried out for each case.

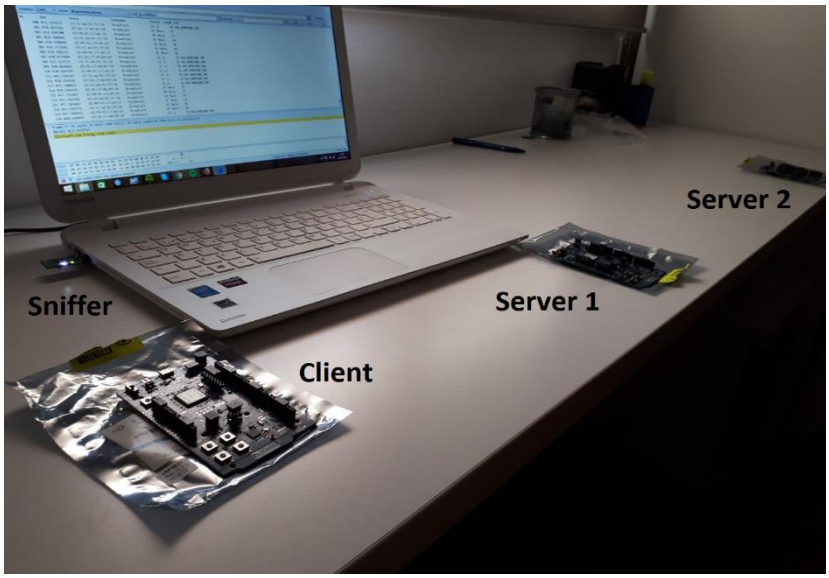


Fig. 3.8. A 2-hop RTT calculation setup

3.2.2.1. 1 hop

The first and most simple scenario is the 1 hop case (**Fig. 3.9**). In it, there are just two devices in the network: one client and one server. The client transmits the packets in broadcast, which are received by the server, which in return sends the response back, as expected based on the way the Simple on/off model works. These are the devices MAC addresses:

- Client MAC address: e5:88:09:4c:36:6a
- Server 1 MAC address: d3:49:dc:c3:ab:ce

Therefore, the purpose is to determine the time that passes between sniffing an outgoing packet from Client (e5:88:09:4c:36:6a) and the response from the Server (d3:49:dc:c3:ab:ce)

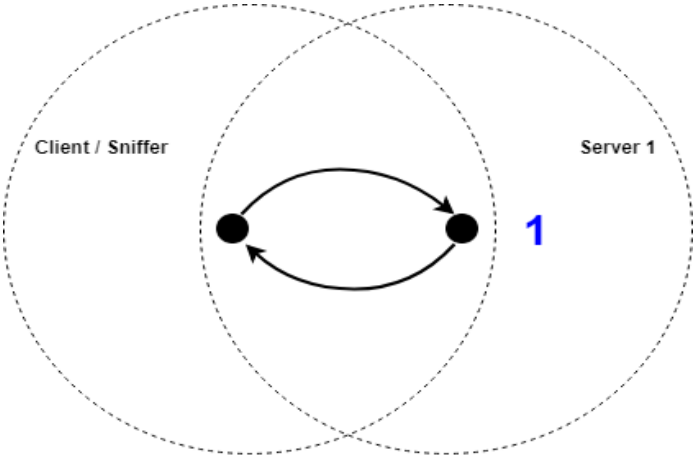


Fig. 3.9. Scenario for a 1-hop RTT calculation

As can be seen, the value is quite similar to the one-way latency value calculated with the *pb_remote* example. This may be because almost all the time delay is processing time done in the server, rather than transmission time. The following table shows the average RTT for this scenario (**Table 3.1**). This final value is the result of calculating the average values of 20 transmissions (**Fig. 3.10**).

Hop number	Hop	Average (Total) Hop Time (ms)
1	**Server 1 - Client**	**12.67**

Table 3.1. Average RTT Time for a 1-hop scenario transmission

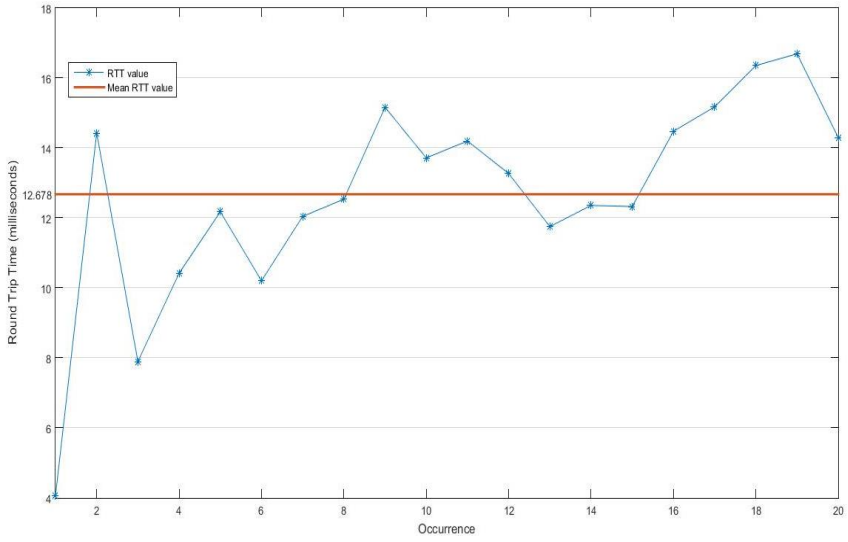


Fig. 3.10. Mean RTT value for several samples in a 1-hop RTT scenario

3.2.2.2. 2 hops

The following scenario includes one more device: another server (**Fig. 3.11**). In this case, to ensure that the communication follows two hops in a stable way

during the experiments, it is ensured that the client and Server 2 cannot directly communicate with each other. Therefore, Server 1 takes the role of relaying the packets. The MAC addresses of the used devices are:

- Client MAC address: e5:88:09:4c:36:6a
- Server 1 (Relay) MAC address: d3:49:dc:c3:ab:ce
- Server 2 (Remote Server) MAC address: e3:3c:aa:9c:37:1d

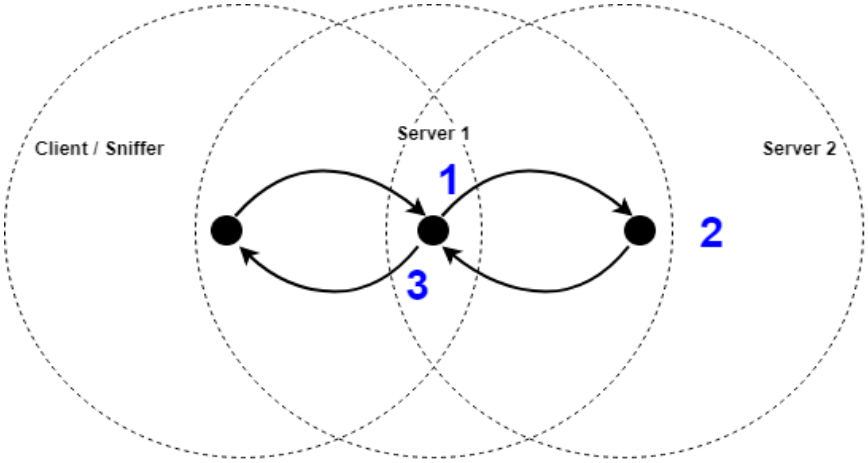


Fig. 3.11. Scenario for a 2-hop RTT calculation

The RTT in this scenario is the time it takes between sending a packet from the client (e5:88:09:4c:36:6a), relaying it through Server 1 (d3:49:dc:c3:ab:ce), receiving it by Server 2 (e3:3c:aa:9c:37:1d), and retransmit a response back to Server 1 (d3:49:dc:c3:ab:ce), which will relay it again to the client.

Therefore, the packet pattern is the following: e5:88:09:4c:36:6a - d3:49:dc:c3:ab:ce - e3:3c:aa:9c:37:1d - d3:49:dc:c3:ab:ce (**Table 3.2**).

Hop number	Hop	Average Hop Time (since last hop) (ms)	Average Total time (RTT) (ms)
1	**Server 1 - Server 2**	13.64	**39.18**
2	**Server 2 - Server 1**	12.85	
3	**Server 1 - Client**	12.69	

Table 3.2. Average RTT Time for a 2-hop scenario transmission

As can be seen, the RTT has increased substantially, up to more than three times the value for one hop. The calculation of the final value has also been calculated from the average value of 20 transmissions (**Fig. 3.12**).

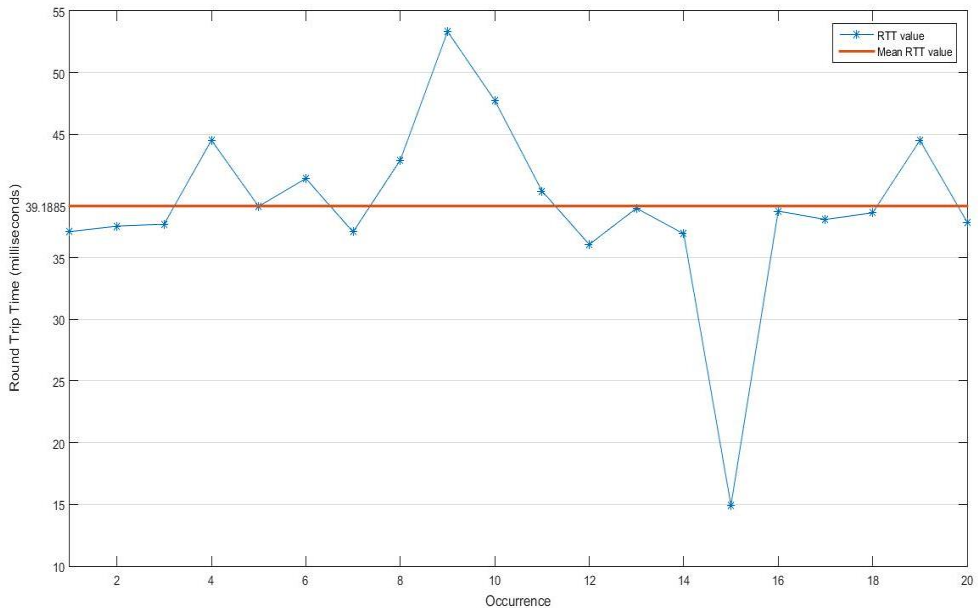


Fig. 3.12. Mean RTT value for several samples in a 2-hop RTT scenario

3.2.2.3. 3 hops

For this scenario, a new server has been included to the network again (**Fig. 3.13**). In this case, it has been made sure that each device can only communicate directly to its adjacent devices (their immediate neighbours), in order to ensure the multihop scenario. In this case, both Server 1 and Server 2 act as relays, to get the packets from the client to Server 3. The MAC addresses of the used devices in this case are:

- Client MAC address: e5:88:09:4c:36:6a
- Server 1 (Relay 1) MAC address: d3:49:dc:c3:ab:ce
- Server 2 (Relay 2) MAC address: f5:e1:55:e3:85:c6
- Server 3 (Remote Server) MAC address: e3:3c:aa:9c:37:1d

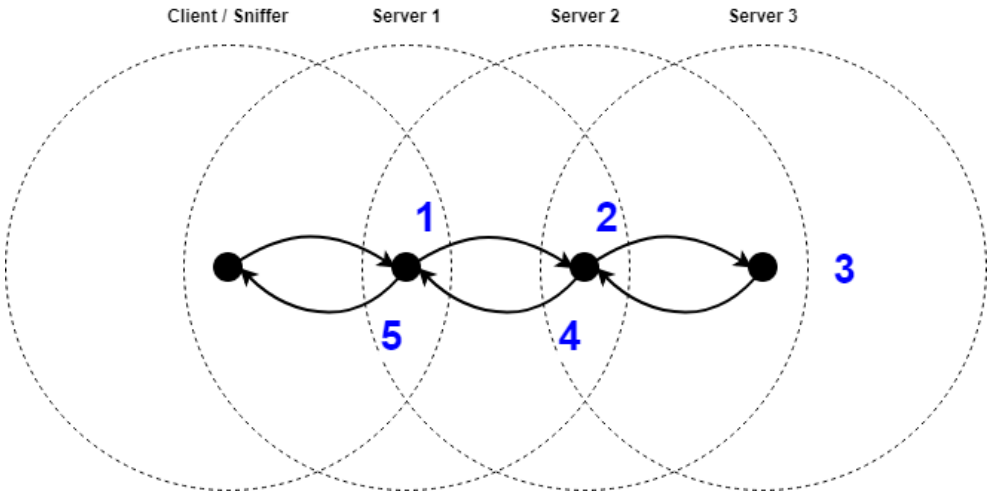


Fig. 3.13. Scenario for a 3-hop RTT calculation

In this case, the RTT is the time that takes between sending a packet from the client, relaying it through Server 1, receiving it in Server 2, relaying it again to Server 3, receiving it in Server 3, and retransmit a response back to Server 2, which will relay it again to Server 1, which will also relay it again to the client.

