Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Anders Storrø

GATT Based Network Solution For Bluetooth Mesh

Master's thesis in Electronic Systems Design
Supervisor: Arne Morten Midjo
June 2021

NTNU
Norwegian University of
Science and Technology

Anders Storrø

GATT Based Network Solution For Bluetooth Mesh

Master's thesis in Electronic Systems Design
Supervisor: Arne Morten Midjo
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of
Science and Technology

Electronic Systems Design

TFE4930

GATT Based Network Solution For Bluetooth Mesh

Author:
Anders Storrø

E-mail:
Anders.Storro@nordicsemi.no

Supervisors:
Arne Morten Midjo
Omkar Kulkarni
Trond Einar Snekvik

E-mail:
Arne.Midjo@ntnu.no
Omkar.Kulkarni@nordicsemi.no
Trond.Einar.Snekvik@nordicsemi.no

Norwegian University of Science and Technology
& Nordic Semiconductor ASA

June 9, 2021
Trondheim

Abstract

This thesis presents the exploration, design, and implementation of an alternative network solution for the Bluetooth mesh standard. This solution aims to provide better performance with regard to throughput and data consistency compared to the legacy Bluetooth mesh network solution. The completed implementation is based on the existing GATT bearer feature and is implemented to comply with the current revision of the Bluetooth mesh specification. The complete implementation provides all necessary means for establishing, configuring, and maintaining a GATT-based Bluetooth mesh network. This has been achieved by utilizing existing Bluetooth mesh intrinsic features and a custom vendor model. The implementation shows promising results concerning both throughput and data consistency. Further work is necessary to explore the full potential of this network solution. This thesis provides a foundation for future work that may aim to enhance the performance and features of the Bluetooth mesh standard.

Sammendrag

Denne avhandlingen presenterer undersøkelser, designet, og implementeringen av en alternativ nettverksløsning for Bluetooth mesh-standarden. Formålet med denne løsningen er å oppnå bedre ytelse med tanke på utførselshastighet og datakonsistens sammenlignet med den nåværende Bluetooth mesh-nettverksløsningen. Implementasjonen er basert på den eksisterende GATT-bearer-funksjonaliteten, og er implementert i samsvar med gjeldende revisjon av Bluetooth mesh-spesifikasjonen. Den komplette implementasjonen innehar all nødvendig funksjonalitet for etablering, konfigurering og vedlikehold av et GATT-basert Bluetooth mesh nettverk. Dette er oppnådd ved å ta i bruk eksisterende innebygde Bluetooth mesh-funksjoner og en proprietær leverandørmodell. Implementasjonen viser lovende resultater med hensyn til både utførselshastighet og datakonsistens. Ytterligere arbeid er nødvendig for å kunne avdekke potensialet til denne nettverksløsningen fullt ut. Denne oppgaven danner et grunnlag for fremtidig arbeid som kan ta sikte på å forbedre ytelsen og funksjonaliteten til Bluetooth mesh-standarden.

Contents

Contents 2

List of Figures 4

List of Tables 4

Nomenclature 5

1 Introduction 6
1.1 Background . . . 6
1.2 Project Scope . . . 7

2 Theory 8
2.1 Bluetooth Low Energy . . . 8
2.1.1 BLE Stack . . . 8
2.1.2 GAP . . . 10
2.1.3 GATT . . . 11
2.2 Bluetooth Mesh . . . 13
2.2.1 General Overview . . . 13
2.2.2 Bluetooth Mesh Stack . . . 15
2.2.3 GATT bearer . . . 17
2.2.4 BTM Advertising Message Format and Capabilities . . . 18

3 Preliminary Research 22
3.1 ADV bearer Performance Assessment . . . 22
3.2 Device Firmware Update Use-Case . . . 22
3.3 Implementation Approach . . . 23
3.3.1 Approach Research . . . 23
3.3.2 Communication Over GATT . . . 24
3.3.3 Control of the GATT Network . . . 25
3.4 GATT Network Topology . . . 25

4 High Level Modeling and Simulation 27
4.1 Simulation Environment . . . 27
4.2 General Simulation Conditions . . . 27
4.3 Modeling Transmission behavior . . . 28
4.3.1 Common Behavior . . . 28
4.3.2 ADV bearer Solution . . . 29
4.3.3 GATT Bearer Solution . . . 29
4.4 Modeling Noise . . . 30
4.4.1 Uniform Noise . . . 30
4.4.2 Internal Noise . . . 31
4.5 Simulation Output Data . . . 34
4.6 Simulation Results . . . 35
4.6.1 Simulations for Network Alpha . . . 35
4.6.2 Simulations for Network Beta . . . 41
4.6.3 Topology Exploration for the GATT Bearer . . . 45

4.6.4 Simulation Summary 48

5 Development **49**
5.1 Development Environment 49
5.2 Proxy Client Module 49
5.2.1 Beacon Handling 50
5.2.2 Connection Establishment and Discovery 51
5.2.3 Interfacing With the Proxy Client Module 51
5.3 GATT Proxy Configuration Model 53
5.3.1 Mapping Functionality 54
5.3.2 Connection Establishment and Maintenance 57
5.3.3 Controlling the ADV bearer 61
5.4 Utility Tools for Test and Development 63
5.4.1 GPC Client Terminal 63
5.4.2 GPC Test Commands 63
5.4.3 Mapping Assessment Program 64
5.4.4 Miscellaneous 66

6 System Testing **67**
6.1 Functional Testing of the System 67
6.1.1 Mapping and Configuration Assessment 68
6.1.2 Heterogeneous Configuration Testing 69
6.1.3 Homogeneous Configuration Testing 70
6.1.4 Introduction of Unforeseen Events 70
6.1.5 Function Test Summary 71
6.2 Performance Testing 71
6.2.1 Test Setup 71
6.2.2 ADV bearer baseline 74
6.2.3 GATT bearer baseline 76
6.2.4 Three Devices GATT Bearer Performance 77
6.2.5 Four Devices GATT Bearer Performance 78

7 Discussion **80**
7.1 Limitations of the Final Implementation 80
7.1.1 Internal Routing 80
7.1.2 Flow Control 80
7.2 Changing the BTM Specification 81
7.2.1 Transmission of Data With the Proxy Service 81
7.2.2 GPC Model 82
7.3 Heterogeneous vs Homogeneous Approach 84
7.3.1 Homogeneous Evaluation 85
7.3.2 Heterogeneous Evaluation 85
7.3.3 Evaluation Summary 86
7.4 Assessment of the System Testing 86
7.5 Assessment of the High Level Model 87
7.5.1 Model Weaknesses, Inaccuracies, and Misconceptions 88
7.5.2 Predicted vs Observed Result 88
7.5.3 Summary 90

8 Conclusion **90**
8.1 Performance Results 90
8.2 Implementation Evaluation 91

9 Future Work **92**
9.1 Reduction in Power Consumption for BTM . . . 92
9.2 Mapping and Configuration Algorithm . . . 92

Bibliography **94**

A NCS Source Code **94**

B High Level Model Source Code **95**

List of Figures

2 The BLE stack . . . 8
3 Structure of GATT server . . . 12
4 Bluetooth mesh stack . . . 15
5 Exemplified Mesh Proxy Service . . . 19
6 BLE on-air packet . . . 20
7 Advertising physical channel PDU . . . 20
8 Non-connectable and non-scannable undirected advertising payload . . . 20
9 Complete on-air packet structure . . . 21
10 GATT connection topology example . . . 25
11 Internal Noise Model . . . 32
12 Regular nodes impact on internal noise . . . 34
13 Overview of network α . . . 36
14 Overview of network β . . . 41
15 Advertising chain example network . . . 42
16 Advertising triangle example network . . . 43
17 Overview of network β, GATT chain topology . . . 46
18 Overview of network β, GATT tree topology . . . 47
19 Mapping procedure between two GPC server models . . . 57
20 Standard connection over GATT proxy . . . 60
21 Recovery after proxy client reset . . . 61
22 Recovery after loosing proxy server . . . 62
23 Initial view of functional test setup . . . 67
24 Link mapping processing output . . . 68
25 Functional test setup overview after mapping . . . 68
26 Functional test configuration solutions . . . 69
27 Performance test setup . . . 72
28 Logic pulse sample . . . 72
29 ADV bearer performance baseline setup . . . 75
30 Restricted ADV bearer TX period . . . 75
31 GATT bearer performance test baseline setup . . . 76
32 GATT bearer performance test setup with three devices . . . 77
33 GATT bearer performance test setup with four devices . . . 79

List of Tables

1 ADV bearer simulation network α (1xtransmission) . . . 37
2 GATT bearer simulation network α (No buffer restrictions) . . . 38
3 ADV bearer simulation network α, node 18 (1-4 x transmission) . . . 39
4 GATT bearer simulation network α (Buffer max = 8) . . . 40
5 ADV bearer simulation network β (4xtransmission) . . . 42
6 Advertising chain simulation summary . . . 43
7 Advertising triangle simulation summary . . . 43

8 GATT bearer simulation network β 44
9 GATT bearer simulation network β, chain topology 46
10 GATT bearer simulation network β, tree topology 47
11 ADV bearer performance test 75
12 GATT bearer performance test baseline (Srv->Cli) 76
13 GATT bearer performance test baseline (Cli->Srv) 77
14 GATT bearer performance with three devices. (Node 2 as origin) 78
15 GATT bearer performance with three devices. (Node 1 as origin) 78

Nomenclature

ADV bearer: Legacy BTM advertising bearer

API: Application Programming Interface

ATT_MTU: Attribute Protocol Maximum Transmission Unit

CCCD: Client Characteristic Configuration Descriptor

DFU: Device Firmware Update

DK: Development Kit

DUT: Device Under Test

GPC: GATT Proxy Configuration

GPIO: General Purpose Input/Output

GUI: General User Interface

IoT: Internet of Things

ISM: Industrial, Scientific and Medical

NCS: nRF Connect Software development kit

RSSI: Received Signal Strength Indication

RTOS: Real-Time Operating System

SDK: Software Development Kit

SIG: Special Interest Group

1 Introduction

1.1 Background

Bluetooth mesh (BTM) is a standard that provides many-to-many communication between Bluetooth devices in wireless networks. This standard is a superset of the well-established Bluetooth Low Energy (BLE) standard and was introduced in 2017. Today, the BTM standard is primarily utilized commercially in lighting products, replacing traditional electrical wiring in commercial buildings and private homes. Nevertheless, the standard does also provide a framework that can be used for many application domains. It could, for example, be used to implement various sensory network applications, an application field that has exploded after the emergence of the Internet of Things (IoT). By utilizing a gateway that bridges a BTM network with the Internet, the BTM standard could compete for the market shares within IoT-related technology.

The desired performance qualities for a sensory network vary, depending on the target application use case. In some sensory networks, it might suffice to have similar capabilities as for the lighting application domain. The performance demand identifying the lighting application domain is a network where the general message workload is small. Each application event may often be executed by passing a single message in the network, containing a few bytes of application data. These are often occasional events, causing the network to be idle for large periods. Since all data may be enclosed by a single message, there are no issues associated with data consistency.

Further, in this domain, it is not critical if a message is sporadically lost in transmission. If the press of a light switch on rare occasions does not turn the light on, this can be regarded as an acceptable inconvenience. The requirements mentioned above are all well within the capabilities a BTM network is capable of providing.

However, other sensor applications might require capabilities that greatly exceed these requirements. Some sensory networks require near real-time monitoring capabilities of the sensor data in the network. This implies that the average message workload is much higher than for the lighting domain, requiring significantly higher throughput performance. Another challenge is that some sensory data sets might exceed the number of bytes that a single message transmission can carry. In this case, the data set must be transmitted as a train of several messages. Losing one of these messages in transmission might result in inconsistent data on the receiving device. For such application domains, the BTM standard has certain challenges.

One of the events that led to this thesis's creation occurred while working on another Bluetooth mesh related project. In the report *Bridging Home Automation and Bluetooth Mesh for Nordic Devices*[15], I present an implementation solution for creating a gateway that enables interfacing between BTM networks and a home automation platform using communication over the LTE-M cellular network. One of the central use-cases that were evaluated during this work was high throughput sensor applications for BTM. As a part of this evaluation, I investigated the bandwidth capabilities of a legacy BTM network [15, p. 14]. These findings showed that the throughput capabilities of a legacy BTM network advertising bearer, henceforth referred to as the ADV bearer, is restricted, making BTM less suited for communication tasks that require high throughput capabilities. Additionally, the BTM standard does not guarantee message delivery. This is caused by limitations in the current communication scheme of the standard. This implies that BTM, as of now, is neither ideal for applications where data consistency is essential.

Failing to provide a technology that provides satisfactory throughput and data consistency capabilities may exclude BTM as the preferred networking technology for several application purposes. These shortcomings should be investigated and, if possible, mended to improve the desirability of BTM as a communication standard.

1.2 Project Scope

The primary objective of this thesis has been to find and provide a solution that can rectify the previously mentioned shortcomings of the BTM standard. An ideal in the search for this solution has been that it should adhere to the existing BTM specification. Failing to find such a solution, the suggested alternative should have as little impact on the existing specification as possible. The reason for this criterion is due to the acknowledgment of the fact that changing the existing BTM specification is a long process. Even small changes or amendments to the specification require extensive reviews from the Bluetooth Special Interest Group (SIG). In addition to being a time-consuming process, the outcome of these reviews has a significant chance of ending up discarding the proposed changes.

This report is structured into eight main sections. Section 2 presents the background theory of the central concepts and standards utilized in this project. Section 3 presents the research and assessment that was conducted before starting the main work of the thesis. It initially presents the shortcomings of legacy BTM concerning throughput performance. Further, it presents a couple of different approaches to solve this issue before landing on the GATT bearer solution. Lastly, it suggests how a GATT bearer solution can be implemented using existing BTM features and discusses different network topology choices that can be utilized for this specific approach. Section 4 presents high level modeling and simulation for different bearers. Initially, it presents the theory and methodology used for deriving the different models. This includes modeling of both the ADV bearer and the proposed alternative GATT bearer. The conducted simulations in this section focus on comparing the performance metrics between the different bearers. In section 5 the design and implementation phase of the thesis is described. It includes implementing a proxy client module, a BTM vendor model for the configuration of the GATT bearer network, and a description of the utility tools implemented and used throughout the development. Section 6 presents the testing that has been conducted on the completed new implementation. It consists of a compound functional test where all essential features of the implementation are assessed and certified and performance testing for both the existing ADV bearer and the GATT solution. Finally, section 7, 8, and 9 respectively presents the discussion, conclusion, and future work for the thesis. The discussion part addresses some of the more interesting and complex aspects associated with the topic. These are considerations with regard to system performance and configuration, the impact of the BTM specification, and choices of network topology. The Conclusions summarize the thesis work result and assess the outcome based on the initial objectives and goals. In the future work section, I describe potential work that can expand on the work and findings of this thesis.

2 Theory

2.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) was introduced as a part of the Bluetooth 4.0 Core Specification, released in 2010. Compared to classical Bluetooth, the BLE standard is optimized for applications that need to run on a small power budget. It is well suited for devices that run on small batteries, making it possible for these devices to run for months or even years without a need to change the batteries[1].

The most prominent benefit of BLE, in addition to the mentioned low power consumption, is that it can run on devices that have relatively few resources, making the design cost of various wireless applications lower. In addition, this standard has become so familiar over the last decade that most smartphone providers on the market support it. This feature often makes it the preferred choice among similar wireless technologies. However, BLE also has some drawbacks. The data throughput is somewhat restricted, with an initial peak performance of 1 Mbit/s. A later revision of the standard has introduced a high-speed feature, enhancing the maximum throughput up to 2 Mbit/s. This implies that BLE is best suited for applications that require a relatively low bandwidth[1].

2.1.1 BLE Stack

The BLE stack can be seen in figure 2. We can see that the majority of the layers are confined inside either the host or controller block, with the HCI layer linking them together. On the top, we have the application layer, which contains the actual application. This layer is defined by the application developers and will not be discussed in this section. Most of the remaining layers will be described briefly, while some are described in more detail in separate sections.

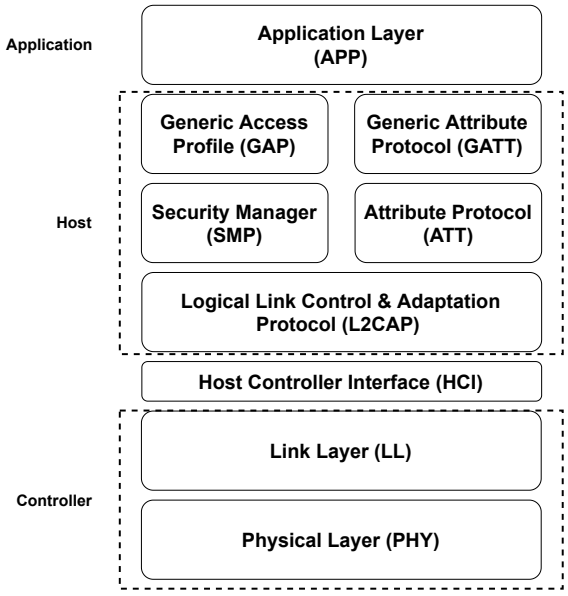


Figure 2: The BLE stack.

2.1.1.1 Physical Layer

The physical layer of BLE utilizes the 2.4 GHz ISM band to communicate. It uses 40 different channels that are spaced between 2.4000 GHz and 2.4835 GHz. Channels 37, 38, and 39 are

used for advertising, and they are spaced evenly throughout the band to minimize the risk of all advertising channels being disturbed at the same time. The remaining 37 channels are used for standard communication in BLE connections. BLE utilizes the frequency hopping spread spectrum technique to change channels between each connection event on a link. This scheme reduces the chance of interference on any channel in the 2.4 GHz band. This technique is crucial since this band is shared with other applications, such as Wi-Fi[6, p. 16].

2.1.1.2 Link Layer

The Link layer interfaces with the physical layer and functions as an abstraction layer to enable control of the radio on a device[1]. It manages the state of the radio and is the only layer of the entire stack that handles hard real-time constraints[6, p. 17].

2.1.1.3 HCI Layer

The Host Controller Interface (HCI) layer defines a protocol that allows the Host layer to communicate with the Controller layer of the stack. This allows the host and controller block to potentially exist on separate hardware, communicating through a serial interface [1].

2.1.1.4 L2CAP Layer

The Logical Link Control and Adaptation Protocol (L2CAP) layer have two main tasks. It acts like a multiplexer that encapsulates the protocols from the upper layers to form standardized BLE packets. It is also responsible for fragmentation and reassembly of packets that are larger than the maximum payload size of 27 bytes that a single BLE packet can contain[6, p. 25].

2.1.1.5 ATT Layer

The Attribute Protocol (ATT) defines a stateless client/server protocol. It defines how data is structured and exposed by a server in order for a client to interact with it[1]. The attribute layer is further explained in section 2.1.3.2.

2.1.1.6 SMP Layer

The Security Manager Protocol (SMP) is responsible for generating and distributing security material in BLE. It supports features like pairing, bonding, authentication, encryption, and message integrity[1].

2.1.1.7 GATT Layer

(See section 2.1.3.)

2.1.1.8 GAP Layer

(See section 2.1.2.)

2.1.2 GAP

The theory described in this section is in its entirety retrieved from the book *Getting Started with Bluetooth Low Energy*[6, p. 35].

The Generic Access Profile (GAP) defines how BLE devices should interoperate with each other. This framework specifies how a device discovers the presence of other devices, how they should establish connections to each other, how they broadcast their data, security considerations, and many other fundamental operations that ensures a standardized way of communication for BLE.

2.1.2.1 GAP Roles

GAP defines four different operational modes. Each role is associated with a set of restrictions and behavioral requirements. The four GAP roles are:

- Broadcaster – The broadcaster role is mainly used in applications with only a need to transmit data. It transmits this data through advertisement packets, available to all devices in proximity of the broadcaster.
- Observer – The observer role is the counterpart of the broadcaster role, intended for application where it is only a need to receive data. The observer scans for incoming advertising packets and extracts data from the packets sent from broadcasting devices.
- Central – The central devices inhabit the link layer "master" role of GAP. Devices with the central role are responsible for initiating connections to other devices, thus forming the BLE network. It is capable of maintaining several connections at the same time. The central role entails a higher demand on computing resources than the link layer "slave" counterpart. To establish connections, the central listens for incoming advertising packets from nearby devices. By utilizing the information in these packets, the central can connect to one or more of these devices.
- Peripheral – The peripheral role corresponds to the link-layer "slave" role. As the central role's counterpart, a peripheral device make its presence known by transmitting advertisement packets. These packets can be scanned by nearby central devices, which can initiate a connection to the peripheral device. A peripheral device will have lower resource requirements concerning computing power and memory than the central role.

The BLE specification does not mention that the GAP roles need to be mutually exclusive, I.e., a device may operate in one or several of these roles at any given time.

2.1.2.2 Modes and Procedures

GAP defines a set of modes and procedures that are associated with one or more roles. This section will mention the most relevant modes and procedures.

For the broadcaster and observer roles, we have broadcast mode and the observation procedure, respectively. These define the framework through which a broadcaster may send data that one or more observers receive. The rest of the modes and procedures mentioned here is associated with discovery and connection between central and peripheral devices.

For the peripheral role, the two most vital discovery modes are the non-discoverable and general discoverable modes. The first mentioned is, as the name suggests, a mode where the peripheral device does not wish to be discovered. In contrast, the general discoverable mode is the most common mode used when a device wishes to connect to a central device. The central role has more than one discovery procedure, but it is the general discovery procedure that is of most

interest for most practical purposes. This discovery procedure is the counterpart of the general discoverable mode, used by devices looking for all possible discoverable peers.

The three connection modes are the non-connectable, directed connectable, and undirected connectable modes. The non-connectable mode is used by peripherals that do not allow connections, the directed connectable mode is used for rapid reconnection with familiar devices, while the undirected connectable mode is the standard mode for connection where a peripheral device makes itself connectable for a longer period of time. The main connection procedures are the auto-, selective-, direct-, and general connection establishment procedure. Among these four, the general connection procedure is most commonly used when forming a connection to a new unknown peripheral.

2.1.3 GATT

The theory described in this section is in its entirety retrieved from the book *Getting Started with Bluetooth Low Energy*[6, p. 51].

The Generic Attribute Profile (GATT) defines how data is formatted and exchanged between connected devices in BLE. While GAP handles the interaction on the base level, GATT manages the actual structuring and transmission of data. GATT utilizes the Attribute protocol (ATT) to exchange data between devices.

2.1.3.1 GATT Roles

GATT operates with two distinct roles, the client- and the server role. It is essential to mention that these roles are completely decoupled from the previously mentioned GAP roles. E.g., a GATT client may be present on either a central or peripheral device, depending on the behavior of the application used.

The GATT client is the one with the initiative in a GATT client/server pair. It is the entity that sends requests to the server and receives complimentary responses. In some cases, the client might also receive server-initiated updates, but the key aspect here is that the client is in control of this relationship. The client does not initially know anything of the composition of attributes that are contained in the server. To retrieve this information, the GATT client has to perform initial service discovery. After this procedure is done, the client can start operating on the attributes provided by the server and receiving server-initiated updates.

The GATT server is responsible for storing and making user data available for the GATT client. As mentioned, it receives requests from the client to which it responds, with the additional opportunity to send self-initiated updates.

2.1.3.2 Data Hierarchy

The smallest entity of data in GATT is called an attribute. An attribute consists of the following fields:

- Handle – The handle can be considered as the address of the attribute.
- Type – The attribute type defines what kind of data that the attribute value represents.
- Permissions – Permissions tell which operations can be performed on the attribute, like readable, writable, read and writable, or none. It also determines the encryption level that a client requires in order to access the attribute.
- Value – The value holds the actual data of the attribute. It can contain any data type, like integers, floats, UTF-strings, and arrays. The maximum size of an attribute is 512 bytes.

The data structure inside a GATT server can be regarded as a list of attributes, where the attribute is sorted in rising order depending on their handle values.

The structure of a GATT server is exemplified in figure 3. This figure shows that a GATT server may contain one or more services, where every service may contain zero or more characteristics. The characteristics can, in turn, contain one or more descriptors.

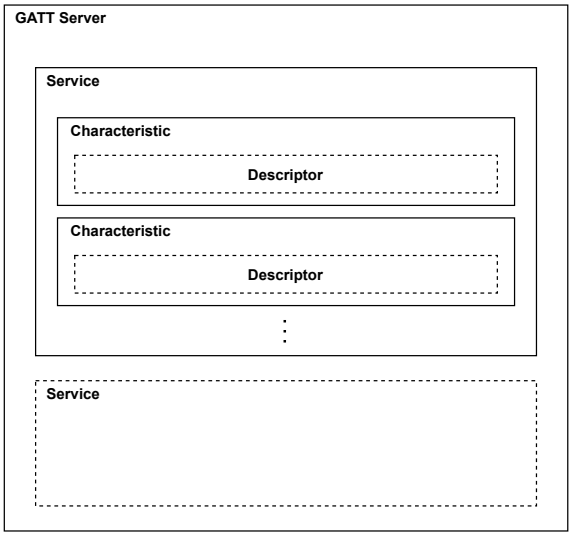


Figure 3: Structure of GATT server.

A service structure attributes that are conceptually related. The service declaration attribute is used to define the beginning of a new service. This can either be the primary or the secondary service UUID, where the first-mentioned is the most common. The permission for this attribute is read-only since it must be available for clients to read. The value of the declaration refers to the specific service introduced, E.g., the Heart Rate Service of BLE.

The characteristics are containers for data inside a service. The characteristic consists of at least two attributes; the declaration and the value. The declaration is of the characteristic type, a unique UUID type that defines the characteristic attribute. Like the service attribute, this is also an attribute of the read-only type. The value of this attribute contains three fields of data. The properties, value handle, and the UUID of the characteristic in question. The properties consist of a bitfield that defines what operations and procedures that may be performed with the characteristic. The main properties are:

- Broadcast – Allows this characteristic value to be placed in advertising packets on the server.
- Read – Allows clients to read this characteristic.
- Write without response – Allows clients to use the Write operation on this characteristic (no acknowledgment).
- Write – Allows clients to use the Write operation on this characteristic (with acknowledgment).
- Notify – Allows the server to send notifications to the client (no acknowledgment)
- Indicate – Allows the server to send indications to the client (with acknowledgment)
- Signed Write Command – Allows clients to use the Signed Write Command operation on the characteristic.

The value handle contains the attribute address that holds the actual value of the characteristic. The UUID identifies what type of characteristic this is. It can either be a SIG-defined characteristic

or a vendor-specific characteristic. The value attribute of the characteristic holds the actual data. The handle and type of this attribute are the same as those defined in the value of the characteristic declaration. Both the permission and the value field of this attribute can be of any type, depending on the characteristic type.

In addition to the declaration and value attributes, a characteristic may contain one or more descriptors that provide additional metadata for the characteristic. I will only mention the Client Characteristic Configuration Descriptor (CCCD) since it is the only descriptor type of relevance to this thesis. The purpose of this descriptor is basically to act as an enabling switch for server-initiated updates, meaning indications and notifications. Since the GATT client has the master role in the server/client relationship, it needs to manage the server-initiated updates. Using the CCCD descriptor, it can decide when the server is allowed to send these updates.

In section 2.2.3.2 the Mesh Proxy Server is presented, which may contribute to clarify the concept of GATT services.

2.2 Bluetooth Mesh

The Bluetooth mesh (BTM) standard is a many-to-many wireless communication protocol. It is built on top of the Bluetooth Low Energy Core specification. As long as a device supports a version equal to (or higher) than the 4.0 version of the BLE spec, it can support BTM. Both the BTM specification, as well as the underlying BLE core specification, is developed and maintained by the Bluetooth Special Interest Group (SIG)[8].

The introduction of the BTM standard was done to extend the capabilities of BLE also to include mesh networking. It has the potential to create networks with thousands of devices that can communicate and collaborate. Together these devices may support a multitude of different application domains, like collecting sensory data, control and monitoring lighting installations, performing predictive maintenance, and positioning applications[3].

2.2.1 General Overview

2.2.1.1 Addressing Scheme

There are three different types of addresses used in BTM; unicast, group, and virtual addresses. Unicast addresses are assigned to elements and always represent a single element in the network. There are 32767 unicast addresses per mesh network. Group addresses are multicast addresses that can represent multiple elements in the network. There are 16384 group addresses per mesh network. The virtual addresses are practically an extension of the group addresses. These addresses can also represent multiple elements, but the virtual addresses are based on 128-bit label-UUID's, making the address space very large[10, p. 23].

Within every BTM device, we have at least one **element**. Elements define the addressable entities within a BTM network and are assigned a unique unicast address. The first element in a node is called the primary element and contains the mandatory foundation models explained in section 2.2.2.2. The primary element must possess the lowest assigned unicast address of the device, while the eventual secondary elements will be assigned the subsequent addresses. Each element may contain several models but is not allowed to contain more than one unique instance of a single model[10, p. 22]. Models are described in section 2.2.2.1.

2.2.1.2 Network Communication

The three advertising channels of BLE mainly provide on-air communication in BTM. When the mesh utilizes this form of communication, it is the ADV bearer that is responsible for transmitting

and receiving messages, as explained further in section 2.2.2.7. Each BTM ADV message can contain up to 31 bytes of data. This includes all addressing information, security material, payload, and other metadata.

Conventionally, BTM utilizes the concept of "Managed Flooding" to pass messages to other nodes in the network. Using the ADV bearer implies no inherent way of forming links between communicating devices in a BTM network. Whenever a device wishes to forward a message to other network members, it has to do this by broadcasting over the advertising channels. This means that all devices within radio proximity will receive this message. They receive this regardless of if it is associated with the device or not. When a message is received, the node needs to check if the message is associated with any elements located on the device.

To make it possible for a message to travel further than the limitations of the radio range of the origin device, one or more of the adjacent nodes needs to relay this message. Relaying is a crucial feature of BTM and can be configured individually for each device in the BTM network. When a relay node receives a message, it performs a check to see if it should be retransmitted or not. The criteria it bases this decision on depends on the destination address of the message and the value of the Time to Live (TTL) parameter contained in the message. The purpose of the TTL value is explained later in this section. Upon the first setup of a BTM network, it is up to the installer to provide enough relay nodes to ensure that messages can travel between any two nodes.

Inherently, a transmitting node will not get any verification of whether a message is successfully received or not. While it is possible to acknowledge messages in a higher layer of BTM, this involves using two or more base message transmissions to ensure message delivery. To increase the likelihood of successfully transmitting a single message, BTM is reliant on redundancy. In other words, a message may be transmitted multiple times to improve the likelihood of successful message delivery.

To maintain the number of redundant messages that flow in the network at any given time, BTM specifies a couple of message control mechanisms. The first one is the TTL parameter. Each message that is transmitted is equipped with this parameter. Each message is initialized with a TTL value at the origin device. Whenever a relaying node retransmits the message, the TTL value is decremented. When this parameter reaches a value lower than two, it will be discarded by the next relaying node that handles it, effectively ending the message's lifetime in the network. The other control mechanism is the message cache. The message cache of a device stores a unique ID for all messages that it accepts. If the same message arrives at the device for a second time, it is immediately discarded[10, p. 36-101].

2.2.1.3 Additional Features

In addition to the functionality mentioned in section 2.2.1.2, BTM provides some additional features. The first is the proxy feature. This feature facilitates the forwarding of network PDUs between GATT and ADV bearers. This feature is central for this thesis and is thoroughly described in section 2.2.3.

The two remaining features are the Friend and Low Power feature. These two features combined provide the means for having mesh nodes with a significantly lower power consumption than regular mesh nodes. The concept is that a friend node establishes a bond with a low power node (LPN). The friend node is responsible for holding and forwarding messages to the LPN node whenever it polls for information updates. This enables the LPN node to turn off the radio scanning for a large portion of the execution time, thus saving a significant amount of power. The LPN feature is ideal for devices that run on a limited power source and is not dependent on continuously communicating with the rest of the network. An example of such an application can be a switch in a lighting installation[10, p. 37].

2.2.1.4 Security

BTM offers authentication and security on two levels. First, there is the network level. Using network keys enables the opportunity to create area isolation in the network. Only devices sharing the same network key may communicate with each other. A device may belong to one or more subnets, depending on the number of network keys it owns. The second level is the application level. Using application keys, a model may communicate with models on other devices that share the same keys. This level of security enables privacy for different applications that operate on the same subnet[10, p. 19].

If a device wishes to join a mesh network, it must undergo the process of provisioning. Provisioning is the process of authenticating and providing basic information like unicast addresses and a network key to a device. When the provisioning process is completed, a node may transmit or receive messages in a mesh network[10, p. 15]. To ensure that the process of provisioning is executed securely, BTM utilizes a Security manager protocol derived from the BLE protocol (See section 2.1.1.6)[10, p. 20].

The security measures of BTM provide protection against known security attacks like eavesdropping attacks, man-in-the-middle attacks, replay attacks, trash-can attacks, and brute-force key attacks[10, p. 18]. Detailed information about provisioning and mesh security can be found in the Mesh Profile spesification[10, p. 229, 101].

2.2.2 Bluetooth Mesh Stack

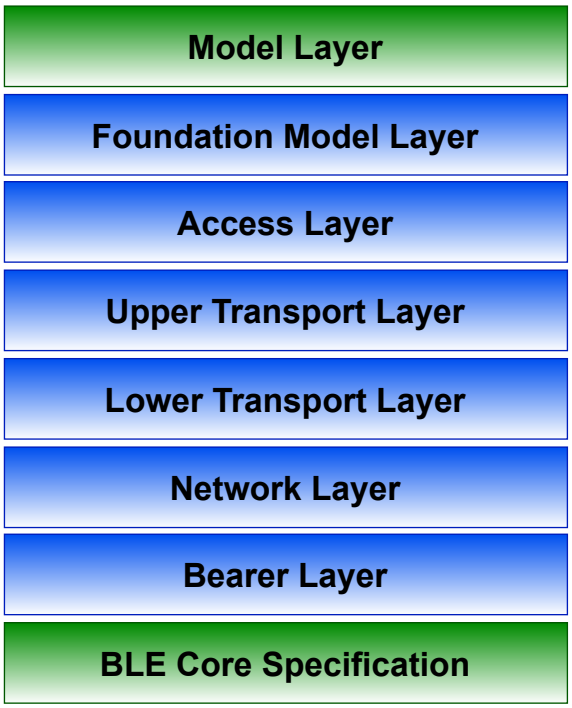


Figure 4: Bluetooth mesh stack.

