UNIVERSITY OF CASTILLA-LA MANCHA

Computing Systems Department

Improvements to Bluetooth Low Energy and Bluetooth mesh towards the new Generation of IoT-based Heterogeneous Networks

A dissertation for the degree of Doctor of Philosophy in Computer Science to be presented with due permission of the Computing Systems Department, for public examination and debate.

Author: Diego Hortelano Haro
Advisors: Dr. Teresa Olivares Montes
Dr. M. Carmen Ruiz Delgado

Albacete, December 2020

UNIVERSIDAD DE CASTILLA-LA MANCHA

Departamento de Sistemas Informáticos

Improvements to Bluetooth Low Energy and Bluetooth mesh towards the new Generation of IoT-based Heterogeneous Networks

Tesis Doctoral presentada al Departamento de Sistemas Informáticos de la Universidad de Castilla-La Mancha para la obtención del título de Doctor en Tecnologías Informáticas Avanzadas.

Autor: Diego Hortelano Haro
Directores: Dr. Teresa Olivares Montes
Dr. M. Carmen Ruiz Delgado

Albacete, diciembre de 2020

A todos mis seres queridos,
vosotros habéis hecho esto posible.

Acknowledgments

The completion of this Doctoral Thesis represents the end of a stage, and therefore the beginning of another. During this time I have not only learned and grown as a researcher, but also as a person, and I owe all this to the people I was lucky to share this stage with. Without knowing what the future holds, they have made it difficult to imagine anything being better than my experience with them. Therefore, I would like to dedicate at least these few lines to them.

I must necessarily begin by mentioning my advisors, Teresa and M. Carmen. I would like to thank them for their invaluable help, as well as for their support and dedication, finding time whenever I needed them. I would also like to thank them for their advice, given with their inestimable voice of experience. Without them, this stage would not have been the same. I am sure I will miss our weekly meetings.

I would also like to thank the remaining lecturers at the Faculty of Computer Science Engineering, both for providing the basis for completing this Doctoral Thesis and for their approachability, always available for help when needed.

I would also like to thank all the people of the High-Performance Networks and Architectures group, extending my thanks to the entire Albacete Research Institute of Informatics. My colleagues during these years, you have greatly facilitated this task with your help and support, even after your stay here was finished. I will not forget the coffees together, nor the after-hours trips from the laboratory before a deadline.

I would like to thank Marco Zimmerling, who was responsible for my research stay in Dresden, for his predisposition and help before my arrival. I would also like to include his entire team, who welcomed me from the first day and from whom I learned much more than network synchronization.

I would also like to thank enormously all my friends, although I consider that many of those mentioned above are also included in that category. Thank you for putting up with me all this time, for your interest, your encouragement and your support throughout this stage.

I would also like to thank my family, especially my parents and my brother, but also the rest of the family. Thank you for putting up with me during this stage, which I know

has not always been easy. You have always been there to support me and help me with everything, not only in this stage, but since I can remember.

Finally, I would like to thank Alba for all her support. I know that only a few lines are never enough, but you have always given me the necessary determination when mine was weak, and you have been there for me. I would love to continue discovering the villages of Castilla-La Mancha with you.

This work was supported by the Castilla-La Mancha Community Council under the grant POII-2014-010P, by the Spanish Ministry of Economy and Competitiveness and the European Commission (MINECO/FEDER funds) under the grants TIN2015-66972-C5-2-R and TIN2015-65845-C3-2-R, by the Spanish Ministry of Science, Education and Universities, the European Regional Development Fund and the State Research Agency under the grant RTI2018-098156-B-C52, by the University of Castilla-La Mancha with the European Regional Development Fund under the grant 2019-GRIN-27060 and also by the University of Castilla-La Mancha R&D plan with the European Social Fund.

Agradecimientos

La finalización de esta Tesis Doctoral supone el final de una etapa, y por lo tanto, el comienzo de otra. Durante este tiempo no solo me he formado y he crecido como investigador, si no también como persona, y todo ello se lo debo a las personas con las que tenido la suerte de compartir esta etapa. Ellas han hecho que, sin tener conocimiento de qué deparará el futuro, me cueste mucho imaginar que pueda superar lo vivido con ellas. Me gustaría por ello dedicar al menos estas líneas.

Se me hace imposible no comenzar estas líneas con mis directoras de Tesis, Teresa y M. Carmen . Me gustaría agradecerles su inestimable ayuda, así como su apoyo y dedicación, sacando tiempo siempre que me ha hecho falta. Me gustaría agradecer igualmente sus consejos, provenientes de esa voz de la experiencia de incalculable valor. Sin ellas esta etapa no habría sido la misma. Estoy seguro de que voy a echar de menos esas reuniones semanales.

Me gustaría agradecer también al resto de profesores de la Escuela Superior de Ingeniería Informática, tanto por aportar los cimientos que han permitido completar esta Tesis Doctoral como por su cercanía, dispuestos siempre a ayudar cuando se les necesita.

Así mismo, me gustaría dar las gracias a toda la gente del Grupo de Investigación de Redes y Arquitecturas de Altas Prestaciones (RAAP), extendiendo mi agradecimiento al Instituto de Investigación en Informática de Albacete completo. Compañeros durante estos años, me habéis facilitado enormemente esta tarea con vuestra ayuda y apoyo, incluso tras finalizar vuestra estancia aquí. No olvidaré los cafés compartidos, ni las salidas del laboratorio a las tantas antes de un deadline.

Me gustaría recordar en estas líneas al responsable de mi estancia de investigación en Dresde, Marco Zimmerling, por su predisposición y ayuda desde antes de mi llegada. Me gustaría incluir también a todo su equipo, que me acogió desde el primer día y de los que aprendí mucho más que sincronización de redes.

También me gustaría agradecer enormemente a todos mis amigos, aunque considero que muchos de los mencionados anteriores se encuentran también aquí incluidos. Gracias por aguantarme todo este tiempo, por vuestro interés, vuestro ánimo y vuestro apoyo durante toda esta etapa.

Quiero agradecer también en estas líneas a mi familia, especialmente a mis padres y a mi hermano, pero también al resto de la misma. Gracias por aguantarme durante esta etapa, que sé que no siempre ha sido fácil. Siempre os ha faltado tiempo cuando he necesitado algo, y siempre habéis estado ahí para apoyarme y ayudarme con todo, no solamente en esta etapa, si no desde que tengo uso de razón.

Por último, me gustaría agradecer todo su apoyo a Alba. Sé que en ningún caso unas pocas líneas son suficientes, pero siempre me has aportado la determinación necesaria cuando la mía flaqueaba, y has estado ahí para mí. Me encantaría seguir descubriendo los pueblos de Castilla-La Mancha junto a ti.

Este trabajo ha sido cofinanciado por la Junta de Comunidades de Castilla-La Mancha bajo el proyecto POII-2014-010P, por el Ministerio de Economía y Competitividad y la Comisión Europea (fondos MINECO/FEDER) bajo los proyectos TIN2015-66972-C5-2-R y TIN2015-65845-C3-2-R, por el Ministerio de Ciencia, Innovación y Universidades, el Fondo Europeo de Desarrollo Regional y la Agencia Estatal de Investigación bajo el proyecto RTI2018-098156-B-C52, por la Universidad de Castilla-La Mancha y el Fondo Europeo de Desarrollo Regional bajo el proyecto grant number 2019-GRIN-27060 y por el plan I+D de la Universidad de Castilla-La Mancha cofinanciado por el Fondo Social Europeo.

Summary

In recent years, the term Internet of Things (IoT) has been gaining momentum, moving from traditional sensor networks in home automation to large networks in different areas, such as smart buildings, smart cities or smart factories. The last of these is having a great impact, with different initiatives arising to facilitate its adaptation, in what is already known as Industry 4.0. The applications of these new trends bring with them new requirements that are challenging the current wireless communication networks. These requirements include transmission reliability (zero fails), total area coverage and sustainability, both in terms of network efficiency and network cost.

In this context, Bluetooth Low Energy (BLE) technology was developed, maintaining the classic Bluetooth objective of facilitating the connection between devices, but with a minimum power consumption. We opted for research into this particular technology because of the ease with which users can be included in the network, through their smartphone or wearable device, while providing low consumption, ideal for sensor devices. However, after an initial evaluation, we found that the topologies included in the specification did not fulfil the requirements demanded by the latest trends in IoT and Industry 4.0.

In order to use BLE technology in these new networks, we proposed a standard compliant mesh network using the broadcast capability of the devices. This network was deployed in a real environment, enabling communication between a wide range of devices, including sensor nodes, tablets, wearable devices and a BLE server. The evaluation carried out showed an excellent percentage of transmissions successfully completed, as well as total coverage of the area.

Due to the rising number of applications demanding a new topology and the large number of proposals from both academia and companies, the Bluetooth Special Interest Group (Bluetooth SIG) released a suite of specifications to include mesh topology in BLE. This specification is called Bluetooth mesh, and is built over the lower layers of BLE (the Link Layer and the Physical Layer). Bluetooth mesh uses a controlled flood routing through the broadcast capabilities of the BLE devices. The provisioning procedure stands out in the specification. This procedure enables the devices to receive the network key and other information required to take part in the network in a secure manner.

Following its release, we decided to focus on the Bluetooth mesh specification, as well as on the devices which used the early versions of BLE. The Bluetooth SIG advanced that

any BLE device capable of sending broadcast messages could be part of a Bluetooth mesh network. However, there were no studies on this, and the available implementations were built on the lower layers of the latest versions of BLE. Therefore, and in order to verify this compatibility, this Doctoral Thesis also includes our implementation of Bluetooth mesh, as well as the provisioning procedure, which is supported by many devices. The evaluation carried out with devices with the first versions of BLE showed their compatibility, although these early versions had more limitations than the later versions. Moreover, after the evaluation of the standard provisioning procedure, a lighter alternative is proposed, specially designed for use on devices with the earlier versions of BLE.

Finally, focusing on the Bluetooth mesh specification, it is remarkable for its great power consumption, since the devices need to constantly scan the network to receive the messages, which can be sent at any time. To address this problem, and to enable battery-powered devices to take part in the network, the Bluetooth mesh specification proposes a mechanism called friendship. Thus, one of the devices, called friend node, is constantly scanning the network, receiving and storing the messages sent to the low power node. These messages are sent to the low power node on demand, allowing it to scan the network only at short intervals. However, this friendship mechanism uses data transmissions similar to the stop-and-wait protocol, which is highly inefficient. For this reason, Burst Transmissions and Listen Before Transmit (BTLBT) technique was proposed, which minimises the consumption of these nodes.

Resumen

En los últimos años, el término Internet of Things (IoT) está teniendo un gran impacto, pasando de las tradicionales redes de sensores en domótica a grandes redes en diferentes áreas, como los edificios inteligentes, las ciudades inteligentes o la industria inteligente. Ésta última ha tenido un gran impacto, surgiendo diferentes iniciativas para facilitar su adaptación, en lo que ya se conoce como Industria 4.0. Las aplicaciones de estas nuevas tendencias traen consigo nuevos requisitos que desafían las redes de comunicación inalámbricas actuales. Entre estos requisitos destaca la necesidad de realizar transmisiones fiables, proporcionar una cobertura total del área y ser sostenibles, tanto en lo referente a la eficiencia de la red como a su coste.

En este contexto surge la tecnología Bluetooth Low Energy (BLE), que mantiene el objetivo del Bluetooth clásico de facilitar la conexión entre dispositivos, pero con un consumo mínimo. Nosotros nos decantamos por la investigación de esta tecnología en particular por su facilidad para incluir al usuario en la red, a través de su teléfono inteligente o dispositivo wearable, a la vez que proporciona un bajo consumo, ideal para los dispositivos sensores. Sin embargo, tras una evaluación inicial, comprobamos que las topologías incluidas en la especificación no permitían satisfacer los requisitos impuestos por las últimas tendencias en IoT y la Industria 4.0.

Con el fin de utilizar la tecnología BLE en estas nuevas redes, propusimos una red en malla que cumplía totalmente con el estándar utilizando la capacidad broadcast de los dispositivos. Esta red se desplegó en un entorno real, permitiendo la comunicación entre dispositivos muy variados, entre los que se incluyeron nodos sensores, tablets, dispositivos wearables y un servidor BLE. La evaluación realizada mostró un excelente porcentaje de transmisiones realizadas correctamente, así como una cobertura total del área.

Debido al creciente número de aplicaciones que demandaban una nueva topología y al gran número de propuestas realizadas tanto por la comunidad investigadora como por diferentes compañias, el Grupo de Especial Interés de Bluetooth (Bluetooth SIG) lanzó un conjunto de especificaciones para incluir la topología en malla en BLE. Esta especificación se denominó Bluetooth mesh, y está construida sobre las capas inferiores de BLE (la capa de enlace y la capa física). Bluetooth mesh utiliza un enrutamiento por inundación controlado a través de las capacidades broadcast de los dispositivos BLE. Destaca en la especificación el procedimiento de provisionamiento, que permite a los dispositivos recibir la clave de

red, así como el resto de información necesaria para formar parte de la red, de una manera segura.

Tras el lanzamiento de esta especificación, decidimos centrarnos en ella, así como en los dispositivos que utilizan las primeras versiones de BLE. A pesar de que el Bluetooth SIG anunció que cualquier dispositivo BLE con capacidad para enviar mensajes en broadcast podría formar parte de una red Bluetooth mesh, no existían estudios sobre esto, y las implementaciones disponibles se construían sobre las capas inferiores de las últimas versiones de BLE. Por ello, y con el fin de comprobar esta compatibilidad, esta Tesis Doctoral también incluye nuestra implementación de Bluetooth mesh, así como del procedimiento de provisionamiento, compatible con multitud de dispositivos. La evaluación realizada con dispositivos con las primeras versiones de BLE demostró su compatibilidad, aunque estas primeras versiones cuentan con más limitaciones que las versiones posteriores. Así mismo, tras la evaluación del procedimiento de provisionamiento estándar, se propone una alternativa más ligera, especialmente diseñada para su uso en dispositivos con las primeras versiones de BLE.

Finalmente, pasando a centrarnos en la especificación Bluetooth mesh, destaca su elevado consumo de energía, debido a que los dispositivos necesitan escanear constantemente la red para recibir los mensajes, que pueden ser enviados en cualquier momento. Ante este problema, y para permitir que dispositivos alimentados con baterias formen parte de la red, la especificación Bluetooth mesh propone un mecanismo denominado relación de amistad. Así uno de los dispositivos, el nodo amigo, está constantemente escaneando la red, recibiendo y almacenando los mensajes destinados al nodo de bajo consumo. Estos mensajes son enviados al nodo de bajo consumo bajo demanda, permitiendole escanear la red únicamente en momentos puntuales. Sin embargo, este mecanismo de amistad utiliza un intercambio de datos similar al protocolo de parada y espera, poco eficiente. Por ello, se propuso el uso de transmisiones a ráfagas con escucha antes de transmitir, lo que permitió minimizar el consumo de los nodos de bajo consumo.

Contents

List of Figures

List of Tables

List of Acronyms

ACK	Acknowledgement
ADV	Advertising
AES	Advanced Encryption Standard
AES-CCM	Advanced Encryption Standard - Counter with Cipher Block Chaining Message Authentication Code
AES-CMAC	Advanced Encryption Standard - Cipher Block Chaining Message Authentication Code
AFH	Adaptative Frequency Hopping
API	Application Programming Interface
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
BTLBT	Burst Transmissions and Listen Before Transmit
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
CTL	Network Control
DK	Development Kit
DST	Destination Address
ECDH	Elliptic Curve Diffie-Hellman
FCS	Frame Check Sequence
FN	Friend Node
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GFSK	Gaussian Frequency Shift Keying

GPCF	Generic Provisioning Control Format
HCI	Host Controller Interface
I3A	Albacete Research Institute of Informatics
ICT	Information and Communication Technologies
IIoT	Industrial Internet of Things
IoT	Internet of Things
ISM	Industrial, Scientific and Medical
IV	Initialisation Vector
IVI	Initialisation Vector Index
L2CAP	Logical Link Control and Adaptation Protocol
LBT	Listen Before Transmit
LL	Link Layer
LoRaWAN	Long Range Wide Area Network
LPN	Low Power Node
LTS	Lightweight Time Synchronization
MIC	Message Integrity Check
MTU	Maximum Transmission Unit
NID	Network Key Identifier
OOB	Out Of Band
PDU	Protocol Data Unit
PHY	Physical Layer
PN	Proxy Node
ppm	Parts per million
PRR	Packet Reception Rate,plural=PRRs
RF	Radio Frequency
RFU	Reserved for Future Use
RN	Relay Node
RSSI	Received Signal Strength Indicator
RTOS	Real Time Operating System

SDK	Software Development Kit
SEQ	Sequence Number
SIG	Special Interest Group
SM	Security Manager Protocol
SoC	System-on-Chip
SRC	Source Address
TTL	Time To Live
TX	Transmission
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier

CHAPTER 1

Introduction

This chapter firstly introduces the motivation behind this Doctoral Thesis. Secondly, the main objective of this Doctoral Thesis is presented. Thirdly, the methodology followed to achieve this objective is described. Finally, an outline of the dissertation is included.

1.1 Motivation and Justification

Internet of Things (IoT) is a new paradigm that combines features and technologies from different approaches (ubiquitous computing, embedded devices, sensor networks, Internet protocols and communication technologies). The main elements of IoT are smart objects. These are objects of daily use capable of collecting environmental information and interacting with or controlling the physical world. They can also be interconnected, to exchange data and information [1].

The field of Industry 4.0 is one of the most active in IoT [2]. Industry 4.0 [3] or Industrial IoT (IIoT) is the new trend emerging today, although it has a long way to go before its standards are established. This paradigm will enable the development and optimisation of industrial infrastructure with new manufacturing practices that use Information and Communication Technologies (ICT). However, the purpose of this trend is not only to introduce new technologies in industry, but also to connect and unify the different components of ICT in a network system. The innovations implemented in smart factories eliminate deficiencies, save costs and enable the automation of processes, while improving operator safety [4].

Today, we are surrounded by different types of devices and applications that collect large amounts of data. Of all these devices, smartphones are still the leaders. However, wearable devices are also enjoying unstoppable success. A problem arises, however, in this new scenario, since not all these new devices use the same communication protocols, so they are unable to work together. New communication standards, such as Bluetooth Low Energy (BLE), provide great possibilities for communicating sensors, clothing, smart

phones and smart objects with end users, allowing the use of network infrastructure to introduce or enhance a wide variety of emerging applications. Moreover, these communication protocols must take into account another key requirement in IoT: energy efficiency. BLE will become an important technology for IoT, due to its low power, low cost and small devices [5].

BLE was included for the first time in the Bluetooth 4.0 Core Specification [6]. Since then, it has been growing at extraordinary speed, due to its connection with smartphones, tablets, wearables and mobile computing in general, as well as its early and active adoption by mobile industry heavyweights [7]. This rapid development enabled BLE to be used in several IoT applications and projects. Some of these applications were: the Array of Things project of Chicago [8]; inter-vehicular communications [9]; power management in smart homes [5]; passengers control [10]; or remote lock system [11].

From the time BLE was first introduced in Bluetooth until its latest Bluetooth specification (version 5.2 [12]), it has maintained the same network topologies: point-to-point (1:1) and broadcast communications (1:m). However, its performance in terms of range, throughput, power consumption and payload capacity has been improved with each specification to deal with the new challenges arising. For example, broadcast topology was initially designed to send advertisements enabling a point-to-point connection. It was later used to send beacons, and now it enables the sending of data to multiple devices simultaneously. Meanwhile, point-to-point topology has increased the number of devices connected simultaneously to the same central node, and the combination of central/peripheral roles has become more flexible, enabling a peripheral device to be the central device of another node.

The topologies defined by the BLE specification can be applied to a multitude of applications, but they are unable to meet the requirements of the latest trends, such as smart cities, smart buildings or smart factories. These cases require great reliability, total coverage and sustainability, both in terms of energy consumption and the number of devices used.

In order to meet these new requirements, academia and companies made different proposals for the use of a new BLE network topology: the mesh network. Some of these proposals were BLEMesh [13], RT-BLE [14], CSRmesh [15] and the nRF OpenMesh [16]. Among the different proposals of BLE mesh networks was our proposal: the Collaborative Mesh, presented in depth in Chapter 4. Finally, the mesh topology was standardised by the Bluetooth Special Interest Group (SIG) with the release of a suite of specifications [17, 18, 19], providing many-to-many communications in large-scale device networks through a standard mesh topology.

Compared to the previous broadcast topology, the Bluetooth mesh specification not only provides a greater range, but also improves data capacity due to its extremely compact packet format [20]. In addition, it includes significant improvements in security capabilities at different levels. This has relegated broadcast communications to use in indoor location and beacon solutions (to provide point of interest information and item and path finding services). Point-to-point communications, the other network topology provided by the BLE

specification, has a completely different approach than the Bluetooth mesh topology. This topology is mainly designed for data transmission between connected devices in the same area (Body Area Network, BAN) [21].

The mesh network topology available in the BLE allows for the creation of large-scale networks. This converts this topology into the ideal candidate for use in IoT and IIoT applications [21], as lighting or smart warehousing solutions [22] (tracking or reporting data). The Bluetooth mesh specification defines a managed-flood-based mesh network, which uses the broadcast capability of BLE devices to transmit mesh messages. These messages are received and relayed by other nodes, extending the coverage range of the original device.

After the release of Bluetooth mesh, different devices especially designed for this topology were launched, as well as multiple implementations developed by companies such as Nordic Semiconductor [23] or Silicon Labs [24]. Other open-source implementations have been developed by communities (such as Zephyr [25] or BlueZ [26], the official Linux Bluetooth protocol stack), and the number of products qualified by the Bluetooth SIG and of Bluetooth mesh stack implementations is continuously increasing [27].

Despite the development of new devices and implementations designed for Bluetooth mesh, all of them focused on the latest versions of BLE, leaving devices with previous versions aside. The devices using the first versions of BLE were not designed for the Bluetooth mesh. However, the Bluetooth SIG claimed that all BLE devices could be part of the mesh network, since it is built on top of the lower layers of BLE, as detailed in Chapter 2. Could these devices be updated and used in heterogeneous Bluetooth mesh networks together with the new devices, or would the limitations of the first versions make their use unfeasible?

In order to answer this question, we developed and then evaluated an implementation of the Bluetooth mesh protocol stack in devices with the first version of BLE. This implementation and the subsequent evaluation are described in detail in Chapter 5, as well as our proposal to improve devices with earlier BLE versions.

Bluetooth mesh provides a solution that fulfils the requirements of the latest IoT trends. However, the mesh nodes must constantly scan the medium to relay the received messages and thus increase the total coverage range. This involves a high energy consumption that battery-powered nodes cannot withstand. For these cases, the Bluetooth mesh specification provides the friendship mechanism. This mechanism enables the nodes that are constantly scanning the network to store the messages destined to the low power nodes, allowing these nodes to be in sleep mode for most of their lifetime.

The friendship mechanism proposed by the Bluetooth mesh specification is necessary to avoid the exhaustion of batteries. However, it cannot currently be adjusted to the needs of specific applications. This may be related to the fact that it is the first version of the Bluetooth mesh specification and, given the novelty of the specification, the friendship mechanism is insufficiently optimised. The current standard friendship mechanism is based

on a stop and wait protocol and although it is simple to implement, it has very inefficient channel utilisation.

Chapter 6 of this dissertation addresses the proposal for an improvement of the standard friendship mechanism, in order to optimise the use of the channels and minimise energy consumption of the low power nodes.

1.2 Objectives

The main objective of this Doctoral Thesis was to design a heterogeneous BLE network, which allows the inclusion of any BLE device, and which fulfils the requirements of the new trends in IoT and Industry 4.0 in a sustainable way. The way in which this objective was addressed changed significantly during the progress of the Doctoral Thesis with the release of a new Bluetooth specification, the Bluetooth mesh specification, which was designed to take into account the requirements of the new applications. In order to deal with this main objective, the following goals were proposed:

- **Goal 1**. Review of the state of the art in the area of wireless networks, with special emphasis on BLE technology. This review focused on both the topologies included in the standard and the proposed topologies to improve the specification. Moreover, an in-depth study of the Bluetooth 4.0 specification (the first where BLE appears) was carried out. A review of the requirements of new applications for IoT and Industry 4.0 was also necessary, in order to be able to design and implement networks that fulfil them.
- **Goal 2**. Preliminary evaluation of both the topologies included in the BLE specification and the proposed topologies to improve the existing ones. In order to verify whether these topologies satisfy the requirements demanded by IoT and Industry 4.0, an evaluation with current hardware platforms in a real environment was required.
- **Goal 3**. Design, implementation and deployment of a BLE network with mesh topology that enables the integration of many different BLE devices, both static (beacons and nodes with sensors) and mobile (users through their smartphone or different wearables). Furthermore, this network had to satisfy the communication requirements imposed by the latest trends.
- **Goal 4**. Inclusion of different types of devices in our heterogeneous network. In addition to the sensor and relay nodes, a BLE server was included, which acted as a gateway between different technologies (BLE and Long Range Wide Area Network (LoRaWAN)). Moreover, in order to support the human-in-the-loop approach of Industry 4.0, special importance was given to the inclusion of user devices, with smartphones and wearables being the most outstanding.
- **Goal 5**. In-depth study of the Bluetooth mesh specification, including both the protocol stack and the provisioning procedure, which allows new devices to be added to

the network. Furthermore, a review of the state of the art regarding the Bluetooth mesh specification was necessary.

- **Goal 6**. Implementation and evaluation of the Bluetooth mesh protocol stack and the provisioning procedure in BLE 4.0 devices. Although the Bluetooth SIG claimed this was the first version to be supported, no previous studies were conducted using the earliest BLE versions.

- **Goal 7**. Design, implementation and evaluation of the Lightweight Provisioning procedure, our proposal to improve the provisioning procedure defined by the Bluetooth mesh specification.

- **Goal 8**. In-depth study of the friendship relationship proposed by the Bluetooth mesh specification. In addition, a review of the state of the art regarding this mechanism was necessary.

- **Goal 9**. Design, implementation and evaluation of our proposal for the optimisation of the friendship relationship, the Bluetooth mesh approach to reduce consumption in low power devices.

1.3 Methodology and Work Plan

In order to achieve the goals introduced in the previous section, a methodology and work plan needed to be defined. This methodology established the set of tasks, each of which was directly related to a goal, in order to achieve it. Each task was divided into sub-tasks, which are described below:

- **Task 1**. Review of the state of the art.
  - **Task 1.1**. To study the Bluetooth 4.0 specification, which includes the first version of BLE, in order to understand its functionality and limitations, focusing particularly on its lower layers: the Physical Layer and the Link Layer. Furthermore, it is necessary to have an in-depth understanding of the two topologies proposed by the standard: point-to-point connections and broadcast communications.
  - **Task 1.2**. To analyse the requirements demanded by the new IoT applications for wireless networks.
  - **Task 1.3**. To understand the new Industry 4.0 paradigm, in order to obtain a global image of the requirements of the new trends.
  - **Task 1.4**. To review the research work related to BLE, in order to know the different existing projects, as well as the strengths, weaknesses and challenges of this technology.

  - **Task 1.5**. To review research work related to proposals regarding different topologies than those included in the BLE specification.

- **Task 2**. Preliminary evaluation.
  - **Task 2.1**. To evaluate the hardware platforms used in the following experiments, in order to facilitate the understanding of the results obtained in these experiments.
  - **Task 2.2**. To evaluate the broadcast topology proposed by the BLE specification in a real environment, drawing conclusions from the results obtained.
  - **Task 2.3**. To evaluate the point-to-point connection topology included in the BLE specification in a real environment, drawing conclusions from the results obtained.
  - **Task 2.4**. To implement a basic BLE server using node.js environment to be used in the CSRmesh evaluation.
  - **Task 2.5**. To evaluate CSRmesh, the proprietary mesh network topology proposed by CSR to deal with the limitations of the initial BLE topologies.
  - **Task 2.6**. To identify the BLE parameters with a significant impact on the network performance.
- **Task 3**. Implementation of our BLE mesh network proposal.
  - **Task 3.1**. To design a mesh network that: (1) fulfils the requirements of Industry 4.0, (2) complies with the BLE standard and (3) is a heterogeneous network, enabling the data to be exchanged between any BLE devices.
  - **Task 3.2**. To implement the proposed mesh network in the available BLE devices, which use BLE 4.0 and 4.1.
  - **Task 3.3**. To design and to conduct the necessary experiments to evaluate the proposed mesh network in a real environment with the coexistence of other wireless networks, verifying its correct operation. These results can be compared with those obtained in the CSRmesh evaluation.
- **Task 4**. Inclusion of new devices in our BLE mesh network.
  - **Task 4.1**. To design an Android application for mobile devices: smartphones and tablets.
  - **Task 4.2**. To develop the application designed in Android, enabling any Android device to be part of the mesh network.
  - **Task 4.3**. To verify the correct operation of the Android application.
  - **Task 4.4**. To implement the necessary functions so that the wearable devices are able to transmit information using the mesh network.

  - **Task 4.5**. To verify the correct transmission of information by the wearables through our BLE mesh network.

  - **Task 4.6**. To design and carry out the experiments to evaluate the operation of the proposed heterogeneous network, using different devices, focusing on the mobile devices that facilitate the inclusion of users in the deployed mesh network.

- **Task 5**. In-depth study of the Bluetooth mesh specification.

  - **Task 5.1**. To study the architecture of Bluetooth mesh in depth, the specification of which entailed the official inclusion of mesh topology in BLE.

  - **Task 5.2**. To study and understand the provisioning procedure defined by the Bluetooth mesh specification. This procedure is of crucial importance as it provides BLE devices with the necessary information to send and receive messages from the rest of the devices in the network.

  - **Task 5.3**. To review the research work related to this specification in order to know its strengths and limitations.

- **Task 6**. Implementation of the Bluetooth mesh in BLE 4.0 devices.

  - **Task 6.1**. To conduct a study of existing Bluetooth mesh implementations, despite being developed over later BLE versions.

  - **Task 6.2**. To study the security algorithms used in Bluetooth mesh. Bluetooth mesh considers data security to be of great importance and therefore implements symmetric encryption algorithms for data transmissions as well as asymmetric encryption algorithms for sending keys.

  - **Task 6.3**. To design the implementation of the provisioning procedure, in which the provisioner device sends the necessary information (including the network key) to the new network devices. This design establishes, among other things, the relationship between the different security modules, as well as their integration.

  - **Task 6.4**. To implement the provisioning procedure, as defined in the Bluetooth mesh specification. To verify the correct functioning of our implementation, a set of experiments will carry out using BLE 5.0 devices with different implementations of this procedure as provisioners.

  - **Task 6.5**. To design the implementation of the different layers of the architecture defined in the Bluetooth mesh specification, taking into consideration aspects such as the roles required at each moment, the data structures or the use of the appropriate parameters, among others.

  - **Task 6.6**. To implement the Bluetooth mesh architecture for BLE 4.0 devices. Being built on top of the lower BLE layers, this implementation is compatible with most BLE devices.
  - **Task 6.7**. To verify the correct operation of our implementation by exchanging data with different BLE 5.0 devices using Bluetooth mesh, once the provisioning procedure is completed.
  - **Task 6.8**. To evaluate the performance of BLE 4.0 devices. Although Bluetooth mesh is compatible with the first version of BLE, this has a number of limitations that can reduce its performance, so an evaluation is necessary.

- **Task 7**. Improvement of the provisioning procedure.
  - **Task 7.1**. To design a proposal to make the provisioning procedure lighter, allowing devices with the first versions of BLE to be provisioned in a shorter time.
  - **Task 7.2**. To carry out the implementation of our proposal, verifying its correct operation.
  - **Task 7.3**. To evaluate the implementation of our provisioning proposal, comparing the results with those obtained for the standard provisioning procedure.

- **Task 8**. In-depth study of the friendship relationship.
  - **Task 8.1**. To study the friendship mechanism provided by the Bluetooth mesh specification in depth.
  - **Task 8.2**. To review the state of the art related to this mechanism, in order to be familiar with the releated research.

- **Task 9**. Optimisation of the friendship relationship of Bluetooth mesh.
  - **Task 9.1**. To study different approaches for the optimisation of low consumption Bluetooth mesh nodes. These approaches include: improving the Bearer Layer, improving time synchronisation and improving advertising channel utilisation.
  - **Task 9.2**. To design a proposal to optimise low power Bluetooth mesh nodes using Burst Transmissions with the Listen Before Transmit technique. This design requires taking into account parameters such as the number of messages per burst, or the established listening time.
  - **Task 9.3**. To implement our optimisation proposal for the low power nodes.
  - **Task 9.4**. To evaluate our proposal, comparing the results with those obtained by using the method proposed by the Bluetooth mesh specification.

1.4 Dissertation Outline

In order to achieve the goals described above, the rest of this Doctoral Thesis is organised into the following chapters:

- **Chapter 2. Background**. This chapter presents an overview of the context of this dissertation, the IoT and IIoT paradigms, as well as the BLE and Bluetooth mesh technologies.
- **Chapter 3. Preliminary Evaluation: BLE Topologies for Industry 4.0**. This chapter contains our initial evaluation, conducted using the topologies included in the BLE specification.
- **Chapter 4. Our Proposal for BLE Mesh**. This chapter details our proposal for a BLE mesh network, called collaborative mesh, as well as its evaluation, with special emphasis on the different interconnected devices.
- **Chapter 5. Providing Interoperability in Bluetooth mesh**. This chapter covers our implementation of the Bluetooth mesh protocol stack for devices equipped with BLE 4.0, including the provisioning procedure, as well as our proposal for improvement and posterior evaluation.
- **Chapter 6. Optimisation of the Friendship Mechanism**. This chapter focuses on the proposed improvement of the friendship mechanism provided by the Bluetooth mesh specification.
- **Chapter 7. Conclusions and Future Work**. This chapter concludes this Doctoral Thesis and presents some ideas for future work.
- **Appendix A. Hardware and Software**. This appendix includes the hardware platforms and software used throughout this Doctoral Thesis.

CHAPTER 2

Background

The requirements of current IoT applications are highly challenging for wireless systems. These requirements include: interoperability, robustness, reliability, low delay and low energy consumption [28]. In this scenario, and especially related to IoT, BLE was released. BLE was included for the first time in the Bluetooth 4.0 Core Specification [6] in 2010. Since then, it has been growing at extraordinary speed, due to its connection with smartphones, tablets, wearables and mobile computing, and its active and early adoption by mobile industry heavyweights [7]. However, the latest trends of IoT and Industry 4.0 demands new requirements [29] unmet by the available BLE topologies. For this reason, a new topology was proposed for BLE, firstly by academia and companies and finally by the Bluetooth SIG itself in the Bluetooth mesh suite of specifications [17, 18, 19] released in 2017.

The following section presents the terms of IoT and Industry 4.0, while the remaining sections describe in detail the main concepts of Bluetooth Low Energy and Bluetooth mesh.

2.1 IoT and Industry 4.0

IoT, also called the Internet of Everything, is a new technological paradigm conceived as a global network of machines and devices with the capacity to interact with each other. With IoT, many objects will take part in this global network invisibly embedded in the environment around us. Among the most important components of IoT we can distinguish [30]:

- Connected smart objects. As mentioned, smart objects are the core element of IoT. These objects are equipped with all types of sensors that allow a large amount of data to be collected. Furthermore, these smart objects are connected to each other, usually through wireless technology (such as Bluetooth, Wi-Fi, LoRa or ZigBee), which enables them to transmit the data collected.
- The IoT gateway, which manages the data flow between the different networks and protocols. These gateways are usually compatible with TCP/IP, allowing the data collected to be sent for processing.

- IoT cloud. The data collected is usually sent to the IoT cloud for efficient processing. IoT clouds are high performance server networks that enable huge amounts of data to be processed, analysed, managed and stored. In addition, they can be remotely accessed.
- Analytics. The process of converting data collected by sensors into useful information that can be interpreted and analysed in depth.
- User interface: the visible part of the IoT systems, which allows users to access the information obtained.

IoT has very different areas of application, such as the manufacturing or industrial sectors, transportation or mobility, energy, retail, cities, healthcare, supply chain, agriculture, buildings, enterprise, finance, smart homes and wearables. Figure 2.1 [2] shows the main areas of the 1414 published IoT projects (not including consumer projects) and the trend in each area compared to the 2018 study.

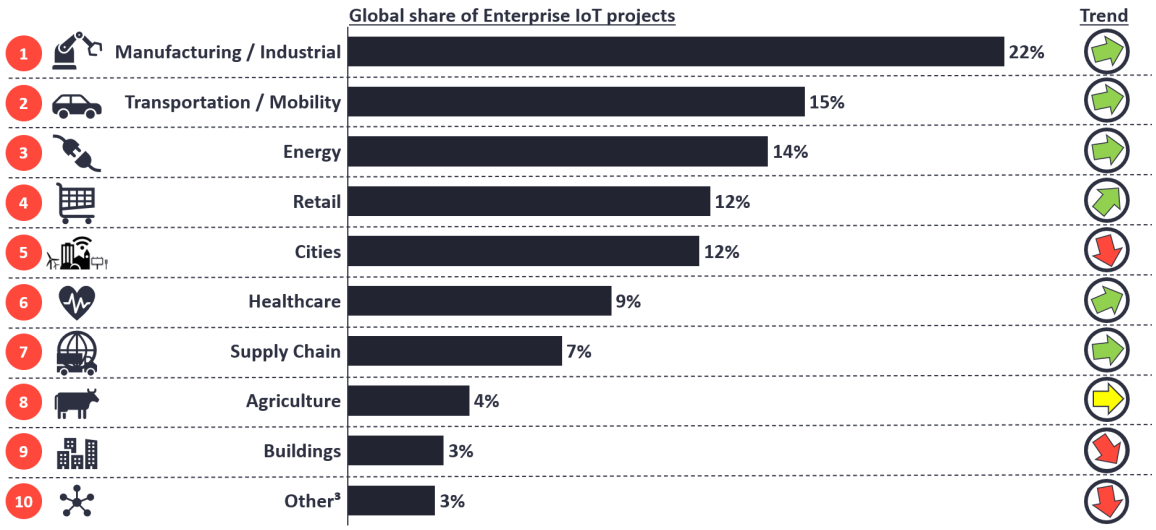


Figure 2.1: Top 10 IoT application areas 2020 according IoT analytics [2].

As stated above, the main area of application of IoT is the manufacturing or industrial area. This is closely related to another concept, Industry 4.0.

IoT is recognised as one of the most important areas of the technology of the future and is receiving extensive attention from a great variety of industries. However, the real value of IoT for companies lies in the interconnected devices becoming fully integrated with their different systems and applications [31].

Industry 4.0 represents the Fourth Industrial Revolution and can be defined as the integration of complex machinery, devices with connected sensors and software used to predict, control and plan, enabling improved business and societal outcomes [32]. The five main characteristics of Industry 4.0 are digitalisation, optimisation and customised production, automation and adaptation, Human-Machine Interaction (HMI), value-added services and

companies with automatic data exchange and communication. These characteristics are not only strongly related to Internet technologies and advanced algorithms, but also indicate that Industry 4.0 is an industrial process of added value and knowledge management [33]. Mobile and Cloud computing, Big Data and the IoT are key-enabler technologies for Industry 4.0 (which corresponds to the Manufacturing area of the IoT [32]) enabling the establishment of smart factories, services and products. Furthermore, Industry 4.0 has a great opportunity to create sustainable industrial value in the three dimensions of sustainability: economic, social and environmental [34].

Industry 4.0 has emerged to transform the current industrial model and introduce digitalisation into traditional factories, improving production rates and promoting collaboration [33]. The 4.0 attribute focuses on the IoT [1] applied to industrial systems in order to interconnect objects, machines and humans in smart factories. However, IoT is not the only pillar of Industry 4.0, being able to differentiate a total of 9 technologies that are transforming industrial production [35]. These are listed below:

- Big Data and Analytics. In an Industry 4.0 context, the collection and comprehensive analysis of data from many different sources will support real-time decision making.
- Autonomous Robots. Robots will interact with each other and with humans, learning from them.
- Simulation. Simulations will make it possible to use the data in real time and mirror the physical world in a virtual model. This will permit testing and optimising changes in the virtual world before physical change, thus reducing time and increasing quality.
- Horizontal and Vertical System Integration. Companies, departments, functions and capabilities will become much more cohesive, as universal data integration networks between companies evolve and allow for truly automated value chains.
- The Industrial Internet of Things. Industry 4.0 means that more devices will benefit from embedded computing. This will allow field devices to communicate and interact with each other as well as with centralised controllers.
- Cybersecurity. With increased connectivity and use of communication protocols, the need to protect industrial systems from cyber security threats increases dramatically. As a result, secure and reliable communications and sophisticated identity and access management for machines and users become essential.
- The Cloud. More companies will require greater data exchange across sites and enterprise boundaries. With the improved performance of cloud technologies, data and machine functionality will increasingly be deployed in the cloud, enabling more data-based services.
- Additive Manufacturing. Additive manufacturing, such as 3-D printing, is mainly used to make prototypes and produce individual components. With Industry 4.0,

these additive manufacturing methods will be widely used to produce small batches of custom products.

- Augmented Reality. Augmented-reality-based systems will support a variety of services, to provide workers with real-time information that improves decision making and work procedures.

Once the concepts of IoT and Industry 4.0 have been established, their relationship can be appreciated. This relationship is illustrated by Figure 2.2, with the work of this Doctoral Thesis being contextualised at the intersection between both concepts: connectivity and communication among devices. Previous network topologies were enough to cover small and medium IoT installations. However, the recent emergence of Industry 4.0 includes IoT networks in factories, and changes the requirements of these networks. These new requirements demanded in the new communication protocols are as follows [36, 37]:

- Total coverage: the entire building space must be completely covered by the network, avoiding dead zones where users cannot communicate.
- Zero Fails: each transmitted packet must be received at its destination, with a Packet Reception Rate (PRR) close to 100% in communications to provide high performance.
- Sustainability: covering both software (devices using efficient programs) and hardware (reducing the number of devices and using a low power standard). In this context, two new concepts emerge: green-by (IoT network linking physical devices with operators to afford efficient operation) and green-in (techniques to encourage the deployment of cost efficient networks) [34, 38, 39].

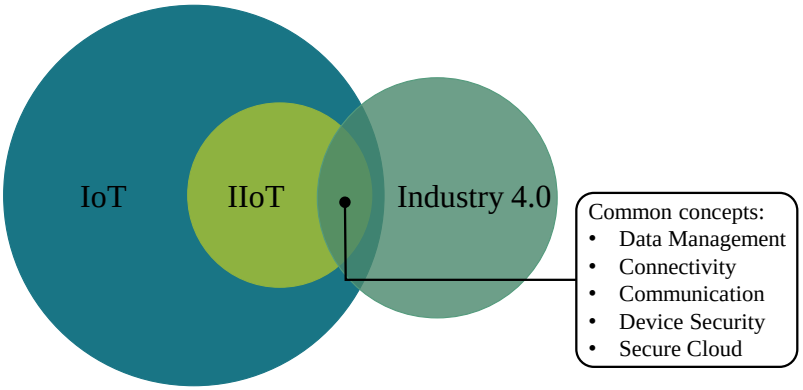


Figure 2.2: IoT and Industry 4.0 [40].

2.2 Bluetooth Low Energy

BLE was first included in the Bluetooth specification in its 4.0 version, and the aims of its design are different to those of classic Bluetooth. The starting point of BLE was Wibree, a very low power technology designed by Nokia in 2006 to replace Bluetooth in this type

of low power application [7]. Despite this initial idea, Wibree technology was incorporated into the main Bluetooth standard in 2010 by the Bluetooth SIG with the adoption of Bluetooth Core Specification 4.0 [6]. This section focuses on BLE, given the importance of this technology for the realisation of this dissertation.

2.2.1 Definition and Objectives

BLE can be defined as a smarter, low-power version of Bluetooth, designed to complement it and have the lowest power consumption possible. Although BLE operates in the same ISM band and borrows much of the technology from its predecessor, it should be considered a different technology as it pursues different objectives and is targeted at different market niches [41]. To achieve this, BLE has taken a different path to that of Bluetooth, opting for minimum consumption and renouncing throughput. In exchange for not having large data transfers, BLE provides the possibility of maintaining a connection for several days. This minimal consumption has made it ideal for battery-powered devices that do not yet have appropriate wireless technology.

For all these reasons, it is clear that the main objective of BLE was to create a short-range wireless technology with the lowest possible consumption, but without forgetting the objectives established by Bluetooth:

- Worldwide use, through the use of the 2.4 GHz Industrial, Scientific and Medical (ISM) frequency band, which is available worldwide without license cost.
- Low cost, an objective related to that of low consumption, being possible to satisfy both using a smaller volume of memory and a lower processing power, at the cost of forgoing high power and the use of more complex network topologies, for example.
- Robustness, using the Adaptative Frequency Hopping (AFH) technology of Bluetooth, which helps to detect and avoid interference, being very useful in such a widely used frequency band.
- Short range, a simple objective for BLE to fulfil, as, being a low consumption system, it must emit with low power and maintain high sensitivity in the receiver, reducing the power required to collect the signals. In any event, a short range does not imply that the devices should be excessively close, as BLE is designed to be a personal area, with the target range being around 100 metres.
- Low consumption, an objective that, although already included in the classic version, BLE has intensified, reducing consumption by one or two levels of magnitude.

2.2.2 Technical Information

Regarding the technical information of BLE, some of the most important features defined by the specification are:

- Data transmission in very small packets (8 to 27 bytes) Version 5.0 of BLE increased this maximum size to 255 bytes with the extended advertisements.
- Throughput of 1 Mbps, increased to 2 Mbps in the latest versions.
- Use of AFH technology to minimise interference.
- Control of the host from the controller, allowing the host to be suspended and reactivated when required.
- Very low latencies in data transmissions and connection establishment (< 3 ms).
- Range of more than 100 metres. In the latest versions the communication range has been increased up to about 300 metres.
- Use of 24-bit Cyclic Redundancy Check (CRC) for greater robustness against interference.
- Use of Advanced Encryption Standard (AES) encryption to authenticate and protect the packets.
- 32-bit device addresses, allowing a huge number of devices in the star topology. Later versions have made this topology more flexible, providing more complex connections.

2.2.3 Protocol Stack

BLE is defined in the specification as a layered architecture. This architecture is divided into three parts, namely application, host and controller, which are shown in Figure 2.3:

- The application is the upper layer, and contains the logic and user interface. It is also responsible for controlling the data related to the application that is running at any given time.
- The host includes the following layers of the BLE architecture.
  - Generic Access Profile (GAP).
  - Generic Attribute Profile (GATT).
  - Security Manager Protocol (SM).
  - Attribute Protocol (ATT).
  - Logical Link Control and Adaptation Protocol (L2CAP).
  - Host Controller Interface (HCI).
- The controller contains the following layers of BLE:
  - HCI
  - Link Layer (LL)
  - Physical Layer (PHY)

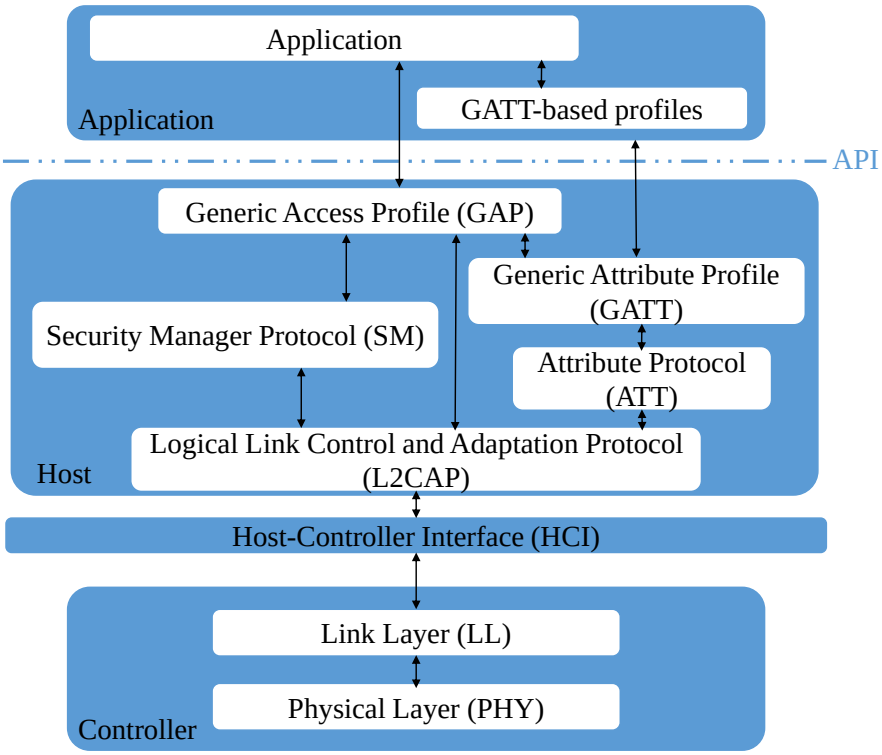


Figure 2.3: BLE protocol stack.

The main characteristics of each of the layers of the BLE protocol are presented below. Given the importance of the lower layers, both for our work and for the Bluetooth mesh specification, they are presented in more detail.

2.2.3.1 Physical Layer

The physical layer is responsible for modulating and demodulating the analogue signals sent and received, respectively, transforming the digital signals into analogue ones and vice versa.

For communication, BLE uses the 2.4 GHz ISM band, which is divided into 40 2-MHz channels, ranging from 2.4000 GHz to 2.4835 GHz. Of all these channels, the last three (37, 38 and 39) are called advertising channels, and are used to send broadcast messages and advertisements to establish connections, while the remaining channels are used for data transmission once the connection has been established. This division is shown in Figure 2.4.

The ISM radio band has significant advantages, including being free to use and having no licence requirements worldwide, making it cheaper for devices using BLE. However, this means that the ISM band is used by different technologies, such as Wi-Fi, ZigBee or Bluetooth, and interference can easily occur.

BLE technology is designed to minimise possible interference. In order to do this, broadcast messages are normally sent through all three advertising channels, while messages sent

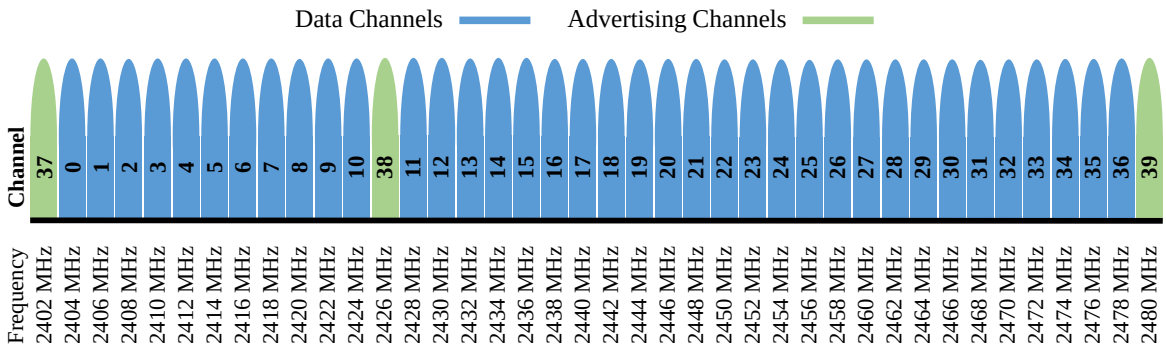


Figure 2.4: BLE frequency channels.

through the data channels use AFH technology. By means of this technique, the channel through which data messages are sent is changed, avoiding possible collisions if any technology is using a channel on the same radio frequency.

Like classic Bluetooth, BLE uses Gaussian Frequency Shift Keying (GFSK) modulation to encode the information. The bit rate defined for BLE is 1 Mbit/s, although in practice this limit is never reached, due to the extra data introduced into each of the layers of the protocol.

2.2.3.2 Link Layer

The Link Layer is directly connected to the Physical Layer, and consists of a hardware part and a software part, since most of its functionality is easily automated by hardware but complicated to implement by software. The Link Layer is the only layer with real time restrictions, since it needs to meet the requirements defined by the standard, so is usually isolated from the rest of the layers to hide its complexity and time requirements.