Therefore, the packet pattern in this scenario is the following: e5:88:09:4c:36:6a - d3:49:dc:c3:ab:ce - f5:e1:55:e3:85:c6 - e3:3c:aa:9c:37:1d - f5:e1:55:e3:85:c6 - d3:49:dc:c3:ab:ce (**Table 3.3**).

Hop number	Hop	Average Hop Time (since last hop) (ms)	Average Total time (RTT) (ms)
1	**Server 1 - Server 2**	15.13	**69.63**
2	**Server 2 - Server 3**	14.77	
3	**Server 3 - Server 2**	13.97	
4	**Server 2 - Server 1**	13.75	
5	**Server 1 - Client**	12.02	

Table 3.3. Average RTT Time for a 3-hop scenario transmission

As done with the previous analysis, the average RTT has been calculated out of 9 individual transmissions (**Fig. 3.14**).

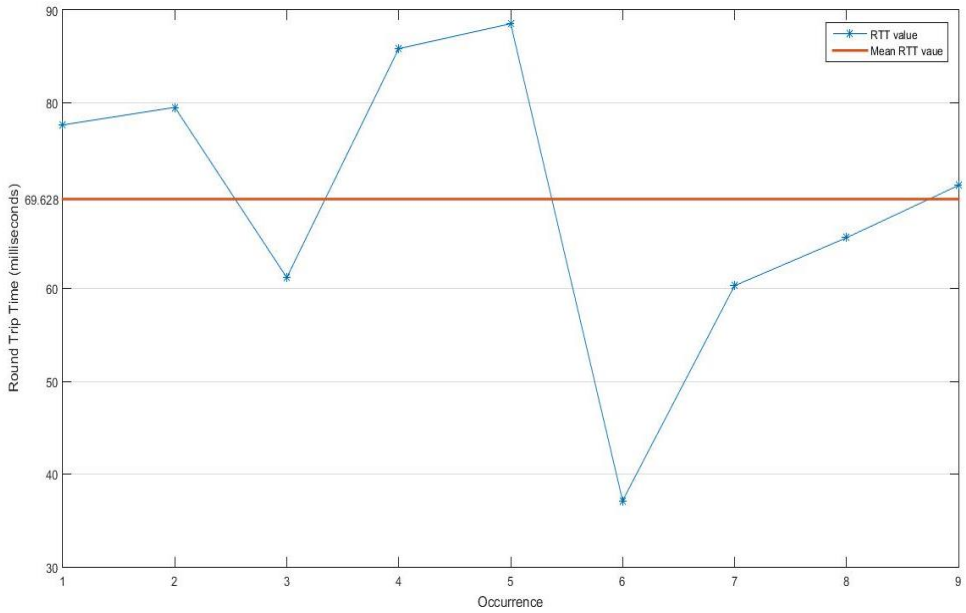


Fig. 3.14. Mean RTT value for several samples in a 3-hop RTT scenario

Again, the RTT value increases significantly. By analysing the RTT values for one, two and three hops, it can be seen that the RTT increases almost linearly with the number of hops, although such tendency is not strict, since the contribution of each hop to the RTT seems to increase a little bit for each added hop. In the next subsections a theoretical analysis to determine the RTT evolution with a greater number of hops is carried out.

3.2.2.4. More than 3 hops (theoretical analysis)

In the previous paragraphs, the results given for the RTT values are totally experimental, and done with real devices and real measurements. The purpose of this paragraph is further the analysis in order to know what would be the RTT values for more than 3 hops, by doing a theoretical measurement starting from the three values already known.

Therefore, the purpose is to predict what would be the RTT values for four, five, and even more hops. To do this, the method of second differences of quadratic sequences has been used (**Fig. 3.15**). A quadratic sequence is a sequence of numbers where the second differences between numbers are constant. This method is the used to calculate further RTT values for more hops.

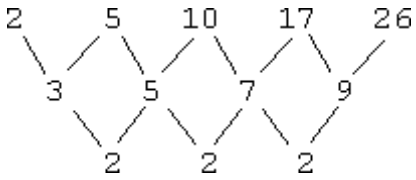


Fig. 3.15. Representation of the second differences method for quadratic sequences

As a reminder, here are the experimentally measured RTT values given for one, two and three hops (**Table 3.4**).

Number of hops (n)	RTT (ms)
1	**12.67**
2	**39.18**
3	**69.63**

Table 3.4. Average RTT times for a 1-hop, 2-hop and 3-hop scenario transmission

Taking **n** as the number of hops, the n_{th} RTT value will take form of a quadratic equation:

$$RTT(n) = an^2 + bn + c$$

Since already RTT(1), RTT(2), and RTT(3) values are known, there is a three equation system with three unknowns, that can be used to get the values of a, b and c:

$$\begin{aligned} RTT(1) &= a + b + c = \mathbf{0.01267} \\ RTT(2) &= 4a + 2b + c = \mathbf{0.03918} \\ RTT(3) &= 9a + 3b + c = \mathbf{0.06963} \end{aligned}$$

By solving this system, the following values are given:

$$\begin{aligned} a &= \mathbf{0.00196} \\ b &= \mathbf{0.02061} \\ c &= \mathbf{-0.0099} \end{aligned}$$

Now that these values are known, any RTT value for any number of hops can be predicted. It is very important to point out that the predictions that will be given for further RTT values with a greater number of hops are merely to have a general idea of the RTT value evolution, but such predictions are in no case to be taken as the real values, since this method may lack enough precision to do so. Furthermore, there are only three real values to do the measurements, and this method would work better with a greater number of real values.

In the next paragraphs, to see further examples and to have some more RTT values, this method will be applied to four and five hops.

- **4 hops**

By following the previous steps, we have that:

$$RTT(4) = 16a + 4b + c = \mathbf{0.10399}$$

Therefore, the predicted RTT value for four hops is **0.10399** seconds.

- **5 hops**

Same as before:

$$RTT(5) = 25a + 5b + c = \mathbf{0.14229}$$

Therefore, the predicted RTT value for five hops is **0.14229** seconds.

- **Further hops**

Following the same steps, many other RTT values can be calculated. As it has been seen in Chapter 2.2.9, the maximum number of hops a mesh network can support is 126. By using a MATLAB [15] script that does the previous calculations, the RTT values for all possible hops have been calculated (**Fig. 3.16**).

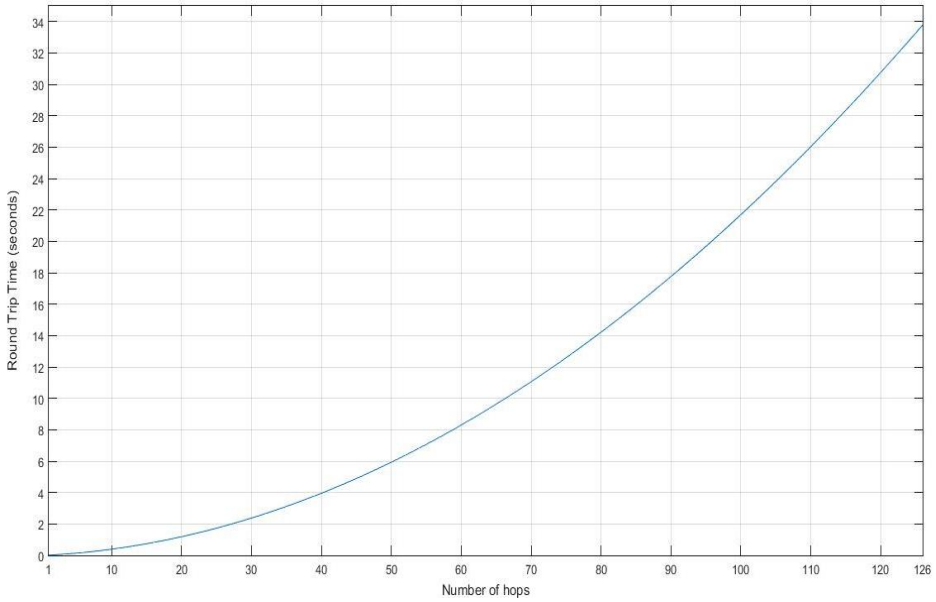


Fig. 3.16. RTT evolution for an increasing number of hops, according to the calculations using the second difference method for quadratic sequences

As expected, the RTT seems to increase exponentially with the number of hops. The RTT value for one hop is 0.01267 seconds, as has been already seen. The prediction for the RTT value with the maximum possible number of hops (126) is 33.78 seconds.

3.3. Received Signal Strength Indicator (RSSI) measurements

In order to calculate the latencies and RTT in the previous examples, we have seen that it is important that the client and the remote server(s) cannot directly see each other. This situation can be accomplished easier if the transmission power of the devices can be controlled.

The devices that are used have a range of programmable transmission powers, which are the following: -30 dBm, -20 dBm, -16 dBm, -12 dBm, -8 dBm, -4 dBm, 0 dBm and 4 dBm. The default value is 0 dBm.

This modification is possible by modifying some code in the examples, more specifically, by changing the variable *params.radio_config.tx_power*, present in the function *set_default_broadcast_configuration* in the *advertiser.c* script. The possible configurations allow a transmit power of -30 dBm, -20 dBm, -16 dBm, -12 dBm, - 8 dBm, -4 dBm, 0 dBm and 4 dBm. The default value is 0 dBm. Therefore, the possible *params.radio_config.tx_power* values are:

- *RADIO_POWER_NRF_NEG30DBM*
- *RADIO_POWER_NRF_NEG20DBM*
- *RADIO_POWER_NRF_NEG16DBM*
- *RADIO_POWER_NRF_NEG12DBM*
- *RADIO_POWER_NRF_NEG8DBM*
- *RADIO_POWER_NRF_NEG4DBM*
- *RADIO_POWER_NRF_0DBM*
- *RADIO_POWER_NRF_POS4DBM*

Furthermore, the sniffer that gets the packets with Wireshark can provide the RSSI value of the broadcast packets, and therefore, it is possible to know the signal strength of the received packets from the transmitting devices in terms of their transmit power and their distance from the receiving device.