2.2.2.1 Model Layer

The top layer of the BTM stack is the model layer. Here we differentiate between two different categories of models; SIG defined models and vendor models. These two categories provide the communication interface for the application level of a BTM device.

The first category is the SIG-defined models. These are standardized models that are designed to provide interoperability between BTM devices, regardless of manufacturer. This includes models that handle generic functionality like basic on/off and level features, models used to gather sensory data, and models for lighting applications. The specification for these models is defined and maintained by the BT SIG. Any developer who wishes to utilize these models must abide by the strict set of rules for state and message interaction when implementing these models.

The second category is the vendor-specific models. These are custom models that can be freely defined by any developer and can target any application imaginable[9].

2.2.2.2 Foundation Model Layer

The foundation models are mandatory models that are required to make a BTM device functional and configurable. There are two sets of these models. The Configuration models, as the name suggests, is used to configure the network. It is used to distribute and set security keys, assign publication and subscription configurations, and many other parameters for the devices in the network. The second one is the Health models. These are used to monitor and assess the state of the network, with the intention to unveil issues and problems on the nodes[10, p. 136].

2.2.2.3 Access Layer

The access layer provides the means the models use to communicate with each other in the network. It is in charge of the formatting of the payload and publishing and subscription parameters. Further, it defines if a message should be acknowledged by the receiver or not and the number of hops the message is allowed to take during transmission (TTL). It also handles sequence numbering of messages, ensuring that the network is not prone to replay attacks[10, p. 92].

2.2.2.4 Upper Transport Layer

The upper transport layer handles encryption and authentication of messages delivered from the access layer. This is the first of two iterations of encryption performed on these messages, the second one being performed on the network layer. This enables different applications to share the same network without exposing the internal state to other applications on the network. These messages are encrypted using an application key bound with the respective model that passed the message from the model layer. The exception is the messages associated with the configuration foundation models secured using the device keys.

This layer also manages a set of control messages. These include messages associated with the Friend/LPN-node feature and Heartbeat functionality. These messages do not undergo the same encryption procedure as the access messages on this layer[10, p. 62].

2.2.2.5 Lower Transport Layer

The primary purpose of the lower transport layer is to perform segmentation and reassembly of outgoing and incoming mesh messages, respectively. Due to the restricted payload a BLE advertising packet may contain, larger messages must be sent as a train of smaller messages. Messages smaller than 12 bytes may be passed unsegmented, while messages longer than this must be sent as two or more segments.

In addition, this layer is responsible for queuing messages associated with the Friend/low power node feature of BTM[10, p. 49].

2.2.2.6 Network Layer

The network layer is responsible for handling the message traffic flowing through a device. It performs the following tasks[10, p. 39]:

- Network encryption/decryption – The network layer performs network-level encryption on all messages passed from the lower transport layer. The message is then either passed to the bearer layer or an internal interface. This depends on if the message destination is located outside the origin device or not. For all incoming messages, decryption and authentication are performed to check if the message is associated with a network key belonging to the device in question.
- Interface filtering – The network layer facilitates filtering of both incoming and outgoing messages. The filtering intends to reduce the strain on the on-air medium by reducing unnecessary message traffic.
- Message relaying – For nodes with the relay feature enabled, the network layer handles the forwarding of messages. These are messages that are not associated with a unicast address present on the receiving device or a message addressed to a group address. Before retransmission, the network layer checks the message's TTL value to see if it qualifies for retransmission. If it does, the TTL value is decremented by one, and the message is passed to the bearer layer.
- Message caching – In order to reduce the number of security checks and retransmissions in the network, the network layer is equipped with a message cache implemented as a ring buffer. When a message arrives from the bearer layer, the message cache fetches an ID from it. It then performs a lookup for this ID in the cache. If the message is already present, it is immediately discarded. Otherwise, the ID is placed at the head of the cache, potentially evicting the oldest entry. The message is then passed along to the decryption stage of the network layer.

2.2.2.7 Bearer Layer

The bearer layer consists of two primary bearer types; the ADV bearer and the GATT bearer[10, p. 38].

As mentioned in section 2.2.1.2, the ADV bearer is the conventional way of transmitting data inside the BTM network. Messages transmitted with the ADV bearer shall be put inside the advertising data of a BLE advertising PDU. The *Mesh Message AD type* advertisement packet is used for this purpose, and it contains a BTM network PDU. Any advertisement using the Mesh Message AD type shall be non-connectable and non-scannable undirected advertising events. These messages provide room for 31 bytes of payload. Devices supporting the ADV bearer should support both the GAP Observer role and GAP Broadcaster role, as described in section 2.1.2.1. Since the messages sent with the ADV bearer are broadcasted, BTM devices should opt for a scanning duty cycle that is close to *100%* to avoid missing incoming messages.

The GATT bearer is initially intended to enable devices that cannot support the ADV bearer to participate in a mesh network. Due to the relevance the GATT bearer has for this thesis, it is described separately in section 2.2.3.

2.2.3 GATT bearer

A GATT bearer uses the Proxy protocol to transmit and receive Proxy PDUs between two devices over a GATT connection. The GATT bearer defines two roles: The GATT Bearer Client and the GATT Bearer Server. The server instantiates one and only one Mesh Proxy Service, while the GATT Bearer Client shall support the Mesh Proxy Service[10, p. 38].

2.2.3.1 Proxy Protocol

A GATT bearer utilizes the proxy protocol to exchange data between devices. This protocol supports the exchange of network PDUs, mesh beacons, proxy configuration messages, and Provisioning PDUs over a GATT connection. The protocol defines two roles: the Proxy server and the Proxy client. The server must support a bearer using the proxy protocol, typically a GATT bearer, and at least one additional bearer like the ADV bearer or a secondary GATT bearer. The client needs to support a bearer using the proxy protocol. A proxy PDU may contain all the four mentioned message types. The header of the proxy PDU contains information about what type of data contained in the payload and information regarding segmentation and reassembly[10, p. 262]. The last-mentioned parameter is used when a message is larger than the maximum allowed payload size of a proxy PDU.

The proxy protocol supports filtering to reduce the amount of information passed from the server to the client. This filtering is based on mesh addresses and can either be of the white- or blacklist type. If the server is configured to use a white list, it will only pass along messages associated with addresses in the list. If the server uses a blacklist, it will exclude messages associated with addresses in the list. The default configuration for the server when a connection is established is an empty white list. Whenever the server receives a valid mesh message from the client, the filter shall update itself to allow messages with these unicast addresses to be passed from the server to the client[10, p. 264].

2.2.3.2 Mesh Proxy Service

A general description of GATT services can be found in section 2.1.3.2.

The Mesh Proxy Service is used to enable a proxy server to send and receive data to and from a proxy client. The service shall be initiated as a primary service, and the type shall be set to the *Mesh Proxy Service*[10, p. 275].

This service consists of two characteristics. The first is the Mesh Proxy Data In characteristic and is used to send proxy PDUs from a client to a server. The mandatory property for this characteristic is *Write without response*. The second characteristic is the Mesh Proxy Data Out. This characteristic is used to send proxy PDUs from the server to a client. The mandatory property for this characteristic is *Notify*. In addition, this characteristic includes the CCCD descriptor to enable control of when a server may send notifications to the client. In this context, the Proxy client will enable notifications as soon as the connection is established[10, p. 279]. An exemplified layout of the attributes inside a Mesh Proxy Service can be viewed in figure 5.

A server must use the General Discoverable mode with undirected connectable advertising events, as described in section 2.1.2.2. There are two ways of advertising the presence of the Mesh Proxy Service, either by using network IDs or node identity. A server advertising with the network ID will send advertisement packets periodically for every subnet the server is associated with. This method is suited for clients that wish to connect to the mesh network without concern about which device it is connecting to. When advertising with the node identity, the packets contain the unicast address of the primary element of the advertising node. It is originally intended for situations where we need to deliver large amounts of data to a node that cannot be easily identified or is not advertising[10, p. 276].

2.2.4 BTM Advertising Message Format and Capabilities

The format and capabilities of a BTM advertising message are relevant to this thesis. This section clarifies in detail some of the most central aspects regarding advertising in BLE. This theory is a product of both BLE and BTM specific topics.

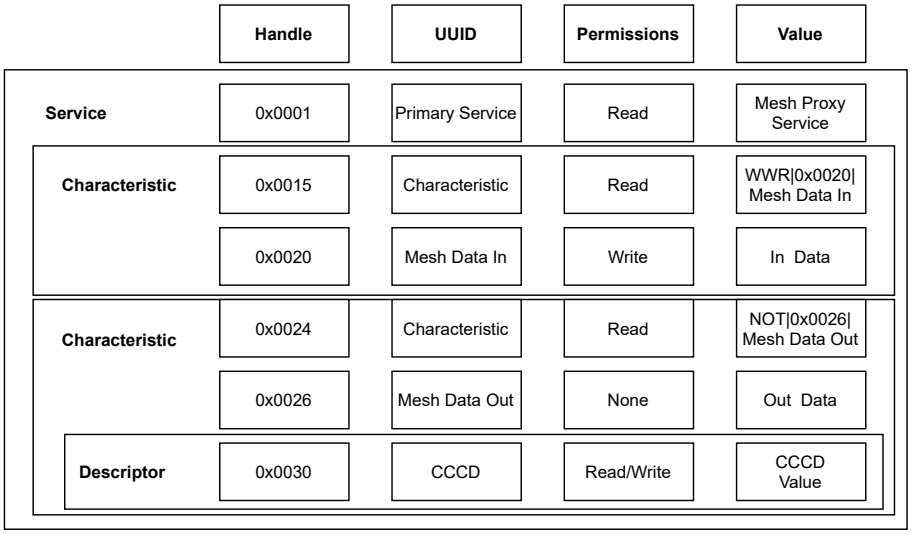


Figure 5: Exemplified Mesh Proxy Service.

2.2.4.1 Minimum Advertising Period

The minimum time it takes to transmit a single BTM advertising PDU, further denoted as A_{min}, can be derived from the BT Core specification[8, p. 2939]. It states that an advertising interval shall be in the range of 20 milliseconds to 10485.76 seconds. Furthermore, it states that every advertising interval should be followed by a pseudo random delay between 0 and 10 milliseconds. Since we are interested in the peak sending capabilities, we will focus on the minimum advertising interval of 20 milliseconds, denoted as α_{min}. To account for the advertising delay we can add the mean value of the delay range, denoted as $\overline{\delta}$. This gives us the following equation for the minimal time it takes to transmit a single BTM advertising PDU:

$$\begin{aligned} \mathrm{A_{min}} &= \alpha_{\mathrm{min}} + \overline{\delta} \\ &= 20\,\mathrm{ms} + \frac{10\,\mathrm{ms} - 0\,\mathrm{ms}}{2} = 25\,\mathrm{ms} \end{aligned} \tag{1}$$

2.2.4.2 BLE Transmission Speed

This thesis focuses on the LE 1M physical layer(LE 1M PHY) implementation since this is the only mandatory physical layer implementation for BLE. The LE 1M PHY has a symbol rate of one mega symbol per second[8, p. 190]. The LE 1M PHY is uncoded, meaning that each symbol represents exactly one bit. Since one symbol represents on bit, this correlates to a bit rate of *one megabit per second*.

2.2.4.3 BTM Advertising Message Format

The structure of the uncoded physical layer on-air packets is described in Volume 6 part B of the BT Core specification[8, p. 2865]. Figure 6 shows the format of the packet.

Here is a brief description of the different fields in the packet:

- Preamble — The preamble is used in the receiver to perform frequency synchronization, symbol timing estimation, and Automatic Gain Control training. For packets transmitted on the LE 1M PHY, the preamble is 8 bits.

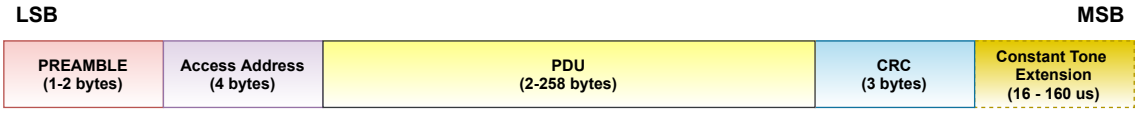


Figure 6: BLE on-air packet.

- Access address — Primarily used to avoid collisions between different connections on the link layer of the BLE-stack.
- PDU — The protocol data unit is the payload of the packet. The length of the PDU depends on the message type it contains.
- CRC — Cyclic redundancy check. Used to detect errors in a received packet
- Constant Tone Extension — Optional field that is associated with location and direction finding in BLE.

By assessing the different fields of the packet, we can conclude that an uncoded LE 1M PHY packet so far will consist of one byte of preamble, four bytes of access address, and three bytes of CRC. We can disregard the optional Constant Tone Extension field since it is not relevant in this context. This leaves the PDU, which can be between two and 258 bytes. Since the PDU in this instance is of the advertising type, we can refer to the advertising physical channel PDU section BT core specification[8, p. 2871]. Figure 7 shows the fields of an advertising physical channel PDU.

Figure 7: Advertising physical channel PDU.

This consists of a two-byte header and a variable payload between one and 255 bytes. The header contains the PDU type, the byte length of the payload, and three flags called ChSel, TxAdd, and RxAdd. The three flags have different purposes depending on the PDU type. To obtain the advertising PDU type used in BTM, we must refer to the BTM profile specification[10, p. 38]. It states that any advertisement using the Mesh Message advertising data type shall be non-connectable and non-scannable undirected advertising events. Figure 8 shows the fields of a non-connectable and non-scannable undirected advertising payload [8, p. 2874].

Figure 8: Non-connectable and non-scannable undirected advertising payload.

This payload consists of the AdvA and AdvData fields. The AdvA field contains the advertiser's public device address if the header's TxAdd flag is set. Otherwise, it contains the random device address. The AdvData field holds the data containing the BTM PDU. When fully utilizing a BTM network PDU the size of the BTM PDU will have a size of 31 bytes[10, p. 43]. Figure 9 summarizes the final BLE packet structure considering a AdvData field consisting of 31 bytes.

The figure shows us that *a fully utilized non-connectable and non-scannable undirected advertising message on-air packet will contain 376 bits.*

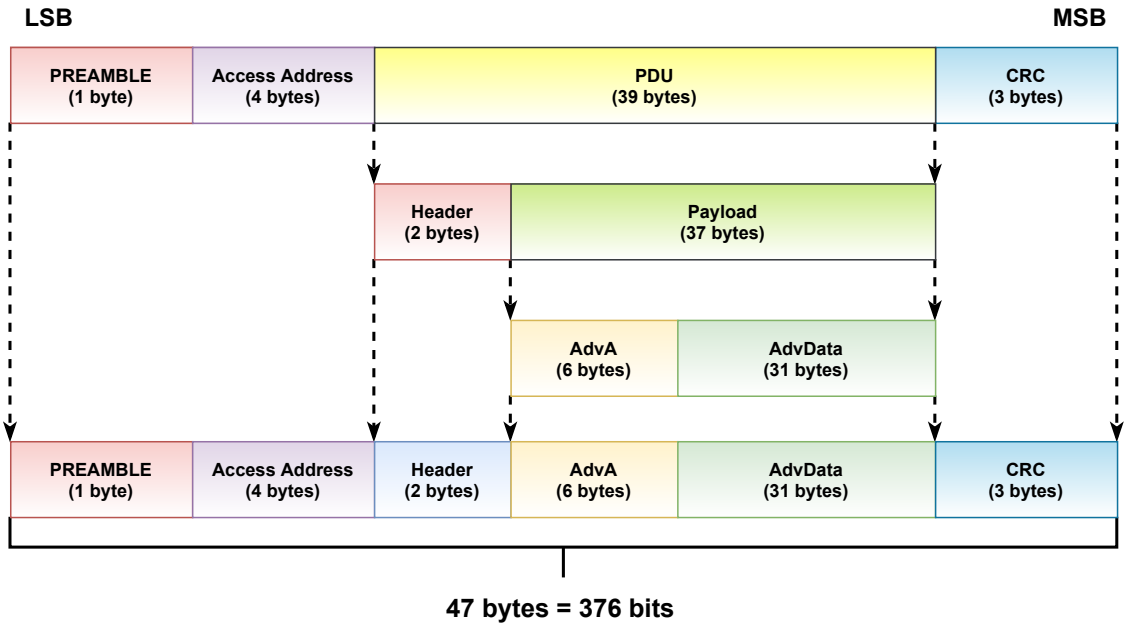


Figure 9: Complete on-air packet structure.

3 Preliminary Research

3.1 ADV bearer Performance Assessment

As a starting point for this thesis, I need to identify why the ADV bearer struggles to provide high throughput and data consistency capabilities. Section 2.2.1.2 describes the general communication scheme that is associated with the ADV bearer. The reason why legacy BTM struggles to provide these capabilities is caused by how the on-air communication is conducted. The ADV bearer utilizes BLE advertising packets to convey data between devices. These messages are sent over the three advertising channels provided by BLE. There is no concept of links between communicating devices during a transmission, meaning that there is no session or lower layer concept of acknowledgment of message delivery. Each mentioned aspects contribute to constraining the capabilities of a BTM network. The main issues that cause the restricted capabilities are:

- Poor Utilization of the Physical Medium — In this scheme, the ADV bearer only utilizes three of the total 40 available channels that BLE provides (2.1.1.1). In order to receive an incoming message, a device must continuously scan for incoming advertisement packets. A device can only scan one of the channels at a time, meaning that the same packet must be transmitted on all three channels, one after the other, by the transmitting device. This means that the ADV bearer effectively uses a single channel for message transmissions. Compared to connection-based BLE communication, which uses 37 channels, this is quite poor utilization of the available physical medium. Consequently, the probability of internal noise disruption on the medium becomes higher, restricting the rate at which messages can be sent.
- Restricted Transmission Speed of Advertising Packets — As described in section 2.2.4.1, advertising packets may be issued at a maximum rate of one packet per 25 ms. In comparison, a connection based BLE link can operate with connection events as small as 7.5 ms, where it is possible to send a full packet per event [6, p. 7].
- Reliant on Redundancy — Since there is no concept of links or lower layer acknowledgment for message transmission, there is no inherent way to guarantee that a single message has arrived at its destination. This is handled by adding redundancy to the transmission, sending the same information several times in the hope that at least one of the attempts will succeed. This situation adds to the above-mentioned issue with poor utilization of the physical medium. Here we are potentially retransmitting information that already has reached its destination, causing a delayed transmission for any consecutive messages from the transmitting device. Redundancy may improve the chance for the successful transmission of a message, but it does not guarantee message delivery.

3.2 Device Firmware Update Use-Case

During the preliminary research, I looked for an application use case where both targeted performance metrics are of high importance. This research aimed to provide an application example that can be utilized for both simulation and performance assessment purposes. Through this work, I have found a use case where both throughput and data consistency are paramount. Additionally, this use-case is valid for all BTM networks, regardless of what functional application task the network is performing.

From time to time, an electronic device might require an update of its firmware. The underlying reason for this might be that there are bug fixes that have been applied to the initial firmware or that a new feature has been added. When BTM devices are shipped from the manufacturer, they will, in most cases, already contain the necessary firmware. This enables the installer to deploy the network without concerns to the device firmware. When the network is operational, the devices may be scattered across a large physical area, and the placement of the devices may vary greatly.

Some devices might be easily accessible, while others are placed in environments that require a great deal of effort to access after deployment. For instance, an example of this could be a BTM network mounted in the ceiling of a 12-meter high storage building, only accessible by lift. If these network devices require a firmware update, it would be cumbersome and costly to access each device to update the firmware manually. Fortunately, this issue can be solved through the use of a Device Firmware Update (DFU). A BTM DFU is a procedure where it is possible to utilize the on-air communication capabilities of the network to swap out the firmware of multiple devices at the same time. The DFU consists of a train of BTM messages, each containing a small fragment of the new firmware for the devices in the network. This message train is sent from an origin device with access to the complete block of data that compromises the new firmware. When all DFU messages have been received, the devices will reassemble the complete block of data and overwrite the old firmware memory. For a DFU procedure to succeed, the complete set of data that each device possesses at the end must be consistent with the block of data on the origin device. This implies that all devices must receive every single message in the transmission train. Additionally, the time it takes to transmit the entire DFU depends on the network's throughput capabilities.

A BTM DFU procedure is an ideal example of a task where both throughput and data consistency are most important. It also represents a universal feature applicable to any BTM network. On this basis, I have decided that the DFU use-case shall be the primary performance benchmark for this thesis.

3.3 Implementation Approach

3.3.1 Approach Research

To improve the targeted performance metrics, I must find a viable alternative to the ADV bearer scheme. In the search for this alternative, there are two criteria of importance. The first is that the alternative solution must solve one or more of the issues that were discovered for the ADV bearer in section 3.1. The second criterion is that the solution ideally should be possible to utilize without violating the existing BTM specification.

The first alternative that I explored was the concept of *extended advertising*. Extended advertising is a feature released as a part of Bluetooth 5, where it is possible to advertise more data than through legacy advertising. In this scheme, the advertiser can use the 37 data channels as secondary advertisement channels, thus allowing the advertiser to utilize a larger portion of the available medium. The concept here is that a regular advertisement packet is first sent on the three advertising channels. This packet contains a pointer to a secondary channel where additional data will arrive[11]. Extended advertising could solve a portion of the issues that are associated with the ADV bearer. In this case, we can utilize the physical medium of BLE to a higher degree since we are utilizing both the three advertising channels **and** the 37 data channels. There are, however, some issues associated with this approach. Since extended advertising was introduced as a part of Bluetooth 5, there is no guarantee that all existing BTM devices will support a solution where extended advertising is utilized. This is because BTM is available for all devices that support Bluetooth version 4.0 or higher. Choosing this implementation approach has a high risk of causing an incompatibility gap between new and older BTM devices, which would be concerning from a marketing perspective. Considering this, it is not surprising to see that extended advertisement currently is not a topic that is covered by the BTM profile specification[10].

Another alternative that presented itself was an approach where the BTM network utilizes link-based BLE connections over GATT to communicate. In this way, the network would benefit from all the features that conventional BLE connections inhabit. In GATT based connections, devices may utilize all 37 data channels that is provided by BLE(2.1.1.1). This greatly improves the utilization of the physical medium compared with the ADV bearer. All communication over GATT is acknowledged, meaning that there would be no need for redundancy to ensure that messages successfully arrive at their destination.

Furthermore, the maximum theoretical transmission speed over GATT is one message per 7.5 ms, which is four times faster than the allowed maximum transmission speed of the ADV bearer[6, p. 7]. If we review the shortcomings of the ADV bearer from section 3.1, we can see that the properties of a GATT connection solve the complete set of issues that was depicted here. If it is possible to implement a BTM network where the primary communication is performed over GATT connections, both the throughput and data consistency would most likely improve significantly. Then comes the question if it is feasible to implement such a solution with respect to the existing BTM specification. Fortunately, the specification already supports GATT-based connections in BTM networks. According to the BTM profile specification[10, p. 38], a *GATT bearer* is provided to allow devices that can not support the ADV bearer to participate in a BTM network. Through the associated *proxy protocol* the GATT bearer facilitates transmission of BTM messages over a GATT connection.

If we compare the two explored approaches, it seems clear that the GATT bearer approach is the most promising solution regarding the existing specification. As of today, the BTM specification does not mention anything about the use of extended advertising, indicating that this approach is likely to violate the existing specification. Based on these considerations, I have decided to continue the implementation research focusing on the GATT bearer approach.

3.3.2 Communication Over GATT

The main alteration required to realize the new GATT bearer solution is that the existing GATT bearer must be altered to possess similar capabilities as the ADV bearer provides. The most important feature is the ability to transmit and relay messages over several bearer interfaces simultaneously, allowing the network's messages to flow freely.

To evaluate if it is possible to achieve the desired behavior without breaking the spec, I have consulted the BTM profile specification. As described in section 2.2.3.1, the proxy protocol provides all the means for allowing full BTM communication over GATT. The only obstacle that the proxy protocol presents is that the proxy filter of a proxy server by default should be an empty whitelist. This would potentially deny free communication between devices in the network if not handled since the whitelist filter would block most messages from being transmitted from a proxy server. This issue can, however, be solved. By sending a proxy configuration message from the connected proxy client, the filter can be altered to an empty blacklist. This filter type allows all BTM messages to flow freely as long as the address of the message is not contained in the blacklist. Since the blacklist is empty, any message is passed from the server to the client. At this point, I am quite certain that a proxy client/server pair can provide the same communication capabilities as the ADV bearer.

However, a single proxy client/server bond only represents the communication between two specific devices in a network. To fully replicate the ADV bearer, the new implementation relies on several GATT connections to ensure that the BTM messages are allowed to flow freely in the network. Fortunately, the specification does not restrict how many GATT bearers a single device can support. Therefore, it is possible to create a BTM network consisting of multiple GATT connections without violating the specification. A notion of how this could look is exemplified in figure 10.

At the point where all the necessary GATT connections are established in a network, the last thing that must be handled is to ensure that all transmission and relaying activity from the upper layers of the BTM stack is passed to the GATT bearer interfaces. These messages must be transmitted by any GATT server or client bearer that is present on a device. This behavior is already inherent for the server roles and therefore needs no alterations to act as intended. For the GATT bearer client, the specification does not define any specific behavior regarding how and when BTM messages shall be transmitted over their interface. To allow the GATT bearer client to transmit messages over the GATT connections freely, I need to implement a function that sends a BTM message over all active client interfaces. This function must be added to all existing modules where the BTM stack normally would send a message over ADV bearer.

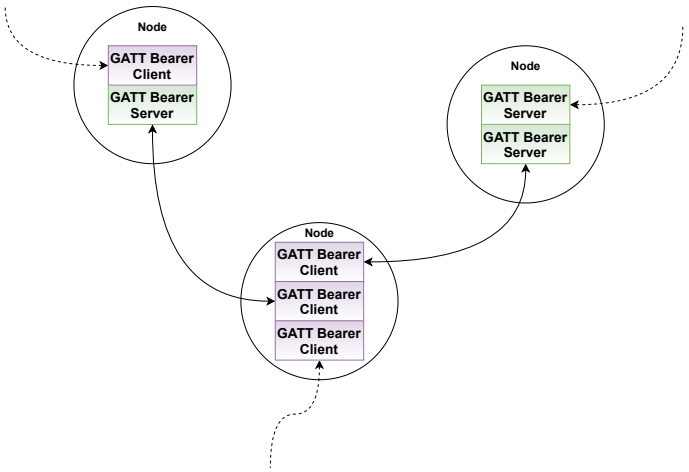


Figure 10: GATT connection topology example.

3.3.3 Control of the GATT Network

To establish and maintain a network that utilizes the GATT bearer solution, I am dependent on a set of control mechanisms that provides remote control of the GATT connections in the network. Such mechanisms are typically provided by the configuration foundation model [10, p. 197]. The most crucial feature that my implementation requires is a configuration message that allows a proxy client to initiate and maintain a connection to a target proxy server. This is not provided by any existing model that is defined by the BTM specification.

In this regard, my only option is to implement the control features of the implementation as a custom vendor model. This situation has an upside and a significant downside. The positive aspect is that a vendor-specific model allows me to customize the control features to accommodate the precise needs for the implementation. Additionally, a vendor-specific model is not technically a violation of the specification. However, using a vendor-specific model has a large impact on the market value of the implementation. A vendor model is often a custom model created by a single manufacturer and is in most cases only compatible with devices that that specific vendor creates.

Nevertheless, I have decided to continue the implementation work as planned, using a vendor-specific model to provide the configuration features of the implementation. In section 7.2.2 I discuss the implementation of the features for this model if the specification facilitates the means to provide equal operations and how the specification could be altered to accommodate the same features.

3.4 GATT Network Topology

Throughout the exploration of the GATT bearer, it has occurred to me that this approach can provide more than one solution for the BTM network topology. A straightforward approach would be to utilize the GATT bearer exclusively for the communication between all participating devices in the network. From this point, this will be referred to as the *Homogeneous approach*. Another solution is to utilize a combination of both bearers. A subset of devices forms the main communication path in the network, enabling non-adjacent devices to communicate with each other. These devices will act as relay nodes, where the communication between relaying devices is conducted using the GATT bearer. The remaining devices will act as legacy BTM nodes, where the communication with the relay nodes is conducted over the ADV bearer. From this point, this will be referred to as the *Heterogeneous approach*.

Both approaches possess certain advantages and drawbacks. The homogeneous approach will most likely be simpler to implement since it will not require interfacing with the ADV bearer.

On the other hand, this solution will require extensive configuration to form the network since all communication will require establishing a GATT connection. An additional concern is that a GATT connection requires one of the devices to take the role of the GAP central.

As described in section 2.1.2.1, a GAP central requires significantly more resources than a GAP peripheral. In a situation where a single device needs to inhabit the GAP central role for many connections simultaneously, it might result in a situation where the device cannot cope with the workload. For the heterogeneous approach, these considerations are of less concern. Since only the relay nodes of the network will utilize the GATT bearer, the number of GATT connections that need to be maintained is likely to be reduced. This approach would also be possible to configure/alter the BTM network without major configuration overhead. The potential drawback with the heterogeneous approach, in addition to a more intricate implementation, is that the combination of bearers might affect the enhancement we want to achieve for the throughput and reliability of the BTM network.

For this thesis's software implementation, I will strive to create functionality that can accommodate both of these suggested solutions. The evaluation and assessment of the heterogeneous and homogeneous approach are further discussed in section 7.3 of this thesis.

4 High Level Modeling and Simulation

Before implementing the GATT bearer solution, I have decided to do high-level modeling of the network solution. The model will also include the behavior of the ADV bearer. Together, these models will provide the basis for collecting simulation data for both implementations. By conducting simulations of different network topologies, I will be able to compare and assess the performance of both bearers. The purpose of this work is to obtain data that can provide a deeper understanding of how the network will behave under a real-world situation, potentially contributing to better design choices for the actual implementation. Simultaneously, it will estimate the performance of the GATT bearer implementation and how it compares to the legacy implementation.

4.1 Simulation Environment

To create the simulation model, I have decided to use Python as the chosen implementation language. Python is a software language that I am already familiar with, and in my experience, it is a suitable language for high-level modeling. Further, I have decided to utilize the NetworkX package. NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks[7]. I believe that this package will provide means that will make it easier to conceptualize a network of simulated BTM nodes. The complete simulation module will be composed of custom functionality implemented by me and existing functionality provided by the NetworkX package.

4.2 General Simulation Conditions

The simulations for both bearer solutions are performed with a couple of shared parameters and conditions. As mentioned in the introduction, one of my main goals for the implementation is to enhance throughput and data reliability in a BTM network. With this in mind, I have decided to simulate a device firmware update for an entire BTM network. The DFU will pass a relatively large block of data into the BTM network from a single entry point. The intention is to pass all data from this block to all nodes in the system. The most vital criterion in this operation is that every node in the network receives the entire data set. A secondary measure is that this distribution of data should take as little time as possible.

An essential general condition I have chosen for the simulation is that the virtual simulation network will represent nodes with relay functionality. This has different implications on the simulation result, depending on which approach we are studying.

For the ADV bearer solution and GATT bearer solution where we use the heterogeneous approach, this decision has a specific impact. Here the simulation will not represent data transmission to all end-points within a DFU procedure but rather to all relay nodes in that particular network. For a GATT bearer solution where we use the homogeneous approach, this has no impact on the simulation. In this instance, all participating devices in a network will effectively act as relay nodes since all communication must be transmitted over the GATT bearer.

The same simulation result for a network using the GATT bearer will be valid for heterogeneous and homogeneous solutions. It is only the perspective that changes. For the heterogeneous solution, the results represent the data transmission statistics for transmitting the data to all relay nodes, not including the transmission to the standard nodes. For the homogeneous solution, the result represents the complete result. Concerning the differences between the heterogeneous and homogeneous solution, it is crucial to keep in mind that the simulation model is intended for comparing the ADV bearer solution against the GATT bearer solution in general, regardless of which of these two approaches we are using. The merits of these two approaches will be discussed in section 7.3 of this thesis.

I have decided to include only the relay nodes because the final hop from a relay node to a regular node member will be common for both of the bearers. Since this factor is common, I have decided to leave it out of the model for simplicity reasons. The impact this might imply regarding the internal noise calculations is described in section 4.4.2.3.

In a real-life DFU, every node in the network must receive every single byte of data. In the ADV bearer solution, there is always a chance that a message won't arrive at all nodes on the initial attempt. Since this is an unacceptable situation, most DFU routines are split into several sub-routines. At the end of each sub-routine, the origin node of the DFU requests a status update from all nodes in the network. Suppose any node is missing one or more packets from the message stream of the previous sub-routine. In that case, the origin node will retransmit these packets to ensure data consistency on every single node in the network. This verification procedure is, however, costly concerning time consumption.

We can assume that this procedure adds a certain amount of overhead regardless of if all nodes have received all data in the previous sub-routine. This is because the origin must communicate with every node to verify that they have received all data. Further, if messages are missing from one or several nodes, the origin must retransmit these into the network once more and verify that the nodes now have received all the messages. Retransmitting and verifying the arrival of messages to nodes that did not receive all messages initially could, in some cases, prove to be challenging. It might require several attempts before actually mending the issue. This is because the communication during this verification routine uses the same communication channels as the actual DFU. If there are nodes that are missing messages, likely, they do not have an optimal path of communication to the origin. If this is the case, it is more likely that messages sent during the verification procedure are in transmission, including request, acknowledging, and retransmission messages.

