Tampere University

Md Shohidur Rahman

# SECURE LIFECYCLE MANAGEMENT OF BLE NODES

Information Technology
Master of Science Thesis
August 2020

# ABSTRACT

Md Shohidur Rahman: Secure Lifecycle Management of BLE Nodes
Master of Science Thesis
Tampere University
Information Technology, Communication Systems and Networks
August 2020

---

Aiming the novel applications in the healthcare, fitness, beacons, security, and home entertainment industries Bluetooth Special Interest Group (Bluetooth SIG) has designed and marketed a low power personal area network, which is known as Bluetooth Low Energy (BLE). Billions of BLE devices are shipped every year across the world, and these devices have become very common in different types of deployments. The key consideration of this research is to find a suitable way of managing bulk Bluetooth Low Energy(BLE) nodes in any low powered wireless networks, remotely and in a secure manner. Some research in the low powered BLE networks and device management were required to materialize this objective. A literature review was done for that required research. After having a good understanding, which came from the literature review, a study was done to explore a few available device management solutions that existed until today. This study helped to understand the application domains, solutions, and their lacking if there were any. There were some shortcomings in the remote secure management of the vast number of BLE nodes, which become more prevalent in many wireless applications of today. Due to the resource constraints in those devices, it's not possible to try trusted and tested solutions from other domains here. A couple of different management strategies were proposed to address these issues. The base of the proposed device management solution is the limitations that were found in earlier studies. The theoretical solutions seemed very promising and has been implemented in the lab. The proposed theoretical solutions were implemented and tried in the testbed as a proof of concept. The proposed solutions have eased the secure device management of bulk BLE nodes remotely in any BLE low powered networks. One of the proposed solutions has also opened the possibility to try our trusted and tested IP protocol to manage BLE nodes. The work of enhancing device management in low powered network will continue beyond this thesis.

Keywords: M.Sc. thesis, IoT, 6LoWPAN, BLE, Zigbee, SIG

# PREFACE

This thesis has been written as a partial fulfillment of Tampere University, Finland's pre-requisite for a Master of Science in Information Technology. In collaboration with the Pervasive Computing Laboratory, Tampere, University, Finland, and Helvar Oy Ab, Espoo, Finland, all the research and practical tasks covered under this thesis were carried out. The work was supervised by the University of Tampere's research scientist Bilhanan Silverajan and Prof. Mikko Valkama.

I am starting with the name of God, the Most Gracious, the Most Merciful. Without his mercy and favor, It wouldn't be possible for me to continue my studies here, which differs significantly in different aspects compared to my home country.
After that, I would like to thank Bilhanan Silverajan most sincerely for believing in my abilities and taking me to his research team. I wouldn't end up finding an exciting and motivating research topic like this without his help. I would like to extend my profound gratitude to Antti Kolehmainen, who assisted me a lot throughout my thesis. He was my go-to guy for any Linux related help. It was, indeed, an immense pleasure to work with everyone at the team. I would like to take this opportunity to thank everyone in the team.

I would like to devote this study to my family, my parents, brothers, sisters, and my lovely wife. I couldn't get that far without their sincere help. Every little effort they made has laid the stairs for me that helped me pursue my dreams.

Finland, 18th August 2020

Md Shohidur Rahman

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| AFH | adaptive frequency hopping |
| ATT | Attribute Protocol |
| BLE | Bluetooth Low Energy |
| DFU | Device Firmware Upgrade |
| ECDH | Elliptic Curve Diffie Hellman |
| GAP | Generic Access Profile |
| GATT | Generic Attribute Profile |
| HCI | Host Controller Interface |
| IoT | Internet of Things |
| IRK | Identity Resolving Key |
| L2CAP | Logical Link Control and Adaptation Protocol |
| LTK | Long Term Key |
| MITM | Man in the middle |
| OOB | Out of Band |
| OUI | Organization Unique Identifier |
| RDM | Remote Device Management |
| SIG | Bluetooth Special Interest Group |
| SM | Security Manager |
| STK | Short Term Key |
| TK | Temporary Key |
| URL | Uniform Resource Locator |
| URN | Uniform Resource Name |
| UUID | Universally Unique ID |
| WHAN | Wireless Home Automation Network |
| WPAN | wireless personal area network |

# 1 INTRODUCTION

As the Internet of Things (IoT) continues to grow, more and more devices constrained by processing power, memory, and battery are getting connected to gather and share data. One of the biggest challenges of this kind of deployment is finding a communication protocol that can fulfill all the requirements by letting those smart objects to talk with the expense of the lowest possible energy. Bluetooth Low Energy (BLE) has been developed by the Bluetooth Special Interest Group (SIG), focusing on low power communication. Despite having the Bluetooth brand, BLE is a different technology from Bluetooth, addressing different design goals and targets different market segments. This technology was adopted in more than one billion devices within the first couple of years since the smartphone manufacturers made BLE available to their platforms. Now BLE is an emerging low powered wireless technology powering various applications. It is expected that there will be billions of BLE devices in the upcoming years since it has gained lots of research interest in different application [1], [2], [3], [4], [5], [6] domains. Some of these applications require many to many communications, while some others require one to one communication. Different networks have been built based on Bluetooth Low Energy to fulfill such requirements. BLE Mesh [7] is one of the two prevalent types of networks that have been built based on BLE. BLE mesh has enabled many-to-many communication over Bluetooth radio. The other common type of BLE based network is known as IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) [8]. The potential management challenges BLE devices in such networks will bring have not been studied much, and the current IoT device landscape has not reached the desired level of maturity. For BLE devices to thrive, device management capabilities need to be evolved. BLE device management is the process of authenticating, provisioning, configuring, monitoring, and maintaining the device firmware and software that provides its functional capabilities. The execution of these management tasks remotely in a secure fashion would be even more challenging. This thesis investigates and addresses the challenges in the remote secure device lifecycle management of BLE nodes in low power networks.

## 1.1 Research Questions

Following are the research questions of this thesis:

1. What is the current state of low power networks?
2. What is the current state of device lifecycle management in low power networks?

3. What is the current state of the firmware update in BLE nodes?
4. What are the ways to perform device management of BLE nodes in BLE mesh networks?
5. Is it possible to use Lightweight Machine-to-Machine (LwM2M) Client for managing BLE nodes in 6loWPAN network?

First three questions were explored to have sound background information and to form a firm foundation to answer the fourth and fifth questions which are the focus of this thesis.

## 1.2 Scope

The scope of the first research question is to explain what is meant by low power network and studying different low powered networks existed until today. The second research question covers the IoT device lifecycle management and challenges in the device lifecycle. The third question focuses on a specific device management task which is the firmware update. State and the challenges of this specific management task have been explored to answer this question. To answer fourth and fifth questions, this work has implemented three solutions for remote and secure lifecyle management of BLE nodes.

## 1.3 Methodology

This thesis work has focused on two purposes; the first focus was to give a literature review of low powered networks and, device lifecycle management in low powered networks. The research papers, articles, and news related to the topic were the basis of the literature review. Access to those resources was gained via Tampere University, while on campus, Andor and Science Port were used to access resources off-campus. The Google search engine was used in some cases to find some specific news, articles, or any other related documentation. In the list below, some of the used search phrases have been mentioned:

- Low powered networks
- Application domains for low powered networks
- Comparison between different low powered protocols
- Future prospect of low powered networks
- Prospect of BLE in future low powered networks IoT Management
- Device lifecycle management
- Device management challenges
- Device lifecycle management practice in the industries
- Device lifecycle management in the low powered networks
- Challenges in device lifecycle in low powered networks
- Management of BLE Nodes in low powered networks

All the documents that appeared using the search phrases mentioned above were scrutinized to find the most relevant one to the thesis. Those pertinent papers were used while writing the literature review and have been mentioned as references. Proposed device management for BLE Nodes was done by researching existing models and finding the limitations of these models in the application domains of interest. A suitable device lifecycle management for BLE nodes was developed. The developed solution was tested with different scenarios in the test lab.

## 1.4 Structure

The writing of the thesis is organised in eight chapters. It commences with an introductory chapter. The initial idea behind this opening chapter is to introduce the topic and the thesis to the readers. Creating interests and preparing for the yet to come information is also another intent of this chapter. Chapters 2, 3, and 4 cover the theory part. Chapter 2 covers the state of the art of the low powered networks in the perspective of Bluetooth Low Energy (BLE). Chapter 3 focuses on the device lifecycle of IoT devices in general. Chapter 4 focuses on firmware update. Chapter 5 gives the insights of the architecture and design of the implemented device management. In chapter 6, the implementation and testing of the implemented solutions have been covered. Chapter 7 discusses the result and the future work that could be done. The conclusion has been done in the chapter 8.

# 2 LOW POWERED COMMUNICATION PROTOCOLS

In the past, conventional communication protocols evolved to meet the needs of bandwidth-hungry applications like video streaming or large file downloads. In today's booming IoT industry, there have been myriad applications to address different challenges of modern life. In many of these applications, the senders and receivers of data are not human, but thousands of tiny, constrained devices. Those tiny devices don't require high up-link or down-link speeds for communication to happen since they communicate sparingly and transmit only a few kilobytes. As high data rates is proportionate to high power consumption, the traditional bandwidth-hungry communication protocols are not suited for these kind of IoT applications. Such applications requires a different sorts of communication protocol which will fulfill the applications requirements and will be very thrifty in case of energy consumption. The type of communication protocols that are suitable for such application domains are known as low powered communication protocols. In other words, low powered communication network is a type of wireless telecommunication that has been designed focusing on communication at a low bit rate, with low power, among things, also known as objects. Such objects or things operate on battery. The low powered networks have been classified based on their area of coverage [9]. In this section, related short-range communication protocols will be discussed briefly.

## 2.1 IEEE 802.15.4

To develop and maintain wireless and wired communications standards, the Institute of Electrical and Electronics Engineers (IEEE) supports many working groups. The 802.15.4 is arguably the most substantial standard for low data rate wireless personal area networks (WPANs). The intention was to offer the underlying lower network layers of a type of wireless personal area network, which focuses on low-cost, low-speed ubiquitous communication between devices. The initial target was low-data-rate monitoring and controlling applications to extend life through low-power-consumption. The IEEE STD 802.15.4 [10] specifies the Radio Frequency (RF), Physical Layer (PHY) and Medium Access Control (MAC) layers. There has been a variety of custom and industry-standards based networking protocols that can sit atop this IEEE stack. This has added the capability to create self-healing mesh rapidly. The frequency, power, modulation, and other wireless conditions of the link are defined in the PHY layer. The MAC layer is responsible for defining the format of the data handling. The other layers define other measures for handling the data and related protocol enhancements, including the appli-

cation itself. The 802.15.4 standard defines the star and peer-to-peer common network topologies.

## 2.2 ZigBee

The Zigbee [11] specification created by ZigBee Alliance is the most deployed enhancement to the 802.15.4 standard. The organization maintains, supports, and develops more sophisticated protocols for advanced applications. It has been developed based on low-power wireless IEEE802.15.4 networks standard and designed to connect personal low powered wireless devices within a small area. It uses layers one and two from 802.15.4

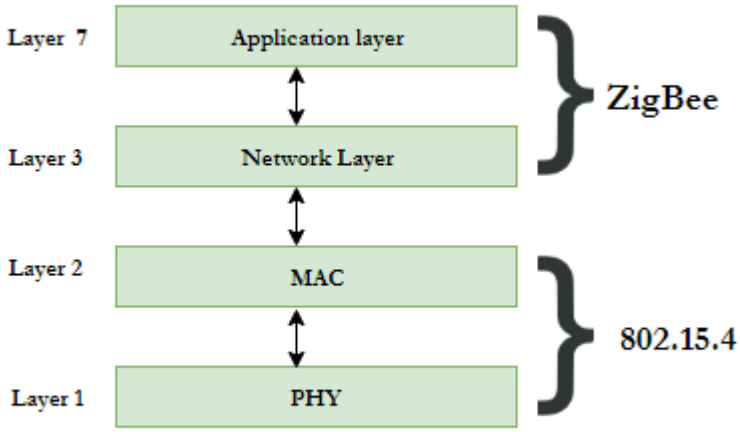


***Figure 2.1.*** *ZigBee protocol stack*

as base and uses layer three above to define the ZigBee protocol. The additional communications features are defined using layer 3 and 4 (Fig 2.1). These features include encryption for security, authentication with valid nodes, and a data routing and forwarding capability that enables mesh networking. The most common use of ZigBee is wireless sensor networks using the mesh topology. The ZigBee supports mesh, star and tree network topology [12], [13]. The main benefit the mesh protocol brings is that it allows any node to communicate with any other node directly if within the range. If not directly within the range, still communication can happen through multiple additional nodes. The network then can encompass a large area. In the case of a disabled node, it still can function, thus increase the network reliability. Zigbee also has a version that supports energy harvesting [14] for which it does not have battery or ac mains power available. With a maximum 250 kilobytes data rate, this low powered wireless technologies best fits with the applications, which are delay tolerant. ZigBee can accommodate up to 255 devices within a maximum of 100 m [15], [11]. It has been widely used in different applications like home and commercial building automation, industrial plant monitoring, fitness, wellness, and intensively in health and aging population care [16]. In agriculture and environmen-

tal monitoring [17] Zigbee has been identified as a suitable solution. ZigBee has been around for more than 20 years now and is widely used. It is a great option for many applications.

## 2.3 6LoWPAN

IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) [18] allows constrained devices with limited power and processing capabilities to participate in the IoT natively using IPv6.

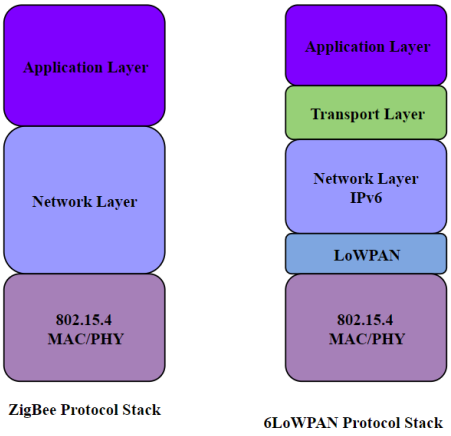


***Figure 2.2.*** *ZigBee and 6LowPAN protocol stack*

To allow IPv6 packets to be sent and received over IEEE 802.15.4 based networks, the 6LoWPAN defined an encapsulation and header compression mechanism. The primary function of 6LoWPAN is to transmit and receive IPv6 packets over 802.15.4 links. To achieve this, it has to accommodate the 802.15.4 maximum frame size over the air. A packet with the size of the IPv6 Maximum Transmission Unit (MTU) (1280 bytes) can be easily sent as one Ethernet frame, but in the case of 802.15.4, it needs an adaptation layer. To solve the issue of transmitting an IPv6 MTU by fragmenting the IPv6 packet at the sender and reassembling it at the receiver, 6LoWPAN acts as an adaptation layer between the IPv6 networking layer and the 802.15.4 link layer. 6LoWPAN supports different types of topologies like mesh and star topology. Since it is using the IEEE 802.15.4 protocol at the physical layer [6], [19] like Zigbee, this is the most competitive alternative of Zigbee. Figure 2.2 [19] shows Zigbee and 6LoWPAN protocol stack side by side.

## 2.4 BLE

Bluetooth Low Energy (BLE) [20], formerly marketed as Bluetooth Smart, is a variation of the Bluetooth wireless standard [21]. BLE has been developed by Bluetooth Special Interest Group (Bluetooth SIG), aiming at low power consumption noble applications in the healthcare, fitness, beacons, security, and home entertainment industries in December

2009 as part of the Bluetooth 4.0 specification. Devices having BLE are capable of communicating without cables while maintaining high levels of security. Due to its low power consumption and low cost, BLE has played a pivotal role in the evolution of applications from high-speed automotive devices to sophisticated medical devices. It is one of the most widely used protocol in today's world [22], [23]. BLE protocol stack uses popular 2.4 GHz [24] band.

