MASTER

Design of an IoT gateway for connecting legacy networks to the IoT

Bale, A.

Award date:
2017

Link to publication
Design of an IoT Gateway for Connecting Legacy Networks to the IoT

Master’s Thesis

By

Ankith Bale

(a.bale@student.tue.nl, ankith.bale@gmail.com)

University Supervisors: Dr. Tanir Ozcelebi and Dr. Johan Lukkien, TU/e
Company Supervisor: Michael Verschoor, Philips Lighting

Eindhoven, The Netherlands

24th January 2017
Abstract

One of the key application areas of Internet of Things (IoT) technology is building automation and lighting control systems. Although IoT based systems and networks have numerous advantages over legacy systems and networks, potential customers such as building and other infrastructure owners are not fully inclined to move to IoT based systems and networks. This is mainly because the existing legacy systems and networks are not compatible with the new IoT based networks and systems. This is a key element that needs to be solved as part of solving the interoperability problem in IoT. This master’s thesis proposes an IoT gateway to connect legacy systems to the IoT. Multiple architectures are possible for an IoT gateway based on key network parameters such as node, service and network.

In this master’s thesis, a set of potential gateway architectures are analyzed and the architecture that is most suitable, particularly for building and lighting control systems is selected. This is followed by proposal of a design for the selected IoT gateway architecture. The IoT gateway design covers three important use cases: discovery of nodes and services, unicast service interaction and multicast communication between legacy and IoT networks. The specifications of the IoT network is according to the Philips Lighting co-founded Fairhair Alliance which is an alliance of leading companies and envisions to create a common network infrastructure for interoperable lighting and building automation systems. A ZigBee network is considered for the legacy network in this project. A proof-of-Concept implementation of the IoT gateway architecture design is implemented using Philips in-house tools. With the IoT gateway implementation, a device in a Fairhair IoT network can discover devices and services in the ZigBee network, switch the state of ZigBee devices, add ZigBee devices to a group and control a ZigBee group. The use cases considered for the project cover most of the fundamental operations involved in connecting legacy systems to the IoT. Thus, the project’s implementation demonstrates an approach to overcome the challenge of adopting IoT technology for Building Automation and Lighting Control systems. The details of the analysis, implementation work flow and results, along with a supporting discussion are presented.
Acknowledgement

This Masters thesis work has been carried out at Philips Lighting B.V. with the intent to design an IoT gateway to connect legacy networks to the IoT. I would like to acknowledge all the people and institutions that have contributed in this work.

Firstly, I would like to thank Philips Lighting B.V. for providing me a wonderful opportunity to take up this project for my Masters Thesis and for giving me a chance to put my knowledge into practice and also develop my skills. I would like to thank my company supervisor, Michael Verschoor who provided me a deep insight and understanding of IoT and system architecture. He was very kind, helpful and supportive in guiding me through every phase of my project. Besides the relationship from a student to supervisor, I always felt a good level of friendship and cooperation. I learned a lot from Michael during my internship and when I needed support, he was always ready for me despite his busy schedule and business travels. I also thank Michael for inviting me to participate in the Fairhair Alliance weekly meeting calls which helped me to get a better understanding of Fairhair and get a better exposure during my internship at Philips Lighting.

I would like to thank my university supervisors Dr. Tanir Ozcelebi and Dr. Johan Lukkien of Eindhoven University of Technology for their scientific support and valuable feedbacks throughout the project. I am deeply grateful to Dr. Tanir Ozcelebi for motivating me to push myself during challenging times in the project. I am thankful to Dr. Johan Lukkien for introducing me to his research paper which served as a base for the analysis part of this project work.

I would like to thank Teresa Zotti of Philips Lighting for sharing her expertise on network protocols and providing resources on the same. I thank Sander Raaijmakers and Paul Zander of Philips Lighting for providing me with implementation tools for my project and am deeply grateful for their timely advises on implementation of my project, despite their busy schedules. I also thank Esko Dijk and Rob Davies of the Fairhair Alliance project team at Philips Lighting for sharing Fairhair related technical drafts which helped me to easily understand the details of Fairhair. I also thank other members of the Fairhair Alliance global team whose technical drafts on Fairhair helped me in getting a better understanding of Fairhair.

I also thank EIT Digital and TU/e for providing me an opportunity to work on the business side of this thesis work and thus in writing an ‘Innovation and Entrepreneurship’ minor thesis on the same.

I thank my parents, family and friends who were always supportive and encouraging throughout the course of my project work, to which I am eternally grateful. Lastly, I thank my parents once again for presenting me with Walter Isaacson’s book- the Innovators which was a great source of inspiration throughout my project.

Ankith Bale
# Table of Contents

1. Introduction.................................................................................................................9
   1.1 Background- IoT.................................................................................................9
   1.2 Smart Building Automation and Lighting Control Systems.................................10
   1.3 Fairhair Alliance...............................................................................................11
   1.4 Motivation for the Project................................................................................12
   1.5 Thesis Layout...................................................................................................13

2. Problem Description..................................................................................................14
   2.1 Application Scenario .........................................................................................14
   2.2 The Interoperability Problem ...........................................................................16
   2.3 Problem Description .........................................................................................17
   2.4 Use Cases..........................................................................................................19
   2.5 IoT Gateway .....................................................................................................20
   2.6 Related Works..................................................................................................23

3. IoT and Non IoT Networks.......................................................................................26
   3.1 Fairhair IoT Network .........................................................................................26
   3.2 BACnet Network ...............................................................................................34
   3.3 ZigBee Network ................................................................................................38

4. Gateway Architecture Analysis and Design..........................................................42
   4.1 Analysis.............................................................................................................42
      4.1.1 Standard IoT Communication.................................................................42
      4.1.2 ZigBee Frame Formats..............................................................................45
      4.1.3 Discovery, Switching and Multicast in ZigBee ........................................48
      4.1.4 Gateway Architecture Options..............................................................50
   4.2 Design................................................................................................................69
      4.2.1 Discovery Use Case..................................................................................71
      4.2.2 Switching Use Case................................................................................73
      4.2.3 Multicast Use Case..................................................................................77

5. Implementation..........................................................................................................80
   5.1 Implementation Tools.........................................................................................80
      5.1.1 ZCLIP Tool: For Fairhair IoT Network Implementation..........................80
      5.1.2 HSE Tool: For ZigBee Network Implementation .......................................82
   5.2 Implementation Details......................................................................................84
5.2.1 Implementation Overview. ................................................................. 84
5.2.2 Implementation Work Flow: Gateway Application. .............................. 85
5.2.3 The IoT Gateway Translation Algorithm. ............................................. 89

5.3 Gateway Communication Mechanism. .................................................. 104

6. Results and Discussion. .................................................................. 105
   6.1 Result Terminologies. ...................................................................... 105
   6.2 Results. ......................................................................................... 107
      6.2.1 Gateway Initialization. ............................................................. 107
      6.2.2 Discovery Use Case. ................................................................. 118
      6.2.3 Switching Use Case. ................................................................. 123
      6.2.4 Multicast Use Case. ................................................................. 131

7. Conclusion. ....................................................................................... 142
   7.1 Conclusion. ................................................................................... 142
   7.2 Challenges. ................................................................................... 143
   7.3 Future Prospects. .......................................................................... 144

References. ............................................................................................. 146
Appendix. .............................................................................................. 150
   A. Abbreviations. ............................................................................... 150
Chapter 1. Introduction

With the advent of the Internet of Things (IoT), Building Automation and Lighting Control (BA and LC) Systems are undergoing transformation at an accelerated rate and are able to offer large number of benefits such as significant energy savings, remote monitoring, ambient living environments etc. This chapter starts by briefly describing how BA and LC systems are being shaped by the IoT. This is followed by a section that provides an insight into the possibilities of IoT enabled BA and LC systems- Smart BA and LC systems. Since this Master’s Thesis project work uses an IoT network that conforms to the Fairhair IoT alliance, an introduction to the Fairhair alliance is provided. Using this as a base, the motivation behind the Master’s Thesis project work is described which is followed by the Thesis layout details.

1.1 Background

The Internet of Things (IoT) is said to revolutionize multiple application domains such as Transportation and Logistics, healthcare, smart environments and entertainment. In order to stimulate market growth in the smart environments domain through the use of new features and benefits unlocked by the IoT, there has been a dedicated collective effort by various organizations working in the field. This has resulted in the formation of important industry alliances. A few of such prominent alliances include the Fairhair Alliance [1], the Thread group [2], AllJoin [3] etc. Current BA and LC systems are successfully served by the key network ecosystems such as BACnet, ZigBee and KNX. Several of these ecosystems will extend to the use of IP and IoT technology. But, existing deployments and investments need to be compatible with this transition. Thus, ‘interoperability’ between legacy and IoT is the key for success of this transition. Out of the many issues that are being tried to be addressed by these alliances, an important issue is that of ‘Interoperability’ between legacy and IoT enabled devices.

This Master’s thesis focuses on overcoming the interoperability problem between legacy and IoT enabled devices in the context of BA and LC systems. BA and LC companies would be very reluctant to invest in a technology which is completely incompatible with all their existing installations, which they need to maintain. Thus, any new technology such as the IoT needs to address how it incorporates with existing systems when it is to be accepted. Thus, overcoming the interoperability problem addresses this concern and offers significant cost savings for building owners since they do not need to replace their numerous existing legacy BA and LC devices due to the shift in technology from legacy to IoT technology. A few other benefits that could be achieved by overcoming the interoperability problem are: scalability, reliability and robustness. Also, since the solution would involve IoT which is primarily different in its ability to allow tooling, data retrieval and other end-to-end application services, the solution would allow more focus on application layers. Besides, the solution would provide key end user IoT benefits such as optimizing building performance without human intervention.
1.2 Smart Building Automation and Lighting Control Systems

Building Automation systems have been around for a few decades but with the addition of the Internet capability, they are now undergoing transformation leading to new frontiers such as smart buildings, smart homes etc. Apart from providing the basic building automation functionalities such as temperature regulation and humidity control, smart building automation systems interact with the building occupants through their PDAs and can also function autonomously. This interaction provides added services for all stakeholders, right from building users to building owners. Smart building automation systems offer exciting features such as vehicle parking assistance, irrigation control, indoor navigation, meeting room reservations and all the way up to real time detailed tracking of energy consumption in a building and evaluation of building’s performance [4][5]. Therefore, smart building automation systems engage all the different stakeholders into the system to offer higher value services to all.

Similar to building automation systems, lighting control systems, both indoor and outdoor are also undergoing transformation and are becoming more intelligent by gaining Internet connectivity. Internet connectivity not only involves the systems/devices that are being directly connected to the Internet but also involves those systems/ devices that use Internet-inspired technologies such as IP and HTTP to facilitate integration to mobile apps or web browser user interfaces. Smart outdoor lighting control systems now provide benefits such as remote control and monitoring of outdoor lights such as street lamps through the Internet, reporting of malfunctioning lamps etc. These in turn offer multiple benefits to end users and infrastructure owners. A few of the prominent ones include adaptive street lighting for citizens and significant cost and energy savings for lighting infrastructure owners. [6] Indoor lighting control systems also offer value added services such as customizable lighting profiles for buildings and homes, facilitating personalized work spaces that improve efficiency and comfort levels, provide effective visualization of energy consumption by lighting devices etc. Thus lighting control systems equipped with Internet connectivity go far beyond mere illumination [63].