To summarize, there is currently a need to periodically verify that the DFU is going according to plan when using the ADV bearer. This adds a certain amount of overhead to the total execution time of the entire DFU. In cases where one or more nodes are struggling to receive messages on the initial transmission attempt, it could prove challenging to communicate with them during the verification procedures, thus prolonging the execution time even more[16]. I choose to leave the verification and retransmission procedure out of the model. The reason for this is that the way the verification procedure is executed is an implementation-specific decision. In the model, I instead focus on how many messages are lost during standard DFU transmission. My statement is that a large magnitude of lost messages in this phase will imply that there would be required a substantial amount of execution time overhead to ensure data consistency on all nodes.

4.3 Modeling Transmission behavior

4.3.1 Common Behavior

Both solutions have some common behavior regarding transmission behavior.

4.3.1.1 Message Cache

Both solutions are fitted with a message cache to avoid nodes handling and retransmitting messages that have already been handled. All unique messages contain a separate transaction ID (TID), which is used to identify them. After successfully receiving a message, the node will do a lookup in the message cache to see if the TID is already present. If it is, the incoming message will be discarded. If it is not present, the incoming message will be accepted and handled for further relaying, and the cache will add the message's TID to the message cache. To avoid the message cache from growing infinitely large, it is designed to grow up to a predetermined configurable size before it starts acting as a FIFO, discarding the longest living entry to make room for a new one.

4.3.1.2 Message buffer

Both solutions utilize a message buffer. Whenever a node successfully receives and handles an incoming message that ought to be relayed, this message is put at the end of the message buffer. At the start of a new transmission event, a node will fetch an entry from this buffer and attempt to transmit it. Further details of how this buffer is handled are different for the two implementations. The size of these buffers can be pre-configured at the initialization of the simulations.

4.3.1.3 Connected Nodes & Adjacent Nodes

To fully model the individual impact nodes have on each other, I have introduced the concept of connected- and adjacent nodes. For the ADV bearer, these two are, for all practical purposes, the same since there is no concept of connection links in legacy BTM. For the GATT solution, however, there is a clear difference between adjacency and connections. While a GATT bearer node might have only a few connections, it might have several adjacent nodes. This has an impact on the calculations of the internal noise of the system, further explained in section 4.4.2, and therefore it is essential to distinguish between adjacent nodes and connected nodes.

4.3.2 ADV bearer Solution

For the ADV bearer solution, a single message is transmitted to all adjacent nodes simultaneously. The number of attempts the transmitting node will try to propagate a single unique message is decided by the internal transmission state of the individual node[10, p. 153]. If E.g., the transmission parameter is set to three, the ADV bearer will transmit the message thrice before the message is discarded from the message buffer. The messages populating the buffer is handled at the peak theoretical rate of 25 milliseconds per message(4.4.2.1). This means that as long as the buffer is not empty, the node will advertise a new message every 25 ms. The ADV bearer will always fetch the messages from the buffer in a first-in, first-out manner, discarding the front entry of the buffer as soon as it has been transmitted the required amount of times. At the time of transmission, the likelihood of successfully transmitting the outgoing message is decided by the receiving node(s), depending on their uniform loss chance and the internal noise calculations. Since the ADV bearer does not have any concept of acknowledging successful transmission, the failure of passing a message to one or several of the neighboring nodes will not affect the transmitting node. As soon as a message has been transmitted the required amount of times, the message is discarded from the buffer without regard to if the message has been successfully received or not.

4.3.3 GATT Bearer Solution

In the GATT bearer solution, the propagation of messages throughout the network is done differently than for the ADV bearer. First, all messages in the GATT bearer are acknowledged, meaning that the transmitting node will identify if the bearer successfully transmitted an outgoing message to a neighboring node.

While the ADV bearer will advertise the outgoing message to all nodes within radio proximity as a single advertising event, the GATT bearer must pass every message to neighboring nodes as individual events. This is because the GATT bearer operates on different channels for each connection. However, these events can co-exist, so for all practical purposes, these events can be modeled as a single event in time. However, this difference in transmission behavior brings forth a case that needs to be addressed. While the ADV bearer will advertise a message to all neighboring nodes regardless, the GATT bearer can choose which of the nearby nodes we wish to relay the message to. When an ADV bearer relays a message, one of the potential receivers will include the node which was the origin of this message in the first place. If the origin node successfully receives this rebound message, it will find the message TID already present in the message cache and thus

discard it. This is an innate behavior for the ADV bearer, which fortunately does not come at an extra cost for the transmitting node. However, imitating this behavior for the GATT solution would be utterly pointless since we with certainty know that the relay message's origin node has already received and handled the message in question. Therefore, I have decided that the passing of messages in the GATT solution will exclude the node which was the origin of the message when relaying it. In an actual implementation, this feature would require a routing mechanism on the bearer layer. I believe that a routing feature could obtain this feature in an actual implementation with a reasonable amount of effort, and I have therefore chosen to add it to the model.

The rate at which the GATT bearer can transmit messages is dependent on the length of the connection interval in BLE. According to the BT Core Specification a connection interval can be in the range of 7.5 ms to 4 s[8, p. 1396]. To model the GATT transmission behavior, I have decided to use a connection interval of 20 ms. A single connection interval will represent the time it takes to pass a message over all connections to a neighboring relay node, except the origin node of the message. Within this window, the receiving node will pass an acknowledge message, telling the sending node if the transmission was successful or not. As long as a message has not been successfully transmitted over all the required links, it will remain in the message buffer. Since each GATT transmission within the connection interval can be considered individual events, one or more connections might fail to transmit a message while others were successful. If one connection fails in transmission, it will have to retransmit the same message at the next available connection interval, while the others may continue to the next message. This means that each connection can be handling the same entries at different connection intervals. However, if one connection starts to lag too far behind the others, it might cause the message buffer to go into saturation. While one single connection has yet to transmit a message successfully, the buffer can not remove the entry. This will prevent new messages from entering the buffer, effectively stopping the connections on track from transmitting any new messages. They must wait until the failing connection eventually successfully transmits the message occupying the front of the buffer, thus freeing space for a new message.

On the other hand, losing messages due to buffer overflow on the receiver is not a situation that is acceptable in this regard either. Fortunately, the introduction of flow control can solve this problem. Flow control is an opportunity that the GATT bearer presents that is not possible with the ADV bearer. All GATT messages are acknowledged, which enables the opportunity to give meaningful feedback to the sender within a communication interval. One of the opcodes in the acknowledgment messages can be utilized for flow control, making it possible to tell the sending part in a transmission that the receiver does not have space in its buffer to handle the incoming message. When the sender receives this ACK, it will keep this message entry stored and transmit it later.

4.4 Modeling Noise

The modeling of noise enables introducing non-ideal conditions to the simulation environment and is thus crucial to the model. I have chosen to introduce the noise to the model as the sum of two different factors; the uniform noise and the internal noise.

4.4.1 Uniform Noise

The uniform noise is a simplified representation of all external factors that might harm the radio transmission capabilities of the nodes in the network. This represents factors like:

- Physical distance between nodes — The quality of the radio signal will deteriorate as the distance between two communicating nodes increases.
- Obstacles between nodes — The quality of the radio signal is affected by physical objects in the traveling path of the radio signal, like walls of different materials, windows, etc.

- Noise from third party sources — Since BLE utilizes the ISM-band(2.1.1.1) numerous applications might be using the same radio frequencies at the same time. One common example of this is Wi-Fi, which also utilizes the ISM-band. There is also a significant chance that other BLE applications might be in proximity of the BTM network since BLE has become a widespread standard for various applications. These other applications may affect the quality of the communication between nodes.

The uniform noise parameter is set at initialization of the simulation and is for simplicity common for all nodes in the network.

4.4.2 Internal Noise

Modeling the internal noise of the system is an essential factor in the simulation. Neglecting to account for internal noise would bias the simulation. A node would have an everlasting higher probability of receiving a message as the number of other nodes within radio proximity increases, the reason being that a message has an increasing number of possible paths a message can arrive through. In reality, there is a compromise between the number of links the node gains from adjacent nodes and the internal noise that the added link produces. For node A to receive an incoming message from an adjacent node B, there can be no other on-air traffic on the channel while the message is transmitted. As soon as another adjacent node C tries to utilize the same channel within the transmission window, both the message from node B and C can be regarded as lost.

4.4.2.1 ADV bearer Solution

To model the probability of losing a message due to internal noise, we must find the probability of a single node not transmitting on a channel at a given time.

We denote the time spent sending on-air data within an advertising period as ϕ_t. During an advertising period the node will transmit the message on all three of the BLE advertising channels. Since we are interested in the probability of the node sending on a single channel, we can choose to disregard this. This means that we must find the time spent transmitting on a single channel during an advertising period. From section 2.2.4 we know that an on-air BTM message will contain 376 bits, and that the LE 1M PHY bitrate of one megabit per second provides the ability to transmit one bit per microsecond, further denoted as T_{rate}. This gives us the following result for the on-air time of a single packet on a channel:

$$\phi_t = \frac{\mathrm{Bit_{cnt}}}{\mathrm{T_{rate}}} = \frac{376\,\mathrm{bit}}{1\,\frac{\mathrm{bit}}{\mu\mathrm{s}}} = 376\,\mu\mathrm{s} \tag{2}$$

From section 2.2.4.1 we know that the minimum advertising period is 25 ms. By considering the result from equation 2 we can now find the fraction of the total advertising period spent sending data on a single channel. This fraction is equal to the probability of transmitting on a channel at an arbitrary time within A_{min}. We will denote this as $P(\alpha)$.

$$\mathrm{P}(\alpha) = \frac{\phi_t}{\mathrm{A_{min}}} = \frac{376\,\mu\mathrm{s}}{25\,\mathrm{ms}} = 0.01504 = 1.504\% \tag{3}$$

Following equation 3, the probability of **not** transmitting on a channel at an arbitrary time within A_{min} is:

$$\overline{\mathrm{P}(\alpha)} = 1 - \mathrm{P}(\alpha) = 1 - 0.01504 = 0.98496 = 98.496\% \tag{4}$$

If we assume that a node is continuously transmitting data, this would mean that there is a *1.504%* chance that the node is utilizing the on-air medium of a single channel at any given time. This means that there is a *98.496%* chance that a node is **not** sending on that particular channel.

The issue is that it is not a reasonable assumption to think that a node will be continuously transmitting data throughout an entire DFU. The portion of the time spent transmitting packets will largely depend on retransmissions, network topology, and packet losses in the network. To more appropriately model the chance of a node transmitting data, we must consider how many packets have been transmitted within a given period previous to the current timestamp.

Let the probability for a node N not transmitting on a channel be denoted as $P(N)$, the given time period as ΔT, and the packets transmitted within this time period as n. This gives us the following equation for $P(N)$:

$$\mathrm{P(N)} = \frac{\mathrm{n} \cdot \phi_\mathrm{t}}{\Delta \mathrm{T}} \tag{5}$$

The probability of a node successfully receiving a message can be obtained by finding the probability that no other adjacent node is transmitting on the same channel simultaneously. To exemplify this we can review figure 11.

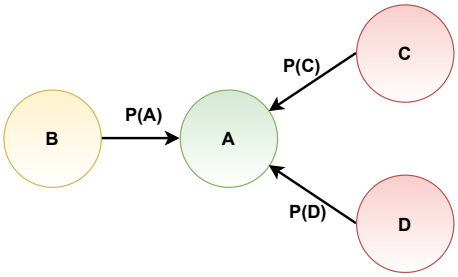


Figure 11: Example of the internal noise model.

The probability for node A successfully transmitting on a channel to Node B is $P(A)$. The probability for node C and node D not transmitting on that same channel is $P(C)$ and $P(D)$ respectively. $P(C)$ and $P(D)$ can be regarded as independent events, giving us the following calculation for $P(A)$:

$$\mathrm{P(A)} = \mathrm{P(C} \cap \mathrm{D)} = \mathrm{P(C)} \cdot \mathrm{P(D)}. \tag{6}$$

A generalized formula for calculating the chance of successful transmission due to internal noise is as following:

$$\mathrm{P(S)} = \prod_{\mathrm{i}=1}^{\mathrm{j}} \mathrm{P(N)}_{(\mathrm{i})} \tag{7}$$

Here $P(S)$ is the probability of successfully transmitting the message, j is the number of other links that is connected to the receiving node and $P(N)_{(i)}$ is the probability for each of these nodes not sending in that same time window respectively. The equation can be extended by inserting equation 5:

$$\mathrm{P(S)} = \prod_{\mathrm{i}=1}^{\mathrm{j}} \frac{\mathrm{n}_{(\mathrm{i})} \cdot \phi_\mathrm{t}}{\Delta \mathrm{T}} \tag{8}$$

4.4.2.2 GATT Bearer Solution

The model described in section 4.4.2.1 assess the transmitting behavior of the ADV bearer. Accurate modeling internal noise for the GATT bearer solution is significantly more complex. It operates on a higher number of channels, and it utilizes frequency hopping to decrease the probability of on-air collisions(2.1.1.1). To simplify the complexity of the simulation environment, I have decided to use the model for advertisement bearer solution as the basis for the GATT internal noise model. While admitting that this will not give a realistic scenario for the internal noise in the GATT solution, it is reasonable to think that it at least is not biased in the GATT solutions favor.

One of the main differences between the GATT and ADV bearer is that the ADV bearer uses the three advertisement channels to transmit data. During transmission, there is, in reality, only one of these channels that are being used to convey data. While the transmitting node will broadcast the same message on all three channels, the receiver will only scan one of the channels at a time. The GATT solution has the remaining 37 data channels at its disposal, and in this case, the network can make full use of every single channel. By assessing these facts, I have come to the following model for the internal noise of the GATT bearer solution:

$$P(S)_{Gatt} = \prod_{i=1}^{j} \frac{n_{(i)} \cdot \phi_t}{\Delta T \cdot \omega} \tag{9}$$

Here ω denotes the number of available channels, which as mentioned is 37 in the case of the GATT solution. This equation is also valid for the advertising solution, wherein reality $\omega = 1$ and does not affect the calculation as such. Here we assume that the GATT solution uses the same amount of time within a connection interval to transmit on a channel. We assume that it randomly picks one of the 37 available channels, making the 36 other channels available for other transmissions. This greatly decreases the chance of interference between concurrent transmissions in the network.

4.4.2.3 Noise from the Regular Nodes

As mentioned in section 4.2, I have decided to leave any standard ADV bearer nodes out of the simulation environment. Standard nodes will only be applicable in a situation where we are comparing a heterogeneous network solution to the ADV bearer solution. In this situation, we may consider all participating nodes in the simulation as relay nodes.

To justify this concerning internal noise calculations, we must make certain assumptions about the network we are performing the DFU in. As both equation 8 and 9 shows us, the internal noise during transmission is determined by the probability of other adjacent nodes transmitting messages on the same medium at the same time. To make the simulation model valid, we must assume that there is minimal or no transmission activity in the network other than the transmissions associated with the DFU itself. Since regular nodes do not retransmit messages that they receive, there will be no TX action on their part with regard to the DFU itself. Then we must ask ourselves if it is reasonable to assume that there is none or minimal other traffic in the network during the DFU. If we review the lighting application domain, which represents the majority of the BTM applications used to date, the assumption is reasonable. In a BTM used solely for lighting control, the transmission rate in the network would, in the majority of cases, be comprised of the Mesh beacons[10, p. 119] and a handful of lighting commands per day. At least concerning the model, a transmission workload like this would have a minuscule effect on the internal noise.

While I have decided to leave the regular nodes out of the simulation model for simplicity, I will still assess how their presence would impact the ADV bearer solutions and the heterogeneous GATT bearer solution. The assessment will be done by means of the following example:

If we review the exemplified mesh network in figure 12, we can assess how the self-noise is impacted for both the GATT- and ADV bearer solution. In this figure, we can see relay node A

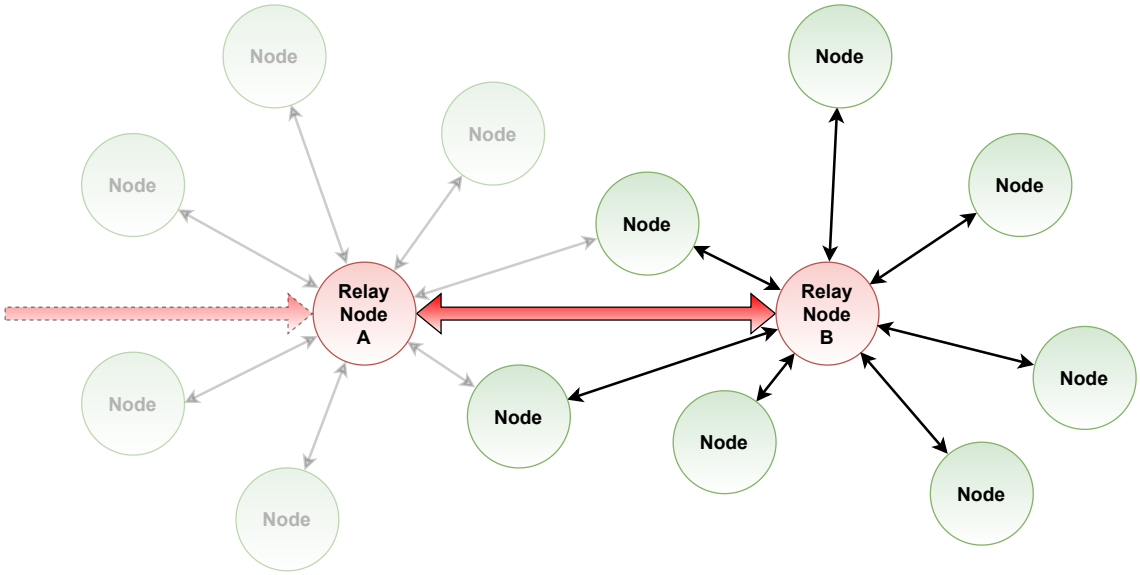


Figure 12: Regular nodes impact on internal noise.

and B, surrounded by several ordinary nodes that is also a part of the mesh network. The solid red arrow represents the link that is used to pass messages between the two relay nodes. This can either represent passing messages with the ADV bearer or the GATT bearer solution. The smaller black arrows represent links used by the regular nodes to communicate with the relay nodes. This communication is exclusively performed by using the ADV bearer.

Let us say that there is some additional transmitting activity going on in the network while performing a DFU. Relay node A is trying to propagate a DFU message to pass a message over to relay node B. Let us first assess the regular nodes' impact on the internal noise for relaying with the ADV bearer. In this case **all** communication, both relaying and regular node transmission is performed using the ADV bearer. All communication in the network shares the three BLE advertising channels, meaning that the regular node's transmission behavior affects the internal noise magnitude for the relay transmission as well. In the case of the mentioned transmission from relay A to relay B, all the seven regular nodes adjacent to relay node B affect the likelihood of successful transmission, depending on how much data traffic they generate. In the model, the impact could be calculated by using the established equation 8 from section 4.4.2.1. Still, as I have mentioned, I have chosen to leave this out of the simulation for simplicity reasons. Let us now briefly review how the regular nodes would impact the GATT bearer solution. Since the GATT bearer and the ADV bearer operates on independent BLE channels, the regular nodes have practically no impact on the internal noise in this case. This means that if regular nodes were a part of the simulations, they would not impact the internal noise in the simulations when using the GATT bearer solution.

If we are considering a simulation for the homogeneous approach, we may ignore the above considerations completely.

4.5 Simulation Output Data

To assess the outcome of simulations of the different solutions, it is paramount to monitor a multitude of parameters for the relay network in its entirety and the individual nodes. The most significant parameters in this regard are, of course, DFU execution time and the total number of lost messages for each node during the DFU. Each node keeps a record of the timestamp of their last successfully received message to provide the execution time. Simultaneously, the nodes will increment a counter tracking the number of unique messages they have received. By comparing this parameter to the total amount of unique messages injected into the network, we can derive

how many messages have been lost for each node. At the end of the simulation, these parameters can be extracted from the simulation setup.

Other relevant parameters that are monitored are:

- Peak buffer size — The model keeps track of the peak buffer size for both solutions during a DFU simulation. It does this by assessing how many unique message entries are present in a node's message buffer at any given time. However, the peak buffer size will never exceed the preconfigured maximum allowed buffer size.
- Mean loss chance — The mean loss chance is calculated based on the uniform noise parameter and the mean internal noise chance of each node, respectively. While the uniform noise chance is constant throughout the entire DFU simulation, the internal noise of a node changes dynamically according to the transmission behavior of the adjacent nodes as described in section 4.4.2.
- Number of links — The number of links for each node is calculated on initialization of the simulation. It indicates how many nearby nodes a node is communicating with.
- Number of adjacent nodes — The number of adjacent nodes for each node is calculated on initialization of the simulation. It indicates how many nearby nodes a node is within radio proximity with. In difference to the number of links, this parameter has implications on the internal noise for the node.

4.6 Simulation Results

4.6.1 Simulations for Network Alpha

The scenario for the simulation setup is shown in figure 13. It represents the relay nodes of a BTM network installed in an office building. The network consists of 20 nodes spread across the office floor. Nodes 0 through 17 have three links to other nodes, while nodes 18 and 19 have two, giving a reasonably even edge distribution throughout the network. We can imagine that the number of nodes within radio proximity of any given node in this instance is determined by a combination of the distance between them and the structural composition of the building's walls, windows, doors, etc. Further, I will denote this initial network configuration as the "α network".

In the simulations, I have decided that the DFU data will be represented by a 150 kbyte data block. The simulation will be utilizing unsegmented messages, meaning that each sent BTM network PDU can carry a payload of 11 byte[10, p. 63]. This gives us the following calculation of the total number of unique BTM network PDUs that will be needed to pass the data block throughout the network:

$$\text{Msg}_{\text{tot}} = \lceil \frac{\text{Data}_{\text{size}}}{\text{Unseg}_{\text{size}}} \rceil = \lceil \frac{150\,\text{kbyte}}{11\,\text{byte}} \rceil = 13637 \tag{10}$$

The origin device of the DFU is set to be node 0, located on the left side of the office. Due to external noise applications in the office area, every node has a 10% uniform chance of failure to receive any single message transmission. The total chance of failure for all nodes is the sum of this uniform noise and the internal noise associated with each node, respectively. When setting the transmission count for the simulations, the same value will be set for all nodes in the network. Initially, the maximum buffer size of each node is set to a value of *128* entries. This is done to ensure that we can observe the buffering behavior without any restrictions.

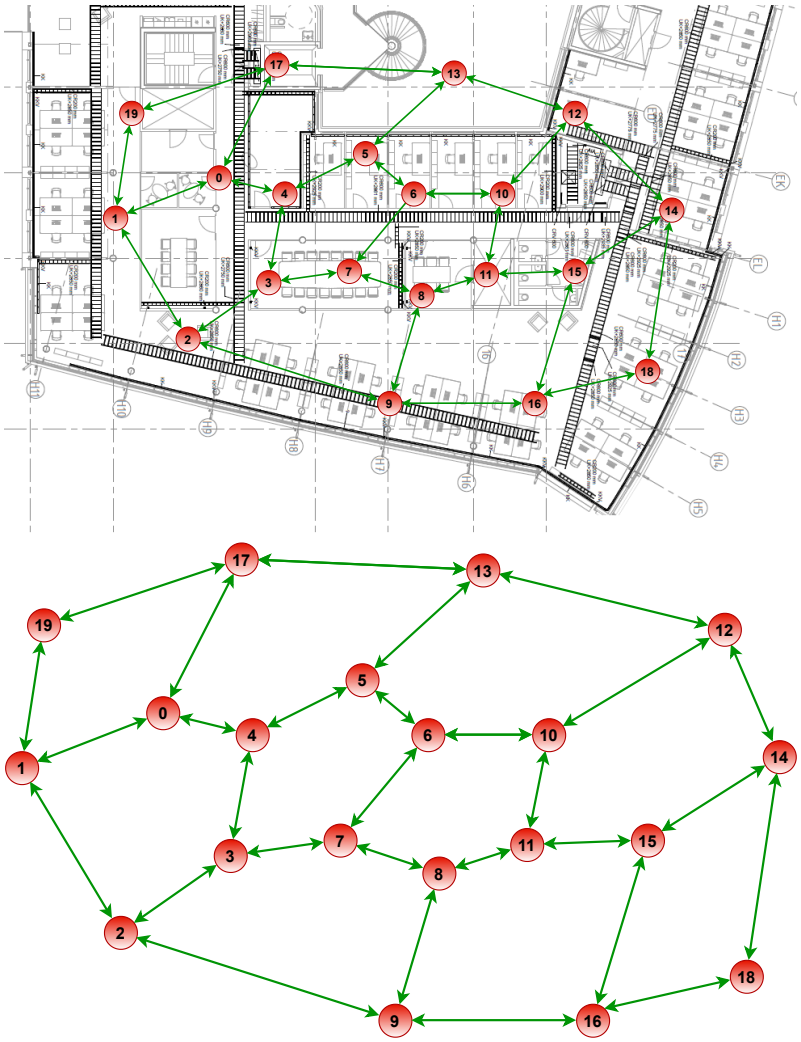


Figure 13: Overview of network α.

4.6.1.1 Advertiser Simulation with Single Transmit

I will start by simulating the DFU in network α using the ADV bearer with the transmission count set to the minimum value of one. The simulation is performed 100 times, and the accumulated result of these simulations is shown in table 1.

As we can see from the results, there is a significant amount of lost messages for every single node in the network, spanning from *0.761%* on node 17 to *2.975%* on node 18. Further, the results show that node 18 is the weakest link in the network. It is the node with the highest amount of message losses and the node with the highest timestamp of the last received message. By reviewing the topology of the network in figure 13, this result seems reasonable. Node 18 is located relatively far from origin node 0 and has only two links connecting it to the rest of the network. One interesting observation is that both nodes with only two links deliver the poorest performance in the simulation. Node 19 has the second-worst performance concerning message loss in the simulation, with a message loss of *2.017%*. Compared to node 18, node 19 is located relatively close to the origin node. If we compare the message loss of node 19 with, E.g., the results for node 14, which is placed significantly farther from the origin, we see that node 14 still outperforms node 19 with a message loss of only *1.728%*. This might indicate that the number of links to a node has a more significant impact on the chance of successful transmission than the number of hops required to propagate a message. If we review the peak buffer size of the simulation, we can see that it spans from *7* to *9* entries for the nodes in the network. My conclusion is that the magnitude of these buffers is within the acceptable range concerning the resource demand it would require to realize them for each node.

The timestamp for the last message that was transmitted from origin node 0 is 340 925 ms. If we ignore the latency, which is negligible in this regard, the time to execute this DFU simulation took approximately 5.7 min to perform. However, this is the execution time before we account for the significant amount of lost messages in the network. In order for the DFU to be successful, every single packet **must** arrive at all the nodes in the network. With the overall performance of this particular simulation, it would be reasonable to assume that there would be consumed a substantial amount of time on status updates, retransmission, and verification of data consistency, as explained in section 4.2. It might even prove to be infeasible in an actual implementation.

Node ID	Number of links	Number of adjacent nodes	Single transmission loss chance (%)	TS last msg (ms)	Packets lost	Packet loss (%)	Peak buffer size
0	3	3	12.766	340925	0.0	0.000	0
1	3	3	12.564	341026	106.1	0.778	8
2	3	3	12.579	341121	140.5	1.030	9
3	3	3	12.669	341089	139.9	1.026	9
4	3	3	12.606	341041	115.3	0.845	8
5	3	3	12.691	341101	138.0	1.012	9
6	3	3	12.750	341143	154.2	1.131	9
7	3	3	12.749	341139	154.1	1.130	8
8	3	3	12.783	341185	167.2	1.226	9
9	3	3	12.701	341191	171.0	1.254	9
10	3	3	12.800	341187	166.9	1.224	8
11	3	3	12.840	341215	176.9	1.297	7
12	3	3	12.696	341185	171.2	1.256	9
13	3	3	12.618	341102	140.1	1.027	9
14	3	3	12.801	341234	235.7	1.728	8
15	3	3	12.890	341233	214.9	1.576	8
16	3	3	12.803	341227	235.4	1.726	8
17	3	3	12.580	341021	103.8	0.761	8
18	2	2	11.458	**341255**	**405.7**	**2.975**	7
19	2	2	11.420	341057	275.0	2.017	7

Table 1: ADV bearer simulation for network α, using single transmission.

4.6.1.2 GATT Simulation Without Buffer Restrictions

The next simulation for network α is performed using the suggested GATT bearer solution. The GATT solution utilizes all available links between nodes in this instance, imitating the same network composition as the ADV bearer. The simulation is performed 100 times, and the accumulated result of these simulations is shown in table 2.

As we can see from the results, the entire network received every single message of the DFU. This result was as expected due to the way the GATT bearer solution is modeled. Section 4.3.3 describes that a transmitting node will always initiate retransmission upon a failed attempt, thus ensuring that all messages will arrive at their destination eventually. In this instance, we can see that node 18 has the highest timestamp of the last received message, with a value of 305 329.4 ms. This gives an execution time of approximately 5.1 min. We note here that node 18 here again seems to be the weakest link in the network, just like in the previous advertising simulation.

However, if we take a look at each node's peak buffer size, we see that this simulation reveals a significant issue. The overall peak buffer size is extremely high, spanning from *71* entries for node 15 to *199* entries on node 1. For actual implementation, the magnitude of these buffers would be utterly unreasonable concerning the amount of resources it would require.

Node ID	Number of links	Number of adjacent nodes	Single transmission loss chance (%)	TS last msg (ms)	Packets lost	Packet loss (%)	Peak buffer size
0	3	3	10.090	304030.6	0.0	0.0	0
1	3	3	10.082	303146.4	0.0	0.0	199
2	3	3	10.082	304065.4	0.0	0.0	108
3	3	3	10.079	304095.8	0.0	0.0	137
4	3	3	10.084	303428.4	0.0	0.0	162
5	3	3	10.080	304120.0	0.0	0.0	139
6	3	3	10.079	304629.6	0.0	0.0	101
7	3	3	10.079	304622.4	0.0	0.0	117
8	3	3	10.077	304967.4	0.0	0.0	81
9	3	3	10.082	304642.6	0.0	0.0	118
10	3	3	10.077	304908.8	0.0	0.0	89
11	3	3	10.079	305165.4	0.0	0.0	86
12	3	3	10.079	304667.8	0.0	0.0	130
13	3	3	10.085	304133.4	0.0	0.0	114
14	3	3	10.081	305140.4	0.0	0.0	84
15	3	3	10.083	305262.4	0.0	0.0	71
16	3	3	10.085	305184.0	0.0	0.0	133
17	3	3	10.080	303385.8	0.0	0.0	168
18	2	2	10.042	**305329.4**	0.0	0.0	87
19	2	2	10.039	303768.6	0.0	0.0	87

Table 2: GATT bearer simulation for network α with no buffer restrictions.

4.6.1.3 Assessment of Initial Simulations

Let us now compare these first simulation results for the different solutions in network α. If we compare the DFU execution time, we can see that the two solutions had similar performance, with an execution time of 5.7 min and 5.1 min for the ADV bearer and the GATT bearer solution respectively.

However, this first comparison is without any regard to the issues associated with the two simulations. For the ADV bearer, the issue was the significant amount of lost messages in the simulation. As earlier mentioned, the actual execution time of this DFU will most likely have to be considerably larger to ensure that all nodes receive the complete DFU payload. In the GATT solution, we could see that every single message arrived at its destination, meaning that there is no additional overhead to consider in this case. The issue for the GATT bearer was that the buffers for each node grew so large that it would be infeasible to realize it in an actual implementation.

Based on this evaluation, I have decided that the simulation must be re-run for a more realistic scenario. For the ADV bearer, this implies performing multiple simulations with increasing transmission values. It would be more reasonable to compare the two solutions for a situation where the amount of lost messages is equal. This implies that we must find a configuration for the ADV bearer where the amount of lost messages is comparable to the results of the GATT solution. Since the GATT solution model guarantees delivery of every single message, we need to find a value for the transmission parameter that gives a simulation result where the ADV bearer loses approximately zero messages. We can disregard the execution overhead associated with lost messages for the advertising solution for such a situation. For the GATT bearer, the simulation needs to be run with restrictions on each node's maximum allowed buffer size. After these simulations are performed, we can assess the impact these alterations and restrictions have on the overall performance for both the advertising- and GATT bearer.

4.6.1.4 Advertiser Simulation with Increasing Retransmits

The new simulation setup for the ADV bearer is conducted using transmission values ranging from one to four. To simplify the output data representation, I have decided to present the output parameters for node 18 exclusively since this node consistently gives the worst performance metrics. The result of these simulations can be seen in table 3.

Retransmit count	TS last msg (ms)	Packets lost	Packet loss (%)	Peak Buffer size
1	341255	405.69	2.9749	7
2	682355.5	2.72	0.0199	5
3	1023243.5	0.04	0.0003	4
4	1364128.75	0.00	0.0000	3

Table 3: ADV bearer simulation for network α, using one through four transmissions. The results displayed are for node 18.

The simulation results show the effect the increasing transmission value has on the network performance. We can see that by just adding one additional transmission, we can reduce the number of lost messages to node 18 from *405.69* to just *2.72*, which is equal to a *99.33%* reduction. Simultaneously the execution time effectively doubles, from 341 255 ms to 682 355.5 ms. This increase in execution time is as expected since latency is negligible. The origin node now needs to send double the amount of messages, which translates to a doubling in execution time. While the results from the simulation with two transmissions vastly improve the performance, we can still expect node 18 to lose *2.72* messages on average. This means that we still would require special routines to ensure data consistency in the network. If we consider the performance when using three transmissions, we can see that the probability of message loss during the DFU is almost completely eradicated. Here the probability of losing a single message during a DFU is starting to become so small that most DFUs would complete without losing a single message. A message loss of *0.04* can be translated to four out of 100 DFU simulations losing a single message for node 18. Using a transmission count of three gives an effective tripling of the initial execution time, with a value of 1 023 243.5 ms. We can see that not a single message was lost during the 100 performed simulations when using four transmissions. By increasing the number of simulation iterations gradually, we would, of course, see that there is still a tiny chance of losing a message. However, for all practical uses, we can regard the message loss chance as zero. Here the execution time is increased to 1 364 128.75 ms.

Another observation that can be made from table 3 is that the peak buffer size decreases as the transmission value increases. The reason for this is most likely that, as the origin node needs to push out more and more duplicates of the same message, it leads to a situation where the other nodes have greater breathing room to propagate their message buffers further in the network.