A series of experiments have been carried out to determine the RSSI of the sniffer for a device transmitting at different transmission powers and at different distances. The purpose of this was to get a suitable transmission power for the devices and an appropriate distance to put them apart. In the following tables we can see the results of the experiment (**Table 3.5**) (**Table 3.6**).

In all of the previous experiments a transmission value of -16 dBm or -12 dBm has been chosen.

3.3.1. RSSI mean values

TX Power (dBm)	Sniffer RSSI 0.5 m (dBm)	Sniffer RSSI 2.5 m (dBm)	Sniffer RSSI 5 m (dBm)	Sniffer RSSI 10 m (dBm)	Limit (m)
-30	-90 to -87	Not detected	Not detected	Not detected	1-1.5
-20	-75 to -74	-100 to -88	-102 to -97	Not detected	5
-16	-72 to -68	-100 to -85	-99 to -93	-100 to -93*	7-10
-12	-67 to -66	-93 to -82	-94 to -86	-100 to -93**	8 -10
-8	-64 to -61	-100 to -80	-100 to -89	-100 to -92	> 10

-4	-64 to -57	-94 to -77	-100 to -90	-97 to -87	> 10
0	-57 to -52	-88 to -73	-100 to -82	-96 to -84	> 10
4	-54 to -48	-98 to -68	-88 to -77	-100 to -77	> 10

Table 3.5. RSSI mean values in the sniffer for different transmit powers and distances

3.3.2. RSSI median

TX Power (dBm)	Sniffer RSSI 0.5 m (dBm)	Sniffer RSSI 2.5 m (dBm)	Sniffer RSSI 5 m (dBm)	Sniffer RSSI 10 m (dBm)	Limit (m)
-30	-88	Not detected	Not detected	Not detected	1-1.5
-20	-74	-95	-99.5	Not detected	5
-16	-70	-92	-95	-99*	7-10
-12	-66	-85	-91	-97**	8 -10
-8	-62	-87	-92.5	-98	> 10
-4	-59	-82	-92.5	-92	> 10
0	-56	-76	-89	-90.5	> 10
4	-53	-74	-80	-82	> 10

Table 3.6. RSSI median values in the sniffer for different transmit powers and distances

** 7-10 meters approx. Barely detected at 10 meters*

*** 8-10 meters approx. Barely detected at 10 meters*

3.4. Current consumption and battery lifetime measurements

This section presents the results of several experiments regarding current consumption and battery lifetime of the devices that have been used in this thesis to crate Bluetooth Mesh networks.

The purpose of the experiment is to generate a current consumption model for Low Power devices, in the time when they are advertising, scanning or in sleep mode. For both advertising and scanning, a thorough analysis has been done to identify each state and both its current consumption and its time period. Since the Low Power feature is not yet supported by the platform used in the project (and we are neither aware of any other platform currently supporting it), the

experiments emulate the behaviour of the Low Power node role as per the Bluetooth Mesh specification.

Furthermore, battery lifetime has been calculated, as well as its variation in value under several circumstances.

The hardware setup for the experiments is the following:

- N6705A DC power analyser [16] (**Fig. 3.17**). This device can store the data into files, which will later be used to make the current consumption figures and analysis, and it can also make screen captures.
- NRF51422 Nordic Development Kit, PCA10028 board. It is running different programs to measure current consumptions.

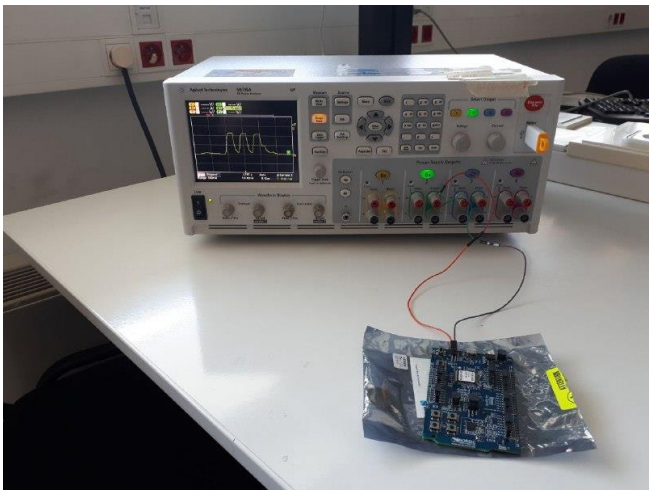


Fig. 3.17. Current consumption calculation setup

The board is powered by using the External Supply pins (**Fig. 3.18**), which are connected to one of the power supply outputs of the power analyser (**Fig. 3.19**). The supply voltage is set to 3 V, a chosen value within the supply range of the board.

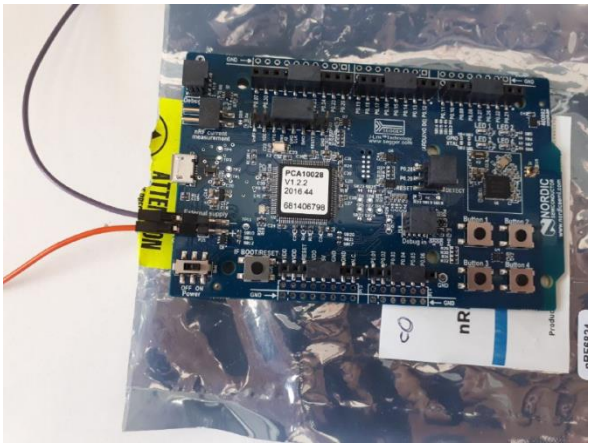


Fig. 3.18. Supply Voltage and ground wires connected from the power analyser to the DK External Supply pins (1)

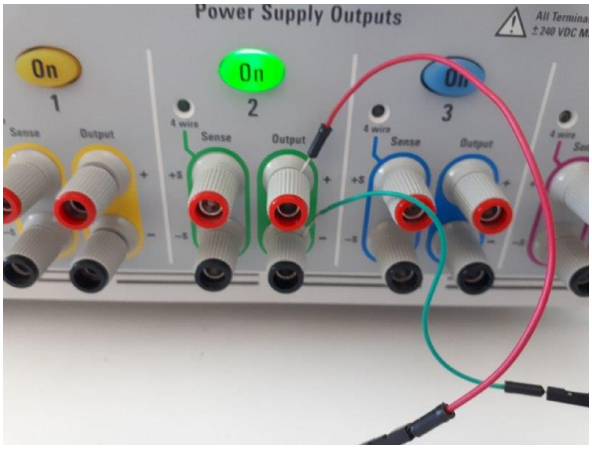


Fig. 3.19. Supply Voltage and ground wires connected from the power analyser to the DK External Supply pins (2)

3.4.1. First experiment

The first experiment used the *pb_remote* example, seen before in Chapter 3.2.1, which it was used to calculate one-way latency values. The purpose of this first experiment we just try to calculate the current consumption for a Development Kit that is advertising every ten seconds (**Fig. 3.20**).

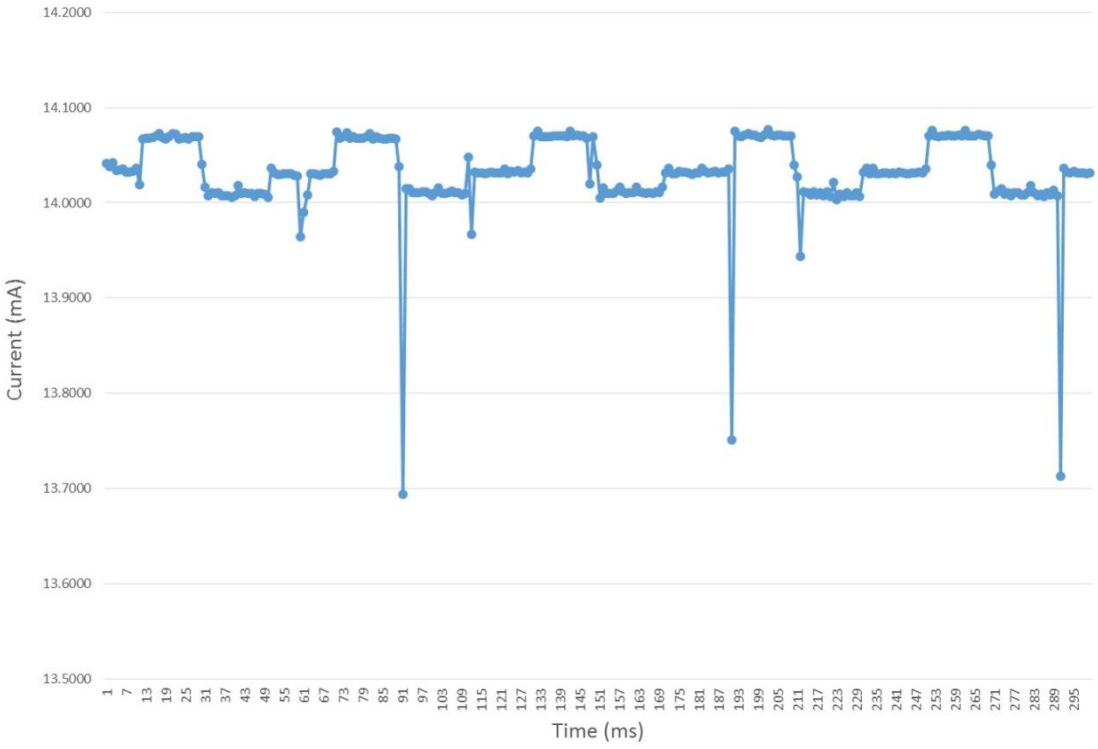


Fig. 3.20. Pb_remote example current consumption profile

As can be seen, the average current consumption of the device is around 14 mA. Every ten seconds there is a down peak in the current consumption, which corresponds to when the device sends and advertising packet. Therefore, it can

be seen that, in normal circumstances, the whole device current consumption is greater when it is not advertising. This is because when the device is not advertising it is in receive mode, while when it advertises it is in transmit mode, and it never enters sleep mode.

According to the product specification, the device consumes a peak current of 13 mA when in receive mode, and 10.5 mA when it is in transmit mode. Therefore, it is logical to see these down peaks when it is advertising. Therefore, this result will not be taken for further analysis for current consumption and battery lifetime, since the purpose is to know the sleep mode current and battery lifetime for Low Power devices. This will be seen in the second experiment.

3.4.2. Second experiment: Low Power node advertising

In this second experiment the purpose was to get a device to advertise every certain period of time, in order to get the current consumption of the device in the event of advertising. This experiment has been carried out using the *ble_app_pwr_profiling* example that is found in the Nordic BLE SDK. An analysis has been made in order to calculate the current and the time for each step of the advertising.

From the figures it can be seen that there are three current consumption peaks, due to the fact that the device advertises one time in each advertising channel (channels 37, 38 and 39). In this example, the sender is able to send both connectable and non-connectable advertisements. The final analysis focuses on non-connectable advertisements. However, both advertisement types are reviewed and explained in the following subsections.