### 2.4.1 BLE Network Topology

**Broadcasting and Observing:**
A Bluetooth Low Energy device communicates with the outside world in two ways: broadcasting or connections. The act of sending data out to all the listening devices is known as broadcasting. In this connectionless broadcasting, data could be sent out to any scanning device or receiver in the listening range. While talking about broadcasting, two roles are defined. These are broadcaster and observer. The device in the broadcasting role sends non-connectable advertising packets periodically to anyone within the range whoever willing to accept them. The device in an observer role repeatedly scans for any advertising data. Once the observer receives the advertising data, it further requests scan response data. Broadcasting is the only way to send data to more than one pear simultaneously.
**Connections:**
The permanent, periodical data exchange of packets between two devices is known as connection in BLE. This is, therefore, inherently private. The master (central device) scans the advertising packets that are connectable. In a suitable time, it initiates a connection. In a connected state, the central device manages the timing and initiates the periodical data exchanges.

The slave (peripheral device), on the other hand, sends connectable advertising packets periodically and accepts incoming connections. When a successful connection is established, the peripheral follows the central's timing and exchanges data regularly with it. When connected, the two devices usually define this is as a connection event. In other words, a connection is the periodical exchange of data at certain specific points in time (connection events) between the two peer devices involved in it. Though the central device manages the connection establishment, data can be sent independently by either device during each connection event, and the roles do not impose restrictions on data throughput or priority. Two devices involved in the connection event can power up, exchange data, and then go to sleep until the next connection event, which is one of the key benefits for power saving.

### 2.4.2 BLE Addressing

A device address is used to identify each BLE device. Nevertheless, these addresses are identical to the MAC addresses used in other communications protocols, BLE de-

vice addresses may often be modified at will. Because of this resemblance, BLE device addresses are commonly referred to as BLE MAC addresses.BLE currently supports following four different address types, each with a length of 48 bits:

- **Public IEEE Format-** Such addresses were obtained by the IEEE Registration Authority and are unique to the manufacturer. The 24 most significant bits of this address are the IEEE assigned to the Organization Unique Identifier (OUI). The company is free to change the 24 least significant bits. Because these addresses do not alter, they do not offer protection against identity tracking.
- **Random Static-** During manufacturing, these addresses are either burned into the silicon of the device or created during the power cycles of the device.
- **Random Private Resolvable-**This method of addressing can only be used if, during the bonding process, the Identity Resolving Key (IRK) is shared between the two devices. A second device which also has the IRK can then convert the random address back to the real address.
- **Random Private Non-Resolvable-**The device address is simply a random number in this addressing system, and at any time, a new device address can be created. This approach provides significant protection against identity tracking if new addresses are created relatively frequently.

### 2.4.3 BLE Throughput and Range

The theoretical data rate for Bluetooth Low Energy (BLE) is 1 Megabits per second (Mbps) and 2 Mbps for BLE 4.2 and BLE 5, respectively. Nonetheless, a BLE system's throughput would be much lower as it needs to compensate for various protocol's overhead and interference. There has been an investigation [25] to maximum throughput achievable in a simple BLE 4.2 network of two nodes. The maximum amount of throughput was found 221.7 Kilobits per second (kbps) under the condition of the wireless link is error-free, and the application always has data to transmit.

## 2.5 6LoWPAN over BLE

The implementation of 6LoWPAN over BLE does not require all of the features defined in the classical 6LoWPAN specifications, so the Internet Technology Task Force (IETF) proposed a standard RFC 7668 [26] which specifies the 6LoWPAN layer that enables IPv6 communication on BLE link layers. Figure 2.3 shows the 6LoWPAN layer which is on top of the BLE L2CAP layer and below the IPv6 layer. Although other 6LoWPAN standards define data fragmentation and reassembly functions, those functions are not included in RFC 7668. Data fragmentation and defragmentation are already being done in the BLE communication stack's L2CAP layer.

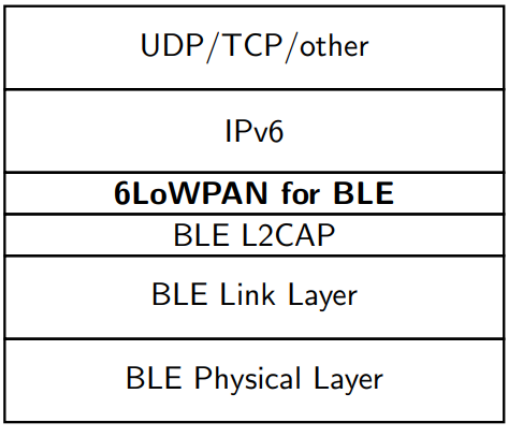


***Figure 2.3.*** *IPv6 on the BLE communication stack (Adopted from [26])*

### 2.5.1 Network topology

Figure 2.4 demonstrates the simplest case of a subnet that exchanges data over BLE links using IPv6. This simple subnet consisting of six BLE nodes, each of which acts as a peripheral BLE device and one BLE enabled border router which acts as a central BLE device. Each node device is connected via a single BLE data connection to the border router.

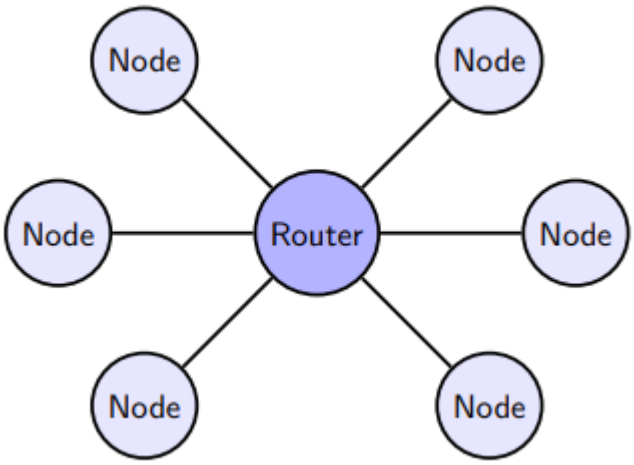


***Figure 2.4.*** *simple IPv6 over BLE subnet with 6 node devices and a single border router; nodes in the subnet may exchange data but cannot communicate with devices outside of the subnet (adapted from [26])*

An IPv6 convention specifies that IPv6 subnets will span a single link layer [27], yet a multilink model. In this kind of setup, each node maintains a separate link to the router which has been a more suitable network topology of IPv6 over BLE for constrained devices. The use of link-local communication in this multilink model has been limited only to individual BLE connections, link-local communication is not achievable between two

nodes. The nodes connected to the same boarder router have to use the shared prefix to exchange data among them. The border router does not act as a link-layer switch in IPv6 over BLE networks but as an IPv6 router [26]. Figure 2.5 shows a practical setup. In such a setup, BLE nodes have access to the Internet. All nodes share a 64-bit IPv6 prefix in the subnet. The border router takes the responsibility to create a BLE connection to the BLE Nodes for distributing the shared IPv6 prefix to all connected node devices and for forwarding IPv6 packets from and to the Internet.

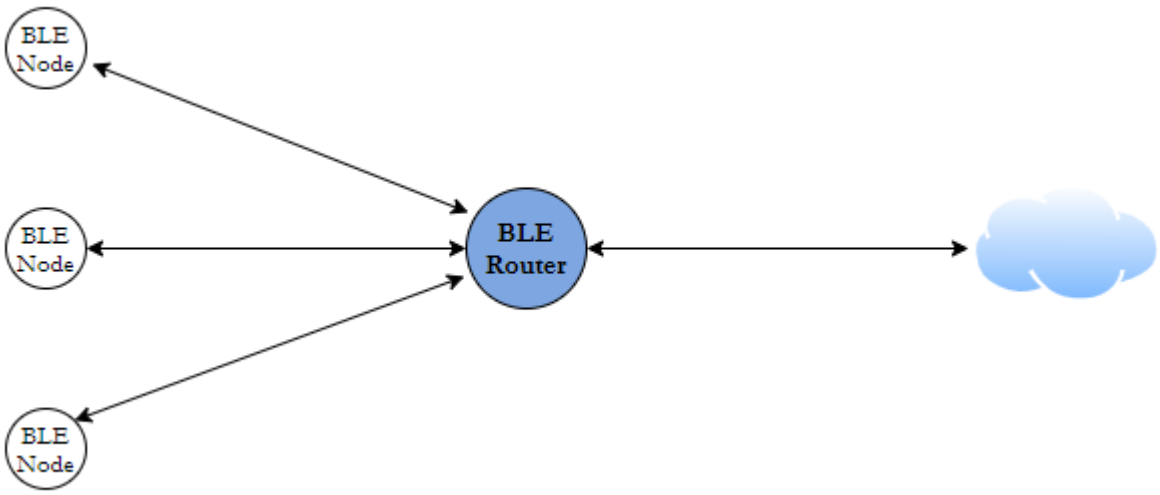


***Figure 2.5.*** *IPv6 over BLE subnet connected to the Internet; nodes may exchange data within the subnet or interact with devices on the Internet (adapted from [26])*

## 2.6 BLE Mesh

Even though BLE got widespread adoption in the industries, it lacked one most important feature, which is a requirement for many of today's applications in different domains. BLE does not support many-to-many topology, often referred to as mesh where multiple BLE devices are able to send messages to each other and forward messages to other devices in the network. In 2017, Bluetooth SIG released the "mesh specification" standardizing BLE's many-to-many potentially unrestricted features. Bluetooth Mesh is a Bluetooth Low Energy-based mesh networking protocol that enables many to many communication over Bluetooth radio. Following figure 2.6 taken from Nordic Semiconductor shows Bluetooth mesh stack (light blue) within the Bluetooth low energy protocol (dark blue).

### 2.6.1 Bluetooth Mesh: Layers and Functionalities

Bluetooth Mesh has a multi-layered architecture as below:

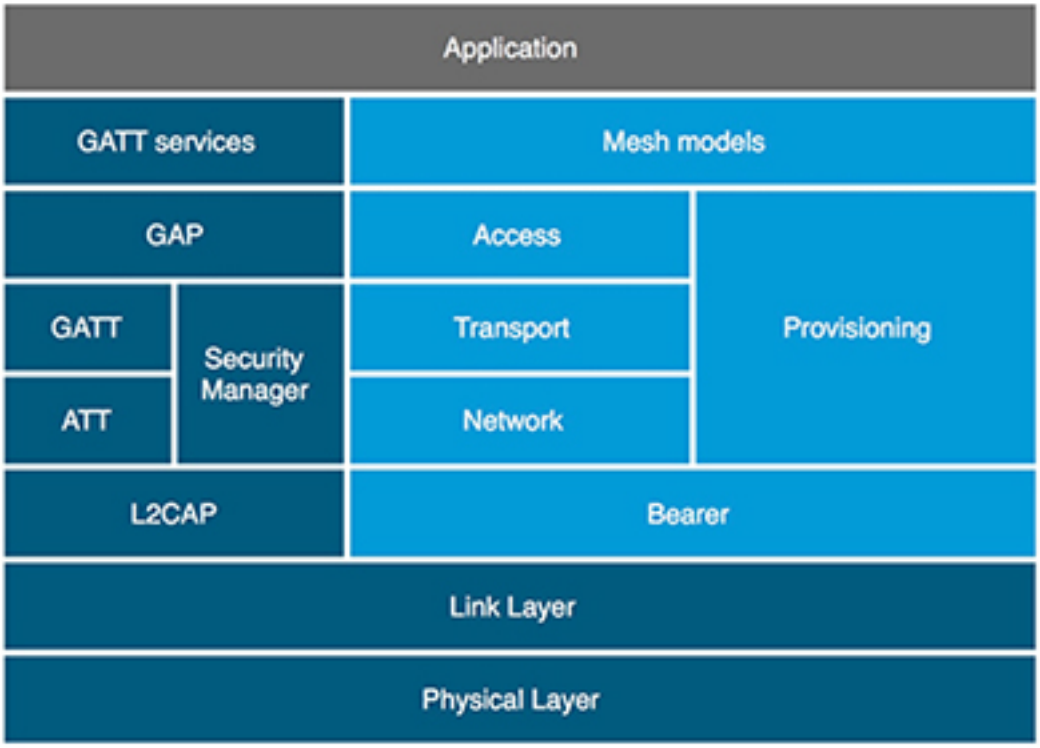


***Figure 2.6.*** *BLE Mesh and BLE*

- **Model Layer:** It defines a standard way to exchange application-specific messages.
- **Access Layer:** Defines a framework for ensuring that data is transmitted and retrieved in the appropriate context of the model and its associated application keys.
- **Upper Transport Layer:** This uses an application (or device specific key) to establish authenticated encryption of access layer packets. It also defines some control messages like Heartbeat messages to manage Friendship or to notify the behavior of nodes.
- **Lower Transport Layer:** This layer is responsible for segmentation and reassemble. When a complete upper layer packet can't be carried in a single network-layer packet, this layer segment this upper layer packet reliably. It does the opposite when it receives segmented packets from the lower layer.
- **Network Layer:** This layer is responsible for addressing the transport layer packets in the network. This layer helps to extend the range by forwarding messages through a relay node. The network layer authenticated encryption using the network key, also done in this layer.
- **Bearer Layer:** It defines how the network packets are exchanged between nodes.

### 2.6.2 Important BLE mesh terminologies

Different BLE mesh terminologies and their purpose will be addressed briefly in this section.

**Nodes:** Any device connected to a BLE mesh is termed as a Node. If a device is not part of a mesh network or has not joined a BLE mesh network is known as an unprovisioned device. After going through a provisioning process, a device can join the BLE mesh network, and then this device is called a node of that BLE mesh network. The figure 2.7 depicts these concepts.

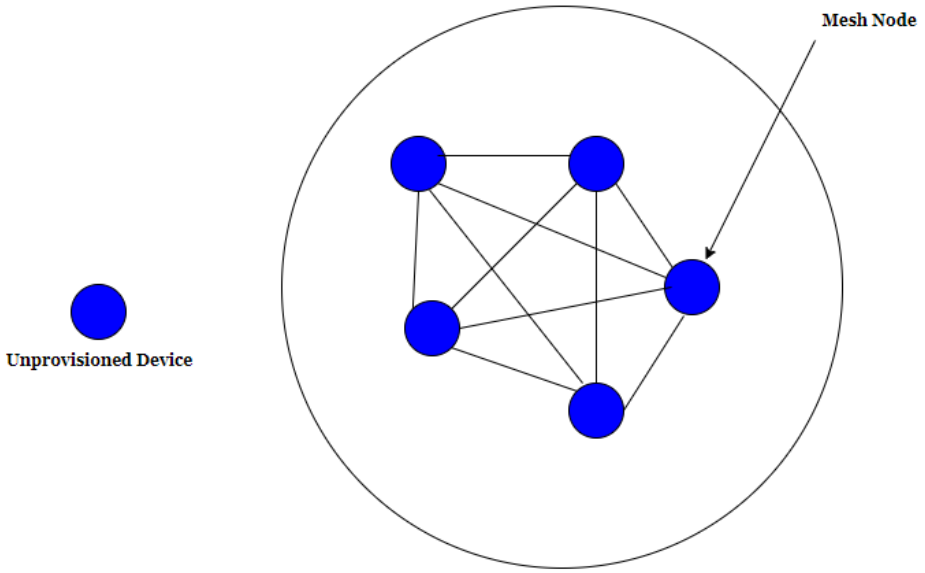


***Figure 2.7.*** *BLE mesh, Nodes and unprovisioned device*

A node could support all, any, or none of the following features in a BLE mesh network, which could be enabled or disabled at any time.

- **Relay nodes-** A relay node has the relay feature, which means it can re-transmit any broadcast messages of other nodes.

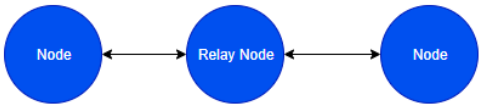


***Figure 2.8.*** *Relay Node*

- **Proxy nodes-** A proxy node helps communication to happen between a BLE mesh node and a non-mesh-supported BLE device. The protocol used for such communication is known as proxy protocol.