4.6.1.5 GATT Simulation with Buffer Restrictions

Before re-running the α network simulation for the GATT bearer, I want to explore the reason for the wast buffer sizes that we experienced in the previous simulation. Opposite to the ADV bearer, a GATT bearer keeps messages in its buffer until they are successfully passed to **all** linked node, except the origin node of the initial message. We can review this by an example. If we take a look at figure 13 we can assess the transmission relationship between origin node 0 and node 1. Whenever a message is transmitted from node 0 to node 1, this message is put in the buffer. At the next possible connection interval, node 1 will try to forward this message to nodes 19 and 2. If either of these attempts fails, the message must be kept in the buffer until the next possible connection interval. If node 0 simultaneously successfully transmits the following message of the DFU, it will increase the number of buffer entries in node 1. To decrease the number of entries in this buffer again would depend on node 0 failing to transmit a future message, while simultaneously node 1 not failing to forward its messages. Since the probability of failing transmission for the origin and node 1 respectively can be regarded as independent events, this will fluctuate throughout the entire DFU. This will cause periods within the DFU where node 1 will be struggling to keep up with the origin, thus making the buffer size high in these periods. Another factor that might worsen this performance is if the origin node has a noticeably lower probability of transmission failure than the other node. In this instance, we can expect to see that the peak buffer size will continue to grow throughout the entire DFU procedure. A solution to this is to take advantage of the possibility of flow control that the GATT bearer provides. Section 4.3.3 describes how the acknowledgment messages in GATT can be used to halt the flow of messages whenever the buffer of a given node grows too large.

The second iteration of the GATT bearer simulation for the α network is performed with an identical configuration as in the first iteration, apart from the maximum allowed buffer size, which is set to *8* entries. This number is comparable to the buffer sizes we saw for the simulations of the ADV bearer and is within the acceptable range concerning the resources required to realize the actual implementation. The result of this simulation can be seen in table 4.

Node ID	Number of links	Number of adjacent nodes	Single transmission loss chance (%)	TS last msg (ms)	Packets lost	Packet loss (%)	Peak buffer size
0	3	3	10.101	312187.0	0.0	0.0	0
1	3	3	10.098	312112.0	0.0	0.0	8
2	3	3	10.099	312203.0	0.0	0.0	8
3	3	3	10.099	312201.2	0.0	0.0	8
4	3	3	10.099	312135.2	0.0	0.0	8
5	3	3	10.099	312194.4	0.0	0.0	8
6	3	3	10.098	312261.2	0.0	0.0	8
7	3	3	10.098	312268.4	0.0	0.0	8
8	3	3	10.098	312316.4	0.0	0.0	8
9	3	3	10.098	312295.6	0.0	0.0	8
10	3	3	10.098	312309.6	0.0	0.0	8
11	3	3	10.098	312355.0	0.0	0.0	8
12	3	3	10.098	312274.8	0.0	0.0	8
13	3	3	10.098	312174.4	0.0	0.0	8
14	3	3	10.097	312359.0	0.0	0.0	8
15	3	3	10.098	312378.2	0.0	0.0	8
16	3	3	10.097	312368.6	0.0	0.0	8
17	3	3	10.098	312079.2	0.0	0.0	8
18	2	2	10.049	**312400.6**	0.0	0.0	8
19	2	2	10.050	312128.2	0.0	0.0	8

Table 4: GATT bearer simulation for network α with a buffer restriction of maximum eight entries per node.

The results show us that even with a restriction on the maximum buffer size in each node, we still get a DFU execution time performance comparable to the results in the first iteration. In this instance, node 18 is still the node with the highest timestamp of the last received message with a value of 312 400.6 ms. This execution time is approximately 7 s slower than the initial simulation where we did not have any restrictions on buffer sizes, giving us a total execution time of 5.2 min

for the GATT bearer.

4.6.1.6 Simulation Summary for Network Alpha

Let us now summarize the findings for the different simulations in network α. If we review the results from the simulations of table 3, we can see that the ADV bearer has nearly equivalent message loss performance with the GATT solution if we use three transmissions. With this configuration, the ADV bearer has an execution time of approximately 17.1 min. Compared to the GATT bearers execution time of 5.2 min, we can see that the GATT solution has a performance that is roughly *3.29* times faster than the ADV bearer in network α.

4.6.2 Simulations for Network Beta

Let us now reconsider the original network in figure 13 from section 4.6.1. Due to some structural altering of the office space near node 12, it has suddenly lost its links to nodes 13 and 10. Figure 14 shows the network topology after the structural changes. Further, I will denote this altered network configuration as the "β network".

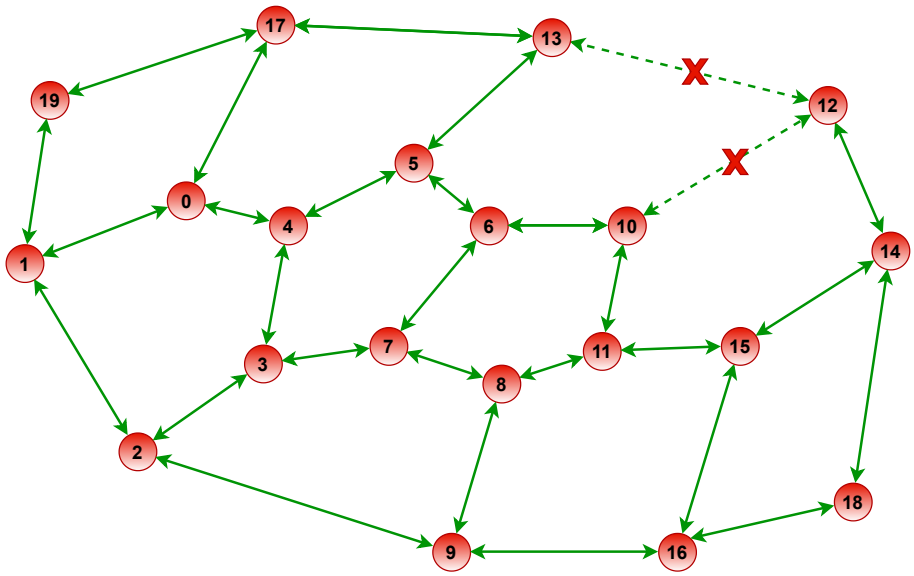


Figure 14: Overview of network β.

In comparison to the α network, this network has a higher degree of unbalance regarding the network connectivity. In this instance, three nodes have only two links, in addition to node 12, which is isolated with only one link to the rest of the network. I will now repeat a similar simulation setup for network β as the ones in section 4.6.1.

4.6.2.1 Advertiser Simulations

The previous simulations for network α showed us that a simulation for the ADV bearer using four transmissions resulted in zero messages lost for 100 simulations of the DFU. Using the same circumstances, I have performed the same simulation for network β. The results from the simulation can be viewed in table 5.

By reviewing the results, we can see that the structural change of the network topology has made a noticeable impact on the network's performance. While the execution time, with just a 168.25 ms increase, is comparable to the equivalent simulation for the α network, we see that the amount of

Node ID	Number of links	Number of adjacent nodes	Single transmission loss chance (%)	TS last msg (ms)	Packets lost	Packet loss (%)	Peak buffer size
0	3	3	12.960	1363700.0	0.0	0.0	0
1	3	3	12.603	1363629.0	0.0	0.0	3
2	3	3	12.511	1363769.3	0.0	0.0	4
3	3	3	12.520	1363775.0	0.0	0.0	4
4	3	3	12.595	1363628.3	0.0	0.0	3
5	3	3	12.543	1363775.5	0.0	0.0	4
6	3	3	12.532	1363893.5	0.0	0.0	3
7	3	3	12.681	1363893.5	0.0	0.0	3
8	3	3	12.606	1364001.3	0.0	0.0	3
9	3	3	12.579	1363889.0	0.0	0.0	3
10	2	2	11.314	1364015.8	0.0	0.0	3
11	3	3	12.726	1364078.3	0.0	0.0	3
12	1	1	10.0	**1364294.0**	**1.34**	**0.0098**	2
13	2	2	11.387	1363767.3	0.0	0.0	4
14	3	3	12.668	1364214.8	0.0	0.0	3
15	3	3	12.672	1364134.3	0.0	0.0	3
16	3	3	12.629	1364018.3	0.0	0.0	3
17	3	3	12.615	1363629.5	0.0	0.0	3
18	2	2	11.344	1364141.0	0.0	0.0	3
19	2	2	11.497	1363759.3	0.0	0.0	3

Table 5: ADV bearer simulation for network β, using four transmissions.

messages lost for node 12 has dramatically increased in the β network simulation. We can observe that the majority of the nodes in the network are still receiving every single message. However, by reviewing the performance of node 12, we can see that it loses on average *1.34* messages per simulation, which is a substantial increase compared to the equivalent simulation for the α network.

Let us now assess the relationship node 12 has with the rest of the network to comprehend the relatively poor transmission behavior. Node 12's sole connection to the rest of the network is through node 14. Node 14 did not lose a single message for any of the 100 simulations. With an average message loss of *1.34* per simulation, node 12 lost approximately *1340* messages in total over the 100 performed simulations. All these *1340* messages **must** have been lost in the transmission from node 14 to node 12. Note that node 14 in this instance has made four attempts to transmit each of these messages to node 12.

As also discussed in section 4.6.1, we yet again experience that the number of links that connect a node to the rest of the network seems to have a great impact on the transmission performance. To better understand this, I have created a simple simulation setup that illustrates the transmission behavior of a single linked chain network using the ADV bearer. This network is shown in figure 15.

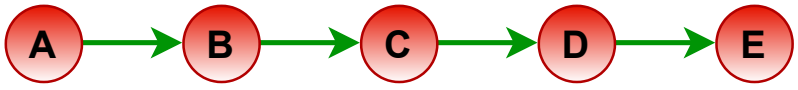


Figure 15: Advertising chain example network.

This simulation is performed as a 100 iteration simulation, with node A as the origin. To make the result easier to interpret, I disabled the internal noise generation for this simulation session. This means that the noise in this simulation is equal for all nodes, at a value of *10%* loss chance. In this simulation, we are still using the DFU consisting of *13637* unique messages. The results are presented in table 6.

Let us first review the column for node B in this simulation. Node B is stationed only one hop away from the origin node. Still, we can see that the messages lost for this single hop are significant for transmitting counts spanning from one through three. For a transmission count of four, we can recognize a similar result to the one we found for node 12 in the β network. At a transmissions count of five, we can see an average message loss that implies that most DFU will execute without

Retransmits	B packet loss cnt	C packet loss cnt	D packet loss cnt	E packet loss cnt	Time to complete (min)
1	1365.68	2590.83	3689.43	4683.75	5.68
2	137.13	270.56	401.17	533.08	11.37
3	14.31	28.48	42.97	56.18	17.05
4	1.35	2.61	4.07	5.60	22.73
5	0.12	0.32	0.49	0.59	28.41
6	0.0	0.01	0.02	0.03	34.10
7	0.0	0.0	0.01	0.01	39.78

Table 6: Result summary for the advertising chain simulation. The simulation is performed for one through seven transmissions.

message loss and a transmission count of six before no messages are lost at all in this first hop. If we review the development for the transmission performance as the messages are passed to the remainder of the nodes, we can see that the messages lost increase for each hop in the chain. We can also see that the number of lost messages for each hop declines as we move farther from the origin node. This result is as expected since node B will be able to successfully transmit an equal fraction of its successfully received messages as the origin could do for the entire DFU message count. Since the number of messages that node B will transmit is less than the complete message count, it is clear that the number of messages that will be lost during transmission to node C will be less than the amount lost in the first hop.

This implies that it is the first hop that defines the further development in the chain of nodes. If we review the overall results for the simulation where six transmissions were used, we can see that the amount of messages lost between each new hop does change notably.

So why does a single-linked node perform so much worse than a node with two links? To explore this, I have produced another simple setup to show the behavior of a node that has two links to the rest of the network. This setup is displayed in figure 16.

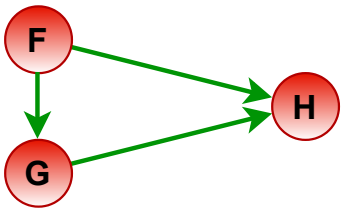


Figure 16: Advertising triangle example network.

The setup for this simulation is similar to the setup for the previously linked chain network simulation. The difference for this setup is that node F is the origin for the DFU, and the transmission loss chance for node G has been configured to be zero. This implies that node H will have the opportunity to receive the entirety of the DFU from both its links. The results from this simulation is presented in table 7.

Retransmits	H packet loss cnt	Time to complete (min)
1	135.76	5.68
2	1.27	11.36
3	0.0	17.05
4	0.0	22.73
5	0.0	28.41
6	0.0	34.09
7	0.0	39.77

Table 7: Result summary for the advertising triangle simulation. The simulation is performed for one through seven transmissions.

These results reveal an interesting factor. By comparing these results to the results for node B

in table 6, we can see that it is a correlation between the results. A single transmission in this simulation matches the results for two transmissions in the single link simulation. Similarly, the results for two transmissions in this simulation match four transmissions in the single link simulation. This situation shows us that the effective retransmit count to a node is equal to the number of links multiplied by the network's common transmission count. Here we assume that the linked nodes have access to all the unique messages in the DFU. This last assumption is probably highly optimistic, if not unreasonable, in the standard simulation setup. Nevertheless, this statement gives an implication of why the ADV bearer simulation for network β performed significantly worse than for the α network. By losing two of its links, node 12 has potentially lost close to $\frac{2}{3}$ of its possible attempts to receive any unique message successfully.

Let us consider the retransmission configuration for the network. One might think that this problem could be solved by introducing a dynamic retransmission count for the network. To compensate for nodes with a low number of available message paths, adjacent nodes may be configured to perform a higher number of retransmissions than the rest of the network, increasing the chance of successful transmission to an isolated node. However, the problem with such an approach is that it will cause a massive bottleneck. These bottlenecks will occur at nodes with a higher transmission count than the rest of the network. The time it will take for a node to convey a single message will be longer than the time it takes before a new one arrives, thus causing a buffer overflow at some point. This means that this issue can not be resolved by a dynamic configuration of the transmission count.

The previous section has shown that the ADV bearer simulations are highly topology-dependent. The likelihood of successful message delivery is highly affected by the number of available paths a message can take to reach a specific node.

4.6.2.2 GATT Simulation

Just like for the ADV bearer, I have also performed a simulation for network β with the GATT bearer solution, using the exact same parameters as for the simulation performed in section 4.6.1.5 for the α network. The result can be viewed in table 8.

Node ID	Number of links	Number of adjacent nodes	Single transmission loss chance (%)	TS last msg (ms)	Packets lost	Packet loss (%)	Peak buffer size
0	3	3	10.010	312279.6	0.0	0.0	0
1	3	3	10.097	312211.0	0.0	0.0	8
2	3	3	10.099	312301.4	0.0	0.0	8
3	3	3	10.099	312286.6	0.0	0.0	8
4	3	3	10.100	312208.4	0.0	0.0	8
5	3	3	10.098	312254.6	0.0	0.0	8
6	3	3	10.098	312346.0	0.0	0.0	8
7	3	3	10.099	312356.0	0.0	0.0	8
8	3	3	10.098	312414.0	0.0	0.0	8
9	3	3	10.099	312405.0	0.0	0.0	8
10	2	2	10.048	312424.2	0.0	0.0	8
11	3	3	10.097	312471.8	0.0	0.0	8
12	1	1	10.000	**312662.8**	0.0	0.0	0
13	2	2	10.050	312194.0	0.0	0.0	8
14	3	3	10.048	312590.0	0.0	0.0	8
15	3	3	10.098	312531.0	0.0	0.0	8
16	3	3	10.097	312495.6	0.0	0.0	8
17	3	3	10.099	312112.4	0.0	0.0	8
18	2	2	10.049	312565.2	0.0	0.0	8
19	2	2	10.050	312191.2	0.0	0.0	8

Table 8: GATT bearer simulation for network β with a buffer restriction of maximum eight entries per node.

Like with the ADV bearer, these results indicate that node 12 is the weakest link in this network topology because it is the node with the highest timestamp for the last reviewed message. However,

if we compare this execution time of 312 662.8 ms with the results from section 4.6.1.5, we can see that the execution time has only increased with 262.2 ms in this instance. For all practical purposes, the result is the same, consuming approximately 5.2 min to execute the entire DFU in both networks. This tendency reveals something interesting. While the number of links between nodes seems to be an essential factor for the performance of the ADV bearer, the GATT bearer seems to be more or less topology independent.

4.6.3 Topology Exploration for the GATT Bearer

The combined results from the α and β network show that the GATT solution does not seem to be highly dependent on the topology of the network when it comes to the transmission performance. This is most likely since the GATT bearer can guarantee message delivery.

As we can see from the simulations in section 4.6.1 and 4.6.2, the ADV bearer solution benefits from a topology where there are multiple possible paths from the origin to the destination. This is because the ADV bearer does not have any inherent way of confirming if a message transmission is successful. It is reliant on redundancy to ensure that the messages arrive safely. The redundancy in this instance is realized by two main factors, as discussed in section 4.6.2.1. The first is the number of transmissions performed for each unique message, while the other is the number of possible paths a message can take to reach its destination.

While multiple paths significantly improve the likelihood of message delivery in the ADV bearer, the GATT bearer does not seem to gain the same benefits. The GATT bearer does not need additional paths since it will ensure that every message is passed along before it is discarded. The GATT bearer solution might even be halted when used in a network topology that forms loops in the communication path because it stores all messages until they are successfully passed along. An example of this is if node A is trying to forward a message to node B. Node A struggles with this and cannot successfully transmit this message before its third attempt. In this instance, node B has already received the same message from node C. Ergo, when node A eventually will pass this message to node B, it will be discarded since the message is already present in the node B message cache. This means that the transmission from node A was utterly redundant. Still, node A has spent time and resources trying to forward this message. In the worst-case scenario, continuous failed attempts to forward this message to node B might even have caused the internal buffer of node A to fill up, making a stall for the flow of incoming messages to node A. Another matter that can be of concern is the number of GATT connections a node has to maintain. While the model does not account for this issue, each additional GATT connection a node has to maintain puts an increasingly higher strain on the node itself. By eliminating redundant links in the network, it might be possible to improve the performance of several nodes for the actual implementation.

With these considerations in mind, it might be reasonable to use a topology that does not provide multiple paths between nodes for the GATT bearer. Opposite to the ADV bearer, we can choose which links we want to utilize between nodes in this instance. With this in mind, I want to explore the possibility of altering the link configuration in the β network and assess the simulation outcome. The two alternative configurations I have decided to explore are a chain configuration and a tree configuration.

4.6.3.1 Chain Topology

The chain configuration can be seen in figure 17. We can observe that it is the same physical network as network β, providing the same number of possible links between each of the nodes. The main difference in this instance is that we are not utilizing every link possible. This topology is built by making a single chain of links from the origin node to the last node, which is node 12. In this instance, the possible number of links for a node spans from one to two links. The majority of the nodes have two links, while the two nodes located at each end of the chain have only one. In this way, we distribute the number of GATT connections each node has to maintain evenly.

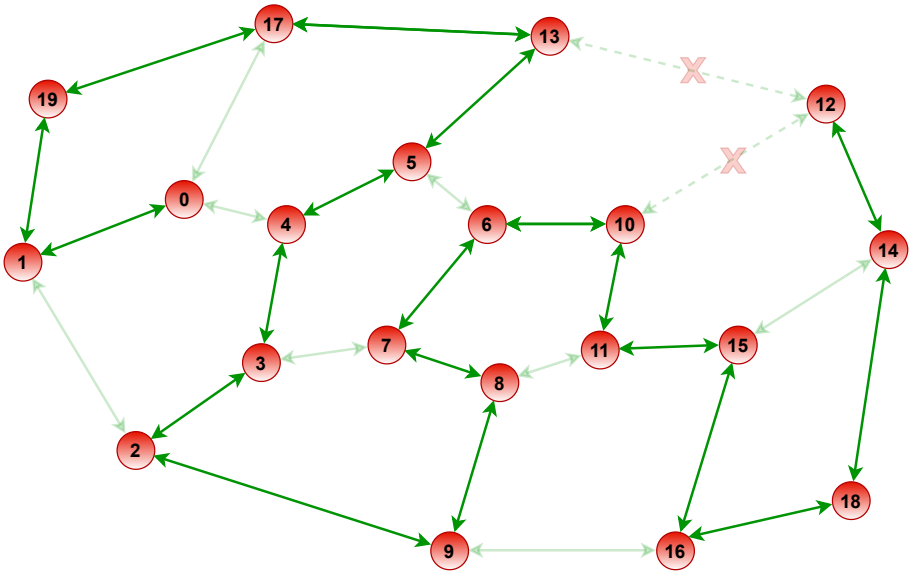


Figure 17: Overview of network β, using the GATT chain topology.

The simulation is performed with the same setup as before, running for 100 simulations with node 0 as the origin. The maximum buffer size of each node has been set to eight entries. The results from the simulation can be seen in table 9.

Node ID	Number of links	Number of adjacent nodes	Single transmission loss chance (%)	TS last msg (ms)	Packets lost	Packet loss (%)	Peak buffer size
0	1	3	10.000	313290.2	0.0	0.0	0
1	2	3	10.100	313290.2	0.0	0.0	8
2	2	3	10.068	314082.4	0.0	0.0	8
3	2	3	10.101	313978.2	0.0	0.0	8
4	2	3	10.073	313870.2	0.0	0.0	8
5	2	3	10.100	313753.6	0.0	0.0	8
6	2	3	10.069	314509.2	0.0	0.0	8
7	2	3	10.085	314400.6	0.0	0.0	8
8	2	3	10.100	314296.8	0.0	0.0	8
9	2	3	10.099	314200.6	0.0	0.0	8
10	2	2	10.050	314613.8	0.0	0.0	8
11	2	3	10.086	314703.8	0.0	0.0	8
12	1	1	10.000	**315115.4**	0.0	0.0	0
13	2	2	10.050	313645.0	0.0	0.0	8
14	2	3	10.043	315046.6	0.0	0.0	8
15	2	3	10.099	314796.6	0.0	0.0	8
16	2	3	10.071	314889.2	0.0	0.0	8
17	2	3	10.090	313525.2	0.0	0.0	8
18	2	2	10.049	314972.6	0.0	0.0	8
19	2	2	10.050	313408.4	0.0	0.0	8

Table 9: GATT bearer simulation for network β, using the chain topology. Buffer restriction of maximum eight entries per node.

Reviewing these results, we can see that the execution time of the DFU is within the same scope as the previous GATT bearer results for the β network. With an execution time of 315 115.4 ms it exceeds the previous result by a magnitude of 2452.6 ms, which amounts to an increase of *0.8%* in the total execution time.

4.6.3.2 Tree Topology

The tree configuration can be seen in figure 18. This topology is built by making two branches from origin node 0 and expanding these throughout the network. A general rule that has been used in

this instance is that each node on a branch may only create two new branches when extending the network. This is done to create a network where the number of links for every single node is as balanced as possible, making the possible number of links for a node span from one to three links. Similar to the chain topology, the GATT connection each node has to maintain is distributed fairly even while still keeping some flexibility to arrange the network topology.

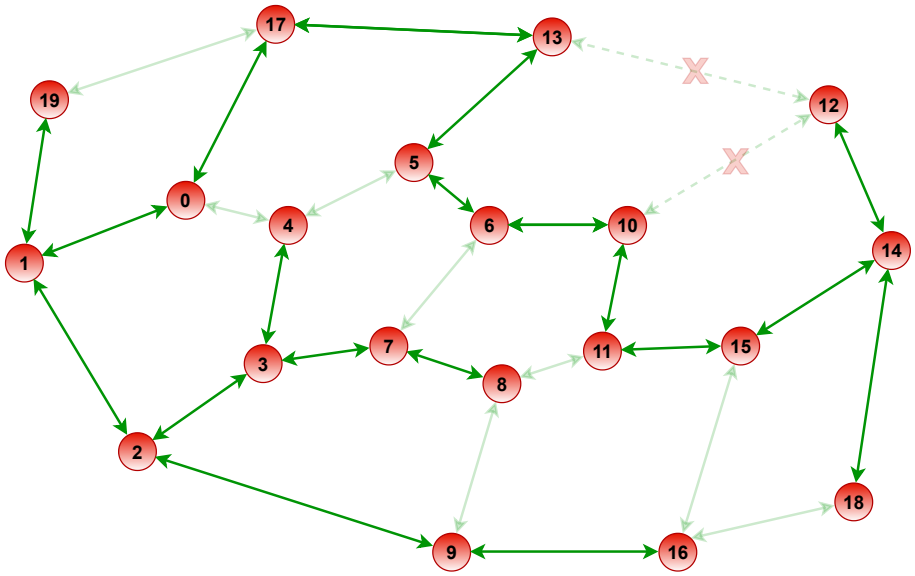


Figure 18: Overview of network β, using the GATT tree topology.

The simulation is performed with the same setup as before, running for 100 simulations with node 0 as the origin. The maximum buffer size of each node has been set to eight entries. The results from the simulation can be seen in table 10.

Node ID	Number of links	Number of adjacent nodes	Single transmission loss chance (%)	TS last msg (ms)	Packets lost	Packet loss (%)	Peak buffer size
0	2	3	10.000	313388.0	0.0	0.0	0
1	3	3	10.051	313270.8	0.0	0.0	8
2	3	3	10.099	313374.8	0.0	0.0	8
3	3	3	10.049	313466.4	0.0	0.0	8
4	1	3	10.064	313537.2	0.0	0.0	0
5	2	3	10.050	312883.2	0.0	0.0	8
6	2	3	10.094	313004.4	0.0	0.0	8
7	2	3	10.026	313538.8	0.0	0.0	8
8	1	3	10.074	**313603.4**	0.0	0.0	0
9	2	3	10.000	313445.6	0.0	0.0	8
10	2	2	10.050	313103.8	0.0	0.0	8
11	2	3	10.050	313191.8	0.0	0.0	8
12	1	1	10.000	313433.6	0.0	0.0	0
13	2	2	10.050	312763.2	0.0	0.0	8
14	3	3	10.000	313364.4	0.0	0.0	8
15	2	3	10.050	313276.8	0.0	0.0	8
16	1	3	10.033	313504.8	0.0	0.0	0
17	2	3	10.050	312655.0	0.0	0.0	8
18	1	2	10.000	313432.8	0.0	0.0	0
19	1	2	10.022	313344.0	0.0	0.0	0

Table 10: GATT bearer simulation for network β, using the tree topology. Buffer restriction of maximum eight entries per node.

Reviewing these results, we can see that the execution time of the DFU is within the same scope as the previous GATT bearer results for the β network. With an execution time of 313 603.4 ms it exceeds the previous result by a magnitude of 940.6 ms, which amounts to an increase of *0.3%* in the total execution time.

4.6.3.3 Assessing the Alternative Topologies

By reviewing the outcome of the simulations of both the chain- and tree topology, we can see that they both provide a DFU execution time similar to the initial result for network β. The chain topology gave a simulation result that increased the total execution time by approximately *0.8%*, while the tree configuration increased the total execution time by only *0.3%*. By comparing the results for the two topologies, it is likely that the results would have been more or less alike if the origin node of the chain topology had been located somewhere in the middle of the chain. This would have made the maximum amount of hops in the network effectively half, presumably making the latency impact smaller.

Still, the tree topology has some benefits over the chain solution. As we just saw, the overall latency will most likely be larger in the chain configuration. This is because the number of hops in the longest path in the network is equal to $Node_{Cnt} - 1$. Another issue with the chain topology is that it might prove difficult to realize it for an arbitrary network. Suppose the number of nodes with only one available link to the rest of the network exceeds two. In that case, it will prove impossible to realize the chain topology since the number of nodes that may have only one link is restricted to two for this topology. If there are more than two nodes that only have one link to the network, the best possible outcome will result in a variant of the tree topology. In a choice between these two alternative topologies, the conclusion is that the tree topology seems to be the preferable one.

As mentioned earlier, the DFU simulations for the tree topology gave an execution time that resulted in a *0.3%* increase compared to the previous result for network β. For all practical purposes, we may consider these results as equivalent. Considering this and the issues described in the introduction to this section, I believe that the tree topology might be worth exploring for actual implementation.

4.6.4 Simulation Summary

The general tendency observed throughout these simulations is that the GATT bearer on a general basis outperforms the ADV bearer concerning both data consistency and throughput. This difference becomes substantial for networks where we introduce isolated nodes, making the required transmission count for the ADV bearer large to ensure that we do not lose messages. This phenomenon shows that the ADV bearer performance is very dependent on the topology of the network. In comparison, the impact on the GATT bearer's performance is negligible, regardless of the number of available message paths to each node. This finding is quite interesting since it indicates that it is possible to boost the network performance without maintaining numerous GATT connections on a single device. At this stage, it is unclear how many connections a device in actual implementation can maintain simultaneously. The topology exploration of section 4.6.3 showed that it might be possible to create larger GATT networks using only two or three GATT connections per device. Nevertheless, the simulation results indicate that the GATT solution is likely to enhance both the throughput and data consistency performance of the BTM network, thus legitimizing a transition into the implementation phase for the GATT approach.

5 Development

The results from the simulations in section 4 showed that a GATT communication-based BTM network might enhance both the throughput and data consistency performance of BTM. The results also indicated that a GATT solution would not be as topology-dependent as the ADV bearer solution concerning the mentioned performance metrics. Based on these results, I have decided to move forward with the GATT-based approach, thus starting the implementation work.

The realization of the GATT bearer solution requires several module alterations and implementations. First, I need to establish a functioning GATT bearer that can support multiple connections to other devices. Further, I must implement a configuration tool for the establishment and maintenance of the GATT relay network. This tool must be able to initiate connections between the devices, disable the ADV relay functionality, and perform other utility functions for the network.

5.1 Development Environment

To implement the GATT bearer solution, I have chosen to utilize the nRF Connect Software development kit (NCS). NCS is distributed and supported by Nordic Semiconductor ASA and supports many different features and functionality targeted for Nordic hardware. This software development kit (SDK) consists of a gathering of Git repositories with application code, drivers, libraries, and forks of open source code[2]. The backbone of NCS is the Zephyr project. Zephyr is a scalable RTOS that is optimized for devices with constrained resources. It supports multiple hardware architectures, among them the ARM Cortex M-series, which is the architecture of many Nordic devices. Zephyr provides extensive kernel services like multi-threading, interrupt services, memory allocation services, and power management. It also provides a long list of other features like Multiple Scheduling Algorithms, Memory Protection and Bluetooth support[12].

There are several reasons why I have chosen to use NCS for the implementation. One crucial factor is that it supports Nordic hardware, which is the preferred hardware considering that this thesis is written in collaboration with Nordic Semiconductor. It is hardware that I currently have access to in sufficiently large quantities, a consideration that is important since realistic testing of the BTM implementation will require several devices to form a BTM network. Another crucial factor is that Zephyr provides a functioning implementation of the BTM stack, which will provide the framework for my software development. A final consideration for this choice of development environment is that it is an SDK that I am already familiar with. I have utilized NCS for several tasks and projects in the past. Through this work, I have acquired knowledge of the NCS features on a general level and familiarized myself with the BTM stack provided by Zephyr. I deem that this factor will contribute to a faster development pace for the implementation.

5.2 Proxy Client Module

The proxy client functionality of BTM is already a well-established feature, implemented for a multitude of devices on the market. However, it is not currently supported by the BTM module that Zephyr provides. The consequence of using the Zephyr project as the code base for this thesis is that I will have to implement the proxy client feature myself.

Most of the work on this feature is well established and defined by the BTM profile specification[10]. With this in mind, I will strive to keep the implementation description brief for a large portion of this implementation and instead focus on the specific implementation details for the work on this thesis.

5.2.1 Beacon Handling

For the proxy client to initiate a GATT connection to a proxy server, it must be able to receive and handle the advertising beacons issued by the server. As described in section 2.2.3.2, the mesh proxy service defines two beacon types for advertising the presence of the service; network beacons and node ID beacons. Both of these beacons are connectable undirected advertising events, consisting of the flags, service UUID list, and service data. The initial procedure for handling both of these beacon types is common. Whenever a connectable undirected advertising packet arrives at the scanner, the handler starts by parsing through the flags field, checking to see if it is of the expected length.

Provided that this is in order, the handler moves on to parse the content of the service UUID list. Here it checks if the UUID list contains the **Mesh Proxy Service** UUID. If this is the case, the handler continues to the service data field. First, it conducts a check if the length of the service data is compatible with the expected length of either beacon type before checking if the first two bytes of data contains the **Mesh Proxy Service** UUID once more. At this point, the handler checks if the beacon is of the network or node ID type by evaluating the ID type contained in the service data.

5.2.1.1 Network Beacon Handling

If it is a network beacon, the rest of the service data will contain the network ID the proxy server is advertising for. By setting a network ID that it wishes to connect to, the proxy client may compare the incoming network ID to this desired network ID.

The implementation of this feature consists of a function call in proxy client API that can set the network ID the user wishes to connect to. As soon as this function is called, the handler for incoming network beacons will use this subnet to check if an incoming network beacon matches the subnet it wants to connect to. If the two IDs match, the advertising message may be used to initiate a connection to the advertising proxy server.

5.2.1.2 Node ID Beacon Handling

If it is a node ID beacon, the rest of the service data will contain an eight-byte hash and an eight-byte random value. Compared to the network beacons, the parameters following the node ID beacon require a greater extent of handling before they can be utilized for connection purposes. When the advertising proxy server creates a node ID beacon packet, it first creates the eight-byte random value. A buffer is then filled with six bytes of zero padding, the random value, and the root source address of the proxy server device. This buffer is then sent through a crypto block, using an identity key associated with the subnet the proxy server is advertising for. The output from this procedure is the eight-byte hash that will follow the packet. The hash is put inside the packet along with the random value and the beacon type.