The solutions offered by Smart BA and LC systems in new application areas such as smart homes and smart buildings are primarily focused on energy management, environment monitoring, assisted living and convenience. In this context, many companies are considering developing platforms that integrate building automation with other services such as energy monitoring and lighting control. This would allow IoT applications to gather data through sensors, and cloud based analytics softwares to analyze disparate data which ultimately helps building owners to manage buildings at high efficiency. Technologically, a smart building can be divided into three layers: Envelop (materials and design), connectivity (devices and network infrastructure) and the software layer (data analytics). This thesis is focused on the connectivity layer, particularly in achieving interoperability between IoT and non-IoT networks. Apart from the technical requirements, it also requires stakeholder co-operation to achieve interoperability. In the context of the IoT, smart BA and LC systems can be considered as part of a much larger information system which is based on the existing intranets and the Internet and also the wireless sensor networks [8].

In order that smart BA and LC systems fully utilize the potential of IoT, it is crucial to achieve interoperability in IoT; between the Internet network world and the wireless sensor network world. Interoperability issues in the IoT arise due to its characteristic features: i) high dimensional- co-existence
of many systems ii) highly heterogeneous- vast systems are conceived by different manufacturers for different application domains iii) dynamic and non-linear- new things enter the environment with different protocols iv) hard to describe or model due to many data formats. These features also present themselves as complex IoT interoperability problems. [8]

With reference to the above mentioned IoT characteristic features, this thesis formulates the problem, primarily in terms of the heterogeneous systems- IoT system and legacy system. It explores multiple gateway architectures to achieve interoperability between these systems and simulates the most suitable gateway architecture, which demonstrates the interoperability achieved between the two heterogeneous systems.

1.3 Fairhair Alliance

One of the key solutions for addressing the IoT interoperability problem is, creation of global agreements and a widely accepted specification. Fairhair alliance is a concerted effort in this direction. Fairhair is an open industry alliance co-founded by global lighting leader- Philips Lighting B.V. along with Cisco Systems Inc. The alliances’ members include key ICT industries such as Siemens AG, Silicon Labs Inc. etc. Fairhair alliance aims to facilitate the use of IoT technology by BA and LC systems. [9] The alliance believes that, systems such as lighting control systems are the first logical steps towards the idea of bringing IoT to enterprises worldwide, in a way that it becomes intelligent to sense and measure the environment [10]. Fairhair Alliance provides draft specifications for web services solutions and others for key BA and LC system ecosystems such as BACnet, KNX and ZigBee. The scope of the specifications includes resource models and a unified solution for application services linked to the transport layer. Fairhair aims to achieve harmonization of application layer services across the relevant, aforementioned ecosystem standards. This effort seeks to achieve interoperability for common network services such as device and service discovery, security and network management [9].

Fairhair defines a framework that specifies the constrained web services solution for application ecosystems. The Fairhair architecture stack of an IoT node consists of resource constrained wired or wireless PHY and MAC layer protocols, IP v6 and UDP protocols at the network and transport layers respectively, and CoAP at the application layer. On top of the CoAP layer, lie two Fairhair application sublayers- Fairhair application service sublayer and Fairhair resource model sublayer. The application service sublayer consists of application service blocks such as security, service discovery, network management and ecosystem specific services. The Fairhair resource model sublayer consists of an ecosystem specific resource model such as a ZigBee resource model or a BACnet resource model.

Fairhair facilitates ecosystems to adopt a common RESTful interaction model, established IETF data formats and a set of uniform application services such as service discovery which would ultimately help in achieving a high degree of interoperability [9]. Fairhair achieves this with its framework that specifies the following [9]:

- A generic model description of a domain model in the form of web resources and mapping of elements in the data model to URIs
• Mapping of existing methods to interact with elements in the data model to RESTful methods as defined by CoAP protocol
• Data encoding formats that conform to the IoT, such as JSON or CBOR
• Scalable mechanism for device and service discovery that is independent of ecosystem specific semantics
• Application services related to the security model and network management

With regard to the above features, Fairhair offers solutions that are as much as possible on the state of the art from the IETF. The Fairhair frame work is neutral towards the ecosystems. Thus, it would not affect the ecosystems wrt their individual strategies for the IoT market which differ in technology choices for services that are orthogonal to the ecosystems’ data models [9].

One of the examples of services that will benefit from Fairhair is Service Discovery which deals with discovery and identification of a device along with its capabilities and application functionalities. Fairhair provides a scalable discovery solution and also the rules which specify how the resources are registered, maintained and discovered. Fairhair also provides a generic data model that highlights the information such as unique identifier and object type, which the discovery mechanism requires. With these, an ecosystem can exploit the unified Fairhair solution and make it useful for it application by using its specific semantic such as a ZigBee endpoint, in case of the ZigBee ecosystem. With regard to what is not under the scope of Fairhair is the certification of the solution; it needs to be done according to the rules and specifications of the ecosystem [9].

Thus, the key role of Fairhair in the domain of BA and LC systems is, it defines a framework for an IoT solution, which includes a generic data model that represents the application domain in terms of web resources, and a set of orthogonal application services. The key challenge of the Thesis is to facilitate the transition of BA and LC ecosystems to IoT technology. It addresses this challenge by designing an IoT gateway that facilitates easy ‘interoperability’ between legacy and IoT enabled devices. Fairhair is taken as the embodiment of the IoT system and ZigBee is selected as a representative technology for current LC systems.

1.4 Motivation for the project

An IoT gateway, interacting between a Fairhair IoT system and a ZigBee system is designed and implemented in this thesis. For the intranet communication, the IoT devices in an IoT network use IP protocol whereas legacy devices in a network such as a ZigBee network use ZigBee communication protocol. As described in the background, the success and adoption of the IoT in the area of BA and LC system largely depends on making the IoT devices compatible to work with the existing non IoT based/legacy devices- interoperability. Interoperability would allow BA and LC companies and building owners to migrate to IoT based devices for new installations and also keep their existing legacy devices. This would ultimately provide easy maintenance and significant cost savings for BA and LC companies and building owners. To achieve interoperability between devices of the two aforementioned different types of networks
necessitates the use of IoT gateways. With respect to the key gateway characteristics - gateway services offered and interface granularity, multiple approaches are available for the design of gateways as described in [11]. Each of these approaches have their own advantages and disadvantages. Thus the choice of a particular approach (es) for a gateway design depends on the application area. This thesis analyses each of these options in detail, by going down to a packet level analysis and proposes a gateway architecture for BA and LC systems. The proposed architecture is a combination of more than one approach and provides a high degree of interoperability between Fairhair IoT network devices and ZigBee network legacy devices. A prototype of the proposed gateway architecture is implemented through simulation. The thesis also evaluates the proposed gateway architecture for key functional and nonfunctional requirements - scalability, reliability, robustness and group management.

1.5 Thesis Layout

The thesis report is organized as follows: chapter 2 deals with the problem description. It considers an application scenario and describes the interoperability problem and discusses the approach of using IoT gateways as a possible solution. Chapter 3 provides a detailed discussion on IoT and legacy networks by considering relevant Fairhair alliance specifications and legacy networks: BACnet and ZigBee. Chapter 4 deals with analysis of potential IoT gateway architectures and design of the selected IoT gateway architecture. Chapter 5 provides details of the implementation workflow and the implementation tools. Chapter 6 presents the implementation results with a supporting discussion. Chapter 7 concludes the thesis and discusses its future prospects.
Chapter 2. Problem Description

An IoT gateway, introduced in the previous chapter may serve for many IoT application areas. This thesis work however confines itself by proposing an IoT gateway for Building Automation and Lighting Control (BA and LC) systems. One of the key problems for building and related infrastructure owners in adopting new IoT based BA and LC systems is, the concern that whether the new systems are interoperable with the already existing systems. This is because of the significantly high costs involved in completely replacing the existing systems with the new ones. This discourages the building owners to migrate to the new IoT based systems, despite the large number of benefits offered by them such as scalability, reliability, energy and cost savings and operation without human intervention. A solution thus needs to be found to facilitate easy adoption of new IoT based BA and LC systems such that the existing systems can work along with the new IoT systems, thereby avoiding replacement of existing systems and thus encouraging building owners to migrate to IoT based systems without any hassle.

This chapter: introduces the application scenario in which the proposed IoT gateway would work in, describes the Fairhair IoT and ZigBee systems, defines an IoT gateway and its characteristics, and defines the interoperability problem between legacy and IoT devices with use cases. The chapter ends by presenting and introducing a few important related works.

2.1 Application Scenario

In a typical large building environment, building users can remotely control devices such as luminaires, heating systems, air conditioning systems and ventilation systems located in one part of the building through switching and controlling devices located in totally different part of the building. This is made possible due to a systematic commissioning process, which deals with connecting various network components together in the network so that the normal application can run. A simple instance for this could be connecting a switch to a light wirelessly in a legacy network such as ZigBee. The key steps involved in the commissioning process include service discovery and binding. Service discovery deals with discovering end devices and the corresponding services that they offer. For example, an application may involve a switch manufactured by Siemens controlling a light manufactured by Philips. In order for the application to work successfully, the application needs to first find the end devices offering the service of switch and light. This is facilitated by Service Discovery. The other key step in commissioning process is binding, which deals with attaching a service offered by one device to a service offered by another end device. From the aforementioned example, binding facilitates linking of the light service with the switch service.

Although the commissioning process is a tedious process, it is relatively simple in legacy networks such as ZigBee or BACnet as compared to IoT networks. This is mostly because a single type of network is involved in legacy networks. But in buildings using the IoT networks, commissioning needs to be facilitated across two types of networks- the new IoT network and the existing non-IoT/legacy network. This necessitates the use of an intermediary device such as an IoT gateway to operate between the two types of networks. When a building user uses a new IoT based switching device connected to an IoT network, the IoT gateway
device receives the data packet from the switch through the IoT network. Based on the data/command packet received, the IoT gateway translates the packet contents and forwards it to the appropriate legacy device address. This will accordingly change the state of the destination device. The destination can also be a group of devices. In such a case, the gateway discovers the destination group and changes the state of all devices of a particular destination group. Finally, the IoT gateway communicates the new status of the destination legacy device/group back to the source IoT device.

A typical IoT System architecture is shown in figure 1. The architecture depicts the role of IoT gateways in connecting legacy systems/devices to the Internet [22]. Figure 2 considers the specific example of building automation systems in illustrating the role of IoT gateways, in connecting legacy systems in buildings such as temperature sensor based systems and water flow measurement systems to the Internet. The figure also highlights the difference between IoT based systems and legacy systems. From the figure, the electric use sensor device, which is IoT based is directly connected to the Internet/cloud whereas the non-IoT based devices such as the temperature sensor device are connected to the Internet through the IoT gateway. However, in the context of this thesis, the IoT gateway does not only serve to connect legacy devices to the Internet but also to other devices that enable building automation at a floor or room level. Thus, for instance, the communication scope is not just limited to a legacy luminaire being connected to the Internet through a gateway but, it also includes the legacy luminaire being connected to an IoT switch through the gateway.