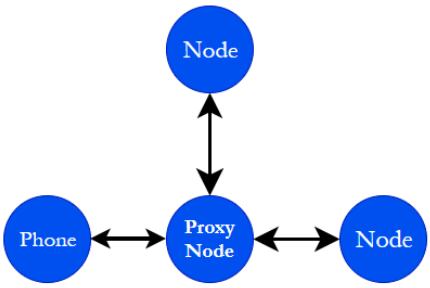


***Figure 2.9.*** *Proxy Node*

- **Friend nodes and Low power nodes-** Friend nodes operate together with a low-power node(LPN). A low-power node has a very scarce resource, therefore keep its radio off as much as possible to save energy. If any node wants to communicate with this low power node while it's sleeping, its friend node receives any data intended for the low power node. If any data needs to be sent by a low-power node, it sends it to its friend node and immediately goes to sleep mode to save energy.

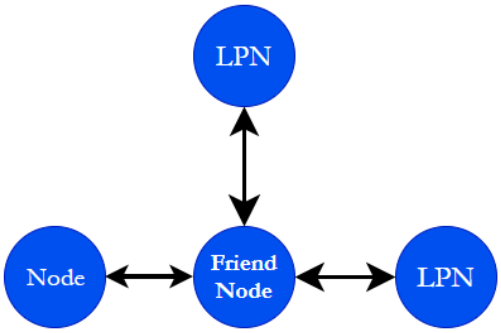


***Figure 2.10.*** *Friend nodes and low power nodes*

**Elements:** A node may have multiple components that can be handled independently. Such individual components are known as Elements.
Let's consider a luminaire as a node with multiple bulbs. Let's also think, every bulb of the luminaire could be turned on/off independently. In this case, different bulbs of the luminaire are considered as elements.
**States:** Elements can be in different situations, which are represented by state value. In the earlier example of the luminaire, the light bulbs can be on or off, which are the two-state of these elements.
**Properties:** Properties are additional state value information. Setting a temperature value, for example, as an outdoor or indoor temperature. There are two property types:

- Manufacturer property: only has read-only access

- Admin property: has both read-write access

**Models:** The whole or some functionality of a specific element is called a model. There exist the following three types of models:

- **Server model:** A set of states, state transitions, state binds, and messages that an entity containing the model may send or receive.
- **Client model:** The client model consists of GET, SET, and STATUS messages to be used during communication with the server model.
- **Control model:** It contains a server and a client model that enables communication with other server and client models.

**Messages:**Any data transmitted between BLE mesh nodes is referred to as the message. Among the nodes of a BLE mesh, there are different types of data transfer happens. To define these, a special opcode is used. The following are the three types of messages.

- **GET message:** This sort of message is used to get the status of one or more BLE mesh nodes.
- **SET message:** This type of message is used if it is required to change the value of a given state.
- **STATUS message:** The status message's value depends on the perspective. It may be a response to a GET message or an answer to a SET message that has been acknowledged.

**Addresses:** Messages should be sent to and from an address in a Bluetooth mesh network. There are three kinds of addresses available:

- **Unicast Address:** This kind of address is used to identify a BLE node uniquely
- **Group Address:**For identifying a group of BLE nodes, a group address is used. This grouping resembles physical grouping, such as all nodes within a specific room. An address for a group will either be:
  - Bluetooth SIG defined, also known as a SIG-Fixed Group Address. Defined address groups are All-proxies, All-friends, All-relays, and All-nodes group.
  - User-defined through the configuration of the application.
- **Virtual Address:** The address allocated to elements instead of nodes is known as a virtual address. A virtual address could be allocated to more than one element and spread across more than one node. This form of address is typically pre-configured by manufacturers during the manufacturing process.

**Publish/Subscribe:** Publish/Subscribe is another crucial concept in the BLE mesh. This publish-subscribe pattern is used for exchanging all message in the BLE mesh network. Publishing is known as the process of sending a message. Subscription is a setting that causes chosen messages to be sent to specific addresses to be processed. Messages are usually addressed to group addresses or virtual addresses. Following figure taken from BLE mesh developer guide explains this in a lucid way.

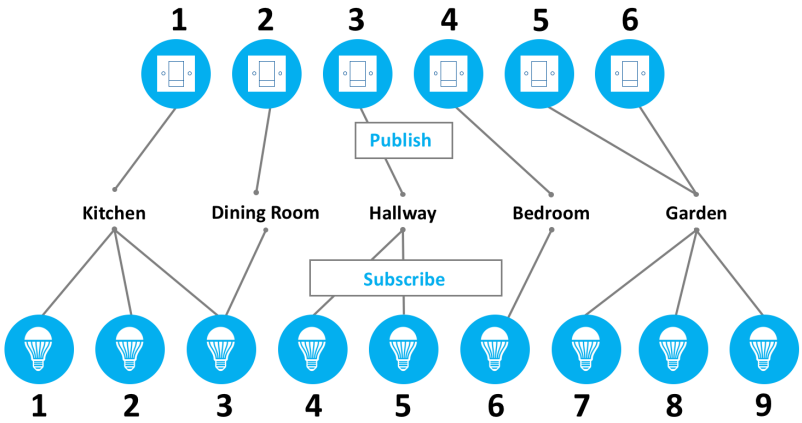


***Figure 2.11.*** *Publish-subscribe in Bluetooth mesh lighting control system [28].*

This is a BLE mesh network of 9 bulbs and 6 switches to control these bulbs. Every bulb is a node that can subscribe to more than one group addresses. Group addresses such as kitchen, dining room, hallway, bedroom, and garden are also available. As shown in the figure, the third light bulb has subscribed to two group addresses, namely Kitchen and Dining Room. Similarly, multiple nodes may publish their messages to the same group. In figure 5th and 6th, switches publish to the Garden address group, where light bulbs (7, 8 and 9) have subscribed. This kind of grouping makes reconfiguration of the devices that are subscribing to the same group easy.

### 2.6.3 Provisioning

Provisioning is the process by which a device becomes a working node in a BLE mesh. The provisioner is the system that provides the data needed to make other devices a node in the mesh network. Often the Provisioner is a Smartphone, Tablet, or PC. The entire provisioning process could be divided into five stages, as follows:

- **Stage 1: Beaconing-** The provisioning process begins with this stage. In this step, the unprovisioned device shows its interest to be provisioned. For this reason, the unprovisioned device sends an advertising known as the mesh beacon. Pressing the sequence button on the unprovisioned device activates the beaconing process.
- **Stage 2: Invitation-** When the provisioner senses the existence of an unprovisioned device, a provisioning invitation is sent to the unprovisioned device.

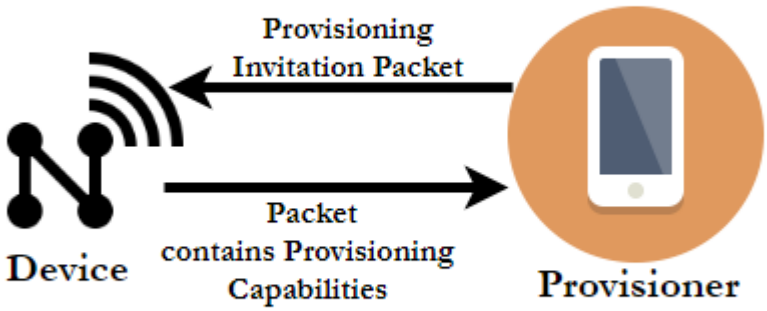


***Figure 2.12.*** *Provisioning Invite*

The unprovisioned device responds with its provisioning capabilities to the invitation with followings:

- Total number of device-supported elements
- List of supported security algorithms.
- Its supported Out-of-Band (OOB) technology.
- Device's output capabilities
- Device's input capabilities

- **Stage 3: Public Key Exchange-** In ECDH, public keys need to be transferred between two communication parties.

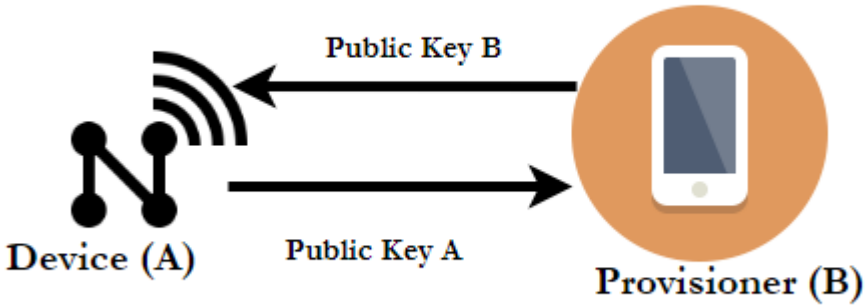


***Figure 2.13.*** *Public Key Exchange*

In this step, BLE mesh provisioner and the to be provisioned transfer their public keys over BLE or via an out-of-band (OOB) procedure supported by both devices.

- **Stage 4: Authentication-**The device to be provisioned is authenticated in this stage. The user's interaction is required with the provisioner and the unprovisioned device in this stage. The method of authentication depends on the previously agreed method in the earlier stage.
- **Stage 5: Provision Data Distribution-** At this point, a session key is generated by the provisioner and the device to be provisioned. The session key is created from its private key and obtained public key from the other side of the device. The generated session key is used to encrypt provisioning data like network key (NetKey), a device key, IV index security parameter and, unicast address. After this process, the unprovisioned device successfully joins the network and becomes a node in the mesh.

### 2.6.4 BLE Mesh Security

BLE mesh has made security a mandatory feature, whereas, in BLE, the implementation of security was optional and up to the developer. Every message in the BLE mesh is encrypted and authenticated. BLE mesh has an independent security procedure in place in every layer, such as Network Layer, Application Layer, and device level.Their purposes are as follows:

- **Network Layer Security (NetKay):** A NetKey ensures the network layer security. This is used to extract the network encryption key and the privacy key. The encryption and authentication of the network layer is done with this key. However, this key does not decrypt any application data.
- **Application Key (AppKey):** AppKey is assigned to a node group that works for the same application. This key does application-level decryption and authentication and valid only for one mesh network.
- **Device Key (DevKey):** Device Key (DevKey) is used to secure communication between the unprovisioned device and the provisioning device during the provisioning process.

**Protection against common attacks:** BLE mesh also has protection in place for frequent attacks in mesh networks.

- **Trash Can Attack:** It's not uncommon to remove or discard mesh nodes in the lifetime of a mesh network. An attacker can use this discarded node to gain access to the mesh network. This type of attack is known as trash can attack. BLE mesh has defined a procedure for removing any node to guard against such attacks. When a node is removed, its added in the blacklist, and network keys and application keys get renewed. These new keys are not distributed to the nodes on the blacklist.
- **Replay Attacks:** Valid messages are captured and stored with a malicious intent to replay these messages later to attack, which is known as replay attacks.

**Privacy:** BLE mesh uses a privacy key generated from NetKey to hide the message

header. This makes it difficult for some attackers to trace back the source of the message.

### 2.6.5 BLE Mesh Networks: Standardization

To meet the burgeoning interest of the industries in BLE mesh, standardization organizations like Bluetooth Special Interest Group (SIG) and the Internet Engineering Task Force (IETF) have come up with different initiatives. In this section status of such initiatives will be briefly discussed.

- **Bluetooth SIG: Bluetooth Smart Mesh-** The formation of the Bluetooth Smart Mesh Working Group in early 2015 focused on developing an architecture to enable support for the mesh topology with BLE. Since its inception, this initiative has received substantial support from 80 participating companies from a wide range of industries, including automotive, mobile telephony, industrial automation, home automation, and consumer electronics. Bluetooth SIG announced in their 2017 road map that mesh support many-to-many (m:m) device communications and is optimized for creating large-scale device networks. Such features are ideally suited for building automation, sensor networks, and other IoT solutions where tens, hundreds, or thousands of devices need to reliably and securely communicate with one another.
- **IETF: IPv6 over BLE Mesh Networks-** In late 2015 the IETF released the 'IPv6 over Bluetooth Low Energy' specification as RFC 7668 [29] in order to extend the IoT capillarity and its range of supported technologies. This specification considered only the star topology [30] and adapted 6LoWPAN [31], to support IPv6 over BLE networks. For enabling IPv6 over BLE mesh networks, a new draft specification has been developed by the IETF which extends the RFC 7668. This draft [29] was built on the assumption that there exist Link-Layer connections between a node and its neighbors over which IPv6 packets could be exchanged. The Internet Protocol Support Profile (IPSP) [32] is used to build such connections. In RFC 7668, there is a 6LoWPAN-based adaptation layer in between IPv6 and L2CAP. Such an adaptation layer provides IPv6 and User Datagram Protocol (UDP) header compression, which increases communication efficiency, and optimizes IPv6 neighbor discovery. These have made network configuration suitable for constrained devices in any BLE mesh topologies.
  Routing in such network is done in the IP layer. However, this specification did not determine the routing protocol to be used.

### 2.6.6 BLE Mesh Flooding vs. Routing

Two basic types of multihop model exist in BLE mesh networks, flooding, and routing. An benefit of flooding is its simplicity, because it does not require neighboring devices to link or a routing protocol. This is faster and consumes less memory since there is no

need to create a route and maintain the routing table. The downside of such a flooding based technique is the number of messages sent by network nodes for the purpose of end-to-end communication between two devices. Since data are flooded throughout the network, this approach may be inefficient. The inefficiency of flooding based technique is proportional to the network size, which means in a bigger size network routing based technique will have a benefit over flooding based technique. There have been some efforts to mitigate the inefficiencies in flooding based techniques. The Trickle [33] and node-density-aware rebroadcasting [34] are such technique which limits the overhead of flooding-based solutions.

# 3 DEVICE MANAGEMENT

The volume of connected BLE nodes being part of different IoT solutions is increasing significantly. The management of these BLE nodes in any IoT deployment a big challenge. To overcome these management issues, a detailed understanding of the related device management should be explored. In this chapter, the focus will be given on IoT device management, which is, in many aspects, very similar to BLE node management.

## 3.1 Device Management Goals

A well managed connected device might bring enormous benefits to the stakeholders. The goal of device management in a network could be broadly categorized as follows:

- **Optimizing the Cost:** Keeping costs low while meeting the goals is a fundamental requirement of any system; IoT systems are not an exception. Device management should be designed in such a way that, deployed IoT systems fulfill the user requirements, while keeping the cost under control.
- **Transparent Usage of the System:** Since the IoT system might be massive, there is a possibility for unforeseen activity, which may lead to unplanned usage of the system. Device management should be equipped with strategies and actions to control the impacts of those unforeseeable situations.
- **Fault Tolerance:** Some failure situations are predictable; the goal of this step is to detect those predicted failures, to control the effect of these, and finally to fix them.
- **Dynamic Adaptability:** Mostly, IoT systems deal with very dynamic environments where user requirements [35] might change in real-time. Dynamic Adaptability will help to merge those dynamic changes in the existing system without making any effect on the system. The flexible device management system will be able to address those changing requirements of the users. This is applicable in every stage of the life cycle, e.g., booting, run time, and commissioning.
- **To know about devices:** The device lifecycle management can help to manage, monitor, protect, and maintain all devices. A device lifecycle management dashboard can be accessed remotely to allow devices to be provisioned, controlled, and dismantled in real-time.
- **Enhancing productivity:** Network devices support applications that create value for end-users. However, application developers often have no time or willingness

to take responsibility for device management. Management of the device lifecycle assists in the efficient management of resources, thereby reducing the burden on developers of applications. It helps the developer to concentrate on the development of core value and consequently boost the organization's production.

- **Downtime reduction:** Device management assists in identifying flaws and bugs and amend them in real-time. Through keeping downtime low, it helps to improve service efficiency.
- **Data Feedback:** The vast connected devices producing huge amounts of data, a thorough analysis of each device's activities, could be achieved through a device lifecycle management.

## 3.2 Device Management Functions

Regardless of underlying protocol and devices in any IoT systems following key device management functionalities are expected to be performed:

- **Registration:** The primary device management function comprises a device registering/deregistering itself to a management server to maintain, retrieve, and update registration data.
- **Provisioning:** Provisioning is the process by which a device obtains necessary data to join a network from the management server.
- **Management services:** Once the device becomes a fully functioning member of the network, there will be few essential management services to maintain designated IoT services, and those include parameter configuration, status query, connection diagnose, and remote control.
- **Observation:** Devices will send data to the server periodically according to configured frequency. The server can also observe the device's data and make a representation.
- **Application data transmission:** This involves data from any application that could be obtained and distributed to any IoT clients or any other program for further analysis or processing.