The operation of the link layer can be described as a state machine with five different states:

- Standby: In this state no packets are transmitted or received, and the device can only switch to another state
- Advertising: This state allows for the transmission of packets through the channels intended for advertisement packets. A device in this state is known as an advertiser.
- Scanning: The device listens to the advertising channels to receive the advertising packets that other devices are transmitting. A device in this state is also known as a scanner
- Initiating: The device scans the advertising channels for packets from one or more particular devices, responding to these packets to initiate a connection. A device in this state is called an initiator.

- Connection: Two devices in this state are connected, transmitting data between them. Within this state two different roles are defined:
  - Master: This is the device that has reached the connection state from the initiating state. It will communicate with one or more devices in the slave role, defining transmission times.
  - Slave: This is the device that has reached the connection state from the advertising state. It can only be connected to one master device at a time.

In the BLE specification 4.0, a device can be in a single state at any given time. Although these limitations have been made more flexible in later versions, the states as defined in specification 4.0 are presented here, as this is the one used in our work. The state machine is shown in Figure 2.5. The different states are presented in detail below

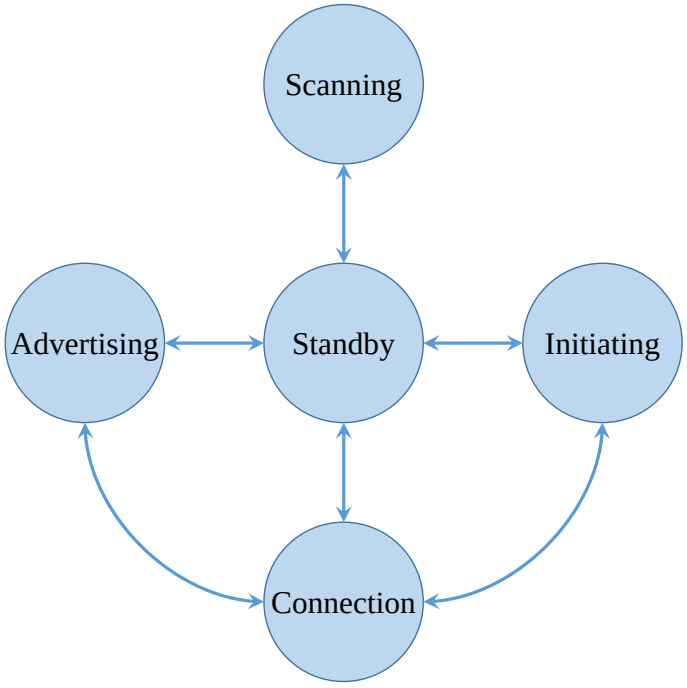


Figure 2.5: Link Layer state machine [6].

Connectionless States

Regarding the connectionless states of the Link Layer state machine, we find four different states: standby, advertising, scanning and initiating.

The standby state is the default state of the link layer. In this state the device does not receive or send packets, but keeps the controller in a standby mode that allows it to consume less power. A device in this state can access the advertising, scanning or initiating states.

Concerning the advertising and scanning states, both work with advertisement type packets. These packets are sent by an advertiser at a fixed time rate, defined by the advertisement interval, which can vary from 20 ms to 10.24 s (plus a pseudo-random delay, introduced by the Bluetooth controller to avoid collisions, which ranges from 0 to 10 ms). The shorter this interval, the higher is the transmission frequency, which increases the possibilities of the message being received, but also increases the power consumption of the device. A longer interval decreases the power consumption, but also the possibility of the packet being received. The Bluetooth specification defines sub-ranges of values within this advertising interval depending on whether the function of the advertiser is to establish a connection (shorter time intervals), or to send data in broadcast mode (longer time intervals).

Since there are three frequency channels for transmitting advertisement packets and the advertiser and scanner are not synchronized in any way, the packets will only be received when the listening coincides randomly with the transmission, as illustrated in Figure 2.6.

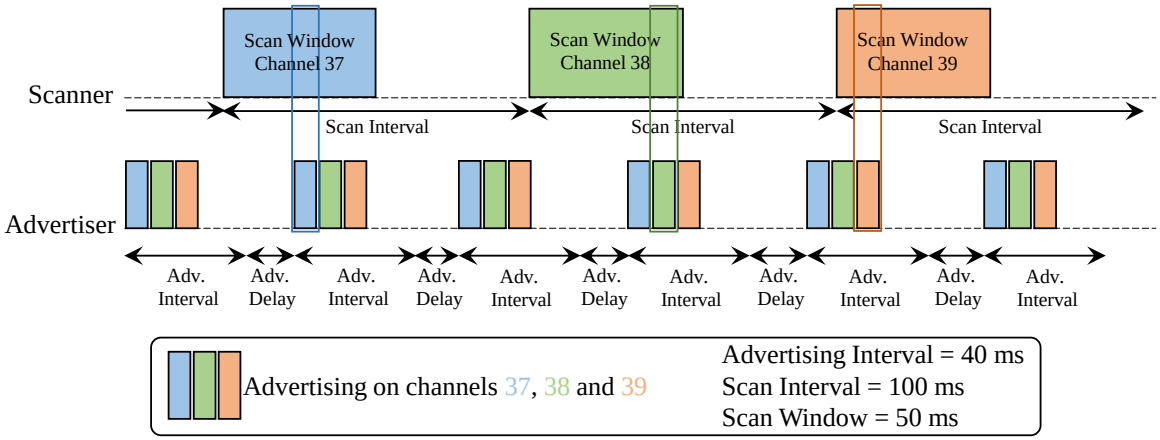


Figure 2.6: Advertising and scanning representation.

The parameters related to the scanner are the scan interval and the scan window. These parameters define the frequency and the time for which a scanner will scan the network for advertisement messages. As in the previous case, these parameters affect the power consumption of the scanner. Regarding the scanning process, the specification defines two types, which are detailed below:

- Passive scanning: The scanner only scans the network, so the advertiser is unable to know if any device is receiving its messages
- Active scanning: The scanner sends a scan response packet after receiving the advertising packet. This allows the advertiser to send a greater amount of data, although, theoretically, this type of scan is not designed to provide a way for the scanner to send data to the advertiser.

Finally, the initiating state is similar to the already mentioned scanning. However, an intiator scans the network for advertisers that could establish a connection, sending a con-

nection request to the desired device. If this process is successfully completed, the two devices involved will change their status to connection, as detailed below.

Connection State

When establishing a connection between two devices, the advertising and initiating states intervene, according to the following steps:

1. The initiator, which will play the role of master, scans the network for advertisers that accept connection requests. A master device can select the slave based on its address or the data sent.
2. Once the desired advertiser has been found, the initiator sends a connection request. This packet contains the frequency hop increment, which determines the frequency hops that will be used by the master and slave during connection (see AFH, in Section 2.2.3.1).
3. The advertiser sends a response to the initiator, establishing the connection and changing its states to connection. As mentioned above, the connection state has two roles: the master, which will be played by the initiator, and the slave, which will be played by the advertiser.

A connection can be defined as a sequence of data that is exchanged between the master and slave devices, using the parameters specified when it is established. Each of these data exchanges is called a connection event and is shown in Figure 2.7. During a connection event, the master and the slave device alternately send and receive packets.

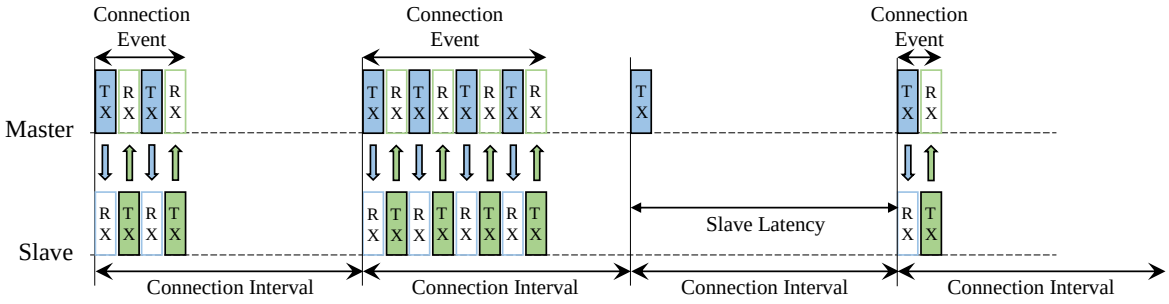


Figure 2.7: Connection events.

A connection event is considered open as long as the devices continue to send packets and can be closed by any of the devices. The reasons for closing a connection event are that: (1) data transmission has been completed, (2) multiple messages have been received with an erroneous CRC, or (3) packets have stopped being received.

There are different parameters in the connection events:

- Connection interval: Sets the time between the start of two consecutive connection events. The master device is responsible for preventing overlaps by closing one connection event before the start of the next. The value of this parameter ranges from

7.5 ms (higher data flow and also higher consumption) to 4 s (lower data flow and lower consumption).

- Slave latency: Enables the slave device to use a reduced number of connection events, establishing the maximum number of connection events that the slave device can skip without losing the connection.
- Connection supervision timeout: Sets the maximum time between the reception of two data packets before the connection is considered lost.
- Connection Transmit Window: Enables the master device to efficiently manage the various connection events or other activities that need to be performed. This provides flexibility in choosing the start of a connection event, within a time window.

Once two devices have established a connection, data packets are used to bidirectionally transport information between the master and the slave device.

BLE Packet Format

The Link Layer has only one packet format used for both advertising and data packets. This format is shown in Figure 2.8.

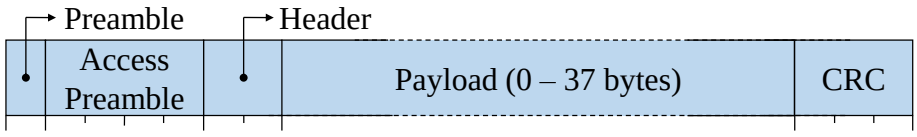


Figure 2.8: BLE packet format.

A brief definition of each of the fields that each packet consists of is shown below:

- Preamble: 1-byte size field, which is sent first. This field contains a succession of alternate ones and zeros to enable the receiver to carry out tasks related to the transmission, such as synchronization of the frequency. This field has a fixed value depending on whether the packet is sent through an advertising channel or a data channel.
- Access Address: 4-byte length field. This field also varies depending on the type of packet involved. The advertisement packets always carry the same access address (0x8E89BED6). Data packets contain a 32-bit access address that is generated in the Initiating state and sent to the device which the connection is established with.
- Header: 2-byte field. Although usually included in the PDU, both advertisement and data packets include it, although its content varies:
  - The advertisement packet header contains: (1) Type of Protocol Data Unit (PDU); (2) TxAdd and (3) RxAdd fields that specify whether the included addresses are public or random; (4) Length of the included payload field; and (5) some Reserved for Future Use (RFU) fields.

  - The data packet header includes: (1) Logical Link ID (LLID), which indicates the type of data packet in question: control packet, full data packet or data fragment; (2) Next Expected Sequence Number (NESN), which allows a device to notify its peer that a packet has been received; (3) Sequence Number (SN), used to identify the packet; (4) More Data (MD), which indicates whether there are more data fragments; (5) Length in bytes of the Payload field and the MIC (if used); and (6) RFU fields.

- PDU: This field varies depending on the packet type:
  - In advertisement packets, the PDU contains the 6-byte public or random address of the advertiser device. The rest of the PDU is divided into AD structures with three fields: (1) AD structure length; (2) AD type, which indicates the type of data in the structure; and (3) AD data. The specification defines different AD types, but additional types can be defined by the user.
  - Regarding data packets, two types of payloads are defined, data and control. Data payloads contain the data of the upper layers. If the connection established is encrypted, the payload contains an MIC field to allow the receiver to decrypt the data. Control payloads contain the code of the control operation to be performed as well as the data necessary to complete this operation.
- CRC: 24-bit CRC calculated using the PDU field.

Link Layer Security

To provide security and maintain the confidentiality and integrity of data, the BLE Link Layer uses the following techniques:

- CRC: this layer is responsible for checking that all received packets are free of CRC errors.
- Encryption: the link layer provides an encrypted link for secure data exchange. The keys are generated and managed by the upper layers, but this layer is responsible for encrypting and decrypting the packets transparently to the upper layers.
- White List: These lists contain only BLE device addresses, filtering the received messages when the device is in the advertising, scanning or initiating states.

2.2.3.3 Host Controller Interface

The HCI is a protocol that enables communication between host and controller parts via a serial interface. This separation is due to the controller module having a temporarily stricter protocol stack, as well as being less appropriate for more advanced CPUs (common on the host part). Some examples of this configuration are found in most devices, such as computers, tablets or smart phones, where the host module runs on the main CPU, which sends commands to a separate hardware chip, connected via a UART or USB interface.

The Bluetooth specification defines the HCI interface as a set of commands and events that enable interaction between the host and the controller, as well as a format for data packets and a set of rules for flow control. There are different standards for the HCI transport layer, used according to the different hardware interfaces, allowing the same commands, events and data packets to be transmitted. These standards include USB (for PCs) and UART (for mobile devices).

Currently, since semiconductor technology is sufficiently affordable, it is possible to incorporate the controller, host and application layers into a single package (System-on-Chip (SoC)). This can be very of great use in embedded system applications, since it reduces the size and cost of the end device. In the case of BLE, it is common to implement all three parts on a single chip with a low-power CPU. Although, in these cases, the HCI is optional, it is usually implemented as an internal software interface.

2.2.3.4 Logical Link Control and Adaptation Protocol

L2CAP offers the service of a multiplexer protocol that receives packets from the upper layers and encapsulates them in the standard BLE packet for sending, performing the reverse action on reception.

L2CAP also provides fragmentation and recombination of packets from the upper layers, enabling the division of oversized packets into packets with a load of up to 27 bytes (including the header, which implies that the size of the final data field is 23 bytes), which is the maximum size supported by the BLE technology. This protocol can receive multiple fragmented packets, recombining them into the original packet and sending it to the upper layers. This enables applications to send and receive data packets of up to 64 kilobytes.

L2CAP is responsible for routing packets to two superior protocols: the ATT, for the exchange of application data, and the SM, which provides a framework for generating and distributing security keys between devices. Both are described below.

2.2.3.5 Attribute Protocol

The ATT protocol enables a device, known as a client, to access a set of attributes, with their associated values, exposed by another device called a server. Usually, the device in the master role described in Link Layer corresponds to the client device, while the device in the slave role corresponds to the server. Each attribute has the following associated fields:

- Attribute type, defined by a variable size Universally Unique Identifier (UUID), specifies what the attribute represents.
- 16-bit length attribute handler, which uniquely identifies a server attribute
- A set of permissions defined by the upper layers that use the attribute.
- Value of the attribute, in an array of bytes, with a length that can be fixed or variable, depending on the type of data that is stored in the attribute.

The ATT protocol also defines a series of operations to work with these attributes. These include operations related to protocol configuration or error handling, to find information on attributes, to allow the client to read and write the attributes from the server, and to indicate and notify the client of changes in attributes. There are different versions for each operation, allowing the action to be performed in different ways, as well as specifying whether an Acknowledgement (ACK) packet is required when the action is completed.

2.2.3.6 Security Manager Protocol

The SM protocol defines security methods and algorithms for pairing, generating and distributing keys between devices, as well as a set of security algorithms for these methods. This provides BLE with the capabilities to generate and exchange security keys, enabling secure communication between a pair of connected devices through an encrypted link.

The SM protocol provides support for performing the following security procedures:

- Pairing: This procedure generates a temporary security key in order to create a secure and encrypted link. The SM protocol provides three different methods for the pairing procedure: (1) Just Works, through an exchange of plain text packets, (2) Passkey Entry, in which the user is required to enter into the device a 6-digit key (passkey) generated by the other device; and (3) Out Of Band (OOB), which transfers the data through a different technology.
- Bonding: This process starts with the pairing procedure, followed by the generation and exchange of permanent security keys, which are stored in the device's non-volatile memory. This enables the creation of a permanent link between two devices, which will allow the creation of secure links in subsequent connections without the need to repeat the entire process again.
- Encryption Re-establishment: This procedure defines how the keys generated and stored in the bonding procedure shall be used to re-establish a secure and encrypted connection without the need to perform the previous processes again.

The SM protocol also includes three types of security mechanisms that can be used to provide different levels of security. These mechanisms can be used in a connection or during the advertising state:

- Encryption: This feature allows all the packets transmitted in an established connection to be encrypted.
- Privacy: This mechanism enables an advertiser to hide its public Bluetooth address, using instead a randomly generated address, which can be recognised by the scanners with which it has a bonding type connection.
- Signing: This feature allows devices to send signed and unencrypted packets through an established connection..

Each of these mechanisms can be used independently of the others, and more than one can be used simultaneously.

2.2.3.7 Generic Attribute Profile

The GATT Profile is built over the ATT protocol and establishes common operations and a framework for the transport and storage of data by this protocol, defining in detail how the ATT protocol is used to work with the attributes. In contrast to GAP, which defines the low-level interactions between BLE devices, GATT is responsible only for the processes and formats actually related to the data transfers.

GATT uses the ATT protocol to transport data between devices, for which it organises this data hierarchically into sections called services. Each of these services is divided into characteristics. Thus, each of the characteristics contained in a service contains the union of different user data with metadata which describes them.

Moreover, GATT defines a number of generic data objects that can be used in a variety of application profiles, which are known as GATT-based profiles. These profiles maintain the same client/server architecture used by the ATT protocol, but encapsulate the data in services. GATT-based profiles are the top elements in the GATT Profile data hierarchy. These profiles are designed to be used by an application or another profile, and allow a client to communicate with a server.

Each GATT-based profile specifies the structure in which its data is exchanged. This structure mainly defines two elements used in the profiles (upper levels of the hierarchy): services and characteristics, which are stored in attributes. Therefore, a GATT-based profile is composed of one or more services, which are necessary to perform a use case. Similarly, each of these services consists of characteristics or references to other services. A characteristic contains a value and may optionally contain information about that value. In other words, the services, characteristics and their components contain the data corresponding to their profile, all of which are stored in attributes.

2.2.3.8 Generic Access Profile

GAP enables a BLE device to work in conjunction with other devices using the same protocol. In order to do so, it provides a framework that every BLE device shall follow to allow other devices to correctly perform the operations specified in the standard. This framework includes the GAP service, based on the service offered by the GATT profile and that all BLE devices must include among their attributes.

GAP defines the procedures related to the discovery of BLE devices and to aspects of connection management. It also defines procedures related to the use of different levels of security, in addition to the data formats required to provide accessibility to parameters at the user interface level.

Four roles are defined for the devices in the GAP profile:

- Broadcaster: a device in this role sends advertisement-type messages.
- Observer: a device operating in this role receives advertisement-type messages.
- Peripheral: when a device accepts the establishment of a BLE connection using any procedure, it enters the role of Peripheral, which coincides with the slave role of the Link Layer.
- Central: this role is assigned to the devices that initiate the establishment of a connection, and coincides with the role of master of the Link Layer.

Modes and Procedures

This section shows the modes and procedures available in GAP, defined in the BLE specification, divided into groups according to their functionality:

- Broadcast mode and Observation procedure. This mode and procedure enables two devices to communicate unidirectionally, without the need to establish a connection, using advertisement messages instead.
- Discovery modes and procedures. All devices shall be in one of the three modes included in this group: Non-discoverable (allows the device not to be discovered by any procedure), Limited Discoverable (allows the device, temporarily, to be discovered by devices using the Limited Discovery or General Discovery procedures), General Discovery (allows the device to be discovered by devices using the General Discovery procedure).
- Connection modes and procedures. These modes and procedures enable the device to establish a connection with other devices, as well as to modify the parameters of connections already established. The modes included are: Non-Connectable (the device does not permit a connection to be established), Directed Connectable (the device accepts connection requests, but only from previously known devices) and Undirected Connectable (the device requests connection from any device).

  Regarding the procedures related to the connections between devices, GAP provides the following: Auto Connection Establishment procedure (enables connections to be established automatically), General Connection Establishment procedure (enables connections to be established with a set of known devices), Selective Connection Establishment procedure (enables connections to be established with a set of devices in the White List), Direct Connection Establishment procedure (enables connections to be established with a specific device), Connection Parameter Update procedure and Terminate Connection procedure.
- Bonding modes and procedures enable two connected devices to exchange and store information related to security and identity, allowing the creation of a reliable connection. There are two modes depending on whether the devices allow (Bondable)

or not (Non-Bondable) this type of connection, and a procedure to create it (Bonding Procedure).

- In addition to the procedures presented, GAP provides a number of security-related procedures and modes to authenticate, authorize and encrypt transmissions

2.3 Bluetooth mesh

This section presents an overview of the entire Bluetooth mesh, delving deeper into the most important aspects for our work. The Bluetooth SIG recently released the Bluetooth mesh suite of specifications [17, 18, 19]. These provide BLE with many-to-many communications in large-scale networks using a standard mesh topology.

The Bluetooth mesh is defined as a layered architecture built over the lowest layers of BLE specification (see Section 2.2), as illustrated in Figure 2.9. This enables all BLE devices to take part in a Bluetooth mesh network. There are five Bluetooth mesh layers (see Sect. 3 in [17]): Bearer Layer, Network Layer, Transport Layer, Access Layer and Mesh Models.

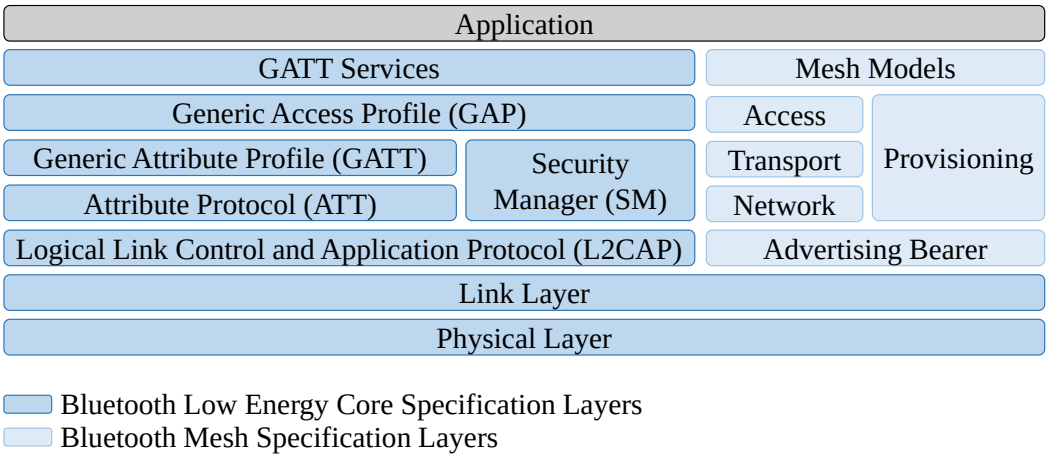


Figure 2.9: Bluetooth Low Energy and Bluetooth mesh Protocol Stack [42].

In order to join the mesh network, a BLE device needs to follow a procedure beforehand. This is known as the provisioning procedure, and it allows the device to securely receive the information needed to be a node in the mesh.

2.3.1 Overview of Mesh Operation

The Bluetooth mesh specification defines a mesh network based on a managed flooding mechanism. Using this mechanism, the nodes transmit messages using the broadcast capability of the BLE. These messages are relayed by the devices that receive them, which increases the coverage range of each individual device to the entire mesh network area. This specification uses two main mechanisms to manage flooding: Time To Live (TTL) and the message cache. Every message in the mesh network includes a TTL field, limiting the packet lifespan (hops) in the network. The use of message cache allows devices to avoid

relaying messages more than once. To do this, all received messages are added to a cache list. This list is revised when receiving a message, discarding it if it is already in the list. When a packet is first received, it is stored in the list for future comparison and relayed.

Bluetooth mesh is defined as a layered protocol built over the two lowest layers of BLE: the Physical Layer and the Link Layer, both explained in detail in Section 2.2. The Physical Layer uses the 2.4 GHz frequency band, where 40 channels are defined: 37 data channels and 3 advertisement channels. The Link Layer defines the two communication modes: establishing a connection between two devices, using data channels to transmit information, or using broadcast messages, where the advertising channels are used. Bluetooth mesh uses these broadcast transmissions as long as the BLE devices allow it, usually transmitting each message over all three advertisement channels. The other layers defined by the Bluetooth mesh specification are explained in Section 2.3.2.

With regard to the network topology, all the nodes are at the same level, with no central nodes or hierarchies. However, Bluetooth mesh defines some optional features that the nodes can use to improve the operation of the network. These features are:

- Relay feature: the nodes that enable this feature are called Relay Nodes (RNs). These nodes are the core of the mesh network, relaying the received messages over the advertising bearer, increasing the individual coverage range up to the entire coverage of the network.
- Proxy feature: this feature enables Proxy Nodes (PNs) to act as a bridge between the devices that need to establish a point-to-point connection to send data (over the GATT bearer) and the rest of nodes in the network (using the advertising bearer).
- Low power feature: this feature enables the nodes to remain in sleep mode for long periods. Since messages are easily lost when devices are not permanently scanning the network, Low Power Nodes (LPNs) require another device, the Friend Node (FN), which is continually scanning and stores the messages sent to LPNs. These nodes periodically wake up and obtain data from their FNs using asynchronous requests.
- Friend feature: this feature allows a node to be an FN. These nodes are usually also RNs that are constantly scanning the network. They have sufficient capacity to store the messages sent to the LPNs, permitting them to spend a significant part of their life cycle in sleep mode.

Figure 2.10 shows an example of a Bluetooth mesh network. Nodes 1-6 are LPNs which have established a friendship relationship with an FN. Nodes 7 and 8 are FNs and RNs. These nodes have relay and friend features enabled, so they relay the received and relay messages, as well as storing the messages for the associate LPNs. Nodes 9 and 10 are RNs which relay the received messages. Node 11 is an RN and a PN, and therefore, besides forwarding messages, it acts as a bridge to devices that are unable to take part in the mesh using the advertising bearer. This is the case of Node 12, which maintains a connection via

the GATT bearer. Finally, Nodes 13-18 have no enabled features; they receive packets at any time, but only send their own messages.

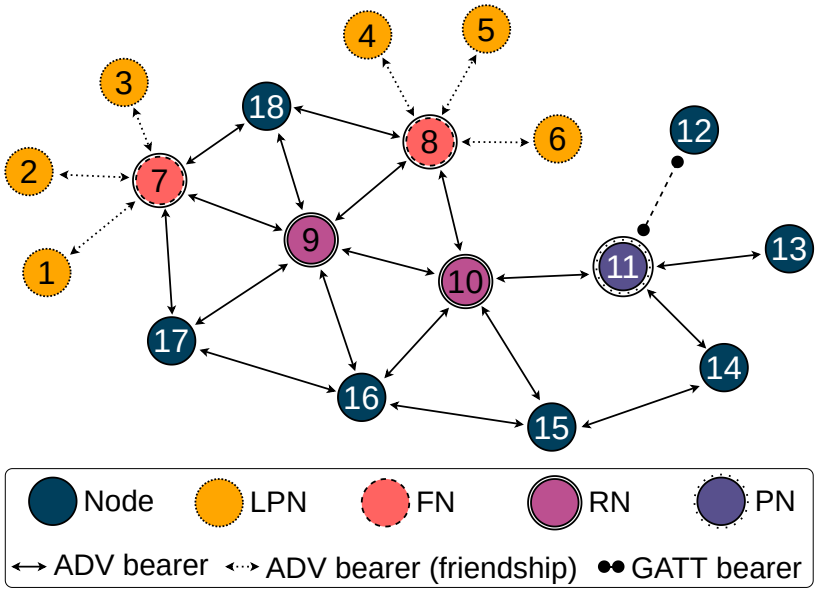


Figure 2.10: Example topology of a Bluetooth mesh network.

Both the low power and the friend features are used in the most important Bluetooth mesh optimisation mechanism, which is known as friendship. Considering its importance for our work, the friendship relationship is described in detail in the next section.

2.3.1.1 Friendship

Due to the mode of operation of Bluetooth mesh, battery-powered devices can drain their batteries quickly as a result of the consumption required by constant scanning of the network. To prevent this and enable any BLE device to be part of the mesh network, Bluetooth mesh defines the friendship relationship in its Upper Transport Layer. This allows an LPN to establish a relationship with a nearby (one-hop) FN. Once the friendship relationship is established between two devices, the FN stores the messages addressed to the LPN, as well as all those related to the groups the LPN is subscribed. In order to receive these messages, the LPN makes asynchronous requests to the FN (using Friend Poll messages), which sends timed responses to the LPN.

When the friendship is established, the FN and the LPN define the parameters for friendship data exchange. These parameters are the PollTimeout, the ReceiveDelay and the ReceiveWindow. The PollTimeout represents the maximum time between two consecutive LPN requests. The specification defines this time as a range from 1 second to 96 hours. When the LPN makes a request, there is a delay period for the FN to process the request and prepare the message, known as a ReceiveDelay, in which the LPN may enter sleep mode. The ReceiveDelay ranges from 10 to 255 ms. After this delay, the LPN scans the medium for a period of time called ReceiveWindow, which gives the FN sufficient time to

send the message. Once the message is received, the LPN returns to sleep mode. When no message is received in the ReceiveWindow, the LPN retries the request, until reaching the maximum number of retries. Exceeding this number terminates the friendship. The range of the ReceiveWindow is from 1 to 255 ms. Figure 2.11 shows the LPN state machine.

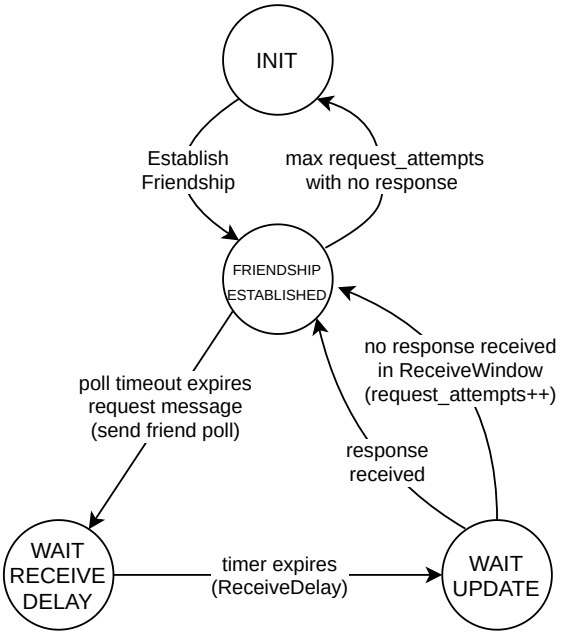


Figure 2.11: Bluetooth mesh LPN state machine.

The FN stores the packets destined for the LPN in a Friend Queue, the size of which is determined by the hardware specifications of the device. After making a request and receiving a data message from the FN, the LPN should make further requests until the Friend Queue is empty. To indicate this, the FN sends a Friend Update message. When this message is received, the LPN switches into sleep mode until the time required to make a new request expires. This process is shown in Figure 2.12.

2.3.2 Layered Architecture

Extending the capabilities of Bluetooth Low Energy, Bluetooth mesh networks support reliable, scalable and secure solutions for a wide range of commercial and industrial IoT applications for control, monitoring, and automation where tens, hundreds, or thousands of devices need to reliably and securely communicate with one another [43]. This section provides a brief summary of the main features and functionalities of the Bluetooth Mesh layers, which are shown in Figure 2.9.

2.3.2.1 Bearer Layer

Bearers permit the transport of mesh messages through the lower layers of BLE. Bluetooth mesh defines two bearers: the Advertising (ADV) bearer and the GATT bearer.

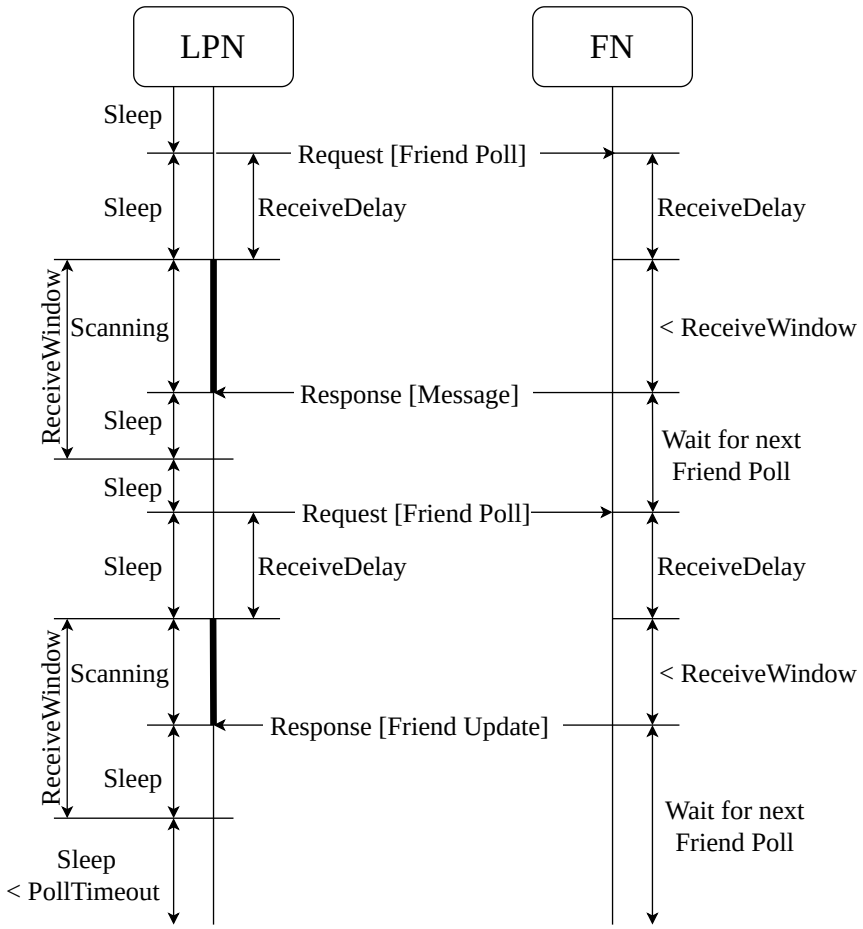


Figure 2.12: Bluetooth mesh friendship procedure.

ADV Bearer

When using the advertising bearer, which is shown in Figure 2.9, a mesh packet is encapsulated in a BLE advertising PDU using the Mesh Message AD Type identifier. The Mesh Message AD Type contains the Network PDU received from the Network Layer. These advertising packets are sent in the broadcaster role, which is the essence of the Bluetooth Mesh.

The advertisement packets which uses the Mesh Message AD Type are non-connectable and non-scannable undirected advertising events.

GATT Bearer

The GATT bearer enables devices that are incapable of supporting the ADV bearer to participate in a mesh network. Using the GATT bearer requires the Proxy protocol defined in Bluetooth Mesh specification to transmit and receive Proxy PDUs over a BLE connection. The GATT bearer uses a characteristic to write and receive notifications of mesh

messages using the Attribute Protocol (ATT) defined by BLE specification and explained in Section 2.2.

2.3.2.2 Network Layer

The Network Layer defines the network message format required to address transport messages towards one or more mesh elements through the Bearer Layer. It is responsible for deciding whether messages are relayed, accepted for processing in upper layers or rejected. This layer also defines how the network messages are encrypted and authenticated using the Network Ley, as well as how the header of each message is obfuscated using the Privacy Key. This subsection provides an overview of the most important aspects of the Network Layer.

Network PDU

The mesh Network PDU is encapsulated in a BLE packet. Depending on the bearer used to send it, the format of the final BLE packet will be one or another. Figure 2.13 illustrates the fields of the Network PDU, and they are briefly explained below:

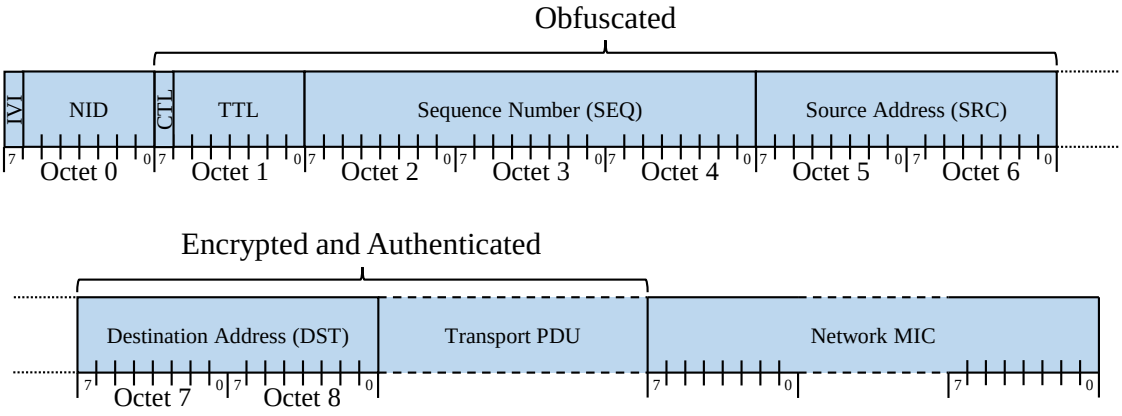


Figure 2.13: Network PDU fields.

- Initialisation Vector Index (IVI), the least significant bit of the Initialisation Vector (IV) used to encrypt and authenticate the Network PDU.
- Network Key Identifier (NID), the 7-bit identifier of the Network Key from which the Encryption Key and the Privacy Key used to authenticate and encrypt, respectively, the Network PDU were generated.
- Network Control (CTL), a 1-bit value that indicates whether the message is a Control Message or an Access Message.
- TTL, a 7-bit value used to limit the maximum number of packet hops through the network.

- Sequence Number (SEQ), 24-bit value which, combined with the IV Index, shall be unique for each Network PDU, protecting against replay attacks.
- Source Address (SRC), 2-byte address to identify the element that originated the Network PDU.
- Destination Address (DST), 2-byte address to identify the element (or group of elements) that the Network PDU is directed towards.
- Transport PDU: data received from the Transport Layer, which is not changed by the Network Layer. The Transport PDU and the Network Message Integrity Check (MIC) shall be a maximum of 160 bits.
- Network MIC: 32-bit or 64-bit (depending on the CTL bit) field which authenticates that the DST and Transport PDU fields have not been modified.

Network Layer Behaviour

When a message is received by a bearer and has passed the input filter, it is delivered to the Network Layer. In this layer, the NID value of the packet is checked to match one or more NIDs In that case, the message is deobfuscated and decoded using the Privacy and Encryption Keys relating to the matched Network Key, respectively. If the message authenticates the Network MIC, the addresses are valid, and the packet is not yet in message cache, the message is forwarded to the Lower Transport Layer.

Additionally, if Relay feature is supported and enabled, the TTL value has a value of 2 or greater, and the DST field is not a unicast address of an element of this node, then the TTL field is decremented, maintaining the rest of the original values, and the message is again encrypted, obfuscated and delivered to the Bearer Layer. Figure 2.14 represents the flowchart of an incoming Network PDU.

When a message is transmitted by an element of the device, it crosses the Network Layer. Although some fields are defined by upper layers, the Network Layer is responsible for setting the rest of them, namely the IVI, the NID and the SEQ. Furthermore, this layer encrypts the Transport PDU, calculating the Network MIC.

2.3.2.3 Transport Layer

The Transport Layer is split into the Lower Transport Layer and the Upper Transport Layer.

The Lower Transport Layer segments and reassembles Upper Transport Layer PDUs if necessary. These PDUs include the fields needed to identify a single/segmented packet and to reassemble whether required.

The Upper Transport Layer encrypts and authenticates Access Layer messages using the Application Key configured for the specific model or the Device Key, and is also responsible

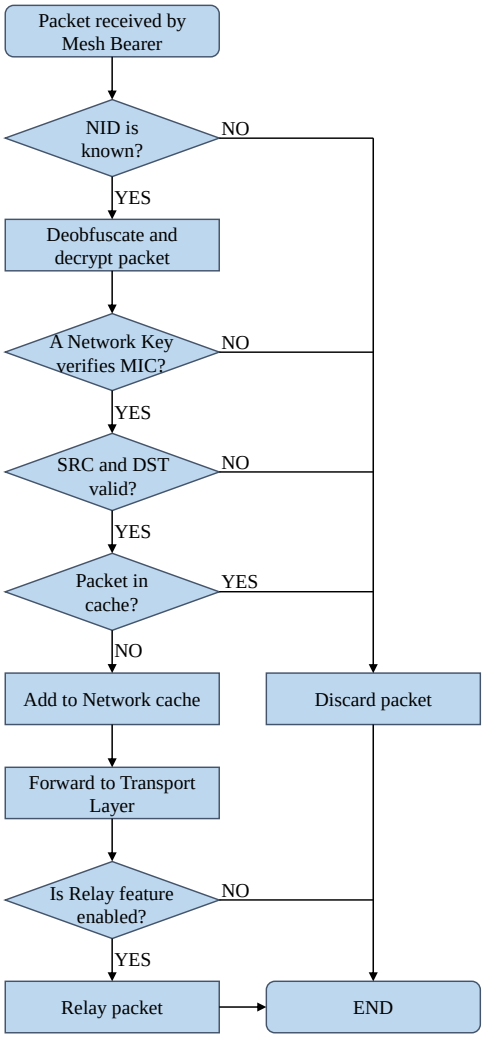


Figure 2.14: Incoming Network PDU Processing Flow [17].

for the Friendship mechanism. The Upper Transport Layer PDU contains the encrypted Access Layer PDU and the generated MIC.

2.3.2.4 Access Layer

The Access Layer defines how higher layers (applications) can use the Upper Transport layer, defining the format of the application data. This layer also defines and controls the security aspect (encryption and decryption) of the Upper Transport Layer. Moreover, it checks whether the application data has been received in the correct context (regarding Network Key and Application Key), before sending it to the upper layer.

2.3.2.5 Mesh Model Layer

The Mesh Model Layer is usually divided into Foundation Model Layer and Model Layer. Both layers define different models, messages and states: the Foundation Model Layer defines those required to configure and manage the mesh network, while the Model Layer

defines those used to standardise the operation of the different use cases. The Foundation Models are defined in [17], while models for some of the most typical use cases are defined in the Bluetooth Mesh Model specification [18].

2.3.3 Bluetooth mesh Security

Bluetooth Mesh uses the Advanced Encryption Standard-Counter with a Cipher Block Chaining-Message Authentication Code (AES-CCM) algorithm to secure messages at two different layers: at the Upper Transport Layer and at the Network Layer. The Upper Transport Layer uses the *Application Key* to encrypt and authenticate access payloads received from the Access Layer. The Network Layer uses the *Encryption Key* to encrypt and authenticate, as well as the *Privacy Key* to obfuscate different fields of Network PDUs, as shown in Figure 2.13. Both encryption and privacy keys are derived from the *Network Key* received in the provisioning procedure (see Section 2.3.4) using the Advanced Encryption Standard - Cipher Block Chaining Message Authentication Code (AES-CMAC) algorithm, among others, as defined in Sect. 3.8.2.6 in [17].

This double encryption-authentication system permits the data to remain protected not only from external attacks (using the *Network Key*) but also from network devices which can rely these messages but are not authorised to access its specific data (using the *Application Key*). Additionally, the header of all network messages is obfuscated to hide identifying information, while the use of the IVI and the SEQ fields provides protection against replay attacks.

2.3.4 Provisioning Procedure

The provisioning procedure is the fundamental process that permits a BLE device to take part in a Bluetooth mesh. In other words, through this process, an unprovisioned device becomes a mesh node. The provisioning data received by the unprovisioned device in this process include a network key, the current IV and the unicast address for each element. All these data are necessary to correctly send and receive messages in the mesh network: For this reason, they are securely exchanged over an insecure medium, using different encryption algorithms, which are presented later.

The device responsible for managing this process is called a provisioner, and is usually a mobile computing device. Each mesh network must have at least one provisioner, although multiple provisioners may be used (coordinated and sharing data).

When provisioning an unprovisioned device, the Provisioning Bearer needs to be established between this device and the provisioner. The unprovisioned device can be identified using its Device UUID. After that, the provisioner and unprovisioned device establish a shared secret using the Elliptic Curve Diffie-Hellman (ECDH) protocol. This protocol enables devices to authenticate and encrypt communications, using a key derived from the shared secret.

The provisioning protocol uses a layered architecture, which is shown in Figure 2.15. This protocol defines the PDUs transmitted in the communication between the provisioner and an unprovisioned device during this process, using the Generic Provisioning Layer. This layer defines the encapsulation of the provisioning PDUs in transactions, which can be segmented and reassembled. Finally, these transactions are sent over a Provisioning Bearer, which defines how communication is established, permitting the transactions to be sent from the Generic Provisioning Layer to a single device.

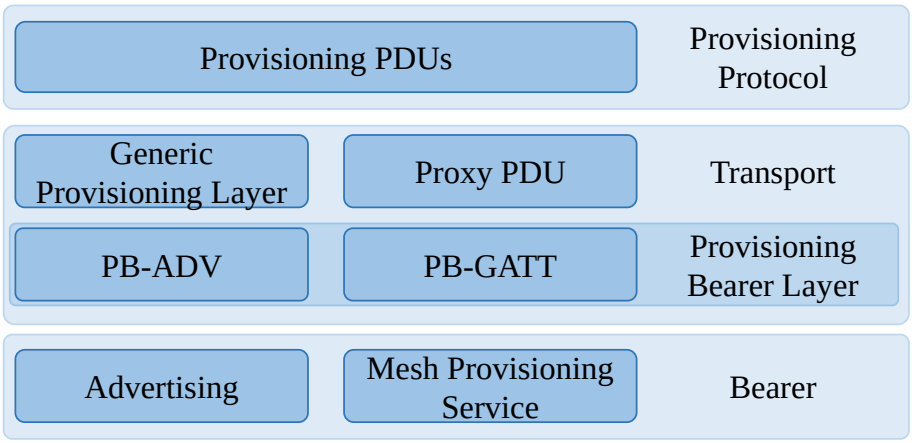


Figure 2.15: Provisioning Protocol Stack.

This section details the entire Provisioning protocol, giving a complete view of this process. To this end, this section is divided into the following parts: Provisioning Bearer Layer, where the two different bearers are exposed; Generic Provisioning Layer, which details the Generic Provisioning PDUs (Transport Layer); and Provisioning Protocol, where the entire packet exchange is described.

2.3.4.1 Provisioning Bearer Layer

Two different Provisioning Bearers are defined in Bluetooth mesh specification, depending on the BLE communication used: PB-ADV, over the BLE advertising channels; and PB-GATT, which uses Proxy PDUs within a point-to-point connection.

PB-ADV

PB-ADV permits the provisioner to provision a device using Generic Provisioning PDUs in advertisement packets over the advertising channels. The Advertising Type field in these packets is 0x29, to indicate it is a PB-ADV packet. The PDU of PB-ADV packets has three different fields:

- Link ID: This 4-octet field is used to identify a link between the provisioner and an unprovisioned device. Since the broadcast transmissions used in this type of provisioning do not inherently establish a session, it is provided through the Link Estab-

lishment procedure described below. Specific messages are defined to establish and terminate the link between a provisioner and a device.

- Transaction Number: This one-octet value identifies each individual Generic Provisioning PDU. When a PDU is segmented, all segments use the same Transaction Number. The first transaction sent by the provisioner uses the Transaction Number 0x00, while the first Transaction Number used by an unprovisioned device is 0x80. In both cases, this number is increased by one for each new transaction. In the case of Transaction ACKs, the Transaction Number is set to the same value as the transaction acknowledged.
- Generic Provisioning PDU: the first octet of this field contains the Generic Provisioning Control Format, which indicates the Generic Provisioning PDU Type. The different types are: Transaction Start, Transaction Acknowledgment, Transaction Continuation and Provisioning Bearer Control. These types are explained later. The other fields in the Generic Provisioning PDU depend on the type, so they are presented in their corresponding sections.

When PB-ADV is used, advertisement packets must be non-connectable undirected advertising events.

Finally, to ensure the reception of most incoming Generic Provisioning PDUs, devices should perform passive scanning with a duty cycle as close to 100% as possible. This means that the duration of the scan window should be as close to the duration of the scan interval as possible.

PB-GATT

PB-GATT is used to provision a device through Proxy PDUs, encapsulating the Provisioning PDUs within the Mesh Provisioning Service. This bearer enables devices to be provisioned when the provisioner or the unprovisioned device does not support the PB-ADV. However, PB-GATT requires the unprovisioned device to have the Mesh Provisioning Service.

The Mesh Provisioning Service permits a provisioner to provision a device, enabling this device to take part in the mesh network. This service has two different characteristics: Mesh Provisioning Data In and Mesh Provisioning Data Out. The Mesh Provisioning Data In characteristic is used by the provisioner (Provisioning Client) to send a Provisioning PDU encapsulated in a Proxy PDU to the unprovisioned device (Provisioning Server). The Mesh Provisioning Data Out characteristic is used to notify and send a Provisioning PDU (also encapsulated in a Proxy PDU) from a Provisioning Server to a Provisioning Client.

Mesh Provisioning Data In and Mesh Provisioning Data Out characteristic values are both 66 octets long, providing enough space for the longest Proxy PDU containing a Provisioning PDU. However, if the ATT Maximum Transmission Unit (MTU) negotiated in the connection is less than the size required by a Proxy PDU, transmissions need to be frag-

mented and reassembled by lower layers. Each PDU must be reassembled before being processed by the GATT profile.

Bluetooth mesh specification recommends establishing the connection interval between 250 and 1000 milliseconds. This enables devices to perform calculations with a high computational cost (such as Diffie-Hellman shared secret, or the different keys used by security algorithms) in a low power operation mode, avoiding the energy waste to maintain an idle link.

This service and its characteristics are compatible with BLE devices using Bluetooth Core Specification 4.0 or later. However, not all BLE devices include this service. Some of them can be upgraded easily, but in other cases, reprogramming the BLE chip is necessary.

2.3.4.2 Generic Provisioning Layer

As already explained, the different Provisioning PDUs used in the provisioning process are encapsulated in transactions, which can be segmented and reassembled. The Generic Provisioning Layer defines the different Generic Provisioning PDUs. The last two significant bits of the first byte of the Generic Provisioning PDU denote the Generic Provisioning Control Format (GPCF), which indicates its type. The corresponding values of the GPCF field for each PDU are shown in Table 2.1.

Table 2.1: Generic Provisioning Control Format values for each Generic Provisioning PDU.

Value	Generic Provisioning PDU
0b00	Transaction Start
0b01	Transaction Acknowledgment
0b10	Transaction Continuation
0b11	Provisioning Bearer Control

Transaction Start PDU

The Transaction Start PDU is used to start the transmission of a message. This can be a complete or a segmented message. Figure 2.16 illustrates the fields of a Transaction Start PDU. These fields are:

- SegN (6-bit size), which indicates the number of segments of the transaction. When it is a single-segment transaction, the SegN is zero.
- The GPCF value (2-bit size). That corresponding to Transaction Start PDU is 0b00.
- Total Length (2-octet size), in octets, of the complete Provisioning PDU, considering all segments.
- Frame Check Sequence (FCS) (1-octet size), calculated following the 3GPP TS 27.010 protocol [44] and considering the complete Provisioning PDU.

- The Data field of the Transaction Start PDU, which contains the segment 0 of the transaction.

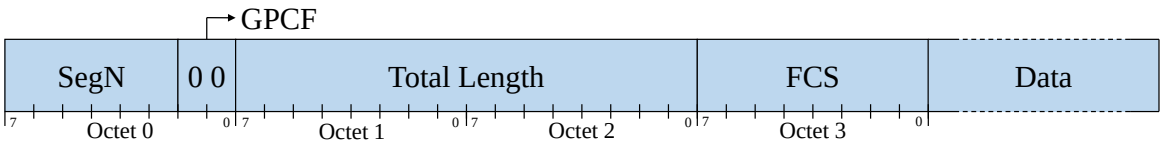


Figure 2.16: Transaction Start PDU.

Transaction Acknowledgment PDU

The Transaction Acknowledgment PDU is used to acknowledge an entire transaction, regardless of its number of segments. The fields of this PDU are shown in Figure 2.17, and they are:

- A 6-bit padding, which shall be 0b000000.
- The corresponding 2-bit GPCF, which is 0b01.