Fig 1. A Typical IoT System Architecture (figure from [4])
Fig 2. A Smart Building Example (figure from [4])

2.2 The Interoperability Problem

Interoperability is a challenge for any distributed system in which equipment of multiple vendors and/or industries is integrated. Since IoT applications are typical examples of distributed systems, interoperability offers a major challenge. Achieving interoperability refers to the idea of making the entities/devices involved to communicate seamlessly, irrespective of the make, model, manufacturer or industry. In a fully interoperable environment, any IoT device would be able to connect to any other device and communicate information as required. In practice, it is hard to achieve full interoperability. Some of the important considerations and challenges offered in achieving interoperability include, proprietary ecosystems and consumer choice, technical and cost constraints, legacy systems and configuration issues [14].

The key challenge that this project tries to address is that of legacy systems, which are the ones that are already deployed and are operating [14]. The application area that is targeted is that of BA and LC systems. Legacy systems impose a strong requirement on the IoT systems to provide a way to integrate themselves with an existing legacy system in a cost-effective way. Thus the success of IoT systems is proportional to the extent to which the IoT systems can be made integrable with the legacy systems. The successful integration requires a bidirectional communication, with consistent semantics between the networks. This forms the essence of the Interoperability problem.
Using figures 1 and 2 as a background to introduce IoT based devices, legacy devices and the IoT gateways, the interoperability problem considered by our project is described by figure 3. The IoT network is constituted by Fairhair IoT devices and the non-IoT network is constituted by devices that implement a legacy BA or LC protocol. Since the IoT application area considered by the project is BA and LC systems, the devices in the networks include luminaires, HVAC, sensors and actuators. The IoT devices communicate using an IoT protocol such as CoAP and the legacy devices in the legacy network communicate using a non IoT protocol such as ZigBee, BACnet or KNX. Thus an IoT device cannot communicate with a legacy device since they essentially use two different types (IoT and non-IoT) of protocols which constitutes the interoperability problem.

2.3 Problem Description

As briefly discussed in the above paragraphs, the interoperability problem is a complex problem and it arises due to multiple factors such as coexistence of multiple systems, vast systems conceived by many manufacturers, unforeseen communication protocols, and existence of many data formats. Thus, to effectively address the interoperability problem, it is essential to quantify it and define a scope. In this direction, figure 4 quantizes the generic Interoperability problem to a more focused problem that is considered in the project. The figure is used as a base to formulate the problem and propose a suitable architecture for the IoT gateway in the next chapters. The application scenario consists of Fairhair IoT devices communicating with legacy devices of a ZigBee network. The IoT devices conform to the Fairhair standard described in chapter 1 and hence are referred to as Fairhair IoT devices. Two Fairhair IoT devices and three ZigBee legacy devices are considered. Fairhair IoT devices use the Fairhair IoT / IP protocol for communication whereas the legacy devices use non IP based protocols- ZigBee. Although there exist multiple legacy protocols such as BACnet and KNX, ZigBee was chosen for study of the legacy system part because multiple reasons which include: i) In the context of BA and LC systems, ZigBee is the most widely used legacy protocol. ii) It provides the concept of clusters which can be very easily related to
services, which are the key entities of the IoT. iii) It offers unique characteristic features such as: low power, low cost and low data rate communication which make it preferable in both academia and industry for implementing prototype and mature applications. iv) It caters to the special requirements of device-device communication which is crucial for IoT related applications. In the context of BA and LC systems, any service offered by networks, in principle is similar to one of the three fundamental services which include discovery, switching (On/Off) and group control/management. E.g., an alarm service, in principle is similar to the switching (On/Off) service. Thus discovery, switching and group management are the services considered in our project and they provide a representative view on the overall problem description. In figure 4 which illustrates the problem description, the devices are represented as ‘nodes’ since the devices are part of networks. The IoT nodes- #1 and #2 are composed of On/Off and level control switches and the ZigBee legacy nodes- A, B and C are composed of On/Off and dimmable luminaires as shown in figure 4. We assume that the On/Off luminaires in the ZigBee legacy network are initially in their Off states and the dimmable luminaire is at level zero. Also, the corresponding On/Off and level control switches of the Fairhair IoT network are assumed to be in their Off states and at level zero respectively. These assumptions are considered by the use cases described under section 2.5 which highlight the inter-network communications.

Fig 4. Problem Description
This master’s thesis project tries to address the interoperability problem of the IoT technology used in BA and LC systems through the following research questions:

- What are the potential IoT gateway architectures possible, with regard to configurations based on nodes, services and networks and what is the most suitable IoT gateway architecture for BA and LC systems?
- How to achieve service discovery, node management and control, and group management across IoT and non-IoT networks?
- How does the most suitable IoT gateway architecture perform with regard to scalability, reliability, and grouping and what are the suitable metrics for these performance parameters?

2.4 Use Cases

The problem of interoperability arises when a building user tries to control a non-IoT legacy device from an IoT device or vice versa. In this project work, use cases involving control of legacy devices from IoT devices alone are considered. The interoperability issues can be summarized in the form of the following use cases:

1. Use case #1:
   A building user using the IoT network tries to find out all the available devices and services present in the legacy network. Since the user and the services are present in two different networks, the IoT gateway needs to receive the command from the user through an IoT device such as a switch and perform a service discovery in the legacy network. The gateway should then acknowledge the user with a list of available services.

2. Use case #2:
   A building user tries to switch on a single On/Off legacy luminaire from an IoT On/Off switch. Since the devices belong to two different types of networks, the IoT gateway should receive the command from the IoT switch, find the corresponding destination legacy luminaire and change its state. The IoT gateway should also communicate the changed status of the legacy luminaire back to the IoT switch.

3. Use case #3:
   A building user tries to switch on a group of On/Off legacy luminaires from an IoT switch. In this case, the IoT gateway needs to first create a group out of a set of On/Off services. Since the switch and the group of On/Off luminaire devices belong to different types of networks, the IoT gateway needs to receive the command from the IoT switch, find the On/Off luminaire group and switch on all the luminaires in the particular destination group. Also, the IoT gateway needs to communicate the changed status of the On/Off luminaire group back to the IoT switch.
The IoT gateway implements the above use cases through a set of algorithms. Figure 5 provides an overview of the role of the IoT gateway; it connects the Fairhair IoT devices and the legacy devices. It receives requests/commands from the Fairhair IoT devices and achieves node discovery, node management and control, and group management in the Legacy network by using device/service discovery and group management algorithms. Multiple options exist for implementing the mechanisms such as service discovery, which can be implemented through two approaches—centralized approach and distributed approach [6]. These approaches and the associated algorithms are discussed in the next chapter.

2.5 IoT Gateway

Multiple approaches are possible to overcome the Interoperability problem. One of the simple approaches is, designing every component in an IoT system to be backward compatible with legacy systems. But this comes with a severe drawback which is, it imposes design tradeoffs on IoT systems which limits the...
capabilities of the IoT system. An alternative and an efficient approach is the use of IoT gateways. An IoT gateway is a device that connects an IoT network to a non IoT network. Besides providing seamless integration of the two networks, it facilitates management and control of IoT applications with non IoT networks [15]. The IoT gateway is associated with three key functionalities: i) It forwards messages/data packets from a particular source node in an IoT network to a particular non IoT network destination node. ii) It facilitates nodes of IoT network to subscribe to services offered by nodes of non IoT network and also control those. iii) In some cases, it facilitates loading and unloading of programs into the non IoT network from the IoT network [15]. These functionalities are bidirectional, with regard to the participating networks. An IoT gateway is composed of multiple functional blocks, which include service discovery, group management, binding, protocol conversion, caching service and prioritization. It is important to note that some of these functions are directly related to the protocols which are to be supported, i.e. group management for instance is only a gateway function when the protocol does support the concept of groups. Other functions are related to the gateway itself: prioritization and caching do not need to exist as a concept in the protocol, but may still be needed in the gateway to enable efficient integration. In the case of IoT/M2M applications, devices have to operate autonomously and have to be maintained at low cost. [6] This necessitates the need of service discovery block, which facilitates the identification of nodes and services in networks. Another important gateway block is, group management. Group management deals with addressing or controlling a collection of similar network nodes simultaneously. Group management is essential for applications such as Building Automation. For instance, when a single occupancy detector device wants to use multiple similar services at the same time such as turning on several smart lights, it can be facilitated through group management. [19]

A few other gateway blocks that were considered for the IoT gateway design but are not explicitly implemented in our project include binding, event reporting and prioritization. The Binding block allows an endpoint on a node in IoT network to be connected or bound to a receiving node in the non IoT network [19]. Binding is unidirectional and an example in our project would be an On/Off switch bound to an On/Off light. The event reporting block enables to define alarms or other types of similar events and to indicate which nodes should be notified when they occur [20] [21]. Event reporting is a key feature of BACnet networks. Because of this and also since the use cases do not consider an alarm/ similar service, an exclusive event reporting block is not considered for implementation in our project. However, an event/alarm would conceptually be similar to a unicast command, which is included under the On/Off use case to study that aspect. Also, an event subscription which is a characteristic feature of the BACnet protocol is similar to binding. The Prioritization block facilitates assignment of different priority levels to commanding entities of an IoT network. A good example for the importance of this block is, if the IoT gateway simultaneously receives an ‘On’ command and a ‘set Level = 0’ command from nodes of the IoT network, the Prioritization block enables the gateway to decide to process the ‘On’ command first, assuming that the ‘On’ command has the higher priority. Since prioritization is mostly used in BACnet networks, it is not considered for implementation in our project. However, prioritization may also be implemented on the gateway, independent of whether the protocol supports that feature or not to ensure commands are processed in the desired order.
Although the IoT gateway is made up of multiple functional blocks as discussed in the above paragraphs, only a selected set of functional blocks have been chosen for the gateway designed and implemented in this project and they are as discussed below.

1. Service Discovery

Service Discovery is a mechanism that allows automatic detection of devices and services offered by devices in a network. It is the process of providing a service provider for a requested service. Retrieving the demanded service, which is typically the address of the service provider enables the user to further access and use it. Every service discovery mechanism consists of at least two entities - server and client [38]. Since the IoT gateway operates between two network types, the service discovery block would have two parts - the part that works on the Fairhair IoT network-IoT gateway interface operates as the server and the part that works on the IoT gateway-ZigBee legacy network operates as the client.

Although service discovery is bidirectional with respect to the participating networks, for our project the IoT gateway service discovery block design is limited to one direction; the services offered by devices in the ZigBee legacy network are discovered by devices in the Fairhair IoT network. Two types of services are offered by the ZigBee legacy network which include turning on or off the on/off luminaires and adjusting the level of level control luminaires as indicated in figure 4.