## 3.3 Device Management lifecycle

IoT's future growth depends on how well we can simultaneously conduct remote bulk operations on many devices. The five fundamental Remote Device Management (RDM) building blocks of any device's entire lifecycle will be addressed here. Figure 3.1 summaries the stepwise function of a typical device management lifecycle.

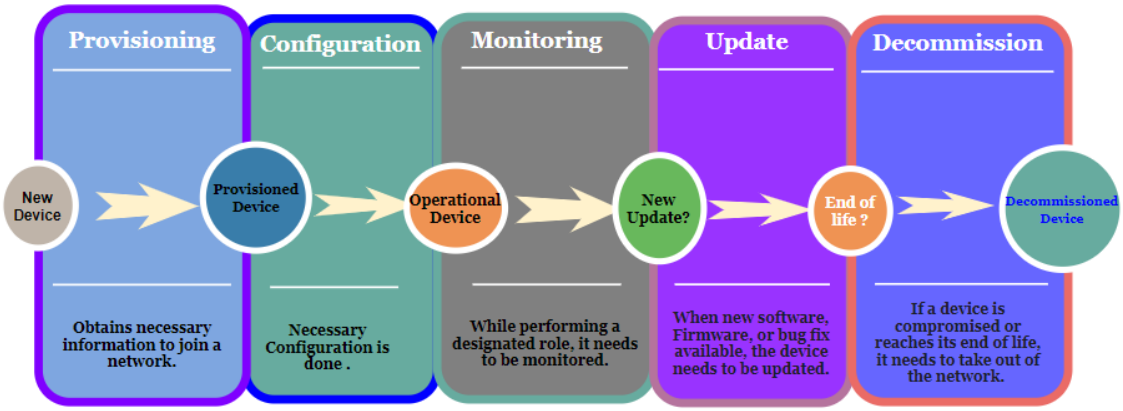


***Figure 3.1.*** *Device Management Lifecycle*

### 3.3.1 Provisioning:

There should be a mechanism to ensure that only trustworthy and secure devices are added when it is necessary to add new IoT devices to the network. The system by design should not allow bad actors to connect the IoT application and run untrusted software or operate on behalf of any unauthorized users. By definition, the word provisioning refers to the process of obtaining necessary information to join in the network, which includes Authentication as well. Authentication is the process by which ensures that devices only with the proper credentials get registered in the system. In this stage, the following two tasks are performed:

- **Boot-loading:** For making a device operational within a security domain, an appropriate firmware image must be located and securely booted. This happens once the device is installed and commissioned in its new premises. Secure boot loading a boot-loading variant refers to booting a device only with images that are trusted. Any additional update to the current firmware version (boot-loader or device image) must be made from a trusted source. For ensuring secure boot loading, there must be a way to check the firmware integrity on each boot-up. This check is programmed during configuration and enabled in the boot-loader.
- **Bootstrapping:** The term bootstrapping has been defined differently in different contexts. Bootstrapping is the process by which a new device uses its temporary credentials to derive subsequent keys and finally becomes a node of the network. Bootstrapping is a collective name for authentication, authorization, and finally distributing required keys securely to any smart object that eventually becomes a part of the network. Two entities participate in the bootstrapping procedure; one is the smart object that wants to be a network member, and another is the controller that provides the bootstrapping data. The smart object's interaction with the controller depends on the smart object's input/output capabilities.

### 3.3.2 Configuration

IoT devices are often delivered with a default configuration of some kind. With this initial configuration, no device is put into operation. Most of the time, devices with attributes such as its name and location and application-specific settings will need to be further configured by the end-user. If a device is installed to engage in an application that measures a particular room's light intensity and sends it back to the server, then specific parameters should be configured for this device. Parameters like a device ID, location, and frequency of sending data to the server should be configured. Parameters such as device ID, location, and frequency of sending data should be configured in the device. The end-user may want to remotely reset the device to a known-good state or recover from errors and apply new settings in order to have some level of control over the deployed devices. There may also be a situation where the device needs to be decommissioned. While doing a decommission, the trash can attack stated in the earlier section should be taken into consideration. Before decommissioning, it is good to reset the device to be removed to the default factory configuration. It makes it difficult for the attackers to use the existing configuration of the decommissioned device to attack the network.

### 3.3.3 Monitoring

The IoT system can be small and simple like home with few devices to big and complex like industry, even a city where thousands of devices connected to one network. In any case, the device's proper and secure operation has an immediate consequence on the purpose of the deployment. Small issues might lead to a devastating impact on the purpose of the deployment. Continuous monitoring can make any potential issues evident to the end-users, thus helps reducing potential damage or downtime. Monitoring can help to identify unusual activity, which could lead to identifying potential issues. If device resources such as compute, storage, networking, and I / O statistics are monitored, potential problems could be quickly tracked. For example, if the use of CPU in a certain system goes beyond regular usage, the end-user will focus more on this device for any potential abuse. This makes troubleshooting faster, and even end-user can take preventive initiative. Monitoring network activities can also help to identify potential breaches of security.

### 3.3.4 Update

No software is perfect; that is, the software is released with bugs that will later be exposed at some point in its life. Also, the security is taken as a key component by very few developers as many of the developers don't consider security as core value-adding features. In the case of startups who are aiming to get to business fast, this is more true. All these often lead to install insecure devices that are vulnerable to different threats. On the other hand, the device manufacturers make continuous efforts to improve the programs

(firmware) that are responsible for efficiently running the device. In the aforementioned circumstances, a secure update of the device is required. Maintenance of software has a lot of potential stages. There might be a situation where the firmware has a new bug fix available; in that case, all system software, including bootloaders, must be modified through a process. Fixing a bug in the application or adding a new feature to the running application is another stage of update. In that case, the firmware is left untouched, and only the application is updated to save bandwidth of the network. Nonetheless, the remote system software update is a long-term operation, and the type of connection to the remote devices plays an important role. Remote devices may not have a reliable link. The link may not be reliable if its wireless connection and devices are on the move. There may be only a temporary link in the case of cellular connection to save the cost. The most critical part of the update is that it must be achieved without hindering the continuity of the business.

### 3.3.5 Decommissioning

A device can reach the end of its life or be compromised or even physically damaged; it must be removed and dismantled in any case. If not, essential bandwidth can be wasted, and the network might get unnecessarily congested. If the compromised device is not decommissioned, it opens the door to attack the whole network. Even if a device is not properly decommissioned, attackers could make an effort to attack the whole network with the scrapped device's residual. In those perspectives, decommissioning is considered an important stage of the device management lifecycle.

## 3.4 OMA Lightweight M2M(LWM2M)

The Lightweight M2M (LWM2M) protocol [36] has been specified by the OMA SpecWorks to meet the unique needs of constrained IoT devices. The client-server modeled LwM2M is a secure, effective, and deployable protocol for the management of the constrained devices in different networks. It's a light, fast, and structured protocol, ideal for low-capacity devices. It was ratified by OMA SpecWorks on February 15, 2017, and has the potential to accelerate the promise and potential of the Internet of Things (IoT). It has been built on top of a secured data transfer standardized by the Internet Engineering Task Force (IETF), which is known as the Constrained Application Protocol (CoAP) [37]. CoAP is a variant of HTTP [38]. LwM2M uses modern REST-based architecture and has defined an extensible resource and data model, which could be reused. The following figure 3.2 shows how LWM2M Server interacts with the LWM2M clients resided in any IoT devices. The LWM2M client houses one or more LWM2M object instances representing the device being managed. An LWM2M object contains several resources. There might be single or multiple instances for each object or resource. A network environment implemented using LwM2M protocol consists of three types of entities:

- **LwM2M Clients:** The client is implemented on the end devices. They maintain

communication with the server(s). Client expose its resources to the server via data model so that server can manage and monitor end device's resource. An LWM2M Client is identified by Endpoint Client Name - a Uniform Resource Name (**URN**) uniquely assigned to a device by its manufacturer.

- **LwM2M Bootstrap Server:** This is the server which is contacted by every client during its first boot-up or every boot-up. The purpose of this is to provide bootstrapping and security information to LwM2M clients and LwM2M servers.
- **LwM2M Servers:** This talks with the clients and have the ability to read from and write to the data model that is exposed by the clients.

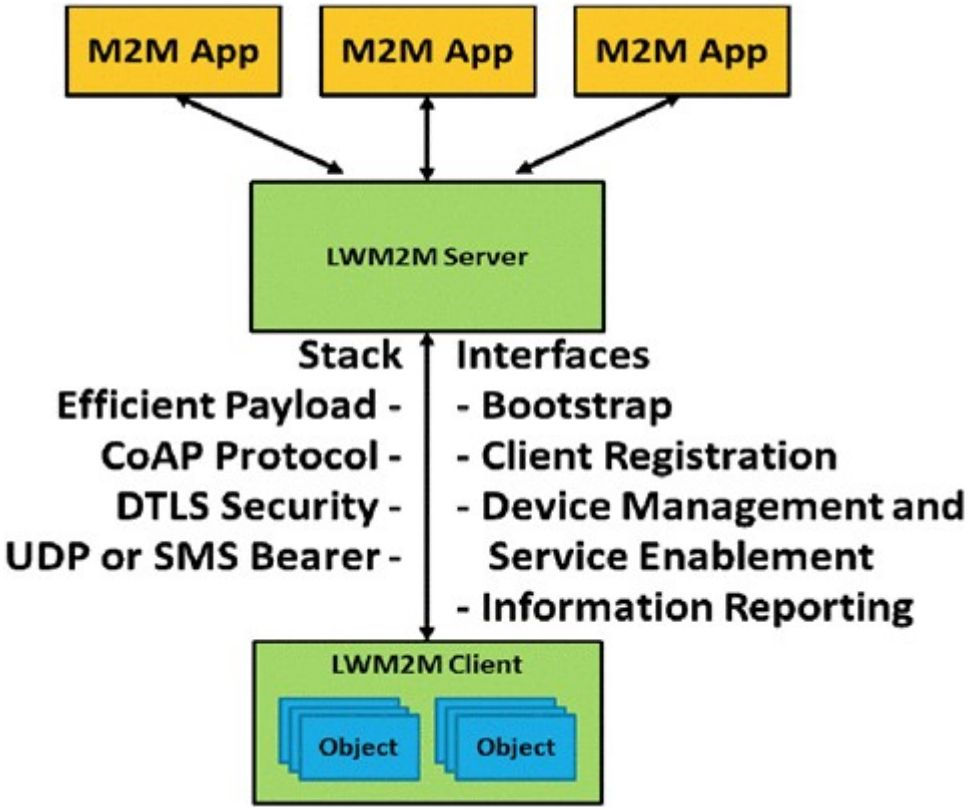


***Figure 3.2.*** *OMA LWM2M high level architecture (source: OMA)*

**Data Models:** Each LwM2M client has a data model which is a standardized, symbolic representation of its configuration and state. The server has read/write privilege on this data model. The data model can be thought of as a combination of a hierarchical configuration file, and a view on statistical information about the device and its environment. The LwM2M data model is very strict and has been organized as a three-level tree. Numerical identifiers are used to identify the entity of those level. Those three levels are briefly discussed below:

- **Object:** The object represents accessible data within the client. For example, the firmware object is defined for performing firmware update in the client. Each object has a unique numerical identifier in the range 0-65535, inclusive. Each object defines a set of resources which has the same meaning throughout all instances of that object. The LwM2M specification defines the object definition with the following

features:

**Name -**Any descriptive name, which is not used in the on-wire protocol.

**Object ID -** A number to identify the object.

**Instance -** Single or Multiple.

**Mandatory -** Mandatory or Optional

**Object URN -** Uniform Resource Name is location independent identifier specified by the authority which is globally unique and persistent over a long period of time. It can be used to identify resources even if it becomes unavailable.

**Resource Definitions -** Used to define resources inside an object. For example, the /0 represent the LwM2M Security object which contains confidential part of information about connections to the LwM2M servers configured in the client.

- **Object Instance:** An object might have one or more instances. Device object which describes the device itself is a single instance object and is identified by 0. Firmware update is also another example of single instance object. The object which represents optional software packages installed on the device is an example of multiple instance object.
- **Resource:** Each instance of a given object supports the same resources, as it is defined by the object definition.Within a given Object, each Resource ID (which may be within the range 0-65535, inclusive) has a well-defined meaning, and represent the identical concept. The numerical identifiers on each of these levels form a path which is used in the CoAP URLs. For example a path a path /1/2/3 refers to followings:
  - Refers to resource ID=3
  - Object instance ID=2
  - Object ID=1

  Whole object instance could be referred by /1/2 or whole object by /1 in this format. The LwM2M specification has specified following fields for any resources:
  - **ID-** A number to identify the object.
  - **Name-** Any descriptive name, which is not used in the on-wire protocol
  - **Operations-** one of:
    - **R-** read-only Resource
    - **W-** write-only Resource
    - **RW-** writable Resource
    - **E-** executable Resource
    - *Empty-* used only in the LwM2M Security Object; denotes a Resource not accessible via the Device Management and Service Enablement Interface

- **Instance-** Single or Multiple.
- **Mandatory-** Mandatory or Optional
- **Type-** Data type of the resource value.
- **Range or Enumeration-** Specifies the valid values for the resource
- **Units-** The units of the numerical values.
- **Description-** A detailed description of the resources.

**Interface:** In LWM2M there are four interfaces through which the clients, servers, and bootstrap servers communicate. These interfaces are as following:

- Bootstrap Interface
- Registration Interface
- Device Management and Service Enablement Interface
- Information Reporting Interface

□ **Bootstrap Interface:** The bootstrap server uses a set of commands to provision the initial configuration onto the client, and the bootstrap interface defines those commands. In this interface, the LWM2M Client act as a CoAP server, and the LWM2M server act as a CoAP client. The following message might exchange between those:

- **POST /bs?ep={Endpoint Client Name**-} The Bootstrap Request is the request sent from the client to the bootstrap server. It informs the bootstrap server that a new client has appeared on the network and requesting bootstrap information. The bootstrap server can also issue bootstrap commands without having any bootstrap request.
- **PUT-** The request coming to the client from the bootstrap server is known as bootstrap write commands. This command is used for creating and writing to object instance and resource to initialize data model to the proper state, which will be used later to communicate with LWM2M server.
- **DELETE-** For deleting an instance of an existing object, the bootstrap server sends a bootstrap delete command to the client. This command is represented as DELETE command.
- **GET-**The bootstrap server use GET request to information about the data model supported by and present in the client. This request is interpreted as a bootstrap discovery. The object instance with some other additional metadata is accessible by the bootstrap server. However, the bootstrap server cannot read any resource values.
- **POST-** In the final stage, the bootstrap server sends a bootstrap finish command to the client. Once the client receives a finish command, it validates its data model, and in case of success, it then connects to the regular LWM2M server as specified by the data model. This command is represented as POST.

The bootstrap interface is mostly write-only and hence the bootstrap server can not do any actual management or monitoring of the client. It only prepare the client to connect to actual management server. However, the bootstrap server and the management server can coexist in the same server.

**Registration Interface:** The protocol the client uses to inform its availability to an LWM2M server is known as a registration interface. In this interface, the LWM2M client act as a CoAP client, and the LWM2M server acts as a CoAP server. Following are the commands used in this interface:

- **Register-** When the client goes online, it uses this command to register itself to the LWM2M Server. It informs the server that it's ready to accept commands from the server. It resembles CoAP **POST /rd?...** request
- **Update-** This is a CoAP POST request on a URL which is sent in response to register. This can be periodical or when ever a register command initiated earlier.
- **De-register-** This is a CoAP DELETE and sent by a client when a client shutting down.

**Device Management and Service Enablement Interface:** This is the interface through which actual device management is done. In this case, the LWM2M server acts as a CoAP client and sends the request to the LWM2M client, which acts as a server on the CoAP layer. Following are the commands defined in the Device Management and Service Enablement interface.

- **Discover (CoAP GET Accept: application/link-format)-** The server use this commands to get list of all supported and present object, object instance, resources and attribute addherent
- **Read-** Which is a CoAP GET commands and reads data of a single resource, entire object instance or a whole object at once.
- **Write-** Using this commands the server modify the data model. This can be of following two:
  - **PUT /{Object ID}/{Instance ID}[/{Resource ID}] :** This is a replace method which might be called on either a single resource to replace its value or on a whole object instance to erase all existing data with the supplied data.
  - **POST /{Object ID}/{Instance ID}:** This format request is used to do partial update. It's called only on a whole object instance to replace the resource which is supplied in the payload and retaining the other data.
- **Execute -** The execute command is represented as POST /{Object ID}/{Instance ID}/{Resource ID} request. It's used to perform an operation like firmware update or rebooting of the device.