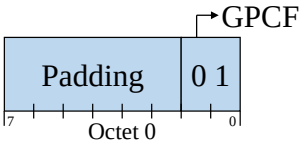


Figure 2.17: Transaction Acknowledgment PDU.

Transaction Continuation PDU

The Transaction Continuation PDU is used to transmit additional segments of a Provisioning PDU. When the length of a particular Provisioning PDU is greater than the length of the data field in the Transaction Start PDU, it is divided into segments. The first segment is sent in a Transaction Start PDU, while the rest of the segments are sent using Transaction Continuation PDUs. Figure 2.18 shows the fields of the Transaction Continuation PDU. These fields are:

- The Segment Index, a 6-bit field which indicates the position of the segment in the current transaction.
- The 2-bit GPCF corresponding to the Transaction Continuation PDU, which is 0b10.
- The Data field in which the segment is transported.

Provisioning Bearer Control

The Provisioning Bearer Control PDU allows sessions to be managed on bearers that have no inherent session management which is the case of the PB-ADV bearer. This PDU has the following fields, shown in Figure 2.19:

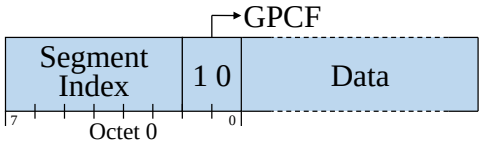


Figure 2.18: Transaction Continuation PDU.

- The Bearer Operation Code is a 6-bit field that indicates the operation code of the PDU. Different operation codes are defined in Bluetooth mesh specification: Link Open (0x00), Link ACK (0x01) and Link Close (0x02). All these operations are described in next sections.
- The 2-bit GPCF corresponding to Provisioning Bearer Control PDU, which is 0b11.
- A variable-length field whose parameters are related to the Bearer Operation Code of the PDU.

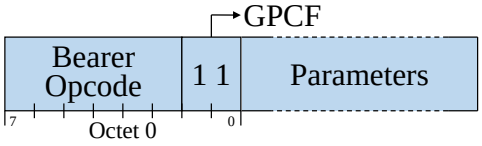


Figure 2.19: Provisioning Bearer Control PDU.

2.3.4.3 Provisioning Protocol

The complete provisioning protocol can be divided into seven phases, enumerated below. In case of using the GATT Bearer, which has an inherent session management, the steps related to the Link Establishment Procedure (steps 2 and 7) are not necessary. However, in that case, the establishment of a BLE connection between the provisioner and the unprovisioned device is necessary .

1. Beaconing.
2. Link Establishment Procedure: Link Open.
3. Invitation and Capabilities.
4. Exchange Public Keys.
5. Authentication and Confirmation.
6. Distribution of provisioning data.
7. Link Establishment Procedure: Link close.

In each of these phases, one or more messages with provisioning PDUs are interchanged between the provisioner and the unprovisioned device. Bluetooth mesh specification defines ten different types of provisioning PDUs:

1. Provisioning Invite.
2. Provisioning Capabilities.
3. Provisioning Start.
4. Provisioning Public Key.
5. Provisioning Input Complete.
6. Provisioning Confirmation.
7. Provisioning Random.
8. Provisioning Data.
9. Provisioning Complete.
10. Provisioning Failed.

Figure 2.20 illustrates the entire process, which is detailed below.

Beaconing

Beaconing or advertising process has been widely used in BLE for different applications. It is used by peripheral devices, which broadcast advertising packets with basic information, allowing a central device to establish a client-server connection [6]. This communication type is also used for BLE devices working as beacons. BLE beacons broadcast always the same packet, permitting other devices to know their approximate location, so they are widely used in solutions for indoor localisation. Advertising packets are also used by BLE devices to broadcast data among them. The Standard Bluetooth mesh [17] transmit and forward data in this way. However, in this phase of the provisioning process, it is used similarly to establish a point-to-point connection: BLE unprovisioned devices send packets containing relevant information to allow the provisioner to start the process.

When the PB-ADV is supported by an unprovisioned device, this sends specific advertising packets, known as unprovisioned device beacons. These beacons are used by a provisioner to discover unprovisioned devices. Figure 2.21 illustrates the advertising data field of an unprovisioned device beacon. It includes the following fields:

- 1-byte Mesh Beacon Length, which, similarly to other advertising types, indicates the length of the Advertising Data field.
- 1-byte Mesh Beacon ID, identifier corresponding to Mesh Beacon Type (0x2B).
- 1-byte Unprovisioned Device Beacon Type, one-byte field indicating that the beacon is transmitted by an unprovisioned device (0x00).
- 16-byte Device UUID, used to identify the beaconing device.

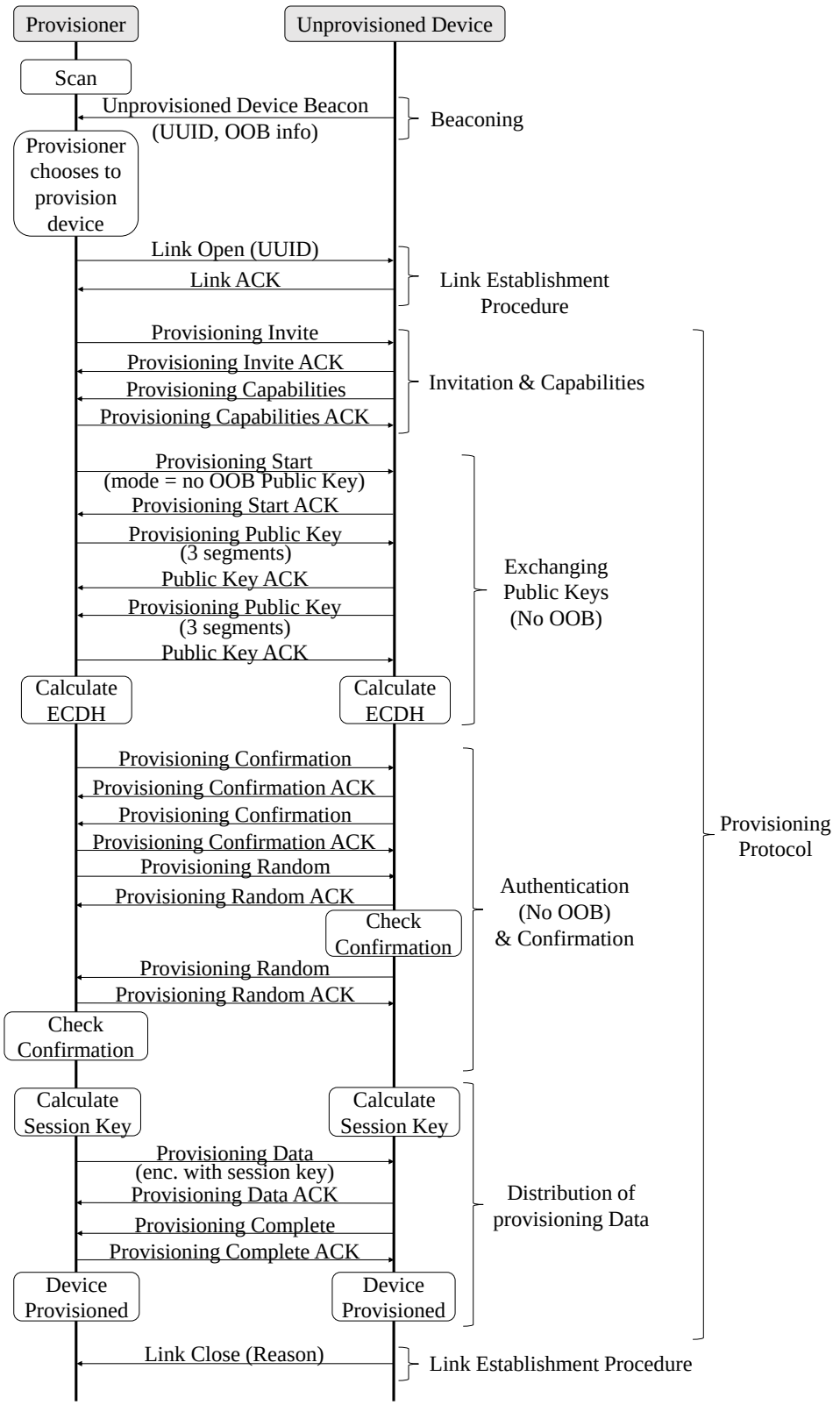


Figure 2.20: Complete Provisioning Protocol.

- 2-byte OOB field with information about the possibility of using OOB authentication in the provisioning process.
- 4-byte Uniform Resource Identifier (URI) Hash, field with the first 4 bytes of the result of the hash function to the URI that can be sent by the unprovisioned device. This URI relates to the OOB information, and the URI Hash field enables the provisioner to associate the URI with the received unprovisioned device beacon.

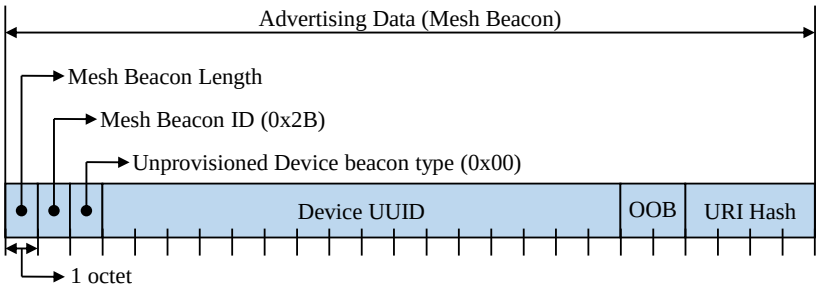


Figure 2.21: Unprovisioned Device Beacon.

If the unprovisioned device uses the PB-GATT bearer, the provisioning process is realised through a BLE GATT service denominated Mesh Provisioning Service. In the beaconing phase, the unprovisioned device broadcasts advertising packets including the identifier of this service, as well as the necessary data to identify the device itself. Figure 2.22 shows the Advertising Data field of an advertisement packet sent by an unprovisioned device. This is divided into three fields. As indicated in the BLE specification, the two first bytes of each of these fields indicates the length (one byte) and the advertising data type (one byte), which is defined by Bluetooth mesh specification. In this beacon packet, the following fields can be found:

- Flags, a 3-byte field that contains the length, the data type and the flag values.
- Complete List of 16-bit Service Class UUIDs. This 4-byte field includes the length, the data type and the value corresponding to the Mesh Provisioning Service (0x1827).
- Service Data 16-bit UUID, which contains the length, the data type and the value itself. This value is divided into:
  - Number assigned by the Bluetooth mesh specification to Mesh Provisioning Service: 0x1827.
  - 6-byte device UUID, which is used to identify the advertising device (see Section 3.10.3 in [17]).
  - 2-byte OOB information, which gives information to the provisioner about the possibilities of use OOB authentication in the provisioning process.

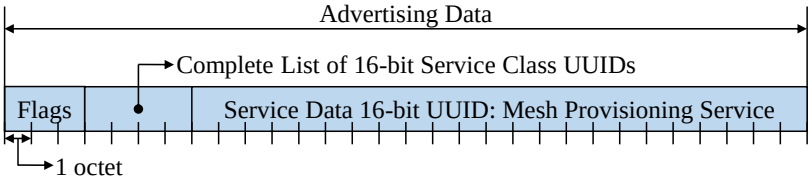


Figure 2.22: Advertising Packet sent by Unprovisioned Devices.

Link Establishment Procedure: Link Open

The Link Establishment procedure enables the provisioner to establish a link or session with an unprovisioned device. This session is established when the used bearer has no inherent session management, such as PB-ADV bearer. While this link is open, a randomly generated Link ID is used to identify the session, being included in every provisioning packet. This procedure allows the provisioner and unprovisioned devices to send provisioning messages avoiding interference between different sessions.

The provisioner shall scan the network for unprovisioned devices, which are sending unprovisioned device beacons. As explained before, these beacons identify devices through their Device UUIDs. When the provisioner receives a beacon from an unprovisioned device, it sends an Open Link message to establish the session. This message is encapsulated in a Generic Provisioning PDU (see Figure 2.23) with the recently created Link ID and the unprovisioned device UUID, and sent in a non-connectable advertising packet.

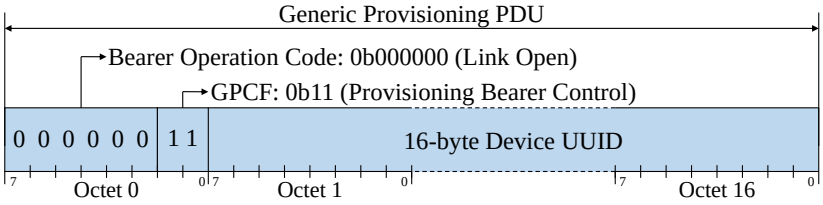


Figure 2.23: Generic Provisioning PDU for Link Open Message.

When a Link Open message is received by an unprovisioned device, unless it is already being provisioned, it shall accept the link establishment sending a Link ACK message, using the same Link ID. The Generic Provisioning PDU of this transaction is only one byte in length, and contains the GPCF (in the two less significant bits) and the Bearer Operation Code (in the six more significant bits). The GPCF is the same for every Provisioning Bearer Control (0b11), and the Bearer Operation Code for Link ACKs is 0b000001. This is illustrated in Figure 2.24.

Invitation and Capabilities

When the session between the provisioner and the unprovisioned device is established using PB-GATT bearer or the link establishment procedure in PB-ADV bearer, the provisioner sends a Provisioning Invite PDU to indicate to the unprovisioned device that the provisioning process is starting. This PDU has two fields: the one-octet Provisioning PDU Type (0x00

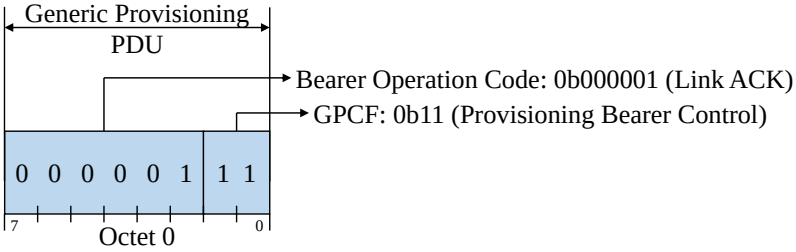


Figure 2.24: Generic Provisioning PDU for Link ACK Message.

for Provisioning Invite); and the one-octet field denominated Attention Duration. The Attention Duration indicates the Attention Timer, which determines, if enabled, how long the primary element of the unprovisioned device shall attract the human's attention in a human-recognisable way. After receiving the Provisioning Invite, the unprovisioned device should acknowledge it with a Transaction ACK.

Once the unprovisioned device has received the Provisioning Invite and acknowledged it, it sends a Provisioning Capabilities PDU. The length of this PDU is 12 bytes, and it includes different information about the unprovisioned device capabilities used in the next provisioning phases. All fields of this PDU type are enumerated below:

1. PDU Type (one-octet size), which for the Provisioning Capabilities PDU must be 0x01.
2. Number of Elements (one-octet size) supported by the device, required to calculate the range of addresses assigned to the unprovisioned device.
3. Algorithms (two-octet size), which refer to the security algorithms supported by the unprovisioned device. Currently, the only algorithm included in the Bluetooth mesh specification is the P-256 Elliptic Curve algorithm [45].
4. Public Key Type (one-octet size) indicates whether the device supports the use of the OOB mechanism to exchange the Public Keys.
5. Static OOB Type (one-octet size) notifies whether the device supports the use of static OOB information.
6. Output OOB Size (one-octet size) refers to the maximum number of digits when the Output OOB Action field is Output Numeric.
7. Output OOB Action (two-octet size) represents actions supported for Output OOB. The Bluetooth mesh specification defines the following actions: blink, beep, vibrate, numeric and alphanumeric.
8. Input OOB Size (one-octet size) indicates the maximum number of digits to be entered when the Input OOB Action field is Input Number.
9. Input OOB Action (two-octet size) refers to actions supported for Input OOB. Four different actions are defined in the Bluetooth mesh specification, which are: push, twist, input number and input alphanumeric.

All fields referring to OOB information are related to the Authentication phase, which is explained below. When the Provisioning Capabilities transaction is received, it should be acknowledged by the provisioner.

Provisioning Start and Exchanging Public Keys

The objective of this phase is to exchange the public keys between the provisioner and the unprovisioned device. These keys permit the calculation of the ECDH shared secret, which is used to exchange secret keys over an insecure channel.

In the Bluetooth mesh provisioning procedure, the two most popular encryption schemes are used: symmetric encryption (also called secret key encryption) and asymmetric encryption (or public key encryption), which are described briefly below:

- Symmetric encryption involves using the same secret key to cipher and decipher information. For this reason, both sender and receiver must know this secret key. One example of this type of encryption is AES [46]. AES-128 (AES with a key size of 128 bits) is used in the last phases of the Bluetooth mesh provisioning procedure and to encrypt Bluetooth mesh transmissions. This algorithm is efficient in both software and hardware, making it perfect for execution in embedded chipsets, which most BLE devices are based on. However, the main weakness of this type of algorithm is that it is difficult to securely exchange the shared secret key over an insecure channel.
- Asymmetric encryption uses a related key pair to encrypt and decrypt messages: the public key and the private key. The public key permits anyone to cipher a message, being freely available for all devices. The private key is kept secret, and allows the receiver to decrypt messages previously encrypted with the corresponding public key. The method leverages this to send public keys over insecure channels, because they are only used for encryption and not for decryption. However, the asymmetric encryption is slower than the symmetric one, and requires more processing power to both encrypt and decrypt messages. Asymmetric encryption is used in Bluetooth mesh provisioning to solve the symmetric encryption problem of securely exchanging secret keys. ECDH is an anonymous key agreement protocol based on asymmetric cryptography that allows two devices, each having an elliptic curve public-private key pair, to establish a shared secret over an insecure channel. Each device generates this shared secret key using the public key from the other device and its own private key, obtaining both devices exactly the same result.

Bluetooth mesh uses both symmetric and asymmetric encryption. The symmetric encryption algorithm AES-128 is used to encrypt and decrypt every mesh message using the Network Key, due to its low computational cost. The Network Key is distributed by the provisioner in the provisioning process. The Network Key transmission is performed securely over an insecure channel using asymmetric encryption: both the provisioner and the unprovisioned device generate a public-private key pair, exchanging the public keys. The provisioner always sends its public key in a provisioning message. However, there are

two different possibilities for the unprovisioned device: over a Bluetooth link or through an OOB tunnel, depending on whether or not the unprovisioned device supports this feature. Although the use of an OOB tunnel provides extra security, it is not a hard requirement, because only using the public key cannot compromise the network security, as explained before.

In the Provisioning Start and Exchanging Public Keys phase, the provisioner sends a Provisioning Start PDU to indicate the options it has selected from the Provisioning Capabilities PDU sent by the unprovisioned device. The Provisioning Start PDU has the following parameters:

1. PDU type (one-octet size), being the Provisioning Start PDU identifier 0x02.
2. Algorithm (one-octet size), which indicates the algorithm selected for provisioning.
3. Public Key Type (one-octet size), setting whether the Public Key of the unprovisioned device should be sent OOB.
4. Authentication Method (one-octet size), which selects the authentication method used in the next phase.
5. Authentication Action (one-octet size), determining the action associated to the Authentication Method if necessary.
6. Authentication Size (one-octet size), which determines the number of characters to complete the Authentication Action, if necessary.

When the unprovisioned device receives the Provisioning Start transaction, it sends the corresponding ACK. Afterwards, the public keys are exchanged. If the unprovisioned device uses OOB, the provisioner sends its public key over a Bluetooth link, while the unprovisioned device sends its public key via an OOB tunnel; if not, both public keys are sent through the BLE technology. These keys are transported using the Provisioning Public Key PDU, which has the following fields:

1. PDU type (one-octet size), being the type assigned to the Provisioning Public Key PDU 0x03.
2. Public Key X (32-octet size), which contains the X component (32 first bytes) of the public key.
3. Public Key Y (32-octet size), which contains the Y component (32 last bytes) of the public key.

Due to the total length of this PDU, it is divided into three different segments to be transmitted in Bluetooth mesh provisioning packets.

Once public keys have been exchanged, both provisioner and unprovisioned device calculate the shared secret key using the ECDH protocol. This protocol allows two devices to

calculate the same shared secret key each one using its own private key and the public key from the other device, as shown in the Equation 2.1:

$$ECDH\ Secret\ Key = P - 256\ (private\ key,\ peer\ public\ key) \tag{2.1}$$

where $P - 256$ represents the P-256 elliptic curve defined in [45].

Authentication and Confirmation

This phase has two different purposes: authentication of the unprovisioned device and the provisioning confirmation. For authentication, three different approaches are defined, depending on what OOB mechanism is used to authenticate the unprovisioned device: Output OOB, Input OOB and Static or No OOB,

Regardless of the authentication method, in the check confirmation value operation each device generates a 16-byte random number, known as *RandomProvisioner* ($Random_P$) and *RandomDevice* ($Random_D$). Each of these numbers is concatenated to the 16-byte *AuthValue* (the value input by the user if output OOB or input OOB method is used, static or zero value in other case). Finally, these 32-byte numbers are encrypted with the *ConfirmationKey* using the AES-CMAC algorithm, generating the *ConfirmationProvisioner* ($Confirmation_P$) and *ConfirmationDevice* ($Confirmation_D$), respectively (see Figure 2.25), as shown in Equation 2.2 and Equation 2.3:

$$Confirmation_P = AES - CMAC_{ConfirmationKey}(Random_P||AuthValue) \tag{2.2}$$

$$Confirmation_D = AES - CMAC_{ConfirmationKey}(Random_D||AuthValue) \tag{2.3}$$

The AES-CMAC [47] is a keyed hash function based on a symmetric key block cipher, such as the AES. It provides stronger data integrity than a checksum or an error-detecting code, since CMAC is designed to detect intentional and unauthorised modifications of the data, as well as accidental modifications, while a checksum or an error-detecting code only allows us to detect accidental modifications of the data. In this phase of the provisioning process, AES-CMAC is used to generate the $Confirmation_P$ and the $Confirmation_D$, encrypting the $Random_P$ and $Random_D$, respectively, with the *ConfirmationKey*, as well as to obtain this key.

The *ConfirmationKey* is generated using several rounds of AES-CMAC and SALT algorithms. The AES-CMAC is presented below, while the SALT algorithm is an AES-CMAC using a 128-bit ZERO key. As seen in Figure 2.25, the *ConfirmationKey* is generated as shown in Equation 2.4:

$$ConfirmationKey = AES - CMAC_{Key}(\text{"}prck\text{"}) \tag{2.4}$$

where *"prck"* is the string used as input value, while the Key is computed as shown in Equation 2.5:

$$Key = AES - CMAC_{Confirmation\ Salt}(ECDH\ Shared\ Secret) \tag{2.5}$$

and the *Confirmation Salt* is calculated using the Equation 2.6:

$$Confirmation\ Salt = AES - CMAC_{Zero\ Key}(Confirmation\ Inputs) \tag{2.6}$$

being the *Zero Key* a 128-bit value where all bits are 0, and the *Confirmation Inputs* the concatenation of the previous provisioning PDU values, as well as both provisioner and device public keys.

Note that the $ConfirmationKey$ is calculated using a double AES-CMAC process, where the output obtained in the first calculation is used as key in the second operation. This operation is called k1 function in Bluetooth mesh specification.

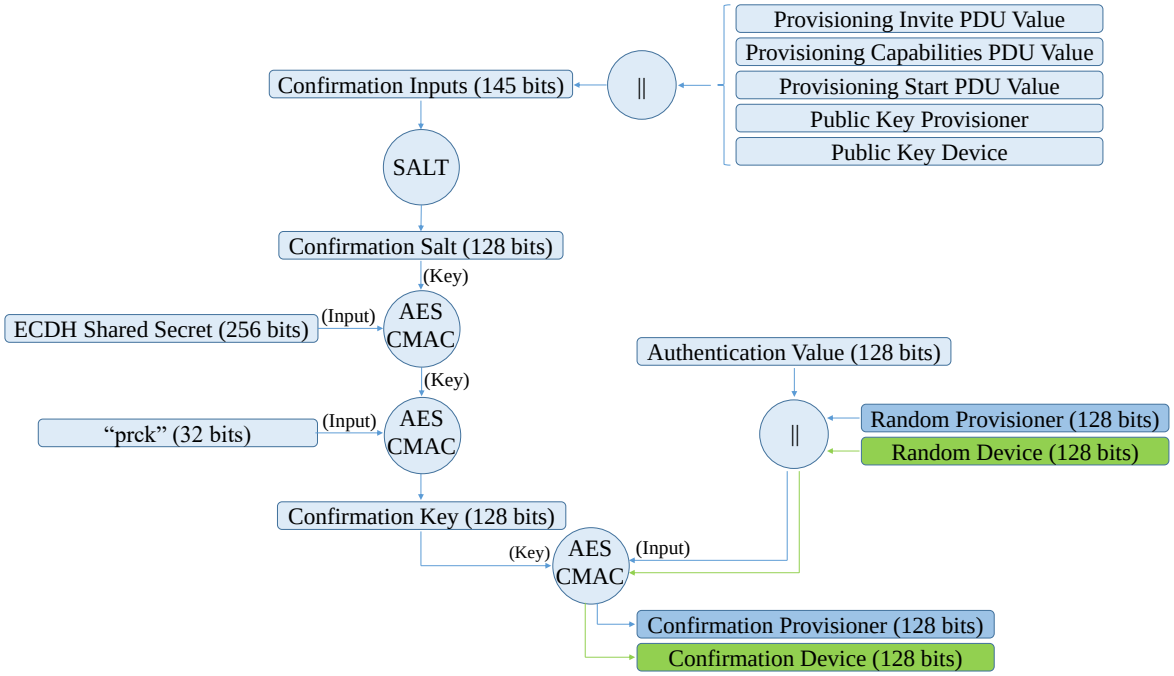


Figure 2.25: Confirmation Provisioner and Confirmation Device generation flowchart.

Once the Confirmation Values are calculated, the provisioner and unprovisioned device exchange them using Provisioning Confirmation PDUs. These PDUs have two different fields:

- PDU type (1-octet size), being the type assigned to the Provisioning Confirmation PDU 0x05.
- Confirmation value (16-octet size).

Subsequently, the provisioner sends its *random number*, the same one used to generate the $Confirmation_P$. This allows the unprovisioned device to recalculate the $Confirmation_P$, and to verify whether it matches with the $Confirmation_P$ previously received. If the two values are not the same, the unprovisioned device aborts the provisioning process sending a Provisioning Failed message. In case of obtaining the same values, the unprovisioned device sends its original *random* value.

When the provisioner receives the *random number* from the unprovisioned device, it repeats the confirmation checking, aborting the process if the recalculated *confirmation device* value does not match with the previously received.

The random values are transmitted in Provisioning Random PDUs, which have two different fields:

- PDU type (1-octet size), being the type assigned to this PDU 0x06.
- Random number (16-octet size).

Distribution of Provisioning Data

Once the previous steps are complete, the bearer is secure enough to continue with the most important step in the provisioning process: deriving and distributing the provisioning data. The provisioning data is generated by the provisioner, and contains the following fields, all of which are necessary to include the unprovisioned device in the mesh network:

- The Network Key (16-octet size) is shared across all nodes in the network, securing communications at the network layer. Possession of a Network Key is the requirement for a node to take part in the Bluetooth mesh network, enabling an unprovisioned device to become a mesh node. For that reason, the secure transmission of the Network Key is one of the primary objectives of the provisioning process.
- Device Key (16-octet size), a security key generated in the provisioning process. This key is possessed by only the provisioner and the corresponding mesh node.
- Key Index (2-octet size), used to identify the network keys shortly, allowing these keys to be managed more efficiently given their length.
- Flags (1-octet size), bitmask used to indicate the current status of the associated key or IV.
- IV Index (4-octet size), which provides entropy when message nonces are calculated.
- Unicast Address (2-octet size) assigned to the primary element of the provisioned node.

The provisioning data is encrypted using the Advanced Encryption Standard - Counter with Cipher Block Chaining Message Authentication Code (AES-CCM) algorithm [48] to distribute it securely. CCM is a generic authenticated encryption block cipher mode that is defined to be used with 128-bit block ciphers such as AES-128.

The AES-CCM uses two additional parameters, known as *Session Key* and *Session Nonce*. These are generated in the provisioner and the unprovisioned device (allowing the provision data to be encrypted and decrypted), as well as the *Device Key*, following the same flowchart, illustrated by Figure 2.26.

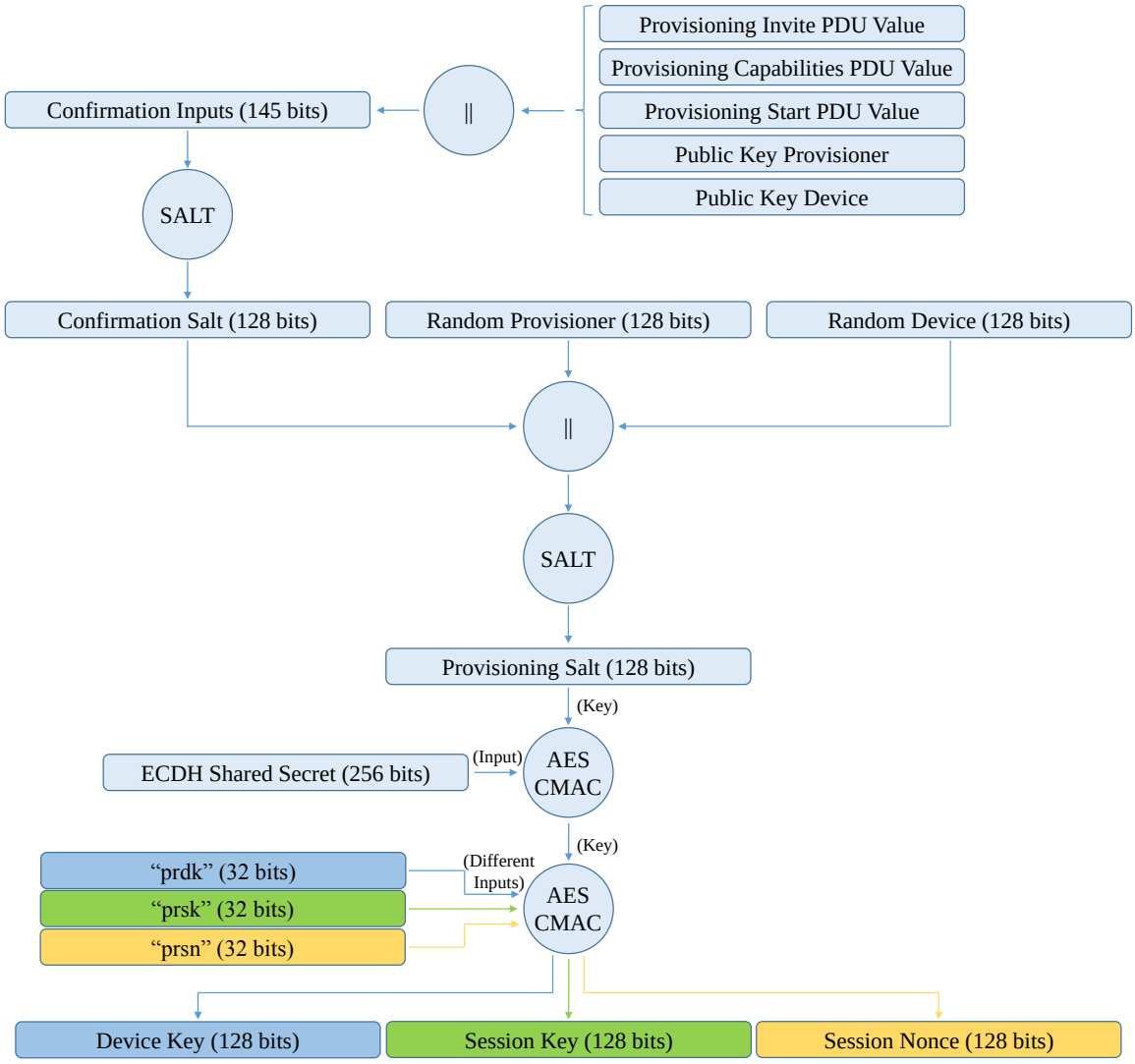


Figure 2.26: Device Key, Session Key and Session Nonce generation flowchart.

As Figure 2.26 shows, *Device Key*, *Session Key* and *Session Nonce* are generated in the same function, depending on the last input, as shown in Equation 2.7 (*Device Key*), Equation 2.8 (*Session Key*) and Equation 2.9 (*Session Nonce*).

$$Device\ Key = AES - CMAC_{Key}(\text{"}prdk\text{"}) \tag{2.7}$$

$$Session\ Key = AES - CMAC_{Key}(\text{"}prsk\text{"}) \tag{2.8}$$

$$Session\ Nonce = AES - CMAC_{Key}(\text{"}prsn\text{"}) \tag{2.9}$$

Where the Key is generated using the ECDH Shared Secret, as shown in Equation 2.10:

$$Key = AES - CMAC_{Provisioning\ Salt}(ECDH\ Shared\ Secret) \tag{2.10}$$

Finally, the *Provisioning Salt* is the result of the *Salt* function, using the concatenation of $Confirmation\ Salt$, *Random Provisioner* and *Random Device* values as input, as shown in Equation 2.11 and Equation 2.12:

$$input = Confirmation\ Salt||Random\ Provisioner||Random\ Device \tag{2.11}$$

$$Provisioning\ Salt = AES - CMAC_{ZeroKey}(input) \tag{2.12}$$

where the *Confirmation Salt* is the result of the *Salt* function of the *Confirmation Inputs*, which are the concatenation of different values previously used in the provisioning process, as explained in the previous phase.

After calculating these values, the provisioner sends the encrypted provisioning data in a Provisioning Data PDU, which includes the following fields:

- PDU Type (1-octet size), being the Provisioning Data PDU type 0x07.
- Encrypted Provisioning Data (25-octet size), which contains the provisioning data, including the network key, the network key index, flags, the IV Index and the assigned unicast address of the primary element, all encrypted using the AES-CCM algorithm with the *Session Key* and *Session Nonce* calculated.
- Provisioning Data MIC (8-octet size), used by the unprovisioned device to authenticate and check the integrity of the Provisioning Data.

The Provisioning Data PDU is divided into two segments. When the unprovisioned device receives both segments, it acknowledges the message. After that, the unprovisioned device sends a Provisioning Complete PDU, which shall also be acknowledged by the provisioner to indicate that the provisioning data has been successfully received and processed. The Provisioning Complete PDU has only a 1-octet size field, the PDU Type, which is 0x08.

Link Establishment Procedure: Link Close

Finally, when the entire provisioning procedure is finished, the session opened using the Link Establishment Procedure is closed using a Link Close message. However, this message is not only used for successful provisioning, and can be sent to indicate that the provisioning

transaction timed out or that the process failed. The link can be closed at any time after the link establishment sending this packet.

A Link Close message is also encapsulated in a transaction, and its Generic Provisioning PDU contains two different parameters. The first byte indicates the type, containing the GPCF (0b11, corresponding with the Provisioning Bearer Control) in the two less significant bits, and the Bearer Operation Code (0b000010, related with the Link Close) in the six more significant bits. The second byte refers to the reason why the process has been closed, and it refer to: a successful provisioning, a timed out provisioning transaction and a failed provisioning.

In contrast to Link Open message, the Link Close message is not acknowledged, so it is sent at least three times to ensure a correct reception.

2.4 Conclusions

This chapter covered the background to this Doctoral Thesis. The first section presented the new trends of IoT and Industry 4.0, as well as the relationship between them. These trends are changing the requirements of communication networks, so these requirements have been emphasised. Thus, the main challenge for wireless communication networks is to fulfil the requirements of these trends: zero failure, total coverage and sustainability.

After that, the chapter provided an in-depth description of the communication standards we have worked with: BLE and Bluetooth mesh. BLE was the proposal of Bluetooth for low consumption networks. However, despite the good performance of BLE in small and medium scale networks, the requirements of Industry 4.0 demanded different network topologies. Therefore, in order to adapt to these requirements, the Bluetooth mesh specification was released. This specification provided BLE with the mesh topology, so widely demanded in these types of applications

CHAPTER 3

Preliminary Evaluation: BLE Topologies for Industry 4.0

This chapter presents a preliminary study conducted with different BLE topologies, with the aim of determining whether they are suitable for the new challenges of Industry 4.0. To this end, different experiments were carried out to evaluate the two ways of sending information specified by the BLE standard: establishing a point-to-point connection or performing broadcast transmissions. With the launch of the CSR (Cambridge Silicon Radio, now Qualcomm Technologies International since acquired by Qualcomm) proprietary mesh BLE topology, the CSRmesh, a new evaluation was carried out that allowed us to determine the strengths and weaknesses of this proposal, verifying whether it fulfilled all the requirements imposed by the new trends in IoT and IIoT.

This chapter first evaluates the topologies included in the BLE specification and then the evaluation of the CSRmesh proposal is presented.

3.1 BLE Standard Topologies Evaluation

This section presents the experiments carried out with the BLE standard topologies: point-to-point connection and broadcast transmissions. This will give us a more complete picture of how the BLE standard works, and enable us to choose the best option according to the case of use. Waspmote devices (see Section A.1.1) were used to conduct the evaluation in this section.

3.1.1 Coverage Range of Waspmote Devices

One of the first questions to be answered when talking about wireless network devices is their range. Therefore, this section begins with a short evaluation of the range of the devices used. The maximum distance at which two devices can transmit data is given by the Transmission (TX) power and the sensitivity of the antenna, so this section has been

divided into two parts. The first part presents the evaluation carried out for two Waspmote devices, while the second evaluation was carried out using a Waspmote device and a mobile device, taking advantage of the excellent compatibility of BLE with this type of device.

3.1.1.1 Maximum Distance between Two Waspmote Devices

Two Waspmote devices were used to carry out this evaluation with three transmission power settings: minimum (-23 dBm), medium (-10 dBm) and maximum (+3 dBm). Each experiment was conducted 5 times. In each experiment, the distance between the two devices was measured for different Received Signal Strength Indicator (RSSI) ranges in line of sight, that is, with no obstacles between them. Table 3.1 shows the average of the measures of the different TX powers according to the different RSSI intervals.

Table 3.1: Maximum distances (in metres) for different RSSI between two Waspmote devices.

RSSI	Minimum TX power	Medium TX power	Maximum TX power
Up to -40 dBm	0.1	0.2	1.5
Up to -60 dBm	4	7	32
Up to -80 dBm	23	33	175
Up to -100 dBm	40	110	350

To provide a context for the results obtained, the BLE specification recommends establishing a connection with an RSSI signal greater than -60 dBm, although up to -80 dBm may be acceptable. The results show that the increase in the transmission power implies an increase in the range of the devices. However, it is necessary to find a balance between the necessary distance and the established TX power, since the greater the TX power, the greater is the battery consumption. Furthermore, if our network consists of several BLE devices, using too much power will increase the possibility of collisions and interference between packets, reducing the performance of the network.

3.1.1.2 Maximum Distance between a Waspmote Device and a Mobile Device

One of the most outstanding features of BLE is the possibility of transmitting data to mobile devices such as smartphones or tablets, which have become everyday devices for a large part of the population. However, these devices use different antennas to the Waspmote devices, with greater sensitivity. For this reason, it was proposed to repeat the previous evaluation, but varying the receiver device. On this occasion the receiving device was an HTC Nexus 9 Tablet [49]. Table 3.2 shows the results obtained in this experiment.

This experiment was carried out following the same steps as in the previous case: each test was carried out five times, with the final result being the average of the results obtained. Similarly, a distinction was established between the RSSI of the received packets, with up to -80 dBm being the acceptable range for establishing a connection and transmitting data.

Table 3.2: Maximum distances (in metres) for different RSSI between a Waspmote device and a mobile device.

RSSI	Minimum TX power	Medium TX power	Maximum TX power
Up to -40 dBm	0	0.1	1
Up to -60 dBm	0.6	1	6.5
Up to -80 dBm	2.5	4	19
Up to -100 dBm	23	82	215

The results show, in all cases, a shorter distance than in the previous case, where two Waspmote devices were used. This is due to the difference between the antennas, as the one used by the Waspmote devices has greater sensitivity than the one used in mobile devices, which results in a greater range when sending and receiving data.

3.1.2 Packet Received in Broadcast Transmissions

As explained above, BLE provides the possibility of deploying two types of networks, depending on whether or not a connection is established between the devices. In the case of deploying a connectionless network, the devices known as broadcasters send broadcast messages that will be received by devices known as observers. This type of network has no restrictions on the number of devices, although it does not provide the mechanisms in the specification to send messages in a secure and/or acknowledged way (which can be implemented over the BLE protocol).

This experiment was conducted to evaluate the PRR of these connectionless transmissions. In this type of network, a number of parameters determine its performance, namely:

- Advertising Interval: time between two consecutive advertising or broadcast packets.
- Scan Interval: time between the start of two consecutive Scan Windows.
- Scan Window: time the device scans the network for packets that are being broadcasted.

In order to carry out this experiment, these parameters were set close to the extremes, to appreciate their behaviour in these cases. In addition, different experiments were carried out varying the number of advertising devices, thus verifying the influence of this number on the results obtained.

Each of the experiments was carried out three times (showing the average of the three executions). The duration of each execution was 3000 seconds (50 minutes), except for the tests where the Advertising Interval was 20 ms, which had a duration of 300 seconds (5 minutes). This was because the packets received with eight broadcast devices in a longer time were not easy to handle.

The results obtained for each of the experiments in this section are shown below. In all these experiments, one node was in scanning mode, while the rest, whose number varied

between 1, 2, 4 and 8, were in advertising mode. Figure 3.1 shows the percentage of packets received for an Advertising Interval of 20 ms; Figure 3.2 presents the results obtained for an Advertising Interval of 1 s; in Figure 3.3 the percentage obtained for an Advertising Interval of 5 s can be seen; and finally, Figure 3.4 shows the percentage for the case in which the Advertising Interval was 10 seconds.

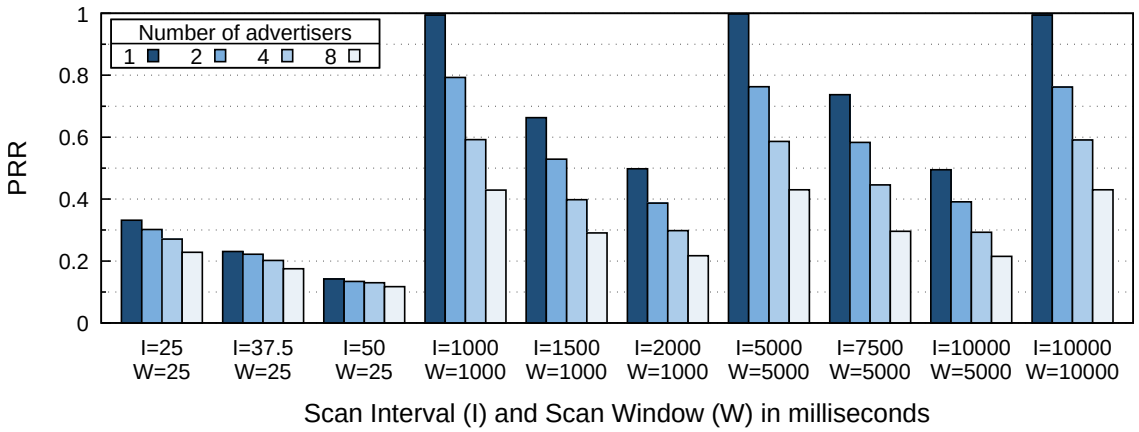


Figure 3.1: PRR for an observer receiving packets from a broadcaster transmitting with an Advertising Interval of 20 ms.

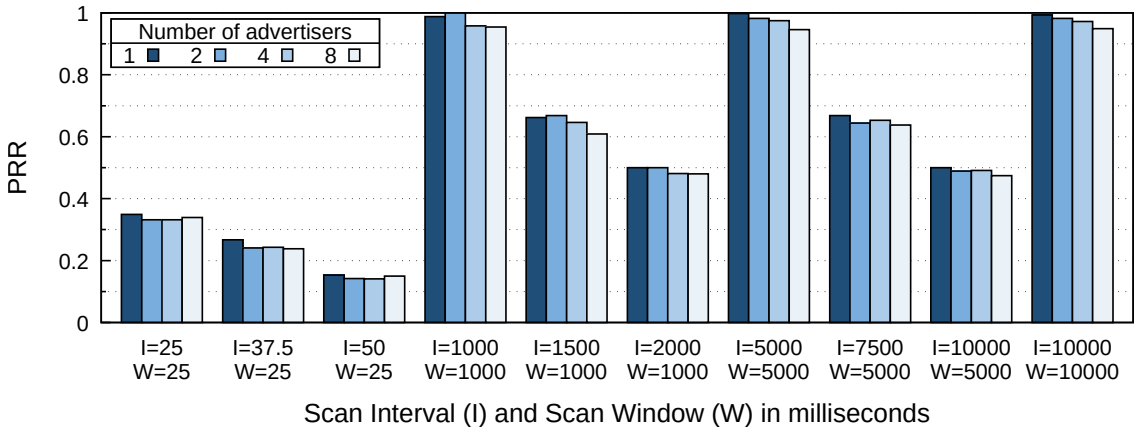


Figure 3.2: PRR for an observer receiving packets from a broadcaster transmitting with an Advertising Interval of 1 s.

Based on the results obtained, the following conclusions can be drawn:

1. When the Advertising Interval of 20 ms was used (see Figure 3.1), the scanner was able to capture a great number of advertising packets, as the number of packets sent was huge. However, the PRR obtained was low. In addition, a decrease in the PRR can be appreciated when the number of advertising devices increases, reaching around 40% for eight advertisers, although the Window and Scan Intervals were the same (better results). This corresponds to what is described in the BLE specification about the Advertising Interval:

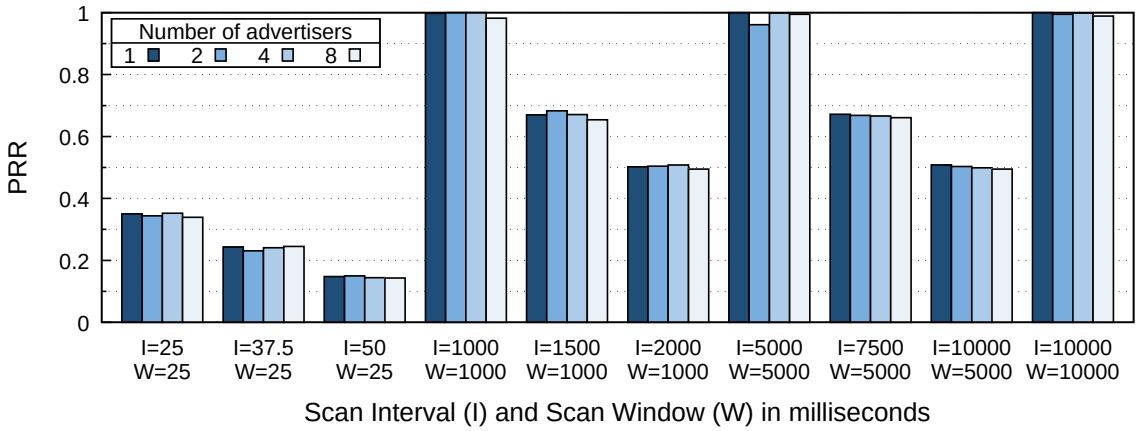


Figure 3.3: PRR for an observer receiving packets from a broadcaster transmitting with an Advertising Interval of 5 s.

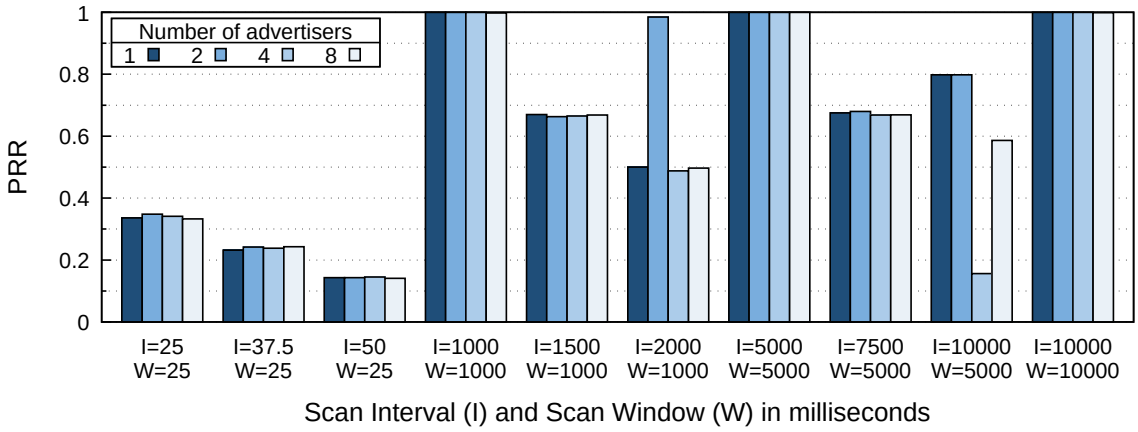


Figure 3.4: PRR for an observer receiving packets from a broadcaster transmitting with an Advertising Interval of 10 s.

- The Advertising Interval for data transmissions should be at least 100 ms. The results support this, since a greater frequency of sending reduces the percentage of packets received by the scanner.
- The Advertising Interval for devices in connectable mode must be less than or equal to 20 ms. This allows the connection to be established more quickly, and the packet loss is unimportant, since all these packets are identical and do not contain user information or data.

2. The pseudo-random delay introduced by the BLE standard between each of the consecutive advertisement events must be taken into account. This delay has a duration ranging from 0 to 10 ms, and stands out in two cases in particular:
   - When the Advertising Interval was 20 ms, (see Figure 3.1), since the delay could be as long as half of the set interval.

- When the Advertising Interval was 10 s (Figure 3.4), since, if the Scan Window does not coincide with the Scan Interval, this delay can affect the number of packets received by the scanner, increasing or reducing the percentage of packets received by the scanner. Without this delay, extreme results where all or no packets were received in the scan window could be obtained. However, although slow, this delay varies the time of sending a packet, which means that, in the long term, the percentage of packets received by the scanner is directly proportional to the relation Scan Window - Scan Interval. In order to verify this, longer-term experiments were carried out for those cases where the results obtained stand out from the rest. The duration of each of these experiments was 20000 seconds (more than 5.5 hours) and the results obtained are shown in Figure 3.5, comparing them with the results previously obtained. The results show that when the duration of the experiments was increased, the percentage of received packets was around 50%, coinciding with the relation Scan Window - Scan Interval.

3. When the shortest Scan Interval and Scan Window parameters were used (between 2.5 and 5 ms, corresponding to the first three columns of Figure 3.1, Figure 3.2, Figure 3.3 and Figure 3.4), the ability to receive the packets sent by the advertisers decreased, obtaining a low percentage regarding the total. The explanation for this situation can be found in how the advertisers sent their packets. By default, the packets are sent using the three existing advertising channels (37, 38 and 39), sending the same packet through the different channels in time intervals shorter than 10 ms. The scanners switch between the different advertising channels at each Scan Interval. When the Scan Interval is wide enough to receive the packet sent on the channel being listened at that time, the packet is received. However, if the Scan Interval does not cover this minimum duration, the percentage of received packets can drop drastically, as shown in the results obtained.

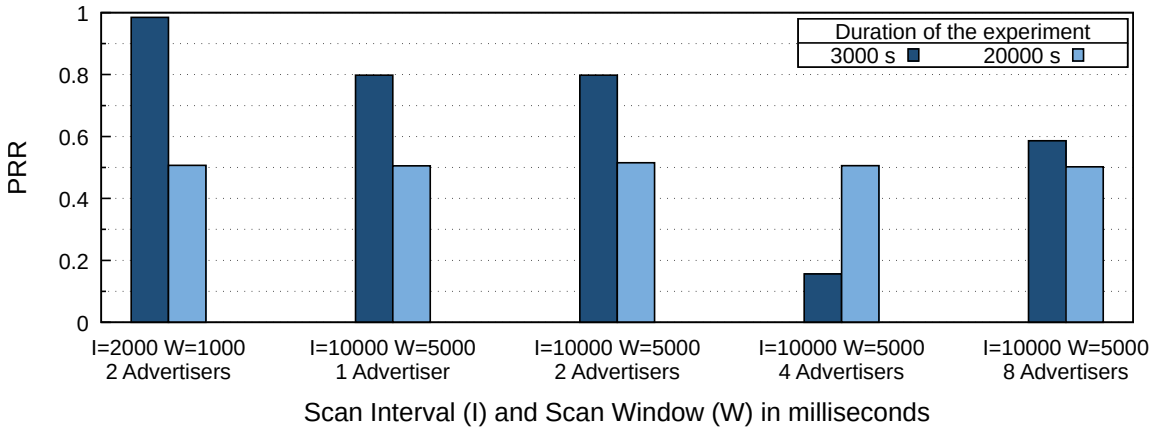


Figure 3.5: Comparison of experiments according to their duration.