2. Management and Control

This block is associated with the function of controlling nodes of one network from nodes of another network. This block allows the gateway to receive data/ commands from nodes of one network and dispatch them to either merely transfer transparently or manage and control nodes in another network [15]. A good example from our project for this block is that of an on/off switch present in the Fairhair IoT network turning on an on/off luminaire present in the ZigBee legacy network, which is a part of use case #1.

3. Group Management

As briefly described above, group management deals with grouping a set of similar nodes based on the services offered. Group management is implemented by collecting a set of nodes into a single addressable entity - ‘group’. A single data request can reach every node in a group [17]. This block mostly consists of two parts: an interface to create/ modify groups, and an interface to control a group.

Group Management is a mechanism that is typically helpful in home and building automation [17]. An example for a group is the case of a ‘enter building’ group. The group can consist of node A and node B. The group can turn on the On/Off state variables of nodes A and B and set the level of dimmable luminaire of node B to 50%. The use case #2 considered by our project is a part of this example.
2.6 Related Works

Berta C Villaverde, et al. [17] provides a detailed description about the service discovery protocols available for constrained Machine to Machine applications. It describes the general service discovery protocols concepts, service discovery protocols for low power networks- taxonomy, CoAP service discovery and its evaluation and open source simulation tools. From the general service discovery protocols concepts and taxonomy, IP networks usually use a centralized approach for service discovery. In the centralized approach, there exists a centralized directory to store descriptions of the services offered by servers and this allows clients to perform look ups on these services. With this approach, all the services offered by servers are registered automatically by them and clients can discover the services with a single request. However, the Fairhair IoT network used in the project uses a hybrid approach, combining centralized and distributed approach. We use a distributed approach for service discovery in the ZigBee network. In the distributed approach, a client directly queries the server, without the involvement of an intermediate directory. Although this approach has the advantage of not requiring an intermediate directory, the client is required to know the IP address or the host name of the server, to which it queries. The main functionality for the service discovery protocols used in these networks is registration, with a stateful mechanism. Registration is a process through which descriptions of services offered by network devices are stored/registered into a directory to make these descriptions available for discovery within the corresponding domain. A stateful registration mechanism is a registration mechanism where in an explicit registration is made directly from the device to the directory. Thus, [17] helps in selecting the service discovery mechanisms for the networks of the project.

Qian Zhu, et al. [15] proposes an IoT gateway system based on ZigBee and GPRS protocols. The proposed gateway system is according to typical IoT application scenarios and requirements from telecom operators. It presents the data transmission between wireless sensor networks, protocol conversion of different sensor network protocols and control functionalities for sensor networks. The paper considers the use case scenario of a smart home which is a typical IoT application area. Since BA and LC systems are closely related to smart home/buildings, the IoT gateway described in the paper can be used as a reference for our project work. The paper provides the software architecture of an IoT gateway, which includes key functional blocks such as Log management, command mapping and protocol conversion. We use this to define the software architecture of our IoT gateway which contains the service discovery and group management blocks, in addition to the aforementioned key functional blocks from [15].

Remi Bosman, et al. [11] provides details of multiple gateway architectures of application level gateways that connect service oriented sensor networks to service oriented standards in the consumer electronics domain. The paper presents a tradeoff between the richness of the exposed service interfaces and the deployment granularity of the gateway services. One of the key requirements involved in the IoT is, connecting wireless sensor networks and IP based networks. Gateway devices are one of the solutions that facilitate this requirement. Paper [11] validates its application level gateway architectures through a

The IoT gateway developed in our project is related to the application gateway implemented in [5], primarily because both the gateways connect an IP based network to a non-IP network and also because the networks involved with both of the gateways are service oriented. Thus, the six characteristic options applied to the application gateway in [11] can also be applied to the IoT gateway developed in our project. Thus, each of the six characteristic options are analyzed at the packet level for the IoT gateway and a combination of these options is chosen for implementing the use case scenarios.

The Bui, et al. [23] examines the problem of bridging ZigBee domain and the IP domain which includes the aspects such as, allowing standardized access and easy integration of advanced lighting scenarios. The paper defines two access levels for bridging - a message based API and an HTTP interface that gives access to an embedded web application for direct control purposes. The paper defines the requirements for the smart bridge by considering the application area of consumer homes. Since an HTTP-ZigBee gateway interface implemented in [23] is conceptually similar to the CoAP-ZigBee interface implemented in this project, paper [23] would be relevant to compare our gateway design against it. Some of the entities that can be taken up for comparison include the interfaces, e.g. the interface defined by Bui et.al. for the interaction for turning ON a light point. Although the project deals with linking the IP/ IoT network to a ZigBee network, the ZigBee network is implemented in hardware and thus this would not be very useful for our project since the ZigBee network in our project is implemented through simulation.

Yoona Kim, et al. [16] proposes the idea of using a combination of centralized and distributed CoAP discovery mechanisms. The combination, which is termed as a hybrid approach first looks up resources in a centralized fashion for resource discovery and switches to the distributed method if the Resource Directory (RD) is overloaded. This overcomes the over burden on the Resource Directory (RD) according to the authors. The paper supports its claim by evaluating the performance of the hybrid approach. The performance is evaluated using a set of mathematical equations and corresponding plots. Since the Fairhair IoT network also uses a hybrid approach for service discovery, the performance evaluation method used in [16] can be referred to for evaluating the Fairhair IoT network performance.

Hao Chen, et al. [24] provides a good introduction to the concept of IoT gateway. It provides a set of essential features of an IoT gateway which include: multiple interfaces, protocol conversion and manageability. Besides, it provides a reference model for an IoT gateway. The reference model contains multiple modules such as service abstraction module, protocol conversion and data forwarding module, service abstraction module and network profile module. The model provides the interaction between the modules. E.g.: the protocol conversion and data forwarding module interact with service abstraction module and multi-interfaces module to analyze and re-pack sensors data based on communication protocols. The module transfers data from one network interface to the other. The paper can thus be used for defining basic requirements for our gateway’s modules during implementation.

Mattia Antonini, et al. [25] proposes a light weight multicast forwarding algorithm for service discovery in low power IoT networks. The authors claim that the proposed algorithm offers benefits wrt memory foot
print when compared with other multicast algorithms for low power IoT networks. The performance of the proposed forwarding mechanism is evaluated through Contiki based nodes in Cooja simulator. The first performance indicator used for the evaluating mechanism is memory occupation, measured in Bytes. The specific performance metrics include: Query Client (QC) time, which is the time needed by a client node to send a query and receive a response. ii) Energy consumption, which is the overall energy consumption in the network for the query operation. Since our project also involves a service discovery as one of the use cases, the aforementioned performance metrics can be considered to evaluate the performance of our service discovery mechanism.

From the above paragraphs of this section, it can be observed that many works related to this project have been carried out and there is little effort done so far to address the interoperability problem with respect to legacy systems. For e.g. Qian Zhu, et al. [15] although provides a gateway implementation, it does not cover service discovery which is very crucial in IoT applications to address the interoperability problem. There are many works related to implementation of gateways and many approaches have been proposed for key IoT concepts such as service discovery, as by Yooa Kim, et al [16]. But there are very few works available which provide gateway implementations based on key IoT services such as service discovery. Besides there are also not many works that provide gateway implementations for BA and LC systems to address the interoperability problem. Thus this project, which attempts to address the interoperability problem by combining the existing knowledge of gateway implementations and that of key network services such as service discovery tries to fill the knowledge gap.
Chapter 3. IoT and Legacy Networks

Previous chapters provided a description of the problem and quantified it. This chapter provides a detailed description of the different types of networked systems that are involved in the project - the IoT and the legacy networks. The IoT network in this project is constituted by an Internet Protocol (IP) based network that uses the specifications laid down by the Fairhair Alliance [1] and the legacy network is constituted by a ZigBee network which uses the ZigBee 3.0 protocol. Thus, this chapter serves as a good base to understand the next chapter, which presents the gateway architecture analysis and design by considering a Fairhair IoT network and a ZigBee legacy network. Since the project is more focused on the application layer, the discussion is mainly confined to the resource model and discovery mechanisms in the networks.

3.1 Fairhair IoT Network

Fairhair defines a kind of framework, facilitating the BACnet, KNX and ZigBee eco-systems in their transition from a current non-IoT solution to an IoT solution. Fairhair is an alliance that does not have a defined end. In that sense, it is different from a project. There are projects like OpenAIS and BaaS, however, in close collaboration with Fairhair alliance. Strictly speaking there is no “Fairhair IoT network” or “Fairhair system”, but a “ZigBee Fairhair inspired IoT system and a “BACnet Fairhair IoT system” etc. These IoT systems will share commonalities as they use the same Fairhair framework. So, Fairhair is a framework for a solution and ZigBee IoT for example is the actual solution. But, for describing the Fairhair framework, the term “Fairhair network” or “Fairhair IoT network” is used. In a Fairhair IoT network, the network devices conform to the specifications laid down by the Fairhair alliance and use an internet based protocol for communication. An example Fairhair IoT network is as shown in figure 1.

![Image of Fairhair IoT Network](image)

**Fig 1. An Example Fairhair IoT Network (Figure from [27])**
The example network architecture consists of two subnetworks, with nodes in both the networks connected by a common interface. The subnetwork topology is a function of the physical interface and is independent of Fairhair. The devices in the Fairhair network contain the UDP/IPv6 stack. A Fairhair subnetwork may be constrained with regard to the available power, bandwidth or other parameters such as node memory or computing power. Such a sub network is termed a constrained Fairhair subnetwork. A Fairhair network can also contain sleepy nodes, that are battery powered and turn off their interfaces to conserve power. These nodes rely on non-sleepy nodes to act as a proxy for incoming communications. The proxying mechanism in the Fairhair network is decided by the physical interface protocol. Fairhair assumes an IPv6 network, whether it consists of IPv6/ Ethernet nodes which obtain an address via DHCPv6 or Thread nodes which obtain their addresses via prefix distribution is not relevant for Fairhair. The router in a Fairhair network could be either a Fairhair node or a dedicated router device. One or multiple servers could be connected to the network backbone for other logical network functionalities. A Fairhair network can be connected to a non-Fairhair network through a gateway which is mainly responsible for protocol translation. The gateway mainly connects a legacy network to an IP based network. The gateway functionality specification is not provided by Fairhair [27].