**Information Reporting Interface:** This is an extension of the Device Management and Service Enablement Interface. The server uses this interface to receive periodic update

of the certain value of interest in the data model. Following are the commands executed in this interface:

- **Read-** When a client receives a read request, it returns its current value and also prepare itself to notify later if an appropriate notification event happens.
- **Cancel Observation-** This also a read commands with observe option value set to 1.
- **Notify-** If an event of interest happens, client use this to notify the server.

## 3.5 OMA Lightweight M2M Implementations

There have been quite a few different implementations of the LWM2M client and server. The following are the most popular implementations which could be thought of while planning someone's customized implementation of any LWM2M client and server.

### 3.5.1 Eclipse Wakaama

The Eclipse Wakaama project [39], hosted by the Eclipse Foundation is an open-source implementation of the OMA LWM2M protocol written in C language. This project was initially created by three experts in embedded systems coming from Intel, David Navarro, Jacques Bourhis and Hatem Oueslati. The companies supporting the project are Intel, Sierra Wireless and IoTerop. It helps the open-source community to easily adopt a standard-based approach while dealing with Security, Device Management and Interoperability requirements in their products and solutions. The commercial version of Wakaama is IOWA. IOWA is an industrial-grade OMA Lightweight M2M stack for embedded devices, server infrastructures, and gateways. The following features are supported:

- Bootstrap - Client only
- Client Registration - full support
- Information Reporting - Client only
- Data formats
  - Plain Text
  - TLV
- Security
  - External elements could provide the CoAP and DTLS layers.

### 3.5.2 Eclipse Leshan

Leshan [40] provides libraries written in Java. Leveraging those libraries people can develop their own Lightweight M2M server and client. The project also provides a client,

a server and a bootstrap server demonstration as an example of the Leshan API and for testing purposes. Following are the details of the project:

- Eclipse project since 2014
- Modular Java libraries
- Based on Californium CoAP implementation
- Based on Scandium DTLS implementation

Leshan supports similar features as Wakaama.

### 3.5.3 Anjay

Anjay [41] is a Software Development Kit (SDK) that can be used to create an LwM2M client and helps vendors of Internet of Things equipment implement support for OMA SpecWorks' Lightweight M2M. The following features are supported:

- Bootstrap - full support
- Client Registration - full support
- Device Management and Service Enablement - full support
- Information Reporting - full support
- Data formats
  - Plain Text
  - Opaque
  - TLV
  - JSON (output only)
- Security
  - DTLS with Certificates, if supported by backend TLS library
  - DTLS with PSK, if supported by backend TLS library
  - NoSec mode

# 4 FIRMWARE UPDATE

This chapter focuses on the current researches and challenges on the firmware update solutions in sensor networks, which closely resembles the firmware update of BLE nodes, which is the core of this thesis.

## 4.1 Challenges in Firmware Update

While talking about updating firmware in IoT environments, it is obvious to face different kinds of challenges as deployed networks might have heterogeneous devices from different manufactures with their own way of performing the firmware update. The hardware limitations make it even difficult. There have been numerous studies to identify these challenges. Such challenges and requirements are based on recent attempts, such as [42], [43], [44], [45] . In this section, these challenges will be highlighted.

- **Integrity and confidentiality-** The images to be used in the firmware update must be safeguarded and handled only by authorized and legitimate entities. In addition, the information about the libraries and configuration provided. Confidentiality could be ensured by using proper encryption techniques channel protection protocols (e.g., the Datagram Transport Layer Security (DTLS)) in the firmware package could be used by potential attackers.
- **Version control-** An intruder could attempt to install a valid (but old) firmware generated by a legitimate entity. In fact, a single component might be updated several times in its lifecycle. In such a case, IoT appliances should be aware of the version of the firmware installed on it. To handle such situations, there should be a mechanism to manage version information that can be used by devices, manufacturers, and software providers.
- **Management of dependencies-** Updating firmware could affect the performance of other components in a certain device. Since this device is part of a big system, it can affect other devices as well. Such dependencies should be tracked to eliminate any potential damages that might be triggered during the updating process.
- **Update time-** The time for performing the update is very important since during the update process there might be some devices performing a critical task, and the update process might stop this. So there should be a way to find the best time to perform a firmware update. In [45] author has proposed a performance-aware mechanism to find the best time to perform a firmware update.

- **Trust management between manufacturers and software providers-** There might be a situation like that where the firmware update needs to be performed from different entities. There has to be trust between all such entities and manufacturers to achieve such. A Key management mechanism should tackle this dimension to embed the necessary cryptographic material in the device, for validation of new firmware/software images.
- **Continuous Security Assessment-** The new EU 'Cybersecurity Act' [46] regulation is intended to define a framework for certifying cybersecurity. The main objective is to reflect the security level of a certain device or system throughout its lifespan. In this context, updating/patching a particular device might involve a re-certification process to keep its safety level up-to-date.
- **Integration with monitoring approaches-** To detect new vulnerabilities or attacks deployed devices would require a new updating/patching of the devices. The use of a monitoring system should be in place throughout the lifecycle of devices to achieve this.
- **Automated update installation-** The firmware update of the devices should be automated or require minimum human interaction. The integration of recent approaches such as the Manufacturer Usage Description (MUD) [47] can achieve this level of automation. That promotes the automated deployment of devices by restricting their communications.
- **Efficient cryptographic algorithms and security mechanisms-** The updating process needs to provide integrity and confidentiality, and hence the security mechanisms must be based on efficient cryptographic primitives to be used even in resource-constrained deployments. The main objective is to diminish the computational power required for cryptographic operations at the end devices and to reduce the overhead of the messages to be sent over the network.
- **Lightweight representation-** To reduce the size of overhead, the firmware image and metadata should be based on compact approaches. The Concise Binary Object Representation (CBOR) [48] can resolve this issue.
- **Digital twins-** A virtualized copy of the real device could be used to test the impact of the new version before deploying it in the real device. The concept behind this is to determine the effect in terms of performance and security on a certain device.
- **Incremental or total updates-** The incremental update requires less computational effort compared to a whole update. However, device heterogeneity and management of the version make an incremental update difficult. There has to make a choice between the cost of replacing the firmware and the effort it takes to manage its version.
- **Failure management-** A firmware update might fail due to many reasons(e.g., loss of connectivity). In such a situation device should revert back to its earlier working state. Tracking the software version could help to resolve such issues.

- **Decentralized models for sending updates-** Most of the latest firmware image forwarding proposals are based on a typical client-server centralized architecture which might not be the best choice for today's scalable and heterogeneous devices. The use of fog/edge computing could assist in this process to minimize the overall network overhead.

## 4.2 Open Standards for Secure Constrained IoT Firmware Updates

The technical community has been working on open standards [49] over the last few years which could be combined to facilitate IoT firmware updates. The outcome of such efforts could be broadly categorized as follows:

- Cryptographic algorithms;
- Firmware metadata;
- Protocols for transferring updates over the network;
- IoT device management protocols.

### 4.2.1 Cryptographic Algorithms

To guarantee a secure firmware update, the use of state-of-the-art cryptographic algorithms is a must. It was used to believe that the cryptographic algorithms used on the Internet could not be used on constrained IoT devices. Later this proved to be wrong. Elliptic Curve Cryptography (ECC) is typically used because of the smaller key size. The Elliptic Curve Digital Signature Algorithm (ECDSA) for use with the P256r1 curve [50] standardized by the National Institute of Standards and Technology (NIST) is very popular in the industry. The ed25519 [51] is another standardized signature algorithm based on a different curve.

### 4.2.2 Firmware Metadata

A format for defining firmware updates has been standardized by IETF SUIT working group. The SUIT group has defined a manifest that contains information about the firmware required to update the device and a security wrapper which ensures the end-to-end protection of the metadata. The SUIT standardization has three components: i) an architecture documents [52], ii) an information model description [52], and iii) a proposal for a manifest specification [53]. The SUIT was built on top of a number of other open standards that provide the generic building blocks. The Concise Binary Object Representation (CBOR) [48] is such a standard that was used as a data format for serialization. For cryptographically secure data serialization the used standard was Object Signing and Encryption (COSE) [54]. To protect the payload from tampering by an intruder COSE defines a sign structure that uses a cryptographic signature.

### 4.2.3 Standards for Firmware Transport

There has been a number of specification for transferring a firmware update over the network. The basic transport scheme includes a way for transferring Device Firmware Update (DFU) over a specific low-power Media Access Control (MAC) layer technology (such as Bluetooth). The SUIT protocol specified by IETF has combined Constrained Application Protocol (CoAP) over UDP [37] and CoAP over TCP/TLS [55] for transferring firmware through multi hops or over heterogeneous low-power networks. The 6LoWPAN specification was designed to provide a layer of adaptation for networks that can not use IPv6 directly. For providing communication layer security DTLS and TLS [56] profiles were standardized.

# 5 ARCHITECTURE AND DESIGN

By running an appropriate software, a BLE node could either support BLE mesh or 6LowPAN networks. BLE nodes in 6LowPAN networks have IP addresses and can communicate over the Internet through TCP/IP protocols are in one group. On the other hand, BLE nodes in the BLE mesh network do not have the IP address and can't communicate over the Internet. This section discusses the architecture for managing BLE nodes in both types of networks.

## 5.1 Architecture for Managing BLE mesh nodes manually

It is assumed that the BLE nodes will form a mesh or any other type of network among them. By default, nodes of such a system will not have ways to be accessed remotely. An IP-BLE Gateway will be in place to make them accessible remotely. The IP-BLE Gateway will hide the BLE network behind it and provide a way to reach the network behind it remotely. Any authorized user should be able to perform any management tasks on these nodes remotely. Proposed architecture for managing BLE mesh nodes in a BLE mesh network has been shown in the figure 5.1.

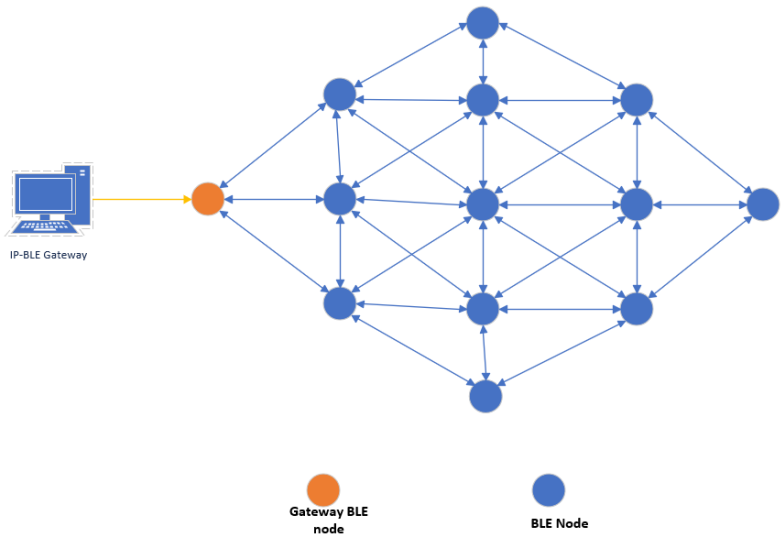


***Figure 5.1.*** *BLE mesh network connected to IP-BLE Gateway*

**IP-BLE Gateway:**The most important part of this architecture is the IP-BLE Gateway. One interface of this Gateway could be connected to the Internet, while the other interface maintains a connection with the mesh network. The BLE nodes of the mesh networks reside behind this Gateway. This Gateway provides a means to remotely manage BLE mesh nodes that would otherwise not be possible.
**Gateway BLE node:** Gateway-BLE-node in the BLE mesh network is a special BLE node. This mesh node maintains a physical connection to the Gateway and, like other ordinary mesh nodes, performs its regular duties in the mesh network. It acts as an interface between the BLE mesh network and IP-BLE Gateway. Any commands it receives from the Gateway through its serial connection, it relays this to its neighboring BLE mesh nodes.
**BLE mesh:** The BLE mesh is the network of low-powered connected devices whose nodes are needed to be managed during their lifecycle.

## 5.2 Architecture for Managing BLE mesh nodes through LWM2M Client

This architecture shown in the figure 5.2 to some extent is similar to the architecture described in section 5.1. The only difference with the previous one is that an LwM2M client resides inside the gateway to facilitate firmware updates. High-level block diagram of such LwM2M client has been presented in the figure 5.2. The objects inside such LwM2M client has been shown in figure 5.3.

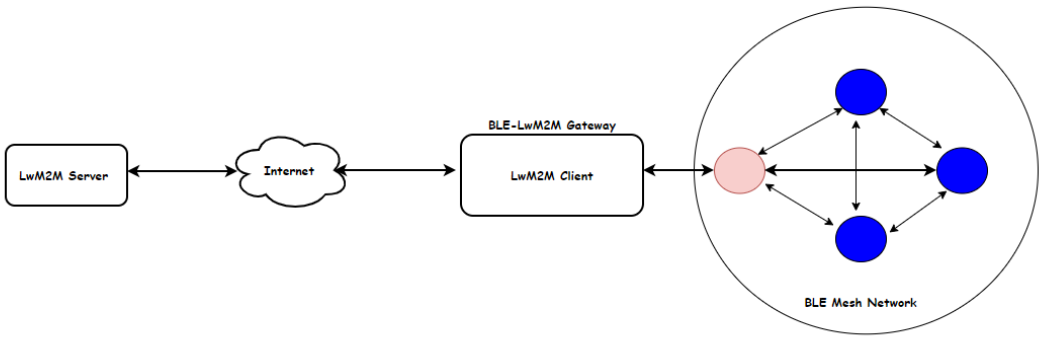


***Figure 5.2.*** *Managing BLE Mesh nodes via LwM2M Client.*

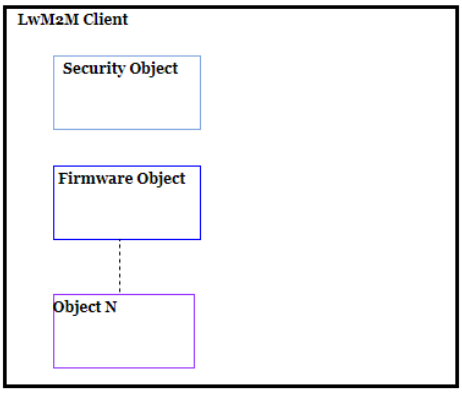


***Figure 5.3.*** *LwM2M Clients with objects.*

## 5.3 Architecture for Managing 6LoWPAN BLE nodes

The figure 5.4 shows the architecture for direct management of IP-based BLE nodes in low-powered networks (6LoWPAN).

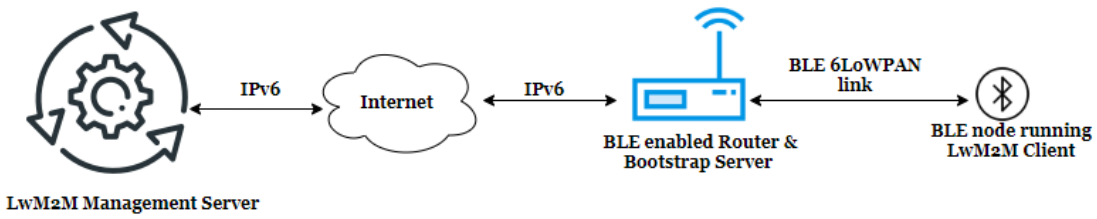


***Figure 5.4.*** *Managing 6LoWPAN BLE nodes*

**LwM2M Client:** In this architecture, every BLE node will run an LwM2M client inside them. The LwM2M will also have the necessary objects implemented as shown in the figure 4.3. The remote management server could manipulate these objects. When the BLE node boots up, it runs the LwM2M clients inside it and contacts the configured bootstrap server. The bootstrap server contains the data needed to join the management server and provides these data to the BLE node. The BLE node registers itself to the management server with the data obtained from the bootstrap server via the BLE enabled router. The communication with the bootstrap server and BLE enabled router should hapen over the BLE link.