For the proxy client to recognize the source address in the encrypted hash, it must perform the same procedure as the server did when creating the packet before comparing the resulting hash to the hash that followed the beacon packet. To do this, it must have a source address, identity key, and the random value used in the encryption. The incoming packet already provides the random value, but the source address and identity key are parameters the proxy client must provide. These two parameters form the identification context that can be used to precisely decide which node and subnet the proxy client wants to connect to. The implementation of this feature consists of a function call in proxy client API that can set this context. As soon as this function is called, the handler for incoming node ID beacons will use the source address and subnet to check if an incoming node ID beacon matches the device and subnet that it wants to connect to. When the two hash values match, the proxy client knows that this beacon comes from the desired device, meaning that it can use this advertising message to initiate a connection to the proxy server in

question.

5.2.2 Connection Establishment and Discovery

All operations regarding proxy connections in the proxy client module utilize an array of server connection entries as a backbone. Each of these entries represents a single connection between the client and a corresponding server and contains all information and metadata required to establish, configure, and maintain a connection. The available number of entries in this array is configurable by the user at code deployment. The number of connections a proxy client module may establish is restricted by the available number of entries confined in this entry array.

When initiating a new connection, the proxy client will try to acquire an unused server connection entry. Succeeding in this, it will continue by inserting some metadata for the connection. This information is provided by the advertising beacon that was used to initiate the connection. It consists of the subnet index that the connection is associated with, along with the root element address if the advertising beacon was of the node ID type. At this point, the module will create the connection to the proxy server. When the connection is established, a connection callback will trigger inside the module. From here, the module will start by handshaking the Attribute Protocol Maximum Transmission Unit (ATT_MTU). This is an agreement between the client and the server for how large a single transmission unit can[10, p. 279]. Following this, the module will start the GATT discovery procedure, starting by performing primary discovery of the Mesh proxy service on the target proxy server. It will then perform discovery of the Mesh Proxy Data In characteristic, storing the characteristic's attribute value handle inside the associated server entry in the module. This handle will enable the proxy client to forward BTM messages to the target proxy server device. In the final step of the discovery procedure, the module will perform discovery on the Mesh Proxy Data Out characteristic of the target proxy server and the CCCD that follows this characteristic. By using the attribute information generated from this discovery, the module will subscribe to this characteristic and enable notifications on the proxy server using the CCCD. The subscription parameters contain a pointer that points to a notification callback inside the module. From this callback, the proxy server will receive any BTM message sent by the target proxy server.

At this point, we encounter an implementation detail that is specific to this thesis. As described in section 2.2.3.1, a proxy server will by default initialize a newly created connection with an empty white list filter. This filter will effectively block all BTM messages that the target proxy server attempts to pass to the client. In a scenario where the proxy client is meant to function as a stand-alone device that needs access to a subset of the messages passed in the BTM network, the white list filter might be of great value. In this instance, however, the proxy clients will act as a vital part of the BTM network's infrastructure, meaning that they need to receive all exchanged information. On account of these considerations, I have utilized the filter type set message described in section 5.2.3.2 to immediately change the filter type of the proxy server to an empty blacklist. This allows all messages to pass from the proxy server to the client.

At this point, the connection sequence is completed. A flag inside the server connection entry is raised, indicating that the connection is ready for normal operation.

5.2.3 Interfacing With the Proxy Client Module

5.2.3.1 Connection Control and Observability

The modules API provides several functions and callbacks to enable control and observability of the initiation and maintenance of the GATT connections in the proxy client module.

In order to enable the module to scan and parse incoming advertising beacons, the API contains an *Advertising Beacon Process* function. This function is called from within the scanning callback of the advertising module, passing all relevant messages to be handled inside the proxy client

module. As described in section 5.2.1, the module provides two functions for discovering and initiating a connection to a proxy server:

- Node ID Connect — Takes a unicast address and a subnet as input. Succeeding this function call the proxy client module will start scanning for node ID beacons containing these parameters. Receiving a matching beacon will initiate a connection to the target proxy server.
- Network ID Connect — Takes a subnet as input. Succeeding this function call, the proxy client module will start scanning for network ID beacons containing the matching subnet. Receiving such a beacon will initiate a connection to any proxy server advertising for the target subnet.

The module provides three callbacks that enable the user to monitor connections to the proxy servers:

- Connection Established CB — This callback will trigger each time a new connection establishment procedure has been completed between the proxy client module and a target server. The parameters following this callback contain the context associated with the connection, the subnet the connection applies to, the unicast address of the root element on the target server device[1], and an error code that indicates if the connection was successful.
- Disconnection Occurred CB — This callback triggers each time a disconnect event occurs between the proxy client module and a server. The parameters that follow this callback are identical to the connection established callback.
- Configuration Completed CB — This callback triggers whenever a discovery and configuration procedure is completed for a GATT proxy connection. The parameters that follow this callback is identical to the connection established callback, except that it does not contain an error code.

The user may set these callbacks by calling the *Connection Callback Set* function provided by the modules API.

5.2.3.2 Message Transmission

The implementation for handling incoming and outgoing messages over a GATT proxy connection is largely a mirrored implementation of the existing proxy server module that Zephyr provides.

The three message types that the proxy client may issue are regular network PDUs, BTM beacons, and proxy configuration messages. In the case of the network PDUs and BTM beacons, the proxy client module provides a single API function call that is used to propagate these message types inside the module:

- Proxy Relay — Like the established proxy server module, this function takes a BTM network PDU as input. It transmits it to all proxy servers that the proxy client module is currently connected to. This API function is called from inside the network layer and beacon module of the existing BTM implementation, ensuring that all message traffic passed by the upper layers is passed down to the proxy client module.

At this point, the messages are handled by the module. Here it will find the proper server connection(s) the respective message shall be sent on before passing it to a segmentation handler. Suppose the outgoing message is larger than the brokered ATT_MTU of the GATT connection. In

[1] In the case where the connection has been established using a network ID beacon, the unicast address following this connection will be set to the unassigned unicast address [10, p. 39]

that case, this handler must divide the original message into two or more segments to allow the message to be transmitted. The receiving server will reassemble these segments as soon as they all have arrived at the destination. The transmission of proxy configuration messages is handled similarly to the other message types but requires additional preliminary handling before being transmitted. These details are described further in this section.

Any incoming message will arrive as a notification from the Mesh Proxy Data Out characteristic of a connected proxy server. These messages will be propagated to the associated callback that was described in section 5.2.2. From here, an incoming message will be passed to a reassembly handler. Suppose the incoming message is a segment of a message larger than the brokered ATT_MTU of the GATT connection. In that case, this handler will ensure that all partial segments are collected and reassembled before passing the message on for further handling. After this stage is completed, the module will parse the message type parameter contained in the message. These types consist of regular network PDUs, BTM beacons, provisioning PDUs, and proxy configuration messages. The three former types are all handled by existing modules of the Zephyr BTM implementation and can be passed along using their respective API's:

- Network Receive — Receive handler of the network layer. Here, the network module handles incoming network PDUs, processing them before passing them to the upper layers and potentially relaying them.
- Beacon Receive — Receive handler of the BTM beacon module. Here incoming BTM beacons are processed.
- Provisioning GATT Bearer Receive — Receive handler for incoming provisioning beacons.

In addition to the three formerly mentioned message types, the proxy client module must also be able to forward and process a set of proxy configuration messages, in accordance with the BTM profile specification[10, p. 264]. The four messages it must be able to handle are:

- Set Filter Type — Sent from client to server. Set the filter type of the server. Either white list or blacklist
- Add Addresses To Filter — Sent from client to server. Add addresses to the servers filter list.
- Remove Addresses From Filter — Sent from client to server. Remove addresses from the servers filter list.
- Filter Status — Sent from server to client. Status response to the client for the three other message types.

The three former configuration messages in this list follow a common pattern when issued. Initially, they are packed with their respective parameters, as described in the BTM profile specification. Then they will receive a transmission context that corresponds to the proxy server targeted. By using this context, the messages will undergo encryption before sent to the same segmentation procedure as any other message that the proxy client passes.

Whenever an incoming status message arrives from a connected proxy server, it will initially be handled in the same way as any other message. When the reassembly procedure is completed, the handler will parse the message type and send it for internal processing in the module. Here the message will undergo decryption and a replay attack check. If both of these procedures succeed, the status message may be sent to a user-defined callback for further processing.

5.3 GATT Proxy Configuration Model

The proxy client and server modules combined provides the base means to establish GATT connections between devices. However, in section 3.3.3 I discovered that the BTM specification does not

provide any inherent feature enabling remote control of the establishment of GATT connections. This thesis's implementation depends on a tool that provides efficient control and observability of GATT connections on a network level.

With this in mind, I have decided to implement the configuration tool for managing the GATT relay network as a vendor model, henceforth referred to as the GATT Proxy Relay Configuration (GPC) model. The GPC model defines two roles; a client model and a server model. The client model provides an interface of mesh messages that enables the control and observability of every single GPC server model in the network.

The GPC server model acts as the backbone for interfacing with the proxy server and client module on each device to control the GATT connections between devices. This model will, in this instance, be regarded as a foundation model, mandatory to each BTM device along with the Configuration and Health server models (section 2.2.2.2). The GPC server will be instantiated in the root element of every participating device in the BTM network. In addition to providing control over the proxy features, the server model also offers functionality to map the connection topology between different devices in the network. This feature is crucial to obtain the necessary information required to choose which devices should act as relay nodes in the network. Lastly, the server enables control over the ADV bearer, making it possible to alter its behavior.

At the initial creation of a network, the participating devices will be provisioned and configured as any regular mesh network. At this point, there are no active GATT connections between any of the mesh devices, and the network communication is fully dependent on the ADV bearer. To prime the GPC models for the following GATT relay transformation, a set of mandatory configuration steps needs to be executed. The following steps apply to all GPC servers, as well as the GPC client that will be used for the GATT relay configuration:

- All models must be bound to a common application key, implicitly meaning that they must share the same subnet.
- All models must be set to publish and subscribe to a common group address.

After these configuration steps are completed, the network is ready to initiate the GPC process. This process starts by conducting mapping of the connections between devices in the network. After the mapping is completed, the installer will utilize this information to decide which nodes should act as GATT relay nodes in the network and their connections. Using the GPC client, the installer will issue command messages to initiate the connections between the chosen relay nodes. After this is completed, the network is in its operational state and will start communicating using GATT relaying.

5.3.1 Mapping Functionality

The model must provide information about the on-air topology between all devices in the network at the initial establishment. This information is crucial for the installer to make a satisfying configuration of relaying devices for the network. The following section describes how the connection mapping is implemented in the GPC models.

There are two different metrics available in order to assess the quality of a link. Contained within the metadata of each BTM message, we have the Received Signal Strength Indication (RSSI) parameter. The RSSI is the power level a message was received with on the radio of the device. One way to measure the link quality can be executed by evaluating the RSSI value of several received messages from a single device. The second metric that can be used to evaluate link quality is message loss. Suppose we can control an environment where we know how many messages were sent from the origin device. In that case, we can compare the amount of received messages against the number of expected messages on the surrounding devices.

Between the two approaches for measuring link quality, I have decided to use message loss as

the sole metric. The background for this decision is based on several aspects. The first is that the complexity of such an implementation is lower than an implementation using RSSI. When measuring the quality based on message loss, the message can either be received or not received. In comparison, while measuring based on RSSI we also have to consider how well a message has been received while still considering any lost messages. In reality, we are primarily interested in knowing if a message will arrive successfully or not, while the RSSI in itself is not of particular interest as long as the message gets to its destination. I admit that an implementation based on both message loss and RSSI most likely would provide a solution with a more fine-tuned output. Still, I deem such a solution somewhat redundant for this particular use case. Another reason why I have decided to discard RSSI as a metric is that a solution based solely on message loss can be implemented by using functionality provided by the configuration foundation models of BTM. This is discussed further in section 7.2.2 of this thesis.

The general idea of a pure message count implementation is to make every participating device of the network transmit a known number of messages. The same devices must also listen for incoming messages from neighboring devices and keep a log over the messages it receives. All devices must be able to perform these tasks simultaneously to avoid making the procedure unnecessarily time-consuming. When all devices have transmitted their messages, the received message log of every device must be collected by a client and assessed to produce the link topology of the network. A crucial factor for the solution is that we must guarantee that each message is received directly from the origin device of the message. To explain the meaning of this, we must revisit some concepts explained in section 2.2.1.2. In a regular BTM network, a message can arrive at its destination through multiple paths, depending on the configuration of the number of relaying nodes. Even if a message is destined for a device, it might arrive via another relaying device under certain circumstances. If this is not handled correctly, we might end up in a situation where we get a false positive for a received message. A situation can occur where the original attempt on the link in question failed but was successfully transmitted via a relay node. The solution to solve this issue is to utilize the TTL parameter of the message we are sending. Setting the TTL value of an outgoing message to zero will immediately be disqualified for retransmission by any relay node that receives the message. In this way, we can guarantee that a message received by a node originates directly from the radio of the origin.

The core component for the transmission of link messages from the server model is implemented as a timer-induced callback associated with the server model that is instantiated on a device. From now on, this will be referred to as the mapping timer. A mapping procedure is initiated when the server receives an initiation message from a client model. Upon receiving this message, the server will activate the mapping timer and send a status response to the client. The initiation message contains a parameter that defines how many messages the server shall emit during the transmission sequence. This value is stored in the server context. When the callback triggers, a new link message is transmitted from the server, containing the primary element address of the server device. The message count is then decremented by one, and the mapping timer is set to trigger a new callback after 200 ms. I have chosen to emit the link messages at a delayed interval to prevent the effects of internal noise in the system. Since several servers emit link messages simultaneously, it is wise to distribute the total number of messages over a larger period of time. During this transmission sequence, neighboring servers will be listening for incoming link messages. The messages these devices receive are stored in an entry buffer on the server. When the message count parameter of the server reaches zero, the transmission procedure is at its end for this device. At this point, the server transmits a status message telling the client that the procedure is finished. At the end of a mapping procedure, the client will collect the data from each server by issuing a separate fetch message. The servers will respond to this message by returning the data it has collected during the mapping procedure.

In order to realize the desired behavior for the mapping between devices, the following model message types and behaviors have been implemented:

- Link Update Initialize — This message type is sent by the client model and received and handled by the server model. A message contains a single parameter that corresponds to the number of link update messages that the server will emit during the mapping procedure.

Upon receiving a link update initialize message, a server shall clear the content of the link entry buffer and store the incoming message count parameter to the server model context. Following this, the server shall activate the mapping timer with a delay of 1 s. This delay is to ensure that all server devices are primed before the link mapping commence. Lastly, the server shall issue a status response message to the client containing a successful status opcode.

- Link Update — This message type is sent, received, and handled by the server model. A message has a single parameter containing the root element address of the origin device of the message. When a server transmits a link update message, the destination address shall be a group address that all GPC server models commonly share in the BTM network[2]. Furthermore, the message's TTL value shall be set to zero to prevent messages from being relayed in the network. When receiving a link update message, a server shall extract the primary address contained within the message. It then checks if this address corresponds to the primary address of its own device. If it does, the message has arrived through the internal Mesh interface of the device, meaning that the message received was transmitted by this server instance itself. When this is the case, the message is immediately discarded, as it has no value for the link assessment. Otherwise, the message is passed to the server's link entry buffer. If the address is not already present in the buffer, a new entry is created, using the address itself as a key. A message count parameter associated with this entry is then incremented by one, corresponding to one successfully received message. If there already exists an entry for this particular address, the buffer fetches this entry and increments the message count by one.
- Link Data Fetch — This message type is sent by the client model and received and handled by the server model. This message type contains no parameters. Upon receiving a link data fetch message, a server shall immediately respond by sending a link data message back to the client.
- Link Data — This message type is sent by the server model and received and handled by the client model. This message type contains the root element address of the sending server device, along with the entire link entry buffer of the sending server. The number of entries in this message depends on the number of valid entries contained in the link buffer, making the length of this message type variable. A receiving client will pass the content of this message to the application level, where it may be collected for further mapping computation.

Figure 20 shows the intended interaction between different devices during a mapping procedure. In this example, we have three participating devices; the client (green) and servers 1 and 2 (red). Here we assume that the servers are within radio proximity of each other. All participants are configured to share a common group address that they are publishing and subscribing to. The sequence is initiated when the client issues a link update initialize message to both of the servers. This message contains a count of N messages that each server shall issue during the mapping procedure. When the servers receive this message, they will start their mapping timers and issue a status response back to the client.

Suppose we assume that the application level on the client device can keep track of how many devices it should expect responses from. In that case, it will now enter a state where it is listening for incoming update complete status messages from the servers. Simultaneously, the servers will begin to issue link update messages periodically. Each server is simultaneously scanning for the issued messages from their server counterpart. Each message that arrives successfully will be collected by the server and stored in the entry buffer. When the servers have issued the required number of link update messages, they will stop the mapping timer and issue an update complete status message to the client. When the client receives the expected number of status messages, it will start collecting the mapping data from the servers by issuing link data fetch messages to each of the servers. The servers will respond to this request by transmitting their collected data in a link data message. At this point, the client device possesses all the data that was generated during the mapping procedure. This data can now be further processed to obtain an overview of the link topology in the network.

[2]This criterion needs to be configured at the initial network set up by the installer.

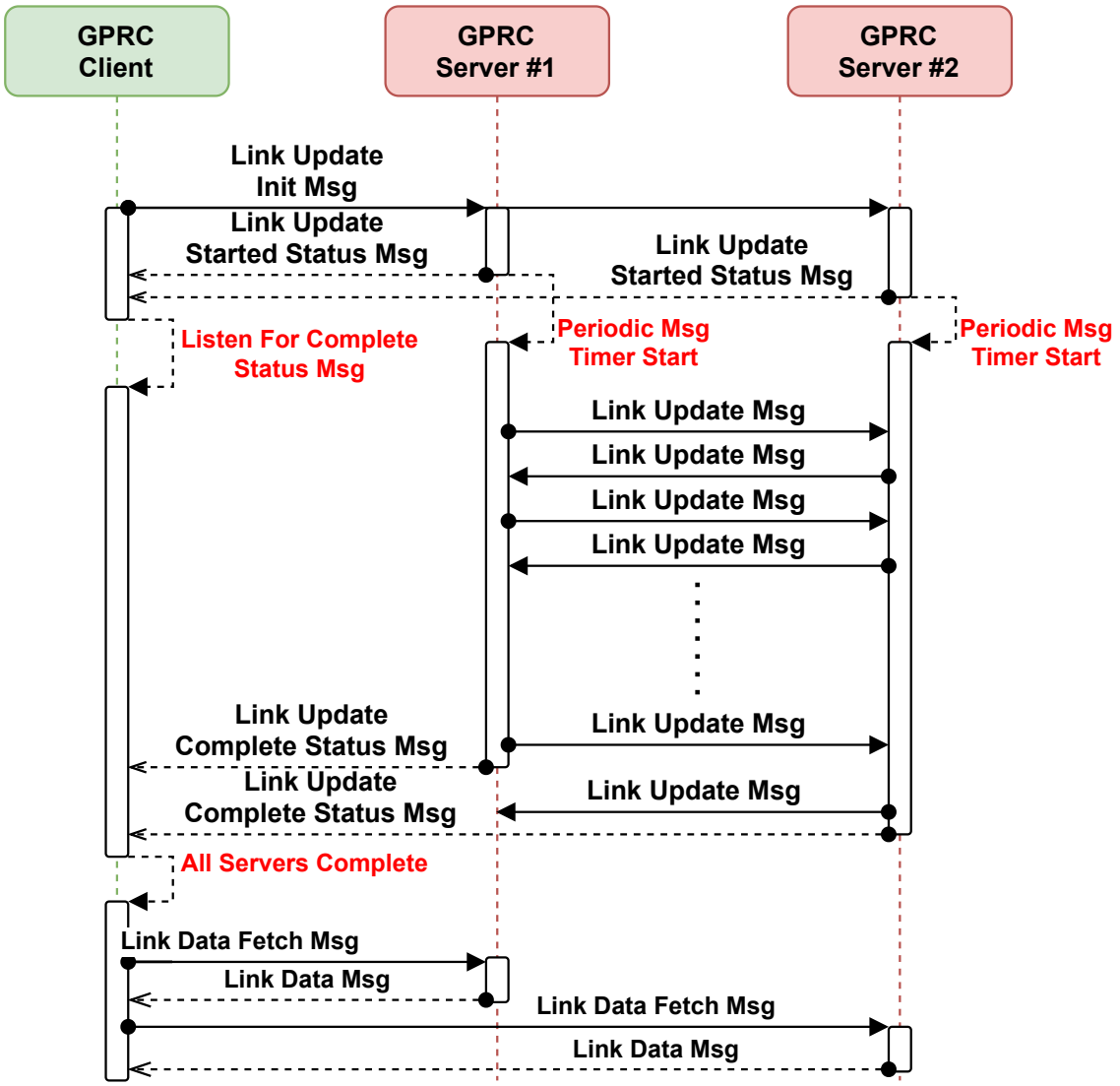


Figure 19: Mapping procedure between two GPC server models.

5.3.2 Connection Establishment and Maintenance

The ability to precisely decide which relay nodes should connect and which should have the client and server role is crucial to the implementation. Additionally, when the network is in an operational state, the model must be self-preserving, meaning that it must gracefully handle certain occasional events. These are events like unexpected disconnections, device reboot, and connection loss to neighboring devices. The following section describes how the connection between devices is initiated and maintained by the GPC models.

According to the BTM profile specification, as described in section 2.2.3.2, there are two ways of advertising the presence of a proxy server. The first one is advertising using network IDs. This advertising message contains no information about the device that emitted the message, making it unsuited for this particular implementation. The other choice is advertising using network IDs. This message type contains the unicast address of the root element of the source device, which satisfies the requirements for the implementation. Considering this, I have decided to utilize node ID advertisement for this implementation. The node ID advertisement is not active by default in a BTM device in difference to the network ID type. The legacy way the node ID advertisement is used is that it is activated for 60 s right after provisioning the device. Apart from this, the advertisement can be activated by using the foundation configuration models. Since I have chosen to contain all

functionality within the same model, the implementation must duplicate the functionality under other circumstances handled by the configuration model to activate the node ID advertisement at will.

When a proxy server has been set to advertise with node ID beacons, each beacon will contain the root element address and the network index of the subnet that it is advertising for. For a proxy client to initiate a connection to the server, we need to pass this information context to it. The proxy client module API provides this feature. When this is done, the client will start to listen for incoming node ID beacons, comparing the address and network index to the context that we have passed to it. As soon as the client receives a beacon with the correct content, the GATT connection is initiated and established. After mapping the network topology, the proxy client of each relay node will be given a fixed set of servers that they should connect to. After this initial setup, it should not be necessary to alter the connection configuration, given that the chosen relay node topology is satisfactory. This means that a device should keep the connection configuration in persistent memory, thus recovering this information after losing power or reboot. Based on this information, it should then be able to re-establish any connection it had before the reset.

I have decided to implement this as a connection entry list that follows the GPC server model. Each entry contains the primary address of the proxy server that we want to connect to, along with the subnet that follows. Any time any of these parameters changes for a given entry, the entry is saved in persistent storage. Each entry also contains a flag that tells if the connection is currently active, along with a pointer to the BLE connection context associated with the entry. The model uses the flag to determine if it needs to re-establish the connection, while the pointer is used whenever there is a need to terminate the connection from the model level. An example of when this could be necessary is when we need to delete the connection entry list. On boot up, the model will retrieve the connection entries from flash memory and initialize the entry list. The connection flag is set to false, and the connection pointer is set to NULL. Upon a connection event, the connection flag of the entry will be raised, and the context pointer is set to the BLE connection context. If a disconnect event occurs, the entry flag will be lowered again and the pointer set to NULL.

Along with the connection entry list, I have implemented a connection link handler. When called, this handler will process the connection entry list and evaluate if there are connections that need to be initiated. For each iteration, the handler will attempt initiation of connection for a single entry before terminating, given that there are unconnected entries in the connection list. The connection link handler is coupled with the timer-induced callback. After completing an iteration, the handler will initiate a recursive delayed call as long as there are unconnected entries in the connection list. Unless a connection event cancels this timer, the handler will run another iteration 5 s after the last iteration is completed. If the connection entry list is empty or all entries are connected, the handler will stop until an event eventually activates it again.

Five events may activate the connection link handler. All these events ensure that the connection link handler always will be active as long as there are unconnected entries in the list:

- Device boot-up — The GPC server model has been started. Any connection entry stored before shutdown will be retrieved from flash and placed in the connection entry list. The handler is called to re-establish these previous connections.
- An incoming new connection entry — A new entry has arrived from a GPC client model. This connection should be established immediately.
- A connection has been completed — The connection sequence for another entry has been completed successfully. The handler will check if other entries are still not connected. This event is provided by the configuration complete callback sent by the proxy client module.
- A connection has been terminated — A previously established connection has been terminated. The handler should try to re-establish this connection. This event is provided by the disconnect callback sent by the proxy client module.

- The preceding connection attempt has timed out. The GPC server has not received any connection event within the 5 s time limit, meaning that the handler should retry the connection attempt. The connection event is provided as a callback from the proxy client module.

Certain real-world considerations have been taken into account when creating the connection link handler. In an actual network, a device may suddenly "disappear" out of view from its neighbors. This can occur for many different reasons, but the important thing is that a GPC server trying to maintain a connection to this device has no way of knowing if and when the device will reappear. The time it takes for the target device to reappear can be in the range of milliseconds, hours, or days, depending on the actual reason for the disappearance in the first place. The device might never reappear, for instance, if it has malfunctioned beyond recovery and needs to be replaced by another device. Let us imagine a situation where we have two or more entries that need to be handled on a GPC server. Suppose a target device "disappears" for good. In that case, the GPC server model will periodically initiate futile reconnection attempts to this device as long as the connection entry stays in the connection list. This could potentially be harmful to the rest of the system if it is not handled carefully. If a device "disappears" for good, the reconnection attempts to this device must not block the reconnection attempt for the other devices present in the connection entry list. In order to prevent starvation between the connection entries, the connection link handler is implemented so that it will always start by checking the entry that follows the entry it previously tried to connect to, treating the connection entry list as a ring. This means that for every futile attempt of connection to an absent device, the server model will check and reattempt connection for all other entries once before initiating the next futile attempt.

In order to realize the desired behavior for connection between devices, the following model message types and behaviors have been implemented:

- Connection Entry Add — This message type is sent by the client model and received and handled by the server model. A message contains two parameters; the root element address of the target proxy server and the subnet index that we wish to connect to. Upon receiving a connection entry add message, the server shall check that the entry is not already present and that the connection list is not full. If either of these is the case, the message should be discarded, and the server shall issue a status response message to the client containing an appropriate error status opcode. Otherwise, the message parameters shall be placed inside a vacant slot of the server's connection entry list. The updated connection list shall then be stored within the server model's persistent storage. Finally, the handler shall initiate the connection link handler before issuing a status response message to the client containing a successful status opcode.
- Node ID Adv Set — This message type may be sent by either a client or server model and is received and handled by the server model. A message contains two parameters; a boolean that indicates if we wish to turn the node ID advertising on or off and the subnet index that we are targeting. Upon receiving a node ID adv set message, the server shall either turn on or off the node ID advertisement for the targeted subnet, depending on the content of the boolean parameter contained in the message. This is an unsolicited message.
- Connection List Reset — This message type is sent by the client model and received and handled by the server model. This message type contains no parameters. Upon receiving a connection list reset message, the server shall execute a disconnect procedure for all active BLE connections present in the server's connection list. Following this, the server shall wipe every single entry in this list and update the flash memory. Lastly, the server shall issue a status response message to the client containing a successful status opcode.

Figure 20 shows the intended interaction between different devices and modules during a connection addition on a relaying device. In this example, we have three participating devices; the GPC client (red), the device that will act as the proxy client (green), and the target device for the proxy client to connect to over GATT (purple). Here we can assume that the proxy client has been operative for some time, without any previous entries in the connection list. The connection

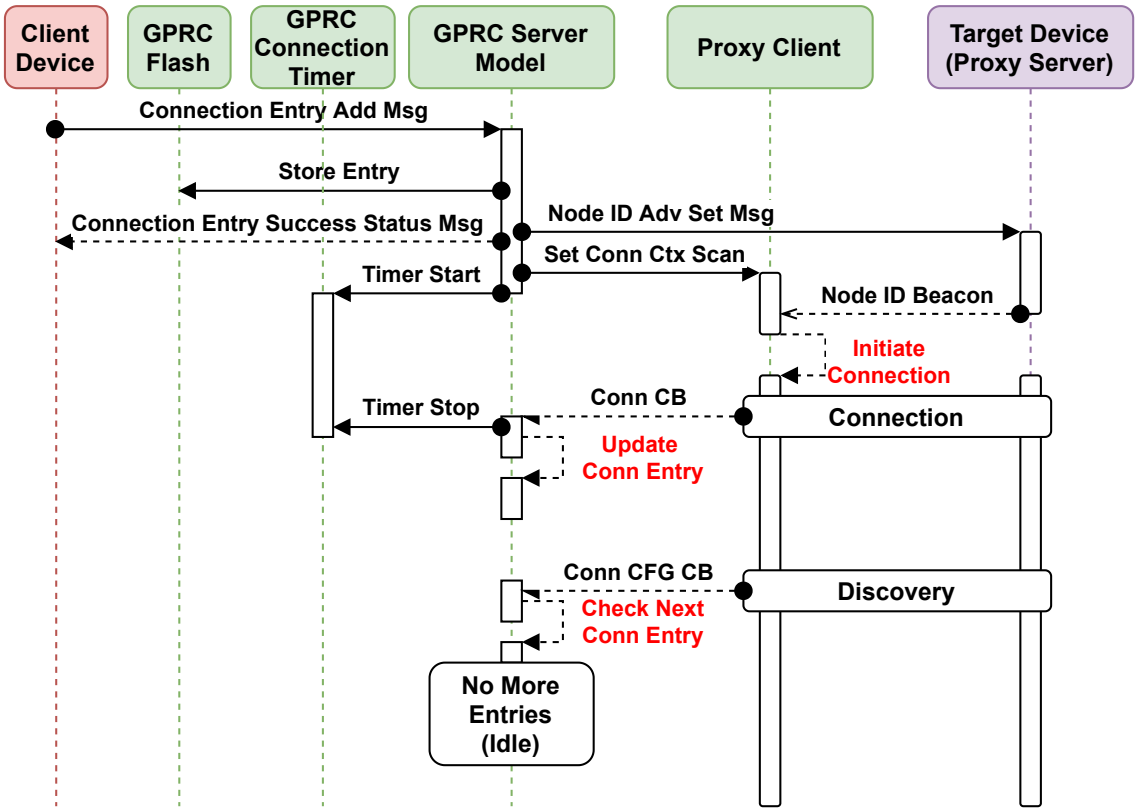


Figure 20: MSC diagram of standard connection over GATT proxy.

sequence is started when the GPC client issues a new connection entry for the proxy client device. As soon as this message is received, the connection entry list is updated and stored to the server models flash context. Then the GPC server will initiate the connection to the target device by issuing a request to the target to enable node ID advertising and setting the node ID scan context in the proxy client module. Simultaneously the model connection timer is started. When the target proxy server receives the initiation message, it will enable node ID advertising and start emitting beacons. As soon as the proxy client discovers one of these messages, it will initiate a BLE connection to the proxy server. When this connection is completed, the proxy client module will issue a connection completed callback that the GPC server module will receive. The connection timer is stopped from this callback, and the entry for the target proxy server is updated to a connected state. Simultaneously the proxy client is performing attribute discovery on the proxy server. As soon as this is completed, the proxy client module issues a configuration complete callback to the GPC server module. This completes the connection sequence between the two devices. Following this, the GPC server model will recheck the connection entry list to see if any additional entries are not connected. Since the entry list in this instance only contains one entry, the list is completed, and the GPC server model enters an idle state.

Figure 21 and 22 is a continuation of the same example described for figure 20. For both cases, we start in a state where the proxy client and server are in a connected state, and the GPC server model is idle. In figure 21 we can see that the proxy client device undergoes a system reset, causing the connection to the target proxy server to cease. The system reset also wipes all volatile memory from the proxy client, including all entries from the connection entry list in the GPC server model. Upon re-initiating the GPC server model instance, the model will start by retrieving all previously-stored connection entries from flash memory, thus restoring the connection entry list. After this task is completed, the model will initiate the same connection procedure as described for figure 20.