The Fairhair stack is as shown in figure 2. Fairhair seeks to define a framework, initially for adoption by the ecosystems: BACnet, KNX and ZigBee. The framework specifies the constrained web services for the aforementioned application ecosystems. A set of application layer services that link application protocols to an IPv6 based infrastructure are provided by Fairhair. The Fairhair application framework is located on top of the generic UDP/IPv6 service. Typically, there exists one physical interface per node. If necessary, an adaptation layer is used to interface between IPv6 and the physical layer. The Application Service sublayer provides mechanisms for application services: network management, service discovery and security. These operate on resources which are described according to a framework provided by the Fairhair resource model that is described later in the chapter. Additional ecosystem-specific services may also exist at the Fairhair Application framework level but they are not in the scope of Fairhair. Fairhair uses the services provided by the IETF CoAP protocol to interface to the UDP/IPv6 stack [27]. A generic model is assumed for the application layer, to which specific application protocols may be mapped. An application process represents the functionality of a physical component such as switch, sensor, etc. It is supported by an ecosystem-specific resource tree. The resource tree is defined in accordance with the Fairhair Resource model framework. There exist at least one or more ecosystem-specific resource trees hosted on a Fairhair device [27].
Fig 2. Fairhair Architecture Stack [34]

Resource Model

The resource model adopted by devices operating in a Fairhair IoT network is as shown in figure 2. The Fairhair resource model is a generic model for the application layer that can be applied to specific resource models used by various ecosystems such as BACnet, ZigBee and KNX. These ecosystems further apply their own restrictions on the Fairhair generic resource model to create ecosystem-specific subsets [27].
In the Fairhair generic resource model, a node is a Fairhair defined concept which describes an instance of the Fairhair protocol stack that includes the physical layer(s) as might be implemented in a physical device [27]. So, a node refers to a Fairhair device that is associated with a single UDP/ IPv6 protocol stack. In general, the term ‘node’ is used to describe an instance of the entire protocol stack including the physical layer(s) as might be implemented in a physical device. At least one device instance is supported by a node and in the case of multiple instances, each device application instance will be identified by a node-unique id. It supports the mandatory Fairhair data items and may support one or more ecosystem specific resource models. In figure 2, device refers to a ‘logical device’, a term that is introduced by Fairhair to refer to an instance of an ecosystem’s resource model. A device may contain one or more applications [27]. An application instance corresponds to the ZigBee endpoint concept. So, an application instance basically distinguishes one application from the other. An application instance consists of a device object and multiple object types. Device object is a specific object that is used to group all properties which characterize an ecosystem resource model as an entity [27]. An application may contain multiple objects of different object types and multiple object of the same object type. An object contains commands and properties. Commands are entities that perform some operation on an object or a physical resource or trigger a process that continues for a duration after the command execution. Properties convey data or metadata about a related application or a physical resource. Metadata provides additional information about a resource, e.g.: physical location of a resource, data type etc. [27].

Fig 2. Generic Resource Model of a Fairhair IoT Device (Figure from [27])

Fairhair enables discovery and addressing of objects in nodes and to perform operations including reading from, writing to and executing object resources. An object describes the properties and behavior of both physical and logical resources associated with the host physical device. An object is identified by an Object type identifier that indicates the contents and the functionality of the object. An object consists of properties and commands. Properties provide data or metadata about a related application or physical resource.
Commands perform operations on an object or a physical resource and can also set a process in motion after the command execution. A few commands can also take in input parameters and return a response to the initiator. Apart from application objects, a device application instance can also contain a device object and metadata. A device object is a special object that defines functions and properties that are relevant to network management and it is defined by the ecosystem. Metadata provides additional information about a resource such as its physical location. Some ecosystems provide additional services such as reporting and group communications. This is covered under the logical block - Other services and interfaces [27].

Operations on Fairhair objects are performed through the CoAP protocol [28]. So, the definition is directly in CoAP. Objects are mapped to URIs according to the ecosystem and implementation requirements. The following is the generic format of the URI that is used to address the objects [27]:

- \texttt{coap://NodeIPv6:port/.ecosystem\_prefix/ecosystem\_object\_path}

Here, the port number, ecosystem prefix and object path are defined at the discretion of the ecosystem.

The generic CoAP URI above may be followed by one of the following extensions [27]:

- \texttt{<URI>/propertyID \{value\}}
- \texttt{<URI>/commandID \{payload\}}

In a Fairhair network, a property represents a state while a command represents an action or a function. The specific services to read or write to multiple properties in current ecosystem protocols is replaced by CoAP operations in Fairhair. Fairhair defines a recommended mapping of the read and write services to CoAP methods, but an ecosystem may deviate (e.g., it may decide to use PUT to initiate commands instead of the recommended use of POST).

To illustrate the above mentioned generic format of the URI, the following running example of a ZigBee object is introduced:


In the above URI, the parameter ‘zcl’ stands for ZigBee Cluster Library and it indicates that the URI is used to access a ZigBee object, which is a ZigBee endpoint with endpoint #11. The URI is used to address the command present on the OnOff cluster (cluster ID: 6). The command that is addressed is the ‘toggle’ command which has a command ID: 2.

With regard to the CoAP payload format that is used in the URIs, the Fairhair network uses a CBOR encoding [29].

**Discovery**

The Fairhair discovery mechanism, which includes both device discovery (DD) and service/ object discovery (SD/ OD) is independent of the ecosystems KNX, BACnet and ZigBee to search for devices/services that match a search criterion. The Fairhair discovery mechanism includes a set of discovery methods that are derived from the IETF [30] and CORE [31] standards. The discovery mechanism is based on the following choices [32]:
• The standard CoAP discovery using Link format [33] is adapted to meet Fairhair requirements.
• Both multicast and unicast discovery are supported, where appropriate.
• Support for optional discovery through Resource Directory (RD) [34] is provided.

All Fairhair devices support discovery queries on the /.well-known/core resource [33]. The CoRE Link Format [8] content format is used to represent discovery results. A Fairhair device uses the following CoAP request format [7]:

```
GET coap://hostname/.well-known/core?attribute=value
```

Here, the host name is the IP v6 address. The response to a query consists of a unicast 2.05 CoAP [28] response with CoRE Link Format [8] payload containing the query result, as given below [32]:

```
2.05 Content (Content-Format: application/link-format (40))
<coap://[nodeIPv6Addr]/uriPath>;attr1=value1;attr2=value2,
<other link 1>,
<other link 2>
```

The Fairhair Device Discovery (DD) and Object Discovery (OD) operations are distinguished through a rule. According to the rule, if the query contains the query parameters ‘ep’ (endpoint) or ‘et’ (endpoint type) and does not have the query parameter ‘rt’ (resource type), then the query type is DD else the query type is OD. DD queries are responded with Fairhair Logical Device(s) whereas OD queries are responded with object(s) [32]. (A Fairhair logical device is a network device that conforms to the specifications laid down by the Fairhair alliance and uses an internet based protocol for communication.)

The following high-level generic discovery operation types are defined for a Fairhair IoT network [32]:

1. Multicast Device Discovery (DD-MQ)

   The queries in this operation type use CoAP multicast to discover Fairhair logical devices. Multiple types of operations are possible within this operation:

   a. Endpoint Type (‘et’)

   The below CoAP request is used to do a query on an Endpoint Type ‘et’ to query for Fairhair Logical Devices of a specific ecosystem-type or device-type:

   ```
   GET coap://[ff0x::fd]/.well-known/core?et=deviceTypeID
   ```

   The response to an ‘et’ query looks typically looks as below:

   `````
   2.05 Content (Content-Format: application/link-format (40))
   <coap://[nodeIPv6Addr]/.ecosystemPrefix>;et=deviceTypeID;ep=logicalDeviceID;rt=val
   ```
b. End Point (‘ep’)

This query is also used to discover Fairhair logical devices. The CoAP request that is used to do a query for a specific Fairhair Logical Device name, or for a Fairhair Logical Device name starting with a specific prefix string are as below:

GET coap://[ff0x::fd]/.well-known/core?ep=logicalDeviceID

This discovery operation type is useful after events such as IPv6 prefix change, IPv6 address change, or relocation of a Fairhair device to another network segment. The response to an ‘ep’ query is similar to the response to the ‘et’ request and is as below:

2.05 Content (Content-Format: application/link-format (40))

<coap://[nodeIPv6Addr]/.ecosystemPrefix>;et=deviceTypeID;ep=logicalDeviceID;rt=val

The multicast device discovery can be illustrated using the running example that makes use of the ZigBee endpoint 11 and the OnOff cluster (Cluster ID: 6):

- GET coap://[ff0x::fd]/.well-known/core?ep=11
- 2.05 Content (Content-Format: application/link-format (40))
  <coap://[2002:8290:4eed::8290:4eed]/zcl>;ep=11;rt=urn:zcl:c:6.s

The request is made to discover the Fairhair logical device- endpoint 11. The response contains the IPv6 address of the device that hosts endpoint 11. The response also contains the resource type (rt) that is available on endpoint 11 which is the ZigBee OnOff cluster in this example. However, the resource type is represented by a URN. The ‘zcl’ in the URN indicates that the target resource type is a ZigBee resource, c:6 indicates cluster ID 6 and ‘.s’ indicates that the request is directed to a server device.

The request is made to discover the Fairhair logical device- endpoint 11. The response contains the IPv6 address of the device that hosts endpoint 11. The response also contains the resource type (rt) that is available on endpoint 11 which is the ZigBee OnOff cluster in this example. However, the resource type is represented by a URN. The ‘zcl’ in the URN indicates that the request is directed to a server device.

2. Multicast Object Discovery (OD-MQ)

Object Discovery deals with discovering objects that have specific types and/ or attributes. As in the case of Multicast Device Discovery, multiple types of operations are possible within this operation. The following sub-sections consider multicast OD queries that are processed by all Fairhair devices. The following query request is used to address both Resource Directories and Fairhair nodes:
GET coap://[ff0x::fd]/.well-known/core? rt=<objectType>

The general format for a single response payload Link Format entry is as below:

coap://[nodeIPv6Addr]/.ecosystemPrefix/DeviceAppInstanceNr/ObjectTypeID, objectInstanceNr;attribute1=value1;attribute2=value2

Here, the number of attributes/value pairs may be any number of zero or more pairs. Nearly always the ‘rt’ attribute key/value pair is included, as otherwise all kind of object and device types are returned. A Resource Directory will only respond to an OD-MQ query with the resource that it hosts/owns itself, not with the contents of registered resources.

3. Unicast Object Discovery (OD-UQ and OD-UB)

a. Unicast Object Discovery (OD-UQ)

All the multicast queries for service/object discovery defined in the above subsections can be reused in unicast. The query format and also the rules for combined attributes would be the same:

GET coap://[node-IP-address]/.well-known/core?queryAttribute1=value1

The response format to the unicast query response is identical to the earlier case discussed for the OD-MQ response. However the unicast response differs from that of the multicast in two ways:

• If the query result set has zero elements, a response to the unicast query request MUST be sent with an empty Link Format document in the payload.
• In case of errors, any error responses (CoAP 4.xx or 5.xx class) MUST be generated by the responding CoAP server as defined in [RFC7252] and other parts of the Fairhair specification.

One of the request types that is related to the unicast object discovery is the ‘browse request’. Browsing is simply done by not specifying criteria on the unicast object discovery. This operation involves a CoAP unicast to the .well-known/core resource which allows regular discovery without a query component to the resources hosted on the node:

GET coap://[node-IP-address]/.well-known/core

The above request provides a list in Link Format of all the discoverable resources of the node at the top level at least and optionally having more levels deep. Such a query will return all the resources as a result of the above request, irrespective of the path depth of the resource. The response is a standard Link Format document as defined for other cases. In cases of error, any CoAP error responses (class 4.xx or 5.xx response codes) to unicast SD queries will be generated by the CoAP server in compliance with the Fairhair specification. Following is an example of Unicast Browse Object Discovery and it is derived from the running example introduced earlier:
Through a GET on the .well-known/core discovery resource:

```
GET coap://[2002:8290:4eed::8290:4eed]/.well-known/core
```

The following response is obtained:

```
2.05 Content (Content-Format: application/link-format (40))
<coaps://[2002:8290:4eed::8290:4eed]/.zcl>;rt=urn:zcl,
<coaps://[2002:8290:4eed::8290:4eed]/.knx>;rt=urn:knx;if=knx.v2,
<coaps://[2002:8290:4eed::8290:4eed]/OnOff>;rt=urn:zcl:phys:OnOff
```

The above response indicates that the Fairhair Device contains a ZCLIP Logical Device and it also contains a KNX device with API version 2 (knx.v2). It also hosts a resource or service ‘OnOff’.