**BLE Enabled Router:** This router maintains a wireless connection over the BLE link with a BLE node, and other interface helps to reach interface.

**LwM2M Management Server:** This is the main component that will be used to manage the BLE nodes. Every BLE nodes register itself to this server. This server will provide an interface that could be used to manage all registered BLE nodes.

# 6 IMPLEMENTATION AND TESTING

This chapter intends to explain how the architectures for managing BLE nodes in low powered networks has been implemented. The devices and tools that were used during implementation also have been introduced in this chapter. This chapter also covers the testing of the implementations.

## 6.1 Managing BLE nodes in BLE mesh network manually

Under this section, a brief discussion will be held about implementing the various components of the proposed architecture to manage BLE nodes in any BLE mesh network manually. The firmware upgrade process on BLE nodes is conducted as proof of concept. A failed upgrade case followed by a successful one is documented here.

### 6.1.1 Implementation of the different components

**IP-BLE Gateway:** The implemented Gateway is an x86-based Linux laptop, that could be IP-enabled either using Ethernet or over WiFi. The Gateway was also natively connected to the BLE mesh by having a serial interface with a BLE-based developer board. The initially implemented IP-BLE Gateway was a physical laptop with the following specification:

Model : Lenovo ThinkPad W530 2447CP4
Linux kernel version : 4.15.0-43-generic
Hardware : x86_64
Memory : 16 GiB
Disk Size : 465GiB (500GB)
Processor : Intel(R) Core(TM) i7-3720QM CPU @ 2.60GHz

In other device management scenarios where it might need rapid deployment, an ISO image of the Gateway was created, which could be deployed on the fly as a docker container. This dockerized version of the Gateway is more secure since it runs in an isolated environment. The BLE-LwM2M Gateway is the critical component of this kind of proposed system. The Gateway runs virtual copies of the end BLE nodes, which have been mapped to different physical BLE end nodes.

**BLE mesh Network:** The BLE mesh has been formed with the nRF52832 and nRF52840

boards from Nordic Semiconductor. The details of the used boards are available in table 6.1, and figures 6.3 and 6.4 show such boards. The mesh network had 16 BLE nodes of both types of boards; one of them was physically connected to the gateway and also a member of the mesh network. All other boards are normal mesh nodes. For creating BLE mesh, nRF5 SDK for Mesh v2.2.0 was used. While creating a BLE mesh network that is capable of performing Device Firmware Upgrade (DFU), a few essential things should take into consideration. First of them is the Softdevice, which is a high-performance Bluetooth 5 qualified protocol stack for nRF52832. In this project Softdevice, s132_6.0.0 was used, which is capable of maintaining 20 concurrent links. Next important one is the bootloader. Nordic has specific bootloaders for specific tasks. For setting up BLE mesh, two different kinds of bootloader were considered. The serial_mesh_bootloader was flashed in the device that is directly connected to the gateway. This bootloader makes a device capable of accepting firmware over the serial interface and simultaneously forwarding it to the neighboring BLE mesh nodes over BLE links. All other nodes were flashed with mesh_bootloader.
The DFU over mesh also requires application-level support. For this reason, an application that has DFU support was flashed. The final stage for preparing a device ready for mesh DFU is the flashing a device page. All DFU-enabled mesh devices require a device page that contains information about the device and the firmware that is installed on the device. For the development, programming, and debugging of Nordic Semiconductor's nRF52 boards the nRF Command Line Tool [57] was used.
The figure 6.1 shows the list of commands that were used to program the Gateway Mesh node whereas the figure 6.2 shows the commands used to program any other mesh nodes in the mesh network.

```
nrfjprog --eraseall
nrfjprog --program /home/iot/NORDIC/nrf5_SDK_for_Mesh_v2.2.0_src/bin/softdevice/s132_nrf52_6.0.0_softdevice.hex
nrfjprog --program /home/iot/NORDIC/nrf5_SDK_for_Mesh_v2.2.0_src/bin/bootloader/gccarmemb/mesh_bootloader_serial_gccarmemb_nrf52832_xxAA.hex
nrfjprog --program /home/iot/NORDIC/nrf5_SDK_for_Mesh_v2.2.0_src/examples/dfu/build/dfu_nrf52832_xxAA_s132_6.0.0_Debug/dfu_nrf52832_xxAA_s132_6.0.0.hex
nrfjprog --program /home/iot/NORDIC/nrf5_SDK_for_Mesh_v2.2.0_src/tools/dfu/bin/device_page_nrf52832_xxAA_s132_6.0.0.hex
nrfjprog --reset
```

***Figure 6.1.*** *Commands to Prepare Gateway Mesh node*

```
nrfjprog --eraseall
nrfjprog --program /home/iot/NORDIC/nrf5_SDK_for_Mesh_v2.2.0_src/bin/softdevice/s132_nrf52_6.0.0_softdevice.hex
nrfjprog --program /home/iot/NORDIC/nrf5_SDK_for_Mesh_v2.2.0_src/bin/bootloader/gccarmemb/mesh_bootloader_gccarmemb_nrf52832_xxAA.hex
nrfjprog --program /home/iot/NORDIC/nrf5_SDK_for_Mesh_v2.2.0_src/examples/dfu/build/dfu_nrf52832_xxAA_s132_6.0.0_Debug/dfu_nrf52832_xxAA_s132_6.0.0.hex
nrfjprog --program /home/iot/NORDIC/nrf5_SDK_for_Mesh_v2.2.0_src/tools/dfu/bin/device_page_nrf52832_xxAA_s132_6.0.0.hex
nrfjprog --reset
```

***Figure 6.2.*** *Commands to prepare a general Mesh node*

**Provisioner:** Nordic Semiconductor has implemented a proprietary mesh network based on BLE. A node first needs to complete the provisioning to be a member of the mesh network. There is a couple of provisioning [58] of options available for BLE mesh nodes. These are:

- Over the advertising and GATT bearer
- Remote provisioning over relaying nodes

| Manufacturer | Device | Radio Board | Flash | RAM(kB) | MCU |
|---|---|---|---|---|---|
| Nordic Semiconductor | nRF52832 | PCA10040 | 512kB | 64 | ARM® Cortex®-M4 32-bit processor with FPU, 64 MHz |
| Nordic Semiconductor | nRF52840 | PCA10056 | 1MB Flash with cach | 256 | ARM® Cortex®-M4 32-bit processor with FPU, 64 MHz |

***Table 6.1.*** *Technical specifications of the BLE development boards used in the formation of the BLE mesh*

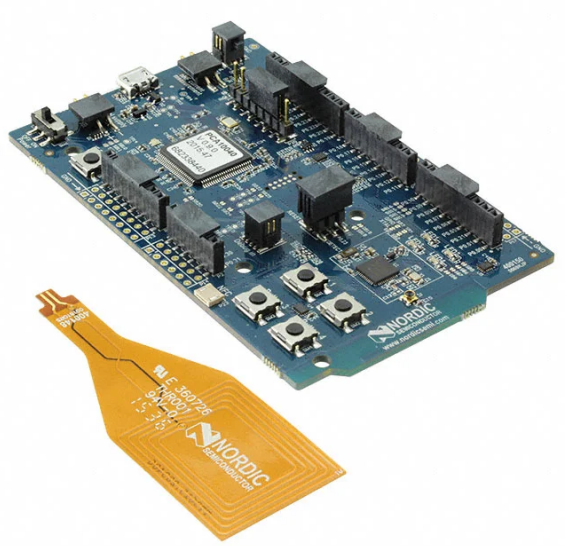

***Figure 6.3.*** *NRF52832 Board by Nordic Semiconductor*

The first method is used when the Provisioner and node to be provisioned are in the direct radio range. The second provisioning option is used to provision a node that is not in the direct radio range of the Provisioner. In this method, other nodes are used to carry provisioning data to the end nodes. The provisioner used in this work was an Android mobile app designed on the basis of the GATT bearer. This type of provisioner requires the device to be provisioned to be in the direct radio range of the provisioner.

### 6.1.2 Firmware Update in Nordic Boards

The firmware update is the most frequent device management task in any IoT node. In this work, the firmware update was considered for demonstrating device management capability in any BLE nodes. Knowing the device's memory layout is important for under-

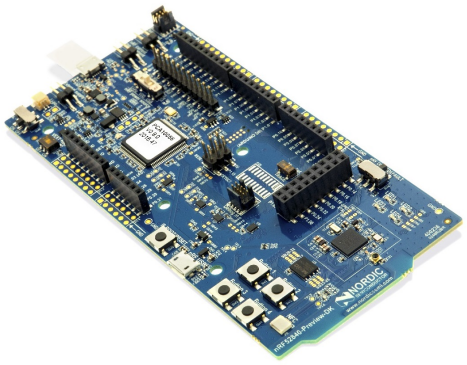

***Figure 6.4.*** *nRF52840 Board by Nordic Semiconductor*

standing the firmware update on that device. Figure 6.5 shows the default memory layout for nRF52 devices, where nRF52832 has a flash size of 512 Kilobytes and, nRF52840 has a flash size of 1024 Kilobytes.

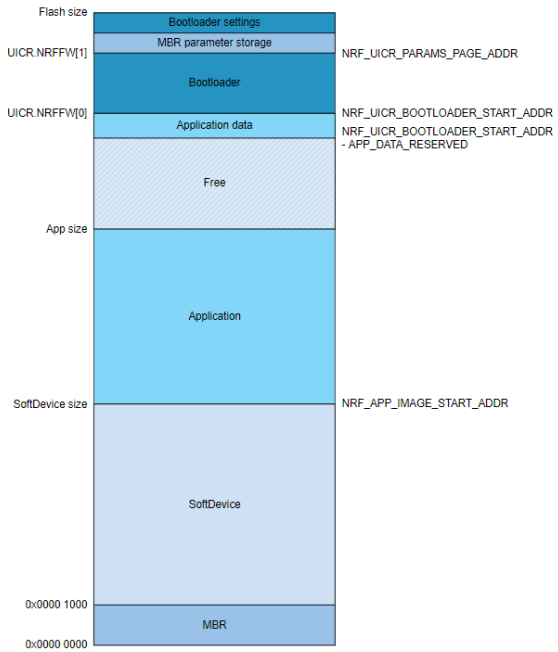


***Figure 6.5.*** *Default memory layout for nRF52 devices[59]*

**Bootloader:** The bootloader module is responsible for:

- booting into an application,
- activating new firmware
- optionally, entering DFU mode where DFU transports are activated and new firmware can be delivered,
- feeding the watchdog timer.

**Bootloader Settings page:** The information about the bootloader and the Device Firmware Update (DFU) stored in non-volatile memory is known as a page. The Settings page includes information on:

- current firmware - size, CRC-32
- pending firmware - size, CRC-32,
- progress of the firmware update,
- progress of the firmware activation,
- current firmware versions (application and bootloader),
- transport-specific data.

**Firmware activation:** The final step of the firmware update process is the firmware activation which is activated based on the information in the settings page. This involves copying the new firmware in place of the existing one and updating the setting page so that new firmware could boot properly. The new firmware could be copied in two alternative ways which are Single-bank updates and Dual-bank updates. In a single-bank update, the running application is overwritten with the new firmware image. If an error occurs during the update, the device will be left without a valid application, the system will return to DFU mode and from that state a new update could be started again. A Dual-bank update is possible only if there is enough free space between the end of the current application and the beginning of the application data to store the new firmware image. In this process, the new firmware image is not copied to the final location without validating the new firmware which ensures that only complete and valid images are activated. During an error, the firmware is not updated and, the old firmware is still available. In this work, the firmware update was done through a dual-bank update.
**DFU mode:** In DFU mode, DFU transport is activated by the bootloader and, the device gets ready to receive new firmware. A device enters into the DFU mode on the following conditions:

- No valid application is present.
- SoftDevice has been activated and a valid application is present. In that case, the bootloader expects that an application update may be requested by the host.
- Entering DFU mode is triggered by one of the optional sources:
  - Button
  - Pin reset
  - Special value in register
  - Request from the application written to the settings page

  In this work, the last option was used.

**Starting the application:** To run the application, the bootloader must know the location of the application. The bootloader determines the location of the application based on the information stored in the setting page. Before starting the application bootloader checks

the integrity of the application. If the integrity check fails or if there is no settings page, the bootloader enters DFU mode.

### 6.1.3 Firmware Update in the Mesh Network:

Firmware maintenance is the most common task of device lifecycle management. As proof of the concept, the firmware update was conducted in the mesh network. The proprietary firmware update solution by nordic semiconductor updates the firmware of the whole mesh simultaneously.
For having better control over the firmware update in the mesh, updating a specific group of the node while leaving others as it is, in this thesis a couple of alternative possibilities have been explored. Both of them use the proprietary mesh protocol for distributing the firmware update package but instead of updating the firmware of the whole mesh network, it only updates targeted nodes in the mesh.

- **Application ID based controlled firmware Update:** Any firmware update operation by Nordic's proprietary protocol is commenced by uploading an init packet which contains the metadata about the firmware package; in other words, it is called manifest. The manifest is a JSON file containing different information about the firmware upgrade package. During the firmware upgrade operation, these fields are checked strictly. The firmware update process starts only after performing the required checking of the manifest file. The two crucial information fields in the manifest file are:
  - **application_id,** used to identify any application type uniquely. If the running application ID matches the application ID of the upgrade, only then firmware update could be performed on that particular node filling other firmware upgrade conditions.
  - **application_version,** denotes the version of the firmware. Once the running application ID is matched with the application ID of the upgrade, it then checks the application version. If the version of the application in the upgrade supersede from the running version of the application on a particular node, upgrade could be applied on that node. Otherwise, firmware upgrade operations will fail.

  Application_id could be used to perform group-based firmware update. For achieving this, the devices need to be programmed with different application IDs. Let's imagine a big mesh covering multiple floors of the building. It could be programmed mesh nodes on the first floor with application id = 1 and for second floor application id = 2 and so on. Now let's say that in the upgrade package, which is supposed to replace the running firmware, application id is set to 1. If the firmware upgrade is performed with this package in the mesh, instead of upgrading the whole mesh, it will upgrade all nodes on the first floor only. Similarly, the firmware update could be

performed only for the second or third floor.

- **Controlled firmware update based on Signature Verification** The idea behind this is a Public-Private key pair. This key pair is generated together and has a strong relationship among them. The private is the secret one in this key pair and not shared with others, whereas the public key is public and can be shared with anyone. A firmware package is signed by the private key. Any node having the corresponding public key can verify this signature. The proprietary mesh protocol by Nordic Semiconductor has the feature that every node verify the signature of the firmware before applying the update on that node. This feature could be utilized to perform the firmware update on a group of mesh nodes instead of the whole mesh network. This has been depicted in figure 6.6.

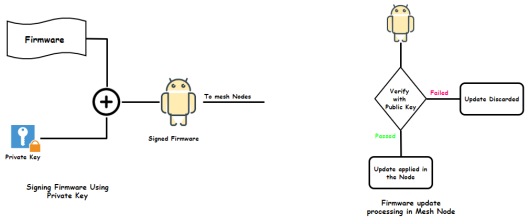


***Figure 6.6.*** *Signing and verification of the Signature in Mesh Node*

### 6.1.4 Performing Firmware Update Manually

For performing a firmware update manually in the Mesh network, the user will log in to the IP-BLE Gateway remotely though SSH and perform update commands on the terminal. The node which is connected to the IP-BLE Gateway through the serial connection will receive the firmware update package and forward it to other neighboring mesh nodes over the BLE links. In this implementation, a firmware which blinks the LED of the BLE mesh nodes has been used to demonstrate the firmware update. Initially, the mesh node contains an application that keeps the LED on the board always on. This application has been replaced with the firmware upgrade package. This new firmware blinks the LED instead of keeping it always on. Transferring of the firmware update package has been shown in the figure 6.7 and 6.8.

After successful transfer node took a while to apply new firmware and then started blinking the LED on the board, which confirms the successful firmware update.