Lastly, we have figure 22. The system starts in the same state as in figure 21. In this instance, the target proxy server device "disappears" out of view for the proxy client. Disappearing in

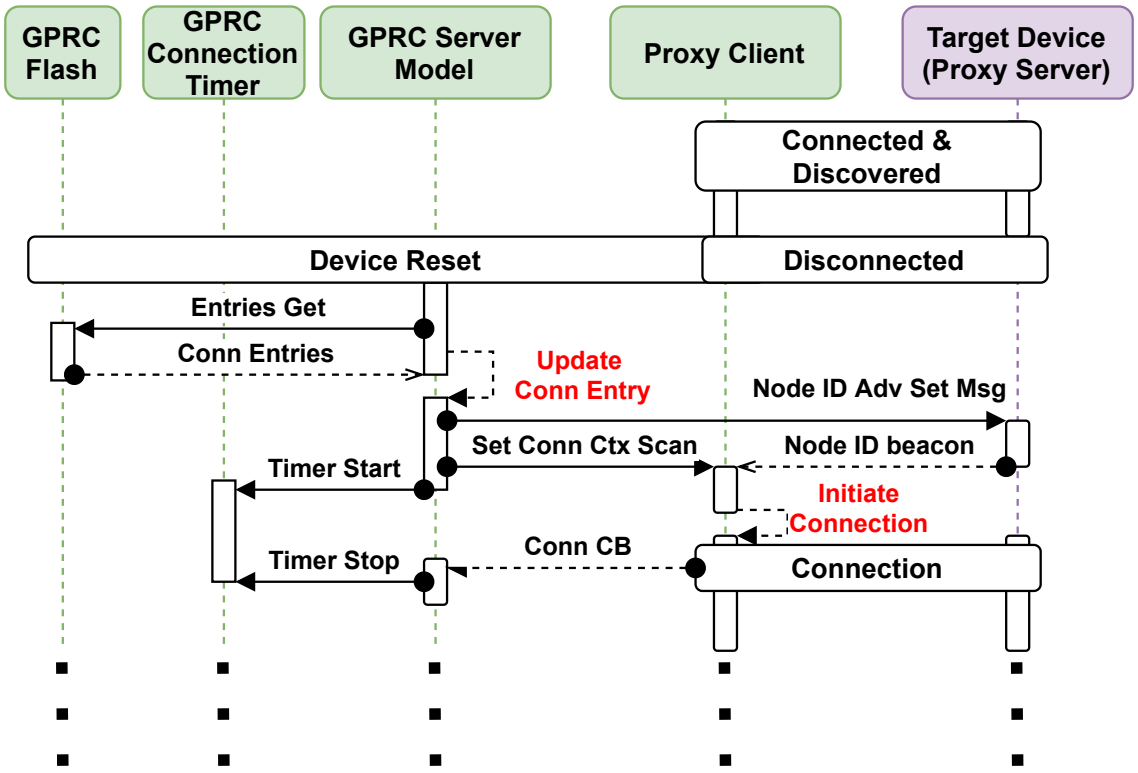


Figure 21: MSC diagram of recovery after proxy client reset.

this instance can imply several different reasons. For instance, the target device can have lost its power supply, it can have been moved out of radio range or it might be experiencing some faulty behavior. Since the target device is no longer responsive, the BLE connection will be terminated shortly after due to missing responses over the connection. This state change will be propagated to the proxy client module, which will fire a disconnect callback inside the GPC server module. Here the connection entry will be updated to a non-connected state, which will initiate a new connection procedure. A node ID advertising request will be issued to the target device, the node ID scan context in the proxy client module will be set once more, and the model connection timer is started. However, during these events, the target device has not yet "reappeared". The issued node ID request message will not be received by the target device, meaning that the proxy client is currently scanning for a node ID beacon that will never arrive. This will cause the model connection timer to timeout, creating an event that will restart the connection process. Before this second attempt, the target device has reappeared, meaning that this second attempt will unfold in the same manner as previously described, ending in a successful connection to the target device.

5.3.3 Controlling the ADV bearer

In an operational state, the behavior of the ADV bearer needs to be altered. Depending on if we are using the homogeneous or heterogeneous approach to the GATT bearer implementation, the transmission behavior of the ADV bearer should either be disabled or reduced.

During the development of this feature, I have examined the possibility of altering the operation of the ADV bearers scanning behavior. After due consideration, I have concluded that this option most likely is not the wisest choice. The main reason for this conclusion is that the ADV bearer is crucial for communicating with the BTM network in any situation where the GATT bearer implementation is not in an operational state. By allowing devices to still scan for and process incoming messages over the ADV bearer, the user is provided a fail-safe measure that allows communication with a device if anything with the GATT connections should malfunction. This gives the possibility of implementing commands that might revert the BTM network to an initial

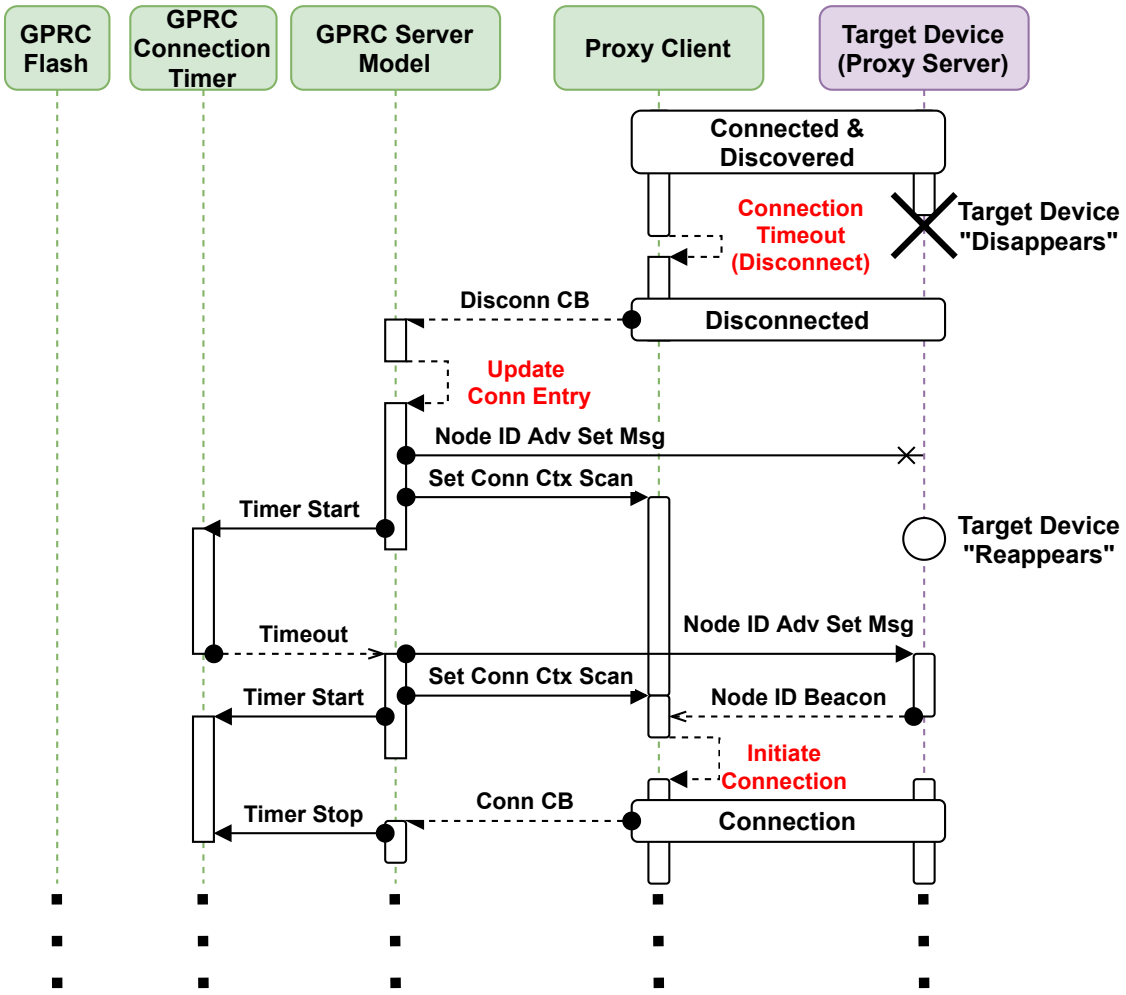


Figure 22: MSC diagram of recovery after loosing proxy server.

state where the issue can be corrected using the command message interface. If the ADV bearer scanning has been disabled and the GATT communication for some reason fails, the user might have to reset the device manually. In a worst-case scenario, it might even be necessary to update the software of the device. This can occur if the internal configuration leading to the faulty behavior is associated with state that is stored persistently and can only be altered using the command message interface. An additional factor is that the establishment of GATT connections is dependent on the ADV scanner to receive the beacons that will initiate a GATT connection.

To achieve the desired behavior, I have decided to implement three individual states for the transmission behavior of the ADV bearer:

- ADV Bearer TX Enabled — The ADV bearer is operating as normal. This state is intended to be used when the network is in its initial configuration phase and when devices recover their operational state after, e.g., reboot. In this state, a BTM device will act as any legacy BTM device.
- ADV Bearer TX Reduced — The ADV bearer's transmission behavior is reduced. A device will relay messages on the ADV bearer in this state, but the TTL value of all these messages is altered to allow only one hop from the transmitting device. This state is intended for a heterogeneous network solution. In this scenario, relay nodes will be using the GATT bearer to propagate messages between each other. In addition to these relay nodes, the network will contain one or more regular nodes utilizing the ADV bearer solely to communicate with the rest of the network. These devices rely on being adjacent with one or several relay nodes to enable communication with the entire network. A relay node will inherently not be aware of

any regular nodes nearby that depend on it forwarding messages to them. To ensure that no standard node is denied communication, each relay node must relay all messages over the ADV bearer. However, since each relay node is only responsible for forwarding messages to standard nodes adjacent to itself, there is no need to pass it using the original TTL value of the packet. By altering this parameter only to allow one hop on the ADV bearer, we can prevent the message from being scanned and relayed by any neighboring relay node, thus preventing unnecessary internal message handling and redundancy.

- ADV Bearer TX Disabled — The ADV bearer's transmission behavior is entirely disabled. This state is intended for a BTM network where all participating devices are interconnected through the GATT bearer. All outgoing messages will only be passed over the GATT bearer interface.

The active state is stored persistently by the GPC server model. The following model message type and behavior has been implemented to enable control of this state for a BTM device:

- ADV State Set — This message type is sent by the client model and received and handled by the server model. A message consists of one parameter that contains the new state for the advertising transmission behavior. Upon receiving an ADV state set message, the server shall update its internal ADV bearer TX state and store it in persistent storage.

As long as a GPC server is in an operational state, the ADV Bearer TX state will be equivalent to the configured state. However, as long as the connection link handler of a server is processing unconnected GATT entries, the ADV Bearer TX state will be overridden to the *ADV Bearer TX Enabled* state. This is to ensure that the server can pass the necessary *Node ID Adv Set* messages to any target proxy server to initiate a connection since these messages must be passed through the ADV bearer.

5.4 Utility Tools for Test and Development

Apart from the core functionality of my implementation, I am dependent on specific tools and functionality in software to enable me to control and observe system behavior under test and development properly. This section briefly describes the implementation of these tools and their contribution to the project work.

5.4.1 GPC Client Terminal

The GPC Client Terminal is developed to enable flexible control over all GPC server models at run time. The base software consists of the legacy implementation of the BTM stack. In this implementation, I have included an instance of the GPC client model, which provides control over all GPC commands described in section 5.3. From the application level of this software, I have implemented an instance of the shell module, provided by Zephyr[13]. This module provides serial communication over UART from the target device to a terminal program on a computer. It allows the user to implement custom commands in software that control internal functionality on the target device. Using this module, I have created a set of shell commands that correspond to all the GPC model commands. By using the terminal program TeraTerm[17] I can issue GPC commands to any participating device in the network, containing the appropriate parameters that follow each command.

5.4.2 GPC Test Commands

Whenever testing in a real network, it becomes more complicated to get detailed feedback from the devices. This is because it is not feasible to extract the log from every single device when remotely

deployed. In this situation, we must settle for the means that the device itself can provide to observe the system's state, commonly provided by controllable LEDs on the devices. By turning these LEDs on or off in response to certain events, we can at least get an indication of if the system is behaving as expected.

For functional testing in a real network, I require a way to verify that a specific device can communicate with the other devices in the network. A crucial criterion for this is that we must be sure that the observed result from a test is a product of a message sent from the device we are testing. A simplistic way to solve this could have been to use GPIO induced messages that are activated directly from the device we are testing. However, this solution would require physical presence to the device whenever performing a test, which would impede the testing progress. Instead, I have created a solution where this test can be activated using model messages. The messages required to perform this feature are implemented as supplement commands to the GPC model. These messages are only meant for testing purposes and are not a part of the core functionality that the model provides for the GATT bearer solution.

In order to realize this functionality, the following model message types and behaviors have been implemented:

- Test Initiate — This message type is sent by the client model and received and handled by the server model. The message contains a single boolean parameter that represents an on/off state. Upon receiving a Test Initiate message from a client, the server shall retrieve the boolean parameter value before issuing a Test message containing the same parameter value that followed the initiation message. This message is sent to the group address commonly shared by all GPC servers in the network.
- Test — This message type may be sent by the server model and is received and handled by the server model. The message contains a single boolean parameter that represents an on/off state. Upon receiving a Test message from another device, the server shall retrieve the boolean parameter contained in the message. Depending on the value of this boolean, the server shall either turn on or off a LED located on the server device, henceforth denoted as the *Comm Test* LED.

By using the GPC Client Terminal described in section 5.4.1, these message types enables remote control of this functionality. Since the ADV scanner always will be active when using the GATT bearer solution (5.3.3), all devices will be able to receive the Test initiate message from the client device. Upon receiving this, the server device will generate the actual test message, which will attempt to forward the boolean parameter to the whole network. In this way, we can ensure that the message that potentially will alter the state of the *Comm Test* LED originates from the device that is currently being tested.

A communication test is conducted by following these simple steps:

1. Ensure that the *Comm Test* LED of every device in the network is in a commonly known state (On or Off).
2. Issue a Test Initiate message containing the opposite boolean state to the device undergoing testing.
3. Observe the new state of the *Comm Test* LEDs on each device. If the state corresponds to the value sent in the Test Initiate message, we know that the DUT could communicate with that particular device.

5.4.3 Mapping Assessment Program

As described in section 5.3.1, the GPC models offer the base functionality to provide data for the available links in the network. Still, the model does not in itself provide centralized control

for performing the entire task. Performing this task manually would require multiple commands from the GPC client and would be a time-consuming and complex task. Additionally, the data that the model provides comes in a raw format and needs some additional computation to produce a result that would be easily readable. With this in mind, I have decided to create a mapping assessment program. This software will have two main tasks. The first one is to automate the process of initiating the mapping sequence and collect the mapping data when the sequence is completed. The second task is to process the raw mapping data and produce a comprehensible visual result for the human eye.

I have decided to implement this software in Python. The program will interface with the BTM by piggybacking on the already implemented GPC Client Terminal. This is done by using the serial module that Python provides. By parsing the log output typically sent to the terminal program, we can achieve both TX and RX capabilities for the Python software. The serial receiver handler can read every single line that is passed from the target to the terminal. These lines scan for a specific message format and opcode and perform different tasks depending on the actual opcode. When issuing commands to the target, the Python program utilizes the same shell commands that are implemented for section 5.4.1.

5.4.3.1 Mapping Sequence and Processing

A mapping sequence is initiated by sending a Link Update Initialize message to the common group address from the Python API. This message contains the common number of Link Update messages that each server will emit to their adjacent server peers. By utilizing the status messages received from the servers, the Python controller can operate as a state machine. Each server will respond with a status message to the Link Update Initialize message, telling the client that the mapping sequence has started. The Python controller stores the addresses for each of these individual status responses in a list called *initiated_list*. When a server has sent the last Link Update message, it will send out another status message, telling the client that the Link Update sequence has been completed. When these messages start to arrive from the different servers, the Python controller stores each entry in a list called *completed_list*. For each new entry that is appended to this list the controller will compare the *completed_list* to the *initiated_list*. When these two lists match, we know that all servers have completed their Link Update sequence.

At this point, the controller will start issuing Link Data Fetch messages, using the addresses contained in the *completed_list*. As soon as the client receives the link context from the server, the controller will store this entry internally in a dictionary and issue a Link Data Fetch message to the next server. This procedure is continued until the controller has iterated through the entire *completed_list*. At this point, all the raw data from the mapping procedure is stored in the internal Python dictionary. This dictionary is passed to a method that iterates through every entry in the dictionary, creating a *SortedEdge* class object for each link between two devices. These objects are appended to a separate dictionary. The *SortedEdge* object contains the address information of the two devices that form the link, as well as information of how many messages it is expected to receive on the link and the actual number of received messages. Since a link's state depends on the link context from both devices that form the link, the class is fitted with an update method. When the iteration method discovers an address pair that already exists as an *SortedEdge* object, it uses the update method instead of creating a duplicate for the link. The *SortedEdge* object also contains a *weight* parameter. This parameter is calculated by dividing the number of received messages by the number of expected messages. This implies that an ideal link will have a *weight* value of one.

When the sorting is completed, we have a refined set of link data that can be feed into the Python module *networkx* and plotted using the module *matplotlib*. A *networkX* graph object is first created, then the node and link information is added to this graph as nodes and edges, respectively. The edges can be drawn with different colors and with their weights as labels in the plot. For this instance, I have decided that edges with a value greater or equal to *0.95* will be plotted as green, representing links of *Great* quality. The edges with a weight between *0.95* and *0.90* will be plotted as blue, representing links of *Intermediate* quality, while edges with a weight less than *0.90* will be

plotted as red, representing links of *Poor* quality[3].

5.4.4 Miscellaneous

5.4.4.1 All Connections Established LED Indication

As stated in section 5.4.2, the level of observability on a device is significantly reduced when it is deployed in a real network. I have implemented a LED indicator that is activated whenever this list is completed, henceforth called the *All Connections Established* LED. This LED provides a way to verify that a GPC server is connected to all proxy servers in the connection entry list. As long as the GPC server is in its operational state, this LED indicator will be turned on. If the server loses the connection to a target proxy server, the LED will immediately turn off.

[3]The choice of range for the quality of the links in this instance is picked pseudo-randomly to get a better visual view of the links. Generally, the assessment should be based solely on the *weight* value of the link.

6 System Testing

6.1 Functional Testing of the System

At this point, I have concluded the design and implementation of the proxy client module, the GPC model, and the required interfacing with the rest of the BTM stack. In order to assess the quality of my implementation's functional behavior, I have decided to perform a behavioral test conducted in a realistic environment using proper hardware. In this setup, I will test the mapping feature of the GPC model. Based on this data, I will find and apply a sufficient configuration scheme to the network. This includes a solution for both the homogeneous and heterogeneous approaches. When the network is in its operational state, I will conduct tests to see if the participating devices can pass messages throughout the network using the respective solution for each approach. In addition, I will introduce scenarios that will show if the network is capable of recovering following an unforeseen event.

The network consists of seven nRF52 development kits (DK)[5] containing the proposed GATT bearer implementation, including an instance of the GPC server model placed in the primary element of each device respectively. These DKs are placed throughout an office space with a fairly even distribution. This placement intends to provide a BTM network that depends on relaying messages to ensure that messages can be passed between any two participating nodes. The setup can be viewed in figure 23. Initially, I have no knowledge of which devices can communicate directly or the quality of these links. At the starting point of this test, all nodes are configured as relay nodes using the ADV bearer. This is to ensure that it is possible to control any node using the GPC client model remotely.

Figure 23: Initial view of the functional test setup.

The controlling GPC client model is set up on an additional DK. This kit is connected to a laptop through the shell serial interface described in section 5.4.3. This enables control of the client model commands directly through a terminal and the Python mapping assessment software. The initial configuration of each device entails provisioning and configuration through the nRF Mesh mobile application[4]. All GPC models are bound to the same application key, and both the server and client models have been set to publish and subscribe to a common group address. This configuration ensures that all GPC messages will flow as intended in the network.

6.1.1 Mapping and Configuration Assessment

Using the Python mapping assessment program, I have conducted a mapping sequence with a message count of 100 messages per server. The resulting network topology can be viewed in figure 24. The plot is created using a planar layout for the network, preventing intersection between edges. I have converted the results to a representation that adheres to the devices' actual placement in the network. I have done this in order to get a better overview of the topology. This representation can be seen in figure 25.

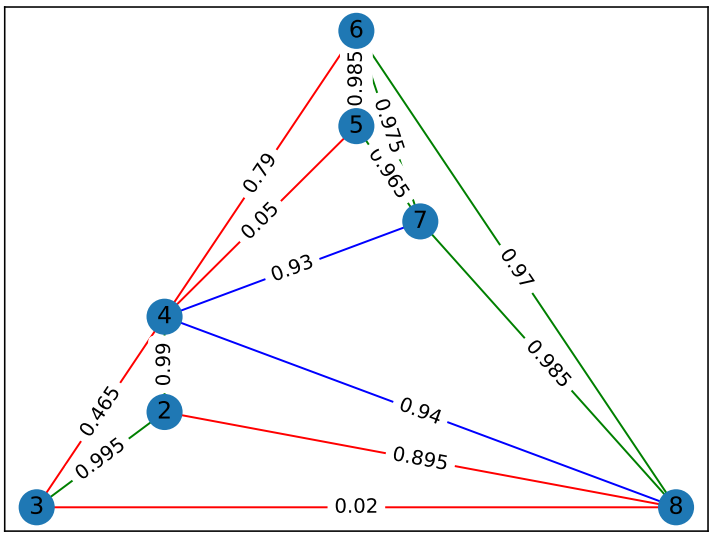


Figure 24: Link mapping processing output.

The left side of this figure shows every single link detected by the GPC server models, while the right side excludes all links that have a *Poor* link quality (5.4.3.1). By assessing the right side, we can quickly see that nodes 2 and 4 should be a part of the relay network since these are the only nodes that can provide links of higher quality to bridge the communication to node 3. In a situation where no link assessment tool was available, it would be reasonable to guess that these nodes would be the best candidates for the role of relay nodes. They are located reasonably close

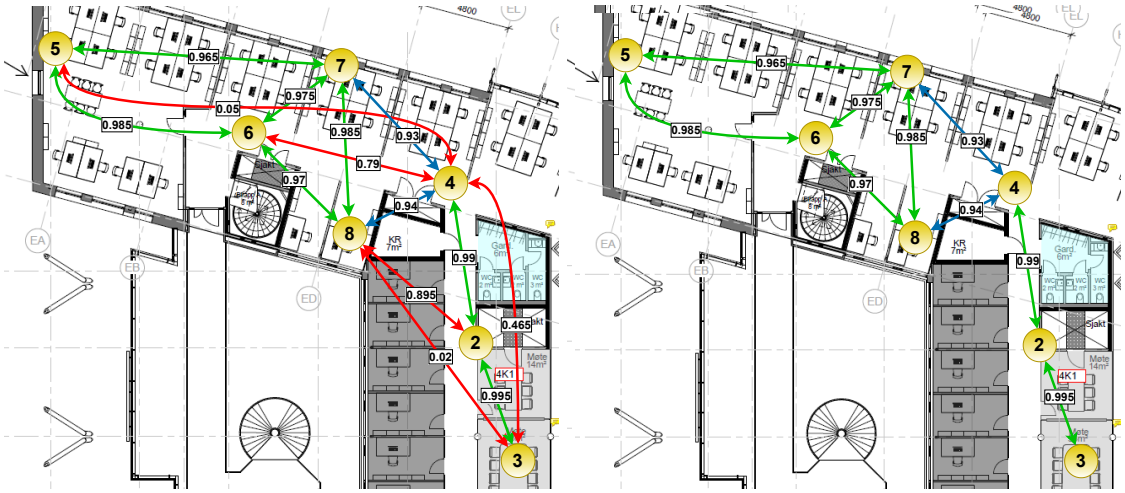
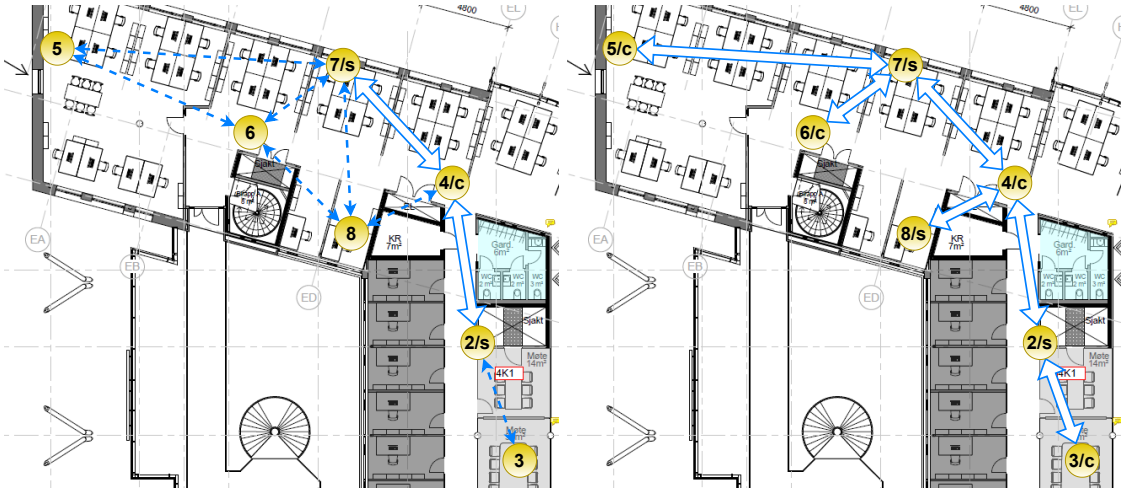


Figure 25: Functional test setup overview after mapping.



(a) Heterogeneous configuration solution.

(b) Homogeneous configuration solution.

Figure 26: Functional test configuration solutions.

to each other, in addition to being in a line of sight.

More than one configuration might provide robust connectivity for the five remaining nodes located in the upper part of the figure. Here we may choose either node 7 or 8 to link with node 4. By, e.g., choosing node 7, we end up with a configuration where nodes 2, 4, and 7 provide the relaying backbone of the network. In this situation, all other nodes are within a single hop from one of the three relay nodes, making this a valid configuration for a GATT bearer network using the heterogeneous approach. In this instance, I have decided that node 4 will act as the sole proxy client device, maintaining two GATT connections to a proxy server on nodes 2 and 7, respectively. The remaining node (3, 5, 6, 8) will act as regular ADV bearer nodes without relaying capabilities. The heterogeneous configuration is shown in figure 26a. Here we can see the GATT connections represented as larger arrows spanning between the three relay nodes.

In comparison, the dashed smaller arrows represent communication that is provided over the ADV bearer[4]. A *c* or *s* is attached to each of the relay nodes. These letters mark the role of the relay nodes

For the homogeneous solution, we can use the same base assessment for the heterogeneous solution, where the same three nodes make the foundation for the base relay network. To complete the solution, we need to choose GATT connections to link the remaining four nodes to one of the three base relay nodes. The chosen configuration is shown in figure 26b.

6.1.2 Heterogeneous Configuration Testing

The configuration of the heterogeneous solution is performed using the following steps:

1. Two *Connection Entry Add* messages are issued to node 4, telling it to establish GATT connections to the proxy server on nodes 7 and 2, respectively.
2. The ADV transmission behavior state of nodes 4, 7, and 2 is changed from the *ADV Bearer TX Enabled* state to the *ADV Bearer TX Reduced* state, using the *ADV State Set* message.
3. The remaining nodes are configured to disable the relay feature, making them act as standard BTM nodes.

[4]In figure 26a all ADV bearer links of lesser quality has been removed to provide a cleaner representation. Note that they are still present.

At this point, the configuration of the heterogeneous solution is completed. By monitoring the *All Connections Established* LED indicator (5.4.4.1) on node 4 I am able to confirm that all GATT connections is successfully established.

Proceeding, I start to utilize the GPC Test Commands (5.4.2) to check if every device can communicate with the rest of the network. This is performed by issuing a Test Initiate message to each device iteratively and confirming that the other devices received this message by checking the *Comm Test* LED. After completing this procedure, I can confirm that all devices can communicate in the network.

6.1.3 Homogeneous Configuration Testing

Configuration of the homogeneous solution is performed as an extension of the configuration of the heterogeneous solution. Node 2, 4, and 7 keeps their current configuration, in addition to the following configuration steps:

1. Node 3, 5, 6, and 8 are configured to enable their relay feature.
2. A *Connection Entry Add* message is issued to node 5, telling it to establish a GATT connection to the proxy server on node 7.
3. A *Connection Entry Add* message is issued to node 6, telling it to establish a GATT connection to the proxy server on node 7.
4. A *Connection Entry Add* message is issued to node 3, telling it to establish a GATT connection to the proxy server on node 2.
5. A *Connection Entry Add* message is issued to node 4, telling it to establish a GATT connection to the proxy server on node 8.
6. The ADV transmission behavior state of all nodes is changed to the *ADV Bearer TX Disabled* state, using the *ADV State Set* message.

After completing the configuration, I proceed by checking the *All Connections Established* LED indicator on all nodes, successfully verifying that all GATT connections are established. At this point, I repeat the procedure, checking if every device can communicate with the rest of the network using the GPC Test Commands. Like for the heterogeneous configuration, I can verify that every device can communicate in this network.

6.1.4 Introduction of Unforeseen Events

After verifying that the network can function in a normal state for both the heterogeneous and homogeneous configuration, the time has come to introduce some obstacles and see how the network handles them. At this time, the network uses the homogeneous configuration, which should provide a sufficient basis to check the behavior under non-ideal circumstances for both solutions. The argument for this statement is that the heterogeneous solution is a subset of the configuration for the homogeneous solution. This implies that if the network can recover in this scenario, it should also recover in the heterogeneous configuration.

In this part of the functional test, I will introduce the following two unforeseen events:

- Power loss for node 4 — I will turn off node 4 for 30 s before turning it on again. At reboot, I will check if node 4 can re-establish its previous connections to nodes 2, 7, and 8. Succeeding in this, I will continue by checking if node 4 can still relay messages throughout the network.

- Power loss for node 7 — I will turn off node 7 for 30 s before turning it on again. At reboot, I will check if nodes 4, 5, and 6 can re-establish their previous connection to node 7. Succeeding in this, I will continue by checking if all these nodes can still communicate with the rest of the network.

I have started by performing the power loss for node 4. After turning off node 4, I have waited for approximately 30 s before turning it on again. While monitoring the device's *All Connections Established* LED indicator, I can confirm that the LED turns on after a couple of seconds, indicating that all prior connections have been re-established. By using the GPC Test Commands (5.4.2), I can confirm that the devices in the network are still able to communicate with each other. After completing this test, I continue by performing the power loss test for node 7. Following turning off node 7, I have verified that the *All Connections Established* LED indicator of nodes 4, 5, and 6 have turned off. After 30 s has elapsed, I turn the device on again. Upon rebooting node 7, I have checked the LED indicators for the connecting nodes once more and confirmed that they have re-established their prior connection to node 7. Using the GPC Test Commands, I can still verify that the devices in the network can communicate with each other.

6.1.5 Function Test Summary

The combined results of this functional test sequence are promising. It has shown that it is possible to form, configure, and maintain a BTM network based on GATT connections. The test has also shown that the utility tools are working as intended and that the network can recover from unforeseen events. I will state that this confirms the proof of concept for a BTM network based on several GATT connections on a functional level.

However, this test setup can by itself not verify any eventual performance improvement of the BTM network. Section 6.2 will cover the performance testing of this thesis, where I evaluate if the GATT bearer implementation can provide better capabilities concerning both throughput and data consistency.

6.2 Performance Testing

At this point, I have been able to prove that the GATT bearer solution can replace the ADV bearer in the network on a functional level. However, the functional tests conducted do not provide any insight into the potential performance enhancement that the GATT solution can provide for throughput and data consistency. In order to evaluate these performance metrics, I need to create an appropriate test environment. I have decided to establish a scenario similar to those used in the simulations in section 4.6.

6.2.1 Test Setup

6.2.1.1 General Setup Conditions

An initial challenge for this test scenario is to provide sufficient monitoring capabilities for the participating devices while deployed in a realistic network topology. While these difficulties could be overcome in the functional testing by utilizing primitive LED monitoring, this issue is not as easily overcome for the performance testing. In this scenario, we rely on a much higher degree of monitoring, especially concerning timing-sensitive matters. This entails using proper measuring equipment, which provides the required timing resolution for the performance test. In a realistic network topology, this creates a problem since the measuring equipment needs to be connected to two or more devices with a substantial physical distance between each other.

My proposed solution to this issue is to utilize a desktop setup for performance testing. The

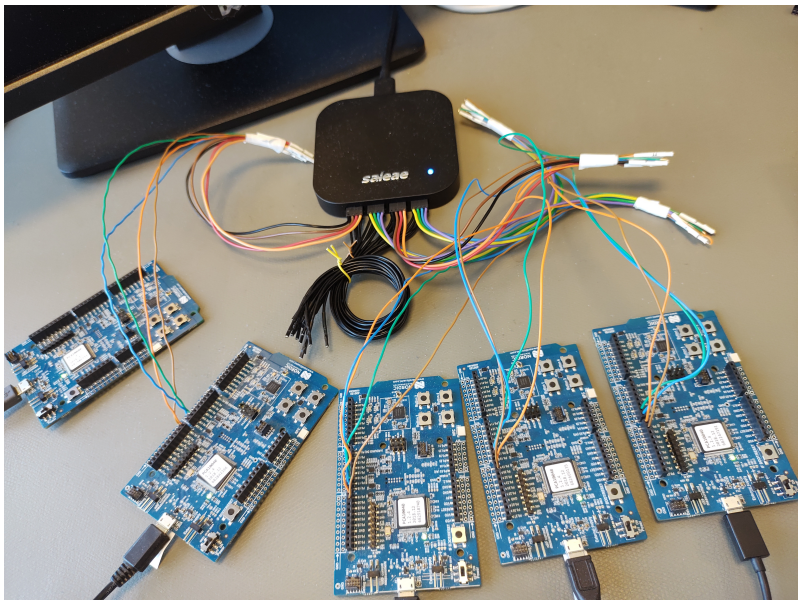


Figure 27: Performance test setup.

devices under test are placed in direct proximity to each other and are connected to a logic analyzer. The setup is shown in figure 27. The hardware used in this setup consists of four nRF52 DKs[5] connected to a Saleae Pro 16 logic analyzer[14], providing the potential to test network performance for up to four BTM nodes simultaneously. The scheme in this setup is to utilize GPIO traces that can be placed inside relevant function calls in the BTM stack that we wish to monitor. A GPIO trace is created by toggling a GPIO pin twice immediately inside the function call, creating a small square pulse signal that the logic analyzer can register. A sample of this pulse is depicted in figure 28.

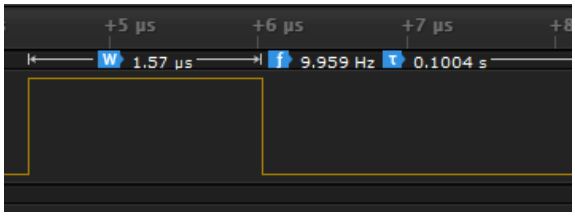


Figure 28: Logic pulse sample.

By simultaneously monitoring pulses on both transmitting and receiving devices, we can now monitor the time it takes to transmit a train of messages and how many of them that successfully reached the destination device. The logic analyzer provides a shared timeline for the measurements, making it easy to extract the exact time it took to perform the message transmission.