3.4.2.1. Connectable advertisements

In this *ble_app_pwr_profiling* example, the device first advertises **connectable advertisements** every 20 ms. Connectable advertisements turn the radio in receive mode after transmitting. This is why a peak in current consumption can be seen after transmission for each of the three channels (**Fig. 3.21**).

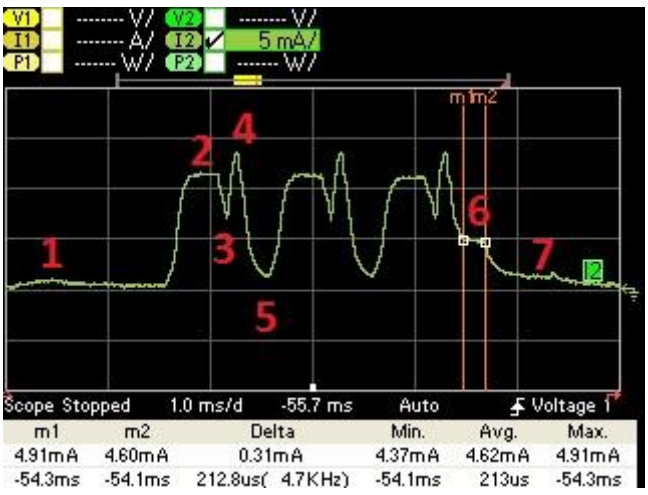


Fig. 3.21. Current consumption states for a connectable advertisement

The following table shows the different states of the approximate current consumption and times for the connectable advertisement (**Table 3.7**).

State Number	Description	Duration		Current Consumption	
		Variable	Value (ms)	Variable	Value (mA)
1	wake-up & radio preparation	T_{wu}	1.6	I_{wu}	0.3
2	transmission	T_{tx}	0.5277	I_{tx}	9.06
3	radio off (TX)	T_{offtx}	0.1	I_{offtx}	7
4	radio on (RX)	T_{offrx}	0.2628	I_{offrx}	13.25
5	channel change	T_{ch}	0.3	I_{ch}	3.66
6	radio off	T_{off}	0.29	I_{off}	6
7	post-processing	T_{post}	0.62	I_{post}	1.6
8	sleep	T_{sleep}	Variable	I_{sleep}	0.015

Table 3.7. Current consumption state table for connectable advertisements

Note that the previous table is not considered for the final current consumption and battery lifetime measurements, since the non-connectable advertisement state number table that will be seen next will be used for that.

3.4.2.2. Non-connectable advertisements

After the transmission of connectable advertisements for a certain period of time, the *ble_app_pwr_profiling* example begins transmitting **non-connectable** advertisements. In this type of advertisements, the device does not enable the radio to listen for incoming packets after transmission, and this is why in the figures we will see there is not the peak that was seen for the connectable advertisement figures. This is the kind of advertisements that will be used for further analysis on current consumption and battery lifetime of Low Power devices.

A table with the state numbers indicating each of the steps of the advertising is done, and in it, the times and currents for each state will be calculated (**Table 3.8**). States are shown in the following figures (**Fig. 3.22**) (**Fig. 3.23**).

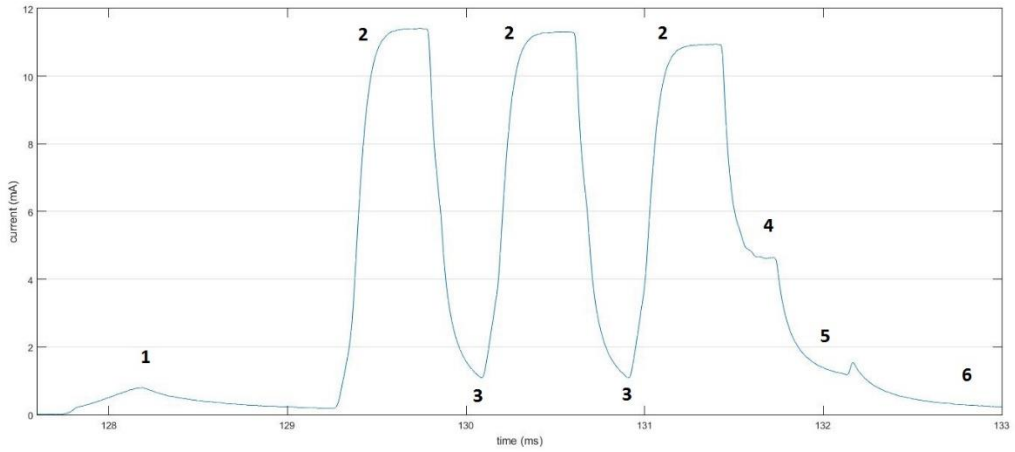


Fig. 3.22. Current consumption states for a non-connectable advertisement (1)

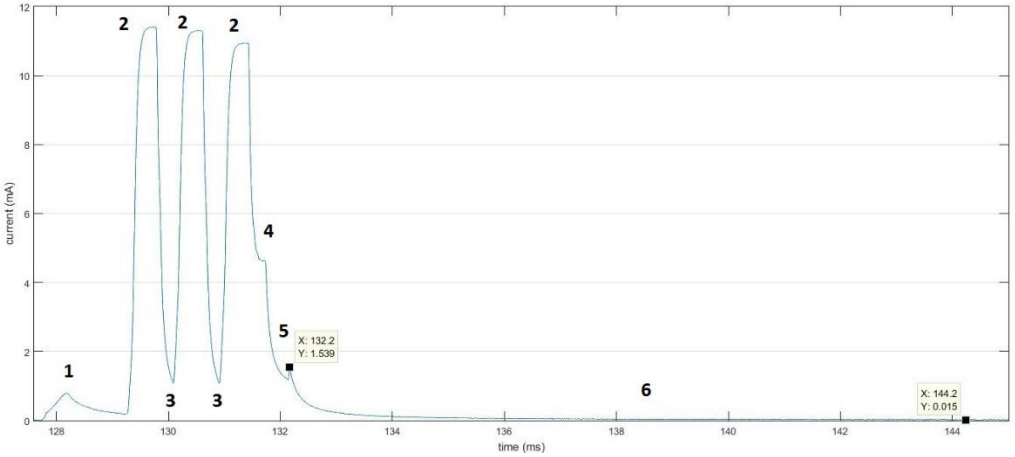


Fig. 3.23. Current consumption states for a non-connectable advertisement (2)

To calculate the times and current consumption of each state, several advertisements for different experiments have been examined, and the final result is the average of these several results.

The current consumption figures have been obtained from a data file created by the power analyser. Therefore, each point in time has a current value. With this, in order to know the average current consumption for each one of the states, a MATLAB function has been developed in order to calculate the average current value between two points in time.

State Number	Description	Duration		Current Consumption	
		Variable	Value (ms)	Variable	Value (mA)
1	wake-up & radio preparation	T_{wu}	1.51	I_{wu}	0.38
2	transmission (Ch. 37,38,39)	T_{tx}	0.52 0.53 0.53	I_{tx}	8.45 8.85 8.71
3	channel change (Ch 37 to 38, Ch. 38 to 39)	T_{ch}	0.30 0.33	I_{ch}	3.66 3.91
4	radio off	T_{off}	0.29	I_{off}	5.48
5	post-processing	T_{off}	0.62	I_{off}	1.67
6	cooldown	T_{cool}	14.00	I_{cool}	0.074
7	sleep	T_{sleep}	Variable	I_{sleep}	0.015

Table 3.8. Current consumption state table for non-connectable advertisements

After analysing the output data files from the power analyser, average sleep power consumption is considered to be 15 µA.

3.4.3. Third experiment: Low Power node scanning

Next experiment is regarded to the current consumption of the Low Power device when it is configured to be an “observer”. An observer is usually in sleep mode, and it wakes up every certain period of time to turn into receive mode. The period of time in which the device is scanning is called the scan window, while the time between two consecutive scans is the scan interval.

In order to do the current consumption analysis of a scanning device, two SDK examples, *ble_app_hrs_c* and *experimental_ble_app_multiactivity_beacon*, are used. Both examples configure the device to act as an observer.

Then, the current consumption of the device has been measured with the power analyser.

Again, with the help of MATLAB, some figures have been taken from the data files from the power analyser (**Fig. 3.24**), and the state number table has been calculated (**Table 3.9**). The final values from the state number table have been calculated by analysing several scanning intervals and by applying the average current consumption values for each state.

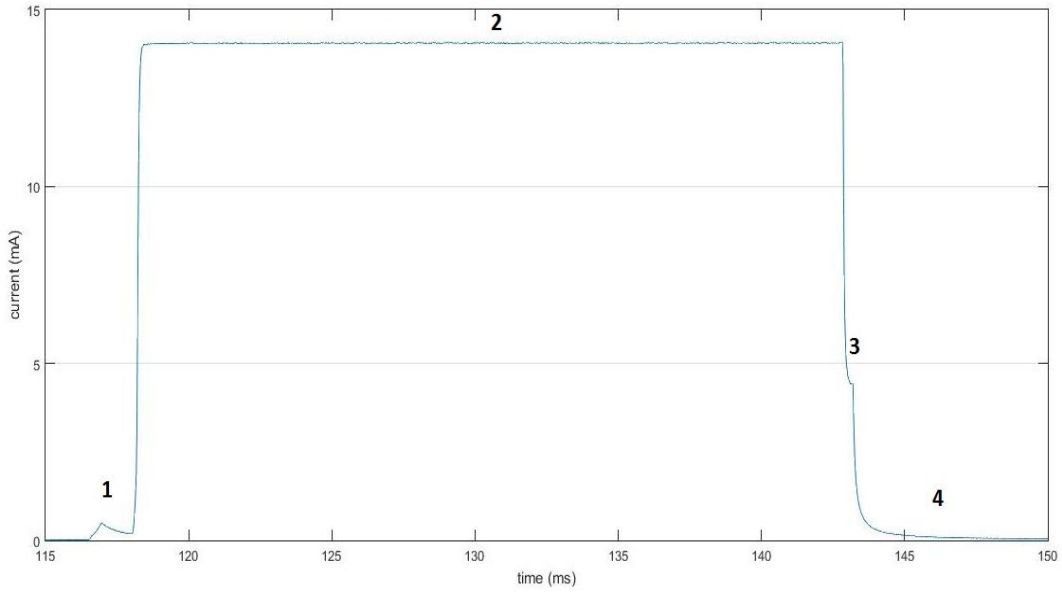


Fig. 3.24. Current consumption states for a scanning event

State Number	Description	Duration		Current Consumption	
		Variable	Value (ms)	Variable	Value (mA)
1	wake-up & radio preparation	T_{wu}	1.573	T_{wu}	0.336
2	scanning	T_{sc}	Variable	T_{sc}	13.9
3	radio off	T_{off}	0.319	T_{off}	6.439
4	post-processing & cooldown	T_{post}	26.322	T_{post}	0.07
5	sleep	T_{sleep}	Variable	T_{sleep}	0.015

Table 3.9. Current consumption state table for scanning events

3.4.4. Average current consumption and battery lifetime measurements for a Low Power node

The purpose of this part of the project is to do an analysis on the current consumption and battery lifetime of Low Power nodes as a function of different parameters. A Low Power node is a device that relies on a friend node to receive its packets, so that, periodically, it only has to wake up for a certain period of time to request information to such friend node, and to get a response back, in case there is any. The rest of the time, the Low Power node can be put to sleep, and

thus consuming much less current, and increasing its battery lifetime substantially (**Fig. 3.25**).