### 3.2 BACnet Network

BACnet is a data communication protocol for Building Automation and Control Networks. The key characteristic of the protocol is that the rules of the protocol relate specifically to the needs of the building automation and control equipment [35].

![BACnet Collapsed Protocol Architecture](image)

**Fig 3. BACnet Collapsed Protocol Architecture [36]**

With respect to the protocol architecture, BACnet uses a collapsed protocol architecture shown in figure 3. BACnet Application and Network Layer protocols are common to all the data links. The Application layer is characterized mainly by the BACnet Objects and Services, and the network layer is characterized by the
Network Layer messages [37]. These application layer and network layer entities are discussed under the next sections- resource model and discovery respectively.

**Fig 4. BACnet Network Architecture [35]**

A typical BACnet network is shown in figure 4. It consists of sensor and actuator based BACnet devices from different vendors connected together through a common Local Area Network (LAN). The BACnet LAN could be implemented through multiple physical and data-link options such as Ethernet, ARCNET, Master Slave/ Token Passing (MS/TP) and LonTalk. Due to the unique BACnet message packaging mechanism, other type of computers such as office PCs and servers can co-exist on the same LAN without any interference [35].

**Fig 5. Resource Model of a BACnet Device (Figure from [27])**
Resource Model

The resource model of BACnet is shown in figure 5. A BACnet node/device is a collection of objects that represent the functions actually present in a given real device. These functions include analog and binary inputs and outputs, schedules and alarms. An object is a collection of related information and it consists of properties that further characterize it by describing its behavior and governing its operation. For e.g. an analog input is represented by an analog input object which has a set of properties such as present value, data (value) type, location and so on. Some of these properties are required whereas others are optional. An important property of an object is the object’s identifier which is a numerical value that enables BACnet to access it. A BACnet device is thus represented by a structured collection of standardized Objects [35]. BACnet 2012 version has about 50 defined Object types which include object types such as analog input, analog output, program and schedule. The choice of Objects that are present in a BACnet device is decided by the device’s function and capabilities. All Objects need not be present in all BACnet devices. A device that controls a Variable Air Volume box would have several Analog Input and Analog Output Objects while a Windows PC that has neither sensor inputs nor control outputs will not. Every BACnet device has a special object called Device Object. The Device Object also contains device and manufacturer information like the serial number and some descriptive information. A BACnet device is fully described by the properties of the Device Object. For instance, the Object List property of the Device Object provides a list of every object present within a BACnet device [9]. With respect to the properties contained within the BACnet objects, there can exist 123 different properties of objects. For each object type, there exist a subset of properties that are specified. A few of the properties are mandatory to be present for every object whereas a few other properties are optional. Each property is characterized by three attributes: identifier, datatype and a conformance code. Here the conformance code can have three values: R, W or O, which refer to readable, readable and writable, or optional respectively. For e.g. the object: analog input is associated with a property that has the identifier: Object_Name, datatype: character string and a conformance code: R [37].

One important entity of the BACnet resource model is services. Services provide a means for a BACnet device to acquire information from another device and initiate commands to perform actions, and perform announcements to other devices about an event that has occurred. There exist a total of 32 BACnet services and these are classified into five classes: Alarm and event services, file access services, object access services, remote device management services and virtual terminal services. The services are further labelled with either ‘C’ or ‘U’ which correspond to Confirmed or Unconfirmed services respectively. Confirmed services are the ones that involve a reply, typically with data and unconfirmed services are those that do not require a reply. The alarm and event services deal with condition changes sensed by a BACnet device, including alarms. File access services are used to read and change files in BACnet devices. The Object Access Services facilitate reading, modifying and writing properties and also addition and deletion of objects. The remote device management services provide distinct functions for remote operator control, specialized message transfer and device auto-configuration [38]. The Virtual Terminal services are intended to provide the capability to develop a virtual terminal interface to BACnet devices but however have now been superseded by the web [37]. Apart from the objects, device object and properties and services, a BACnet device consists of metadata at any level i.e. for e.g., even a single property can have associated metadata.
Discovery

Discovery in BACnet involves discovery of BACnet devices and objects. It is mainly facilitated through the protocol services: Who-Has, I-Have, Who-Is and I-Am [38]. The Who-has and I-Have services are associated with objects discovery and Who-Is and I-Am services are associated with device discovery. These services are intended to facilitate dynamic binding, which facilitates programming a device with an entity such as the name of a commonly used object without knowing where in the BACnet network the object actually resides i.e. the address of the device containing that object [37] is not (yet) known.

a. Who-Has
This BACnet service is used to identify the device object identifiers and network addresses of other BACnet devices whose local databases contain an object with a given Object_Name or a given Object_Identifier. Who-has is an un-confirmed request service and is normally broadcast. It includes ther parameters: Device Instance Range Low Limit, Device Instance Range High Limit, Object Identifier and Object Name. The receivers of the request first check if both the parameters Device Instance Range Low Limit and Device Instance Range High Limit are present and that their device instance number falls in the device instance range. If so, then the receiving device is authorized to see if its own object id or object name match those of the sending device. The query contains exactly one of these parameters i.e. it specifies the search for ID or the name, not both. If there is a match, then the receiving device responds with an I-have response [37].

b. I-Have
This BACnet service is used to respond to Who-Has service requests or also to advertise the existence of an object with a given Object_Name or Object_Identifier. I-Have is an un-confirmed service having the parameters Device Identifier, Object Identifier and Object Name. The I-Have service is always broadcast but since it is only required to reach the sender of the Who-Has, the broadcast will be link-local but can be global, if required [37].

c. Who-Is
This BACnet service is used to determine the device object identifier, the network address, or both of other BACnet devices. Who-Is is also an un-confirmed service with the parameters: Device Instance Range Low Limit and Device Instance Range High Limit. Similar to the Who-Has service, the receivers check if their device instance value lies in the device instance range specified by the sender. If so, the receiver sends an I-am response [37].

d. I-Am
This BACnet service is used to respond to the Who-Is service requests but can also be sent out any time. The network address of the initiating device is obtained from the MAC address of the PDU. The I-Am service is an unconfirmed request with the parameters: Device Identifier, MAX APDU length accepted, segmentation supported and vendor identifier. The service is normally broadcast and is mostly generated in response to Who-Is request service. The device identifier is the Device Object identifier of the device issuing the I-am. The max APDU length Accepted parameter is the Max APDU Length Accepted property value of the device’s device object. The Segmentation
Supported parameter is given the Segmentation Supported property value of the device’s device object. The Vendor identifier parameter is the code assigned to the vendor [37].

3.3 ZigBee Network

A ZigBee network is a wireless network built on the ZigBee standard technology [39] that addresses the needs of remote monitoring and control network applications. A ZigBee network is characterized as a low cost and low power solution. A ZigBee network uses the physical radio & medium access control specified by IEEE 802.15.4 standard but adds its own network, security and application layers as described by figure 7. A typical ZigBee network is as shown in figure 6. A ZigBee network consists of three types of devices: ZigBee Coordinator, router and an end device. The ZigBee coordinator is a fully functional device mainly responsible for the network initiation. The ZigBee router is also a fully functional device which mainly extends the network coverage by discovering and associating with the Coordinator. The End device is a reduced functional device that associates with the Coordinator or router [41].

Fig 6. ZigBee Network Architecture [40]

A ZigBee application stack is shown in figure 7. The application layer is composed of the application framework, ZigBee Device Object and Application Support Sub-Layer. The ZigBee Device Object and the Application framework which is composed of ZigBee endpoints and clusters are discussed under the next section- resource model. The APS sublayer and the network layer are considered for the gateway architecture analysis and design and are hence discussed in the next chapter.
Fig 7. ZigBee Application Stack [40]

Resource Model

Fig 8. Resource Model of a ZigBee Device (Figure from [27])

The resource model of ZigBee is as shown in figure 8. A ZigBee node consists of one or more ZigBee applications on a single ZigBee stack, with a single network address on a single network. Each node consists
of one or multiple endpoints- numbered between 1 and 240. Endpoints are virtual entities that facilitate different application profiles to exist within each node. For e.g. a switch that is sold for both home and commercial markets would have two endpoints, one corresponding to the home market profile and the other corresponding to the commercial market profile. Another example could be: a physical switch panel with two independent switches would also be modelled as two endpoints with the same application. Each endpoint consists of a set of cluster instances which are identified by a cluster identifier. An endpoint can at most contain one instance of the same cluster type. A cluster can be considered as a service or an object and it is the lowest independent functional entity in the model. Each cluster is characterized by a specification that specifies a set of attributes (data), commands (code) and an associated behavior. Commands cause action whereas attributes keep track of the current state of a cluster. A ZigBee application can for instance know whether a light is on or off by querying the OnOff attribute within the OnOffCluster. It can also change the status of the light from Off to On by sending an On command to the cluster to turn the light On. Clusters are the building blocks of a ZigBee application and the cluster specifications are documented in the ZigBee Cluster Library [41]. The Endpoint 0 is a special endpoint, it contains an application called ZigBee Device Object (ZDO) running on this endpoint in every ZigBee device. Through the ZDO, the endpoint 0 keeps track of the ZigBee node on and off the network. It also provides an interface to a ZigBee application profile called the ZigBee Device Profile (ZDP) for discovering, configuring and maintaining ZigBee devices and services on the network. Apart from the aforementioned key constituents, a ZigBee node also consists of dedicated blocks for other services which include reporting, binding and group communications [42] [43].

Discovery

Discovery in ZigBee supports discovery of devices, endpoints and clusters. The ZigBee device discovery is facilitated through the ZigBee Device Object (ZDO), which is the application that runs on the endpoint 0 of a ZigBee device/ node. The ZDO provides an interface to the ZigBee Device Profile (ZDP), which is a specialized application profile for discovering both ZigBee devices and services on a network. The services of the ZDP are classified into client side and server side services. With respect to device discovery, the ZDP contains a commands set to discover multiple aspects of ZigBee nodes in a network. These commands are termed as ZigBee device discovery services and all of those have the following features in common [43]:

- They provide additional information about a ZigBee node.
- They are optional from the client side but server side processing is essential.
- They apply to the ZigBee nodes in a network and not to any particular application or application profile running on an endpoint of a ZigBee node.

A few examples of the ZigBee device discovery services include NWK_addr_req and Node_Desc_req. The former is used to find the network address of a node and the latter is used to find the node type, which could be ZR (ZigBee Router), ZC (ZigBee Co-ordinator) or ZED (ZigBee End Device). Thus Device Discovery, through ZigBee Device Profile is used to discover which nodes to talk to in a ZigBee network [43].