This setup choice has some benefits and some drawbacks. The benefit is that this setup provides optimal monitoring capabilities of the network behavior, giving us a looking glass directly inside the BTM stack of several devices simultaneously with a common time reference. In a setup with satellite devices, the issue with the common timeline would become very hard to overcome since all devices operate with a different system clock. The main drawback is that we do not have a network that has a realistic network topology in this scenario. This means that the test results will not account for all the effects that normally are present in a network, like physical obstructions, long distances between devices, and other considerations. Another consequence is that it becomes challenging to perform elaborate testing on the ADV bearer since it broadcasts all its messages to any device that is within radio proximity.

At this point, I need to review these circumstances and assess if they are acceptable with regard to the purpose of the performance test. The test aims to produce performance data for the

ADV bearer and the new GATT bearer solution. The results will be compared to see if the GATT bearer solution performs better than the ADV bearer with respect to throughput and data consistency. Since this will be a relative comparison between two solutions that have been in the same environment, I deem it acceptable that the test is performed in a setup where the network topology is not realistic. I admit that an ideal test solution would be to conduct the test in a realistic environment, but due to restrictions in equipment, time, and convenience, it is not feasible at this time. The most important aspect is that this setup will produce valid results. When it comes to the restrictions concerning the ADV bearer testing, I think that this setup can provide an adequate environment to produce the required results. In this setup, I am not trying to provide extensive data on the ADV bearer performance in different network setups. What I require is a suitable frame of reference from the ADV bearer that can be compared to the performance of the GATT bearer. Such a frame of reference can be provided by executing a message transmission between only two devices, which this test setup can provide. When it comes to testing the GATT bearer, the same issue will not be present since the GATT bearer utilizes links between devices, meaning that we can perform more elaborate testing. I have chosen to utilize the homogeneous network approach explicitly for the GATT bearer testing. The reason for this decision is that I am mainly interested in comparing the performance between the different bearers in this instance, making the homogeneous approach the preferred choice among the two possibilities. I admit that performance testing of the heterogeneous approach is also of interest, but due to time restrictions, I have decided that this matter must be regarded as future work.

For the test scenario, I have decided to use a similar setup as the one used in the simulation section of this thesis(4.6). A single test will consist of a train of messages sent from a single origin device, emulating a DFU procedure. Every single message should ideally be received by all other participating devices in the network. Each message is an unsegmented message containing 11 bytes of data and will be sent from the model layer of the BTM stack. The speed at which the messages are issued from the origin device is variable, providing the possibility to test the procedure at different rates. In a situation where the speed of the DFU exceeds the capabilities of the BTM implementation, we should expect to see some failure or error from the participating devices. At this point, we will know that we have reached the system's peak throughput performance. In the simulation setup, 13637 messages were used to simulate the DFU procedure. Here I will reduce this amount to 5000 messages. This decision is based solely on convenience since we must experience the DFU emulation in real-time. I deem that 5000 messages are sufficient to prove if the network can handle the workload of a DFU procedure under a specific TX rate. Simultaneously, this will spare enormous amounts of time since the testing will be conducted for several scenarios and TX rates.

6.2.1.2 Test Scenarios

There are two primary purposes for conducting performance testing for this implementation. The first one is to see if the actual performance results comply with the results that were produced under the modeling and simulation of this thesis(4). The results can either validate or reject the initial assumptions made during the thesis's preliminary modeling stage. The other reason for the performance testing is to see if the implementation can provide the desired performance enhancement that is a central goal of this thesis.

The tests I have decided to perform are divided into two categories. The first is a baseline test, where I wish to test and compare the relative performance of the ADV and GATT bearer for a similar test scenario. The purpose of these tests is to determine if the GATT bearer is better suited to provide the properties that we desire in this instance. The second category is an extension of the first one, where I wish to see how the performance of the GATT bearer is impacted when it has to handle more than one GATT connection at the same time.

The test scenarios that will be conducted during the performance testing are:

- ADV bearer baseline — This test will provide the baseline for the legacy ADV bearer, used for further comparison with the other tests in this section. This setup will consist of two

participating BTM devices, where one device will use the ADV bearer to forward the DFU message train to the other device.

- GATT bearer baseline — Similar to the ADV bearer baseline, this scenario will show the performance of the GATT bearer in a single connection environment. While these results may not fully apply to the GATT bearer network solution, they will provide an important insight into the relative performance between the ADV bearer and the GATT bearer.
- Three device GATT network — This test will show the network's performance when utilizing two GATT connections on a single device.
- Four device GATT network — This test will show the network's performance when utilizing three GATT connections on a single device.

6.2.1.3 Tracing and Measurement Methodology

Four main GPIO traces will be utilized under testing. One trace is placed inside the message handler on the model layer. This will enable me to monitor the exact timestamp of arrival for each unique message. The three remaining traces are placed inside the sending call of the ADV bearer, proxy server, and proxy client module, respectively. The total execution time of the DFU emulation will be found by measuring the time elapsed between the first sent message and the last received message. The first sending pulse will be registered in either of the three sending calls, while the last received message pulse is registered in the receiving device model handler. Two considerations need to be addressed here. The first is that the measurement of the completion time will not account for any lost messages during the DFU emulation. The number of successfully received messages during the test will be registered at the model layer and extracted upon test completion. The other is that the completion time is dependent on the rate at which the transmitting device is issuing the messages. This means that the completion time is likely to be comparable to $N \cdot TX_{Rate}$, where N denotes the total number of messages in the DFU and TX_{Rate} denotes the rate at which the messages are issued.

In the initial testing of this setup, I discovered some inaccuracy between the applied transmission rate and the actual transmission rate. The output of the logic analyzer shows that the time-delta between some successive messages tends to be slightly larger than the applied rate. This is likely due to some implementation-specific circumstances either within the Zephyr implementation or the Nordic Hardware. To account for this, I have decided to present both the applied and effective transmission rates to avoid inaccuracy in the presented results.

6.2.2 ADV bearer baseline

The ADV bearer baseline test consists of two devices utilizing the legacy ADV bearer. One device will transmit the entire DFU to the other device, using a transmission count of a single message. The reason for this decision is based on the findings of the simulation setup. In section 4.6 I have shown that the likelihood of successful message transmission with the ADV increases when adding message retransmissions. Retransmitting messages does, however, increase the execution time, where we can expect to see a linear relationship between the execution time and the number of retransmissions per message. Since we here are interested in comparing the relative performance of the ADV and GATT bearer, I deem that it is sufficient to perform a real test for only a single message transmission. I have decided to start the testing at a TX rate of 100 ms. Assuming that the initial test is successful, I will continue to perform tests for lower TX rates until I reach the theoretical peak rate of 25 ms.

In the initial testing on the ADV bearer, I have discovered something noteworthy. During the first attempted test with a message rate of one message per 50 ms, the transmitting device has suddenly prompted an error message. This message indicates that the ADV bearer has run out of space in the transmission buffer. By reviewing the output of the logic analyzer, I have discovered that the transmitting device cannot transmit messages at a rate faster than approximately 62

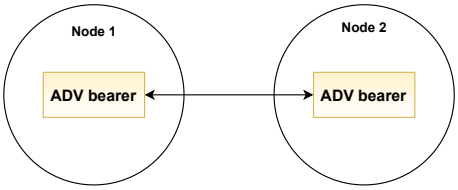


Figure 29: ADV bearer performance baseline setup.

ms per message. A snip out of this measurement is shown in figure 30. This indicates that something within the BTM implementation restricts the maximum rate at which messages can be sent. Attempting to issue messages at a higher rate than this will effectively cause a bottleneck at the transmitting ADV bearer, eventually overflowing the transmission buffer. Since the theoretical maximum speed for advertisement transmission is 25 ms, I need to investigate the reason why the ADV bearer is limited to a peak rate poorer than this.

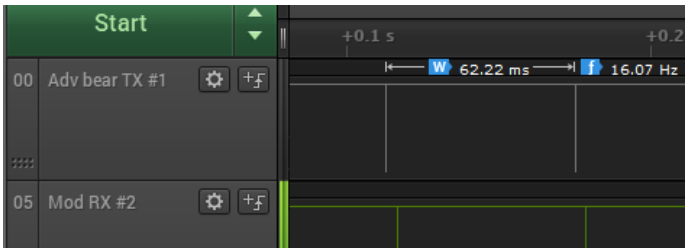


Figure 30: Restricted ADV bearer TX period.

After reviewing the ADV bearer implementation, I have found two factors that affect the maximum transmission rate of the ADV bearer. Three different factors in this implementation define the delay between consecutive outgoing messages. The first one is a 20 ms advertising interval, which is in compliance with the minimal advertising interval as described in section 2.2.4.1. The second factor is a 10 ms constant that represents the adverting delay. This discovery is interesting since this represents a worst-case adverting delay rather than the mean adverting delay that has been utilized in the simulation assessment for this thesis. The last factor is a 30 ms delay, which represents an adverting scanning window. I have found that the reason for this additional delay is to account for some BLE implementations that require a scanning window to complete before taking on a new transmission[16].

I have decided to disable this scanning window for testing purposes since this particular implementation is not dependent on it. In this way, I will be able to assess the performance of the ADV bearer down to a rate of 30 ms per message. In addition, I have decided to alter the adverting delay compensation down to 5 ms in order to see how the implementation will behave when the peak 25 ms transmission rate is applied.

Msg TX Speed Applied(ms)	Msg Loss Cnt	Msg Loss(%)	Execution Time(ms)	Effective Mean TX Speed(ms)
100	133	2.66	501846	100.37
50	103	2.06	253590	50.72
35	331	6.62	179310	35.86
30	219	4.38	154026	30.81
25	2381	47.62	130798	26.16

Table 11: ADV bearer performance test.

Table 11 shows the combined results for the adverting bearer performance. In an initial assessment of these results, there are a couple of circumstances that are worth mentioning. The first is that none of the tests could complete the entire DFU emulation without considerable message loss. This circumstance was not entirely unexpected since the simulation results in section 4.6 showed that

the ADV bearer would struggle to deliver all messages without any redundant retransmissions. I did, however, not expect the magnitude of lost messages to be this high, considering that this test is performed on a desktop setup with only two devices.

Another interesting aspect is the result for the 25 ms transmission rate. This test shows that the transmitting device can successfully deliver roughly half of the messages in the DFU emulation. This is a quite poor result, indicating that it is not feasible to transmit messages at the theoretical peak rate, at least not for this particular implementation. The reason for this issue might be related to the advertising delay. In my assessment of the peak transmission rate for the ADV bearer, I have considered the mean time for the advertising delay. Perhaps we must consider the worst-case advertising delay instead, making the theoretical peak rate 30 ms. The result for the 30 ms transmission rate shows a considerable better performance than the 25 ms rate, losing approximately $\frac{1}{10}$ of the messages in comparison.

6.2.3 GATT bearer baseline

The GATT bearer baseline test is conducted by setting up two devices running the GATT bearer implementation. Both devices have been configured to disable the ADV bearer, and one device is set to establish a GATT connection to the other.

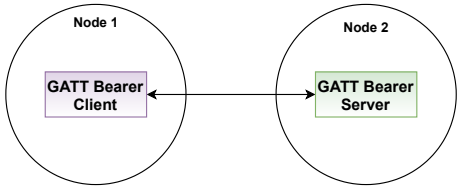


Figure 31: GATT bearer performance test baseline setup.

In order to assess if the GATT role has any impact on the performance under test, I have decided to run each test for two scenarios; one where the server device is the origin of the DFU emulation, and another where the client device is the origin. With the assumption that the GATT bearer will perform better than the ADV bearer, I have decided to start the testing at a TX rate of 25 ms. Assuming that the initial test is successful, I will continue to perform tests for lower TX rates until I reach a point where the implementation cannot handle the workload.

Msg TX Speed Applied(ms)	Msg Loss Cnt	Execution Time(ms)	Effective Mean TX Speed(ms)	Note
25	0	129427	25.89	
10	0	54781	10.96	
9	0	50065	10.01	
8	0	45494	9.10	
7	-	-	-	Failure

Table 12: GATT bearer performance test baseline. Server transmitting to client.

The results for the GATT baseline test is shown in table 12 and 13. Table 12 presents the result for the test where the server was the origin of the DFU emulation, while table 13 shows the results for when the client inhabits this role. If we compare these two tables, we can see that they are quite similar, indicating that the GATT role does not significantly impact the performance on a single connection basis. Acknowledging this, we can move on to the more interesting aspects of the results. The first important observation is that every successful test did not lose a single message of the DFU emulation. The second is that the GATT bearer can handle a considerably higher transmission speed than the ADV bearer.

In this test setup, I was able to run successful tests down to an effective transmission speed of approximately 9.10 ms. If we consider the results from section 6.2.2, this is a transmission rate that

Msg TX Speed Applied(ms)	Msg Loss Cnt	Execution Time(ms)	Effective Mean TX Speed(ms)	Note
25	0	129302	25.86	
10	0	54772	10.95	
9	0	49991	10.00	
8	0	45043	9.01	
7	-	-	-	Failure

Table 13: GATT bearer performance test baseline. Client transmitting to server.

is *3.38* times faster than the ADV bearers peak effective rate of 30.81 ms. Let us simultaneously consider that the ADV bearer lost a considerable amount of messages at its peak rate, while the GATT bearer lost none. This consideration indicates that the GATT bearer is inherently better suited for conducting a DFU than the ADV bearer. Based on the baseline testing, it has been shown that it can provide both better throughput and data consistency performance than the ADV bearer.

At applied TX rates lower than 8 ms the implementation started to experience issues in several ways. These issues consisted of either buffer overflows or loss of connection over GATT. While it would be interesting to explore the possibility of achieving even higher transmission speeds than this, time limitations prevented me from exploring these matters further. The primary purpose of this benchmark was to provide data to analyze the relative performance of the GATT bearer and ADV bearer, which I deem that it has done successfully.

6.2.4 Three Devices GATT Bearer Performance

Extending the baseline performance test for the GATT bearer, I have now set up a test scenario where the performance of the implementation will be tested for a more realistic setup. In this scenario, we have three devices participating in the DFU emulation. An overview of this setup can be viewed in figure 32. One of the devices must, in this instance, handle GATT communication over two links simultaneously. While the baseline testing showed promising results for the GATT bearer regarding both throughput and data consistency, the system must still prove that it can provide similar results for a situation where it is handling multiple GATT links at once. This test is a critical point in the thesis work since two GATT links per device are the minimum requirement needed to facilitate any network topology with the GATT bearer. In section 4.6.3.1 a potential chain topology was discussed for the GATT bearer. This scheme could be utilized if the implementation can provide acceptable performance under these circumstances.

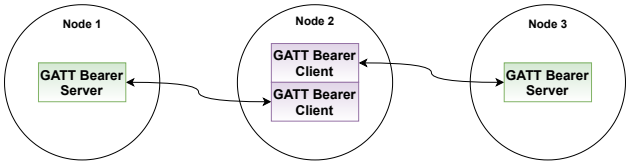


Figure 32: GATT bearer performance test setup with three devices.

As a starting point for this setup, I will start at the same transmission speed as for the GATT baseline test and gradually increase it if the system can handle the applied speed. This will eventually reveal if the two-link configuration can provide similar performance, as shown in the baseline testing. For this test setup, I will run two DFU emulations for each applied transmission rate. The first one will be executed using node 1 as origin, which is located on one of the ends of the network chain. The second emulation will be performed with node 2 as origin, located in the center of this three device chain.

The results of the performance testing for the three-device GATT network is presented in table 14 and 15. By comparing the results of these two tables, we can see that both test scenarios

Msg TX Speed Applied(ms)	Msg Loss Cnt Node 1	Execution Time(ms) Node 1	Msg Loss Cnt Node 3	Execution Time(ms) Node 3	Effective Mean TX Speed(ms)	Note
25	0	129947	0	129972	25.99	-
20	0	104956	0	104984	20.99	Sporadic Failure
17	0	89996	0	90001	18.00	Sporadic Failure
15	0	80269	0	80288	16.05	Sporadic Failure
12	-	-	-	-	-	Failure

Table 14: GATT bearer performance with three devices, with Node 2 acting as DFU origin.

Msg TX Speed Applied(ms)	Msg Loss Cnt Node 2	Execution Time(ms) Node 2	Msg Loss Cnt Node 3	Execution Time(ms) Node 3	Effective Mean TX Speed(ms)	Note
25	0	129224	0	129250	25.84	-
20	0	104977	0	105052	21.00	Sporadic Failure
17	0	89889	0	89916	17.98	Sporadic Failure
15	0	80132	0	80158	16.03	Sporadic Failure
12	-	-	-	-	-	Failure

Table 15: GATT bearer performance with three devices, with Node 1 acting as DFU origin.

provide similar results concerning both message loss and execution time. This shows us that the amount of hops between the origin and the furthest device in the network has little impact on the total execution time of the DFU. The testing showed that it is possible to perform successful DFU emulations down to effective transmission rates of approximately 16.05 ms. In this instance, we are executing the entire DFU procedure in roughly half the time compared to the 30 ms ADV bearer baseline test, not losing a single message in the process. There is, however, a significant catch to this presented result.

The most crucial revelation the result provides is that the implementation starts to struggle as soon as we move past the 25 ms transmission rate. For the testing conducted with the 20 ms, 17 ms, and 15 ms transmission rates, I experienced that approximately $\frac{1}{3}$ of the tests performed failed similarly as the test for the 7 ms transmission rate in the GATT baseline setup. At a rate of 12 ms seconds, the setup was not able to complete a single test without the implementation crashing on one or several of the participating devices. Considering that I was able to conduct tests with TX rates as high as 8 ms in the baseline setup, these results indicate that maintaining several GATT connections has a substantial impact on the capacity of the devices in the network. An interesting observation is that the GATT bearer solution either executes the DFU emulation perfectly or fails.

Nevertheless, this setup was still able to provide consistent results for the 25 ms transmission rate. At this rate, the GATT bearer can still provide better throughput and data consistency compared to the ADV bearer baseline.

6.2.5 Four Devices GATT Bearer Performance

In this scenario, we have four devices participating in the DFU emulation. An overview of this setup can be viewed in figure 33. One of the devices must, in this instance, handle GATT communication over three links simultaneously. Promising results for this setup will show if the current implementation is capable of handling a network solution that is based on the tree configuration, as earlier discussed in section 4.6.3.2. Taking the observations from section 6.2.4 into account, we might expect to see a decreasing overall performance of this setup compared to both the GATT baseline testing and the three device setup. However, as long as the system can still improve data consistency and throughput, I will consider the result promising.

In this setup, I will start by applying transmission speeds similar to those used for the three-device test. If the system should fail under these rates, I will gradually decrease the transmission rates to find a rate where the implementation can handle the DFU emulation workload. For this test setup, I will run DFU emulations for a situation where node 1 acts as the origin and another for where node 2 acts as the origin.

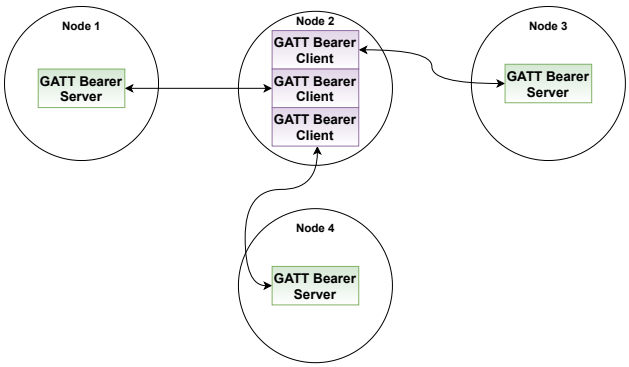


Figure 33: GATT bearer performance test setup with four devices.

The outcome of the four device testing did not provide the result that I was hoping for. It proved to be quite challenging to perform any successful DFU emulation at higher speeds without the system crashing. The testing was conducted for several reconfigurations where the different GATT roles were changed for node 2, but this effort did not provide any changes for the final result. After a failed completion of a test for transmission at a rate of 200 ms, I have concluded that the system must undergo a thorough investigation to see if it is possible to identify the reason for these failed performance tests.

I have not been able to identify the exact reason why the implementation crashes for higher speeds in the four device configuration. However, it is clear that it is associated with the coexistence of several GATT connections simultaneously. One observation that I have made is that node 2 experiences a tremendous flow of messages it has to handle. I suspect that the reason for this is partially caused by the lack of an implemented routing control mechanism, as discussed in section 7.1.1. The consequence of the absence of this feature is that node 2 has to forward the combined set of DFU messages over all available GATT interfaces, even if one of these interfaces was the origin of the message. To get a complete overview of this circumstance, I will review a test that was partially completed for a DFU emulation with a transmission rate of 50 ms. In this instance, the origin node was node 1, passing messages through node 2, which then extended these messages to nodes 3 and 4. In an ideal situation, it would have sufficed that each message is passed by node 1 to node 2, which passes the message further to nodes 3 and 4. The observed behavior is that node 1 passes the message to node 2. When node 2 receives this message, it not only relays the message to nodes 3 and 4 but also back over the GATT interface to node 1. In addition, node 3 and 4 relays the message back to node 2 again, completing the entire flow cycle for a single message. If we consider that this flow will repeat for all messages in the DFU emulation, it is easy to see that the workload of node 2 is considerable. In this instance, it has to handle a situation where we want to simultaneously send and receive messages on all three GATT interfaces at the same time. It is not unlikely that these circumstances contribute to the system failures I am experiencing for this test setup. In a scenario where the routing mechanism is present, the total workload would have been reduced. In this situation, the system might have been able to handle the workload during the DFU emulation for higher transmission rates.

The conclusion for the four device testing is that the system cannot perform any high throughput-related work due to the instability observed under these conditions. A full evaluation of the system testing is discussed in section 7.4 of this thesis.

7 Discussion

7.1 Limitations of the Final Implementation

In the later stages of the implementation phase of this thesis, I realized that I would not be able to implement all desired features within the set time constraints of the assignment. After completing the GATT solution's core functionality, there are a couple of central features that are not yet accounted for.

7.1.1 Internal Routing

As described in section 4.3.3, there is no reason for a BTM node to relay an incoming message over the same GATT connection from where the message initially arrived since we can guarantee that this message already has been handled by the origin of the message. Sending the message over this link will be utterly redundant and should, therefore, ideally be avoided. In order to achieve this, there must exist some message entry for each connection interface at the GATT bearer layer. These entries must register any incoming message over their respective connection, using a tag that must follow the incoming message when it is passed to the network layer. If the network layer decides that this message qualifies for relaying, it will be sent back to the GATT bearer layer. Here the associated message tag must be compared against the message entry list of each active GATT connection, ensuring that the message is only relayed on the interfaces where the tag is not present in the associated entry list. My initial idea was that these entry lists could be implemented like the message cache, where each entry buffer can hold a small set of tags. Whenever a new message arrives over the connection interface, the oldest existing in the entry will be discarded to make room for the new one. Since the time that expires between the arrival of an incoming message and an eventual relaying of the same message is small, I deem it sufficient to store a tag for only a short amount of time.

The consequence of the absence of this feature is that a device will relay messages over all available GATT interfaces, even the interface the message originally arrived over. This contributes to additional strain on each device that utilizes the GATT bearer, using a significant portion of the network combined resources on unnecessary and redundant transmissions. This shortcoming will likely have a negative impact on the overall network performance. Admitting this shortcoming, it is clear that an eventual continuation of this thesis work should strive to provide this desired routing feature.

7.1.2 Flow Control

A flow control feature for the GATT bearer implementation is a subject that has been recapitulated several times throughout the thesis work. At the simulation stage(4.3.3), I was under the impression that flow control could be easily provided by inherit GATT functionality. In section 7.2.1 it is however revealed that this assumption was incorrect when taking the BTM specification into account.

The consequence of this realization was that an eventual implementation of flow control was likely to consume significantly more time than initially expected. Additionally, the quality of an eventual implementation was uncertain, as described in section 7.2.1. Based on these considerations, I decided to degrade the priority level of the flow control implementation. While the significance of flow control could be considerable in an ideal GATT bearer solution, the time it would require to implement it could not be justified within the time limitations of this thesis.

The risk of lacking flow control was initially discussed in the simulation chapter of this thesis(4). An implementation that lacks flow control might experience that message buffers overflows for higher transmission rates in the network, potentially causing data loss. Like for the routing feature,

future work should aim to provide flow control for the GATT bearer solution since it is expected to contribute to the network's overall performance.

7.2 Changing the BTM Specification

As stated in the introduction to this thesis, one of my main goals for the GATT bearer implementation software development was to strive to use implementation choices that heed the BTM specification. I set this goal because I recognize that an eventual request to change the specification would involve a comprehensive, time-consuming process with an uncertain outcome. Failing to implement a solution that agrees with the specification would devalue it since the interoperability between BTM devices manufactured by different vendors is crucial for the market value of these products.

To the best of my knowledge, the final implementation for this thesis complies with the BTM specification. In order to achieve this, I have been forced to make certain sacrifices concerning performance and implementation quality to heed the specification. Nevertheless, since I had to use a vendor-specific model to control the implementation, I fear that sticking to the specification did not have the completely intended effect on the final result. While vendor models are within the confines of the specification, they are not in any way mandatory in an arbitrary BTM product. Since this alternative bearer solution is dependent on functionality that may be considered optional from a specification point of view, the commercial value is likely to be significantly less than a solution where all utilized functionality is mandatory. However, the need for the GPC model was identified already at the preliminary research stage of this thesis(3.3.3), and I can now state that the implementation would have been unattainable without it.

In this section, I will discuss topics regarding the specification. I will discuss how it has impacted my work, its potential to provide the functionality I needed, and how the specification could be changed to enable a simpler and better implementation solution.

7.2.1 Transmission of Data With the Proxy Service

Section 4.3.3 describes how the introduction of BLE connections between devices had the potential to offer inherent opportunities for flow control in the network. The results of the simulation chapter showed that the GATT bearer solution encounters buffering issues when running close to the maximum capable throughput rate. The solution to this problem was solved in the simulation model by introducing flow control functionality that ensured that the data flow in the network would halt if a device's buffer were full, thus preventing losing messages during the DFU.

When the simulation work was conducted, I was under the impression that the flow control functionality could be provided directly from the BLE core specification. Since the general communication in GATT is performed over an established link, every message transmission is acknowledged by the receiver. This enables the possibility for a receiving device in transmission to respond with an error code if the receiving device's buffer was full. Whenever the transmitting device receives this error code, it will know that the target device cannot handle any new messages at the moment, making the transmitting device responsible for holding on to the message and retry transmission at a later time. This would offload the buffering strain to the transmitting device, which in turn might reject incoming messages from another device if its buffer is filled. In the DFU scenario used in the simulation chapter (4.6), the continuation of buffers filling in the network would eventually propagate back to the origin of the DFU. This would make it stall further message transmissions until the network has processed and passed a portion of the messages that are filling the buffers.

An issue with this initial plan was discovered during the development phase of the proxy client module. The BTM profile specification defines the behavior of the Mesh Proxy Data In and Out characteristics. It states:

> «When the client sends a Proxy PDU to the server by executing a GATT Write Without

> Response procedure on this characteristic, the Attribute Value field of the ATT Write Command packet contains the Proxy PDU.
>
> ...
>
> The server can send a Proxy PDU to the client by executing a GATT Notification procedure on this characteristic. The Attribute Value field of the ATT Handle Value Notification packet contains the Proxy PDU.»
>
> **– BTM Profile Specification[10, p. 279]** .

While all messages sent over a GATT connection are acknowledged, certain behavioral differences depend on the utilized property. As described in section 2.1.3.2, the *Write Without Response* and *Notify* property are unacknowledged properties. They still perform acknowledgment on the radio level but do not propagate these events to the higher levels of the stack. Therefore, using these properties to convey messages will make it impractical to introduce flow control using the initial suggested approach.

Changing these properties to *Write* and *Indicate* would solve the issue. Both of these properties allow the acknowledgment message to be propagated to the GATT level of the stack. This is, however, a clear violation of the specification. Changing these properties in the specification would require substantial alteration to any existing implementation of the BTM proxy module.

The *Write Without Response* and *Notify* properties are, as of now, the mandatory properties required by the Mesh proxy service. Just swapping these to the *Write* and *Indicate* property would most likely not be possible. The reason for this is that such a solution would not be backward compatible with previous revisions of the specification, meaning that existing BTM products using a previous revision for the proxy modules would not be able to communicate over GATT with the new revision. A specification revision that requires software updates for all existing BTM products is not feasible. A more viable alternative would be to include these new properties as optional to their respective characteristics. Such a solution might be used for a revision that would be backward compatible with existing BTM products. In this way, the user could choose the new way of communicating over GATT whenever the hardware allows it and resort to the legacy solution whenever older devices are present in the network.

Other approaches might provide the desired flow control functionality. One way to solve this could be achieved by using the GPC to monitor the buffer's state in the GATT bearer and issue a message to the sounding devices whenever this buffer is full. This solution is, however, not as prominent as the original idea. Such a solution would not act directly in response to the ongoing transmission, meaning that a transmission conducted while the receiver's buffer is full potentially would lead to losing messages. When the buffer gets full, the GPC model needs to detect that the buffer is full and then issue the message telling connected devices to halt the message flow. The connected devices must then receive these messages before halting the message flow to the device in question. This procedure will consume a certain amount of time, in which it may have received several additional messages that might have been discarded due to insufficient buffer space. Another potential issue is if one or several connected devices simultaneously are experiencing full buffers. In this instance, a situation can occur where the initial flow control message is lost by a connected device where the buffer is also full, meaning that it will continue to forward messages to the origin device. A solution to prevent this situation is to let the devices issue the message when the buffer is nearly full. This would provide a buffer within the buffer that gives the devices a small window of time to issue the *buffer full* message before it is full. The downside of this approach is that it will cause a poorer message buffer utilization.

As I have shown, it is possible to provide flow control for the system without breaking/changing the BTM specification. However, such a solution will most likely be of poorer quality than a solution that could have been provided by using acknowledged GATT properties.

7.2.2 GPC Model

The tasks of GPC the model can be divided into three categories of responsibility:

- Mapping of the network.
- Establishing and maintaining connections.
- Control the ADV bearer.

Here I will go through each of these categories and evaluate what functionality the BTM specification already can provide and what alterations it needs to undergo to provide the rest of the required functionality.

7.2.2.1 Mapping of the Network

The mapping of the connection links in the network is a task that can be performed solely using existing functionality provided by the BTM profile specification. Prior to this thesis, I have implemented a system that performs similar results as the mapping functionality in the GPC model. This work has been conducted for Nordic Semiconductor ASA while employed as a part-time student. This implementation was based on the functionality of the foundation configuration models of BTM, more precisely, the heartbeat feature of this model.

According to the BTM profile specification[10, p. 90], the heartbeat feature is originally intended for two purposes; to see if a node is still active in the network and to tell how far apart (hops) nodes are from each other. This suggests that the heartbeat feature originally is not intended for mapping the links in a network. It is, however, possible to tweak the feature to provide this information. Like other BTM messages, the heartbeat message contains a TTL value that defines how far it can travel. Setting this parameter to zero ensures that a message may not travel farther than one hop from the origin. By sending a known number of messages periodically from a transmitting device, we can see how many of these messages arrive at the surrounding devices and then assess the link quality based on the received messages.

While this approach produces comparable data as the implementation of the GPC module, it comes with a couple of unavoidable drawbacks. The specification states that the minimum period that must elapse between two heartbeat messages from a single device must be one second [10, p. 149]. Additionally, a device can only process incoming heartbeats from a single device at a time. This means that the mapping process must be conducted iteratively. A single iteration consists of a single device publishing heartbeats while the rest of the devices listen for them. Both these conditions cause this approach to be considerably more time-consuming than the mapping implementation of the GPC model. To obtain a sufficient set of data, we must send a certain amount of heartbeats from the origin device. The time it takes to issue these messages for a single iteration is defined by $Iter_T = N * 1\,\text{s}$, where N is the number of heartbeats issued. The total execution time, not accounting for the overhead associated with setup between each iteration, is defined by $Tot_T = M * Iter_T$, where M denotes the number of devices participating in the network. To exemplify the impact this has on the execution time we can imagine a network of 100 nodes, where we want to assess the links based on 50 heartbeat messages. This setup would result in an execution time of $5000\,\text{s} \approx 1.4\,\text{h}$, not including overhead.

I deem it reasonable to state that this execution time is far from ideal. The GPC model can perform a similar mapping in a fraction of this time because it can parallelize the process and has no lower restrictions on the period that must pass between issued messages. Reviewing the development process, I think it was wiser to utilize the vendor model implementation for the mapping functionality. It enabled me to develop and test the mapping swiftly, making it possible to complete this work in a reasonable amount of time.

If I were to suggest an alteration to the BTM specification regarding the mapping functionality, I would have proposed updating the existing heartbeat feature to rectify the shortcomings I have discussed in this section. This would make it able to offer an inherent mapping feature that can execute within a reasonable amount of time. I believe that this alteration should be possible to apply without major alterations to the specification. In my opinion, this is a feature that is not

only relevant for the implementation of this thesis. It would also be quite useful for legacy BTM use-cases.

7.2.2.2 Establishing and maintaining connections

The task of connection establishment and maintenance is, as stated in section 3.3.3, not provided by any existing BTM functionality. In general, all BTM configuration tasks are handled by the configuration models. By reviewing the message specification of this model[10, p. 156], I found that there are no current messages that are associated with the proxy client functionality. I assume that the reason for this is that the need for GATT proxy connection control on a network level has not been a commonly requested feature until now. Still, the proxy feature has been present in the specification for several years and is a commonly used feature. This means that connections must be controlled in some manner. I believe that the most common way to control the proxy client for most existing implementations is done through an internal API of each device. This is because the GATT proxy feature has primarily been used to enable singular devices to access the mesh. The BTM profile spec confirms this:

> «The GATT bearer is provided to enable devices that are not capable of supporting the ADV bearer to participate in a mesh network.»
>
> **— BTM Profile Specification[10, p. 38].**

This shows us that the GATT bearer originally is not intended to provide primary communication in the network but is rather a supplement for devices that cannot communicate with the network using the ADV bearer.

I would suggest that all messages and their associated behavior described in section 5.3.2 should be included as a part of the configuration foundation models. This would accommodate generic network-level control over the GATT connections in the network.

7.2.2.3 Control of the ADV bearer

As for the connection establishment and maintenance functionality, there is currently no way to control the ADV bearer using existing BTM specification functionality.