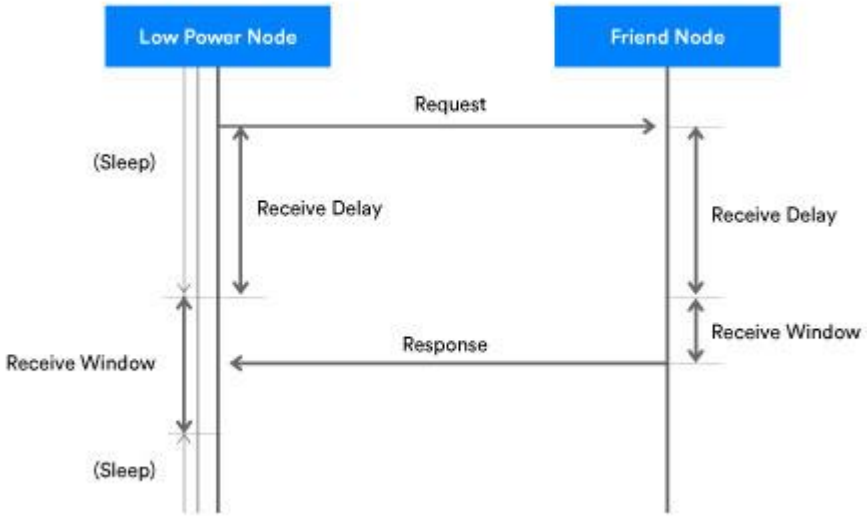


Fig. 3.25. Low Power and Friend node Request/Response behaviour [5]

In order to do the average current consumptions and battery lifetime of the devices, the Low Power behaviour has been considered (**Fig. 3.26**). This can be considered the Low Power node behaviour of advertising plus scanning, with the difference that the advertisements are Non-connectable advertisements, as opposed to the figure, that shows Connectable advertisements. As can be seen in the figure, first the device sends its Request (via Non-connectable advertisements), then it waits for a certain period of time called the ReceiveDelay, where it stays in sleep mode, and then it wakes up to stay scanning (for potentially incoming packets) during a certain period of time called ReceiveWindow. The connection between the Low Power node and its friend has to be maintained within a time controlled by the PollTimeout timer. The lower the PollTimeout is, the more advertising and scanning will be made, and thus, the more current consumption and the less battery lifetime the node will have.

The ReceiveWindow value is also highly important with regard to current consumption. The greater this value is, the more the device will consume its battery as well.

Therefore, it is expected that both the PollTimeout and ReceiveWindow values play a major role in the device current consumption and battery lifetime. In the next pages an analysis has been carried out to determine different outcomes in these two aspects for different PollTimeout and ReceiveWindow values.

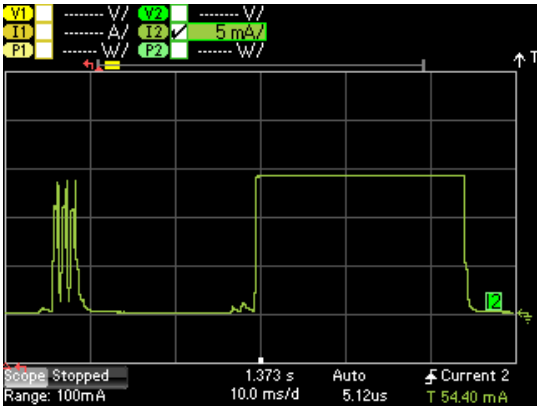


Fig. 3.26. Low Power node current consumption profile considered for the current consumption and battery lifetime analysis

According to the Bluetooth Mesh specification:

- The **ReceiveDelay** is the time between the Low Power node sending a request and listening for a response. This delay allows the Friend node some time to prepare the response.
- The **ReceiveWindow** is the time that the Low Power node listens for a response. When the Low Power node receives a message from its Friend node, it can stop listening for additional messages. The values can range from 1 ms to 255 ms.
- The request can be a Friend Poll message, a Friend Subscription List Add message, or a Friend Subscription List Remove message. The response to a Friend Poll message can be a Friend Update message or a stored message. The response to a Friend Subscription List Add message or a Friend Subscription List Remove message is a Friend Subscription List Confirm message.
- **PollTimeout** timer is used to measure time between two consecutive requests sent by the Low Power node (**Fig. 3.27**). If no requests are received by the Friend node before the PollTimeout timer expires, then the friendship is considered terminated. The values can range from 1 second to 345599.9 seconds (4 days).

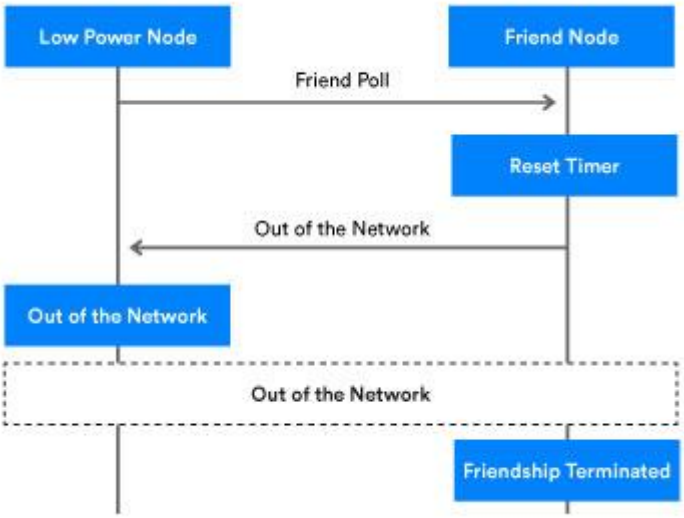


Fig. 3.27. Low Power node current consumption profile considered for the current consumption and battery lifetime analysis [5]

In earlier parts of this project, both connectable, non-connectable advertisements and scanning events have been analysed separately. Taking both information from tables 3.8 (**Table 3.8**) and 3.9 (**Table 3.9**), corresponding to the states for scanning and non-connectable advertisement events, the following table has been created to depict the current consumption states of the Low Power node (**Table 3.10**). This table has been used to carry out the average current consumption and battery lifetime calculations.

State Number	Description	Duration		Current Consumption	
		Variable	Value (ms)	Variable	Value (mA)
1	wake-up & radio preparation (advertising)	T_{wuadv}	1.51	I_{wuadv}	0.38
2	transmission (Ch. 37,38,39)	T_{tx}	0.52 0.53 0.53	I_{tx}	8.45 8.85 8.71
3	channel change (Ch 37 to 38, Ch. 38 to 39)	T_{ch}	0.30 0.33	I_{ch}	3.66 3.91
4	radio off (advertising)	T_{offadv}	0.29	I_{offadv}	5.48
5	post-processing (advertising)	$T_{postadv}$	0.62	$I_{postadv}$	1.67
6	cooldown	T_{cool}	14.00	I_{cool}	0.074

7	sleep (ReceiveDelay)	T_{rd}	Variable	I_{rd}	0.015
8	wake-up & radio preparation (scanning)	T_{wusc}	1.573	I_{wusc}	0.336
9	scanning (ReceiveWindow)	T_{rw}	Variable	I_{rw}	13.9
10	radio off (scanning)	T_{offsc}	0.319	I_{offsc}	6.439
11	post-processing & cooldown (scanning)	T_{postc}	26.322	I_{postc}	0.07
12	sleep	T_{sleep}	Variable	I_{sleep}	0.015

Table 3.10. Current consumption state table for a Low Power node

Several average current consumptions analysis have been carried out. The experiment has taken the PollTimeout and the ReceiveWindow as variables to calculate the different current consumptions of the Low Power node. To do the calculations and the plots, MATLAB has been used.

3.4.4.1. Average current consumption

In order to calculate the average current consumption for each PollTimeout and Receive Window, the values from the state **Table 3.10** have been considered. To calculate the overall average current consumption for each case, the following expressions have been used:

$$I_{avg} = \sum_{k=1}^{n} I_k \cdot \frac{T_k}{T_{total}}$$

Where:

I_{avg}: Average total current consumption of the Low Power node for one interval of time (advertising and scanning).
n: Number of states.
I_k: Average current consumption in state k.
T_k: Time period for state k.
T_{total}: Total time for one interval of advertising and scanning (PollTimeout).

$$T_{total} = T_{wuadv} + T_{tx} + T_{ch} + T_{offadv} + T_{postadv} + T_{cool} + T_{rd} + T_{wusc} + T_{rw} + T_{offsc} + T_{postsc} + T_{sleep}$$

$$T_{total} = PollTimeout$$

Fig. 3.28 shows graphically the results of the analysis.

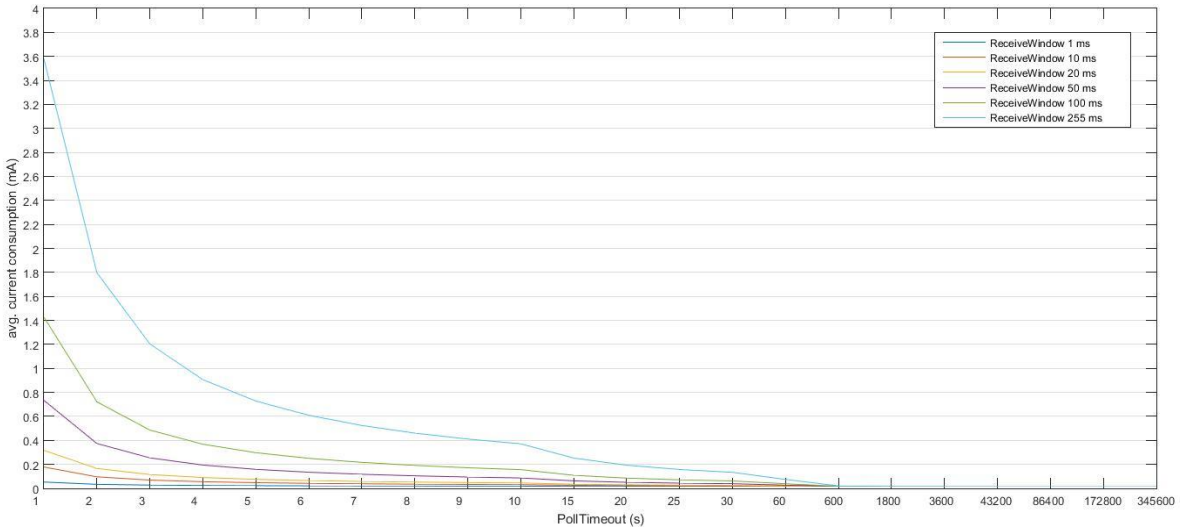


Fig. 3.28. Average current consumption of a Low Power node for several PollTimeout and ReceiveWindow values

As can be seen, doubling the PollTimeout value decreases the current consumption in (almost) half. In the same way, reducing the ReceiveWindow value in half also decreases the average current consumption in (almost) half. For high PollTimeout values, the average current consumption tends to the sleep current consumption value (I_{sleep}) that is 15 µA. This could be considered for PollTimeout values of 600 seconds or larger. It can be seen that in the worst case, with a 255 ms ReceiveWindow and a 1 second PollTimeout, the device would be consuming near to 3.6 mA in average for each transmission-scanning event. However, such high value can be easily reduced as shown by the figure.

3.4.4.2. Battery lifetime

To calculate the battery lifetime (**Fig. 3.29**), the previous average current consumption results have been considered. The battery considered is the Energizer CR2032 [17], which has a nominal voltage of 3 V, and a typical capacity of 235 mAh.