Service Discovery in ZigBee deals with discovery of applications within ZigBee nodes/ devices. It also deals with finding instances of a specific cluster type. Service discovery is facilitated through the ZigBee Device Profile (ZDP). Similar to ZigBee device discovery services, there exist service discovery services.
A few examples of service discovery services include: Simple_Desc_req (unicast), which returns a description of the endpoint, and Active_EP_req (unicast), which returns active endpoints on a node. ZigBee service discovery involves discovering and matching endpoints. Discovering the endpoints and the corresponding services that they support is a key step in a ZigBee commissioning process since different manufacturers can have different endpoints for their applications. For service discovery, active endpoints can be located through the service- Active_EP_req. Then the service- Simple_Desc_req can be used to get the description of a particular endpoint which includes the endpoint's parameters such as application profile ID and application device ID. Once the list of active endpoints and the description of a particular endpoint is available, the service- Match_Desc_req can be used to find a matching endpoint that matches with the endpoint whose description has been fetched. This is done by matching both the profile ID and the input/output cluster lists; the profile IDs should be the same and at least one input must match one output cluster. As an example, a switch node that has an On/Off cluster (0x0006) as output matches with a light node that has an On/Off cluster as an input [43].

**Fig 9. Binding in ZigBee (figure from [40])**

The next step after discovery is binding, which provides a mechanism to attach endpoint on one node to one or multiple endpoints on another node as shown in figure 5. Binding is facilitated through a local binding table, which contains the fields source and destination endpoints, destination address, group ID and cluster ID. The table keeps track of network and MAC addresses of the participating nodes and updates itself when a ZigBee end point changes its address due to displacement from one node to the other. A few of the services that are used for binding include: Bind_req and Unbind_req which are used to bind and unbind nodes respectively [43].
Chapter 4. Analysis and Design of IoT Gateway Architecture

The first part of this chapter deals with the analysis of the possible gateway architecture options available. The analysis includes the gateway architecture diagram for each possible option, followed by a detailed discussion on the suitability of a particular option. The second part deals with the design of the gateway for a particular chosen option. The gateway architecture design includes both the gateway design and the design of the IoT and non-IoT network packets.

4.1 Analysis

4.1.1 Standard Interaction

Before considering the various possible gateway architecture options, it is important to understand the standard interaction between nodes in an IoT network. For this, we consider the three use cases of our project: (1) Discovery, (2) Switching and (3) Group creation and control. As described in chapter 2, the networks configuration considered by the project is as shown in figure 1. The Fairhair IoT network consists of nodes/ endpoints implementing client side clusters (switches) and the ZigBee legacy network consists of nodes/ endpoints implementing the server side clusters (lamps).

![Networks Configuration Diagram](image-url)
The following message flow diagrams describe how the aforementioned use cases are implemented for communication between IoT nodes in the Fairhair IoT network. These flow diagrams later serve as a reference to analyze how a particular gateway architecture option is close to or far from the standard IoT interaction. To understand the interaction between IoT nodes in the Fairhair IoT network, let us assume that the nodes: node 1, node A, node B and node C of figure 1 are all IP nodes and are now present in the Fairhair IoT network. In such a case, the interaction between these IP nodes would be as shown in figure 2.

Fig 2. Discovery in Fairhair IoT network [32]

Figure 2 considers the scenario of node 1 trying to discover all endpoints implementing the server side OnOff cluster (Cluster ID 0x0006). For this, node 1 sends a CoAP request to the well-known/core resource [33] with the query part specifying the OnOff cluster in the ZigBee namespace. The CoAP request URL consists of a GET CoAP command with the multicast address [ff03::fd]. The ‘.s’ in the URL indicates that the request is sent to a server side cluster.

In response to the request, the nodes A, B and C that implement the server side OnOff cluster respond, providing links to matching clusters, by sending a 2.05 CoAP response that contains the node address (e.g. IP address of node A), the endpoint IDs (e.g. 1), the cluster ID (6- OnOff cluster) and the same resource type.
Figure 3 considers the scenario of node 1 turning on a light point on node A. For this, node A sends a CoAP POST request to node A. The request URL contains the IP address of node A, the ZigBee entry to the resource ‘/zcl’, endpoint id, OnOff cluster ID (6) and command id for ‘ON’ (1). The characters in the URL - e, S and C indicate endpoint, server slide cluster and command respectively.

In response to the request, the endpoint 1 on node A that implements the server side OnOff cluster responds by sending a 2.04 CoAP response, indicating a change in value of the server side OnOff cluster.

Figure 4. Multicast Communication in Fairhair IoT network [44]
Figure 4 shows the group communication use case in a Fairhair IoT network. It consists of two parts: group creation and group control. The group creation method in the Fairhair IoT network/ ZCLIP is not defined yet; the discussion is open. Two options are considered: 1. Specify the multicast address in the specification. 2. The Create Group command would have a third parameter to specify the multicast address that is linked to the group id. Figure 4 considers the option 2. Thus in the group creation phase, node 1 adds similar endpoints (in this case, the end points are similar since they all contain the OnOff cluster) to a group-group 1 which is associated with a unique multicast address. This is done by sending unicast requests to the nodes. The request URL contains a POST CoAP command with the parameters: node IP address, the ZigBee entry resource /zcl, endpoint id (e.g. 1), the Groups cluster id (0x0004), the add group command id (0x0).The characters e, S and c in the URL indicate the endpoint, server side cluster and command respectively. The parameters: 0 in the JSON payload corresponds to the group id, 1 corresponds to the group name and 2 corresponds to the multicast address. These parameters are not part of the URI, but of the JSON payload. In response to the requests, node 1 receives an OK (2.04) response from nodes A, B and C which contain the similar endpoints 1, 1 and 1 and 2 respectively.

Once a group is created, the group members- similar endpoints can be controlled simultaneously through a single multicast request. From figure 4, node 1 sends out a multicast request to toggle all the group members. The request URL contains a POST CoAP command containing the multicast IP address, the ZigBee entry resource /zcl, group id (1), OnOff cluster id (0x0006), command id (0) and the toggle command parameter (2). The character ’g’ in the URL indicates that the request is made to a group. The responses to the multicast request are suppressed since the sending node will be flooded with multiple unicast responses from the member nodes of the group and also because some nodes might not respond which makes it inefficient to collect responses from all the member nodes of the group.

4.1.2 ZigBee Frame Formats

In order to analyze the various possible gateway architecture options, it is essential to first consider the structure of a ZigBee frame/ packet. Figure 5(a) provides an overview of the ZigBee application layer and network layer frames.

To communicate to a ZigBee node, it is essential that a source node provides the destination node address, source node address and a sequence number for the network layer ZigBee frame. Here the addresses refer to the 16 bit ZigBee short addresses of the nodes. The Sequence number field is a one octet length field and its value is incremented by 1 with each new frame that is transmitted by the same source. The sequence number field and source address field values taken in pair can be used to uniquely identify a frame. With regard to the ZigBee application layer APS (Application Support Sublayer) frame, a source node needs to provide the destination and source endpoint IDs, cluster ID, profile ID and APS payload. The endpoint IDs define the application running on a node. The cluster ID defines the ZigBee cluster. The profile ID defines the ZigBee Application Profile; an application profile can be considered as a domain space of related ZigBee applications and devices. The APS frame also contains a counter field- APS counter. This is an 8-bit field value and is used to prevent the reception of duplicate frames. This counter will be incremented for each new transmission. The APS counter is different from the sequence number. With regard to the APS counter, the ZigBee Application Support Data Entity (APSDE) maintains a duplicate rejection table that contains at least the source address, APS counter and timing information such that frames transmitted and received more than once are identified as duplicates and are only delivered once to the ZigBee Next
Higher Layer Entity (NHLE) [8]. With regard to the Sequence number, the ZigBee network (NWK) layer maintains the Sequence number which is initialized with a random value. The number is incremented by 1, each time the network layer constructs a new NWK frame, either as a result of a request from the next higher layer to transmit a new NWK data frame or when it needs to construct a new NWK layer command frame. After being incremented, the value of the sequence number is inserted into the sequence number field of the frame’s NWK header. Apart from being used to uniquely identify frames, the Sequence number can also be used to detect lost frames. The Sequence number, when used with the count of received beacons and data frames can be used to estimate the probability of reception [8]. The APS payload- AF (Application Framework) frame consists of: the transaction count, frame type and transaction IDs. The transaction count specifies the total number of transactions appearing in a general frame. The frame type field specifies the service type of transactions and thus distinguishes a message from a command. The transaction ID consists of two parts: transaction sequence number and transaction data. The transaction sequence number specifies an ID for the transaction so that a response can be related to the corresponding request. The transaction data field contains the data of a transaction. It can either be a message frame or a KVP (Key Value Pair) frame. A KVP frame enables an application to manipulate attribute values and is thus associated with commands. It consists of command related fields: command type IDs, attribute IDs, attribute data type, and attribute data. The message frame enables vendor specific extensions and is not relevant in the scope of interoperability.

In conclusion, the above two paragraphs could be used to determine the key parameters of a ZigBee message that need to be explicitly added by the IoT gateway which is used to connect the ZigBee legacy network and the Fairhair IoT network. The key ZigBee parameters to be added include: The ZigBee destination address, ZigBee source end point and the transaction sequence number. Since a sending IoT node sends a CoAP message to either the gateway IP address or to the IP address of one of the gateway services contained within the gateway, the gateway needs to explicitly add the destination ZigBee address that relates to the destination address used by the sending IoT node. Since a ZigBee message contains the source endpoint parameter, it needs to be added by the IoT gateway since a sending IoT node does not include it in its CoAP message. The source endpoint value added by the gateway relates to the IP address of the sending IoT node. The CoAP message sent by a sending IoT node does not include any tracking parameter to keep track of the messages- to uniquely identify messages. However, this is required in a ZigBee message. Thus, the gateway needs to keep a counter type of value for each sending IoT node. The gateway would add this value for creating the Transaction Sequence Number parameter that is essential in a ZigBee frame. The other parameters of a ZigBee message are generated by the gateway by translating the CoAP message parameters on a one-to-one basis.
Fig 5. (a) ZigBee Protocol Frame Overview [47]

Fig 5. (b) ZigBee APS Frame Structure (acc. to ZigBee Spec 2008 [48]) - includes APS Counter field
4.1.3 Discovery, Switching and Multicast in ZigBee

I. Discovery

1. [Connection, GetAddress] [Connection, GetAddress, L=11:22:33:44:00:00:00:00, S=0x0001.0]

2. [Mgmt_Lqi_req] [Mgmt_Lqi_rsp]

3. [Active_EP_req] [Active_EP_rsp]

4. [Simple_Desc_req] [Simple_Desc_rsp]

II. Switching

[DST NWK Add=0x0002, SRC NWK Add=0x000A, DST Endpoint= 11, SRC Endpoint= 64, Sequence No.= 161, Command: Toggle]

[DST NWK Add=0x000A, SRC NWK Add=0x0002, DST Endpoint= 11, SRC Endpoint= 64, Sequence No.= 161, Command: Default - Response, Status: Success]

[DST NWK Add=0x0003, SRC NWK Add=0x000A, DST Endpoint= 11, SRC Endpoint= 64, Sequence No.= 161, Command: Add Group]

{Payload: Group Id=0x0001, Length=6, String: Group 1}

[DST NWK Add=0x000A, SRC NWK Add=0x0003, DST Endpoint= 11, SRC Endpoint= 64, Sequence No.= 161, Command: Add Group Response]

{Payload: Group Status= Success, Group Id=0x0001}

[DST NWK Add=0x0004, SRC NWK Add=0x000A, DST Endpoint= 11, SRC Endpoint= 64, Sequence No.= 161, Command: Add Group]

{Payload: Group Id=0x0001, Length=6, String: Group 1}

[DST NWK Add=0x000A, SRC NWK Add=0x0004, DST Endpoint= 11, SRC Endpoint= 64, Sequence No.= 161, Command: Add Group Response]

{Payload: Group Status= Success, Group Id=0x0001}

[DST NWK Add=Multicast, SRC NWK Add=0x000A, Group= 0x0001, SRC Endpoint= 64, Sequence No.= 161, Command: Toggle]

III. Multicast

Device Discovery

Service Discovery

Group Creation

Group Control

Fig 6. Discovery in ZigBee Network
Figure 6 shows the ZigBee network configuration used in the project. The configuration contains three ZigBee nodes: node 1, 2, and 3 connected to a network bridge- HUE Bridge. Node 1 is a ZigBee node that wants to perform discovery, switching, and multicast in the ZigBee network.