3.1.3 Path Through a BLE Broadcast Network

A preliminary study was carried out to determine whether the broadcast network topology was suitable for new Industry 4.0 challenge. For this study, Waspmote devices with a BLE radio module were used to deploy our BLE network.

A 10-BLE-device broadcast network was deployed in our institute, the Albacete Research Institute of Informatics (I3A) [50]. BLE devices used the broadcasting capability for data transmissions. The static BLE devices transmitted packets as broadcasters, and an user equipped with a mobile device (which is a observer) received these packets in different points while moving through the building (from P1 to P16 in Figure 3.6, Figure 3.7 and Figure 3.8). The user was equipped with the HTC Nexus 9 tablet [49], for which an application was developed using the Apache Cordova framework [51] (see Section A.2.2) that allowed the number of packets received from each device to be measured, as well as its RSSI. The PRR and the RSSI of each packet were measured in different points of the user's walk, which showed us whether the deployed network worked correctly. This experiment was carried out for three different configurations for Waspmote devices, varying the TX power and the Advertising Interval: (1) minimum TX power and 1-second Advertising Interval (Figure 3.6), (2) medium TX power and 1-second Advertising Interval (Figure 3.7) and (3) minimum TX power and 0.5-second Advertising Interval (Figure 3.8).

Figure 3.6, Figure 3.7 and Figure 3.8 show the situation of the static BLE broadcasters (D1, ..., D10) and Testing Points (P1, ..., P16). The figures also show in which points the greatest RSSI corresponds to packets from the nearest device (in green) and in which ones the higher RSSI received packets do not come from the nearest device, or the observer has not received any packet from this nearest static device (in yellow).

Figure 3.6 shows the results obtained for the experiment carried out using a minimum TX power and an Advertising Interval of 1 second. As the results show, only in 11 of the 16 (68.75%) testing points did the greatest RSSI correspond to packets from the nearest device. In the remaining points, the packets with the greatest RSSI corresponded to other devices:

- P4: the RSSI for the D8 device (-78 dBm) was greater than that for the D3 device (-83 dBm). However, the number of packets received from D3 was higher (3 against 1).
- P5: the RSSI of device D5 (-85 dBm) was greater than that of device D4 (-88 dBm). In addition, the number of packets received from D5 was also higher (2 against 1).
- P6: the RSSI of device D7 (-79 dBm) was greater than that obtained for devices D4 (-84 dBm) and D5 (-93 dBm), which should theoretically have been higher due to the absence of obstacles.
- P7: the RSSI obtained from device D5 (-87 dBm) was not the greatest, although more packets were received (3 compared to 1) than from the rest of the devices (D4 and D6).

- P12: at this point the proximity of the D3 device to the user stood out, from which, despite the existence of an obstacle between the user and this device, the greatest number of messages were received (3). The highest RSSI corresponded to device D10 (-80 dBm).

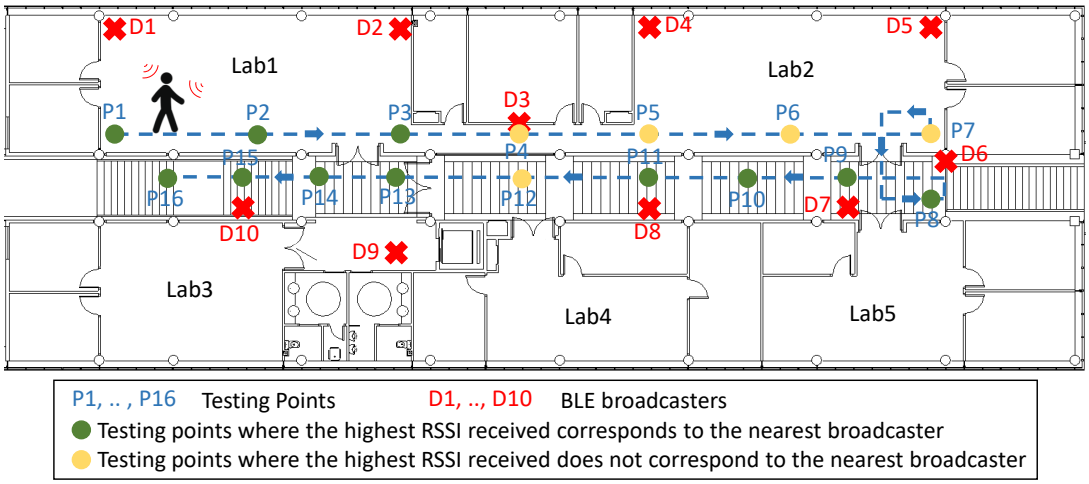


Figure 3.6: Results obtained in preliminary test to check the performance in a BLE IoT installation with minimum TX power and 1-second Advertising Interval.

Figure 3.7 shows the results obtained using a medium TX power and an Advertising Interval of 1 second, to determine the possible impact of the transmission power of the devices. As can be observed, again, only 11 of the 16 testing points obtained the expected result (68.75%), while in the rest the highest RSSI corresponded to other devices:

- P6: the RSSI for devices D3 (-69 dBm), D6 (-77 dBm) and D8 (-81 dBm) was greater than that for devices D4 (-73 dBm) and D5 (-85 dBm), and there was no significant difference in the number of packets received from each of them.
- P7: the greatest RSSI was for devices D4 (-64 dBm) and D6 (-69 dBm), although the closest device in line of sight was D5 (-78 dBm). In addition, the number of packets from each device was similar.
- P9: the RSSI of the nearest device (D7) obtained at this point (-70 dBm) was lower than that of devices D4 (-66 dBm) and D5 (-70 dBm), with the number of received packets being similar for all of them.
- P10: although the closest devices with line of sight to this point were D7 (-85 dBm) and D8 (-81 dBm), being located in the centre of a large number of devices meant that a better RSSI was received from devices D1 (-74 dBm), D2 (-77 dBm), D3 (-76 dBm), D4 (-75 dBm), D5 (-74 dBm) and D6 (-78 dBm).
- P12: the RSSI of the D9 device was very high (-59 dBm) and, even that of the D3 device was acceptable (-76 dBm), despite being on the other side of a sheet of glass.

However, regarding D8 (-82 dBm) a high RSSI was not received, with that of devices such as D1 (-72 dBm), D2 (-73 dBm) or D4 (-74 dBm) being greater.

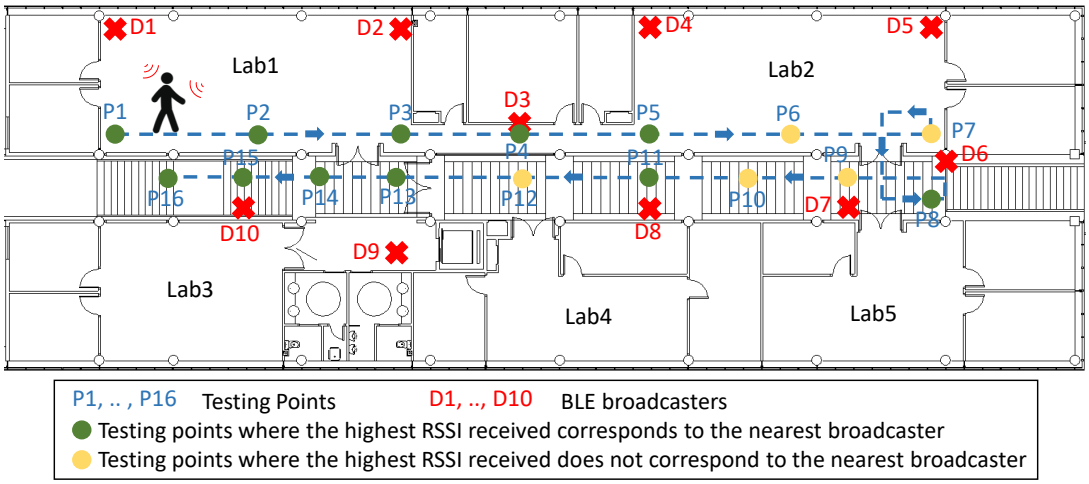


Figure 3.7: Results obtained in preliminary test to check the performance in a BLE IoT installation with medium TX power and 1-second Advertising Interval.

Figure 3.8 shows the results of the experiment carried out with a minimum transmission power and an Advertising Interval of 0.5 seconds, in order to determine the impact of the Advertising Interval of the broadcasters. As shown, on this occasion again only 11 of the 16 (68.75%) testing points were as expected, while, in the rest, the greatest RSSI corresponded to other devices:

- P5: the RSSI corresponding to device D5 (-72 dBm) was slightly greater than that of D4 (-76 dBm), which was the closest device.
- P6: this point was located between devices D4 and D5, from which an RSSI of -88 dBm and -76 dBm respectively were obtained, with D4 being below the RSSI of D7 (-87 dBm).
- P8: in this case, for the D5 device a slightly greater RSSI (-75 dBm) than that of D6 (-76 dBm) was obtained, but the number of packets from D6 (6) was significantly higher than that from D5 (2).
- P12: this point was between devices D8 (-78 dBm) and D9 (-96 dBm), but despite the proximity of both, there were devices with a greater RSSI: D7 (-88 dBm), D10 (-86 dBm), and even D3 (-87 dBm), with an obstacle (glass) between them.
- P13: the closest device to this point was D9, but its RSSI was lower (-81 dBm) than that of other devices, such as D1 (-76 dBm) or D10 (-77 dBm).

As the results of this section shows, only in 68.75% of the testing points did the greatest RSSI correspond to packets from the nearest device. These results leads us to conclude that

this topology does not fulfil the requirements of the Industry 4.0 identified above: there were several dead zones where without network coverage.

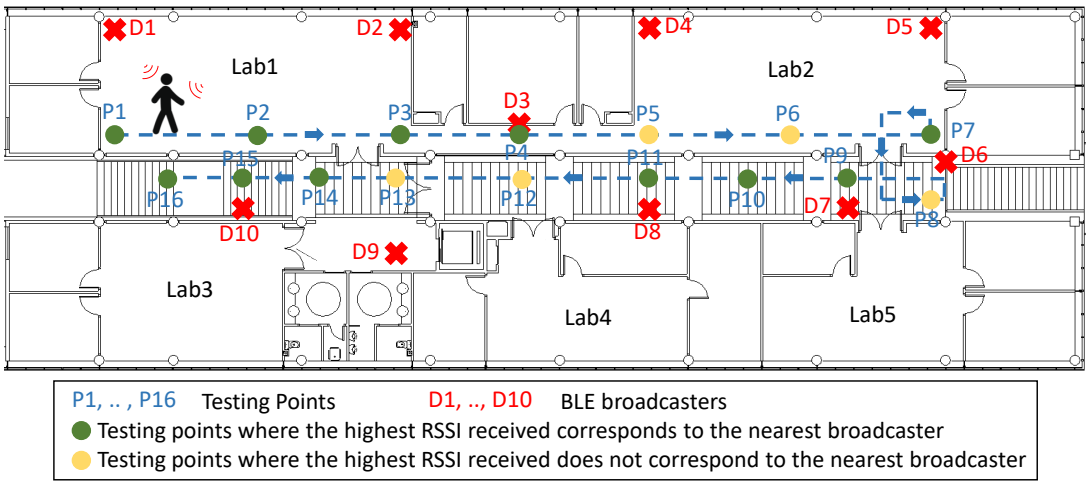


Figure 3.8: Results obtained in preliminary test to check the performance in a BLE IoT installation with minimum TX power and 0,5-second Advertising Interval.

3.1.4 Time Required to Establish a Point-to-Point Connection

BLE also provides another way of transmitting data, by establishing a point-to-point connection. Although these connections became more flexible in later versions, BLE 4.0 allowed only one master device in the network, which could connect to a certain number of slave devices. The maximum number of slaves was determined by the hardware of the master device, and was usually up to eight devices. The master device is responsible for reading and writing information to the slaves.

However, despite providing greater security, this type of transmission requires previous establishment. In static networks, this establishment is done only once, so the time required by this establishment is not critical. However, in networks with mobile devices, if these devices have to establish a new connection every time they leave the coverage range of a device, the time required can be crucial, preventing the information exchange. Therefore, this experiment was conducted to measure the time required to establish a connection.

The experiment presented in this section measured the time required to establish a connection between a master and eight slave devices (maximum number for the used devices). In addition to the time spent establishing the connection itself, the time needed for the device playing the role of master to detect the other slave devices was also taken into account. In other words, this discovery time was the time needed for the scanner device (which played the role of the master) to receive at least one advertising packet from the advertising devices (which played the role of slaves). This discovery process can be performed in three different ways, which are detailed below:

1. Defining the network scan time. This time has been set to 1 second, during which the initiator device scanned the network to discover devices with which to establish a connection. This time is the minimum supported by the Waspmote Application Programming Interface (API), and was enough to find eight advertisers sending their advertising packets with the minimum possible Advertising Interval, 20 ms.

2. Establishing the number of devices to be detected. In this case, the number of devices the initiator should detect before stopping the scanning process was defined. This number was set to eight, as it was the maximum number of simultaneous slaves that a single master could support.

3. Discovering the devices sequentially, one after the other. On this occasion the initiator discovered and connected to a single advertiser. Once the connection was established, the process of discovery and connection was repeated for another advertiser device, until the limit of eight simultaneous slaves was reached. Although in the two previous cases an individual check on each advertiser device was recommended before connecting to it (to check that it continued to broadcast), in this case the check could be removed, since the time between the discovery of a device and its connection was minimal.

Regarding the important parameters in this experiment, the Advertising Interval recommended by the standard (20 ms) was set to facilitate the initiator device detecting the advertising devices. In addition, a Scan Interval and a Scan Window of one second were set on the initiator device (due to the excellent results obtained in the previous test). Finally, the parameters corresponding to the connection were set to the default values assigned by the Waspmote API. These values were as follows: the Minimum Connection Interval was set at 70 ms, the Maximum Connection Interval was set at 90 ms, the Timeout was set at 1 second and the Slave Latency was set at 0 packets.

Figure 3.9 shows the results obtained in the experiments for each of the scanning methods. The results shown are the average values obtained after five executions, all of them being similar. The results show that the shortest total time was obtained when the discovery of devices was done individually and without checking the device again (which did not make sense, as it had just been discovered). By defining a time or number of devices in advance, the discovery time was visibly shorter. However, in these cases it is highly recommended to check the individual device before sending the connection request. This is due to the long time between the reception of an advertisement packet from a particular device and the establishment of a connection with that device (even more in real use cases), and it is possible that the device is no longer available (especially in mobile networks), producing errors in the connection attempts.

Consequently, in real cases where the devices to be connected are unknown and can quickly move and disappear from the coverage range, it is recommended to connect to them using multiple individual connections, exchange the necessary data and close the connection to allow a new one to be established with another device.

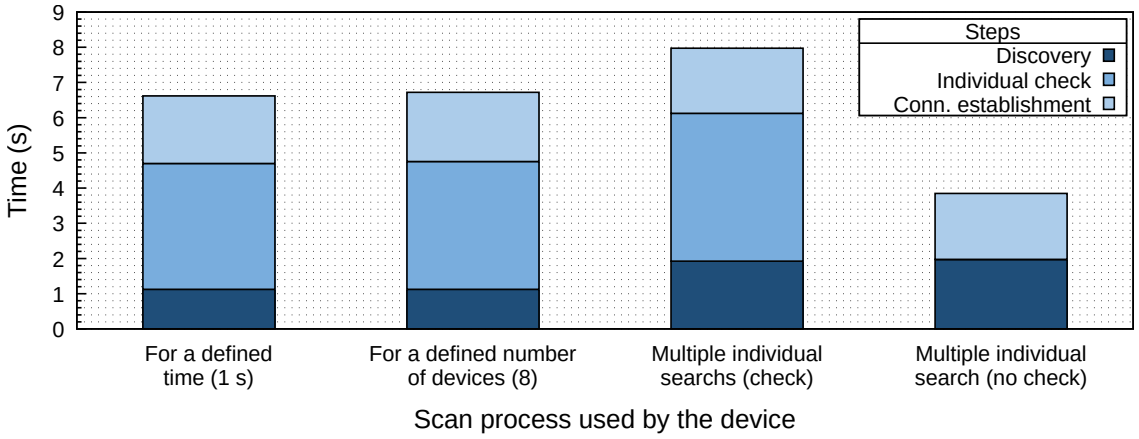


Figure 3.9: Comparison of the time required to establish a connection according to the scan method selected for the advertiser discovery.

3.2 CSRmesh Evaluation

This section first presents the operation of CSRmesh [15], the mesh topology proposed by CSR before this topology was included in the Bluetooth specification. Second, it presents the evaluation of the CSR devices used in CSRmesh, and finally, the evaluation of the performance of CSRmesh itself. In this way, the study cases were:

- Study of PRR in CSR devices. One of the fundamental requirements for communications networks in Industry 4.0 is *zero fails*, so this ratio has a great impact. Therefore, knowing the PRR of these devices in optimal conditions was considered necessary.
- Study of CSR device coverage, which permitted us to deploy a network in a real environment in an effective way.
- Evaluation of the CSRmesh network in a real environment.

3.2.1 CSRmesh

As mentioned above, mesh topology was not included in the BLE specification until 2017, which was seven years after the launch of the first BLE specification. However, some initiatives appeared implementing this topology for BLE, with CSRmesh being one of the first to be released. We focused on CSRmesh because of its early release, as well as for using devices with BLE 4.1 (the latest version at that moment), which enabled the combination of different BLE roles.

The CSRmesh mesh topology proposal is built over the BLE protocol, and uses three BLE roles simultaneously:

- *Broadcaster*. Data packets are transmitted using the broadcaster role. These packets are encrypted to provide security for the data transported, but do not fully respect

the BLE standard. The sending of each packet is repeated three times per advertising channel to ensure its reception.

- *Observer*. This role enables devices to receive packets from other devices at any time, even during idle periods when sending packets. If received packets fulfil the requirements described below, they are relayed in broadcast again, thus increasing the coverage range of the network (although the network traffic also increases).
- *Advertiser*. This role enables other devices to establish a point-to-point connection. CSRmesh is a proprietary protocol, so introducing a new BLE device in the mesh network requires them to establish a point-to-point connection with one of the CSR devices, which acts as a bridge to the network. The CSR nodes relay the packets received from the connected point-to-point devices to the rest of the nodes in the network, which is composed only of CSR devices and the devices connected to them.

The CSRmesh designed by CSR is a flood mesh protocol. Nevertheless, the main problem of this routing protocol is the high number of relays [52]. In order to reduce these relays, CSRmesh defines two parameters:

- TTL: One byte field which indicates the maximum number of times the packet can be relayed, in other words, the maximum number of hops among devices. When a device receives and relays a packet, the TTL is decremented, discarding the packet if it is 0.
- Packet ID: Each packet contains an identifier, which will be the same as long as it remains on the network. This identifier enables devices to discard packets that they have previously received, checking only this field.

3.2.2 PRR in CSR Devices

As already described, packets are sent and relayed three times per advertising channel in the CSRmesh to ensure they are received by the destination device. When a device receives a packet, it relays it three times unless it is the destination device. When the same packet is received more than once by the same device, it is discarded and no action is taken. We conducted a study to verify whether this approach is sufficient to provide the performance required by new applications.

A simple mesh network was deployed for this experiment. This network consisted of a single broadcaster and a single observer, both being CSR1010 devices (see Section A.1.2). The broadcaster sent packets to the observer. The study was conducted by evaluating the PRR based on the number of repetitions of each packet. A new packet was sent every 5 seconds during 3000 seconds. When repetitions were established at the broadcaster, a new packet and its repetitions were sent every 5 seconds. This experiment focused on evaluating the PRR of the devices, so they were placed in optimal conditions to send and receive data: in a direct line of sight and at a distance of 50 cm, minimising packet loss due to distance or obstacles.

Table 3.3 shows the PRR for packets sent once, twice or three times. The results reflect an improvement when the number of transmitted packets was increased. A PRR of 83.76% was obtained when packets were sent only once; the PRR obtained increased to 97.21% when packets were sent twice and it rose to 100% when the number of repeated packets was three. The need to send each message three times is due to two factors: (1) the fact that the radio is not only used to send packets, but that these are also relayed, being unable to receive packets at that particular instant of time; and (2) observers scan the three advertising channels sequentially, so that broadcaster and observer need to match on the same channel for the correct packet transmission.

Table 3.3: PRR (%) in data transmission between a single broadcaster and a single observer, according to the number of packet repetitions.

Transmissions of each data packet	PRR (%)
1	83.76
2	97.21
3	100.00

As this basic experiment shows, the CSRmesh operating mode based on repeating the packets 3 times ensures the reception of all packets. However, it also increases network traffic, power consumption and packet collisions in a saturated wireless frequency band. This could cause a scalability problem when the number of devices in the network becomes higher.

A new experiment was carried out to study these problems in a real network. For this new experiment, a network was deployed using a greater number of devices, completely covering our research institute, the I3A. However, before the deployment of the network, a coverage study was needed.

3.2.3 Coverage Study for CSR1010 Devices

The previous step to deploying a network of wireless sensors is to conduct a coverage study. This allows us to determine the lowest number of devices required to cover a given area at the lowest cost. For this reason, we conducted this coverage study on the first floor of our I3A institute (48 metres long and 15 metres wide).

An important consideration when calculating the range of a device is its antenna. The CSR1010 devices are equipped with an inverted-F antenna. The radiation pattern of this antenna is circular in the XZ plane. Therefore, we can distinguish two different areas depending on the location of the device: the range of coverage is favourable in the XZ-plane (see Figure 3.10) and unfavourable in the other planes.

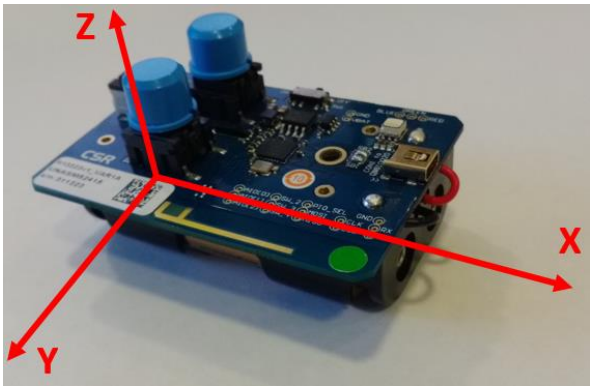


Figure 3.10: X, Y and Z plans in a CSR1010 device.

In open areas with direct line of sight, the coverage range of the CSR1010 devices is approximately 30 metres. However, this range is shorter in indoor environments with obstacles, so a study was required to evaluate this range in our laboratory.

In this experiment, two devices were used: a broadcaster and an observer, placed at a distance of 4 metres with construction elements between them (see Table 3.4). This study was carried out for this particular building, so it should be repeated in other buildings to ensure the minimum number of devices needed to cover their respective area. In our experimental scenarios, 80 RSSI samples were taken in coexistence with other wireless technologies (Wi-Fi and ZigBee).

For each scenario, two experiments were carried out, based on the device position: favourable and unfavourable. Table 3.4 shows the results obtained.

Table 3.4: Average RSSI obtained in different scenarios.

Scenario Description	Average RSSI - Favourable Antenna Position (dBm)	Average RSSI - Unfavourable Antenna Position (dBm)
Direct Line of Sight	−67	−57
Dividing panels	−65	−63
Doors	−68	−73
Columns	−80	−83
Walls	−60	−62
Walls + storage racks	−79	−95
Glass	−65	−70

This previous analysis allowed us to study how the different construction elements affect the range of coverage. Thus, we deployed the CSRmesh network avoiding problematic elements, such as columns, walls and storage racks.

The first network deployed had 10 devices. This network covered the entire coverage area, so we gradually reduced the number of devices to achieve the minimum necessary to

cover the entire building. We therefore concluded that the minimum number of devices for the deployed mesh network to cover the entire desired area was three.

3.2.4 CSRmesh Evaluation

Once the behaviour of a single CSR device was studied, the objective was to know how a CSRmesh network really works. To do this, the following steps were taken:

- The number of CSR devices needed to cover the first floor of our research institute was three, according to the previous coverage study.
- In order to guarantee the most realistic possible testing environment, we emulated an industrial environment where it was necessary to measure different parameters processed by a BLE server. For this purpose, two Waspmote devices equipped with sensors were added to the CSRmesh, using CSR devices as a bridge (following the CSR topology). These Waspmote devices sent the sensor data packet every five seconds. In addition, a BLE server that received, processed and stored BLE packets was also included using the remaining CSR device. Node.js (see Section A.2.3) was used for developing this BLE server. Figure 3.11 shows the final device setup.
- Moreover, to determine the impact of packet repetition in a real mesh network, the CSR devices were configured with two different settings: transmitting each packet once (saving mode), and transmitting each packet three times (default mode).
- Finally, for each proposed configuration, the PRR was measured, as well as the number of packets per second received on each CSR device, to evaluate the network traffic.

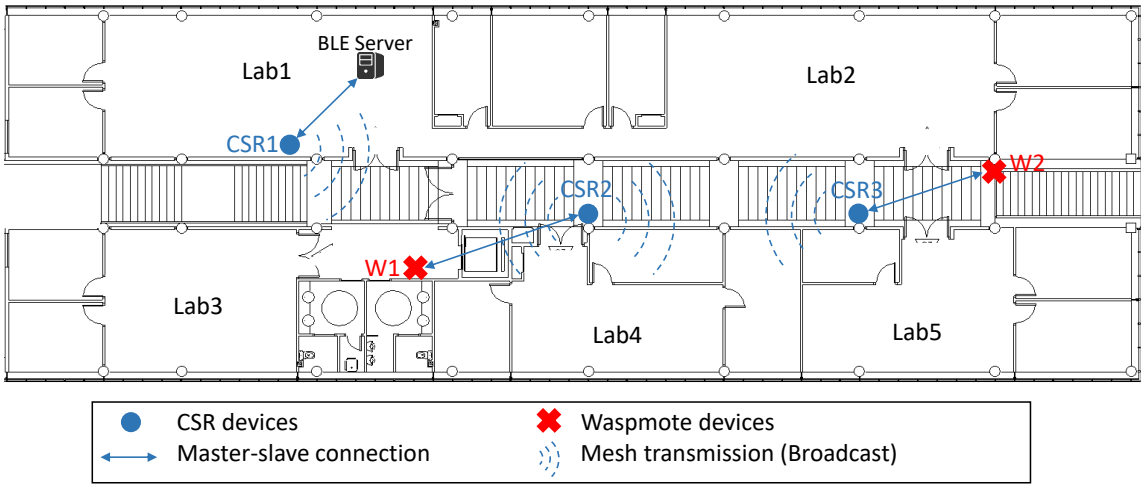


Figure 3.11: Network deployed using 2 sensor devices and CSRmesh topology.

Table 3.5 and Table 3.6 show the network traffic (sent and received packets, respectively, by different devices). As a result of the mesh network topology, which uses broadcast mode, and the proprietary code of the CSR devices, a typical traffic matrix is not available. Table 3.5 shows the number of packets sent by the Waspmote devices (W1 and W2), as well

as the number of these packets received by the BLE server, for two different network configurations. We can observe that the number of packet repetitions is especially significant for W2 packets, which had to be relayed by three different devices (CSR3, CSR2 and CSR1) and finally received by the BLE server. In contrast, for W1 packets, which were relayed from CSR2 to CSR1, there were no significant differences when changing the configuration.

Table 3.5: Packets sent by sensor nodes and received by BLE server, for each CSR devices configuration.

Sensor Nodes	CSR Devices Transmit Each Packet Once		CSR Devices Transmit Each Packet Three Times	
	Sent Packets	Packets Received by Server	Sent Packets	Packets Received by Server
W1	555 (to CSR2)	554	577 (to CSR2)	575
W2	551 (to CSR3)	367	577 (to CSR3)	542

Table 3.6 shows the number of packets received by the different CSR devices and by the BLE server. Important information was obtained from these results. Firstly, as expected, the number of received packets increased when the number of repetitions was higher. In other words, network traffic became higher when CSR devices sent each packet three times. Secondly, the CSR1 device filtered a significant number of packets, as only 42% of the packets in the first configuration and 31% of the packets in the second configuration were non-repeated packets. For this reason, this connection was maintained in the following evaluations. Thirdly, there was an increase in the number of packets received by the CSR3 device, which was greater than the increase in the number of packets received by the CSR2. This was the opposite of what was expected, as the CSR2 device was placed in the centre of the mesh network, so it should have received packets from all other nodes in the mesh. This behaviour was inappropriate and could result in problems in large-scale networks.

Table 3.6: Packets received by CSR devices and BLE server in CSRmesh evaluation, for each CSR device configuration.

Configuration	Packets Received by			
	CSR1	CSR2	CSR3	Server
CSR devices transmit each packet once	2186	4957	3778	921
CSR devices transmit each packet three times	3629	5542	6003	1117

The PRR of each sensor node (the source nodes) can be observed in Table 3.7. In the first configuration, CSR devices relayed each packet once (saving mode), while in the second configuration CSR devices relayed each packet three times (default mode). These packets

contained different data (among which there could be critical data), so preventing their loss is crucial. The first conclusion we can draw is relatively expected: the results show a lower packet loss when CSR devices repeat each packet three times instead of once (with a PRR of 99.82% and 66.61% on average, respectively). The second conclusion is related to the placement of the device: the results of Waspmote 1 were much better than those of Waspmote 2 as the second was farther away from the BLE server (see placement in Figure 3.11), so its packets need to be relayed by more devices (increasing their number of network hops).

Table 3.7: PRR for 2 sensor devices using the CSRmesh proposal, for different CSR device configurations.

Configuration	Packet Reception Rate (%)	
	Waspmote 1	Waspmote 2
CSR devices transmit each packet once	99.82	66.61
CSR devices transmit each packet three times	99.65	93.93

As already stated, an important parameter of this type of network is the number of packets being moved by the devices at any given time (network traffic). Consequently, the number of packets received on the CSR devices was measured. These packets could be repeated (either a repeat from the same device or a relay from another device), so the CSR devices should filter them out and remove the duplicates. In reference to this, Table 3.8 shows the number of packets per second received by each CSR device, for two possible configurations: CSR devices relayed each packet once (saving mode) or three times (default mode).

Table 3.8: Packets per second received by CSR devices in a CSRmesh network with 2 sensor devices and a BLE server.

Configuration	Packets Received per Second			
	CSR1	CSR2	CSR3	Average
CSR devices transmit each packet one	0.62	1.41	1.08	1.04
CSR devices transmit each packet three times	1.16	1.76	1.91	1.61

Table 3.8 shows the cost required to achieve that higher PRR (see Table 3.7): each CSR device received on average 0.57 more packets per second when the packets were repeated three times. Although this may not seem much, it could make the situation intolerable if the number of devices in the network were increased (in this case, the number of sensor devices was only two, being the total number of devices in the network six).

3.3 Conclusions

This chapter has presented the evaluation of the topologies proposed by the BLE specification: broadcast and point-to-point, as well as the CSRmesh, a proprietary topology proposed as an improvement to the available topologies.

The first section of this chapter focuses on the topologies presented in the BLE specification:

- Firstly, this part presents a study of the range of coverage of the devices used in the remaining experiments.
- Secondly, it shows the evaluation of the PRR in broadcast topology, according to different parameters. This evaluation has demonstrated the importance of a correct configuration of the parameters for the correct reception of the packets. However, the options that provide a greater PRR require a longer scanning time, and therefore a greater battery consumption.
- Thirdly, an evaluation of the broadcast topology in a real environment is presented, focusing especially on testing communication with mobile devices (in this case, a tablet with the developed application). Although this device received information from all the network devices in the path of the user, it was not possible to correctly establish which was the nearest device at each moment.
- Finally, the section moves on to the point-to-point topology. Focusing again on the mobile devices, the experiments carried out reflect the time required to establish a connection. Although this time is relatively short, it requires a previous discovery phase, which varies according to the number of devices in the environment. Furthermore, a new connection must be established each time the user leaves the coverage area of a device.

The results obtained reveal that, while the topologies provided by the BLE specification allow data transmission in static and small-scale networks, in larger-scale networks and, especially with mobile devices, they are unable to meet the requirements of new trends, such as the latest IoT and IIoT applications.

Once the current topologies specified in the Bluetooth standard were ruled out for their use in Industry 4.0 because they do not meet its requirements, a new BLE network topology was needed: the mesh topology. As mentioned above, before this topology was included by the Bluetooth specification, there were already some initiatives to create a BLE mesh network.

One of the first initiatives to emerge was CSRmesh, a proprietary CSR topology, now Qualcomm Technologies International. The second section of this chapter therefore evaluates CSR devices and CSRmesh topology:

- First, the CSRmesh topology is presented, explaining how it works.

- Secondly, a study of the PRR of the CSR1010 devices is presented, as they use BLE 4.1, with higher performance than the devices previously used.
- Thirdly, a coverage study of the CSR1010 devices is provided, which allowed to understand these devices before the final deployment of the CSRmesh network.
- Finally, the evaluation of the CSRmesh topology carried out in a real environment with hardware platforms is presented. As seen in this experiment, the PRR of the CSRmesh proposal is acceptable, although, for this, the packets require being transmitted three times. Moreover, CSRmesh is a proprietary protocol built over BLE, and scalability problems arise when different BLE devices are included in the network, since they need a CSR device that works as a slave to bridge them with the rest of the network. Therefore, this limits the maximum number of non-CSR devices in the network to the number of CSR devices in the network.In addition, if a wider range of coverage is required, more CSR devices must be included.

To solve these scalability problems, we proposed a new mesh network approach, also prior to the release of the Bluetooth mesh specification, which we called collaborative mesh. Our collaborative mesh is presented in detail in the next chapter, and enables any BLE device able to use the observer and broadcaster roles to participate in the network without the need of bridging devices. The next chapter also includes the experiments that were carried out to evaluate the PRR and network traffic of our proposal.

CHAPTER 4

Our Proposal for BLE Mesh

As concluded in Chapter 3, one of the limitations of the CSRmesh is its scalability, particularly when adding new sensor nodes; for a non-CSR node to take part in the mesh network requires the use of a CSR device as a bridge. However, scalability is an important requirement for Industry 4.0 networks, regardless of their devices. Moreover, heterogeneity is another important aspect of these networks, and our aim is to enable the connection of different types of devices, from different manufacturers, in the same network.

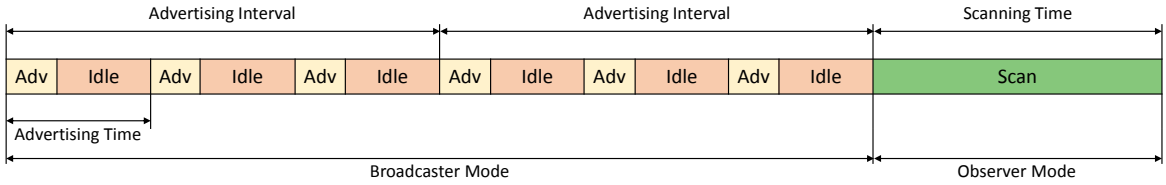
After studying the CSRmesh and understanding its deficiencies, we decided to develop a new proposal for a BLE mesh network. Our proposal eliminates the restriction of establishing master-slave connections when new non-CSR devices are included in the network. This allows all BLE devices to broadcast and relay the mesh packets.

This chapter presents our mesh topology proposal for BLE, called Collaborative Mesh, as well as its evaluation using different configurations. The GreenISF scenario is then detailed, in which different devices were included in our network, evaluating its performance.

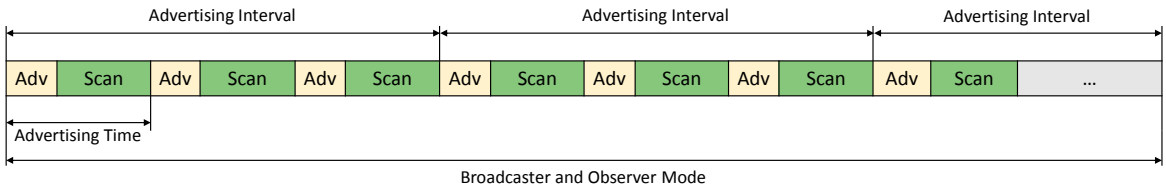
4.1 Collaborative Mesh Proposals

Our collaborative mesh proposal defines a new packet format, encapsulating the new fields in the Advertising Data field of the BLE advertising packets (see Figure 2.2.3.2). In addition, the packet format of our collaborative mesh is compatible with CSR devices, so they can also be included in the network. CSR devices use BLE 4.1, which gives them greater flexibility in terms of operating modes. BLE 4.0 devices need to switch between broadcaster and observer modes, and can only transmit or receive data at a given time, so they can be used as sensory devices if a more efficient backbone of devices is available. However, BLE 4.1 devices can use observer and broadcaster modes simultaneously, without the need to switch between them, using the idle periods of the broadcaster mode to scan the network (observer mode). This difference is shown in Figure 4.1. Therefore, our collaborative mesh network takes advantage of the different versions, being able to include these BLE 4.1 devices in our

network as a backbone, due to their better performance, without misusing the rest of the BLE 4.0 devices.



(a) Advertising and Scanning processes in BLE 4.0



(b) Advertising and Scanning processes in BLE 4.1

Figure 4.1: Differences between Advertising and Scanning processes in BLE version 4.0 and 4.1.

As stated, we proposed our own packet format because CSR uses its proprietary mesh protocol, which is not accessible to users. However, our packet format was designed following the BLE standard, which makes it compatible with any BLE device, including CSR devices. Thus, any two devices can communicate using the mesh when both are in its coverage area. To do this, the devices only have to transmit the advertising packets following the defined format, in broadcaster mode. Figure 4.2 shows the defined format, encapsulated in the Advertising Data field of the BLE advertising packet (see Figure 2.2.3.2), while the different fields are described below.

- 8-bit preamble used in the receiver for frequency synchronization, symbol timing estimation, and Automatic Gain Control training tasks. The preamble must be 0xAA in advertising packets.
- 32-bit access address, which shall be 0x8E89BED6 for advertising packets.
- PDU, a variable size (12-37 bytes) payload, which includes:
  - A 16-bit header where PDU type is specified. The PDU type used in broadcast data transmissions is *non-connectable undirected advertising*.
  - 6-octet advertising address, which contains the BLE address. Our proposal uses random addresses.
  - Advertising Data, a variable size (from 4 to 31 bytes) field, which contains the information itself (data collected from sensors or device information, for example). This field contains:
    * 1-octet Advertising Data Packet Length, following the BLE standard.

  - Data header, which contains two different fields: a 1-octet Type to indicate the PDU service (the value for mesh packets is 0x16, corresponding to Service Data, and a 2-octet UUID. To ensure the compatibility with CSR devices, the UUID for mesh packets shall be the CSR UUID.
  - 1-octet destination device ID, used to identify the mesh devices in a shorter way.
  - Data Fields, which are divide in two sections, a 1-octet ID to indicate the data type and its length, and the data itself.
  - Packet ID, which allows devices to avoid the uncontrolled packet relay. To this end, this field includes the 2-octet source device ID and a 1-octet packet counter, guaranteeing the uniqueness of the packet.
  - 1-octet TTL to limit the packet lifespan.
- 3-octet CRC, which shall be calculated over the PDU.

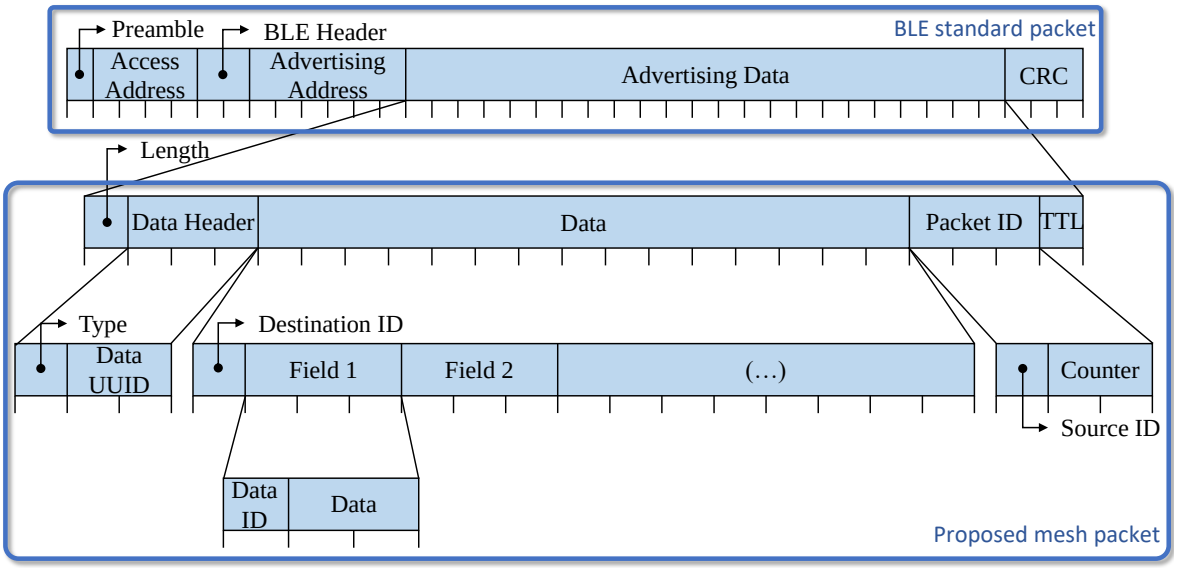


Figure 4.2: Proposed mesh packet format.

In our collaborative mesh network, a device only requires the ID of the receiving device to transmit information. This ID can be programmed in the device's memory or be assigned dynamically by a BLE controller.

Moreover, the proposed packet format provides two possible configurations for sensor devices, since communication with a mesh device is possible without it being a full mesh device:

- *Individual Mesh*, where sensor devices only transmit their data packets, but do not relay the packets of other devices. This option brings less traffic to the network.

- *Collaborative Mesh*, where sensor devices transmit their data packets and they also relay packets received from other devices. This option increases the network coverage, but also the network traffic.

Our collaborative mesh proposal has several improvements over the CSRmesh evaluated in Chapter 3:

- To know the source and destination device of each packet.
- To increase the data field length in each packet.
- To introduce new devices working as mesh devices, both to transmit their own data packets and to relay the packets received, without using a CSR device as a bridge. This is particularly important when the devices included in the mesh network are mobile or wearable devices from the users: smartphones, smartwatches, tablets or smartbands, for example. While the CSRmesh requires a new device to establish a master-slave connection that will be lost when the user leaves the coverage range of the bridge device, our collaborative mesh allows the devices to transmit the data packets to the mesh in general, with these being relayed by any other device, emphasising the concept of mesh and total coverage.
- It does not matter how many bridge devices are available when adding a new device to the mesh, since no bridge devices are required. Our mesh approach provides greater scalability, sustainability and cost savings by reducing the number of devices in the network without compromising performance.

4.2 Evaluation of our New Mesh Proposals

After describing our proposals for a BLE mesh network, this section presents a study of their performance using real hardware platforms. This study exploits the scalability of our proposals to include different devices. Therefore, the experiments were carried out using eleven devices: eight Waspmotes (see Section A.1.1) using BLE 4.0 and three CSR devices (see Section A.1.2) using BLE 4.1. The Waspmote IDE (see Section A.2.1) was used to implement our mesh proposals in Waspmote devices. These experiments were conducted for the two configurations defined above: *Individual Mesh* and *Collaborative Mesh*.

To evaluate the performance of our proposal, the points listed below were followed:

- Three CSR devices were positioned to provide coverage for our entire laboratory.
- As in the experiments presented in Chapter 3, an industrial environment was emulated, where different parameters were measured by the sensor nodes. A BLE server developed using Node.js (see Section A.2.3) was included in the network via a master-slave connection to exploit the BLE 4.1 radio of the CSR devices. Eight Waspmote sensor devices were also included in this scenario. These devices collected data, send-

ing them in packets through the network every 5 seconds. The complete scenario is shown in Figure 4.3.

- The experiments were carried out for the two defined configurations: *Individual Mesh* and *Collaborative Mesh*. For each evaluation, the PRR of each device and the network traffic in CSR devices (in packets per second) were measured.
- For each mesh configuration, experiments were carried out varying the load on the network. Thus, two different configurations were established. On the one hand, when a *Low Network Load* was selected, Waspmote devices sent each packet only once, which reduced the network traffic but increased the probability of the packets being lost. On the other hand, when a *High Network Load* was selected, Waspmote devices sent each packet three times, increasing network traffic but decreasing the probability of packet losses.

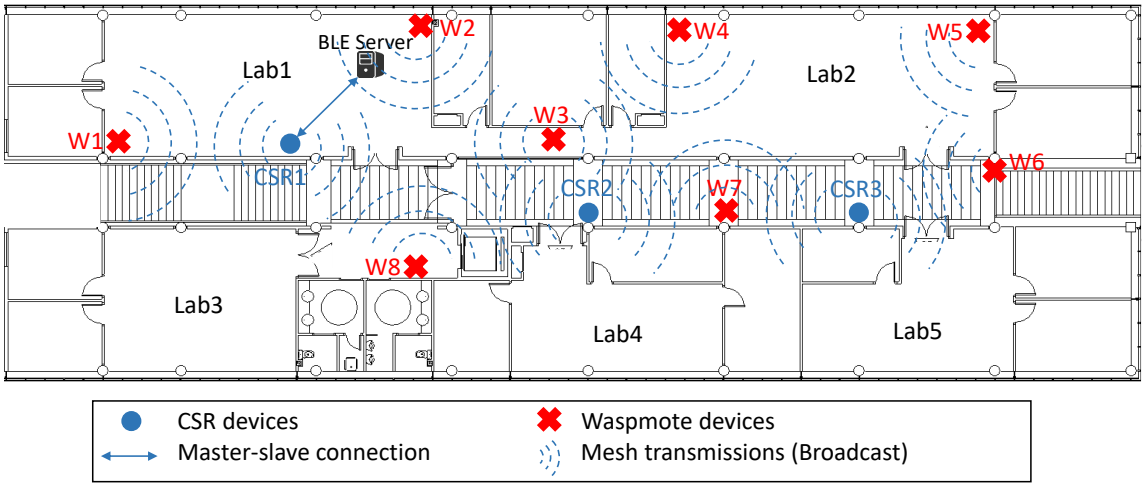


Figure 4.3: Network deployed using 8 Waspmote (sensor devices) and 3 CSR devices.

The following sub-sections detail the results obtained for each of the configurations, in order to determine which is the best option.

4.2.1 Individual Mesh Evaluation

This section presents the evaluation of the *Individual Mesh* configuration of the Waspmote devices. In this configuration the Waspmote devices transmit and receive their data packets using our packet format for mesh networks. When a packet is received by a device, it is processed if it is the destination device, otherwise the packet is discarded.

The most notable improvement in our proposal for the CSRmesh is its scalability. Thus, our mesh network was deployed using 3 CSR devices, eight Waspmote devices and a BLE driver. This increases the sustainability of the network while reducing its cost by removing the need for bridge devices. It also maintains the coverage range of the network, since the number of devices that relay the received packets is the same (in this case, the CSR devices).

By default, CSR devices relay each packet three times. This number can be reduced, although it increases the probability of packet losses. To evaluate the impact of this parameter, our proposal was evaluated using both configurations: the default mode, where CSR devices relay the packets three times, and the saving mode, where CSR devices relay the packets only once.

4.2.1.1 CSR Devices Relay each Packet Three Times: Default Mode

In this scenario, the CSR devices were set to default mode, sending each message three times. The Wapsmote devices were configured with the settings described above: *Low Network Load* (each packet was transmitted only once by Waspmote devices) and *High Network Load* (each packet was transmitted three times by Waspmote devices).

Table 4.1 and Table 4.2 show the packets received and sent, respectively, by the different devices in the experiments carried out. Specifically, Table 4.1 shows the total number of packets received by the CSR devices (in charge of relaying the packets) and the BLE server (network sink). As previously described, CSR devices processed the received packets, relaying them the first time they were received (packets received more than once are discarded). However, the CSR API is not completely open, and the number of packets relayed cannot be accessed. The results show an increase in the total number of packets received in the *High Network Load* configuration (see Table 4.2), since the packets initially sent by each of the Waspmote nodes were repeated three times. In this experiment the node with the highest network traffic was the CSR2 node given its position (see Figure 4.3).

Table 4.1: Number of packets received by CSR devices and BLE server for each Waspmote device configuration in Individual Mesh with CSR devices in default mode.

Number of Packets Transmitted by Waspmote Devices	Packets Received by			
	CSR1	CSR2	CSR3	Server
1 (Low Network Load)	15587	26306	21618	4761
3 (High Network Load)	23950	43306	32070	6132

Table 4.2 shows the number of original packets, excluding repetitions (note that in *High Network Load* configuration, Waspmote devices send each packet three times), sent by Waspmote devices and received by the BLE server for each network load configuration. In this case, Waspmote devices sent broadcast packets to all the devices in their coverage range, it being impossible to know which device received and relayed them. Despite this, all Waspmote devices sent a similar number of packets during the duration of the experiment, obtaining a larger number of packets received by the BLE server in the *High Network Load* configuration.

Figure 4.4 shows how, even using the *Low Network Load* configuration, the average PRR (98.11%) was higher than the average PRR obtained previously for the CSRmesh (around 97%) using the recommended configuration. Moreover, the network deployed to carry out

this experiment had eight Waspmote devices (sensor nodes), compared to the two Waspmote devices that the CSRmesh allowed us to deploy given the need to use bridge devices.

Table 4.2: Number of packets sent by each sensor node and received by BLE server, for each Waspmote device configuration in Individual Mesh with CSR devices in default mode.

Sensor Nodes	Low Network Load		High Network Load	
	Sent Packets	Packets Received by BLE Server	Sent Packets	Packets Received by BLE Server
W1	644	633	781	780
W2	582	571	811	810
W3	594	576	745	744
W4	588	579	748	746
W5	607	598	763	760
W6	635	623	768	764
W7	627	613	747	747
W8	576	568	782	781

In addition, Figure 4.4 shows the PRR for the *High Network Load* configuration. As can be appreciated, if the Waspmote devices sent each packet 3 times, the PRR went up to 99.79% on average.

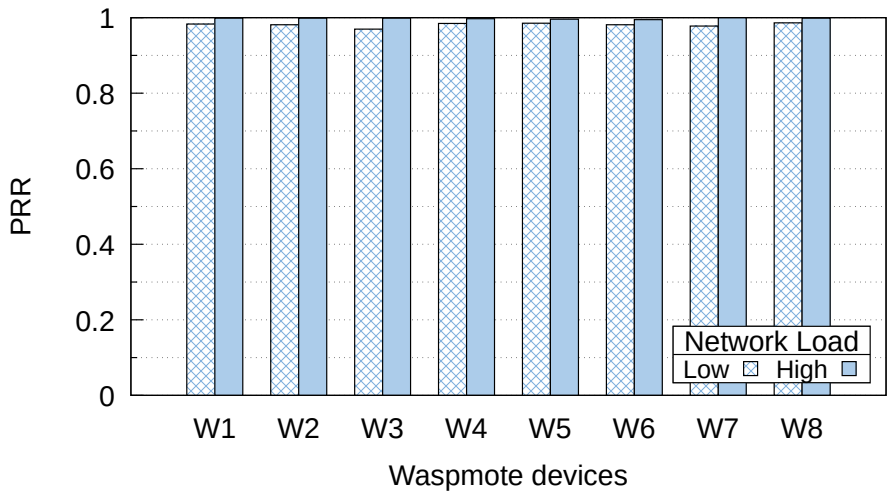


Figure 4.4: PRR in our mesh proposal for 8 sensor devices with *Individual Mesh* configuration and 3 CSR devices with default configuration.