Therefore:

$$B_{lifetime} = \frac{TypicalCapacity}{I_{avg}}$$

Being:

$B_{lifetime}$: The battery lifetime.
$TypicalCapacity$: The typical capacity of the battery (235 mAh)
I_{avg}: The average current consumption, as calculated previously in the average current consumption measurements.

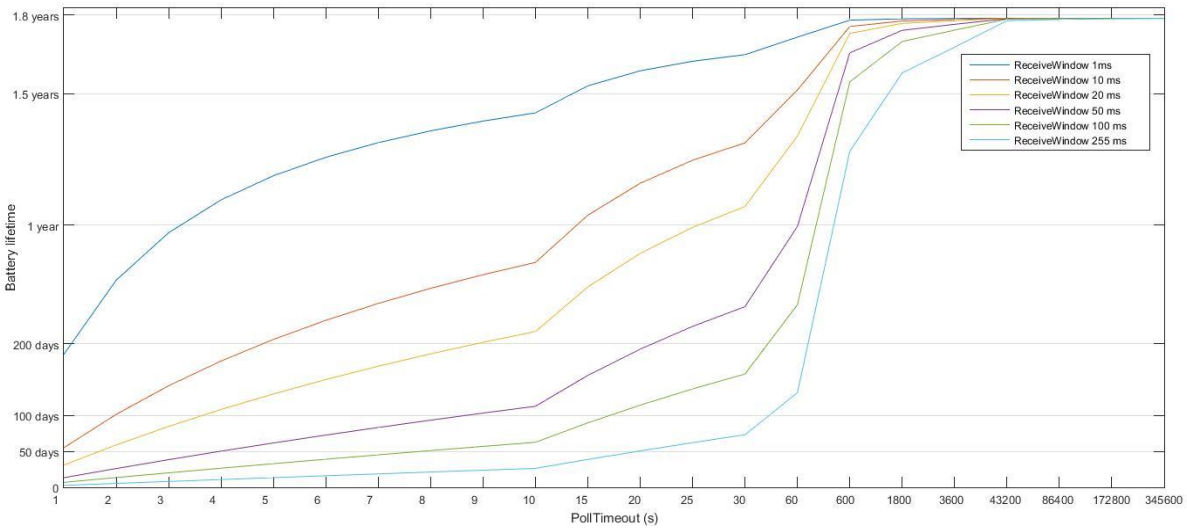


Fig. 3.29. Average battery lifetime of a Low Power node for several PollTimeout and ReceiveWindow values

As can be seen in the figure, higher PollTimeout values, as well as a smaller ReceiveWindow both mean a larger battery lifetime. The maximum battery lifetime that can be achieved is 1.8 years. This situation happens when the average current consumption is very close to the sleep current consumption value (I_{sleep}) that is 15 µA. In the worst case (255 ms ReceiveWindow and 1 second PollTimeout), the battery lifetime is 2.73 days.

It can be seen that for small ReceiveWindow values this optimal battery lifetime is achieve with a much shorter PollTimeout value (60 seconds), whereas for higher ReceiveWindows, much larger PollTimeout values are needed to compensate (i.e. 30 hours for a 255 ms ReceiveWindow).

4. CONCLUSIONS AND FUTURE WORK

This chapter provides the main conclusions from this project, as well as the future work that could be done to further extend the contents of this thesis.

In addition, this chapter also includes some thoughts on the sustainability of the project, as well as some ethical considerations.

4.1. Conclusions

As mentioned at the beginning of this thesis, the main objectives of this work are to introduce the BLE and Bluetooth Mesh technologies, and then carry out several experiments to determine Bluetooth Mesh performance regarding RTT and one-way latency delays, and current consumptions for a Low Power node.

RTT and one-way latency measurements have demonstrated that increasing the number of hops increases the overall RTT and one-way latency delays in a non-linear way, and thus, the delay cost for each added hop is greater every time. This could be considered the expected behaviour, since each added hop is one more added device, larger overall processing times, a more difficult path for the messages to travel through, and more factors in general that can affect the delays. Processing times from the devices may have a great impact on these delays, and therefore, doing the same experiment in different device models may change the results notably.

For these RTT and one-way latency measurements, it has been very important to measure the same message exchange scenario several times, since, although the hop time values were quite consistent between experiments, tending to a common value of around 15 ms, they still may fluctuate notably enough. Therefore, it was important to obtain a mean value for the highest possible number of samples. Furthermore, as already seen, sometimes there are peaks in such time values, that although they are not very frequent, they may contribute to substantially increase the overall final time value. Some of these samples were discarded because it was considered they did not reflect the realistic, usual and normal delays of the message exchanges, and they may be erroneous.

From the battery lifetime calculations in this document, it can be seen that the maximum achievable value calculated is around 1.8 years, which may seem a bit low, since other similar measurements have determined battery lifetime expectancies of over 10 years [1]. One of the reasons may be that the average sleep-mode current consumption in the devices used in this thesis seems to be quite notable (15 µA). For example, BLE platforms (that do not support Bluetooth Mesh) offer sleep current values around 1 µA. As can be expected, a lower sleep-mode current would increase battery lifetime and decrease current consumption.

Apart from this, the current consumption and battery lifetime figures show the expected behaviour. A smaller ReceiveWindow and a bigger PollTimeout value both contribute to a much less energy-intensive behaviour, resulting in much lower current consumption and a much bigger battery lifetime. In the same way,

a big ReceiveWindow and a small PollTimeout value result in a much higher current consumption and a much shorter battery lifetime.

To further extend the conclusions of this project, some sustainability and environmental awareness thoughts, as well as some ethical considerations, are overviewed in the following subsections.

4.1.1. Sustainability and environmental awareness

The work done in this thesis is considered to have little impact on the environment. The small impact that could be considered comes from the use of software and hardware, and their related electricity consumption and CO_2 emissions, and the use of coin cell batteries. Economic impact comes mainly from the cost of buying of the devices. For experiments with bigger networks and a higher number of devices, such impact may increase.

4.1.2. Ethical considerations

As has already been explained in Chapter 1, BLE and Bluetooth Mesh include several security measures including data privacy. Therefore, the use of these technologies should not be a threat in this aspect.

IoT networks, to which Bluetooth technology plays an important role, may come with other ethical issues such as [18]:

- Ubiquity of technologies: making them omnipresent and invasive.
- Miniaturization: IoT devices may be small, transparent, and thus could easily avoid inspection and control.
- Ultra-connectivity: IoT may result in a huge amount of data exchange, which could be used maliciously.
- Difficult control: Since IoT control is not centralized, control will be much more difficult, and thus will require control and governance to emerge.

4.2. Future Work

In this thesis, the Low Power node current consumption had to be emulated by calculating current consumptions for advertising and scanning events similar to those of the Low Power node, because the devices used still do not support the Low Power node feature nor support the concept of friendship. Therefore, a main subject for future work could be to do these experiments in a device that is really configured as a Low Power node. Then, the relationship between this node and its friend node could be further analysed.

Another aspect that could be expanded would be to carry out the RTT and one-way latency delays with a greater number of devices, in order to have more real multihop scenarios with a greater number of hops.

REFERENCES

[1] C. Gomez, J. Oller, J. Paradells. (2012). Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. *Sensors*. **pp.** 11735 - 11741

[2] Mesh Working Group. (2017). Mesh Profile, Bluetooth Specification v 1.0. 16-35.

[3] Nordic Semi Infocenter. (2017, July 19). *Basic Bluetooth Mesh concepts*. Retrieved from https://infocenter.nordicsemi.com

[4] S. M. Darroudi, C. Gomez, (June, 2017). Bluetooth Mesh: a Standard for Bluetooth Low Energy Mesh Networks: a Survey. *Sensors*.

[5] K. Kolderup. (2017, July 18). *Introducing Bluetooth Mesh Networking*. Retrieved from http://blog.bluetooth.com/introducing-bluetooth-mesh-networking

[6] Nordic Semiconductor. (2014). nRF51 Development Kit User Guide v1.0.

[7] Nordic Semiconductor. (2014). nRF51 Dongle User Guide v1.0.

[8] Nordic Semi Infocenter. (2018, July 24). *nRF5 SDK for Mesh v2.1.1 documentation.* Retrieved from http://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk%2Fdita%2Fsdk%2Fmesh_sdk.html

[9] Nordic Semi Infocenter. (2018, March 4). *nRF5 SDK v15.0.0 documentation.* Retrieved from https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v15.0.0%2Findex.html

[10] Nordic Semi Infocenter. (2016, March 10). *nRF5 SDK v11.0.0 documentation* Retrieved from https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Findex.html.

[11] SEGGER. (2018). *SEGGER Microcontroller GmbH*. Retrieved from https://www.segger.com/products/development-tools/embedded-studio/

[12] ARM Keil. (2018). *ARM Keil µVision® IDE*. Retrieved from http://www2.keil.com/mdk5/uvision/

[13] Nordic Semiconductor. (2018). nRF Sniffer User Guide v2.1.

[14] Wireshark. (2018). *Wireshark*. Retrieved from https://www.wireshark.org/

[15] Mathworks. (2018). *MATLAB*. Retrieved from https://es.mathworks.com/products/matlab.html

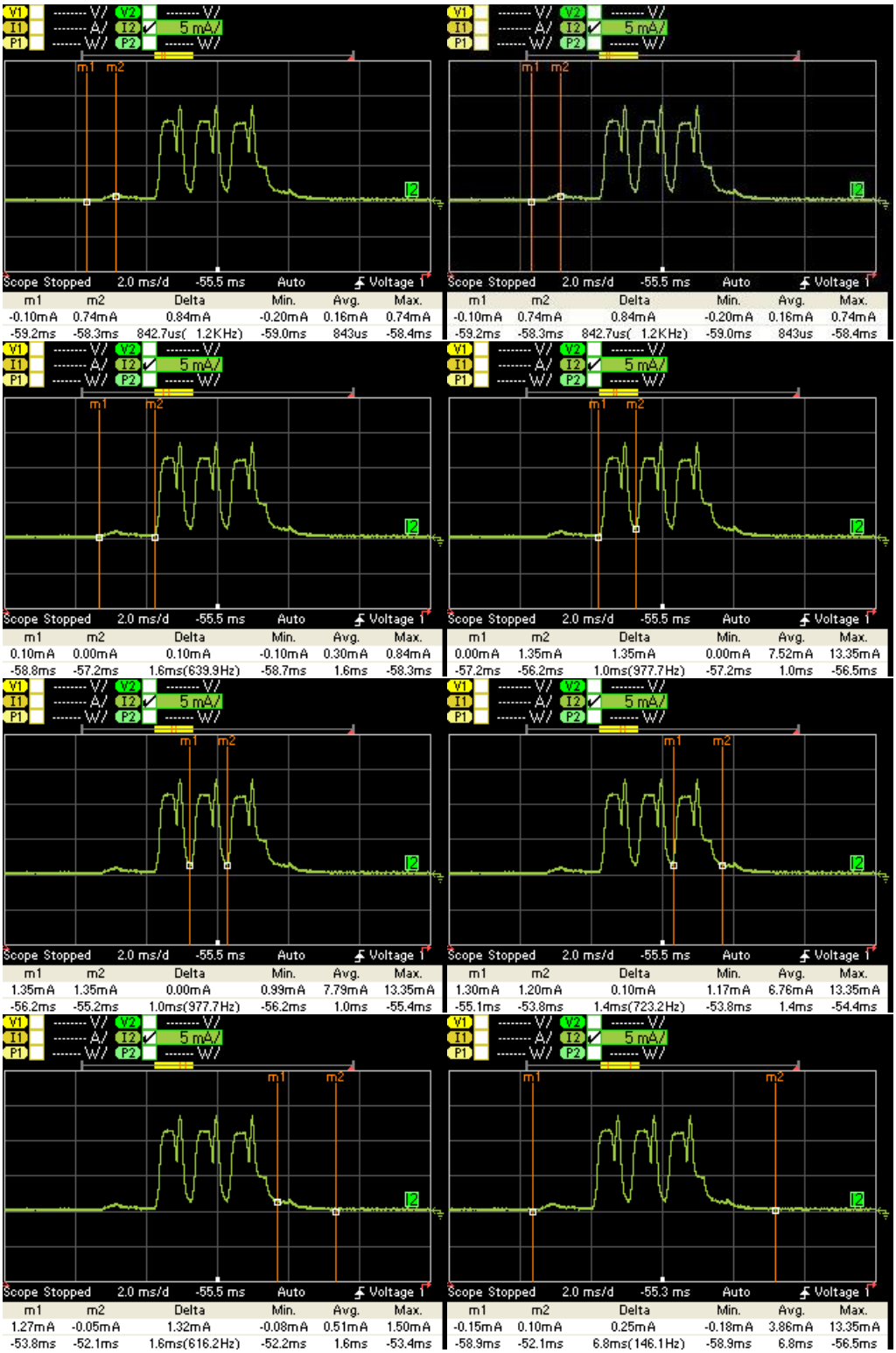
[16] Agilent Technologies. (2010, July 8). Agilent DC Power Analyzer Technical Oveview. Retrieved from https://cdn.testequity.com/documents/pdf/N6705A-overview.pdf