### 6.1.5 Performing Firmware Update using LWM2M Client

The second way uses an LWM2M client-server model to perform the firmware update in the mesh network. In this procedure, instead of performing firmware updates manually, a firmware update is done through an LwM2M client. Within the firmware object, all

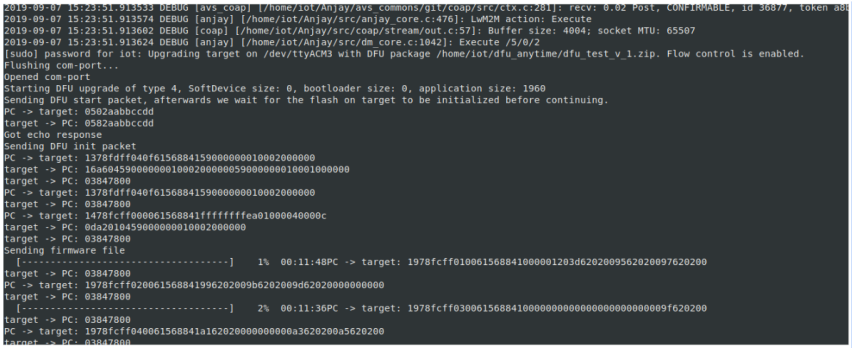


***Figure 6.7.*** *Firmware Upgrade triggered and it's on progress*

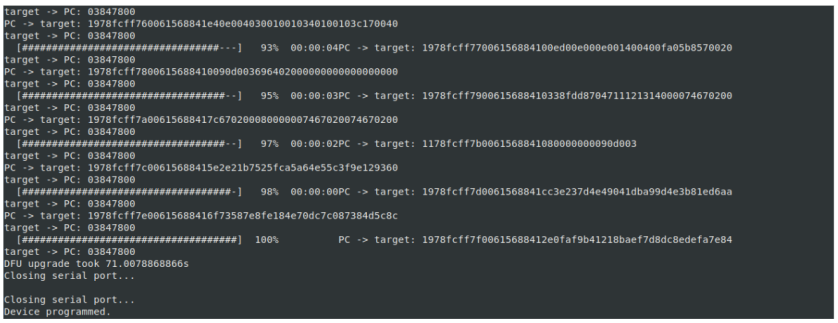


***Figure 6.8.*** *Firmware upgrade has been executed successfully.*

the firmware update commands have been integrated. The idea behind this is that this LwM2M client will register itself to the remote LwM2M management server and expose the resource inside it. Through the interface of the LWM2M server, the user can interact with any object of the LWM2M client and perform different read/write and execute operations on its resources. When the user triggers a firmware update request by clicking the on "Exec" button from the LwM2M client interface, all the integrated firmware update commands get executed. In that case, the LWM2M client has been implemented using the Anjay Software Development Kit (SDK) listed in section 3.5.3. The firmware object was implemented with the aid of this SDK. The programming language used for implementing firmware the object was C. The figure 6.9 shows the different components of the implemented firmware object. The firmware_update.h is the header file and firmware_update.c


***Figure 6.9.*** *Components of Firmware Object*

contains the detail implementation code for the firmware object. Figure 6.10 shows such an LWM2M client registered in the LWM2M. If registered client's name is clicked from the interface of the LWM2M server, it shows its resources. Figure 6.11 shows the interface of our implemented firmware object in the LWM2M server.

Through the interface of the LWM2M server, the user can interact with any object of the LWM2M client and perform different read/write and execute operations on its resources.

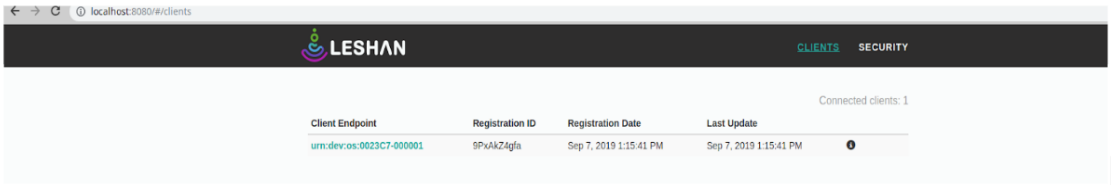


***Figure 6.10.*** *LWM2M Server with one LWM2M client*

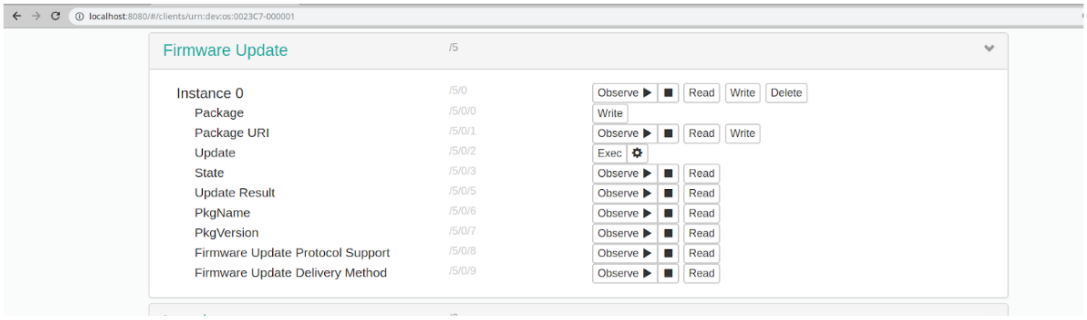


***Figure 6.11.*** *Interface of the LWM2M firmware Object*

The picture 6.12 below shows the outcome of a read operation performed on the object. It's required to click on the "Exec" button of the object to perform the firmware update. If


***Figure 6.12.*** *Read operation performed on firmware object*

an end node is connected to the Gateway through the serial connection, it will immediately trigger a firmware update of that node. In case of failure, it will report the probable reasons, in the log of the LWM2M server. To demonstrate a failure scenario, the physical node that was connected to Gateway through a serial connection was removed. After that, when the "Exec" button is clicked from the LWM2M server, it fails to perform the firmware update, and it reports the probable reasons for failure. LWM2M server's logs and firmware object's response have been added here from such a demonstration. Figure 6.13 is the log taken from the LWM2M server, reporting firmware update failure. Now, a success scenario will be demonstrated. For that, it needs to be sure that the end BLE device is physically connected to the gateway through a serial connection, and proper decryption key has been programmed to decrypt the encrypted firmware. Finally, the firmware upgrade package should have a superseding version number and encrypted

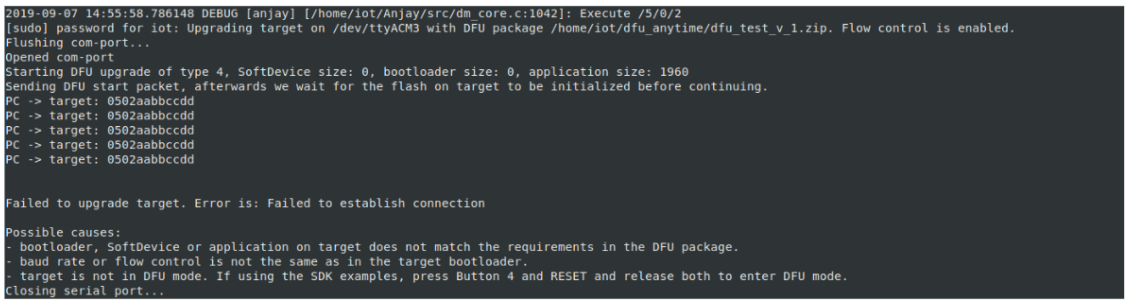


***Figure 6.13.*** *LWM2M Server logs giving hints for failure*

with the respective private key. The following figures 6.14 and 6.15 show successful firmware upgrade.

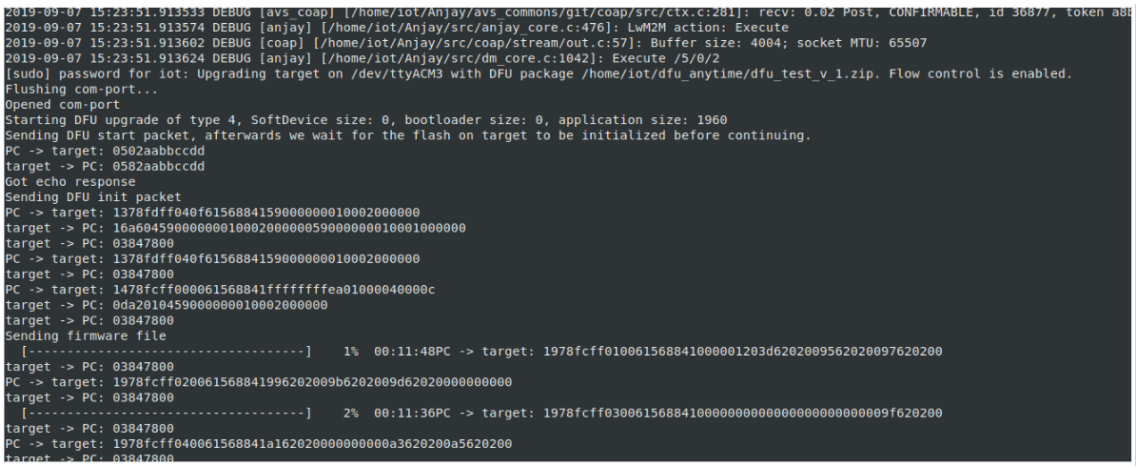


***Figure 6.14.*** *Firmware Upgrade triggered and it's on progress*

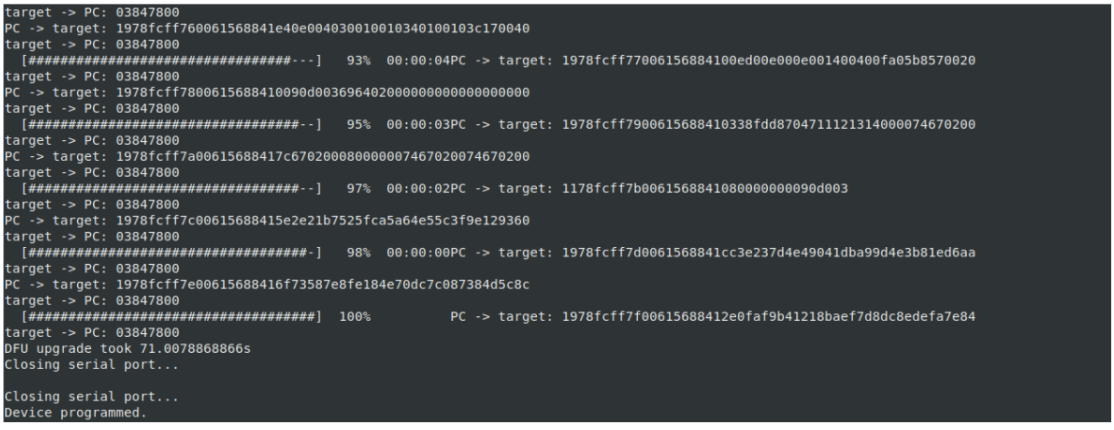


***Figure 6.15.*** *Firmware upgrade has been executed successfully.*

## 6.2 Management of BLE nodes in 6LoWPAN network

This section demonstrates the firmware update in 6LoWPAN nodes over the BLE link.

### 6.2.1 Implementation of the different components

- **BLE Enabled Router:** BLE enabled router is the most important part of such a management system. It has an IP enabled interface connected to the Internet and it also maintains a BLE connection with the 6LowPAN nodes which is running an LwM2M client. In this implementation, a laptop having Ubuntu 18.04.3 LTS was used as BLE enabled router. Commands used to enable BLE in the ubuntu has been shown in the figure 6.16.

```
mount -t debugfs none /sys/kernel/debug
modprobe bluetooth_6lowpan
echo 1 > /sys/kernel/debug/bluetooth/6lowpan_enable
hciconfig
hciconfig hci0 reset
hcitool lescan
echo "connect 00:AA:BB:XX:YY:ZZ 1" > /sys/kernel/debug/bluetooth/6lowpan_control
```

***Figure 6.16.*** *Commands to Enable BLE in Ubuntu*

- **Leshan Server in Ubuntu:** In this work, the LWM2M server has been implemented in a ubuntu system by the libraries provided by Leshan[40]. Both the Leshan bootstrap server and the LWM2M server run on the same computer in this design to keep the setup simple. It is also possible to setup Leshan bootstrap and the LWM2M server in separate computers which can reside in different physical locations. Figure 6.17 shows an interface of the Leshan server.


***Figure 6.17.*** *Leshan Server waiting for LWM2M Clients to be connected*

- **Bootstrap Server:** The bootstrap server also has been implemented in the same ubuntu system. Once each LwM2M client boots up, it first comes into contact with the Boostrap server. The configuration for the respective client should be in the Boostrap server. The configuration is done through a JSON file. The following figure 6.18 has shown the configuration of the bootstrap server. In this configuration file, the DTLS security configuration has been added to secure CoAP communication. The management Server can perform Read/Write and Execute commands on the client's resources exposed to the management server.
- **LwM2M Client:** In this implementation the LwM2M clients have been implemented

```
{
    "servers": {
        "1": {
            "shortId": "1"
        }
    },
    "security": {
        "0": {
            "uri": "coaps://[2001:db8::1]:5684/",
            "bootstrapServer": true,
            "securityMode": "PSK",
            "serverPublicKeyOrId": [105, 100, 101, 110, 116, 105, 116, 121, 48],
            "publicKeyOrId": [105, 100, 101, 110, 116, 105, 116, 121, 48],
            "secretKey": [116, 111, 112, 115, 101, 99, 114, 101, 116, 48],
            "serverId": "0"
        },
        "1": {
            "uri": "coaps://[2001:db8::2]:5684/",
            "securityMode": "PSK",
            "serverPublicKeyOrId": [105, 100, 101, 110, 116, 105, 116, 121],
            "publicKeyOrId": [105, 100, 101, 110, 116, 105, 116, 121],
            "secretKey": [116, 111, 112, 115, 101, 99, 114, 101, 116],
            "serverId": "1"
        }
    }
}
```


***Figure 6.18.*** *Bootstrap Server Configuration for a Client*

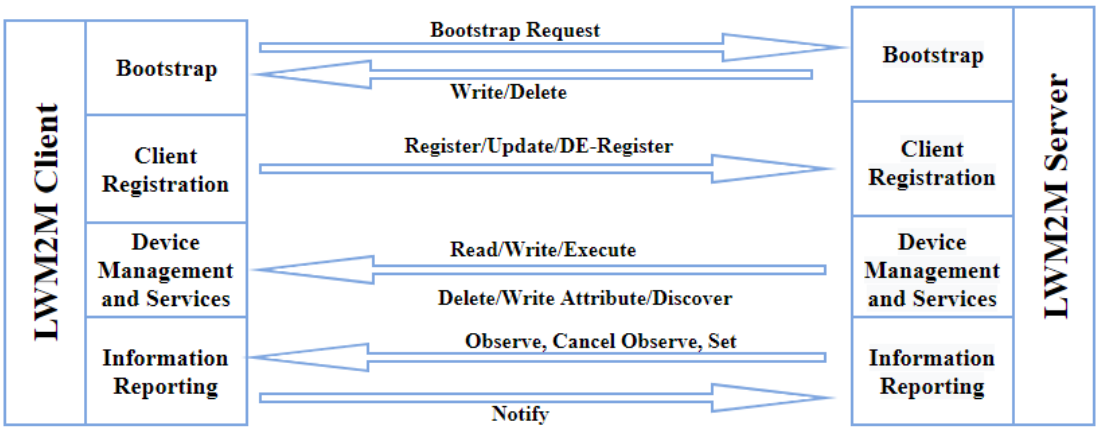


***Figure 6.19.*** *Server Client Interaction*

using the nRF5 SDK v15.0.0 provided by the Nordic Semiconductor. It provides drivers, libraries, examples, and APIs for starting with the Internet of Things (IoT). It also has the LwM2M client with basic objects as. In this thesis, for the management purpose of the BLE nodes, a firmware object has been added with earlier existed basic objects. In addition, the configuration of the LwM2M clients has been customized so that it can contact and authenticate successfully to the bootstrap server and the management server.The nordic boards shown in figure 6.3 and 6.4 were flashed with such an implemented LWM2M client. The interaction between such LwM2M clients and Leshan Server has been shown in figure 6.19.