Naturally, another important parameter to be taken into account is the network traffic, especially in this type of mesh networks. Table 4.3 shows the packets received per second by the CSR devices, which are in charge of relaying the packets. As can be appreciated, even using the *High Network Load* configuration, the number of packets received per second by the CSR devices was, on average, 8.34. The experiments carried out in the CSRmesh obtained an average of 1.61 packets received per second by each CSR device in default

mode, but the number of sensor nodes was only 2. Moreover, when the *Low Network Load* configuration was used, the number of packets received per second by the CSR devices dropped to 6.56 on average, reducing the network traffic.

Table 4.3: Packets per second received by CSR devices in default mode. Waspmote devices were configured as *Individual Mesh*.

Number of Packets Transmitted by Waspmote Devices	Packets Received per Second			
	CSR1	CSR2	CSR3	Average
1 (Low Network Load)	4.83	8.16	6.70	6.56
3 (High Network Load)	6.03	10.91	8.08	8.34

This study demonstrated the advantages of our mesh proposal with respect to CSRmesh topology: our proposal increased the scalability of the network by removing the requirement to use bridge devices, which also significantly reduces costs. In addition, it increased the PRR of the network, maintaining the network traffic.

Finally, it is possible to further reduce network traffic by reducing the number of packet repetitions relayed by CSR devices (by using saving mode). Although this measure can reduce the PRR of the network, an evaluation of this configuration in a real environment is of great interest. Therefore, the following section shows the evaluation of the *Individual Mesh* configuration for Waspmote devices combined with the saving mode of CSR devices.

4.2.1.2 CSR Devices Relay each Packet Once: Saving Mode

In this scenario, CSR devices were configured in saving mode, relaying each received packet only once. Experiments were carried out with Waspmote devices using the two configurations of our mesh proposal: *Low Network Load*, where packets are sent only once by Waspmote devices, and *High Network Load*, where each packet is sent three times by Waspmote devices.

Table 4.4 and Table 4.5 contain the results obtained regarding network traffic. In particular, Table 4.4 shows the number of packets received by the CSR devices, as well as the number of packets received by the BLE server, excluding repetitions. The CSR2 device received the highest number of packets in both configurations, due to the location of the nodes (see Figure 4.3). The increase in the number of packets received by CSR devices when the *High Network Load* configuration was used, outstanding, in comparison to the *Low Network Load* configuration (198% more, on average), although the number of packets received by the BLE server only increased by 12.4%.

Table 4.5 shows the number of packets sent by each Wapsmote device. Although the route taken by the packets cannot be known due to the use of broadcast transmissions, the number of packets received by the BLE server (sink node) is shown. In this case, the number of packets received by the BLE server is increased when the *High Network Load* configuration is used, as discussed below.

Table 4.4: Number of packets received by CSR devices and BLE server for each Waspmote device configuration in Individual Mesh with CSR devices in saving mode.

Number of Packets Transmitted by Waspmote Devices	Packets Received by			
	CSR1	CSR2	CSR3	Server
1 (Low Network Load)	7172	11481	9197	4814
3 (High Network Load)	15628	20480	18395	5410

Table 4.5: Number of packets sent by each sensor node and received by BLE server, for each Waspmote device configuration in Individual Mesh with CSR devices in saving mode.

Sensor Nodes	Low Network Load		High Network Load	
	Sent Packets	Packets Received by BLE Server	Sent Packets	Packets Received by BLE Server
W1	677	642	669	668
W2	668	624	695	695
W3	646	610	681	678
W4	632	590	670	669
W5	637	538	684	679
W6	660	611	682	675
W7	672	587	681	681
W8	654	612	669	665

Figure 4.5 shows the PRR obtained in these experiments for the different Waspmote devices. The average PRR obtained for our mesh proposal is greater than the average PRR obtained for the CSRmesh with the saving mode configuration. Our *Individual Mesh* proposal with *Low Network Load* achieved, on average, 91.75% PRR, while, using a *High Network Load*, the PRR rose to 99.61%. These results reveal a significant improvement in the PRR obtained compared to that obtained when using the CSRmesh, where the PRR was 83.27%. Thus, the PRR obtained by our proposal when using a *High Network Load* is noteworthy, approaching the requirements of Industry 4.0.

Table 4.6 shows the network traffic, measured in packets per second received on the CSR devices. Using the saving mode in the CSRmesh the results showed an average of 1.04 packets per second received by each CSR device in a network with two sensor nodes. Using our proposal, the results were 2.71 packets per second received by each CSR device on average, in a network with eight sensor nodes configured with a *Low Network Load*, providing network scalability. In addition, the *High Network Load* configuration allows us to maximise the network PRR, although the network traffic rose to an average of 5.19 packets received per second by each CSR device in a network with eight sensor nodes.

In this experiment, our mesh network proposal achieved a greater PRR than the CSRmesh. Moreover, although the network traffic was increased, this is a logical consequence considering the higher number of sensor nodes, which were able to take part in the mesh

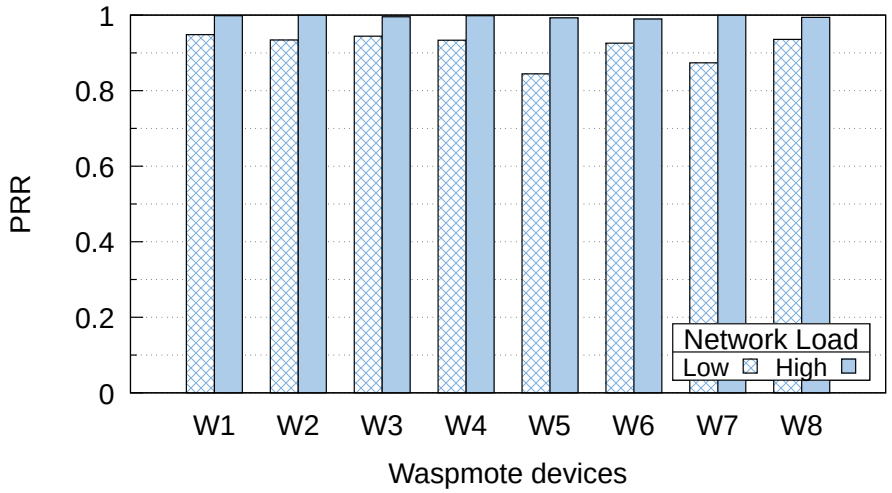


Figure 4.5: PRR in our mesh proposal for 8 sensor devices with *Individual Mesh* configuration and 3 CSR devices with saving configuration.

Table 4.6: Packets per second received by CSR devices in saving mode. Waspmote devices configured as *Individual Mesh* configuration

Number of Packets Transmitted by Waspmote Devices	Packets Received per Second			
	CSR1	CSR2	CSR3	Average
1 (Low Network Load)	2.10	3.36	2.69	2.71
3 (High Network Load)	4.47	5.86	5.26	5.19

network thanks to the higher scalability of our proposal. However, the PRR for this configuration was lower than that obtained in the previous experiments where the CSR devices were in default mode.

With these experiments the *Individual Mesh* configuration of our proposal was evaluated. However, our proposal has another configuration, the *Collaborative Mesh*, where all the devices in the network (including the sensor nodes) collaborate to extend the mesh network, relaying the received packets. This configuration is evaluated in the next subsection.

4.2.2 Collaborative Mesh Evaluation

This section presents the experiments carried out using the second configuration of our proposal: the *Collaborative Mesh*. In this configuration, all devices collaborate to create a mesh network. This provides higher scalability, allowing a mesh network to be deployed using any BLE device, even if all of them are sensor nodes. In addition, this configuration increases the coverage range of the network, since the number of devices that relay the received packets is greater, while maintaining the cost (it does not require the use of dedicated devices). This feature is particularly important when including mobile devices, which can freely move across the network.

It is important to note that this configuration of our proposal cannot be implemented in the CSRmesh for two reasons: first, the requirement that the new mesh devices must use a CSR device as a bridge; and second, this requirement limits the role of the new devices to that of master.

As in the previous section, two different configurations were used for the CSR devices also in this case: the default mode and the saving mode.

4.2.2.1 CSR Devices Relay each Packet Three Times: Default Mode

For this experiment, the CSR devices were set to default mode. In this mode, received packets were relayed three times. Furthermore, the two configurations for the Waspmote devices were used: *Low Network Load*, where the packets were transmitted (or relayed, if the packet was received) only once, and *High Network Load*, where the packets were transmitted (and relayed for a received packet) three times.

Table 4.7 shows the number of packets received by the CSR devices, which were the core of the deployed mesh network. The number of received packets was higher compared to the *Individual Mesh*, since in the *Collaborative Mesh* all devices relay the packets. In both cases, for the *Low Network Load* configuration and for the *High Network Load* configuration, the CSR2 device received the greatest number of packets, due to its placement (see Figure 4.3), as it did in the *Individual Mesh*.

Table 4.7: Number of packets received by CSR devices and BLE server for each Waspmote device configuration in Collaborative Mesh with CSR devices in default mode.

Number of Packets Transmitted by Waspmote Devices	Packets Received by			
	CSR1	CSR2	CSR3	Server
1 (Low Network Load)	36917	49824	34377	6326
3 (High Network Load)	51316	77035	63009	5411

Table 4.8 shows the number of packets sent by Waspmote nodes and the number of these packets received by the sink device: the BLE server. The number of relayed packets has also been included, owing to sensor nodes also relaying packets in this *Collaborative Mesh*, as explained.

Figure 4.6 shows the PRR obtained for each Waspmote device using the two configurations: *Low Network Load* and *High Network Load*. The PRR per Waspmote device was, on average, 97.94% for the *Low Network Load* and 99.89% for the *High Network Load*. The PRR obtained in this experiment was greater than that of the CSRmesh, despite the greater number of sensor devices (eight devices used in our proposal compared to the two deployed in the CSRmesh experiment).

Comparing this configuration with the *Individual Mesh* configuration, whose PRRs were 98.11% and 99.79% for the *Low Network Load* and the *High Network Load*, respectively, no

Table 4.8: Number of packets sent and relayed by each sensor node and received by BLE server, for each Waspmote device configuration in Collaborative Mesh with CSR devices in default mode.

Sensor Nodes	Low Network Load			High Network Load		
	Sent Packets	Packets Received by BLE Server	Relayed Packets	Sent Packets	Packets Received by BLE Server	Relayed Packets
W1	817	800	2928	709	709	4334
W2	787	756	2889	690	689	2143
W3	816	807	3076	688	687	1332
W4	814	795	3101	705	705	939
W5	787	772	3128	626	625	3211
W6	811	798	3148	655	652	4054
W7	838	825	3181	678	678	4354
W8	789	773	2882	666	666	4090

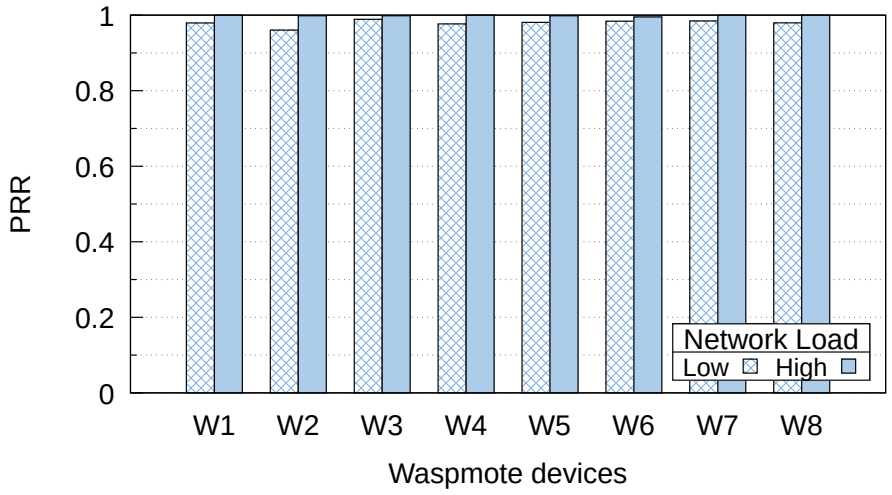


Figure 4.6: PRR in Collaborative Mesh for 8 sensor devices and 3 CSR devices in default mode.

significant difference is appreciated. However, the main difference is that the *Collaborative Mesh* configuration enables the network coverage range to be increased, due to the collaboration of all the devices to relay the packets.

Table 4.9 shows the network traffic in packets per second received by the CSR packets. As discussed, the CSR devices received, on average, 1.61 packets per second using the CSRmesh with the default configuration with two sensor nodes. In the current experiment, the number of received packets was greater (8.44 for the *Low Network Load* and 11.76 for the *High Network Load*), but also the number of sensor devices generating packets, being eight in this case.

Table 4.9: Packet per second received by CSR devices when they relay each packet 3 times (default mode). Waspmote devices configured as Collaborative Mesh.

Number of Packets Transmitted and Relayed by Waspmote Devices	Packets Received per Second			
	CSR1	CSR2	CSR3	Average
1 (Low Network Load)	7.72	10.42	7.19	8.44
3 (High Network Load)	9.46	14.21	11.62	11.76

This configuration maintains the advantages of our proposal: the PRR is close to the ideal, while increasing the scalability of the network compared to the CSRmesh. In addition, the *Collaborative Mesh* configuration increases the total coverage range of the network with respect to the *Individual Mesh* configuration, although it also slightly increases the network traffic.

The following subsection details the evaluation of this configuration in a real environment, using the saving mode for CSR devices.

4.2.2.2 CSR Devices Relay each Packet Once: Saving Mode

In this scenario, the CSR devices were configured in saving mode, relaying each packet received only once. For the Waspmote devices, the two configurations already described were used: *Low Network Load* and *High Network Load*.

Table 4.10 shows the number of packets received by the CSR devices, which are the backbone of our mesh, as well as the BLE server. For *Low Network Load* configuration, the results show a reduction in the number of packets received compared to using the default mode of CSR devices. This is because, in this scenario, the packets were forwarded by the CSR devices only once, with no repetitions. For the high network load, the number of packets received by the CSR devices was greatly increased, although the number of packets sent and relayed by the sensor nodes did not increase (see Table 4.11). This is due to the configuration used, since in the previous scenario CSR devices relayed each received packet three times. These numbers are even greater than in the Collaborative Mesh configuration with the CSR devices in default mode (see Table 4.7).

Table 4.10: Number of packets received by CSR devices and BLE server for each Waspmote device configuration in Collaborative Mesh with CSR devices in saving mode.

Number of Packets Transmitted by Waspmote Devices	Packets Received by			
	CSR1	CSR2	CSR3	Server
1 (Low Network Load)	10384	14700	12589	4617
3 (High Network Load)	95138	116130	100465	5218

Table 4.10 shows the number of packets (excluding repetitions) sent by sensor nodes (Waspmote devices), the number of packets from each sensor node received by BLE server

and also the number of packets relayed by each sensor node. The largest number of packets received by the BLE server was obtained for the *High Network Load* configuration.

Table 4.11: Number of packets sent by each sensor node and received by BLE server, for each Waspmote device configuration in Collaborative Mesh and CSR devices in saving mode.

Sensor Nodes	Low Network Load			High Network Load		
	Sent Packets	Packets Received by BLE Server	Retransmitted Packets	Sent Pakcets	Packets Received by BLE Server	Relayed Packets
W1	695	591	3006	652	652	2081
W2	680	537	2997	638	638	2058
W3	690	650	3089	660	660	2161
W4	666	592	2976	660	659	2176
W5	690	547	3046	641	641	2115
W6	680	529	2979	643	643	2121
W7	683	652	3037	670	669	2199
W8	643	519	2921	656	656	2069

Figure 4.7 shows the PRR for both Waspmote configurations. As shown, the PRR for *Low Network Load* was around 85% on average, too low for our zero fails objective. It was the lowest of all configurations for our mesh proposal, and similar to the PRR obtained for two sensor devices by CSRmesh topology, using the saving mode configuration in CSR devices. However, the PRR for *High Network Load* is 99.97%, the greatest of all configurations, including the CSRmesh.

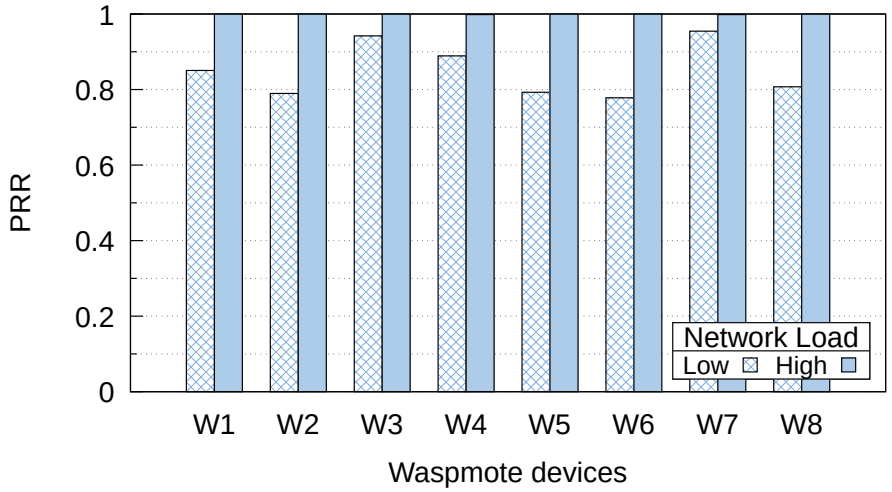


Figure 4.7: PRR in Collaborative Mesh for 8 sensor devices and 3 CSR devices in saving mode.

Table 4.12 shows the network traffic in packets received per second. For the *Low Network Load* configuration, the traffic network was remarkably low for eight sensor devices.

However, its PRR was also too low to fulfil the requirements of a real network. For *High Network Load*, the traffic network was very high although it was compensated for by an excellent PRR.

Table 4.12: Packets per second received by CSR devices in saving mode. Waspmote devices configured as Collaborative Mesh.

Number of Packets Transmitted and Retransmitted by Waspmote Devices	Packets Received per Second			
	CSR1	CSR2	CSR3	Average
1 (Low Network Load)	2.83	4.01	3.43	3.42
3 (High Network Load)	25.31	30.90	26.73	27.65

The experiments carried out in this scenario have shown that the configuration of *Low Network Load* for Waspmote devices with CSR devices in saving mode is not a viable option for the Industry 4.0 due to its zero fails requirement. However, the *High Network Load* configuration achieved an outstanding PRR, although it also involved higher network traffic. As always in wireless networks, we need to consider the importance and priority of these parameters. For our objective, the most important parameters are: to achieve a PRR as close as possible to the ideal and to provide a total network coverage, maintaining the level of traffic at levels comparable to other proposals such as the CSRmesh.

4.3 GreenISF

Green I3A Smart Factory (GreenISF) was a scenario defined in a real environment that emulates a smart factory, and in which different experiments were carried out for testing human-machine collaboration. This scenario is presented in detail in [53], and included a heterogeneous network that uses LoRaWAN and BLE technologies. The GreenISF scenario was deployed at the I3A [50], a building with an approximate area of 710 square metres. This building consists of different sectors that were classified as separate areas. Our prototype consisted of a representative sector with all varieties of existing nodes coexisting and cooperating with humans. This sector is highlighted in Figure 4.8, where a plan view of the entire building is provided according to the deployment carried out for BLE and LoRaWAN technologies. Different research groups are currently working in this building with different wireless technologies (such as Wi-Fi, ZigBee or Sigfox), being an appropriate area to determine the degree of adequacy of the network to improve social sustainability in smart factories. In addition, the sector in which our prototype is deployed to test the latency and reliability of the network contains equipment and machines that help us to make the system performance more difficult and, therefore, more robust for adverse environmental conditions in real factory scenarios.

The LoRaWAN network gathered information from the environment, being conceived as the context information network that surrounded the workers and monitored the rel-

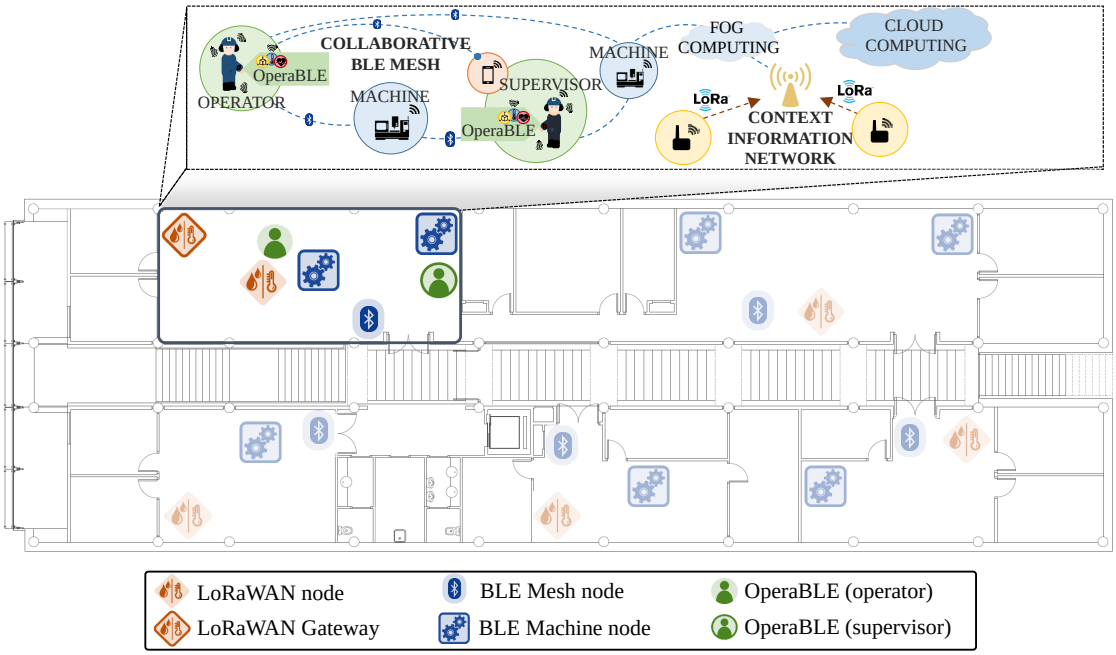


Figure 4.8: GreenIS Factory scenario for human-machine interaction.

evant environmental parameters. Data collected from sensors deployed in GreenISF were sent to a local server and eventually to ThingSpeak[54], a global IoT data analysis platform. For the development of this local server node.js (see Section A.2.3) was used.

Our collaborative BLE mesh presented in Section 4.1 was statically deployed in the GreenISF scenario and connected to the local server. New smart devices were introduced into this network, allowing users to interact with it in an intuitive and simple way, as well as to improve their safety. All collaborative BLE mesh devices are listed in Table 4.13 and detailed below.

Table 4.13: BLE devices in our collaborative BLE mesh network.

BLE device	Role in the network
Machine nodes	To monitor machines To warn supervisors of possible errors
Mesh nodes	To increase the network coverage range
Industrial equipment	To improve worker safety
Mobile devices	To facilitate the interaction of the supervisors with the information from the factory
OperaBLE	To check vital signs of operators To facilitate the interaction with the environment To improve the learning curve of operators

Industrial regulatory clothing, such as safety helmets or sound-isolating headphones, was improved by including them in the deployed BLE mesh network. Our network enabled the system to know the relative location of the operators in the different areas and thus

(a) Smart protection helmet.

(b) Smart protection headsets.

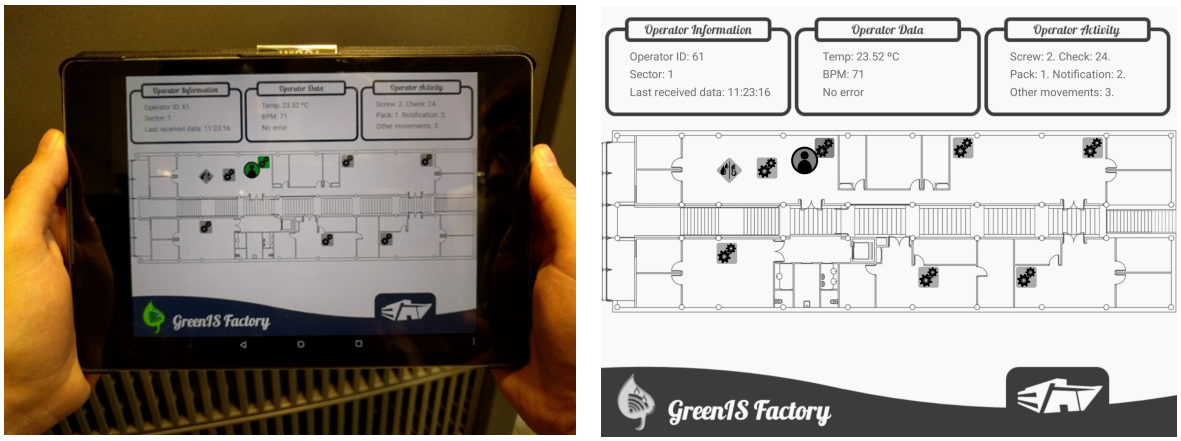
Figure 4.9: Smart regulatory industrial equipment.

verify the appropriate equipment that each operator should wear according to the manufacturing processes. In particular, the safety helmets and headphones were equipped with a LightBlue Bean device (detailed in Section A.1.3) connected to a pressure sensor (see Figure 4.9) that allowed the system to know whether the user was wearing it, informing the other devices of the mesh. In addition, the intelligent helmet incorporated an accelerometer, being able to detect impacts and immediately send an alert message to nearby supervisors.

Another of the devices included in the BLE network were mobile devices such as tablets or smartphones. In order to allow supervisors to stay informed about all the activities in the entire area of the factory, a native Android application was developed using Android Studio (see Section A.2.4). This application communicated directly through the BLE mesh network to receive the relevant activity reports, the working conditions of the operators, information on the nodes installed on the machines and context information from the local server.

The application developed displayed the machine nodes on the building plane, permitting the supervisor to request information from all of them. To do this, the supervisor could either select the node on the tablet, or use the OperaBLE device detailed below to double tap near one of these nodes to display its information. In the first case, the application sent a request to the server, which then requested the most updated information from the device, relaying the information received to the application. In the case of using the OperaBLE device, the nearest machine node that received the request from OperaBLE sent its information to both the mobile device and the server. Regardless of the mode selected, all information is transmitted through the collaborative BLE mesh. Figure 4.10 shows a tablet running the developed application, as well as a screenshot of this application with the information of an operator.

Finally, the last device included in the collaborative BLE mesh was our wearable prototype, OperaBLE, which is defined in depth in [55]. The first prototype of OperaBLE was designed using 3D-printing techniques, and modelled with the shape shown in Figure 4.11.



(a) Tablet running the developed application.

(b) Application developed for the supervisor.

Figure 4.10: Android application developed.

OperaBLE has different sensors and actuators that provide the workers with safety and comfort as well as allowing the workshop areas to be monitored by the supervisors. Several devices were used to develop our wearable prototype of OperaBLE for industrial operators. Regarding the materials used, the controller board was LightBlue Bean (see Section A.1.3), being suitable for the purpose of our prototype due to the integration of the BLE module in the board.

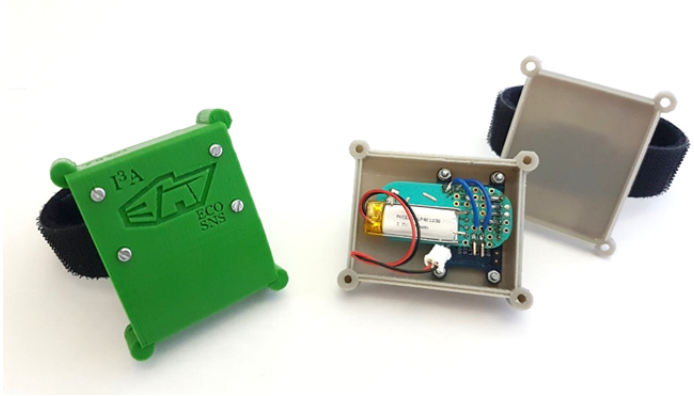


Figure 4.11: OperaBLE device.

In addition to the smart equipment, OperaBLE is a set of devices focused on the most important entity in Industry 4.0, people. For this purpose, two roles were created within the organisation, with two modes of use of the OperaBLE device: supervisors and operators. The supervisors can use the tap movement to request information from nearby machines in an extremely simple way, receiving the information requested in their application. Figure 4.12 shows a supervisor using the tap movement to request a report. The operators, meanwhile, can take advantage of the movement characterisation of OperaBLE to monitor their movements and work safely, or activate certain orders through specific movements, without the need for technological skills. This is because OperaBLE has no buttons and requires no previous knowledge in terms of digitisation. Finally, in both cases, OperaBLE

(a) Tap movement detail.

(b) Supervisor requesting reports.

Figure 4.12: Supervisor equipped with OperaBLE requesting a report.

provides the approximate location, as well as measuring the pulse and sending warnings through the emergency movement or automatically in case of detecting a problem. These warnings are sent to the server and to all mobile devices if necessary.

Thus, thanks to the combination of different technologies and different devices, GreenISF allows a detailed report of labour activity in the factory to be stored, as well as context information, which is available for consultation at any time.

4.4 GreenISF Evaluation

The preceding sections presented and evaluated our proposal for BLE mesh with different configurations depending on the role of the devices in the network. Two operating modes were proposed for our network: *Individual Mesh* (only the core devices of the network relayed the received packets) and *Collaborative Mesh* (all the devices relayed the received packets). For each of these modes, two different configurations were evaluated, varying the number of times each packet was sent. The previous section presented our GreenISF project, where different mobile devices were included in our BLE mesh. This section presents an evaluation focused on these new experiments and these new features, allowing us to see the performance of our proposal in a real application.

For the experiments carried out in this section, a *Collaborative Mesh* configuration was selected, with mesh and sensor nodes relaying each received packet only once. This configuration was selected, firstly, because our mesh network had to provide a total coverage in a real environment, and, secondly, because the combination of these configurations gave us a lower network traffic compared to other collaborative configurations, which is an important point in our sustainability objective.

The use of these new devices, as well as the functionality added to our network was proposed to facilitate the work of users and prevent occupational risks, among others. For this reason, the communications performance of these new devices is of great importance being evaluated in this section. The new communications that appeared with our GreenISF project included: the requests made by the superiors using a mobile device, after which the corresponding responses from the server had to be received. In addition, in the packet flows related to the movements collected by the OperaBLE wearable appear taps and movement characterisation.

4.4.1 Supervisor Request through a Mobile Device

The first new feature evaluated was the request for data. As already mentioned, a smartphone application was developed to facilitate the work of the supervisor. Although all the data collected by the sensors were updated in real time using our BLE mesh network, certain information was sent on demand, either because it was transmitted using another network (the context information was transmitted by the LoRa network), or because this information needed to be pre-processed (such as the movements of users). However, possible errors were notified without the need for a previous request.

This evaluation was divided into three sets of 50 requests each. In the first set, only data from the operator were requested. In the second set, only context information was requested. Finally, the third set presents alternative requests for operator and context information. For each set, the percentage of requests received by the server successfully, the percentage of responses received by the smartphone successfully, and the time interval from request to response are presented.

For operator data requests (see Table 4.14), all requests were received and processed in the server (50/50 successful requests), although some response packets were not received (48/50 successful responses). Figure 4.13 shows the histogram of the time elapsed between a request and its corresponding response, for each successful received response.

Table 4.14: Summary of experimental results obtained for operator information requests from supervisor.

Successful request	Successful response	Average time (s)
100%	92%	0.347

Results for context information requests (see Table 4.15) were similar: the server received and processed all request packets (50/50 successful requests), but not all response packets were received in the mobile device (47/50 successful responses). Figure 4.14 shows the request-response time interval.

Finally, for operator data and context information requested alternatively, the results were close to the previous results (see Table 4.16): all request packets were received and

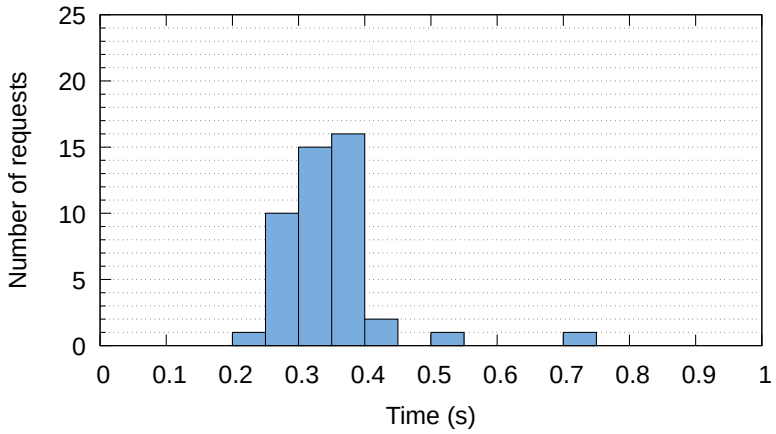


Figure 4.13: Histogram of time lap from sending request to receiving response obtained for operator information requests from supervisor

Table 4.15: Summary of experimental results obtained for context information requests from supervisor.

Successful request	Successful response	Average time (s)
100%	88%	0.318

processed by the server (50/50 successful requests), and only one response packet was lost (49/50 successful responses). The request-response interval times are represented in Figure 4.15.

Table 4.16: Summary of experimental results obtained for context and operator information request from supervisor.

Successful request	Successful response	Average time (s)
100%	96%	0.359

These experiments demonstrated that most of the packet losses were produced in the response packets sent by the server to the mobile device, which contrasts with the perfect results obtained for transmissions from the mobile device to the server. This situation was due to the way in which the relays were managed. The BLE server implemented a buffer that stored the received packets for relaying when possible. However, the mobile device relayed the packets as soon as they were received, to avoid as much as possible the use of the resources of the device. This resulted in the scanning time of the mobile device not being completely delimited, causing losses of packets in certain situations.

Regarding the user experience, we find two factors: on the one hand, the time elapsed between the supervisor selecting a target to obtain its information (which triggered a request) and the reception of the response; and on the other hand, the probability of this response having been received. Our experiments revealed, on average, a response time of

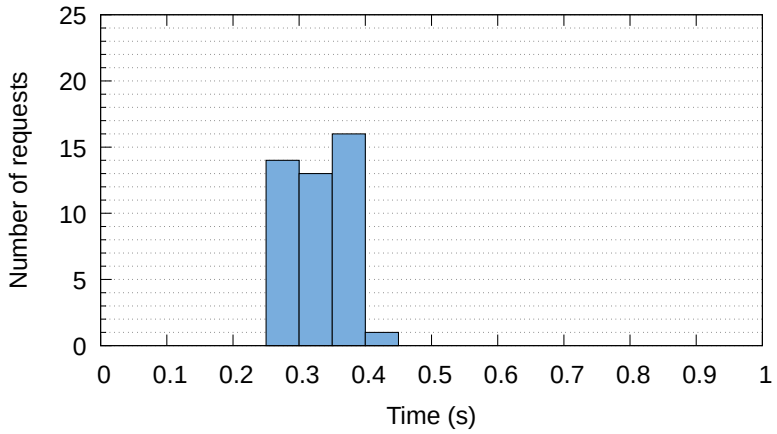


Figure 4.14: Histogram of time lap from sending request to receiving response obtained for context information requests from supervisor.

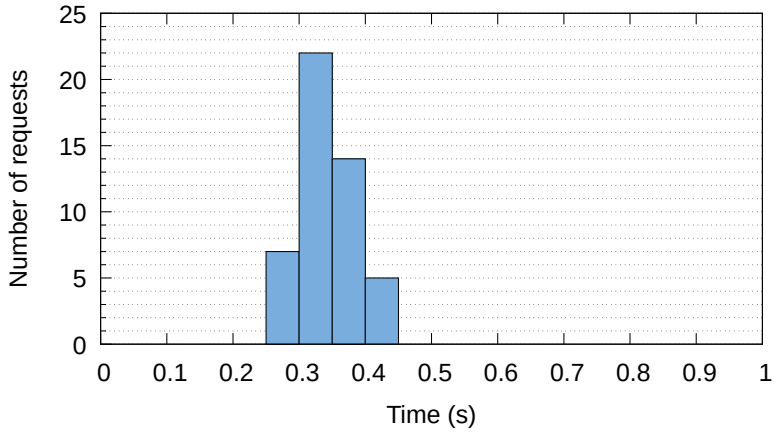


Figure 4.15: Histogram of time lap from sending request to receiving response obtained for context and operator information request from supervisor.

0.341 seconds and a probability of receiving the transmissions of 92%. This provided a good user experience for the supervisors in the developed application. In addition, the success rate in receiving the packets could be improved by resending the request when the response is not received in the stipulated time.

4.4.2 Supervisor Requests using OperaBLE Taps

Sensor nodes included in machines were programmed to send their performance data every 15 minutes, although this time interval could be modified. In case an error was detected or an uncommon value was measured, this information was sent immediately. Additionally, the supervisor could request the information from machine nodes near him or her, using the *tap* movement. When this movement was detected by the OperaBLE device, and the supervisor was close enough to a machine node, the OperaBLE device sent a request to the

corresponding sensor node. The sensor node updated its data and sent a new and updated packet to supervisor mobile device and to server through the BLE mesh network.

For this evaluation, 50 requests were sent to two different machine nodes, alternatively, using the *tap* movement and verifying the correct data reception in the mobile device. Table 4.17 contains the percentage of requests received by machine nodes and the percentage of the corresponding response packets received by the mobile device. Results prove how all request packets were received by sensor nodes (50/50 successful requests), although some data packets from sensor nodes were lost when they were sent to the supervisor mobile device (48/50 successful data packet receiving). Table 4.17 also shows the average time elapsed between a *tap* movement detection and a data packet from sensor node was received. In addition, Figure 4.16 show the time for each request-response.

Table 4.17: Summary of experimental results obtained for information for machine nodes request from supervisor using tap movement.

Detected in machine node	Received in mobile device	Average time (s)
100%	96%	1.407

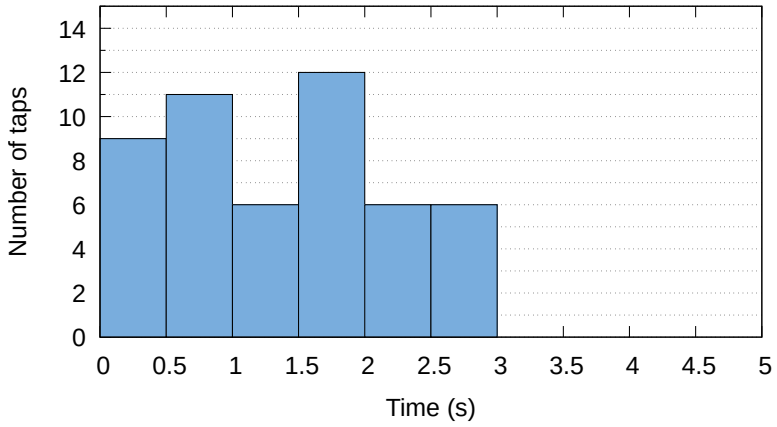


Figure 4.16: Histogram of time lap from sending tap packet (from OperaBLE) to receiving response (in mobile device) obtained for machine node information request from supervisor.

As in previous experiments, the number of responses received by the mobile device was lower than the number of requests received by sensor nodes from OperaBLE. In this case, the sensor nodes used BLE 4.0, and its high percentage of packets received was due to the source device: OperaBLE. This device was developed using LightBlue Bean, because of its small size and low power consumption. However, the broadcaster role was recently included in its firmware, with it not being possible to send a single advertisement packet. For this reason, packets were sent more than once, increasing the receiving probability in sensor nodes.

In addition, the time elapsed from the end of the tapping movement until the information packet was received on the mobile device is a good way to evaluate the user experience.

In this experiment, the time was 1.407 seconds on average. However, this type of interaction was different from others, since it was done with the environment itself, in a simple and very user-friendly way.

4.4.3 Movements Transmission by the Mesh Network

The last evaluation of our mesh network performance was the transmissions of movement data packets. The data collected by the accelerometers integrated in the OperaBLE device when a movement was executed were sent to the server for its processing and characterisation. The size of data collected for a movement was usually greater than the PDU size (for this field type, the length is 12 bytes). For this reason, fragmentation of movement data in the source device and reassembling it in the server was necessary. However, this methodology could produce movement data losses if one of its packets were lost in the transmission.

Table 4.18 shows the results of the evaluation for the transmission of 50 movement data. These movements produced a total of 481 data packets. Although the packet loss rate was only 0.42%, it involved the loss of two complete movements (4%), since full movement data was required for processing.

Table 4.18: Results for evaluation 50 movement data transmissions

Total Packets		Total movements		Average time
Sent	Received	Sent	Received	per packet (s)
481	479	50	48	0.290

The data fragmentation caused the size of the movement data, which was determined by the duration of the movement, to be directly proportional to the time needed to transmit them. Figure 4.17 shows the time required for transmitting different data size movements, according to the number of packets needed to send them. This experiment was carried out using two OperaBLE devices simultaneously, and the results demonstrated that there was no significant variation in the required time. This is because the bottleneck of our mesh network was in the source devices, not in the intermediate devices.

The Operable device collected samples of movements every 100 milliseconds. Each sample had a size of 6 bytes. As already mentioned, the size of the available data field was 12 bytes, so each packet could contain up to 2 samples, which were transported to the server. Therefore, the transmission time depended on the duration of the movement. For example, a movement of 2 seconds required the transmission of 10 packets. These movements were used to ensure that the operators worked correctly and safely.

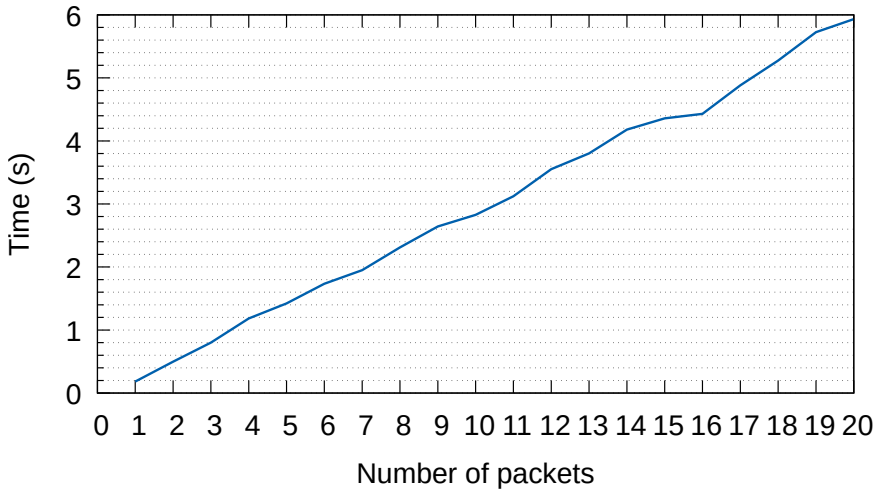


Figure 4.17: Time required for transmitting movements according to their number of packets (data size)

4.5 Conclusions

In this chapter, a new BLE mesh topology was proposed, with different configuration modes, which was evaluated for different use cases in an emulated Industry 4.0 environment. Our topology allowed us to obtain the following benefits:

- To increase the PRR, improving the network performance. The experiments carried out demonstrated that our proposal obtained a greater PRR for sensor devices. Even in networks with a greater number of sensor devices, several configurations of our proposal maintained this rate close to the zero fails required in Industry 4.0.

- To increase the coverage area of the network. All devices relayed the packets received from the mesh, ensuring the total coverage of the network, which is another Industry 4.0 requirement.

- To increase the scalability of the network. Any device with a BLE chip was allowed to be part of the mesh without master-slave connection.

- To improve the user experience. The topology of our mesh network allowed common user devices (smartphones, tablets or wearables) to communicate with the rest of the devices in the network.

Moreover, the results obtained for different network configurations proved that each configuration worked better in a particular use case:

- The *Collaborative Mesh* configuration permits an area to be completely covered using the minimum number of devices.

- The *Individual Mesh* configuration permits an area to be completely covered where the number of network devices is greater than that required to do so (the mesh network is dense).

Both configurations could be combined to create a hybrid network, in which some devices use a configuration while the rest use the other.

Regarding the number of packets per second, we can flexibly select the best configuration for our network topology, according to the particular use case: the *Individual Mesh* configuration is appropriate for small spaces with a large number of sensor devices, while the *Collaborative Mesh* configuration works properly for large areas as it increases the network coverage with a low cost. To combine both configurations in the same network is also possible. However, this is not possible in the CSR topology, since it has a fixed configuration, and also requires a bridge device for each new device in the network.

The second part of this chapter introduces the GreenISF, a Smart Factory scenario where our network prototype was deployed. In this scenario, different roles were defined for testing human-machine collaboration, including operators, supervisors, machines, static nodes and smart devices. Of these, the most outstanding is the OperaBLE device, our smart band prototype.

Our collaborative BLE mesh network was evaluated with these new devices to fulfil Industry 4.0 requirements: zero fails, total coverage and sustainability. The experiments demonstrated how this proposal improves the current standard BLE topologies and other mesh initiatives for this use case.

CHAPTER 5

Providing Interoperability in Bluetooth mesh

This chapter presents our implementation and evaluation of Bluetooth mesh stack and provisioning procedure for BLE devices which were not designed to be used in Bluetooth mesh networks, including BLE 4.0 devices, the lowest supported version. Our implementation permits all compatible BLE devices to be provisioned and to be part of a Bluetooth mesh network as any other node in the network, without using other devices as a bridge (Proxy protocol).

After carrying out our implementation and verifying its correct operation, we proposed a different approach to improve the provisioning procedure, mainly when using devices not initially designed for Bluetooth mesh, taking into account their limitations and operation modes. With the implementation completed, our work was focused on the performance evaluation. In order to evaluate our implementation, different experiments were carried out using BLE 4.0 devices according to provisioning time, provisioning robustness, end-to-end delay and PRR with encouraging results, which are of great interest in IoT and IIoT.

This chapter first presents our implementation in depth. It then introduces the Light-weight Provisioning, our proposal for improving the provisioning procedure. Finally, the experimental results are described.

5.1 Preliminaries

From when BLE (see Section 2.2) was introduced for the first time (version 4.0 [6]) until the last Bluetooth specification (version 5.2 [12]), it has continued with the same network topologies: point-to-point (1:1) and broadcast communications (1:m). Its performance in terms of throughput, range, power consumption and payload capacity has been improved in each specification to meet the new challenges arising. For example, the broadcast topology was initially designed to send advertisements enabling a point-to-point connection

and it was later used to send beacons. Now, broadcast transmissions enable different data to be sent to several devices simultaneously. Meanwhile, the point-to-point topology has increased the number of devices connected simultaneously to the same central node, and the combination of central/peripheral roles has become more flexible, enabling a peripheral device to be the central device of another node.

Although these topologies may be suitable for a multitude of applications, they are not sufficient to fulfil all the requirements of the latest trends, such as smart buildings, smart factories and smart cities. These cases require high reliability, total coverage and sustainability, both in terms of energy consumption and the number of devices used.

In order to meet these new requirements, first academia and companies, and later the Bluetooth SIG itself, proposed a new network topology for BLE: the mesh, included in the Bluetooth mesh specification [17] (see Section 2.3). Compared to the previous broadcast topology, Bluetooth mesh not only provides a longer range, but also improves the data capacity due to its extremely compact packet format [20], as well as security capabilities at different levels. Bluetooth mesh enables the easy connection of a large number of devices for home and industrial automation, building management and many other IoT applications.

Different devices especially created to use Bluetooth mesh have already been launched on the market, as well as the corresponding implementations of the Bluetooth mesh stack developed by several companies (such as Nordic Semiconductor [23] or Silicon Labs [24]). Other open source proposals for these devices have been developed by communities (such as Zephyr [25] or BlueZ [26], the official Linux Bluetooth protocol Stack), with the number of products qualified by Bluetooth SIG being more than 700 [27], including the implementations of the Bluetooth mesh stack.

Despite the arrival of these new devices and implementations, key questions remain unanswered: What about the 20000 Bluetooth SIG qualified BLE devices [56] that were not designed for Bluetooth mesh topology? Can they be upgraded and used in a heterogeneous mesh with new devices or do their limitations make this unfeasible? Bluetooth SIG has affirmed that all BLE devices can take part in the mesh network, but there is no work to prove this claim. Furthermore, if they can be upgraded, are they really usable in the new mesh networks or do they not fulfil all the requirements of the current applications? Will interoperability be possible?

5.2 Bluetooth mesh Implementation

This section presents our Bluetooth mesh implementation in detail, distinguishing between the devices used: Bluetooth non-mesh and Bluetooth mesh devices. Waspmote devices (see Section A.1.1) were selected as non-mesh devices, since they use BLE 4.0. The Waspmote IDE (see Section A.2.1) was used to implement the Bluetooth mesh in these devices. EFR32BG13 devices (see Section A.1.4) with Simplicity Studio IDE (see Section A.2.5) and

nRF52840 (see Section A.1.5) with Zephyr (see Section A.2.6) were used as mesh devices, being the Bluetooth mesh protocol stack already implemented in the software used. Finally, the nRF sniffer (see Section A.1.6) was used to verify that the data was transmitted correctly.

Most of our implementation was developed as a library in C code for Bluetooth non-mesh devices. Our Bluetooth mesh library uses some functions defined in BLE library from Libelium to communicate with the BLE112 radiochip of the Waspmote devices, although direct communications were also included in our implementation, through the sending of binary commands by the UART interface. Moreover, open-source software modules relative to security functions were included and adapted. Both the implemented library and the other software modules are in the BLE Host part. Figure 5.1 illustrates the relations between our Bluetooth mesh library and the rest of the modules.

In addition, we also programmed the Bluetooth mesh devices to perform the functions required for the experiments. This was much simpler thanks to the available Bluetooth mesh stacks. All tasks performed are described below.

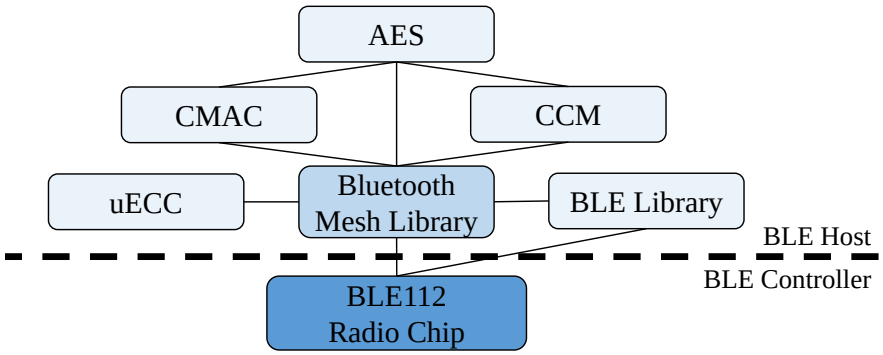


Figure 5.1: Relations between Bluetooth mesh library implemented (in the centre), the software modules needed (light) and the hardware BLE Radio Chip (dark).

5.2.1 Open-Source Software Modules Included

Software modules were included for our implementation, adapting them to work correctly with the rest of the modules, as well as with our library. Most of these modules are related to security, being the implementation of well-known algorithms:

- BLE library from Libelium [57] to manage the BLE chip. This library permits different BLE configurations to be established as well as to be switched between different BLE roles.
- uECC implemented by K. Mackay [58], which provides different functions to generate the private-public key pair and to obtain the ECDH shared secret.
- AES implemented by B. Gladman [59], which enables data to be encrypted using the AES algorithm.

- CMAC implemented by J. Song and J. Lee [47], to encrypt data using the AES-CMAC algorithm. This library uses functions from the AES library for the AES-based part.
- CCM [48] library from [23], which enables data to be encrypted, decrypted and authenticated using the AES-CCM algorithm. This library uses functions defined in the AES library for the AES-based part.

5.2.2 Our Implementation of Bluetooth mesh Library for Bluetooth non-mesh Devices

We deployed the Bluetooth mesh as a library in C code for previous BLE devices preceding the release of Bluetooth mesh such as Waspmote, developing this library in C code. The BLE Controller was implemented in the BLE112 chip, and includes the two lower layers of the BLE 4.0 standard (Link and Physical Layers, defined in Section 2.2). The BLE Host part runs on the Waspmote microcontroller device, being implemented in a C library that includes the rest of the layers of the BLE standard. Following this model, our implementation was in the BLE Host, communicating with the BLE Controller through the use of binary commands through the UART interface. The main functions developed are shown below, starting with those related to the provisioning procedure.