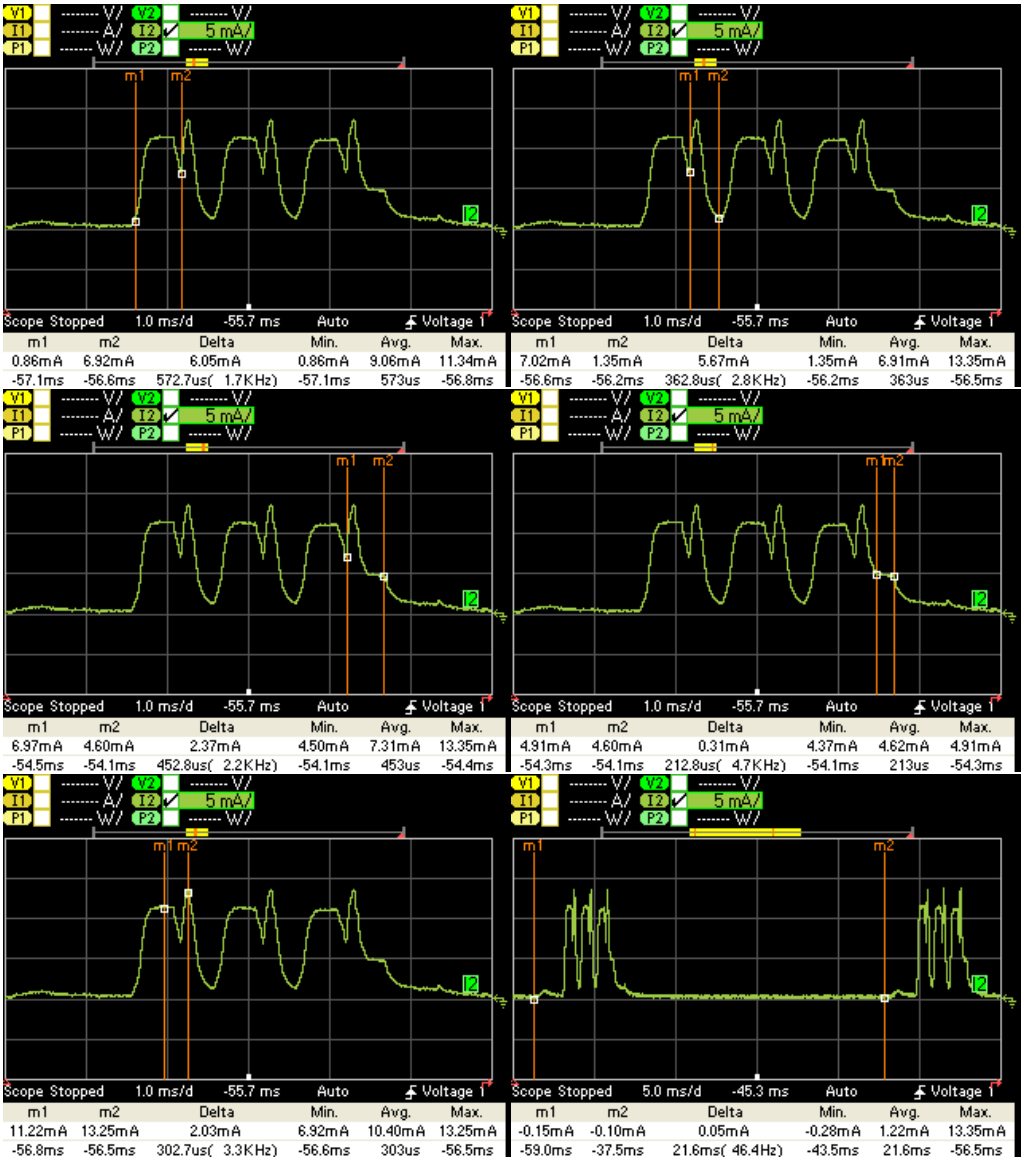
[17] Energizer. (s.f.). ENERGIZER CR2032 product datasheet.Retrieved from http://data.energizer.com/pdfs/cr2032.pdf

[18] D. Popescul, M. Georgescu, (2013, December). *INTERNET OF THINGS – SOME ETHICAL ISSUES.* Retrieved from ResearchGate: https://www.researchgate.net/publication/260290933_Internet_Of_Things-Some_Ethical_Issues

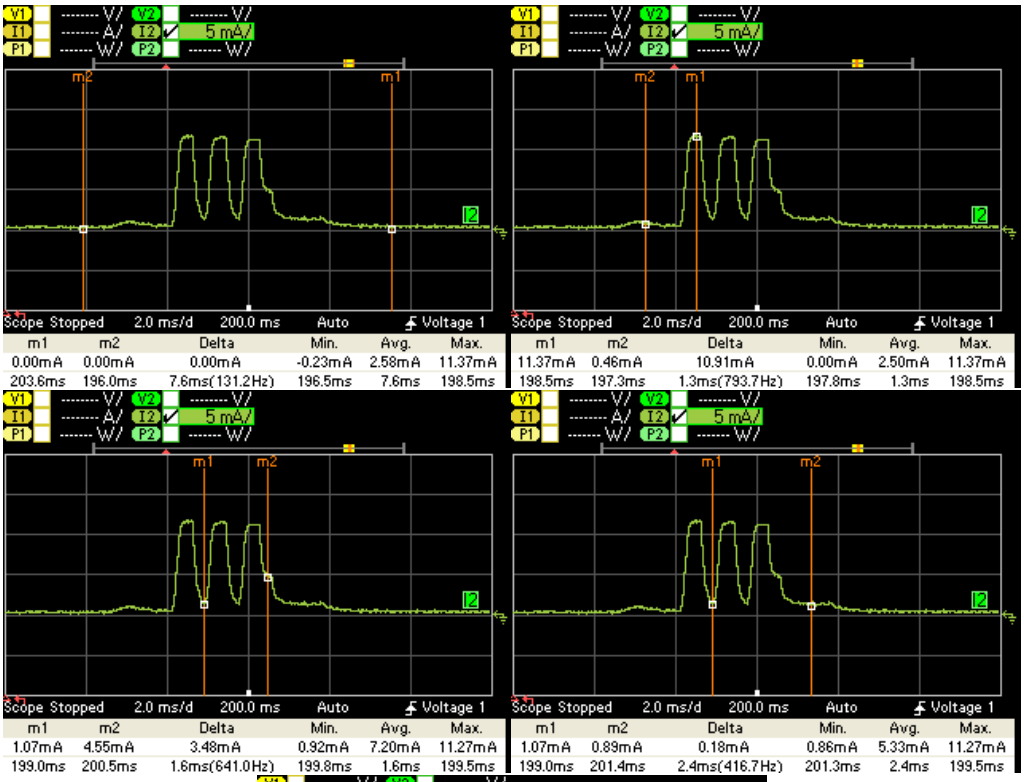
ANNEXES

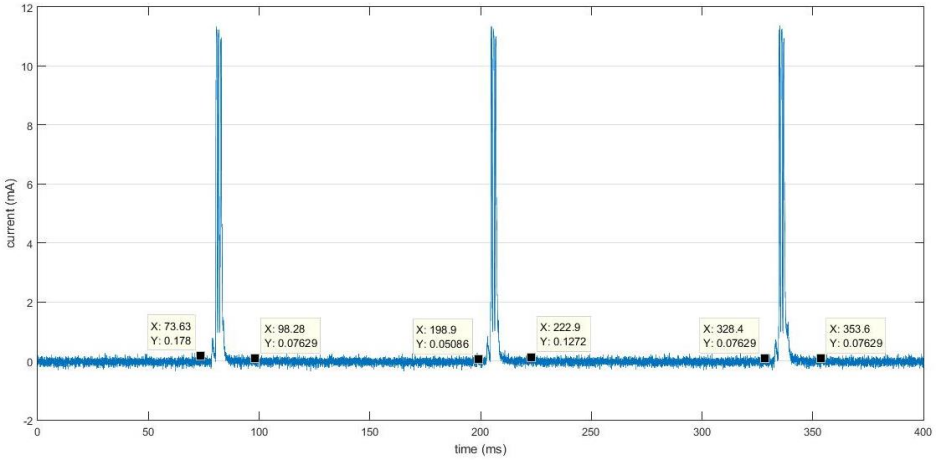
I. Connectable advertisements current consumption captures