### 6.2.2 Firmware Update

Performing the firmware update is done from the LwM2M management server. Once the LwM2M client registers itself to the management server, it exposes its resources. The management server could manipulate these resources. The figure 6.20 shows the resources of the implemented firmware object of the LwM2M client. The end-user should click in the "Execute" as shown in the figure to perform firmware updates in the BLE node. Once the "Execute" button is clicked, the firmware update should initiate. Once the "Execute" button is clicked, the firmware update should initiate. Nonetheless, a real firmware update has not been checked in this work; instead, a firmware update mock-up has been performed. In this mock-up firmware update has been repressed by lighting the LED in the BLE node.


***Figure 6.20.*** *Resources of the firmware object.*

# 7 RESULT AND DISCUSSION

The overall research work has been discussed in this section. It involves, in particular,reflecting on the research questions raised in chapter one. There's also been a discussion about the limitation of this work. The section ends by providing some suggestions about future work in this field.

## 7.1 Reflections of the research questions

This thesis focused on implementing these possibilities and conducting a firmware update on BLE nodes to demonstrate the capabilities of device management through these systems. In this thesis the 4$^{th}$ and 5$^{th}$ research questions in section 1.1 were answered as follows:

1. *What are the ways to perform device management on BLE nodes?*

Various device management possibilities for BLE nodes have been explored to address the question. BLE nodes support BLE mesh networking or 6LoWPAN with a suitable program running therein. There are a couple of ways to manage BLE nodes in BLE mesh networks. The first one is manual, where a user remotely login into the remote Gateway and performs the management tasks (i.e. update firmware) on BLE mesh nodes by executing management commands manually.
The second way of managing BLE nodes in the BLE mesh network is done via an LwM2M client. The LwM2M client has been implemented by the SDK supplied by Nordic Semiconductor. The implemented LwM2M client resides within the Gateway, and the management commands needed to update the firmware are embedded inside the LwM2M client. The LwM2M client registers itself to the LwM2M server, and the firmware update could be done from the interface of the LwM2M server.

2. *Is it possible to use LwM2M Client for managing BLE nodes in 6loWPAN network?*

This question was focused on the management of the BLE nodes in the 6LoWPAN network. To address this question, in this work a firmware object has been implemented with the SDK provided by Anjay. This object has been placed inside the LwM2M client, and the LwM2M resides inside the BLE nodes. The 6LoWPAN nodes connect to the BLE router over BLE links and expose their resources to the LwM2M server. The management tasks could be done from the interface of the LwM2M server. Three different possibilities have been explored to answer the 4$^{th}$ and 5$^{th}$ research questions. Two of them are for man-

aging BLE nodes in the BLE mesh networks. Updating firmware using these techniques do not depend on the size of the BLE mesh network. Among these two, the second one seems more pragmatic since that requires less human interactions and time compared to the first approach. The third technique provides the ability to perform device management per node easily which is very difficult to achieve in BLE mesh networks. However, device management effort using such techniques depends on the number of BLE nodes. Overall, according to research questions, the outcome of this thesis work generally succeeded in achieving two objectives. From the perspective of BLE node management, this work has demonstrated a few possibilities which can facilitate BLE nodes management in BLE mesh networks and 6LowPAN networks.

## 7.2 Limitations

The first limitation of this work is experimenting with only the device firmware update. Throughout the thesis, it has been repeatedly talked about device management, however, in this work, the focus was given only on device firmware update instead of all other device management tasks. Though this thesis has talked about overall device management, it has tested only a firmware update. The second limitation comes from the BLE node management in the BLE mesh network. In this system, a BLE mesh node maintains a physical connection with Gateway which is the only one node that lets other mesh nodes get the firmware from it. This management system has a single point of failure. If this Gateway node somehow gets corrupted or loses the connection with the Gateway then it will not be possible to perform any management tasks on this mesh network anymore. This thesis has not addressed any ways to overcome such situations. The third limitation of this study comes from the device management of the BLE nodes in 6LoWPAN networks via LwM2M client. In this implementation, instead of performing real a firmware update, a firmware update mockup was done. Performing a real firmware update would require the implementation of a transportation protocol that would transport the firmware from the LwM2M server to the LwM2M client. In this firmware update mockup, the LwM2M client changed its LED to reflect a successful firmware update instead of performing a real firmware update.

## 7.3 Future Study

The developed solution, as part of this work, has enabled the management of IP based and non-IP BLE nodes securely by remote users. But there has not been any comparative studies among the proposed solution and other solutions. A qualitative and quantitative comparison of these management systems would be an interesting future study. The memory footprint of the different alternative solutions and time consumption during the firmware update of the identical size of the firmware would be another interesting study. In this work, the firmware package was placed in the Gateway, transferring the firmware securely to the end node could be another topic of further study.

# 8 CONCLUSION

In this thesis work, three different possibilities for managing BLE nodes have been implemented. The firmware update in BLE nodes in three different scenarios has been demonstrated using these implemented solutions to show the device management capabilities of BLE nodes in different BLE networks. It started with exploring different low powered communication protocols, which helped to understand different BLE networks. Later there has been a study to understand the device management challenges. Based on this ground, three management solutions have been implemented. Two of the implemented solutions were dedicated for performing device management in BLE mesh networks. The first one helped to perform the device management manually by a remote user logging into the Gateway. The second approached used an LwM2M client in the Gateway and automated the manual steps in the first approach. The third approach was for managing BLE nodes in the 6LoWPAN network. The test results achieved from these implemented solutions suggest that these approaches are very feasible ways for performing device management in BLE nodes in different networks.

# REFERENCES


[1] Dian, F. J., Yousefi, A. and Somaratne, K. A study in accuracy of time synchronization of BLE devices using connection-based event. *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE. 2017, 595–601.

[2] Somaratne, K., Dian, F. J. and Yousefi, A. Accuracy analysis of time synchronization using current consumption pattern of BLE devices. *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2018, 841–844.

[3] Dian, F. J., Yousefi, A. and Somaratne, K. Performance evaluation of time synchronization using current consumption pattern of BLE devices. *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2018, 906–910.

[4] Lin, J.-R., Talty, T. and Tonguz, O. K. On the potential of bluetooth low energy technology for vehicular applications. *IEEE Communications Magazine* 53.1 (2015), 267–275.

[5] Lin, Z.-M., Chang, C.-H., Chou, N.-K. and Lin, Y.-H. Bluetooth Low Energy (BLE) based blood pressure monitoring system. *2014 International Conference on Intelligent Green Building and Smart Grid (IGBSG)*. IEEE. 2014, 1–4.

[6] Hossen, M., Kabir, A., Khan, R. H., Azfar, A. et al. Interconnection between 802.15. 4 devices and IPv6: implications and existing approaches. *arXiv preprint arXiv:1002.1146* (2010).

[7] Darroudi, S. M. and Gomez, C. Bluetooth low energy mesh networks: A survey. *Sensors* 17.7 (2017), 1467.

[8] Kushalnagar, N., Montenegro, G., Schumacher, C. et al. IPv6 over low-power wireless personal area networks (6LoWPANs): overview, assumptions, problem statement, and goals. (2007).

[9] Al-Sarawi, S., Anbar, M., Alieyan, K. and Alzubaidi, M. Internet of Things (IoT) communication protocols: Review. *2017 8th International Conference on Information Technology (ICIT)*. May 2017, 685–690. DOI: 10.1109/ICITECH.2017.8079928.

[10] Adams, J. T. An introduction to IEEE STD 802.15. 4. *2006 IEEE Aerospace Conference*. IEEE. 2006, 8–pp.

[11] Specification, Z. Zigbee standards organization. *Document 053474r17, Jan* 17 (2008), 26.

[12] Samie, F., Bauer, L. and Henkel, J. IoT technologies for embedded computing: A survey. *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. ACM. 2016, 8.

[13] Salman, T. and Jain, R. Networking protocols and standards for internet of things. *Internet of Things and Data Analytics Handbook (2015)* 7 (2015).

[14] Song, J. and Tan, Y. K. Energy consumption analysis of ZigBee-based energy harvesting wireless sensor networks. *2012 IEEE International Conference on Communication Systems (ICCS)*. Nov. 2012, 468–472. DOI: 10.1109/ICCS.2012.6406192.

[15] Qadir, Q. M., Rashid, T. A., Al-Salihi, N. K., Ismael, B., Kist, A. A. and Zhang, Z. Low power wide area networks: a survey of enabling technologies, applications and interoperability needs. *IEEE Access* 6 (2018), 77454–77473.

[16] Prabh, K. S., Royo, F., Tennina, S. and Olivares, T. A MAC protocol for reliable communication in low power body area networks. *Journal of Systems Architecture* 66 (2016), 1–13.

[17] Ahmed, N., Rahman, H. and Hussain, M. I. A comparison of 802.11 ah and 802.15. 4 for IoT. *ICT Express* 2.3 (2016), 100–102.

[18] Mulligan, G. The 6LoWPAN architecture. *Proceedings of the 4th workshop on Embedded networked sensors*. ACM. 2007, 78–82.

[19] Lu, C.-W., Li, S.-C. and Wu, Q. Interconnecting ZigBee and 6LoWPAN wireless sensor networks for smart grid applications. *2011 Fifth International Conference on Sensing Technology*. IEEE. 2011, 267–272.

[20] Gomez, C., Oller, J. and Paradells, J. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors* 12.9 (2012), 11734–11753.

[21] Haartsen, J. C. Bluetooth radio system. *Wiley Encyclopedia of Telecommunications* (2003).

[22] Mikhaylov, K. and Tervonen, J. Multihop data transfer service for Bluetooth Low Energy. *2013 13th international Conference on ITS Telecommunications (ITST)*. IEEE. 2013, 319–324.

[23] Oliveira, P. F. and Matos, P. BLEGen—a code generator for bluetooth low energy services. *Lecture Notes on Software Engineering* 4.1 (2016), 7–11.

[24] Bluetooth, S. Bluetooth core specification version 4.0. *Specification of the Bluetooth System* 1 (2010), 7.

[25] Dian, F. J., Yousefi, A. and Lim, S. A practical study on Bluetooth Low Energy (BLE) throughput. *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE. 2018, 768–771.

[26] J. Nieminen, T. S. *IPv6 over BLUETOOTH(R) Low Energy.* URL: https://tools.ietf.org/html/rfc7668. (accessed: 03.08.2020).

[27] Thaler, D. RFC4903: Multilink Subnet Issues. *IETF RFC* (2007).

[28] Bluetooth. *Bluetooth Mesh – An Introduction for Developers*. URL: https://www.bluetooth.com/bluetooth-resources/bluetooth-mesh-networking-an-introduction-for-developers/. (accessed: 11.09.2018).

[29] Gomez, C., Darroudi, S., Savolainen, T. and Spoerk, M. IPv6 Mesh over BLUETOOTH (R) Low Energy using IPSP. (2019).

[30] Nieminen, J., Gomez, C., Isomaki, M., Savolainen, T., Patil, B., Shelby, Z., Xi, M. and Oller, J. Networking solutions for connecting bluetooth low energy enabled machines to the internet of things. *IEEE network* 28.6 (2014), 83–90.

[31] Shelby, Z. and Bormann, C. *6LoWPAN: The wireless embedded Internet*. Vol. 43. John Wiley & Sons, 2011.

[32] Bluetooth, S. *Internet Protocol Support Profile-Bluetooth Specification version: 1.0. 0*. 2014.

[33] Gogic, A., Mujcic, A., Ibric, S. and Suljanovic, N. Performance analysis of Bluetooth low energy mesh routing algorithm in case of disaster prediction. *Int. J. Comput. Electr. Autom. Control Inf. Eng* 3 (2016), 1075–1081.

[34] Kim, H.-S., Lee, J. and Jang, J. W. Blemesh: A wireless mesh network protocol for bluetooth low energy devices. *2015 3rd International Conference on Future Internet of Things and Cloud*. IEEE. 2015, 558–563.

[35] Leong, C. Y. and Koshijima, I. Internet of Things (IoT) for Dynamic Change Management in Mass Customization. *SEMANTICS Workshops*. 2017.

[36] Alliance, O. M. Lightweight machine to machine technical specification. *Technical Specification OMA-TS-LightweightM2M-V1* (2013).

[37] Shelby, Z., Hartke, K. and Bormann, C. The constrained application protocol (CoAP). (2014).

[38] Berners-Lee, T., Fielding, R. and Frystyk, H. *Hypertext transfer protocol–HTTP/1.0*. 1996.

[39] www.eclipse.org. *ECLIPSE WAKAAMA*. URL: https://www.eclipse.org/wakaama/. (accessed: 10.09.2018).

[40] Leshan™, E. *Leshan*. URL: https://github.com/eclipse/leshan. (accessed: 20.09.2018).

[41] Anjay. *Anjay*. URL: https://github.com/AVSystem/Anjay. (accessed: 11.09.2018).

[42] Schmidt, S., Tausig, M., Koschuch, M., Hudler, M., Simhandl, G., Puddu, P. and Stojkovic, Z. How Little is Enough? Implementation and Evaluation of a Lightweight Secure Firmware Update Process for the Internet of Things. *IoTBDS*. 2018, 63–72.

[43] Thantharate, A., Beard, C. and Kankariya, P. CoAP and MQTT Based Models to Deliver Software and Security Updates to IoT Devices over the Air. *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2019, 1065–1070.

[44] Weißbach, M., Taing, N., Wutzler, M., Springer, T., Schill, A. and Clarke, S. Decentralized coordination of dynamic software updates in the Internet of Things. *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. IEEE. 2016, 171–176.

[45] Kolomvatsos, K. An intelligent, uncertainty driven management scheme for software updates in pervasive IoT applications. *Future generation computer systems* 83 (2018), 116–131.

[46] Markopoulou, D., Papakonstantinou, V. and Hert, P. de. The new EU cybersecurity framework: The NIS Directive, ENISA's role and the General Data Protection Regulation. *Computer Law & Security Review* 35.6 (2019), 105336.

[47] Baldini, G., Fröhlich, P., Gelenbe, E., Hernandez-Ramos, J. L., Nowak, M., Nowak, S., Papadopoulos, S., Drosou, A. and Tzovaras, D. IoT Network Risk Assessment and Mitigation: The SerIoT Approach. ().

[48] Bormann, C. and Hoffman, P. *Concise binary object representation (cbor)*. Tech. rep. RFC 7049, DOI 10.17487/RFC7049, October 2013,< https://www. rfc-editor. org . . ., 2013.

[49] Keoh, S. L., Kumar, S. S. and Tschofenig, H. Securing the internet of things: A standardization perspective. *IEEE Internet of things Journal* 1.3 (2014), 265–275.

[50] PUB, N. F. *186-4,"Digital signature standard (DSS)," July 2013*.

[51] Josefsson, S. and Liusvaara, I. Edwards-curve digital signature algorithm (EdDSA). *Internet Research Task Force, Crypto Forum Research Group, RFC*. Vol. 8032. 2017.

[52] Moran, B., Meriac, M., Tschofenig, H. and Brown, D. A firmware update architecture for Internet of Things devices. *Internet-Draft draft-moran-suit-architecture-02, IETF* (2019).

[53] Moran, B., Meriac, M. and Tschofenig, H. Firmware Manifest Format. *Internet Engineering Task Force, Internet-Draft draft-moran-suit-manifest-01* (2018).

[54] Schaad, J. Cbor object signing and encryption (cose). *RFC 8152, Standards Track, IETF* (2017).

[55] Sheng, Z., Yang, S., Yu, Y., Vasilakos, A. V., McCann, J. A. and Leung, K. K. A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities. *IEEE wireless communications* 20.6 (2013), 91–98.

[56] Tschofenig, H. and Fossati, T. Transport layer security (tls)/datagram transport layer security (dtls) profiles for the internet of things. *RFC 7925*. Internet Engineering Task Force, 2016.

[57] Nordic. *nRF Command Line Tools*. URL: https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Command-Line-Tools. (accessed: 12.09.2018).

[58] Semiconductor, N. *nRF5 SDK for Mesh*. https://www.nordicsemi.com/Software-and-Tools/Software/nRF5-SDK-for-Mesh. 2018.

[59] Nordic. *Memory Layout*. URL: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v15.0.0%2Flib_bootloader.html. (accessed: 11.10.2018).