- *Mesh Initialisation* initialises the BLE module and related parameters, such as TX power, or advertising/scanning parameters. It also initialises necessary parameters for Bluetooth mesh: the ECDH curve and device UUID; calculates the random number for the provisioning procedure; and calculates the public-private key pair.
- *Beaconing* prepares the advertising packets and starts the beaconing stage necessary for the provisioning procedure, using the device parameters.
- *Provisioning* allows the device to be provisioned. It starts waiting for the Link Open, and goes through the different steps of the provisioning procedure. After receiving a complete transaction, it is processed, triggering the corresponding actions. These actions include sending or waiting for the reception of an ACK or a new transaction from the provisioner device. This function uses the ones defined below.
- *Provisioning Mesh Events Scanning* switches the BLE chip to scanner role, checking the received BLE messages until the reception of a provisioning message. It waits for this message until a configurable timeout up to 5 minutes.
- *Transaction Sending* prepares the transaction to be sent, fragmenting it into several segments if necessary, and switches the role of the BLE chip to advertiser for sending. This function waits for the reception of the ACK if necessary, repeating the sending if the ACK is not received in 1.5 seconds.
- *Encryption and Privacy Keys extraction* extracts the Encryption and Privacy Keys received in the Provisioning Procedure, particularly in the Provisioning Data PDU.

Regarding the Bluetooth mesh layers, we implemented two main functions for each layer, called depending on whether a packet is received or sent:

- *Network Layer Input* processes a received Bluetooth mesh Network PDU, deobfuscating, decrypting and authenticating it. Moreover, the function checks whether the PDU is already in the message cache, discarding it in that case.
- *Transport Layer Input* receives the Transport PDU from the Network Layer Input function, reassembling multiple PDUs if necessary. It also decrypts and authenticates the PDU using the application Key.
- *Access Layer Input* receives the Access PDU from the Transport Layer Input function, checking whether the incoming application data has been received correctly and sending it to the appropriate application.
- *Access Layer Output* receives the application data and encapsulated it in an Access PDU, sending it to the Transport Layer Output function.
- *Transport Layer Output* receives the Access PDU from Access Layer Output function, encrypting and encapsulating it in a Transport PDU, which is fragmented if necessary. Each of these PDUs is sent to the Network Layer Output function.
- *Network Layer Output* encrypts and obfuscates a Transport PDU, encapsulating it in a Network PDU, which is sent using advertising packets.

5.2.3 Implementation using Available Bluetooth mesh Stacks

For the EFR32 and nRf52840 devices from Silicon Labs and Nordic Semiconductor respectively, available Bluetooth mesh stacks were used.

For the EFR32 devices, the IDE provided (Simplicity Studio) was used to develop the programs that permitted us to conduct the experiments. To this end, the necessary calls to its API were used, developing programs to send and receive Bluetooth mesh messages, following the demo examples included in the API. For provisioner device implementation, there was no demo code, so calls to the Silicon Labs API were used following the available documentation. However, using their API, the number of configurable parameters was greatly reduced.

For the nRF52840 devices, Nordic Semiconductor provides the possibility of using its own Bluetooth mesh protocol stack called nRF5 Software Development Kit (SDK) for mesh. However, it does not provide access to the radio peripheral, leaving most of the functionality only exposed as system calls that can be interpreted as a virtual HCI. For that reason, we chose the open source Real Time Operating System (RTOS) Zephyr, which provides full access at all layers of implementation. In addition, Zephyr supports multiple boards, making our code easily exportable to any of them.

5.3 Lightweight Provisioning

After our development, and focusing on the provisioning procedure, we noticed a margin of improvement that could allow us to make this process lighter.

Packet exchanges in the provisioning procedure are acknowledged to ensure their correct reception by the other device. In most of these cases, the communication is restarted from the device that sends the ACK packet. Therefore, our proposal is based on removing the sending of these ACK packets, maintaining only those that are completely indispensable and using the next packets as confirmation of the reception of the previous ones. For this reason, our proposal is called Lightweight Provisioning.

Our proposal not only removes the time for sending and receiving ACKs, but also the time spent waiting for ACK packets. One of the main problems is that in some steps of the standard procedure, the ACK is sent from the same device that starts sending the next transaction. This causes the device sending the ACK to be unable to know whether this ACK has been received and whether it can send the next transaction. To deal with this problem, some implementations such as Silicon Labs wait a prudential amount of time before starting the next transaction (3 seconds in particular). In other Bluetooth mesh stacks, such as that implemented in Zephyr, the following transaction is started just after the ACK, resending the ACK when necessary. Therefore, the improvement achieved by this proposal varies depending on the implementation, but, in any event, it involves an optimisation of the current procedure by eliminating the sending, receiving and principally the waiting times of the ACK.

The Lightweight Provisioning requires only the sending of 15 packets to complete the provisioning, compared to the 25 used by the standard procedure included in the Bluetooth mesh specification. The ACK packets removed without modifying the behaviour of the Provisioning protocol are: (1) Provisioning Invite ACK; (2) Provisioning Capabilities ACK; (3) & (4) both Provisioning Public Key ACKs; (5) & (6) both Provisioning Confirmation ACKs; (7) & (8) both Provisioning Random ACKs; and (9) Provisioning Data ACK. This provisioning procedure proposal is illustrated in Figure 5.2.

In this way, our Lightweight Provisioning permits the total number of packets to be reduced without compromising the robustness of the provisioning procedure. In order to provide an easier understanding of how our proposal maintains the robustness of the standard procedure, the Exchanging Public Keys stage is analysed step-by-step as example.

The first transaction is the Provisioning Start, sent from the provisioner to the unprovisioned device. The Provisioning Start ACK cannot be removed, because without this ACK the provisioner is unable to know whether the Provisioning Start transaction was correctly received, nor when to send the next transaction (Provisioning Public Key).

Following the standard procedure, after receiving the Provisioning Public Key transaction from the provisioner, the unprovisioned device sends the respective ACK and then the

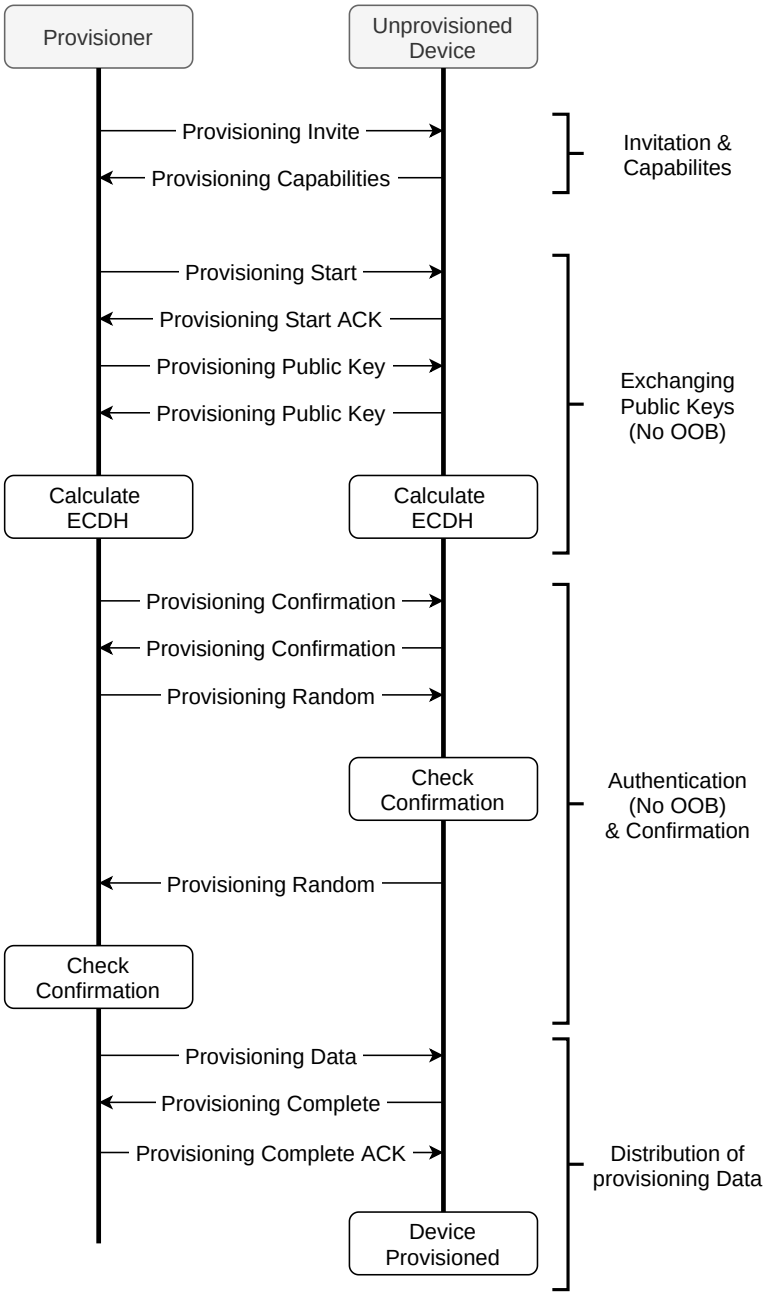


Figure 5.2: Lightweight Provisioning procedure proposal.

next Provisioning Public Key transaction. However, Lightweight Provisioning removes that ACK, using the following transaction (also from the unprovisioned device) as a confirmation of the reception of the previous transaction. Thus, when the unprovisioned device receives the Provisioning Public Key transaction, it responds by sending its Provisioning Public Key transaction. When this transaction is received by the provisioner device, the reception of its previous Provisioning Public Key transaction is confirmed. If no response is received within a defined timeout, the provisioner resends its transaction. Therefore, our

proposal avoids the sending and receiving of most ACK packets, since this situation applies to most ACK sendings.

5.4 Experimental Results

This section begins by briefly describing the configuration and conditions established for the experiments. The experiments are then detailed and their results are analysed. These experiments confirmed the feasibility of a heterogeneous Bluetooth mesh network, the interoperability of devices from different manufacturers and with different BLE versions. The experiments also measured the performance of the Bluetooth non-mesh devices included in the mesh network, verifying whether this type of device is able to take part in a Bluetooth mesh network properly.

All the experiments were conducted in our research institute, the I3A, while different research groups were working. Research at that time included 802.11 and BLE networks, so other wireless systems were operating in the same frequency band (2.4 GHz). This was intended to create a test bed as close as possible to a real environment, with possible Radio Frequency (RF) interferences.

As stated in Section 2.3, Bluetooth mesh uses mainly broadcast packets, which are sent and received using the broadcaster and observer roles, respectively. The combination of these roles is more restrictive in the BLE 4.0 specification than in later specifications, with its performance being also more limited. For this reason, the experiments were carried out using devices from different vendors and equipped with different BLE versions: EFR32 [60] devices from Silicon Labs, with BLE 5.0 and its Bluetooth mesh stack; nRF52840 [61] from Nordic Semiconductor, with BLE 5.0 and Zephyr RTOS; and Waspmote [62] devices from Libelium, with the BLE 4.0 version.

We implemented the provisioning procedure and Bluetooth mesh protocol stacks over the Link Layer of the BLE 4.0 specification of Waspmote devices. These devices were used in the following experiments to evaluate whether the version of the devices affects their behaviour in a mesh network. Taking into account the mesh operating mode, one of the main differences between the BLE radio chips used in these experiments lie in their capabilities to carry out more than one role simultaneously: while BLE 5.0 devices are able to perform the broadcaster and observer roles simultaneously, BLE 4.0 devices require some time to change their roles. Another important difference is the time required to switch the BLE radio mode between transmitter and receiver, which is considerably reduced in the latest BLE versions. These differences result in a greater packet loss on devices using early BLE versions compared to later versions, with this loss being greater, the shorter the packet transmission interval.

The experiments in this section evaluated the performance of our implementation for the BLE 4.0 device, measuring: (a) the provisioning time and provisioning robustness; (b) the end-to-end delay; and (c) the packet reception rate.

5.4.1 Experiment 1: Provisioning Time and Robustness

The provisioning procedure detailed in Section 2.3.4 permits a BLE device to take part in a mesh network. This important process is performed only once in each node, and provides a secure network key delivery from the provisioner. The provisioning procedure is a sequential process: when there is more than one unprovisioned device, they are provisioned separately, so we focused on provisioning a single device.

In this experiment, we measured the time used to perform each stage in the provisioning procedure when a Bluetooth non-mesh device (Waspmote) is provisioned by a provisioner designed for Bluetooth mesh (EFR32 and nRF52840), using different Bluetooth mesh stack implementations and different settings.

Moreover, we also evaluated our improvement proposal, the Lightweight Provisioning presented in Section 5.3, and then we conducted a comparison between both alternatives. Therefore, identifying which steps take more time provides us with deeper knowledge on how this process works and what could be improved.

Finally, the robustness of the standard provisioning procedure and the proposed Lightweight Provisioning are evaluated in a different scenario, in order to ensure the robustness of this procedure is not compromised by our proposal.

In all the following cases, the unprovisioned Waspmote device was configured in the same way: when it needs to send a message, it changes its role to broadcaster, sending the message using the three advertising channels for 60 ms, with an advertising interval of 60 ms. After the message sending, the device changes its role to observer, scanning the network for 1.5 seconds, with a scan interval of 1.5 seconds and a scan window of 100%. If no response is received, the unprovisioned device repeats the advertising and scanning steps up to a total of three times. In case of no response after these three attempts, the provisioning procedure is cancelled by the unprovisioned device, sending a provisioning failed PDU. For ACK packets, the unprovisioned device waits until 1.5 seconds. If no ACK is received, the unprovisioned device continues with the next transaction assuming that the ACK is lost, resending the previous transaction later if necessary.

Another common case is when a PDU is longer than the segment size. In this situation the PDU is fragmented into different segments, which are sent consecutively. For example, in the Public Key exchange, both the provisioner and unprovisioned device need to send three packets. These are sent once per advertising channel, consecutively, as shown in Figure 5.3.

In this scenario, the BLE 5.0 device configured as a provisioner was located at a distance of one metre in the sight line from the Waspmote device (BLE 4.0), to reduce the number of packet losses due to environmental or obstacle interference, as Figure 5.4 shows. For the same reason, the transmission power was adjusted to the maximum value, equivalent to +3dBm for Waspmote devices, to +10dBm for EFR32 devices and to +8dBm for nRF52840 devices. Both the provisioner and the unprovisioned device were connected to the com-

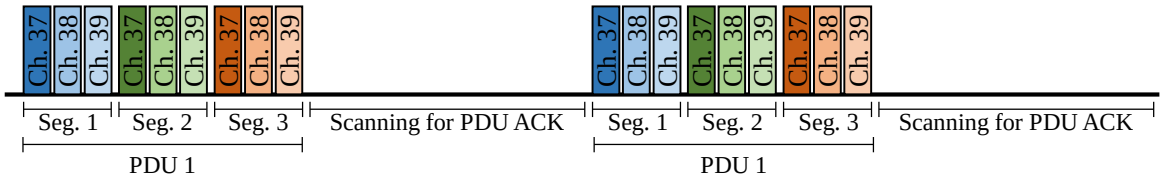


Figure 5.3: Device behaviour when sending and repeating a PDU fragmented into three segments, each sent in a different message.

puter through a USB cable, enabling the use of the computer clock as a common clock, in order to measure the time of the different events. Additionally, we used the nRF Sniffer connected to the computer, displaying all packets in Wireshark, to verify a correct provisioning procedure in each case, as well as the validity of the times collected by devices.

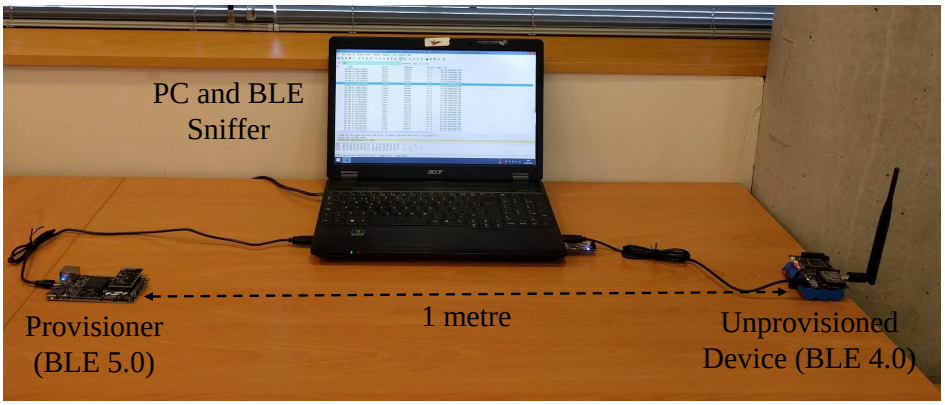


Figure 5.4: Table-top testbed for provisioning procedure evaluation.

In each evaluation, the time required to complete each individual provisioning procedure stage was measured; these stages are defined in Section 2.3.4. A time was defined for each of these stages: Invitation & Capabilities (T_{IC}), Exchanging Public Key (T_{PK}), Calculate the ECDH Shared Secret (T_{ECDH}), Authentication & Confirmation (T_{AC}) and Distribution of Provisioning Data (T_{PD}). These stages and their corresponding times are shown in Table 5.1. Although the calculation of the ECDH Shared Secret is usually included in the Public Key exchange, we found it useful to separate it because it is a local execution that can become very computationally expensive if the processor is not optimised for this task, as in the case of the Waspmote devices.

Table 5.1: Defined Times for each Provisioning Protocol Stage.

Provisioning Protocol Stage	Defined Time
Invitation & Capabilities	T_{IC}
Exchanging Public Key	T_{PK}
Calculate the ECDH Shared Secret	T_{ECDH}
Authentication & Confirmation	T_{AC}
Distribution of Provisioning Data	T_{PD}

5.4.1.1 Provisioning Time using EFR32

We used the Silicon Labs SDK for Bluetooth mesh to develop the necessary code to use the EFR32 Blue Gecko device as a provisioner device. The SDK from Silicon Labs allowed us to configure only certain parameters, such as TX power, with the provisioning procedure being an API call that cannot be modified or configured. The behaviour of the provisioning procedure is the following: the network is scanned constantly with a scan window of 100%; when transmitting, each message is sent only once per advertising channel. When a transaction is fragmented into more than one segment, packets encapsulating these segments are sent with a 25 ms interval between packets (advertising interval). If no response is received, the message is sent again each 3 seconds (retransmission interval). When the provisioner sends an ACK and also the next transaction, the ACK is sent periodically for 3 seconds before starting the next transaction, to ensure its correct reception. Experimentally, we verified that the provisioner sent an ACK for each complete provisioning transaction, except for the Provisioning Complete transaction, which could provide a better understanding of the results.

The time measured for each provisioning procedure stage, as well as the average time, is shown in Figure 5.5, divided into the main steps. Each step is explained as follows:

T_{IC} was 2.382 seconds on average. The time achieved by Test 1 and Test 6 is particularly remarkable in this stage, as it was significantly reduced. In the Invitation & Capabilities stage, the provisioner sent the Provisioning Invite transaction and the Provisioning Capabilities ACK, both of which were received by the unprovisioned device on the first attempt in Test 1 and Test 6. If the Provisioning Invite transaction was not received, the provisioner sent the transaction again after a timeout of 3 seconds, with the consequent delay, as can be noticed in Tests 3, 5, 7, 8, 9 and 10. Regarding the delay produced by the loss of the Capabilities ACK, it is more variable, since it is resent several times to ensure its reception. As mentioned above, when using the API provided by Silicon Labs, ACKs followed by a transaction from the same device (from provisioner to unprovisioned device in this case) are constantly resent until the following transaction is sent.

T_{PK} was, with an average time of 7.480 seconds, the longest step by far. The Provisioning Start transaction was sent by the provisioner after the Provisioning Capabilities ACK: To ensure reception of both messages, the provisioner left 3.2 seconds between the first ACK and the Provisioning Start transaction. The Public Key transaction from the provisioner was fragmented into three segments, with the time needed for its reception being highly variable. The Public Key segments from the unprovisioned device were received easily by the provisioner, but the unprovisioned device could need more than one ACK repetition, increasing the total time.

T_{ECDH} was a fixed time because the algorithm was executed locally, taking 4.666 seconds in all tests. Waspmote devices are not equipped with optimised processors for the computation of security functions and algorithms. This makes this step difficult to optimise, although it is only executed once to include the device in the mesh network.

T_{AC} includes some locally executed algorithms: the confirmation key calculation (symmetric encryption algorithm which takes 4-5 ms); obtaining the device confirmation value (2-3 ms); checking the provisioner confirmation value (2-3 ms). Regarding BLE transmissions, the transactions in this phase were one-segment transactions, and the behaviour can be summarised as follows: transactions from provisioner to unprovisioned device, which depended on the capability of our device to receive the packets, making the times oscillate between 100 to 5000 ms in the worst case, where repeating several lost packets (including ACKs) was necessary; and transactions from the unprovisioned device to the provisioner, which took a maximum of 180 ms due to the performance of the newest BLE version.

T_{PD} also includes algorithms executed locally, taking around 18 ms: calculating Session Key, Session Nonce and Device Key calculation; as well as decrypting and authenticating the provisioning data. Concerning BLE transmissions, the time needed to receive the two segments of the Provisioning Data depended on the number of repetitions the provisioner had to make. In the evaluation, this varied between 190 (received in the first sending) and 3150 ms (received when the segments were repeated). When the Provisioning Complete was sent, the unprovisioned device waited for the ACK, but the Silicon Labs implementation did not take this sending into account. This caused a delay of 3000 ms (the timeout specified in our implementation).

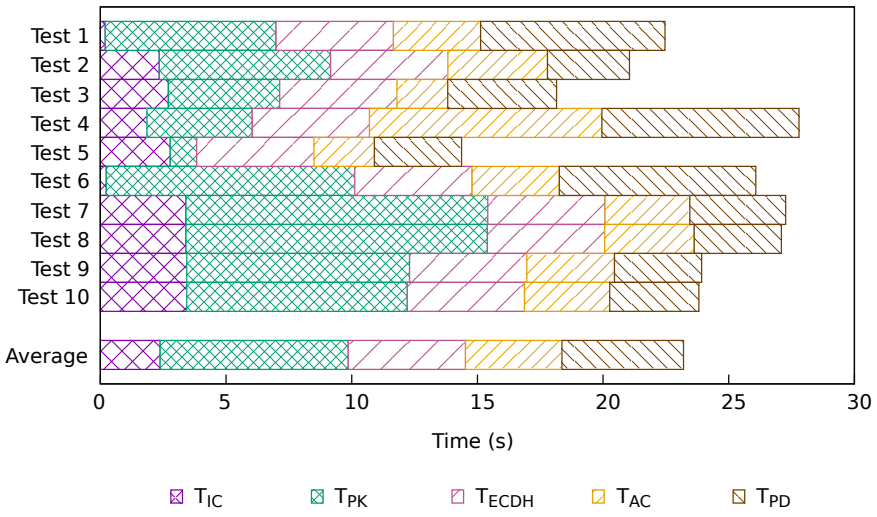


Figure 5.5: Time for each standard provisioning stage using EFR32 provisioner.

5.4.1.2 Provisioning Time using nRF52840

Due to the difficulties to modify the parameters and the configuration in the implementation of Silicon Labs, we also used the nRF52840 device as provisioner. The nRF52840 devices are compatible with Zephyr, which is fully open source and allowed us to evaluate our implementation of the provisioning procedure with a different provisioner.

The nRF52840 devices using Zephyr RTOS were configured as follows: the network was scanned constantly using a scan window of 100%; each message was sent once per advert-

ising channel using an advertising interval of 125 ms between different packets (in case of a multi-segment transaction). In case of no response, two different retransmission intervals were used for this evaluation: 1500 (see Figure 5.6) and 200 ms (Figure 5.7). After sending an ACK, the provisioner sends the following transaction immediately after, retransmitting the ACK if necessary.

T_{IC} averaged 0.962 and 0.503 seconds for retransmission intervals of 1500 and 200 ms respectively. The results show how in cases where all transmissions were correctly received on the first attempt, the time obtained was similar (Tests 1, 3, 5, 6, 7 and 8 for 1500 ms retransmission interval and Tests 1, 2, 3, 4, 5, 6, 7 and 9 for 200 ms retransmission interval). The time obtained in cases where an ACK was lost is also similar (Tests 2 and 10 for 1500 ms retransmission interval and Test 8 for 200 ms). In contrast, when a data packet was lost, the required time was greater when the retransmission interval was increased (Tests 4 and 9 for 1500 ms and Test 10 for 200 ms retransmission interval).

T_{PK} averaged 2.348 seconds using a 1500 ms retransmission interval and 2.256 seconds using a 200 ms retransmission interval. Again, there is no difference in this step when using different retransmission intervals. One of the main difficulties in this step is sending the public key, which is fragmented into three segments. When these segments were successfully received on the first attempt, similar times were obtained (Tests 1, 3, 4, 6, 7 and 9 for 1500 ms retransmission interval and Tests 7 and 8 and 10 for 200 ms retransmission interval). However, when these packets were lost (Test 2 and 8 for 1500 ms retransmission interval and Tests 3, 6 and 9 for 200 ms retransmission interval), the time increased more for the greater retransmission interval. Finally, if more retransmissions were needed, or whether the ACK needed to be resent after the following transaction was sent, the times obtained were very similar (Tests 5 and 10 and Tests 1 and 5 for 1500 and 200 ms retransmission interval, respectively).

As in the previous experiment, security algorithms were executed locally in T_{ECDH}, so in all tests the time measured was the same (4.666 seconds).

T_{AC} were on average 2.469 seconds and 1.172 seconds for 1500 and 200 ms retransmission interval, respectively. As stated above, this step includes some locally executed algorithms which took 11 ms to execute. Regarding BLE transmissions, all transactions in this step were one-segment transactions, and the time measured when all packets were received with no losses was similar (Tests 1, 3, 4 and 7 and Tests 1, 3, 4 and 6 for 1500 and 200 ms retransmission interval, respectively). When data packets were lost, both times were increased, impacting the 1500 ms retranmission interval (Tests 5 and 6) more than the 200 ms retransmission interval (Test 2, 7 and 9). This trend continued if a greater number of data packets or some ACKs were lost, with a greater impact on the 1500 ms retransmission interval (Tests 2, 8, 9 and 10) than on the 200 ms retransmission interval (Tests 5, 8 and 10).

T_{PD} were on average 1.011 and 1.139 seconds for 1500 and 200 ms retransmission intervals, respectively. This step also includes algorithms executed locally, which took around 18 ms. Relative to BLE transmissions, the time measured was similar when all packets were

received correctly on the first attempt, including the fragmented transaction (Tests 1, 2, 3, 4, 6, 7, 8 and 9 for 1500 ms retransmission interval and Tests 4, 8 and 10 for 200 ms retransmission interval). The impact of some data packet losses was greater for the 1500 ms retransmission interval (Test 10) than for the 200 ms retransmission interval (Tests 1, 2, 3 and 7), which was increased when several retransmissions were necessary (Test 5 for 1500 ms interval and Tests 5, 6 and 9 for 200 ms interval.

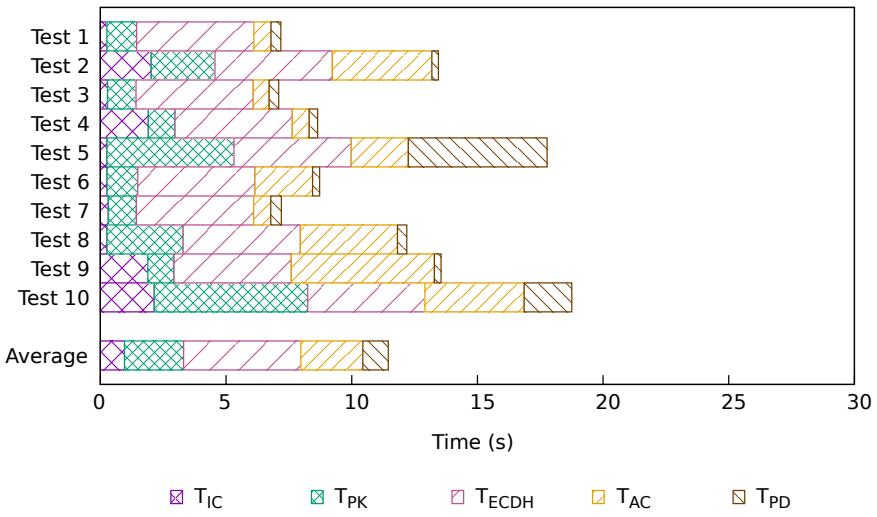


Figure 5.6: Time for each standard provisioning stage for nRF52840 provisioner using Zephyr (1500 ms retransmission interval).

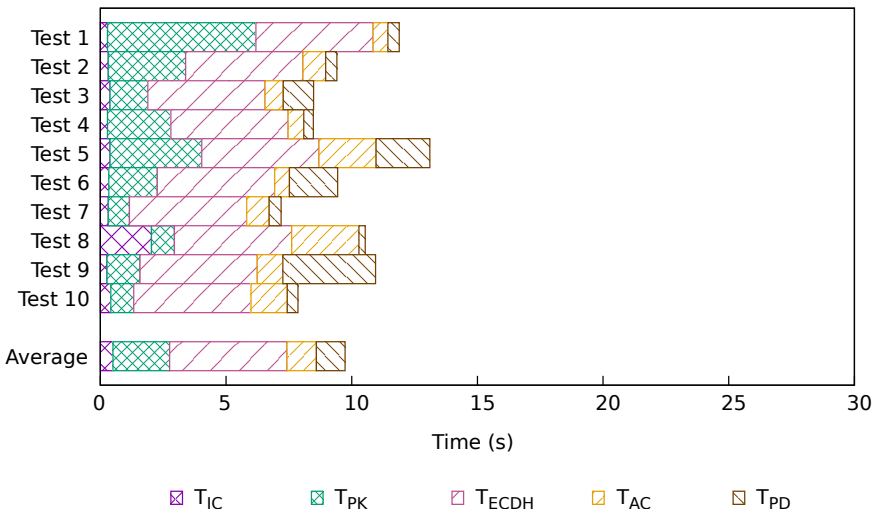


Figure 5.7: Time for each standard provisioning stage for nRF52840 provisioner using Zephyr (200 ms retransmission interval).

5.4.1.3 Lightweight Provisioning Time

The Lightweight Provisioning proposed in Section 5.3 was included in Zephyr RTOS and in our library in C for its evaluation. For this experiment an nRF52840 device was used as a

provisioner and a Waspmote as an unprovisioned device, with the same configuration used before. Waspmote device scanned the network using a scan interval of 1.5 seconds and a scan window of 100%; when sending a message, it was sent once per advertising channel using an advertising interval of 60 ms between different packets. The retransmission interval was 1.5 seconds. The nRF52840 device constantly scanned the network with a scan window of 100%; each message was sent once per advertising channel with an advertising interval of 125 ms between different packets. If necessary, a message was repeated using two different retransmission intervals: 1500 and 200 ms. The results obtained for each of these retransmission intervals are shown in Figure 5.8 and in Figure 5.9, respectively.

The Lightweight Provisioning proposal not only removes the time needed to send and receive ACK packets, it also avoids the problematic situation of when the same device has to send the ACK of the current transaction and start the next one. This would result in greater time savings in protocol stacks such as Silicon Labs, which wait for a time interval before starting the next transaction. However, as this implementation is closed-source, it was impossible to modify it. For the protocol stacks that choose to continue with the next transaction after sending the ACK as a Zephyr, the Lightweight Provisioning avoids a great part of the retransmitted packets, since the next transaction always starts at the same time on both devices.

T_{IC} averaged 0.298 and 0.175 seconds when 1500 and 200 ms retransmission intervals where used, respectively. All messages were received correctly on the first attempt, needing one more retransmission only in Test 1 when using the 1500 ms retransmission interval and in Test 2 when using the 200 ms interval.

T_{PK} averaged 1.866 and 0.649 seconds for 1500 and 200 ms retransmission interval, respectively. When the 1500 ms retransmission interval was used, only one retransmission was needed in Tests 2, 3, 7, 8, 9 and 10, while in the other tests all messages were received at the first attempt. For the 200 ms retransmission interval, a message was retransmitted only in Test 6 and 7, being successfully received on the first attempt in all other tests.

T_{ECDH} remains the same as in the previous experiments because it is executed locally. However, by reducing the time of the other steps, this step has become by far the most time-consuming, with 58% of the total time for the 1500 ms retransmission interval and 77.7% for the 200 ms retransmission interval.

T_{AC} averaged 1.141 seconds for 1500 ms retransmission interval and 0.349 seconds for 200 ms interval. In most tests, the messages were received on the first attempt, requiring an extra retransmission in Tests 2, 4, 6, 9 and 10 when a 1500 ms retransmission interval was used and in Tests 6 and 8 for the 200 ms interval.

T_{PD} were on average 0.074 and 0.166 seconds for 1500 and 200 ms retransmission intervals, respectively. The shorter time measured for the 1500 ms interval was because in all the tests conducted the messages were received correctly on the first attempt, while a

message had to be retransmitted in Tests 5, 7 and 9 when the 200 ms retransmission interval was used.

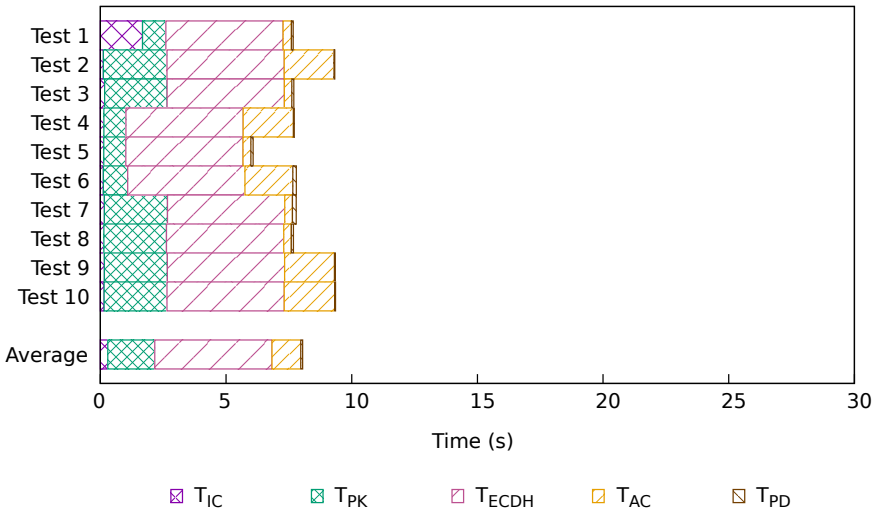


Figure 5.8: Time for each Lightweight Provisioning stage for nRF52840 provisioner (1500 ms retransmission interval).

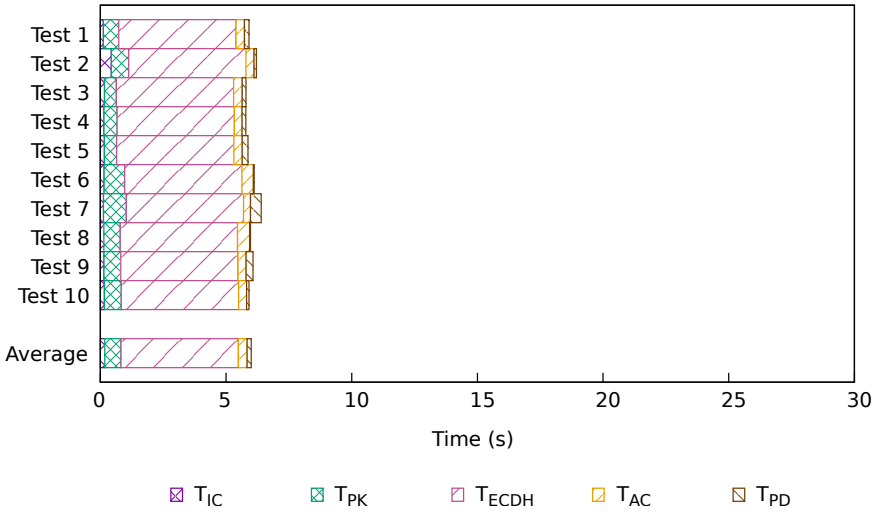


Figure 5.9: Time for each Lightweight Provisioning stage for nRF52840 provisioner (200 ms retransmission interval).

5.4.1.4 Provisioning Time Comparison

This subsection provides a comparative between the standard provisioning procedure and the proposed Lightweight Provisioning. The average of the results obtained in each case are shown in Figure 5.10.

Rhe results show, on one hand, the influence of the retransmission interval in the Zephyr implementation: the time to receive all the packets was shorter when using a lower interval, due to the higher delay to obtain the packet if it is not received at the first attempt. On

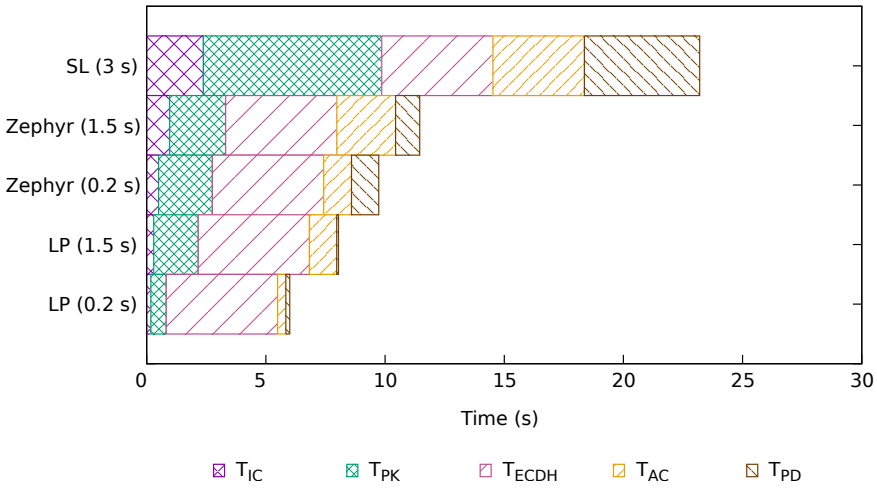


Figure 5.10: Average time of each provisioning stage for every evaluated configuration. SL: standard Provisioning of Silicon Labs SDK in EFR32 provisioner; Zephyr: nRF52840 provisioner using standard Provisioning in Zephyr RTOS; LP: the Lightweight Provisioning proposal. The number in brackets indicates the retransmission interval in seconds.

the other hand, there was a difference between Silicon Labs and Zephyr implementations, with this difference being due to the configuration used. In addition to the retransmission interval and the delay to send a new transaction after sending an ACK, the advertising interval between different packets is particularly remarkable. This interval is involved in sending transactions with several segments (Exchanging Public Keys and Distribution of Provisioning Data steps, the two with the highest increase), and is 20 ms in the Silicon Labs implementation, while the zephyr implementation uses a 125 ms interval. A 20 ms interval between packets is not enough for a Bluetooth non-mesh device to receive a packet, process it (requires decryption and authentication) and be able to receive another one. This causes the device to require a higher number of retransmissions, which, added to the higher retransmission interval, significantly increases the time needed to complete the provisioning procedure.

Regarding our Lightweight Provisioning proposal, great improvements were achieved when compared to the standard implementations. Our proposal improved by 29.77% and 38.31% compared to Zephyr using the same configuration, for 1500 and 200 ms retransmission intervals, respectively. Compared to Silicon Labs, there was an improvement of up to 74.12%, although the configuration used is different. Focusing only on the results obtained for our implementation, the time spent on security-related calculations is particularly remarkable. These calculations represent 58.35% and 78.18% of the total time for a transmission interval of 1500 and 200 ms, respectively. Although this operation cannot be optimised on the devices used, the time required for its execution will decrease dramatically when the devices running it have a specially designed coprocessor.

The results obtained in this experiment provide us with in-depth knowledge of the provisioning procedure. Although this procedure was conducted solely to include a new device

in a mesh network, it is necessary to reduce the execution time as much as possible. The Bluetooth mesh stack implementation from Silicon Labs provides a generic solution for most BLE devices, which increases the delays in most steps. However, this implementation is not open source, and its optimisation for specific devices was not possible. The open source implementation of Zephyr allowed us to evaluate another Bluetooth mesh protocol stack with different configurations, as well as to implement our proposal for the provisioning process: Lightweight Provisioning. After carrying out the experiments, a significant improvement in the time required to complete this procedure was demonstrated, due to the elimination of the time needed to send and receive certain ACKs, as well as by avoiding errors that delayed the reception of the right packets.

5.4.1.5 Robustness of Provisioning Procedures in a Real Environment

According to Bluetooth mesh specification [17], a provisioner is typically a smart phone or other mobile computing device, which allows users to provision an unprovisioned device directly, within its coverage range. Despite this, and the fact that this procedure is only performed once, a robust provisioning procedure is required in real environments in order to be completed in the shortest possible time and in the minimum number of attempts. This experiment was carried out as a demonstration of the robustness of our Lightweight Provisioning.

For this purpose, the unprovisioned device was located 20 metres away from the provisioner device in our research laboratory, as showed in Figure 5.11. The standard provisioning procedure implementation of Zephyr and our Lightweight Provisioning proposal were evaluated, using the same configuration than in the previous experiments. The unprovisioned device scanned the network using a scan window of 100% in a scan interval of 1.5 seconds; each transaction was sent once per advertising channel using an advertising interval of 60 ms between different packets. If necessary, a transaction was repeated using a retransmission interval of 200 ms. The nRF52840 device configured as provisioner constantly scanned the network with a scan window of 100%. Each transaction was sent once per advertising channel using an advertising interval of 125 ms between different packets. The retransmission interval used was 200 ms.

Figure 5.12 shows the results obtained using the standard provisioning procedure of the Bluetooth mesh specification. These results are discussed below:

T_{IC} averaged 0.970 seconds, 92.92% greater than the time obtained for the same setting with the devices located at a distance of 1 metre. The result obtained in Test 1, with 2.36 seconds, stands out significantly. In this case, the Provisioning Capabilities transaction required to be sent several times to be received by the provisioner. In Tests 6, 7 and 9, the time of this stage also exceeds 1 second, due to the resending of the Provisioning Capabilities transaction since the Provisioning Capabilities ACK was not received.

T_{PK} was on average 2.740 seconds, which is 21.47% more than the result obtained for the experiment carried out with the same settings at a distance of 1 metre. In this case,

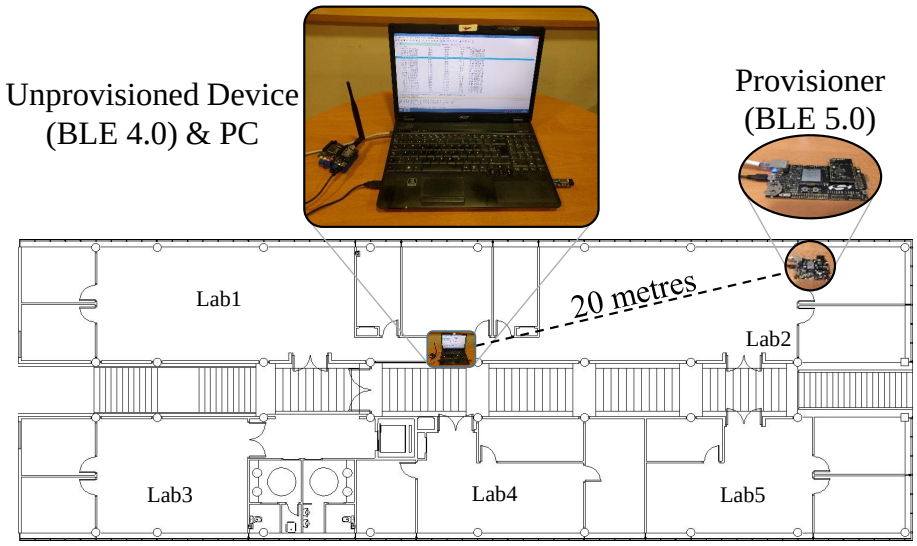


Figure 5.11: Network setup for provisioning procedure robustness evaluation.

the results were closer, since the Provisioning Public Key transaction is divided into three segments, being one of the most challenging to receive on the first attempt, regardless of the distance. The longest time was obtained in Test 10, although it is even shorter than the worst result obtained with devices located 1 metre away.

T_{ECDH} averaged 4.661 seconds. As mentioned above, this stage is executed locally on the device with the data received in the previous transactions, so its execution time is not affected.

T_{AC} was on average 2.652 seconds. This is 126.24% greater than the result obtained when the provisioner and unprovisioned devices were located 1 metre away. Tests 1, 3, 4, 8 and 9 are particularly notable, since in all of them the Provisioning Random transaction from provisioner device was required to be resent several times, due to the loss of the corresponding ACK, as well as Test 10, where the Provisioning Confirmation transaction from provisioner device was resent multiple times, due to the loss of its ACK.

T_{PD} averaged 1.861 seconds. This represents an increase of 63.41% compared to the result obtained in the experiment with the devices located 1 metre away. The results obtained in Tests 6, 8 and 9 are remarkable, as in all of them multiple resendings of the Provisioning Data transaction were required, which is divided into two segments.

The results show how the time required to complete each stage of the standard provisioning procedure increased in all stages (except for the ECDH Calculation stage, which remains unchanged), compared to the results obtained when the devices were located 1 metre away. However, this increase was not uniform across all stages. The greatest increases corresponded to the stages where the transactions were not divided into segments, since these transactions were correctly received in the first attempts when they were in a more friendly environment. For stages with transactions divided into segments, the execu-

tion time was increased slightly. This is because these transactions had to be resent also in the scenario where the devices were located 1 metre away, due to the limitations of BLE 4.0 devices.

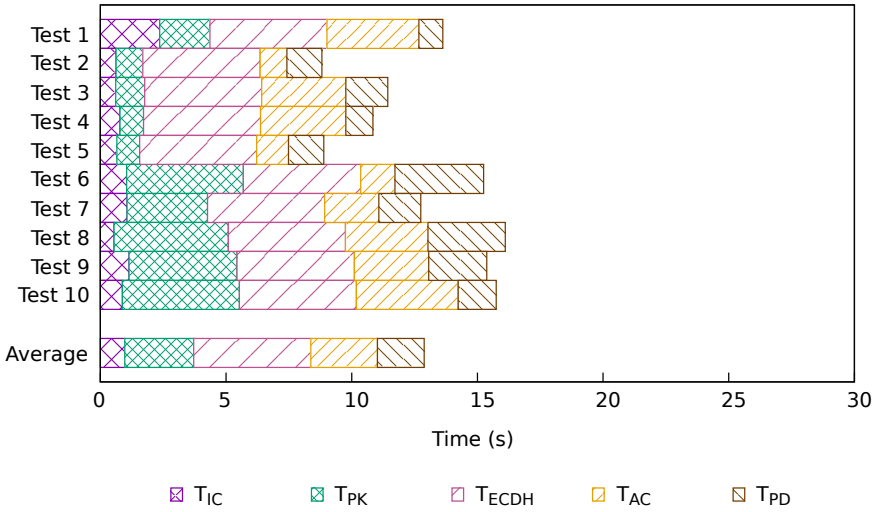


Figure 5.12: Time for each standard provisioning stage for nRF52840 provisioner located 20 metres from the unprovisioned device.

Regarding the Lightweight Provisioning, Figure 5.13 shows the results obtained for our proposal when the unprovisioned device is located 20 metres from the provisioner.

T_{IC} averaged 0.433 seconds, 147.31% greater than the result obtained using the same configuration with the devices located 1 metre away. Test 8 is particularly notable, since the Provisioning Capabilities transaction required multiple resendings in order to be correctly received.

T_{PK} was on average 1.012 seconds,which is 55.98% more than the result obtained using the same configuration when the devices were located 1 metre away. The longest time was obtained in Test 8, where several resendings of the Provisioning Public Key transaction from the provisioner were necessary to receive its three segments.

T_{ECDH} averaged 4.614 seconds, a time similar to that obtained in the other experiments. As in the results obtained with this configuration when the devices were located at a distance of 1 metre, this stage represents most of the time, 56.45% of the total time of the procedure, and depends especially on the hardware of the device.

T_{AC} was on average 1.013, which is 190.17% more than the time obtained for the experiment with the devices located 1 metre away. The result obtained in Test 7 is particularly noteworthy, since it was necessary to resend the two Provisioning Confirmation transactions, both the one sent by the provisioner and the one sent by the unprovisioned device.

T_{PD} averaged 1.101 seconds, 563.49% longer than the time obtained when the devices were located 1 metre away. The greatest time by far was obtained in Test 7. In this case,

the reception of the two segments of the Provisioning Data transaction required several resendings, receiving in many of them the segment previously received.

The results of this experiment also show an increase in the time of all the stages compared to the results obtained when the devices were located 1 metre away when using the Lightweight Provisioning. Similarly, the time increase was greater for the stages where all transactions have a single segment (except Test 7). This is due to the difficulties of BLE 4.0 devices receiving multiple-segment transactions, even if they are 1 metre away.

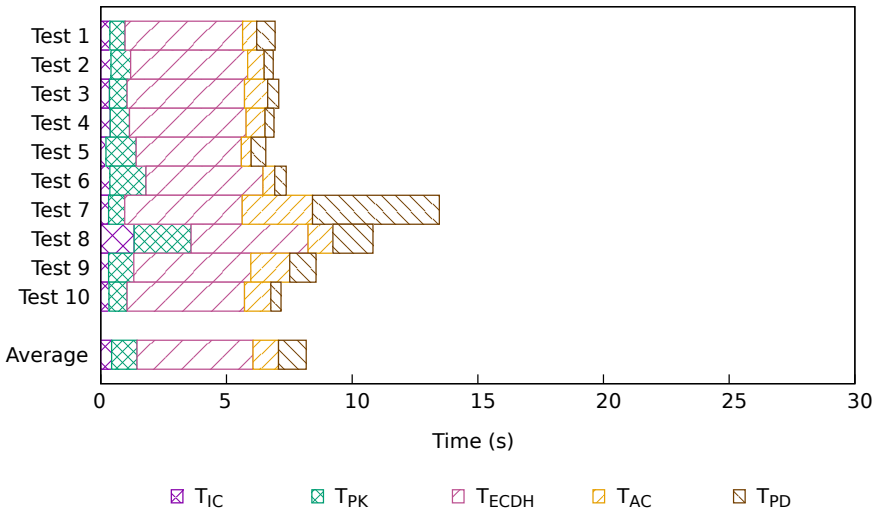


Figure 5.13: Time for each Lightweight Provisioning stage for nRF52840 provisioner located 20 metres from the unprovisioned device.

The experiment described in this section shows a 36.56% reduction in the time required to provision an unprovisioned device when using our Lightweight Provisioning proposal instead of the standard provisioning procedure. Moreover, this experiment also confirms that our proposal maintains the robustness required to operate at a certain distance in real environments where other wireless networks may co-exist.

5.4.2 Experiment 2. End-to-End Mesh Delay

Once the provisioning is completed, the mesh behaviour is based on a controlled flooding. In this type of multi-hop networks, the latency of a message from a source node to a destination node is increased by each relay node. This point is especially critical in Bluetooth mesh, where mesh messages are encrypted and obfuscated to guarantee network security. These processes, however, increase the processing time of each node. Can the security algorithms be executed without increasing the latency to send critical messages? To answer this question, this experiment aimed to evaluate the latency that a device equipped with the lowest BLE version could add to the mesh network, measuring the end-to-end delay in a two-hop network.

For this experiment, a mesh network was deployed in our research institute, the I3A, with three devices: two EFR32 devices equipped with BLE 5.0 and a Waspmote device with BLE 4.0 (see Figure 5.14). The EFR32 devices were used as source and destination nodes, being placed at a distance of 40 metres, while the Waspmote device was configured to relay the received messages, being placed in the middle, at a distance of 20 metres from source and destination nodes. All the nodes were connected to a computer through a Virtual COM Port, using USB (for Waspmote devices) and Ethernet (EFR32 devices) interfaces. This connection enabled the computer clock to be used as a common clock between devices for collecting time data. Moreover, the nRF Sniffer and Wireshark were used in the same computer to monitor the Bluetooth mesh network traffic in order to detect any disturbance. All the nodes were configured at the minimum transmission power (-23 dBm for Waspmote devices and 0 dBm for EFR32 devices) and were not in a sight line, to limit their coverage and force the messages to cross through the relay node, as shown in Figure 5.14.