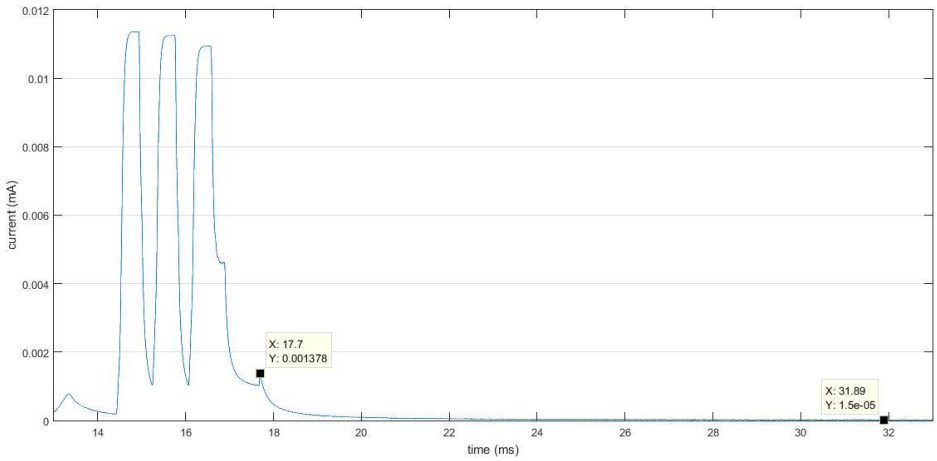


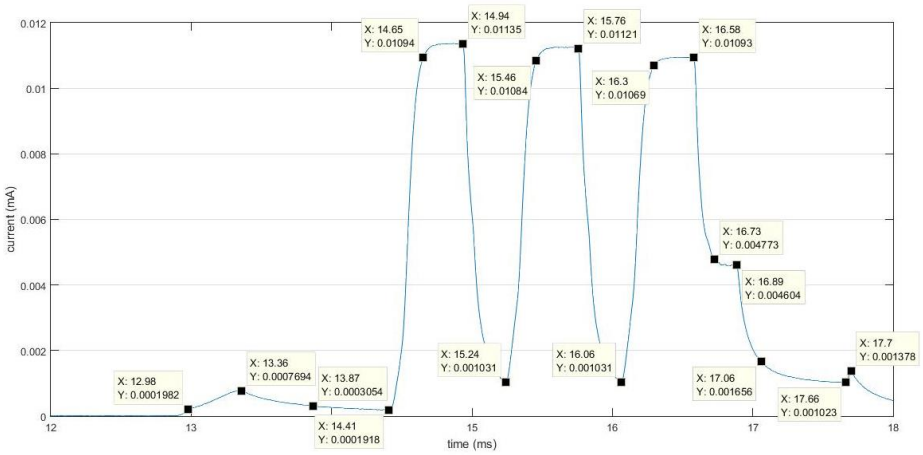

Scope Stopped 1.0 ms/d -55.7 ms Auto Voltage 1
m1 m2 Delta Min. Avg. Max.
0.86mA 6.92mA 6.05mA 0.86mA 9.06mA 11.34mA
-57.1ms -56.6ms 572.7us(1.7KHz) -57.1ms 573us -56.8ms
Scope Stopped 1.0 ms/d -55.7 ms Auto Voltage 1
m1 m2 Delta Min. Avg. Max.
7.02mA 1.35mA 5.67mA 1.35mA 6.91mA 13.35mA
-56.6ms -56.2ms 362.8us(2.8KHz) -56.2ms 363us -56.5ms
Scope Stopped 1.0 ms/d -55.7 ms Auto Voltage 1
m1 m2 Delta Min. Avg. Max.
6.97mA 4.60mA 2.37mA 4.50mA 7.31mA 13.35mA
-54.5ms -54.1ms 452.8us(2.2KHz) -54.1ms 453us -54.4ms
Scope Stopped 1.0 ms/d -55.7 ms Auto Voltage 1
m1 m2 Delta Min. Avg. Max.
4.91mA 4.60mA 0.31mA 4.37mA 4.62mA 4.91mA
-54.3ms -54.1ms 212.8us(4.7KHz) -54.1ms 213us -54.3ms
Scope Stopped 1.0 ms/d -55.7 ms Auto Voltage 1
m1 m2 Delta Min. Avg. Max.
11.22mA 13.25mA 2.03mA 6.92mA 10.40mA 13.25mA
-56.8ms -56.5ms 302.7us(3.3KHz) -56.6ms 303us -56.5ms
Scope Stopped 5.0 ms/d -45.3 ms Auto Voltage 1
m1 m2 Delta Min. Avg. Max.
-0.15mA -0.10mA 0.05mA -0.28mA 1.22mA 13.35mA
-59.0ms -37.5ms 21.6ms(46.4Hz) -43.5ms 21.6ms -56.5ms

II. Non-connectable advertisements current consumption captures and figures

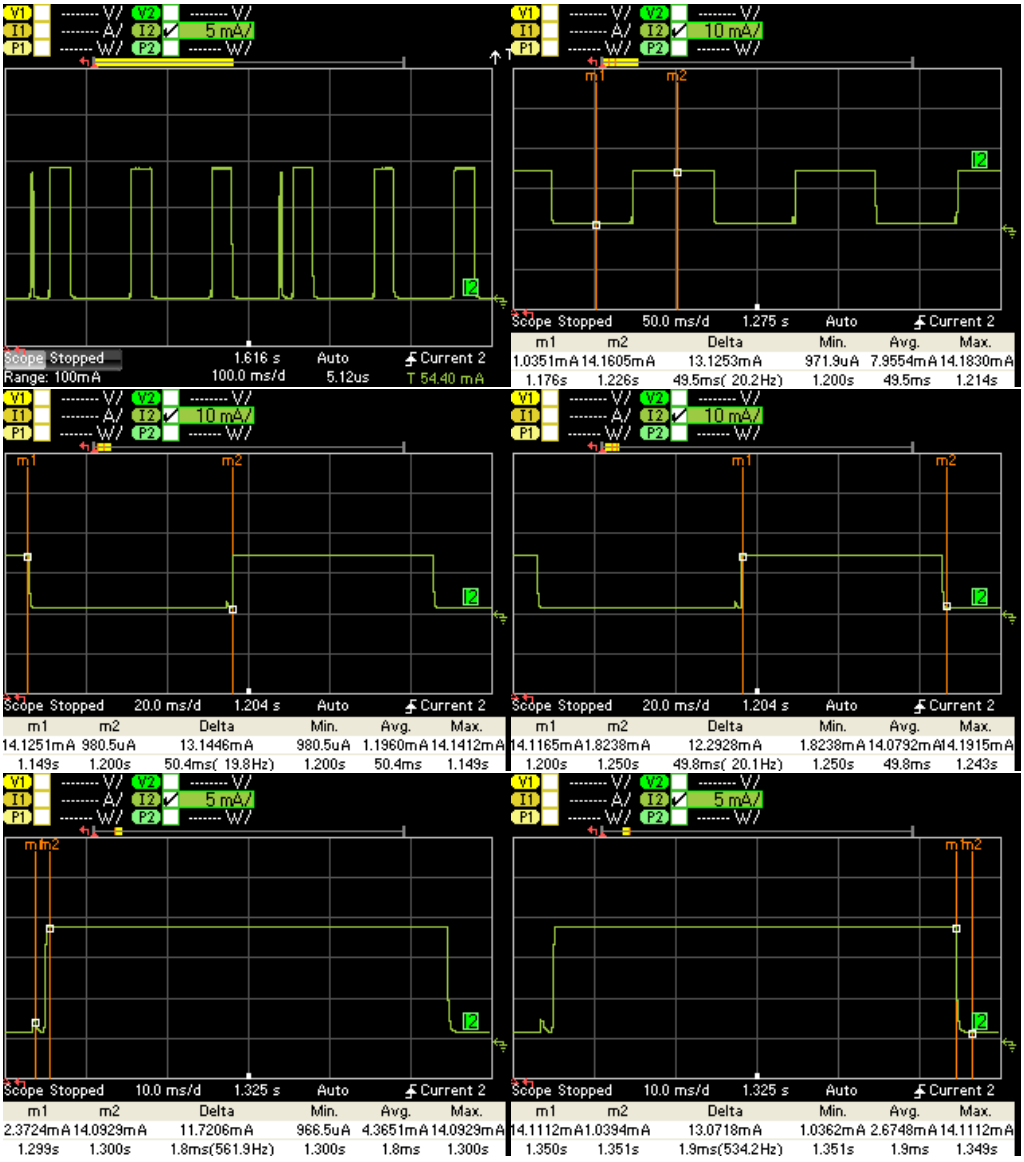


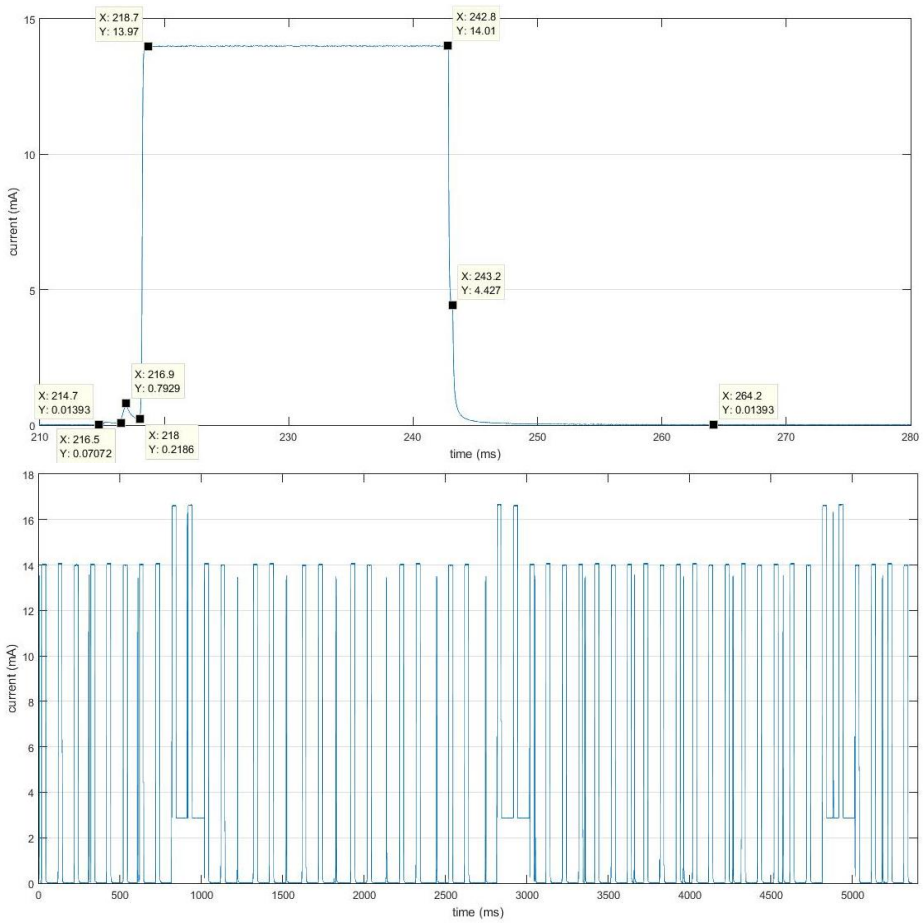



current (mA)
time (ms)
X: 17.7
Y: 0.001378
X: 31.89
Y: 1.5e-05


current (mA)
time (ms)
X: 12.98
Y: 0.0001982
X: 13.36
Y: 0.0007694
X: 13.87
Y: 0.0003054
X: 14.41
Y: 0.0001918
X: 14.65
Y: 0.01094
X: 14.94
Y: 0.01135
X: 15.46
Y: 0.01084
X: 15.24
Y: 0.001031
X: 15.76
Y: 0.01121
X: 16.3
Y: 0.01069
X: 16.06
Y: 0.001031
X: 16.58
Y: 0.01093
X: 16.73
Y: 0.004773
X: 16.89
Y: 0.004604
X: 17.06
Y: 0.001656
X: 17.66
Y: 0.001023
X: 17.7
Y: 0.001378

III. Scanning current consumption captures and figures




X: 218.7
Y: 13.97
X: 242.8
Y: 14.01
X: 243.2
Y: 4.427
X: 216.9
Y: 0.7929
X: 214.7
Y: 0.01393
X: 216.5
Y: 0.07072
X: 218
Y: 0.2186
X: 264.2
Y: 0.01393
current (mA)
time (ms)
current (mA)
time (ms)