As the implementation of this thesis has shown, control of the ADV bearer behavior is a vital feature to prevent unnecessary strain on the network when utilizing the GATT bearer solution. While it might be possible to let both bearers coexist, this will be completely redundant in the homogeneous GATT approach and should therefore be avoided.

These messages and their associated behavior should also be included as a part of the configuration foundation models to provide this functionality.

7.3 Heterogeneous vs Homogeneous Approach

The implementation of this thesis has strived to provide and support two different topology approaches for the GATT bearer solution, namely the heterogeneous and homogeneous approach. In this section, I will assess and discuss the merits and drawbacks of these two approaches, comparing them with each other. The details of these two approaches are described in section 3.4 of this paper.

7.3.1 Homogeneous Evaluation

The homogeneous approach is the simplest of the two approaches from an implementation point of view. In this instance, we utilize the GATT bearer exclusively, forming GATT connections between every participating device in the network. Every device under normal operation will know the devices it is connected to. This facilitates the implementation of both routing and flow control in the system. This is because all communication is performed by one bearer, and all links to other devices are known. Since all devices in the homogeneous approach utilize the GATT bearer, it is expected that the performance enhancement for throughput and data consistency will be in accordance with the results that are shown in section 6.2. This results in a significant performance boost compared to the capabilities of the ADV bearer.

A weakness that the homogeneous approach possesses is that it requires a substantial amount of initial configuration. This requirement can be expected to grow linearly with the number of devices in the network. This is because every single link that the network will utilize must be configured by the user, a procedure that will be time-consuming for larger networks. Additionally, any alteration, additions, or removals in an existing network would require reconfiguration. All in all, this approach implies a considerable amount of attention from the user that is maintaining the network. Another consideration is that the network potentially would require devices to maintain a higher amount of GATT connections on average than the heterogeneous approach requires. As discovered in section 6.2, the number of GATT connections a single device has to maintain has a negative impact on the throughput capabilities, potentially decreasing the overall performance of the network. In general, it should be avoidable to have singular devices that have to maintain more than three connections in any network(4.6.3.2), but keeping within these limitations requires significant evaluation of the network topology as the magnitude of BTM devices increases.

7.3.2 Heterogeneous Evaluation

Compared to the homogeneous approach, the heterogeneous approach provides a network solution that presumably will be easier to install, maintain and alter. The reason for this is that this approach only utilizes GATT connections between the relay nodes of the network, which implies that a smaller number of nodes needs to be configured to establish GATT connections at the initial setup. This might also affect the average number of GATT connections each device would need to maintain, decreasing the strain the device must endure under normal operations. Further, alteration such as adding, moving, or removing devices in the network is less likely to require substantial reconfiguration of the network, as long as the device in question is a standard node that communicates over the ADV bearer. The network will only require reconfiguration if the changes impact the relay topology in the network. This is the greatest benefit the heterogeneous approach has over the homogeneous approach, providing a solution that will require significantly less interaction from the user through the networks' lifetime.

Unlike the homogeneous approach, this implementation requires both bearers to convey the messages in the network. This makes the implementation more complex, making it more complicated to introduce features like routing and flow control. However, the most critical drawback of the heterogeneous approach is that a combination of both two bearers is likely to have an unfortunate impact on the overall network performance. This occurred to me while I was conducting the performance testing for this thesis. Even though the performance testing did not include any specific testing on the heterogeneous approach itself, the baseline tests' results alone are enough to support this claim. I will demonstrate this through an example:

Let us imagine that we are performing a DFU procedure, the example application that has been utilized for both the simulation and performance testing in this thesis. In section 6.2 we have established that the GATT bearer can provide a significantly higher throughput performance than the ADV bearer on a general basis. In a heterogeneous GATT network, the DFU messages will start from the origin node, flowing through the network's relay nodes over the GATT bearer. The message transmissions between relay nodes do not by themselves present any issues. The problem occurs when we consider that the relay nodes are required to forward all these messages

to the regular nodes surrounding them. This forwarding has to be performed with the ADV bearer, with the associated communication limitations that this entails. This will effectively create a bottleneck at each relay node. The incoming DFU messages from adjacent relay nodes arrive at a rate significantly faster than the rate the node itself can forward messages through the ADV bearer. After some time, the outgoing message buffer of the ADV bearer on the relay nodes will be full. At this point, the origin node will have to halt the train of messages, or the DFU procedure will crash. **The conclusion that can be drawn from this is that the network's throughput performance is defined by the weakest link in the chain, in this case, the ADV bearer.** This circumstance was not initially accounted for in the simulation model and is therefore addressed in section 7.5 which evaluates the design of the high-level simulation model. I have now shown that the heterogeneous approach most likely will not benefit from the same throughput performance enhancement expected for the homogeneous approach.

The question now is if the heterogeneous approach gains any improvement compared to a legacy BTM network. Fortunately, the benefits regarding data consistency performance are still present in the heterogeneous approach. By communicating over GATT, the relay nodes can provide a much higher probability of successful message delivery than relay nodes in a legacy ADV bearer network. Additionally, the likelihood of successfully delivering messages over the ADV bearer is also expected to increase to some degree in this scheme since a significant portion of the networks' communication is offloaded to a separate part of the BLE physical medium. However, it is the homogeneous approach that will provide the best performance regarding data consistency. This is because the heterogeneous approach is still dependent on the partial passing of network messages over the ADV bearer, which inevitably has a larger chance of losing messages than the GATT bearer.

7.3.3 Evaluation Summary

Throughout this evaluation, I have concluded that the homogeneous approach is the preferred choice in any system where share performance is emphasized. It is the approach that is likely to provide the best performance with regard to both throughput and data consistency. This comes at the cost of significant configuration requirements for larger networks. However, in reality, it is the only one of the two suggested approaches that can provide improved throughput performance.

While the heterogeneous approach will provide improved data consistency for a network compared to a legacy BTM network, the combination of both bearers prevents it from enhancing the throughput capabilities. Still, this approach could be of interest in networks where throughput performance is not paramount.

7.4 Assessment of the System Testing

The system testing in this thesis has been crucial to assess the final implementation quality. It has provided results on both a functional and performance-based level. Here I will evaluate the combined result of the system testing and discuss how the outcome correlates to the initially set goals for this thesis.

The functional testing in section 6.1 showed that all aspects of the functional behavior are working as intended for the implementation. The combined functionality showed that it is possible to form, configure, and maintain a BTM network partially or solely built on GATT connections between the participating devices. To the best of my knowledge, all implementation details heed the BTM specification, meaning that I have achieved the initial goal of an implementation that complies with the specification.

The functional testing alone has provided proof of concept for a BTM network solution based on GATT but does not guarantee performance improvement. In difference to the functional test results, the outcome of the performance testing is more complex and ambiguous. The initial baseline testing in section 6.2 has, to a large extent, confirmed the assumptions that were made

for the high-level modeling. Here we can see that the GATT bearer can provide a significant performance boost concerning both throughput and data consistency compared to the ADV bearer, at least on a single connection basis.

However, the performance testing has shown that the system starts to experience instability at higher TX rates as soon as more than one GATT connection is maintained by one device. This instability was already discovered for the three device testing and proved to make it infeasible to extract consistent performance results for the implementation when introducing a third GATT connection to a device(6.2.5). The full explanation for why the system starts to struggle under a higher number of GATT connections is not yet unveiled. It is a field that should be thoroughly examined in any continuation of this thesis work. However, certain measures can definitely lighten the workload of devices maintaining several GATT connections. As discussed in section 6.2.5, the introduction of the routing mechanism would significantly decrease the strain on the devices in the network. This measure in itself might not be sufficient to ensure complete stability in the implementation, but it is without a doubt a step in the right direction.

If I were to assess the value of the implementation based on the performance test results alone, I would claim that implementation as a whole is not ready for actual deployment at this point. The instability issues under testing for multiple GATT connections have been too severe to be ignored on that account. Nevertheless, the combined result has shown potential in a BTM network solution based on GATT connections. Every successfully conducted DFU emulation showed that the GATT bearer could provide near-perfect data consistency performance. In contrast, the ADV bearer lost a significant portion of the messages in every test case conducted. The GATT bearer also showed a much higher potential concerning throughput on a general basis. However, this assessment is much more uncertain whiøe the instability issues are present in the system. Still, I am confident that it is possible to provide a BTM network solution that can utilize GATT to enhance both throughput and data consistency. The testing conducted in section 6.2.4 showed that a device maintaining two GATT connections was able to perform consistently at a TX rate of 25 ms. Already at this rate, the GATT solution provided a throughput improvement of roughly *15.62%*, and a reduction in messages lost from *4.38%* to *0%*, compared to the 30 ms baseline test for the ADV bearer[5]. This result by itself shows a noteworthy improvement in throughput and a significant improvement in data consistency. If each device is capable of handling two GATT connections at this transmission rate, it should be possible to form a complete BTM network that utilizes the chain topology discussed in section 4.6.3.1 of the high-level modeling and simulation. Full-scale tests for this situation must, however, be performed in order to verify this statement.

Concerning the suggested tree structure from section 4.6.3.2, the current implementation has shown that it is not capable of supporting this network structure for higher transmission speeds. Since this was evaluated as the preferable topology solution compared to the chain structure, I must concede that it is a shortcoming of the implementation that this topology can not be supported. However, I believe that it should be possible to realize a solution capable of handling more than two GATT connections per device for higher transmission speeds. In addition to introducing a routing mechanism, several configurable parameters inside the BLE stack might help improve the coexistence of multiple GATT connections. However, exploring this will require more time than is permitted within the limitations of this thesis.

7.5 Assessment of the High Level Model

After completing the implementation and the associated system testing, I would like to review the quality of the high-level model and the simulation work for this thesis. The creation of the high-level model was done in the early stages of this work. It has acted as a guideline for many decisions made during the process. It was the results of these simulations that finalized the decision to use the GATT solution as the target approach for the implementation. At this stage, we can see if the results for the high-level simulation are comparable to the observed results that are provided by section 6. This enables me to evaluate the merit of the assumptions made for the

[5]These results are derived from table 11 and 14.

high-level model. Additionally, several realizations have been made during the implementation work that needs to be addressed concerning this process.

7.5.1 Model Weaknesses, Inaccuracies, and Misconceptions

Here I will present the weaknesses, inaccuracies, and misconceptions that I have discovered regarding the HL model throughout the thesis work:

7.5.1.1 Heterogeneous Throughput Modeling

One apparent misconception made for the HL model was that the simulation could assess the throughput capabilities for a heterogeneous network solution. As discussed in section 7.3.2, a network solution that utilizes both bearers will effectively restrict the throughput capabilities to the peak performance of the weakest bearer, which in this instance is the ADV bearer. This realization became clear to me at a late stage in the thesis work, making the initial assumption for the HL model invalid. The HL model can still provide valid results for data consistency, but concerning throughput performance, the model cannot assess this metric for a heterogeneous network solution.

7.5.1.2 Flow Control Assumption

Concerning the introduction of flow control, there was made an inaccurate assumption in the HL model. In section 4.3.3 I have stated that all GATT messages are acknowledged, an inherent feature that efficiently can be utilized to introduce flow control in the implementation. While this statement technically is true, section 7.2.1 reveals that the utilized GATT procedure for communication is crucial if we want to exploit this opportunity for flow control. Since the BTM specification does not permit the use of the required procedures to facilitate flow control, the assumption made was not entirely accurate. It is possible to introduce flow control for the implementation, but the assumption of how easily this could be provided in the implementation was not correct. In difference to the HL model, the actual implementation does not provide any means of regulating the message flow in the network at this point. This means that any buffer restrictions that were applied during simulation would not apply to a real scenario with the current implementation.

7.5.1.3 Impact of Multiple GATT Connections

As revealed by the performance testing conducted in section 6.2.5, the number of GATT connections a single device has to maintain has a significant impact on the throughput capabilities of the system. The HL model does not account for this, which is a weakness in the model. It would, however, require a large amount of effort to account for this aspect, which I deem would not have been feasible within the time limitations of the thesis work. This was a model limitation that I accepted already at the HL modeling stage. A measure that was taken during the simulation stage was to ensure that I did not conduct simulations for scenarios where an unreasonable amount of connections was maintained for a single device.

7.5.2 Predicted vs Observed Result

Two main aspects must be considered when we compare the simulation results to the actual performance results. The first is a comparison for data consistency, while the second is a comparison where we review throughput performance.

7.5.2.1 Data Consistency

For the simulation result, we have a situation where all performed simulations for the GATT bearer have provided a result where not a single message is lost during execution. By reviewing the performance test results in section 6.2, we can observe that all successfully conducted DFU emulation provided the same result, not losing a single message during transmission. This indicates that the HL modeling for the GATT bearer provides a realistic representation for simulation, at least concerning data consistency.

Assessment of the model for the ADV bearer is a little more complex. The reason for this is that the simulation environment assumes a constant noise level in the BTM devices soundings of *10%*, which is added to the internal noise created by the BTM network itself. In the simulations, the average chance of losing a message was roughly between *10%* and *13%*. The decision of choosing a constant noise level of *10%* was somewhat arbitrary since the main point of the simulation was to compare the performance between the two bearers in the same environment. Therefore, it is hard to compare the simulation results directly to the performance results since we do not know the expected loss chance in the performance setup for the ADV bearer. We must instead see if we can identify the same tendency in both results.

In table 1 we can see the result from a DFU simulation conducted for the ADV bearer using a single transmission per message. Node 4 in this simulation is the node adjacent to the origin that delivered the worst data consistency, with a message loss of *0.845%*. If we compare this message loss to any of the conducted performance tests for the ADV bearer in table 11, we can see that the simulation result is considerably better than any of these tests. In comparison, the best data consistency result of these tests was observed at a *2.06%* message loss. Keeping in mind that the performance test was conducted on a desktop setup, where the chance of message delivery should be near ideal, the assumed message loss chance for the simulation was perhaps somewhat optimistic. However, both the simulations and the performance testing show the same tendency, where the ADV bearer struggles to deliver all messages when we are not utilizing retransmission redundancy. With this in mind, I am satisfied with the data consistency predictions that the simulation model provided.

7.5.2.2 Throughput

The throughput modeling for both the ADV bearer and the GATT bearer was based on theoretical throughput capabilities derived from the BLE and BTM specifications. For all ADV bearer simulation the 25 ms peak TX rate was used, in accordance with the findings in section 2.2.4.1. While the ADV bearer baseline testing in section 6.2.2 showed that it is technically possible to run DFU emulation at this speed, the message loss at this rate is so high that it for all practical purposes is infeasible. The testing showed that the Zephyr BTM implementation at most could handle a TX rate of 30 ms without losing a considerable portion of the messages. In this way, the ADV bearer baseline test showed that the original assumption for the peak TX rate might have been too optimistic, using the mean advertising delay to find the peak rate. A better assessment might have been to use the worst-case advertising delay for the HL modeling, making the peak TX speed for the ADV bearer model 30 ms.

The TX rate that was used for the GATT bearer modeling was 20 ms. This was a cautious choice since the research of the BLE specification indicated that the rate could potentially be as low as 7.5 ms(4.3.3). Let us consider only the GATT bearer baseline testing first. In this context, it is clear that the set TX rate during simulation is well within the boundaries of the capabilities of the actual implementation. In table 12 we can see that a single connection was able to handle effective TX rates up to 9.1 ms, which is near the predicted rate of 7.5 ms. However, this assessment does not consider the instability issues experienced during tests for several GATT connections on a device. As discussed in section 7.4, the issues associated with multiple GATT connections have a negative impact on the throughput performance. This is a shortcoming of the model that is described in section 7.5.1.3.

7.5.3 Summary

To summarize the quality of the HL model behavior, I will start by admitting that it has some weaknesses. This is mainly associated with the instability issues experienced under the performance testing, which is an issue that is not yet completely explored. Before this issue is investigated thoroughly, it is hard to assess the HL model's quality fully. Still, I think that it has provided a reasonably accurate depiction of the relative performance between the GATT and the ADV bearer.

8 Conclusion

The goal of this thesis has been to find an alternative network solution for BTM. This solution has aimed to rectify the shortcomings that the standard currently inhabits concerning throughput and data consistency performance. A secondary goal has been to create this solution within the confines of the BTM specification. Here I will assess and draw a final conclusion for to what extent the suggested GATT solution has achieved the thesis goals.

8.1 Performance Results

Throughout this thesis, I have shown that it is possible to create a network solution for BTM that partially or fully utilizes GATT connections for communication between devices. Through modeling, simulation, and testing, I have shown that a solution based on GATT has considerable potential for enhancing network performance for both data consistency and throughput.

The observed results from both simulations and performance testing have shown that the GATT bearer can deliver almost ideal performance when it comes to data consistency, not losing a single message in any simulation or successfully performed test scenario. This result is achieved without compromising the throughput performance of the system. The simulation work has shown that the ADV bearer depends on multiple retransmissions to provide similar results as the GATT bearer, which drastically reduces the throughput performance. With this in mind, it is clear that the implementation of this thesis is capable of providing significantly better data consistency than the legacy BTM implementation. Improving the data consistency performance metric has been a central goal of this thesis. In this regard, I am confident that I have provided an alternative BTM implementation that has achieved this goal.

The final implementation has also shown promising results when it comes to throughput performance. An assessment of this implementation quality is, however, more complex than for the data consistency performance. The reason for this is because of the instability issues that were experienced under the performance testing.

As discussed in section 7.4, higher TX rates for test scenarios with multiple GATT connections on a single device caused the system to fail, either sporadic or consistently depending on the number of GATT connections maintained. This is an issue that has not been fully investigated or resolved at this time. The testing for two GATT connections on a single device did, however, provide consistent results for a TX rate of 25 ms, a rate that provides better throughput performance than was observed for the ADV bearer testing in section 6.2.2. A configuration that can support two GATT connections will be sufficient to create a chain network as described in section 4.6.3.1. This means that it could be possible to achieve a complete BTM network that can provide better throughput performance than the ADV bearer, but this statement is uncertain until an actual full-scale test is performed.

Further, the comparison between the baseline testing for both bearers showed that the GATT bearer, in general, can perform at a significantly higher TX rate than the ADV bearer. Suppose it is possible to solve the issues related to maintaining several GATT connections. In that case,

I am confident that the GATT bearer solution can provide a substantial boost in throughput performance. On the background of this evaluation, I have concluded that the results regarding throughput performance are promising but inconclusive. More research and development are needed to either verify or reject the idea of utilizing GATT to improve throughput performance in BTM. I am, however, cautiously confident that further work will prove the merit of the GATT network solution in this regard.

8.2 Implementation Evaluation

Apart from the performance results, the functional testing has shown that the current implementation can establish, configure, and maintain a GATT-based BTM network. The implementation is made so that the network can sustain itself after initial configuration, just like for the legacy BTM solution. I assess that the combined features of the implementation provide a complete interface that enables relative ease-of-use for a potential consumer of the product. Nevertheless, I am concerned that the current solution will not be eligible for the commercial market at this point. This statement is caused by the current state of the BTM specification.

The main reason it is infeasible to commercialize the implementation of this thesis is not caused by lacking coverage with the core GATT bearer functionality within the BTM specification but rather due to the absence of mechanisms to control and maintain a network consisting of GATT connections. Throughout this thesis, I have shown that it is possible to maneuver within the GATT bearer specification boundaries to provide a BTM network that can benefit from a larger portion of the combined resources that BLE offers. However, the issue occurs as soon as we take the network configuration into account. As I discovered early on, there is no way to manage GATT connections on a network level. As discussed in section 7.2.2, the current BTM specification does not only lack the essential functionality to establish GATT connections in the network. It also lacks proper coverage of important convenience features like the mapping tool and control of the ADV bearer behavior. It is now clear that the implementation would have been unattainable without adding the GPC vendor model. This model has provided near all control features that have made the implementation functional. Since vendor models are an established concept within the BTM, I can also claim that the use of this model is within the boundaries of the specification.

Nevertheless, using a vendor model makes this implementation significantly less appealing from a commercial point of view. Since there is no requirement for any BTM device manufacturer to add specific vendor models to their products, this solution breaks the concept of interoperability. Since interoperability is a cornerstone of the BTM specification, the implementation of this thesis is likely to end up as a novelty from a commercial point of view.

With this in mind, it is appropriate to review the initial goals set for this thesis. In section 1.2 I stated that the implementation of this thesis should strive to heed the BTM specification. This goal was set to avoid a situation where the implementation depended on alterations or amendments to the existing BTM specification. While I have provided an implementation that is technically within the specification boundaries, I have still concluded that the final solution is deficient from a commercial point of view. The commercial aspect was one of the main reasons for this goal in the first place. In pursuing this goal, I have been faced with several decisions throughout the implementation work where performance considerations have yielded to benefit specification compliance. The most prominent example in this regard is the facilitation of flow control that is discussed in section 7.2.1. Deviating from the specification in this instance would have made it significantly easier to implement flow control and would most likely provide a better solution.

Another example is extended advertising, an option that was disregarded early on since the BTM specification did not cover it. Since the specification compliance goal did not provide a completely satisfactory result, it would have been interesting to see what solution could have been realized if performance was the only emphasized metric. Most of the improvements mentioned above would have been possible by changing the specification compliance goal from the BTM specification to the BLE core specification, still providing a BLE-compatible solution.

However, the work of this thesis has shown the potential that a GATT bearer network solution inhabits. While the complete implementation is not a finalized product, it has proved the concept of creating a BTM network solution that utilizes the GATT bearer. In this way, the thesis has established a foundation that future work might expand on.

9 Future Work

9.1 Reduction in Power Consumption for BTM

A drawback of using the ADV bearer in BTM is that this entails almost constant monitoring of the scanner to ensure that a device can receive any incoming message:

> «A device supporting only the advertising bearer should perform passive scanning with a duty cycle as close to 100 percent as possible in order to avoid missing any incoming mesh messages or Provisioning PDUs.»
>
> **— BTM Profile Specification[10, p. 38].**

Running the radio module of a device is generally a power-consuming task, making the operation time of any battery-driven device relatively short compared with other devices that utilize conventional BLE operations. Even though BTM does provide a low power feature(2.2.1.3), it is commonly known that the standard, in general, is associated with high power consumption. Swapping the ADV bearer with a network solution that solely utilizes the GATT bearer could facilitate an implementation where it is possible to save considerable amounts of power in a BTM network. As the name implies, BLE is a standard that emphasizes low power consumption during operation. With the introduction of link-based communication, it could be possible to let the devices of a BTM network sleep for a large portion of the operation time, thus reducing the power consumption considerably.

While this concept has not been targeted in this thesis, it is potentially an addition to the list of performance metrics that can be improved by changing the bearer layer operations of BTM. Future work expanding on this thesis could examine if it could be possible to implement a BTM network where it is possible to operate with considerably lower power consumption.

9.2 Mapping and Configuration Algorithm

Section 5.3.1 and 5.4.3 describes the mapping functionality, and the mapping tool I developed during this thesis. These tools were crucial in providing data that enabled me to find a suitable connection topology between the devices in the network. However, selecting the topology in this instance had to be done manually, selecting each link before issuing the command to establish the connection. This task is cumbersome and time-consuming and also requires a decent amount of experience and know-how from the configurator.

A wast improvement to this scheme would be to develop and implement an algorithm that autonomously could execute the mapping sequence, find a connection configuration that allows communication between all devices before issuing the connection configuration to the devices of the BTM network. Depending on the use case and requirements of the network, the algorithm may be set to find a solution that fulfills a minimum requirement. For BTM networks that are not required to operate at peak performance, it would be sufficient for the algorithm to find and provide one of several possible configuration schemes that fulfill the set requirements. This approach might result in a less computationally heavy algorithm than an algorithm that always opts for the ideal configuration solution, making the demand on hardware resources and computation time significantly less. It might even be possible to make the algorithm so lightweight that it could be implemented directly in BTM devices, making the need for third-party device involvement

obsolete. Alternately, the algorithm could also be implemented on a smartphone and interact with the BTM network over a GATT connection.

An integrated solution for this functionality would be beneficial from a marketing point of view. An attractive feature of electronic products on the market is that they are implemented with plug-and-play principles, meaning that the product should require as little interaction with the user as possible to reach its operational state. Suppose a mapping and configuration algorithm could be integrated directly into a participating device of the network. In that case, the consumer might utilize a smartphone application to start the mapping sequence, and the device will take care of the rest.

Alternatively, if the implementation of a functioning lightweight algorithm should prove to be difficult, the algorithm could be implemented for a smartphone application and interact with the BTM network over a GATT connection. The common smartphones of today are equipped with hardware that is capable of handling a much larger workload than the hardware that a traditional BTM device can handle. By following this approach, we would offload the computationally heavy tasks of the algorithm from the BTM network.

Interestingly enough, the benefits of a mapping and configuration algorithm might also prove to be useful for legacy BTM networks. As described in section 2.2.1.2, a traditional BTM network mainly utilizes the ADV bearer, which does not in itself require any particular configuration to communicate with other devices. However, there are several configuration features and parameters that affect the performance of these networks as well. The main configurable feature is the relay node feature. The relay nodes are responsible for propagating messages throughout the network, with a similar role as the proxy client module inhabits in the implementation of this thesis. The selection of the number and distribution of relay nodes is crucial for the network to operate properly. Too few relay nodes might result in messages not reaching their destination, while too many may cause a higher degree of internal noise which will stress the network. Another configurable parameter is the retransmission count, which has similar concerns as for the selection of relay nodes.

When configuring a legacy BTM network, the goal is to find a configuration that balances the network's need for redundancy to ensure message delivery while simultaneously preventing that the network produces so much traffic that it starts to experience problems with internal noise. Just like for the GATT bearer solution, this could be provided by a mapping and configuration algorithm. It could be built on the same principle of collecting data for the links in the network and produce a sufficient configuration based on this data. The complexity of such an algorithm could vary, spanning from a simple algorithm for picking the right relay nodes to an algorithm that accounts for the relay nodes, retransmission count, and even TTL values of different models on each device.

Bibliography

[1] Mohammad Afaneh. *Bluetooth Low Energy: A Primer*. Retrieved: 2021-03-16. URL: https://interrupt.memfault.com/blog/bluetooth-low-energy-a-primer.

[2] Nordic Semiconductor ASA. *About the nRF Connect SDK*. Retrieved: 2021-05-12. URL: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/introduction.html#.

[3] Nordic Semiconductor ASA. *Bluetooth mesh – A scalable mesh technology*. Retrieved: 2020-10-07. URL: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/Bluetooth-mesh.

[4] Nordic Semiconductor ASA. *nRF Mesh*. Retrieved: 2021-03-25. URL: https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Mesh/GetStarted.

[5] Nordic Semiconductor ASA. *nRF52 DK*. Retrieved: 2021-05-10. URL: https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52-DK.

[6] K. Townsend - C. Cufi - Akiba - R. Davidson. *Getting Started with Bluetooth Low Energy*. O'Reilly, 2014.

[7] NetworkX developers. *NetworkX – Network Analysis in Python*. Retrieved: 2021-05-12. URL: https://networkx.org/.

[8] Bluetooth Special Interest Group. *Bluetooth Core Specification*. Retrieved: 2021-03-16. URL: https://www.bluetooth.com/specifications/bluetooth-core-specification/.

[9] Bluetooth Special Interest Group. *Mesh Model Specification 1.0.1*. Retrieved: 2021-03-16. URL: https://www.bluetooth.com/specifications/mesh-specifications/.

[10] Bluetooth Special Interest Group. *Mesh Profile Specification 1.0.1*. Retrieved: 2021-03-16. URL: https://www.bluetooth.com/specifications/mesh-specifications/.

[11] NovelBits. *Bluetooth 5 Advertisements: Everything you need to know*. Retrieved: 2021-05-01. URL: https://www.novelbits.io/bluetooth-5-advertisements/.

[12] Zephyr Project. *Zephyr Project Documentation – Introduction*. Retrieved: 2021-05-12. URL: https://docs.zephyrproject.org/latest/introduction/index.html.

[13] Zephyr Project. *Zephyr Project Documentation – Shell*. Retrieved: 2021-04-21. URL: https://docs.zephyrproject.org/latest/reference/shell/index.html.

[14] Saleae. *Saleae Technical Specification*. Retrieved: 2021-05-10. URL: https://www.saleae.com/.

[15] A. Storrø. *Bridging Home Automation and Bluetooth Mesh for Nordic Devices*. NTNU, 2020.

[16] Nordic Semiconductor ASA Trond Einar Snekvik. *Inteview – Trond Einar Snekvik*. Retrieved: 2021-05-05.

[17] Wikipedia. *Tera Term*. Retrieved: 2021-04-21. URL: https://en.wikipedia.org/wiki/Tera_Term.

A NCS Source Code

The final code from the development in this thesis can be found on GitHub.

For the GPC model implementation and application code, the source code can be found under the following link:

https://github.com/Andrewpini/sdk-nrf/tree/anders_master_nrf_rev2

As a guideline for finding the relevant code, any reviewer can look at the following source files.

- include/bluetooth/mesh:

  - gpc.h — Common header file for the GPC model *(new)*
  - gpc_srv.h — Header file for the GPC server *(new)*
  - gpc_cli.h — Header file for the GPC client *(new)*
- subsys/bluetooth/mesh:
  - gpc_srv.c — Source file for the GPC server *(new)*
  - gpc_cli.c — Source file for the GPC client *(new)*
- samples/bluetooth/mesh:
  - gatt_cfg — Folder for the GATT solution configuration tool application *(new)*
  - proxy_cfg_srv — Folder for the GATT solution device application *(new)*

For the BTM stack implementation and modification, the source code can be found under the following link:

https://github.com/Andrewpini/sdk-zephyr/tree/anders_master_rev2

As a guideline for finding the relevant code, any reviewer can look at the following source files in the subfolder **subsys/bluetooth/mesh**:

- proxy_client.h — Header file for the proxy client *(new)*
- proxy_client.c — Source file for the proxy client *(new)*
- net.c — Source file for the network layer module *(modified)*
- beacon.c — Source file for the beacon module *(modified)*
- adv.c — Source file for the adverting bearer *(modified)*

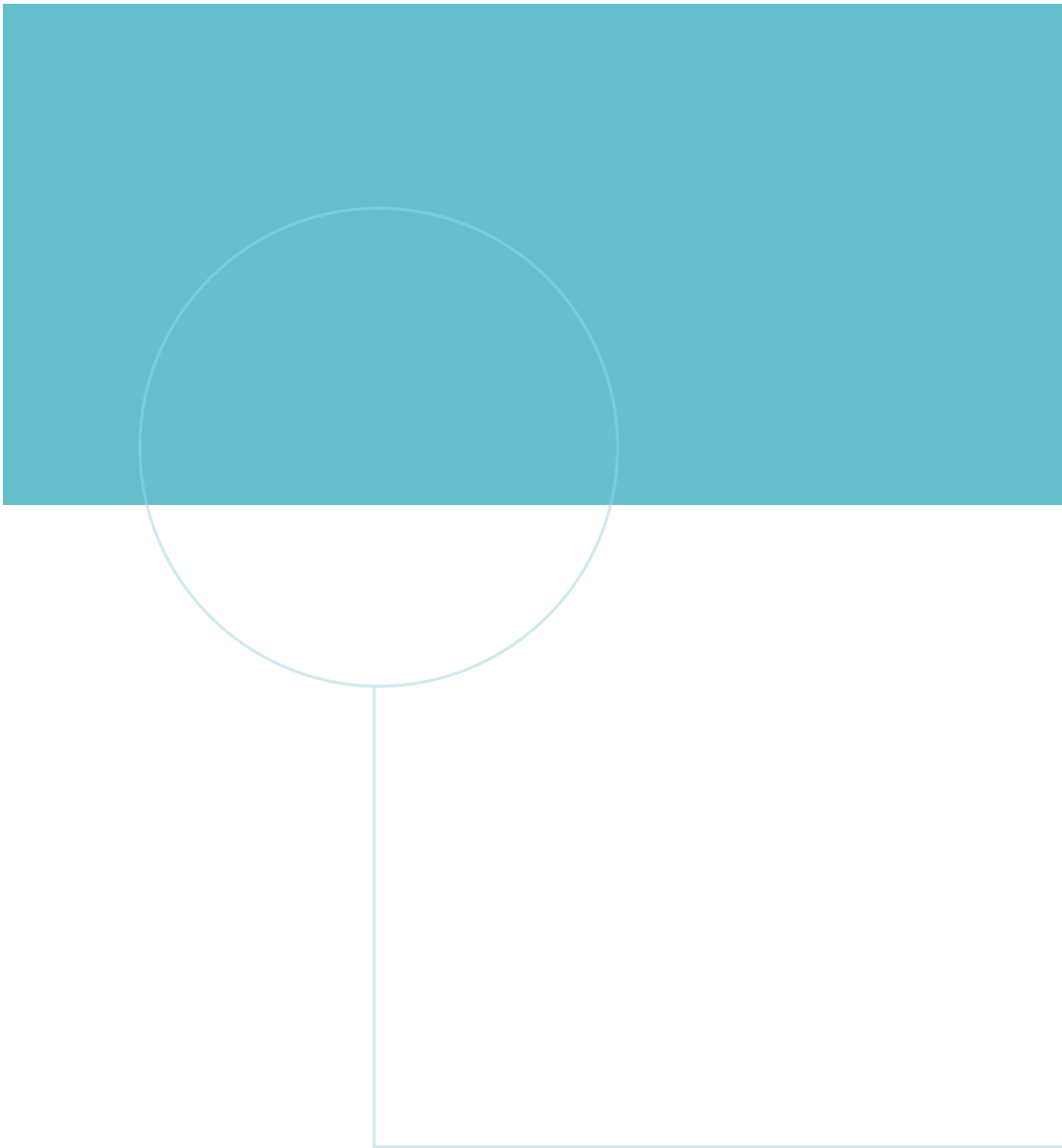
For any questions regarding this code, please refer the contact information provided on the front page of this paper.

B High Level Model Source Code

The high level model source code from the development in this thesis can be found on GitHub under the following link:

https://github.com/Andrewpini/mesh_hl_sim

For any questions regarding this code, please refer the contact information provided on the front page of this paper.



NTNU

Norwegian University of
Science and Technology