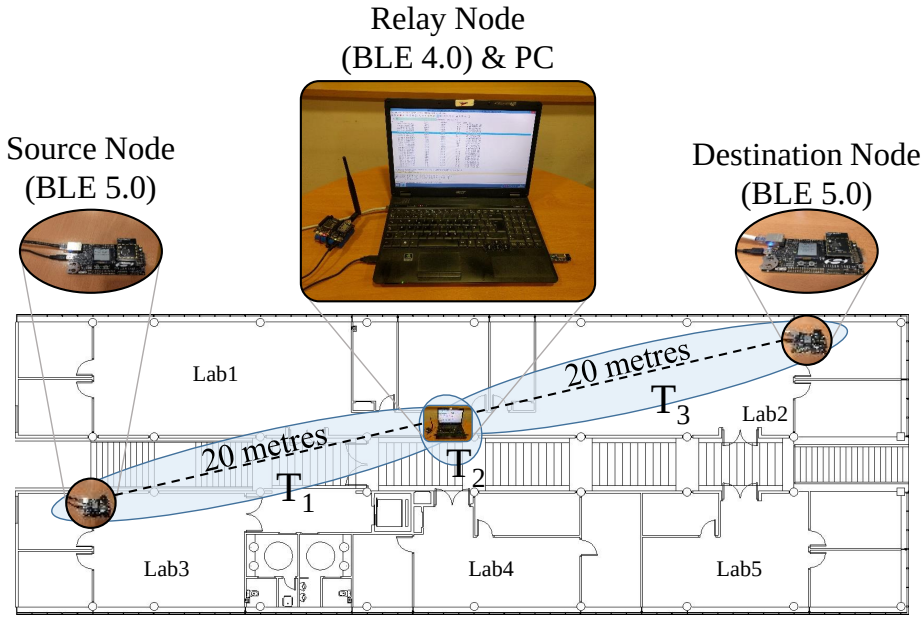


Figure 5.14: Network setup for end-to-end delay evaluation.

Once the mesh network was established and provisioned, 200 packets were sent from the source node to the destination node, with a new packet being sent each 5 seconds using the three available advertising channels (37, 38 and 39). Each packet was repeated three times per advertising channel to ensure its correct reception. The time between repetitions of the same packet was 50 ms. For example, a packet P_0 was sent in a time T_0 using the three advertising channels; after 50 ms (T_{50}), P_0 was resent again through the three advertising channels; after another 50 ms (T_{100}), P_0 was again resent using the three advertising channels. A new packet P_1 was sent for the first time 5 seconds after the first sending of P_0 (T_{5000}).

The following instants of time were measured: when each packet was sent by source node, when it was received and sent by relay node, and when it was received in the des-

tination node. Figure 5.15 shows these results, where horizontal bolder lines represent the median, black boxes represent the 25^{th} and 75^{th} percentiles, and extreme values are enclosed by two black bars. Additionally, statistical outliers are represented by black dots. The different times measured are presented in Table 5.2 and detailed below:

- T_1 includes the transmission time and the propagation time, for each packet, from the source node to the relay node. This is the time from the first bit of the first packet repetition leaving the source node until the last bit of the packet is received in the relay node.
- T_2 is the processing time for each packet in an intermediate node. This time includes all processes of the Network PDU processing flow, where we should emphasise the deobfuscation, decryption of the packet and the subsequent encryption and obfuscation.
- T_3 includes the transmission time and the propagation time for each packet from the relay node to the destination node.
- T_T is the total time, the time taken for each packet to be transmitted across the mesh network from source node to destination node (the end-to-end delay) in a two-hop mesh network, through a BLE 4.0 relay node.

Table 5.2: Definition of Times.

Time	Definition
T_1	$T_{tsr} + T_{psr}$
T_{tsr}	Transmission time: source node $\longrightarrow$ relay node
T_{psr}	Propagation time: source node $\longrightarrow$ relay node
T_2	Processing time in an intermediate node.
T_3	$T_{trd} + T_{prd}$
T_{trd}	Transmission time: relay node $\longrightarrow$ destination node
T_{prd}	Propagation time: relay node $\longrightarrow$ destination node
T_T	$T_1 + T_2 + T_3$

T_1 and T_3 are related to the packet delivery time. While T_1 was 39 ms on average, the mean value for T_3 was only 12 ms. This difference is due to the receiver efficiency. While the receiver device in T_1 was the Waspmote node equipped with BLE 4.0, the destination node in T_3 used BLE 5.0. The improvements in higher BLE versions mean devices are more reliable when receiving data. This provides correct reception when the relay node sends the first packet, with no need to repeat it in most cases. This successful packet reception on the first attempt allows the packet delivery time between the relay node and the destination node to be minimised.

T_2 corresponds to the processing time in the relay node, the Waspmote equipped with a BLE 4.0 radiochip using our implementation. As expected, this is the most regular time, being 9 ms on average and varying between 8 and 10 ms in exceptional cases.

Finally, T_T illustrates the total time, with a mean value of 60 ms. According to R. Miller, a response time of 100 ms is perceived as instantaneous [63], so the results obtained in the network delay make our Bluetooth mesh valid for use, for example, in Industry 4.0. The higher part of the total time is T_1, due to the time needed by BLE 4.0 device to correctly receive the mesh packets. Although, this part is probably one of the most difficult to optimise because of BLE 4.0 limitations, some options could improve the current behaviour. For example, synchronising the clocks of the devices in the network setup, along with establishing certain sending policies, would enable devices to manage scanning/advertising roles, thus being in the proper role exactly when required.

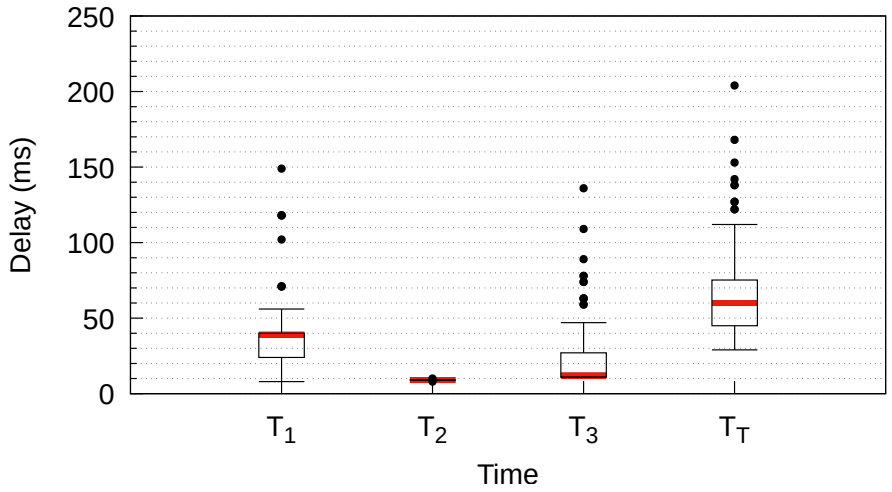


Figure 5.15: Time per different phase when transmitting 200 packets (ms).

Although these possible proposals may reduce the delay obtained, it should be noted that the results obtained were for an experimental two-hop network. Certainly, the number of network hops is directly proportional to the end-to-end network delay. Thus, in end-to-end transmissions in large-scale networks, the delay produced may be too high to provide a real-time user experience, despite the data being correctly transmitted through the network. However, this is a problem common to multi-hop networks, even in Bluetooth mesh networks using the latest BLE versions. This is widely discussed in [64], where Nordic devices especially designed for use in Bluetooth mesh are used. In this article, the authors achieve an end-to-end delay of approximately 23.2 ms in a two-hop Bluetooth mesh network. They also prove that this delay does not depend only on the number of hops, but also on the number of neighbours relaying the packets (the higher the number, the lower the delay), and also on network interference (the more interference, the higher the delay).

This experiment has shown that our implementation enables any BLE device with broadcast capabilities to be part of a Bluetooth mesh, even in networks that require real-time transmissions. However, the delay introduced by the devices using the initial BLE versions in large-scale networks may prevent users from having a correct experience when the packets need to be transmitted between two nodes located at a great number of hops.

5.4.3 Experiment 3. Packet Reception Rate (PRR)

The previous experiments proved how our implementation allowed a Waspmote node using BLE 4.0 to be provisioned and take part in the mesh network, without excessively increasing the network latency. However, the Waspmote devices do not support more than one role simultaneously (like most BLE 4.0 devices), and their performance is lower compared to higher BLE versions. This means they cannot process an incoming packet if the microcontroller is performing another action (such as processing the previous packet). Consequently, the number of lost packets is greater than in devices with higher versions and with more computational power. For example, EFR32 devices use BLE 5.0 and event-driven programming, more adapted to this use case. To guarantee the best performance in all BLE versions, the Bluetooth mesh standard proposes repeating each packet a certain number of times (usually three) to enable its reception by all devices, although this increases battery consumption.

This experiment aimed to evaluate the performance of devices with the lowest BLE version, using different configurations. To this end, a BLE device configured as a source node was placed at a distance of 50 cm from a Waspmote device and an EFR32 device, minimising the packet loss due to obstacles. Additionally, the three devices were connected to a computer through their USB ports, allowing the experiment to be tracked. The BLE sniffer was also connected to this computer, permitting verification of the correct sending of every packet. Figure 5.16 illustrates the network setup for this experiment.

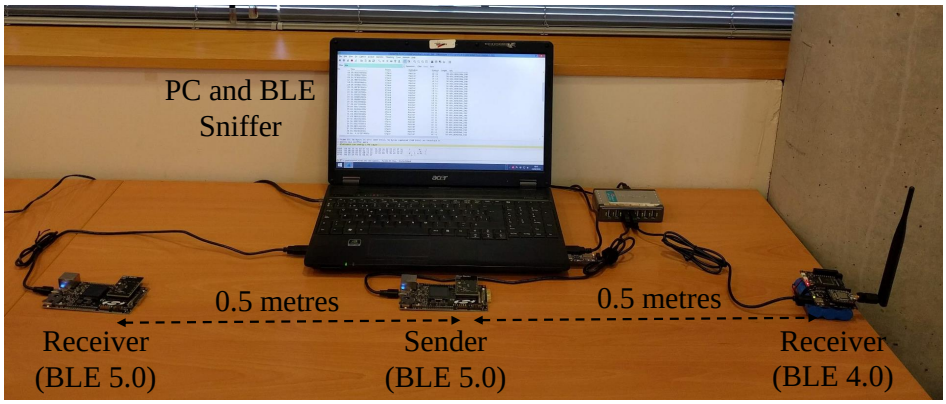


Figure 5.16: Table-top testbed for packet reception rate evaluation.

After provisioning, relay devices received and relayed the mesh packets, testing different configurations. In each configuration, a total of 200 packets were sent and, if a packet was sent more than once, the time between each repetition was 25 ms. Each individual packet was sent through the three advertising channels. Thus, sending a single packet entailed sending it three times, once for each advertising channel, while three repetitions meant a total of nine sendings, three for each channel. In the cases with more than one sending, a complete sending was performed by the three advertising channels before repeating the sending. The chosen configurations evaluated the PRR using two different parameters: (1) the number of repetitions of each packet, varying between 1, 2 and 3; and (2) the interval

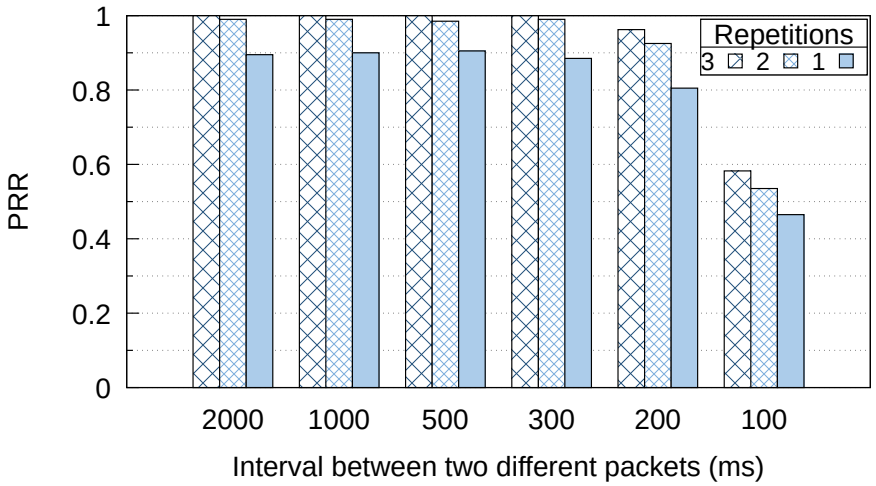


Figure 5.17: Packet Reception Rate varying the number of repetitions and the interval between different packets.

between two consecutive different packets, which was configured to 2000, 1000, 500, 300, 200 and 100 ms. The PRR of the BLE 5.0 device obtained for all configurations was 100%, while the PRR of the BLE 4.0 device for each configuration is represented in Figure 5.17.

As expected, the PRR was lower when the number of repetitions of each packet was lower. Taking as a reference the three repetitions (the best case), to repeat each packet twice reduced the PRR by only 2.25% on average, while to repeat each packet only once meant a reduction of 11.75%, on average, in the PRR. However, this higher variation is related to the interval between two different consecutive packets. Focusing on this parameter, we can appreciate, graphically, the minimum interval necessary to allow the Waspmote device to receive, process and relay the mesh packets. As previously explained, this device does not support scanning and broadcasting simultaneously, and a new packet cannot be received while the microcontroller is performing another task. This delays the reception of a new packet for the time used in the previous packet. This experiment demonstrates this effect, placing the minimum interval between two consecutive packets for suitable performance at 200 ms.

5.5 Conclusions

This chapter presents a double contribution. Firstly, the implementation and evaluation of the Bluetooth mesh specification in devices using BLE 4.0, including the provisioning protocol. Secondly, a proposal to improve the standard provisioning procedure, called Lightweight Provisioning.

Our implementation allows Bluetooth non-mesh devices to be provisioned and to be part of a Bluetooth mesh network, without the use of an additional device as a bridge (proxy feature). According to Bluetooth SIG, Bluetooth mesh is compatible with all BLE devices from BLE 4.0 [21] that support broadcaster and observer roles. To verify this, we imple-

mented the provisioning protocol and the different layers of the Bluetooth mesh protocol stack.

Different experiments were conducted to evaluate the implementation developed, verifying its correct operation in small and medium-scale networks. Although the performance is lower than that of BLE 5.0 devices due to the version's limitations, BLE 4.0 devices can be updated and used in new Bluetooth mesh networks. This will enable a large number of current devices to be used (saving money) in a network topology much demanded by IoT and Industry 4.0.

The Lightweight Provisioning, our proposal to simplify the Provisioning procedure, enables a device to be part of the Bluetooth mesh network with a lower exchange of messages, reducing the time needed to perform the provisioning procedure by 36.56%. Moreover, the Lightweight Provisioning maintains the level of reliability and security of the standard provisioning procedure, adapting it to devices that are not optimised for a rapid exchange of packets.

CHAPTER 6

Optimisation of the Friendship Mechanism

This chapter presents the study of different proposals considered as an alternative to the current friendship mechanism. The alternatives studied are: (1) improving the Bearer Layer, (2) improving time synchronization and (3) improving advertising channel utilisation. However, after conducting different analyses and experiments, the limitations of these approaches meant their final implementation was unfeasible. For that reason, this chapter also presents our proposal for friendship with Burst Transmissions and Listen Before Transmit (BTLBT) technique as an optimisation to the standard friendship mechanism.

This chapter firstly analyses the different proposals for the optimisation of the friendship mechanism, as well as the reasons for their unfeasibility. Then, our BTLBT proposal for friendship is described in detail and evaluated in hardware platforms. Finally, the consumption study conducted is presented.

6.1 Preliminaries

Since its first release, BLE has provided users with point-to-point (1:1) and broadcast (1:m) communications. These topologies enable the deployment of small IoT networks, but do not fulfil the requirements demanded by the latest IoT trends, such as smart buildings, smart factories (where the term IIoT has acquired great importance) and smart cities. These new concepts require total coverage, high reliability and sustainability. In this context, mesh topology for BLE appeared, firstly as a proprietary or academic solution, and finally in the Bluetooth mesh specification [17].

Although Bluetooth mesh provides a solution that fulfils the requirements of the latest IoT trends, the mesh nodes must constantly scan the medium in order to relay the messages received. This implies a great energy consumption that battery-powered nodes cannot deal with. For these cases, Bluetooth mesh provides the friendship relationship, described in

detail in Section 2.3.1.1. In this mechanism, FNs constantly scan the medium and store messages destined to LPNs, enabling these LPNs to be in sleep mode most of their lifetime. LPNs wake up periodically, requesting to the corresponding FN, which sends the messages stored for them.

The role of the FN in Bluetooth mesh is necessary to avoid exhausting batteries. However, it cannot currently be adjusted to the needs of specific applications [65]. This may be due to the fact that this is the first version of the Bluetooth mesh specification, and, given the novelty of the specification, the friendship mechanism is insufficiently optimised. The current standard friendship mechanism is based on a stop and wait protocol and, although it is simple to implement, has highly inefficient channel utilisation.

6.2 Initial Improvement Proposals

In our aim to optimise transmissions between LPNs and FNs, three different approaches were studied: (1) improving the Bearer Layer, through the use of the advertising extensions proposed in the Bluetooth 5.0 specification [66]; (2) improving time synchronization, which would enable LPNs to receive messages at precisely defined time intervals; (3) improving advertising channel utilisation, which would prevent every request and message from being sent through all three advertising channels.

This section presents the above proposals and the reason leading us to think another approach would be better.

6.2.1 Improving the Bearer Layer

In [67], the author proposes to enhance the bandwidth and the power consumption of Bluetooth mesh by improving the Bearer Layer, either as a proprietary solution or as a future modification in the specification. Introducing an improvement in this layer would mean obtaining better results in the upper layers without modifying them, thanks to the abstraction of the bearers.

One possibility to reach this improvement could be to use the Advertising Extensions, features defined in the Bluetooth 5.0 specification [66] and later. These Advertising Extensions enable devices:

- To use packets of up to 255 bytes, instead of the current 37 byte size limitation.
- To chain advertisements, being able to transmit up to a total of 1650 bytes in a row.
- To use the 37 data channels to send advertisements. In Bluetooth 5.0, these channels are also known as secondary advertising channels. This allows a greater number of channels to be available, in order to avoid collisions. This also reduces the advertising interval between packets (from 20 ms to 7.5 ms).

This approach could improve the performance of the Bluetooth mesh by using all possible upgrades of the lower layers of Bluetooth 5.X specifications. However, compatibility with previous versions (4.0, 4.1 and 4.2) would be lost. For example, advertising packets sent over data channels (or secondary advertisement channels) use a different preamble to avoid confusion, so they are discarded by BLE 4.X devices. This goes against the current Bluetooth mesh standard, which emphasises compatibility between all versions. Furthermore, it also goes against our heterogeneous network philosophy, and thus we decided to discard this option.

6.2.2 Improving Time Synchronization

Another encouraging approach is the synchronization of FNs and LPNs. The maximum consumption of a LPN is given by the sending and especially the reception of messages, where the scanning time required to receive a message is fundamental: the greater this time, the greater is the power consumption in this type of nodes. If LPNs and FNs were synchronised accurately enough, LPN requests could be suppressed and the receive windows reduced to the minimum.

With regard to synchronization in wireless sensor networks, a flooding architecture using implicit time synchronization is proposed in [68]. In this proposal, however, the devices periodically scan the network to receive and relay messages, keeping synchronised. Although this approach could be studied for RNs, which might benefit from this architecture, it is not suitable for LPNs since these only wake up to send data or request their messages from the FN.

Another alternative is to use synchronization algorithms between the FN and the LPN, such as the Lightweight Time Synchronization (LTS) algorithm. This algorithm has already been used in sensor networks [69] and BLE [70]. The LTS algorithm requires the exchange of three packets between two nodes, permitting them to estimate the difference between the two clocks and adjust them precisely (down to the micro-second level). The drawback of this algorithm is the requirement to resynchronize the devices from time to time in order to maintain their synchronization, depending on the accuracy of the clocks. For example, Nordic nRF52840 DK devices (see Section A.1.5) used in our experiments have the following clock sources [71]:

- 64 MHz on-chip oscillator.
- 64 MHz crystal oscillator, using external 32 MHz crystal.
- 32.768 kHz RC oscillator, with two modes of operation, normal mode (500 parts per million (ppm) and 0.7μA) and ultra-low power mode (2000 ppm and 0.3μA), enabling the user to trade power consumption against accuracy of the clock.
- 32.768 kHz crystal oscillator, using external 32.768 kHz crystal.
- 32.768 kHz oscillator synthesised from 64 MHz oscillator

Since we are focusing on the friendship mechanism, using high frequency sources is undesirable due to their higher consumption. As for the use of the included low frequency source, its accuracy is too low (500 ppm) to be efficiently used for synchronization tasks. The best option to maintain low power consumption with good accuracy is to use a low frequency external crystal that provides us with high accuracy maintaining a low power consumption. An example of the type of crystal oscillator that could be used is the Epson TG-3541CE [72]. This oscillator offers high accuracy (3.4 ppm) over a wide temperature range. Using these crystal oscillators, in the worst case scenario, the two node clocks drift with 6.8 ppm. The minimum synchronization interval to maintain the clock synchronization of two nodes with an error smaller than 1 ms is therefore around 150 seconds. Despite the high accuracy and low power consumption of the oscillator, it is not possible to use synchronization to reduce the power consumption of the current friendship mechanism, which enables the LPN to remain in sleep mode for up to 96 hours due to its asynchronous requests.

6.2.3 Improving Advertising Channel Utilisation

Another possible improvement approach is the optimisation of the use of the advertising channels, in order to avoid medium congestion. As explained in Section 2.3.1, Bluetooth mesh uses the three advertising channels to send broadcast messages. In Bluetooth mesh, each message is usually sent at least once by each of the advertising channels. Furthermore, sending critical messages more than once per channel is recommended. Is it possible to achieve a more optimised use of the medium without affecting performance?

Focusing on the friendship mechanism, would it possible for LPNs and FNs to communicate using a single channel? In that case, the power consumed sending three messages could be reduced, as well as avoiding an excessive use of the channels. For this purpose, we conducted three experiments that measured the impact on performance of the use of the channels. In all three experiments, we used 1 FN and 4 LPNs, which wake up simultaneously to request messages.

For all three experiments, identical parameters were established: a 60-second PollTimeout, a 255-ms ReceiveWindow and a 100-ms ReceiveDelay. In each experiment, the FN had 160 messages stored in each Friend Queue, which were sent after the corresponding requests from the LPNs. In the first experiment (see Table 6.1), the FN and the LPNs sent all messages through the three advertising channels. In the second experiment (see Table 6.2), the FN and the LPNs sent the messages through advertising channel 37. In the third experiment (see Table 6.3), each LPN used a different channel to communicate with the FN.

As the results show, of the 160 messages sent by the FN, an average of 100% of packets were correctly received when using the three channels compared to 93.44% when using a single channel and 75.78% when each LPN uses a different single channel. Using a single channel produces less medium saturation, as well as less consumption per request sent.

Table 6.1: Results for 1 FN and 4 LPNs using the three advertising channels.

Node	LPN1	LPN2	LPN3	LPN4
Advertising channel	All	All	All	All
Received messages	160	160	160	160
Lost messages	0	0	0	0
Sent Polls	175	173	175	176
Lost Polls	5	3	5	6
Terminated friendship	0	0	0	0

Table 6.2: Results for 1 FN and 4 LPNs using the same single advertising channel.

Node	LPN1	LPN2	LPN3	LPN4
Advertising channel	37	37	37	37
Received messages	147	157	160	134
Lost messages	16	17	16	42
Sent Polls	239	243	249	222
Lost Polls	80	73	77	78
Terminated friendship	0	2	1	1

However, the PRR decreases, and, therefore, the number of requests increases. The main reason for packet loss is that the radio is only capable of scanning a single channel at any given time, so it switches between different advertising channels when scanning the medium (as explained in Section 2.2.3.2. Due to the scanning and advertising intervals, it is possible to receive the messages sent through the three advertising channels, but not when they are only sent using one of them.

Finally, each lost request means that the LPN will scan the network during the interval set by the ReceiveWindow parameter, increasing consumption more than the reduction achieved in sending the request through a single channel. The loss of several of these requests means the termination of the friendship, which causes the FN to discard the packets still stored in the Friend Queue, which are permanently removed. Given the poor results obtained, we decided to omit this approach, focusing on another proposal to improve the current friendship mechanism.

6.3 Our Improvement Proposal: Bursts Transmissions and Listen Before Transmit (BTLBT)

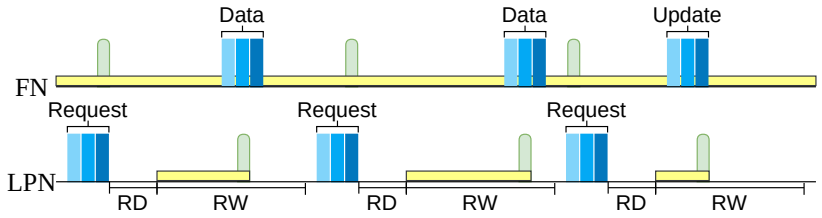
In the experiments described in Section 6.2, a maximum ReceiveWindow (255 ms) was established. However, after receiving a message, LPNs stop scanning the medium. The average time from the beginning of the ReceiveWindow to the reception of the message was 92.27 ms for 1 LPN requesting to the FN, increasing to 147.06 ms for 4 LPNs simultaneously. These results show that an optimisation in the medium access could considerably reduce the time of the ReceiveWindow, with the consequent reduction of consumption in the LPN.

Table 6.3: Results for 1 FN and 4 LPNs using a different single advertising channel.

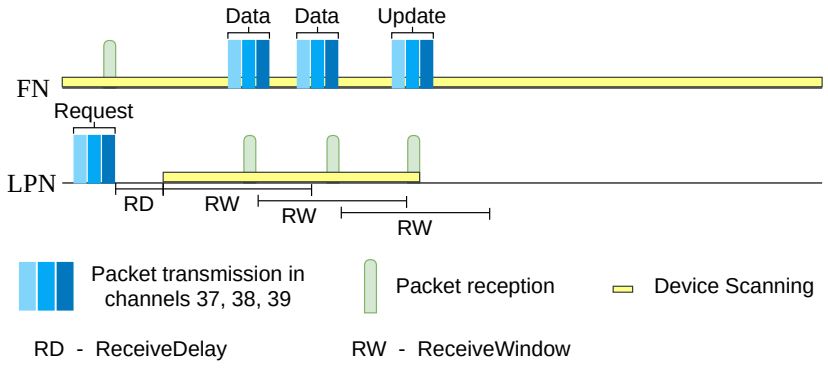
Node	LPN1	LPN2	LPN3	LPN4
Advertising channel	38	39	37	38
Received messages	127	76	154	128
Lost messages	10	12	14	13
Sent Polls	251	220	238	265
Lost Polls	111	126	74	123
Terminated friendship	2	7	0	3

To optimise medium access, the use of burst transmissions is proposed, where a single LPN request initiates the sending of a burst of messages from the FN. This enables the time of the ReceiveWindow to be optimised, enabling a greater number of messages to be received in less time and avoiding the sending of a request for each message.

In our proposal, after receiving a message from the FN, the LPN resets the ReceiveWindow time, waiting for a new message. This continues until the maximum number of messages per burst or an update packet is received. Moreover, maintaining the flexibility of the ReceiveWindow interval allows the FN to perform other functions if needed, such as relaying a received Bluetooth mesh message. This proposal not only allows LPNs to avoid the sending of multiple requests, but also improves the time it takes to receive the messages, enabling them to reduce the total scanning time, and, therefore, to reduce consumption. Figure 6.1a shows an example of Bluetooth mesh friendship standard transmissions while Figure 6.1b illustrates an example of our proposal of Bluetooth mesh friendship burst transmissions.



(a) Bluetooth mesh standard friendship.



(b) Burst transmission friendship.

Figure 6.1: Examples of transmissions using Bluetooth mesh standard friendship and burst transmission friendship.

As already explained at the beginning of this chapter, the number of LPNs simultaneously requesting affects the performance of the FN, which takes longer to respond to requests, increasing the size of the ReceiveWindow and the power consumption in LPNs. This also applies to burst transmissions, increasing the scanning time. Therefore, our proposal involves another improvement in this aspect: Listen Before Transmit (LBT), which works is a similar way to Carrier Sense Multiple Access (CSMA). Due to the combination of these two techniques, we decided to denominate our proposal Burst Transmissions and Listen Before Transmit.

Before requesting, an LPN listens to the shared medium to determine whether the FN is transmitting to another LPN. The listening time is randomly decided and ranges from 1 to 2 times the size of the ReceiveWindow. If the listening LPN detects an FN transmission, it waits in sleep mode for a randomly selected period of time known as backoff period. If the listening time ends without detecting a transmission, the LPN sends its request. Figure 6.2 shows the state machine of an LPN in our proposal, which can be compared to the state machine of a standard LPN in Figure 2.11.

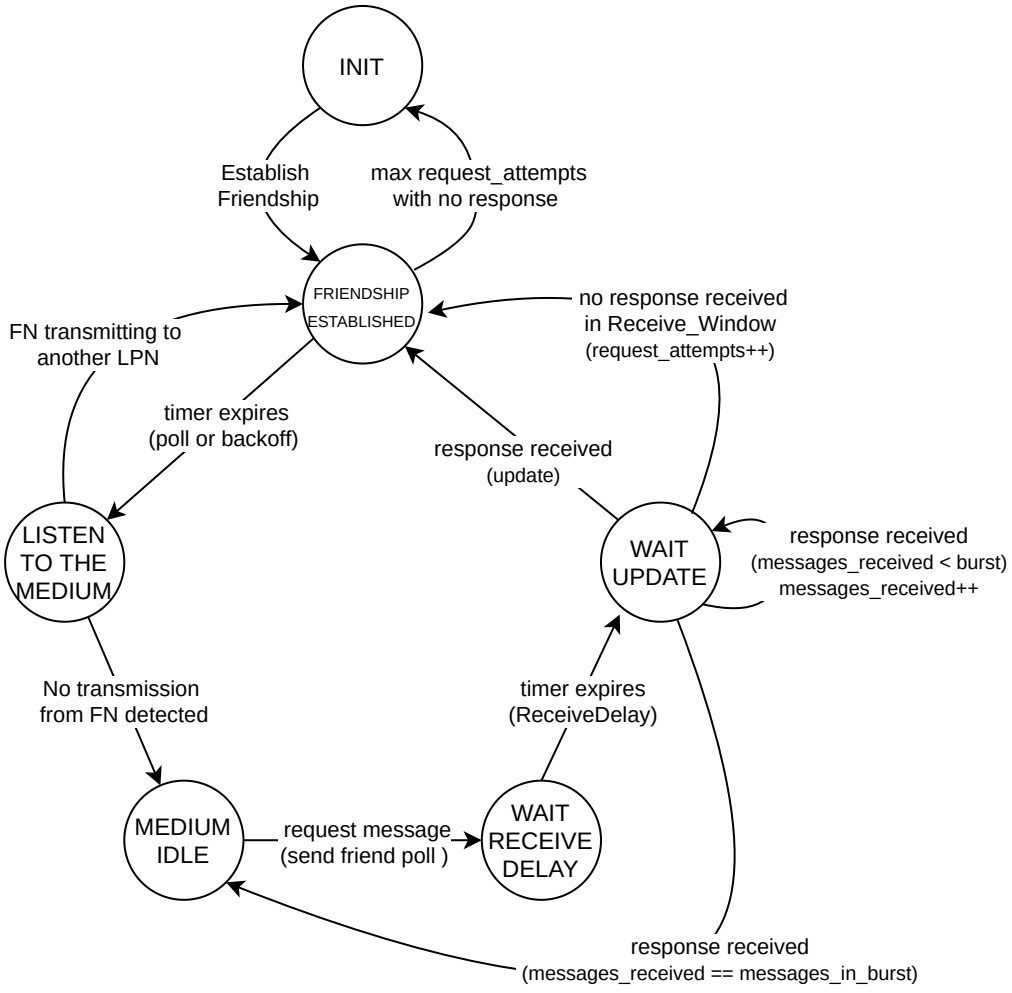


Figure 6.2: Bluetooth mesh BTLBT friendship state machine.

As in the Bluetooth mesh specification, poll packets sent by LPNs requesting messages from the FN act as ACKs for previously received transmissions. In our proposal, the entire burst is acknowledged, needing to be repeated if any packet is lost. This fulfils the requirement of LPN receiving all messages in order, as stated in the Bluetooth mesh specification.

6.4 Experimental Results

This section describe the experiments carried out to evaluate the implementation of the proposal detailed in the previous one (Section 6.3), as well as the results obtained. The proposal was implemented for the Zephyr RTOS (see Section A.2.6), and the experiments were conducted using PCA10056 Development Kit (DK) boards with nRF52840 chipset (see Section A.1.5). These experiments are divided into: (1) performance of Bluetooth mesh standard friendship; (2) performance of Bluetooth mesh friendship using burst transmissions; (3) performance of Bluetooth mesh friendship using our BTLBT proposal. Finally, the results obtained by the Bluetooth mesh standard friendship mechanism and those obtained by our proposed BTLBT for Bluetooth mesh friendship are compared.

The operation of the LPN is based on very low consumption. As explained in Section 2.3.1.1, LPNs spend most of their lifetime in sleep mode, being most of their consumption caused by sent messages (including requests) and especially by received messages, due to the time of the ReceiveWindow. For this reason, the way in which LPNs request and receive the stored messages from the FN requires improvement. However, this improvement would be worthless if messages were lost, since, in this case, messages need to be resent, invalidating the results achieved. Therefore, a reduction in consumption is as important as improving (or at least maintaining) the performance of the current Bluetooth mesh friendship.

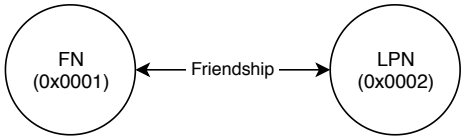
This section describes the experiments carried out regarding the performance obtained by three different approaches, as well as the results obtained. These approaches are:

1. Bluetooth mesh standard friendship mechanism.

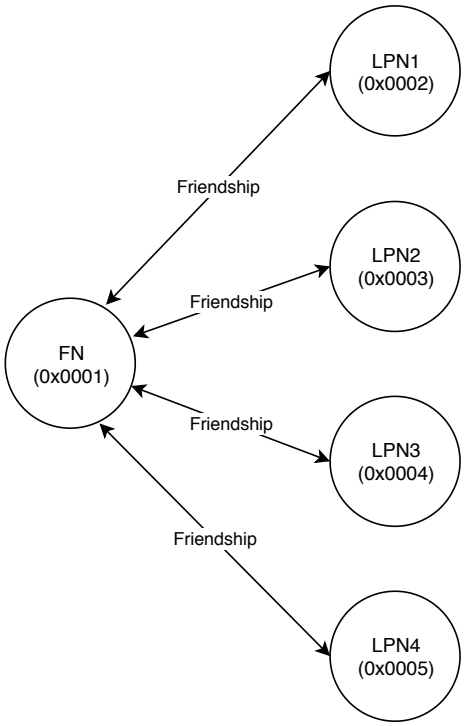
2. Bluetooth mesh friendship using burst transmissions.

3. Bluetooth mesh friendship using BTLBT.

Two different network topologies were used for these experiments: 1 FN in a friendship relationship with 1 LPN (Figure 6.3a) and 1 FN in a friendship relationship with 4 LPNs (Figure 6.3b). At the time of starting all the experiments, the FN had already established the friendship with all the LPNs, and had 160 messages in each of its Friend Queues. Once the LPNs wake up to the first request, they continue to request messages until they receive the last one (update packet).



(a) Network friendship topology with 1 LPN.



(b) Network friendship topology with 4 LPNs.

Figure 6.3: Network friendship topologies used in our experiments.

6.4.1 Performance of the Bluetooth mesh Standard Friendship Mechanism

This section begins with the evaluation of the current performance of the Bluetooth mesh standard friendship. Two experiments were conducted, where 1 FN was in charge of 1 and 4 LPNs (see Figure 6.3a and Figure 6.3b respectively). For the experiment with 4 LPNs, LPNs woke up simultaneously to request messages. In both experiments, the FN stored 160 data messages for each LPN. The LPNs were placed one metre away from the FN, and were configured with a ReceiveDelay of 100 ms and a ReceiveWindow of 255 ms, which terminated when the message was received. When an LPN woke up, it consecutively requested messages (using poll packets) until it received the update message indicating no more stored messages were available. All nodes were configured to send each message only once per advertising channel. The advertising interval was 20 ms, although the Zephyr implementation used requires 60 ms between the sending of one packet and the following one. The scan interval was 30 ms, with a scan window of 30 ms (100%). Table 6.4 shows the results of these experiments. The results obtained from the experiment carried out with 4 LPNs are shown as an average per LPN.

Table 6.4: Results for 1 and 4 LPNs that requested messages simultaneously to the FN using the Bluetooth mesh standard friendship mechanism.

Simultaneous devices	1 LPN	4 LPNs (Avg. per LPN)
Polls sent by each LPN	165	164.25
Messages received (+ update)	160 (+1)	160 (+1)
Total time (s)	30.725	39.797
Average time per message (s)	0.192	0.249
Average Receive window (ms)	92.27	147.06

The results in Table 6.4 show that LPNs received the 160 data messages stored in the FN in all cases, as well as the final update packet. However, additional poll packets had to be sent due to the packet losses between the LPNs and the FN. These losses did not occur from the FN to the LPNs, where all messages were received at the first attempt. Moreover, the results show how increasing the number of LPNs causes a degradation in network performance, since the FN needed to share its capacity among all of them. When 4 LPNs performed simultaneous requests, the total time to complete the transaction (and therefore the time per individual message) was increased by 29.53% compared to the 1 LPN scenario. In addition, not only was the time needed per packet increased, but also the power consumption, since the ReceiveWindow used in the scenario with 4 LPNs was, on average, 147.06 ms, 59.38% higher than the 1 LPN scenario.

6.4.2 Performance of Bluetooth mesh Friendship using Burst Transmissions

This subsection presents the results of the experiments conducted using our proposal for burst transmissions defined in Section 6.3. These experiments were carried out by varying the number of LPNs (1 LPN and 4 LPNs with 1 FN) in both cases, as well as the number of messages per burst (with bursts of 4, 8, 16 and 32 messages), to determine the influence of both parameters on performance.

In order to compare the results obtained in these experiments with those obtained using the standard friendship mechanism, the same parameters were maintained. For this reason, 160 messages per LPN were stored in the Friend Queues of the FN. LPNs were configured with a ReceiveDelay of 100 ms, which in this case was used only at the beginning of each burst, enabling the FN to prepare the necessary messages. The ReceiveWindow was 255 ms, being reset after the reception of each message, regardless of the time remaining for completion. This situation was repeated until one of the following three situations occurred, as explained in the definition of our proposal (Section 6.3): (1) completion of the burst; (2) reception of an update packet; (3) completion of the current ReceiveWindow when no message was received.

Regarding the parameters of the lower layers (Physical and Link layers), which are part of the BLE (see Section 2.2.3.1 and Section 2.2.3.2), we used the same parameters as in the previous experiment. Thus, all the nodes sent the packets through the three advertising channels, using an advertising interval of 20 ms (although the time between different messages increased to 60 ms due to the implementation of Zephyr). The scanning interval was 30 ms, with a scanning window of 30 ms (100%). To minimise the impact of the 60 ms between messages due to the implementation of Zephyr, the LPNs were switched to sleep mode for 25 ms after a message was received. This allowed the LPNs to save power and switch back to scanning mode before the message was sent. Without this limitation, all messages in the burst could be received in a shorter time, and the LPNs would not need to stop scanning until the end of the burst.

The results of these experiments are presented in Table 6.5 and Table 6.6. Table 6.5 shows the results obtained for 1 LPN, while Table 6.6 shows the results obtained for 4 LPNs. This grouping according to the number of LPNs in the tests allows us to verify whether the performance is affected by the coexistence of several LPNs requesting simultaneously.

Table 6.5: Results for 1 LPN that send polls to the FN using different burst sizes.

Simultaneous devices	1 LPN			
Messages per burst	4	8	16	32
Polls sent by each LPN	41	21	11	6
Messages received (+ update)	160 (+1)	160 (+1)	160 (+1)	160 (+1)
Total time (s)	16.457	13.456	11.775	10.870
Average time per message (s)	0.103	0.084	0.074	0.068
Average Receive window (ms)	48.06	43.86	40.37	39.41

Table 6.6: Results for 4 LPNs that send polls simultaneously to the FN using different burst sizes.

Simultaneous devices	4 LPNs (Avg. per LPN)			
Messages per burst	4	8	16	32
Polls sent by each LPN	41	21	11	6
Messages received (+ update)	160 (+1)	160 (+1)	160 (+1)	160 (+1)
Total time (s)	39.521	39.524	39.639	39.560
Average time per message (s)	0.247	0.247	0.248	0.247
Average Receive window (ms)	194.18	205.98	213.08	215.75

The results reflect the variation in the number of polls sent by each LPN. This number is, in all cases, lower than the poll packets required by the standard friendship mechanism, and decreases when the size of the burst is increased, since a single request causes the sending of the entire burst. In contrast to the previous experiment, all polls were successfully received at the FN, and no resending was necessary. Similarly to the results obtained with the standard friendship mechanism, all messages (160 data messages and one additional

update packet) were successfully received with no packet loss, regardless of the number of LPNs or the burst size used.

The results show how the time required to complete the reception of 160 messages (as well as the average time per message) in the experiment with 1 LPN is reduced when using burst transmissions, compared to the Bluetooth mesh standard friendship mechanism. The larger the burst size, the greater is the reduction achieved. The time decreased from 30.725 seconds when using the Bluetooth mesh standard friendship to 16.457 seconds when using 4-message burst transmissions (46.44% reduction); to 13.456 seconds when using 8-message burst transmissions (56.20% reduction); to 11.775 seconds when using 16-message burst transmissions (61.67% reduction); and to 10.870 seconds when using 32-message burst transmissions (64.62%). As explained above, the time reduction achieved by increasing the number of packets in the burst is the result of two factors: (1) there is a lower number of ReceiveDelays, only one per burst; and (2) the largest ReceiveWindow is, in most cases, that corresponding to the reception of the first message of the burst, due to the preparation required by the FN.

In the experiments conducted with 1 LPN, we can notice not only a reduction in the time required to complete the transmission of all messages but also a reduction in the average time of the ReceiveWindows used. The reduction achieved falls from 92.27 ms with the standard method to 48.06 ms when using 4-message burst transmissions (47.92% reduction); to 43.86 ms when using 8-message burst transmissions (52.47% reduction); to 40.37 ms when using 16-message burst transmissions (56.25% reduction); and to 39.41 ms when using 32-message burst transmissions (57.29%). This reduction is the most important improvement achieved, since it enables LPNs to reduce scanning time, and thus, the average consumption. To this consumption reduction, we should add that obtained by reducing the number of requests required.

This situation contrasts with the results obtained with 4 LPNs requesting to the FN simultaneously. In this case too, there was no packet loss (neither the poll packets sent by the LPNs nor the data messages sent by the FN). However, the time required to complete the transmission of all the messages remained practically the same: if the time obtained using the standard friendship mechanism was 39.797 seconds, the time obtained for the burst transmissions ranged from 39.521 to 39.639 seconds. There was an increase in the average time of the ReceiveWindows used, which increased from 147.06 ms to 194.18 - 215.75 ms, depending on the size of the burst used.

Compared to the scenario with 1 LPN using burst transmissions, the results show an exceptional increase, which rises as the size of the bursts grows. The total time required to complete the reception of 160 messages with 4 LPNs simultaneously, compared to the 1 LPN scenario, increased by 140.14% for 4-message bursts, 193.73% for 8-message bursts, 236.63% for 16-message bursts and 263.94% for 32-message bursts. This trend remained in the average ReceiveWindow when we compare the scenarios with 1 and 4 LPNs: the ReceiveWindows when 4 LPNs simultaneously requested messages increased, on average,

by 304.06% for 4-message bursts, 369.65% for 8-message bursts, 427.80% for 16-message bursts, and 447.51% for 32-message bursts.

This situation is caused by the congestion of the FN: after sending a message, the receiving LPN has the highest priority to receive also the next message, unless the ReceiveWindow of another LPN is close to ending. This increases the delay in receiving messages in the rest of the LPNs.

This problem, generated by the coexistence of several simultaneous LPNs, encouraged us to consider another possible improvement: the use of the BTLBT proposal, defined in Section 6.3, and whose results are presented in the following subsection.

6.4.3 Performance of Bluetooth mesh Friendship with BTLBT

The results of the experiments conducted for the burst transmissions for 1 and 4 LPNs, with different burst sizes, were previously presented. As explained, this type of transmission works especially well with 1 LPN, significantly reducing the time and consumption of the friendship mechanism proposed by the Bluetooth mesh specification. However, when the number of simultaneous LPNs increases, the performance of our proposal decreases, obtaining a similar total time with longer ReceiveWindows. For this reason, the LBT technique was implemented, as mentioned in Section 6.3.

As explained, using the LBT technique, LPNs listen to the medium before sending their poll packets. If the FN is sending a burst of messages, the LPNs wait in sleep mode before listening to the medium again. To avoid, as far as possible, two LPNs sending their requests simultaneously, these waiting times are randomly generated. In this experiment, the LPNs listened to the medium for a randomly generated time between 1 and 2 times the ReceiveWindow (255-510 ms). If a burst from the FN was detected during this time, the LPN went into sleep mode for a time between 5 and 10 seconds, which was also random. If the scan time expired without having detected a burst in progress, the LPN sent its poll packet.

This experiment was conducted using 4 LPNs waking up simultaneously. The selected burst size was 16 messages, an intermediate size from those previously presented, which gave us a great reduction in the ReceiveWindow with a lower saturation of the FN than the larger bursts. The rest of the configuration parameters used in this experiment were the same as those used in the previous experiments. The FN generated and stored 160 messages per LPN in the Friend Queues. The ReceiveDelay was set to 100 ms, while the ReceiveWindow was set to 255 ms, being reset when a new burst message was received. This reset occurred until: (1) the burst ended; (2) an update packet was received; (3) the current ReceiveWindow expired before any messages were received.

Regarding the BLE lower layers, all messages were sent once per advertising channel. The advertising interval used was 20 ms. Again, there was a 60 ms delay between the sending of different messages, which was minimised by the LPN switching to sleep mode

for 25 ms after receiving a message. The scanning interval used was 30 ms, with a scanning window of 30 ms (100%).

The results of this experiment are shown in Table 6.7. These results show a reduction in the number of requests in our BTLBT friendship proposal compared to the standard friendship mechanism. In this case, 16-message burst transmissions were selected, and therefore only 11 requests were necessary to receive the 160 messages (and the update packet). All poll packets were received correctly by the FN, so resending them was unnecessary. The results were similar for the data messages: all messages were received on the first attempt.

Table 6.7: Results for 4 LPNs requesting to the FN simultaneously, using our proposal BTLBT.

Device	LPN1	LPN2	LPN3	LPN4	Average
Polls sent by each LPN	11	11	11	11	11
Messages received (+ update)	160 (+1)	160 (+1)	160 (+1)	160 (+1)	160(+1)
Total time (s)	11.009	11.028	11.031	10.995	11.016
Average time per message (s)	0.069	0.069	0.069	0.069	0.069
Average Receive window (ms)	39.20	39.24	39.35	39.14	39.16
Time listening before transmit (s)	0.334	1.163	0.673	0.626	0.699
Avg. time listening per msg. (ms)	2.09	7.27	4.21	3.91	4.37

One of the most significant aspects is the total time required to receive all the messages. This time was reduced to 11.016 seconds on average when using the BTLBT friendship proposal. This means a reduction of 72.32% compared to the standard friendship mechanism and 72.21% compared to the friendship with burst transmissions proposal with 4 LPNs. This reduction was translated into the required average time per message.

However, the most important achievement lies in the reduction of the ReceiveWindow, since it is the effective scanning time of the devices and, therefore, one of the periods of greatest energy consumption. The average Receive Windows per message was 39.16 ms. This represents a reduction of 73.37% compared to using the standard friendship mechanism and 81.62% compared to friendship with burst transmissions when 4 LPNs were requesting to the same FN.

The only drawback of this proposal is that it requires a previous scanning time to check whether the FN was available before requesting messages. This time varied from one LPN to another, due to its randomness as well as the different number of attempts required to start receiving messages. The times obtained were 0.334 seconds for LPN1, 1.163 seconds for LPN2, 0.673 seconds for LPN3 and 0.699 seconds for LPN4. Thus, the average time was 4.37 ms per message received, which is an affordable cost considering the reduction achieved in the Receive Window.

6.4.4 Comparison

Following the description of our results, this section concludes with a comparison between the standard friendship mechanism and our BTLBT friendship proposal, in order to provide a summary of the improvements obtained by our proposal. Table 6.8 presents the results obtained, on average, for 1 FN and 4 LPNs simultaneously requesting, using the standard friendship mechanism (second column) and using our BTLBT friendship proposal (third column).

Table 6.8: Average results for 4 LPNs requesting to the FN simultaneously using the Bluetooth mesh standard friendship mechanism and our proposal BTLBT.

Friendship Proposal	Standard	BTLBT
Simultaneous devices	4 LPNs	4 LPNs
Polls sent by each LPN	164.25	11
Messages received (+ update)	160 (+1)	160 (+1)
Total time (s)	39.797	11.016
Average time per message (s)	0.249	0.069
Average Receive window (ms)	147.06	39.16
Time listen to the medium (s)	0	0.699

The results obtained reveal a drastic reduction in the time required to complete each transaction, from 39.797 seconds to 11.016 seconds, which represents a reduction of 72.32%. This time reduction stems from the following improvements:

1. Sending a lower number of poll packs, since sending a poll pack for each message stored by the FN is no longer necessary. The number of poll packets sent decreased from an average of 164.25 to only 11 (93.30% less).

2. This lower number of poll packets is also related to a lower number of ReceiveDelay, which are only used once per burst.

3. Reduction of the ReceiveWindow required per message. This falls from 147.06 ms to only 39.16 ms (73.37% less). Moreover, reducing this parameter does not only save time, but since the LPN is in scanning mode during this parameter, this reduction generates a lower power consumption.

These improvements were obtained by combining burst transmissions and the LBT technique. Burst transmissions enable the time needed when 1 LPN requests messages from the FN to be reduced. Meanwhile, the LBT technique enables the FN to focus on 1 LPN, despite other LPNs. This increases the time that LPN spend in sleep mode, which is essential for their own features.

Finally, the only drawback of our proposal is that it requires the use of a listening time. For this case (with a transmission of 160 messages for each LPN, with 4 LPNs competing for the FN simultaneously), this listening time was 0.699 seconds per LPN on average. This

accounts for an additional time of 4.37 ms per message received, which is worthwhile taking into account the reduction achieved in the ReceiveWindow.

6.5 Estimated Consumption

This section presents the study conducted on the estimated consumption for LPNs, using both the standard friendship mechanism and our BTLBT friendship proposal. For this study, we drew on [73], where the authors model the current consumption of a battery-operated Bluetooth mesh LPN. This consumption is shown in Table 6.9, according to the main states in which an LPN can be found. These states are detailed below.

1. Sleeping, where the LPN keeps active the parts which are indispensable to its proper functioning, such as the clock.
2. Sending, which may correspond either to a poll packet to request messages from the FN, or to a data message sent to another node. This includes the time and consumption required to perform all the tasks to send a message through the three advertising channels: node wake up; 1^{st} message transmission; 1^{st} channel change; 2^{nd} message transmission; 2^{nd} channel change; 3^{rd} message transmission; radio off; post-processing; and cool down.
3. Receiving a message from the FN, for which it is necessary to put the LPN in scan mode. This includes: wake up pre-scan, scan, radio off, post-processing and cool down. In this case, neither the total time nor the average consumption could be obtained, since these depend on the length of the ReceiveWindow, which varies depending on the configuration. Therefore, the consumption related to ReceiveWindow in Table 6.9 refers to the current consumption, instead of the average consumption.

Table 6.9: Estimated consumption for each state of the LPN, drawing on [73].

State	[T] Time (ms)	[I] Average Consumption (mA)
Sleeping	-	0.015
Sending	18.63	1.075
Receiving	ReceiveWindow	15.9[1]
	28.19 (other actions)	0.157

Data messages are sent from the LPNs to other nodes in the same way in our proposal as in the standard friendship mechanism. Therefore, these sendings were not taken into account in this estimation, since their consumption is the same.

The consumption of LPNs is mainly determined by the PollTimeout. As explained in Section 2.3.1.1, the PollTimeout is the maximum time between the sending of two poll

[1]The value is shown in current consumption, as the average consumption depends on the time of the ReceiveWindow, which is variable.

packets from the LPN. Each poll packet received by the FN triggers it to send a message stored for the corresponding LPN (or of an update packet in case there are no more messages in the Friend Queue). The Bluetooth mesh specification defines this time as a maximum of 96 hours, with friendship termination if exceeded. In other words, the LPN is required to send at least one poll packet and receive a message from the FN every 96 hours.

However, after sending the first poll packet, the Bluetooth mesh specification recommends making successive requests until the reception of the update packet. Thus, if more than one message is stored in the Friend Queue, they will be requested one after another, so the average PollTimeout will decrease with each additional packet stored. In any event, the time in sleep mode (T_{sleep}), can be calculated as shown in Equation 6.1:

$$T_{sleep} = PollTimeout - T_{act} \tag{6.1}$$

where T_{sleep} is considered all PollTimeout, except T_{act}, which includes all other actions performed, such as sending or receiving messages. Although in [73] the sending of data messages from LPNs is also included in this time, in our case it was not taken into account since it is done in the same way in both our proposal and the standard, as already stated. Thus, T_{act} can be defined as shown in Equation 6.2:

$$T_{act} = (Requests \cdot T_{sending}) + (Messages \cdot T_{receiving}) \tag{6.2}$$

where Requests is the number of poll packets sent to the FN and Messages is the number of data messages stored in the Friend Queue, which are sent to the LPN after the corresponding request (and the additional update packet). In the standard friendship mechanism, the number of requests matches the number of messages. However, in our proposal, a request initiates a message burst, reducing the number of Requests as shown in Equation 6.3:

$$Request = \frac{Messages}{Burst\ size} \tag{6.3}$$

Finally, $I_{average}$ can be obtained, for the lifetime of an LPN, including all its states, as shown in Equation 6.4:

$$I_{average} = \frac{1}{PollTimeout} \left(I_{avg\_sleeping} + I_{avg\_sending} + I_{avg\_receiving}\right) \tag{6.4}$$

where $I_{avg\_sleeping}$, $I_{avg\_sending}$ and $I_{avg\_receiving}$, are calculated as shown in Equation 6.5, Equation 6.6 and Equation 6.7, respectively.

$$I_{avg\_sleeping} = T_{sleeping} \cdot I_{sleeping} \tag{6.5}$$

$$I_{avg\_sending} = Requests \cdot T_{sending} \cdot I_{sending} \tag{6.6}$$

$$I_{avg\_receiving} = Messages \cdot T_{receiving} \cdot I_{receiving} \tag{6.7}$$

Using the results obtained previously for the different approaches (see Table 6.4 for the standard friendship mechanism, Table 6.5 for burst transmissions with 1 LPN and Table 6.7 for 4 LPNs using BTLBT), consumption was obtained for some of the different configurations based on Equation 6.4. Figure 6.4 shows the current consumption per LPN for four different configurations: Bluetooth mesh standard friendship mechanism for 1 and 4 LPNs (BM 1LPN and BM 4LPN, respectively) and the BTLBT proposal using a 16-packet burst for 1 and 4 LPNs (BTLBT 1LPN and BTLBT 4LPN). The results show the consumption for each LPN in receiving from 0 to 160 data messages (plus the corresponding update packet) in a maximum PollTimeout, which is 96 hours, as mentioned above.

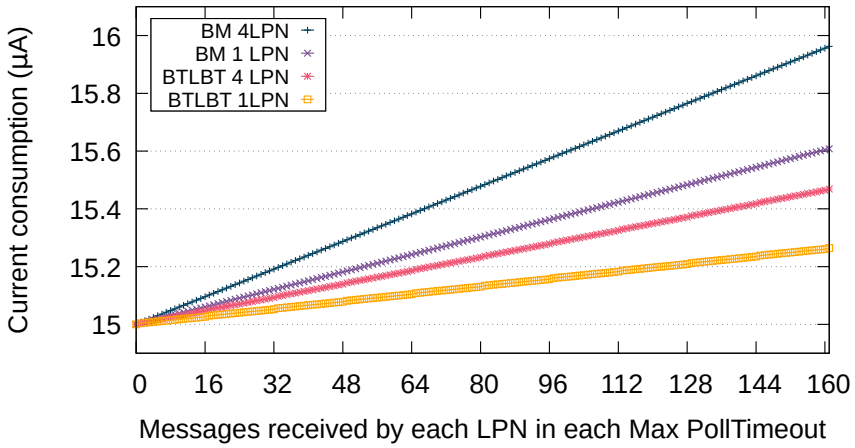


Figure 6.4: Consumption comparison between different approaches and number of simultaneous LPNs.

Figure 6.4 shows the improvement achieved by our proposal, regardless of the number of LPNs, even reducing the consumption of the standard proposal with a lower number of LPNs. This difference is increased significantly when the number of messages for the LPN grows. Thus, our proposal of BTLBT achieves, for the reception of 160 messages in the maximum timeout, a reduction of 2.2% in consumption for 1 LPN, reaching 3.1% for 4 simultaneous LPNs.

Using the consumption data obtained, it is possible to theoretically calculate the lifetime of a LPN using the different configurations, according to Equation 6.8:

$$T_{lifetime} = \frac{C_{battery}}{I_{average}} \tag{6.8}$$

To calculate the life time, in addition to the consumption data obtained, a coin cell CR2032 [74], with a capacity of 232 mAh, was used. The configurations of the LPNs used were the same as in the previous case: Bluetooth mesh standard friendship mechanism for 1 and 4 LPNs (BM 1LPN and BM 4LPN, respectively) and the BTLBT technique using a 16-packet burst for 1 and 4 LPNs (BTLBT 1LPN and BTLBT 4LPN). Figure 6.5 presents the results obtained for a number of messages between 0 and 1000 for each maximum PollTimeout (96 hours).

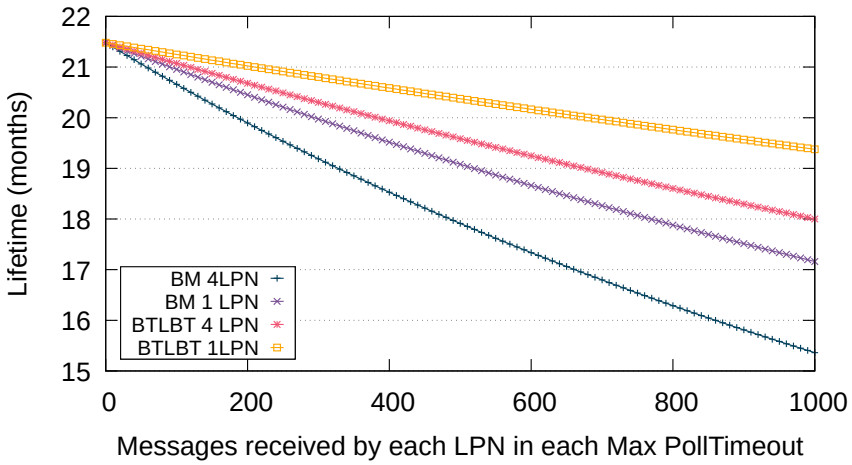


Figure 6.5: Lifetime comparison between different approaches and number of simultaneous LPNs.

In this case, our BTLBT proposal again improves on the standard friendship mechanism. The most extreme case in the figure is the reception of 1000 messages over the maximum PollTimeout. As this maximum is defined in 96 hours, it would imply the reception of a message every almost 6 minutes on average. In this most extreme case, our BTLBT proposal achieves a lifetime increase of more than two months for 1 LPN (a 12.9% improvement), while the lifetime increment for 4 LPNs is more than 2.5 months (representing a 17.2% improvement).

6.6 Conclusions

This chapter has focused on the friendship mechanism, the method proposed by Bluetooth mesh for low-power devices that require a battery to operate. Given the operating mode of the Bluetooth mesh network, the nodes of the network are required to spend most of their lifetime in the scanning mode, in order to receive and relay the received messages. In this context, Bluetooth mesh provides the friendship relationship, where FNs can store messages destined to the LPNs. The transmission of the messages from the FN to the LPN is on request.

This chapter includes the study of different proposals to improve this standard friendship mechanism, which were finally discarded for different reasons. These proposals were:

improving the Bearer Layer, improving time synchronization and improving advertising channel utilisation.

Moreover, the most important contribution of this chapter lies in a new proposal for the friendship mechanism: Burst Transmissions and Listen Before Transmit. Although burst transmissions achieved a better performance when the FN was responsible for a single LPN, this decreased when the number of LPNs was increased. The inclusion of the LBT technique solved this problem, enabling our proposal to reduce the power consumption regardless of the number of LPNs. Although this technique does not guarantee that two LPNs will perform simultaneous requests, the selected listening times enabled LPNs to avoid this situation in all the experiments.

The experiments carried out showed that our BTLBT proposal maintains the 100% PRR obtained by the standard proposal, reducing not only the time needed for the transmissions, but also the number of requests and the scanning time of the ReceiveWindow (this reduction reached 73.15% per received message). This was reflected in a decrease in consumption per LPN, which became more remarkable when the number of messages destined to LPNs was increased. This consumption reduction reached 3.1% with 160 messages per LPN in a 96-hour period (which is the maximum defined by the specification for the PollTimeout). Estimations for a greater number of messages revealed a 17.2% increase in the lifetime of the LPN for the reception of 1000 messages in this maximum PollTimeout when using our proposal, compared to the calculations obtained for the standard friendship mechanism.

CHAPTER 7

Conclusions and Future Work

This chapter concludes this thesis by summarising the conclusions drawn from the work conducted. Subsequently, the possible tasks that could be developed in future work are presented. Finally, this chapter presents the author's biography, listing the projects in which he has participated and the publications produced during the completion of this Doctoral Thesis.

7.1 Conclusions

I would like to start this section with the conclusions on a personal level. The completion of this Doctoral Thesis has permitted me to grow and mature on a personal level. Since I started my Doctoral Thesis, I have had the opportunity and the luck to work with excellent people, from whom I have been able to learn in all aspects. It has also served me enormously to broaden my mind, and prepare me for future challenges, whatever they may be.

As discussed in Chapter 1, the rise of IoT, and especially one of its areas, manufacturing, and the subsequent emergence of the Industry 4.0 trend, has caused the requirements for wireless sensor networks to become more challenging than ever. In industrial environments where the user is a critical part, communications networks must be completely reliable (zero fails), provide total coverage and be sustainable. Therefore, we started working with BLE technology, a promising technology launched relatively recently, with low consumption and which allowed users to become part of networks (where the human-in-the-loop model stands out) in a simple and intuitive way through mobile devices. These mobile devices include not only smartphones and tablets, but also wearables.

In light of this, one of the main conclusions of this Doctoral Thesis is that the development and optimisation of low-consumption wireless communication networks has become a necessity in the new applications of IoT and Industry 4.0. This conclusion led to the definition of the main objective of this work, which corresponds to the design of a hetero-

geneous BLE network that allows the inclusion of any device, while sustainably meeting the requirements of the new applications.

To achieve this objective, the first goal defined for this dissertation was to review the state of the art of wireless networks in general, emphasising BLE technology and its network topologies, as well as new applications of IoT and Industry 4.0 and their requirements. This study is included in Chapter 2. Once familiarised with the standard and the requirements to be fulfilled, the next natural step was to perform a preliminary assessment of the technology at that time. This evaluation is detailed in Chapter 3.

In view of the conclusions obtained from the study and evaluation of the standard, our third goal emerged, the design and implementation of a network topology different from those defined by the specification that would allow the already mentioned requirements to be met. In addition, our proposal had to allow the inclusion of devices other than traditional sensor nodes, such as smartphones and wearables, as established in our fourth goal. Our proposal for a BLE mesh network, which we called Collaborative Mesh, is detailed in Chapter 4. This chapter also contains the evaluation of our proposal, where different configurations were assessed, obtaining a PRR of up to 100% for static sensor nodes and between 88% and 96% for mobile devices, as well as total coverage.

At that point in this Doctoral Thesis the Bluetooth mesh specification was released, which provided BLE with the mesh topology. Drawing on this, our fifth goal was defined, the in-depth study of the Bluetooth mesh specification, as well as the provisioning procedure that allows a new device to become part of the network. Both are detailed in Section 2.3.

With the release of the Bluetooth mesh suite of specifications, continuing to improve our proposal became pointless, as our aim is to be compliant with the BLE standard. However, despite the Bluetooth SIG having claimed that this specification is compatible with any BLE version, current devices and implementations use the latest BLE versions (5.0 and later). Therefore, our next goal was to develop an implementation of the Bluetooth mesh specification that could be used in most devices, including our BLE 4.0 devices. This implementation is described in detail in Chapter 5. This chapter also includes the evaluation of our implementation, which obtained a PRR of up to 100% with an end-to-end delay of about 23.2 ms in a two-hop network.

Our next goal was to optimise the provisioning procedure, which was designed without taking into account the devices of the earlier BLE versions. Therefore we proposed our Lightweight Provisioning procedure, which reduces the number of messages exchanged between the provisioner device and the unprovisioned device, while maintaining its robustness and security. Our proposal reduced the time needed to complete the procedure by up to 38.31%. This proposal and its evaluation are detailed in Section 5.3 and Section 5.4.

Finally, focusing on the Bluetooth mesh specification and leaving the device versions aside, our last goal was to optimise the friendship mechanism, provided by Bluetooth mesh to reduce the consumption of battery-powered devices. To this end, we studied different

alternatives, finally implementing our proposal of Burst Transmissions and Listen Before Transmit. This proposal is presented in detail in Chapter 6. The evaluation carried out shows that our proposal increased the lifetime of the devices up to 17.2%, which grew according to the number of packets sent, while maintaining a 100% PRR.

7.2 Future Work

The Bluetooth mesh suite of specifications was designed keeping in mind the new scenarios of IoT and Industry 4.0. However, it still requires some optimisations, especially in the operating mode of network devices. These devices scan the network constantly to avoid packet loss, with the consequent energy consumption. One of the future works related to this technology is to reduce the scanning time of network devices. To this end, it is proposed to study synchronisation techniques that allow devices to switch to a sleep mode at certain times. With the same objective, it is proposed to study Software Defined Networking (SDN) technology to dynamically configure networks, improving their efficiency.

This Doctoral Thesis focused particularly on BLE technology and its Bluetooth mesh topology. However, in real industrial environments, the coexistence of different devices and standards is indispensable to fully satisfy all requirements, including the coverage of extremely wide areas or real-time transmissions. Therefore, one of the main future tasks is to increase the number of devices in our network, as well as the protocols used. The work done in the GreenISF scenario presented in this Doctoral Thesis already includes the coexistence of BLE technology with LoRaWAN, but other standards would bring different benefits.

Another component closely related to this work is the Cloud technology. Our network deployment included a BLE server that received and sent information, as well as storing it and taking small decisions, such as sending a warning in the event of detecting any anomaly. This server was implemented using node.js, so deploying it in a cloud environment would not require much effort. However, it is also necessary to add the intelligence required in a real network.

When the network deployed in the laboratory has different standards and devices, a deployment in a real industrial environment would be of great interest. Since communication networks are a fundamental part of the digitalisation of industry, their performance must be evaluated first in this type of environment.

7.3 Author's Biography

Diego Hortelano Haro received his B.Sc. and M.Sc. degrees in computer science and engineering from the University of Castilla-La Mancha, Spain, in 2013 and 2015 respectively, where he is currently pursuing the Ph.D. degree in advanced computing technologies. In February 2015, he joined the Laboratory of High-Performance Networks and Architectures,

in the Albacete Research Institute of Informatics, as a Research Assistant. In October 2015, he obtained a pre-doctoral grant from the UCLM co-financed by European funds for his Doctoral Thesis. During the realisation of his Doctoral Thesis, he has participated in different research projects and from his research different publications have been derived. His main research fields are Internet of Things, wireless sensor networks, and wireless communication protocols, especially focused on Bluetooth Low Energy technology.

7.3.1 Projects

During the realisation of this Doctoral Thesis, the author has participated in different research projects:

- Project title: Tecnologías y Aplicaciones innovadoras para centros de datos y computadoras de altas prestaciones (RTI2018-098156-B-C52). Financing Entity: MINECO. From January 2019 to December 2021. Main Researchers: José Duato Marín (UPV) and F. José Quiles Flor (UCLM).
- Project tile: Code is Loading (2018-1-TR-01-KA201-058963). Financing Entity: European Commission. From November 2018 to October 2020. Main Researcher: Teresa Olivares Montes.
- Project title: Wearable Methodology, a new Methodology based on the Use of Innovative Technologies for Education (2016-1-ES-01-KA201-025397). Financing Entity: European Commission. From October 2016 to September 2018. Main Researcher: Teresa Olivares Montes.
- Project title: Técnicas para la mejora de las prestaciones, consumo de energía y gestión de recursos de los servidores. Optimización de la codificación y distribución de los contenidos Multimedia (TIN2015-76972-C5-2-R). Financing Entity: MINECO / FEDER. From January 2016 to December 2018. Main Researchers: José Duato Marín (UPV) and Pedro Cuenca Castillo (UCLM).
- Project title: Ecosense, Sistema Inteligente para la gestión de la energía en edificios basado en redes inalámbricas de sensores y actuadores" (POII-2014-010-P). Financing Entity: Junta de Comunidades de Castilla-La Mancha. From September 2014 to September 2017. Main Researcher: Teresa Olivares Montes.

7.3.2 List of Publications

During the completion of this Doctoral Thesis, the following publications were presented

7.3.2.1 Publications generated from this dissertation

The studies and results included in this dissertation were presented in the following publications:

Publications in international journals

- D. Hortelano, T. Olivares and M.C. Ruiz, "Minimising the Energy Consumption of the Friendship Mechanism in Bluetooth mesh", *Computer Networks*, under review. Impact Factor (JCR 2019): 3.111. Position 14/53 (Q2).
- D. Hortelano, T. Olivares and M.C. Ruiz, "Providing interoperability in Bluetooth mesh with an improved Provisioning protocol", *Wireless Networks*, pp. 1-23, 2020. Impact Factor (JCR 2019): 2.659. Position 72/156 (Q2).
- C. Garrido-Hidalgo, D. Hortelano, L. Roda-Sánchez, T. Olivares, M.C. Ruiz and V. López, "IoT Heterogeneous Mesh network Deployment for Human-in-the-Loop Challenges towards a Social and Sustainable Industry 4.0", *IEEE Access*, vol 6, pp. 28417-28437, 2018. Impact Factor (JCR 2018): 4.098. Position 23/155 (Q1).
- D. Hortelano, T. Olivares, M.C. Ruiz, C. Garrido-Hidalgo and V. López, "From Sensor Networks to Internet of Things. Bluetooth Low Energy, a Standard for This Evolution" *Sensors*, vol 17(2), 372, pp. 1–31, 2017. Impact Factor (JCR 2017): 2.475. Position 16/61 (Q2).

Publications in international conferences

- D. Hortelano, "Collaborative Bluetooth Smart Mesh for improving Communications in Industry 4.0", in *Doctoral Colloquium of the 15th ACM Conference on Embedded Networked Sensor Systems, (SenSys 2017)*, Delft, The Netherlands, 2017.
- D. Hortelano, T. Olivares, V. Lopez and M.C. Ruiz "Improving the BLE Mesh Transmissions with user collaboration in Smart Spaces Management", in *Proceedings of the 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'2016)*, ISBN: 978-1-5090-0802-5, Vienna, Austria, 2016. CORE A*. Conference Rating Class 1.

Publications in national conferences

- D. Hortelano, L. Roda-Sánchez, C. Garrido-Hidalgo, T. Olivares and M.C. Ruiz, "Malla Bluetooth Low Energy para la nueva Industria 4.0", in *Actas Jornadas SARTECO 2017 (JS'17)*, ISBN: 978-84-697-4835-0, Málaga, España, 2017.
- D. Hortelano, T. Olivares, M.C. Ruiz and C. Garrido-Hidalgo, "Redes en malla BLE para comunicaciones inteligentes y colaborativas", in *Actas Jornadas SARTECO 2016 (JS'16)*, ISBN: 978-84-9012-626-4, Salamanca, España, 2016.
- D. Hortelano, T. Olivares, M.C. Ruiz and V. López-Camacho, "Estudio del estándar Bluetooth Low Energy para redes híbridas en IoT", in *Actas Jornadas SARTECO 2015 (JS'15)*, ISBN: 978-84-16017-52-2, Córdoba, España, 2015.

7.3.2.2 Additional publications elaborated during this dissertation

Additionally, during the realisation of this Doctoral Thesis, other publications were presented:

Publications in international journals

- L. Roda-Sánchez, C. Garrido-Hidalgo, D. Hortelano, T. Olivares and M.C. Ruiz, "OperaBLE: an IoT Wearable to Improve Smart Worker Care Services in Industry 4.0", *Journal of Sensors*, vol 2018, pp. 1–12, 2018. Impact Factor (JCR 2018): 2.024. Position 30/61 (Q2).

Publications in international conferences

- M. Carmen Ruiz, C. Garrido-Hidalgo, D. P. Gruska, T. Olivares, D. Hortelano and L. Roda-Sanchez, "Modeling and Evaluation of a Power-Aware Algorithm for IoT Bluetooth Low Energy Devices", in *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, Beijing, China, 2019.
- C. Garrido-Hidalgo, D. Hortelano, L. Roda-Sánchez, T. Olivares and M.C. Ruiz "Experimental Evaluation of Low-Power Communication Protocols for an Industrial IoT Testbed", in *Proceedings of the International Conference on Industrial Internet of Things and Smart Manufacturing, IoTsm 2018* ISBN: 978-1-912532-06-3. London, United Kingdom, 2018.
- C. Garrido-Hidalgo, T. Olivares, V. Lopez-Camacho, M.C. Ruiz, D. Hortelano and V. Brea, "SustainaBLE: a Power-Aware Algorithm for Greener Industrial IoT Networks", in *Proceedings of the 16th International Conference on Ad Hoc Networks and Wireless*, ISBN: 978-3-319-67909-9, Messina, Italy, 2017. CORE B.

Publications in national conferences

- T. Olivares, V. López Camacho, E. de La Guía, C. Garrido-Hidalgo, L. Roda-Sanchez and D. Hortelano "Estudio de la percepción y conocimientos en Lenguajes de Programación de alumnos y profesores en el marco del proyecto Erasmus+ Code is Loading", in *Actas de las XXVI Jornadas sobre Enseñanza Universitaria de la Informática*, vol 5, pp 345–348, 2020.
- C. Garrido-Hidalgo, D. Hortelano, L. Roda-Sánchez, T. Olivares and M.C. Ruiz, "Evaluando el Consumo de Redes BLE/LoRaWAN para IoT", in *Actas Jornadas SARTECO 2017 (JS'17)*, ISBN: 978-84-697-4835-0, Málaga, España, 2017.
- L. Roda-Sánchez, C. Garrido-Hidalgo, D. Hortelano, T. Olivares and M.C. Ruiz, "Pulsera Inteligente para la Caracterización de Movimientos Orientada a IoT", in *Actas Jornadas SARTECO 2017 (JS'17)*, ISBN: 978-84-697-4835-0, Málaga, España, 2017.

- C. Garrido-Hidalgo, T. Olivares, M.C. Ruiz and D. Hortelano, "Implementación Real de un Sistema de Gestión Eficiente de WSAN basado en una Arquitectura BLE para IoT", in *Actas Jornadas SARTECO 2016 (JS'16)*, ISBN: 978-84-9012-626-4 Salamanca, España, 2016.

APPENDIX A

Hardware and Software

This appendix presents the most important hardware platforms used in this thesis, as well as the software related to them.

A.1 Hardware Platforms

This section describes the hardware platforms used for the implementation and experiments carried out throughout this Doctoral Thesis.

A.1.1 Waspmote

Waspmote [62] from Libelium (see Figure A.1) is an open source platform, which offers high modularity with a multitude of radio technologies and sensor configurations. Waspmote is based on Arduino, probably the most popular open-source platform for teaching and creating cheap home projects. However, Waspmote was especially designed for creating wireless sensor networks and is intended to be deployed in a real scenario. Some real projects where Waspmote devices are used can be found in [75].

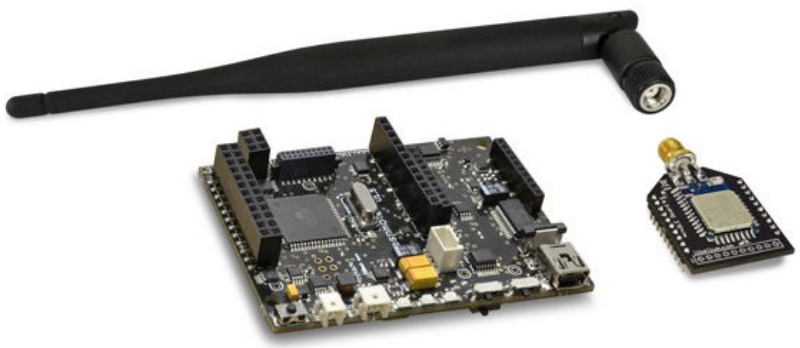


Figure A.1: Waspmote device from Libelium [62].

The Waspmote PRO (v1.2) devices are equipped with the ATmega1281 microcontroller (14.7456 MHz), SRAM memory (8 KB), EEPROM (4 KB) and FLASH (128 KB). The BLE radio module used by Waspmote devices is the BLE112 chipset from BlueGiga (now Silicon Laboratories [76]), and it has the following specifications: RX sensibility of -103 dBm; configurable TX Power interval between -23 dBm and +3 dBm; antenna of 5 dBi; and a maximum range of 100 m.

A.1.2 CSR1010

CSR1010 devices [77] from Cambridge Silicon Radio (CSR), now Qualcomm Technologies International (see Figure A.2), have a BLE 4.1 radio with direct single-ended 50 W antenna connection, which allows them to transmit BLE data in any direction, at a distance between 20 and 30 m depending on existing obstacles; a 16-bit microprocessor with 64 Kbytes RAM and 64 Kbytes ROM; 1 µA integrated key scanning hardware; PWMs and quadrature decoders; peripheral I2C and SPI (debug); analog IOs and UART interface; up to 32 re-assignable programmable digital IOs; up to 4.4V direct supply connection for Li-poly batteries.

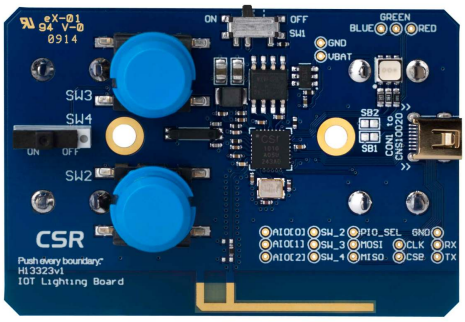


Figure A.2: CSR1010 device from Qualcomm Techonolgies International [77].

A.1.3 LightBlue Bean

The LightBlue Bean from Punch Through [78] (see Figure A.3) was suitable for the objective of our OperaBLE prototype because of the integration of BLE module into the board because of the communication technology used to transmit information. The BLE 4.0 module included in this board is called LBM313 [79].

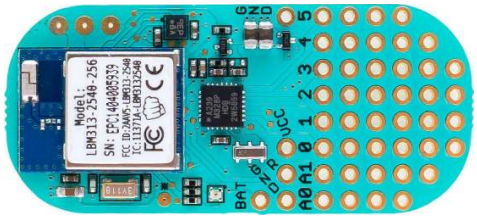


Figure A.3: LightBlut Bean device from Punch Through [78].

LightBlue Bean devices include I2C and SPI interfaces and several I/O pins. This board is additionally composed of RGB LED, temperature sensor, a BMA250 [80] three-axis accelerometer, and a breadboard for connecting other devices. All these parts are managed by an Arduino-compatible microcontroller with a working frequency of 8 MHz.

A.1.4 EFR32BG13

The EFR32BG13 devices from Silicon Labs [81] (see Figure A.4) include a 40 MHz ARM Cortex-M4 with a FPU microcontroller, 512 kB of flash and 64 kB of RAM. This chip has a RX sensibility of -103.3 dBM at 125 kbit/s. The EFR32BG13 chip is available in a BRD4104A radio board, which connects an on-board printed Inverted-F antenna to the 2.4 GHz RF input/output of the EFR32BG13, optimised for +10dBm output power, although the software can be configured between 0 and +10 dBm. The BRD4104A radio board is equipped with an 8 Mbit Macronix MX25R SPI flash, connected directly to the EFR32BG13. This radio board is plugged into the Wireless Starter Kit Mainboard PCB4001 Rev A03, which offers an entire set to develop and evaluate the EFR32BG13.

Figure A.4: EFR32BG13 device from Silicon Labs [81].

A.1.5 nRF52840

The nRF52840 DK is a single board development kit for applications on nRF52840 system on chip [61] (SoC) from Nordic Semiconductor (see Figure A.5). This SoC supports multiple communication protocols, including Bluetooth mesh, on which we focused our work. The nRF52840 SoC includes a 64 MHz ARM Cortex-M4 with FPU, 1 MB of flash and 256 kB of RAM. To speed up security functions, it incorporates a 128-bit AES/ECB/CCM/AAR co-processor. It includes a Bluetooth 5, IEEE 802.15.4, 2.4 GHz transceiver as well as a rich set of security features. Related to Bluetooth, the chip has a sensitivity of -95 dBm and -103 dBm for 1 Mbps and 125 kbps modes, respectively, and a configurable TX power from -20 to +8 dBm.

The PCA10056 Development Kit includes an nRF52840 chipset [82]. This module supports Bluetooth 5 and Bluetooth mesh, as well as other wireless communication protocols. It uses a 32-bit ARM Cortex M4 processor at 64MHz with FPU. It supports different data rates and radio transmission power for Bluetooth transmissions.

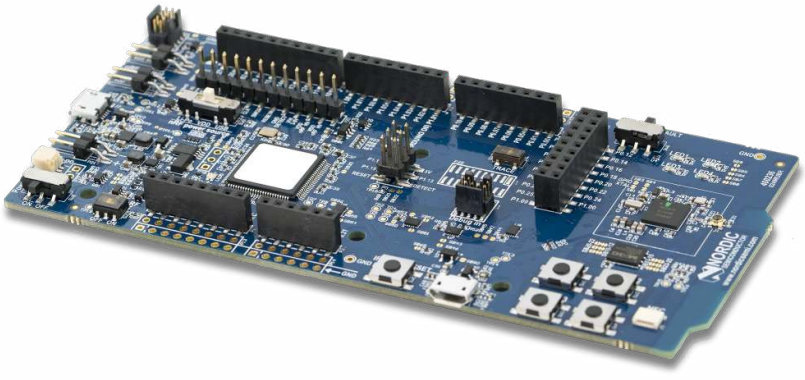


Figure A.5: nRF52840 DK from Nordic Semiconductor [61].

A.1.6 nRF Sniffer

The nRF Bluetooth Smart Sniffer [83] (see Figure A.6) is a tool for debugging BLE applications, sniffing or picking up every BLE packet between a selected device and the device it is communicating with. This tool is especially useful when developing a BLE application, because it allows us to know exactly what happens over the air between devices. The nRF Sniffer has an ARM Cortex-M0 processor; flash memory (128 kB); and RAM (16 kB). The BLE chip is the nRF51822, with a sensitivity of -93 dBm and a TX Power from -20 to +4 dBm.

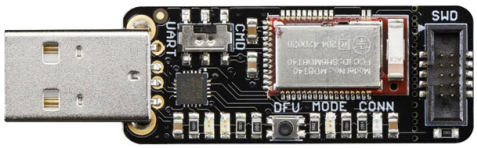


Figure A.6: nRF Bluetooth Smart Sniffer from Adafruit [83].

nRF Sniffer has a USB interface to connect it to a computer. Used in conjunction with Nordic Semiconductor software, it is possible to pass all captured packets to Wireshark to separate their fields.

A.2 Software

In order to provide a deeper understanding of the platforms used, these sections present the software used related to them.

A.2.1 Waspmote IDE

The Waspmote IDE [84] was used to program Waspmote devices. This IDE allows us to easily develop, compile and upload the programs to Waspmote boards via USB. Waspmote IDE is based on Arduino IDE, allowing the developed code to be fully compatible for both platforms with minor adjustments (the pin-out and the I/O scheme). After installation of Virtual COM Port (VCP) drivers from FTDI [85], the USB device can be accessed in the same way as through a standard COM port. This allows programs to be uploaded and debugged, while their serial output can be monitored.

Waspmote IDE includes the Waspmote API, with a multitude of open-source C libraries [57] for the management of the different sensors and radio modules, such as BLE library. The lower layers (Physical and Link layers) of the BLE 4.0 standard are implemented in the BLE chip (BLE controller), being necessary to flash it to modify its firmware. Since the necessary means are not available, our implementations were carried out entirely on the host part (Waspmote), using AT commands to communicate with the controller.

A.2.2 Apache Cordova

Apache Cordova [51] is an open-source mobile development framework. It allows developers to use standard web technologies such as HTML5, CSS3, and JavaScript to develop applications for the main mobile platforms: Android, iOS and Windows Phone. These applications are executed within wrappers targeted to each platform, and relied on standards-compliant API bindings to access each device's capabilities such as sensors, data, network status, etc. The application developed for the BLE preliminary evaluation (see Chapter 3) was developed using Apache Cordova.

A.2.3 Node.js

Node.js [86] is an asynchronous event-driven JavaScript runtime designed to build scalable network applications. Node.js is also open-source, and it allows the development of back-end applications in a simple and scalable way. Being open-source, node.js has a large community of programmers, who have developed multiple libraries, such as noble [87] (for central node) and bleno [88] (for peripherals), used for the development of our BLE server.

A.2.4 Android Studio

In order to develop our Android application for GreenISF (see Chapter 4), we used Android studio [89], the official integrated development environment for Google's Android operating system, designed specifically for Android development.Android Studio provides a unified environment to build apps for Android phones, tablets, Android Wear, Android TV, and Android Auto.

A.2.5 Simplicity Studio

Simplicity Studio IDE [90] allows developers to upload and debug the code in EFR32 devices easily through Ethernet or USB interfaces. Simplicity Studio is a suite of tools that includes an IDE based on Eclipse, as well as software examples and documentation about the Bluetooth mesh API developed by Silicon Labs. However, the SDK for EFR32 is proprietary, and most of the code is in binary and cannot be modified to change the behaviour of the standard.

A.2.6 Zephyr

The Zephyr project [25] is an open-source real-time operating system (RTOS) which supports multiple boards, such as the nRF52840 Development Kit (or PCA10056). This project is continuously updated by the community and already implements most of the Bluetooth mesh functionality. Being completely open source, Zephyr allowed us to develop Bluetooth mesh applications, as well as to fully access all the layers of the protocol to develop our proposal. To carry out the experiments, Zephyr version 2.2.0 was used.

Bibliography

[1] E. Borgia, "The Internet of Things vision: Key features, applications and open issues," *Computer Communications*, vol. 54, pp. 1–31, 2014.

[2] P. Scully, "Top 10 IoT applications in 2020," https://iot-analytics.com/top-10-iot-applications-in-2020/, 2020, accessed on Oct. 16, 2020.

[3] Federal Ministry for economic affairs and energy of Germany, "Plattform Industrie 4.0," https://www.plattform-i40.de/PI40/Navigation/EN.html, accessed on Oct. 15, 2020.

[4] IoT Now Magazine, "The industrial internet: Towards the 4th industrial revolution," https://www.iot-now.com/2016/10/20/53811-the-industrial-internet-towards-the-4th-industrial-revolution, accessed on Oct. 15, 2020.

[5] M. Collotta and G. Pau, "Bluetooth for Internet of Things: A fuzzy approach to improve power management in smart homes," *Computers & Electrical Engineering*, vol. 44, pp. 137–152, 2015.

[6] Bluetooth SIG, *Bluetooth Core Specification v4.0*. Bluetooth SIG, 2010, Available online: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=456433, accessed on Oct. 11, 2020.

[7] K. Townsend, *Getting started with Bluetooth Low Energy*. O'Reilly, 2014.

[8] Array of Things, University of Chicago, "Array of Things," https://arrayofthings.github.io, 2020, accessed on Nov. 4, 2020.

[9] W. Bronzi, R. Frank, G. Castignani, and T. Engel, "Bluetooth low energy performance and robustness analysis for inter-vehicular communications," *Ad Hoc Networks*, vol. 37, pp. 76–86, 2016, special Issue on Advances in Vehicular Networks.

[10] W. Narzt, S. Mayerhofer, O. Weichselbaum, S. Haselböck, and N. Höfler, "Be-in/be-out with bluetooth low energy: Implicit ticketing for public transportation systems," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 1551–1556.

[11] H. D. J. Jeong, W. Lee, J. Lim, and W. Hyun, "Utilizing a bluetooth remote lock system for a smartphone," *Pervasive and Mobile Computing*, vol. 24, pp. 150–165, 2015, special Issue on Secure Ubiquitous Computing.

[12] Bluetooth SIG, *Bluetooth Core Specification v5.2*. Bluetooth SIG, 2019, Available online: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=478726, accessed on Nov. 5, 2020.

[13] H. Kim, J. Lee, and J. W. Jang, "Blemesh: A wireless mesh network protocol for bluetooth low energy devices," in *2015 3rd International Conference on Future Internet of Things and Cloud*, 2015, pp. 558–563.

[14] G. Patti, L. Leonardi, and L. Lo Bello, "A bluetooth low energy real-time protocol for industrial wireless mesh networks," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, 2016, pp. 4627–4632.

[15] Qualcomm Technologies International, "CSRmesh Development Kit," https://developer.qualcomm.com/hardware/csr101x/csrmesh-development-kit, 2020, accessed on Nov. 04, 2020.

[16] T. Snekvik, "nRF OpenMesh," https://github.com/NordicSemiconductor/nRF51-ble-bcast-mesh, 2017, accessed on Nov. 04, 2020.

[17] Bluetooth SIG, *Mesh Profile Specification: 1.0.1*. Bluetooth SIG, 2019, Available online: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=457092, accessed on Sep. 12, 2020.

[18] Bluetooth SIG, *Mesh Model Specification: 1.0.1*. Bluetooth SIG, 2019, Available online: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=457091, accessed on Sep. 12, 2020.

[19] Bluetooth SIG, *Mesh Device Properties 1.2*. Bluetooth SIG, 2019, Available online: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=480005, accessed on Sep. 12, 2020.

[20] S. Slupik, "Bluetooth Mesh Networking: The Packet," https://www.bluetooth.com/blog/bluetooth-mesh-networking-the-packet/, accessed on Oct. 15, 2020.

[21] P. Svensson, "Bluetooth mesh networking FAQs," https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh/mesh-faq, accessed on Oct. 15, 2020.

[22] T. Øvrebekk, "Bluetooth Mesh for Industrial IoT. Nordic Semiconductor Blog," https://blog.nordicsemi.com/getconnected/bluetooth-mesh-for-industrial-iot, accessed on Oct. 15, 2020.

[23] Nordic Semiconductor, "nRF5-SDK-for-Mesh," https://github.com/NordicSemiconductor/nRF5-SDK-for-Mesh, 2020, accessed on Sep. 23, 2020.

[24] Silicon Laboratories, "Bluetooth Mesh Networking Learning Center," https://www.silabs.com/products/wireless/learning-center/bluetooth/bluetooth-mesh, 2020, accessed on Sep. 23, 2020.

[25] Linux Foundation, "Zephyr Project," https://www.zephyrproject.org/, 2020, accessed on Sep. 22, 2020.

[26] BlueZ Project, "BlueZ, Official Linux Bluetooth protocol stack," http://www.bluez.org/, 2020, accessed on Sep. 23, 2020.

[27] Bluetooth SIG, "Qualified Mesh Products," https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/topology-options/le-mesh/mesh-qualified/, 2020, accessed on Sep. 23, 2020.

[28] A. A. Kumar S., K. Ovsthus, and L. M. Kristensen., "An industrial perspective on wireless sensor networks — a survey of requirements, protocols, and challenges," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1391–1412, 2014.

[29] A. Varghese and D. Tandur, "Wireless requirements and challenges in industry 4.0," in *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, 2014, pp. 634–638.

[30] Y. Khan, "5 Essential Components of an IoT Ecosystem," https://learn.g2.com/iot-ecosystem, accessed on Oct. 23, 2020.

[31] I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[32] Consortium II, "Consortium II. Fact Sheet," http://www.iiconsortium.org/docs/IIC_FACT_SHEET.pdf, accessed on Oct. 16, 2020.

[33] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *Journal of Industrial Information Integration*, vol. 6, pp. 1–10, 2017.

[34] T. Stock and G. Seliger, "Opportunities of sustainable manufacturing in industry 4.0," *Procedia CIRP*, vol. 40, pp. 536–541, 2016, 13th Global Conference on Sustainable Manufacturing – Decoupling Growth from Resource Use.

[35] D. Küpper, "Embracing Industry 4.0 and Rediscovering Growth," https://www.bcg.com/en-es/capabilities/operations/embracing-industry-4.0-rediscovering-growth, accessed on Oct. 16, 2020.

[36] Belden Inc, "The Smart Factory of the Future. Part 1." http://www.belden.com/blog/industrialethernet/The-Smart-Factory-of-the-Future-Part-1.cfm, accessed on Oct. 17, 2020.

[37] Belden Inc, "The Smart Factory of the Future. Part 2." http://www.belden.com/blog/industrialethernet/The-Smart-Factory-of-the-Future-Part-2.cfm, accessed on Oct. 17, 2020.

[38] P. A. Laplante and S. Murugesan, "IT for a Greener Planet," *IT Professional*, vol. 13, no. 01, pp. 16–18, 2011.

[39] E. Kern, M. Dick, S. Naumann, A. Guldner, and T. Johann, "Green software and green software engineering - definitions, measurements, and quality aspects," in *First International Conference on Information and Communication for Sustainability*, 2013.

[40] D. Sontag, "Industrial IoT vs. Industry 4.0 vs. … Industry 5.0?" https://medium.com/the-industry-4-0-blog/industrial-iot-vs-industry-4-0-vs-industry-5-0-a5f9541da036, accessed on Oct. 17, 2020.

[41] R. Heydon, *Bluetooth Low Energy: The Developer's Handbook*, ser. Pearson Always Learning. Prentice Hall, 2012.

[42] Nordic Semiconductor, "Basic Bluetooth Mesh concepts," https://infocenter.nordicsemi.com/topic/com.nordic.infocenter.meshsdk.v3.2.0/md_doc_introduction_basic_concepts.html, 2019, accessed on Sep. 19, 2020.

[43] J. Marcel, "Reliable, Scalable, Secure Connections for Industrial IoT Environments," https://www.bluetooth.com/blog/reliable-scalable-secureconnections-for-industrialiot-environments/, 2019, accessed on Sep. 19, 2020.

[44] ETSI, "3GPP TS 27.010 V15.0.0," *Technical Specification*, 2018.

[45] Federal Information Processing Standards Publication, "Digital Signature Standard (DSS)," https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf, 2013.

[46] National Institute of Standards and Technology (NITS), "Advanced Encryption Standard (AES)," http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001, accessed on Sep. 17, 2020.

[47] R. Poovendran and J. Lee, "The AES-CMAC Algorithm," https://tools.ietf.org/html/rfc4493, 2006, accessed on Sep. 17, 2020.

[48] D. Whiting, R. Housley, and N. Ferguson, "Advanced Encryption Standard (AES)," https://tools.ietf.org/html/rfc3610, 2003, accessed on Sep. 17, 2020.

[49] HTC Corporation, "Nexus 9," https://www.htc.com/mea-en/tablets/nexus-9/, accessed on Oct. 6, 2020.

[50] Instituto de Investigación en Informática de Albacete - Universidad de Castilla-La Mancha, "Albacete Research Institute of Informatics," https://www.i3a.uclm.es/i3a_t/en, accessed on Oct. 23, 2020.

[51] The Apache Software Foundation, "Apache Cordova," https://cordova.apache.org/, 2020, accessed on Oct. 26, 2020.

[52] M. Jacobsson, C. Guo, and I. Niemegeers, "An experimental investigation of optimized flooding protocols using a wireless sensor network testbed," *Computer Networks*, vol. 55, no. 13, pp. 2899–2913, 2011.

[53] C. Garrido-Hidalgo, D. Hortelano, L. Roda-Sanchez, T. Olivares, M. C. Ruiz, and V. Lopez, "Iot heterogeneous mesh network deployment for human-in-the-loop challenges towards a social and sustainable industry 4.0," *IEEE Access*, vol. 6, pp. 28 417–28 437, 2018.

[54] MathWorks, "ThingSpeak IoT Analytics," https://thingspeak.com, 2020, accessed on Oct. 1, 2020.

[55] L. Roda-Sanchez, C. Garrido-Hidalgo, D. Hortelano, T. Olivares, and M. Ruiz, "Operable: An iot-based wearable to improve efficiency and smart worker care services in industry 4.0," *Journal of Sensors*, vol. 2018, 2018.

[56] Bluetooth SIG, "Qualified Bluetooth Products," https://launchstudio.bluetooth.com/Listings/Search, 2020, accessed on Sep. 23, 2020.

[57] Libelium Comunicaciones Distribuidas S.L., "Waspmote API Repository," https://github.com/Libelium/waspmoteapi, 2019, accessed on Sep. 22, 2020.

[58] K. MacKay, "ECDH and ECDSA for 8-bit, 32-bit, and 64-bit processors," https://github.com/kmackay/micro-ecc, 2017, accessed on Sep. 23, 2020.

[59] B. Gladman, "AES code," https://github.com/BrianGladman/aes, 2019.

[60] Silicon Laboratories, "Bluetooth Kit for EFR32," https://www.silabs.com/products/development-tools/wireless/bluetooth/blue-gecko-bluetooth-low-energy-soc-starter-kit, 2019, accessed on Sep. 22, 2020.

[61] Nordic Semiconductor, "nRF52840 Product Specification v1.1," https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.1.pdf, 2019, accessed on Sep. 22, 2020.

[62] Libelium Comunicaciones Distribuidas S.L., "Waspmote Technical Guide," http://www.libelium.com/downloads/documentation/waspmote_technical_guide.pdf, 2019, accessed on Sep. 22, 2020.

[63] R. B. Miller, "Response time in man-computer conversational transactions," in *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, ser. AFIPS '68 (Fall, part I). New York, NY, USA: Association for Computing Machinery, 1968, p. 267–277.

[64] M. Baert, J. Rossey, A. Shahid, and J. Hoebeke, "The bluetooth mesh standard: An overview and experimental evaluation," *Sensors (Basel, Switzerland)*, vol. 18, 2018.

[65] Y. Murillo, A. Chiumento, B. Reynders, and S. Pollin, "SDN on BLE: Controlling Resource Constrained Mesh Networks," *CoRR*, vol. abs/1902.02233, pp. 1–7, 05 2019.

[66] Bluetooth SIG, *Bluetooth Core Specification v5.0*. Bluetooth SIG, 2016, Available online: https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=421043, accessed on Sep. 21, 2020.

[67] E. Midttun, "Things your should know about Bluetooh mesh," https://devzone.nordicsemi.com/nordic/nordic-blog/b/blog/posts/things-you-should-know-about-bluetooth-mesh, August, 7 2017, accessed on Sep. 21, 2020).

[68] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2011, pp. 73–84.

[69] J. van Greunen and J. Rabaey, "Lightweight time synchronization for sensor networks," in *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications*, ser. WSNA '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 11–19.

[70] K. Geissdoerfer and G. von Hünefeld, "Pulse Arrival Time estimation in a synchronised Body Sensor Network Project in Advanced Network Technologies," Master's thesis, TU Berlin, 2016.

[71] Nordic Semiconductor, "nRF52840 Product Specification - Clock Control," https://infocenter.nordicsemi.com/topic/ps_nrf52840/clock.html?cp=4_0_0_4_3, 2019, accessed on Sep. 22, 2020.

[72] Epson Timing Devices, "High-Accuracy 32.768 kHzDTCXO," https://www5.epsondevice.com/en/information/technical_info/pdf/pb_tg3541ce.pdf, 2019, accessed on Sep. 22, 2020.

[73] S. M. Darroudi, R. Caldera-Sánchez, and C. Gomez, "Bluetooth Mesh Energy Consumption: A Model," *Sensors*, vol. 19, no. 5, 2019.

[74] Energizer, "Energizer: CR2032 Lithium Coin Technical Datasheet," http://data.energizer.com/PDFs/cr2032.pdf, 2014, accessed on Sep. 22, 2020.

[75] Libelium Comunicaciones Distribuidas S.L., "Success Stories," https://www.libelium.com/success-stories/, 2020, accessed on Sep. 22, 2020.

[76] Silicon Laboratories, "Silicon Laboratories Web Page," https://www.silabs.com/, 2020, accessed on Sep. 22, 2020.

[77] Qualcomm Technologies International, Ltd., "CSR1010 Datasheet," https://www.qualcomm.com/media/documents/files/csr1010-data-sheet.pdf, 2015, accessed on Oct. 26, 2020.

[78] Punch Through, "Bean," https://punchthrough.com/bean/, 2020, accessed on Oct. 26, 2020.

[79] Punch Through, "LightBlue LMB313," http://www.punchthroughdesign.com/products/lbm313-module/, 2020, accessed on Oct. 26, 2020.

[80] Bosch Sensortec, "Digital Triaxial Acceleration Sensor," http://www1.futureelectronics.com/doc/BOSCH/BMA250-0273141121.pdf, 2011, accessed on Oct. 2020.

[81] Silicon Labs, "EFR32BG13 Blue Gecko Bluetooth Low Energy SoC Family Data Sheet," https://www.silabs.com/documents/public/data-sheets/efr32bg13-datasheet.pdf, 2020, accessed on Oct. 2020.

[82] Nordic Semiconductor, "Bluetooth Low Energy, Bluetooth mesh, NFC, Thread and Zigbee development kit for the nRF52840 SoC," https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52840-DK, 2018, accessed on Sep. 22, 2020.

[83] Adafruit, "Bluefruit LE Friend - Bluetooth Low Energy (BLE 4.0) - nRF51822 v3.0," https://www.adafruit.com/product/2267, 2017, accessed on Sep. 22, 2020.

[84] Libelium Comunicaciones Distribuidas S.L., "Waspmote IDE - v06," https://development.libelium.com/waspmote-ide-v06/, 2020, accessed on Oct., 2020.

[85] Future Technology Devices International Ltd, "Virtual COM Port Drivers," https://www.ftdichip.com/Drivers/VCP.htm, 2016, accessed on Sep. 22, 2020.

[86] OpenJS Foundation, "Node.js," https://nodejs.org, 2020, accessed on Oct. 26, 2020).

[87] S. Mistry, "A Node.js BLE (Bluetooth Low Energy) central module," https://github.com/noble/noble, 2015, accessed on Oct. 26, 2020.

[88] S. Mistry, "A Node.js module for implementing BLE (Bluetooth Low Energy) peripherals," https://github.com/noble/bleno, 2015, accessed on Oct. 26, 2020.

[89] Google LLC, "Android Studio," https://developer.android.com/studio, 2020, accessed on Oct. 26, 2020.

[90] Silicon Laboratories, "Simplicity Studio 4," https://www.silabs.com/products/development-tools/software/simplicity-studio, 2020, accessed on Sep. 22, 2020.