Discovery in a ZigBee network consists of two parts: Device discovery- discovery of nodes and Service Discovery- discovery of services. Device discovery deals with determining the network addresses of nodes and Service Discovery deals with determining the services offered. Discovery is a four-step process as shown in figure 6. In the first step, node A makes an API call- [Connection, Get Address] to the ZigBee network bridge. In response, node A receives the long Address-L=11:22:33:44:00:00:00:00 and the short Address-S=0x0001.0 of the bridge. In the second step, node A sends the command- Mgmt_Lqi_req to the bridge. This command is used to generate the neighbor table list of the bridge. The command includes ‘index’ as an argument, apart from the destination address of the bridge. The index is the starting index of the requested elements of the neighbor table. In response to the Mgmt_Lqi_req command, node 1 receives the command- Mgmt_Lqi_rsp. This command contains the neighbor table which includes the long and short ZigBee network addresses of all the nodes that are connected to the network bridge- nodes A, B, and C. Steps 1 and 2 complete the ZigBee Device discovery.

Once the network addresses of nodes are obtained by node A, as step 3, it sends the command- Active_EP_req as a unicast request to a particular ZigBee node, e.g. node A as shown in figure 6. This command is used to discover all the active endpoints present on a node. The command uses network address of destination node and the network address of interest as arguments. In response to the Active_EP_req command, node A receives the command Active_EP_rsp. This command delivers the active endpoints on a node. Along with the active endpoint values, the key parameters that are received through the command include: sequence number, source address, and active endpoints count. With the active endpoints obtained, the node A sends the command- Simple_Desc_req as a unicast request to a particular active end point of a node which obtained in step 3. This command contains the argument- active endpoint, apart from address of destination node and network address of interest. In response to the Simple_Desc_req command, node A receives the command- Simple_Desc_rsp. This command mainly contains the list of cluster ids and the profile id. For e.g.: the presence of the cluster id: 6 (OnOff cluster) and the profile id: 49246 (ZigBee Light Link- ZLL) helps to conclude that the ZigBee node/ device is an OnOff light point. The other contents specific to this command include the parameters: device version, device id, no. of in clusters and no. of out clusters. Service discovery- determining the services offered in ZigBee is achieved through determining the cluster id and profile id. Thus steps 3 and 4 constitute the ZigBee Service Discovery.

Switching in a ZigBee network is simple and an example of it is as shown in figure 6, where in node A tries to toggle the state of node A. It does this by sending a switching request command that contains the key ZigBee parameters- destination and source network addresses and endpoints, sequence number, and the command. In response, node A receives the command with the aforementioned parameters and status. But the command value in the response command is Default Response. Also, the sequence number in the response command is same as that in the request command. This guarantees that the request and the response commands match with each other.

Multicast in a ZigBee network consists of two stages- group creation and group control. An example of multicast is shown in the example of figure 6. In the example node A tries to add node 2 and node 3 to a group with Id=0x0001 and name: Group 1. As part of group creation, node 1 sends an Add Group command to node 2. The command contains the ZigBee parameters- destination and source addresses and endpoints, sequence number, command: Add Group and a payload that contains: group Id, group name and length of the group name string. In response to the Add Group request command, node 2 sends a response command that contains the aforementioned ZigBee parameters and a Group status.
Success payload parameter, indicating that node 2 is successfully added to group 1. The Group Add request command is repeated for node 3, to add it to Group 1. This completes the Group Creation stage. The next stage is the Group Control stage where in Group control command such as a toggle command is sent to Group 1. The parameters in this Group toggle request command are same as those present in the switching request toggle command except that, the destination network address and destination endpoint fields have the values of multicast address and group Id respectively. This completes the Group Control stage and hence the Multicast use case.

### 4.1.4 Gateway Architecture Options

The gateway architecture options are chosen, primarily from paper [11] according to which the services offered by a sensor network such as a ZigBee legacy network constitute the gateway services. The interaction between the participating IoT network and the non-IoT network occurs through the gateway services. Gateway services can be of multiple types. A gateway service can be characterized by two features. Firstly, what does the gateway service represent? The possible options with respect to this feature include, whether the service offered is per node or per service or per network. Secondly, what is the granularity at which the service is exposed by the gateway? Which could be either an interface to the node, an interface to the service or an interface to the entire network. Table 1 briefly presents the possible gateway architecture options.

<table>
<thead>
<tr>
<th>Gateway Service</th>
<th>Interface Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per Node</td>
<td>Node/Service</td>
</tr>
<tr>
<td><strong>Target:</strong> Implicit</td>
<td><strong>Interface:</strong> Fixed, one per node</td>
</tr>
<tr>
<td><strong>Functionality:</strong> Message passing</td>
<td><strong>Functionality:</strong> Message passing</td>
</tr>
<tr>
<td>Per Service</td>
<td>Node/Service</td>
</tr>
<tr>
<td><strong>Target:</strong> Explicit</td>
<td><strong>Interface:</strong> Specific, one per service</td>
</tr>
<tr>
<td><strong>Functionality:</strong> Message passing</td>
<td><strong>Functionality:</strong> Message passing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gateway Service</th>
<th>Interface Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per Node</td>
<td>Service/Service</td>
</tr>
<tr>
<td><strong>Target:</strong> Implicit</td>
<td><strong>Interface:</strong> Specific, many per node</td>
</tr>
<tr>
<td><strong>Functionality:</strong> Message passing</td>
<td><strong>Functionality:</strong> Message passing</td>
</tr>
<tr>
<td>Per Service</td>
<td>Service/Service</td>
</tr>
<tr>
<td><strong>Target:</strong> Implicit</td>
<td><strong>Interface:</strong> Specific, many per node</td>
</tr>
<tr>
<td><strong>Functionality:</strong> Message passing</td>
<td><strong>Functionality:</strong> Message passing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gateway Service</th>
<th>Interface Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per Node</td>
<td>Network/Node</td>
</tr>
<tr>
<td><strong>Target:</strong> Implicit</td>
<td><strong>Interface:</strong> Fixed, one per node</td>
</tr>
<tr>
<td><strong>Functionality:</strong> Message passing</td>
<td><strong>Functionality:</strong> Message passing</td>
</tr>
<tr>
<td>Per Service</td>
<td>Network/Service</td>
</tr>
<tr>
<td><strong>Target:</strong> Implicit</td>
<td><strong>Interface:</strong> Specific, many per service</td>
</tr>
<tr>
<td><strong>Functionality:</strong> Message passing</td>
<td><strong>Functionality:</strong> Message passing</td>
</tr>
</tbody>
</table>

Table 1. Possible Gateway Architecture Options [11]
1. Node per Node approach

**Fairhair IoT Network**

**GATEWAY**

**ZigBee Legacy Network**

**Network Address Translation Table**

| IP Address A' | Zigbee Address A |
| IP Address B' | Zigbee Address B |
| IP Address C' | Zigbee Address C |
| Zigbee Address 1' | IP Address 1 |
| Zigbee Address 2' | IP Address 2 |

Fig 7. Node per Node Approach
In this approach, the gateway service that is offered is per node and the functionality that is offered is an interface to the node. The gateway thus maintains a network address translation table that translates addresses of nodes of the two networks. The target entity in this approach is implicit which refers to the fact that the gateway represents the devices in the legacy network as IP addressable IoT devices and vice versa. Thus, the node addresses are part of the requests/ responses of the communication. The interfaces are fixed and there is one interface per node. The type of functionality that is offered is message passing, where in processes running on two different nodes of different networks can exchange messages (functions, structures and data packets) [11].

Use Case i) Discovery

Fig 8. Sequence Diagram: Node per Node Approach- Discovery Use Case
Use Case ii) Switching

Fig 9. Sequence Diagram: Node per Node Approach- Switching Use Case

(Continued.)
Use Case iii) Multicast

Fig 10. Sequence Diagram: Node per Node Approach - Multicast Use Case
With this gateway architecture option, the nodes alone are addressable and not the services. This architecture option provides message passing type of functionality. The gateway maintains a NAT (Network Address Translation Table) to translate between IP and ZigBee network addresses. Because of this, the message payload that is sent by an IoT node towards a ZigBee network needs to contain all the key ZigBee parameters that are required in a ZigBee frame i.e., the payload needs to contain the ZigBee APS frame. The key ZigBee parameters include: ZigBee destination and source endpoints, cluster id, APS counter and command payload. The parameters- group id and group name are also included in the case of multicast communication. With this gateway architecture option, there is no message translation. So, the content of the payload remains unchanged. Message passing with this architecture option only works, if the legacy ZigBee nodes/ devices can be updated with a function to process CoAP messages. But in our case, message passing only allows network address replacement (Network Address Translation- NAT). Thus a sending IoT node needs to implement the ZigBee Cluster Library (ZCL) in order to send the aforementioned ZigBee parameters that are required for constructing the message payload. The IoT node also needs to keep track of the transaction related information such as Transaction Sequence Number and Sequence number which are not essential in a standard IoT communication. These requirements impose higher resource requirements on the sending IoT nodes wrt resources such as memory and computation power. Besides this, the addition of aforementioned parameters makes the embedded IoT devices complex and increases the implementation effort with respect to development and testing. However, this architecture makes the gateway simple to implement even if the number of devices in the participating networks increases. This is because the gateway needs to maintain a NAT table alone, which is not too difficult for a computationally powerful device such as a gateway.

2. Service per Node approach

![Fig 11. Service per Node Approach](image_url)
In this approach, the gateway service offered is per node but the functionality that is offered is an interface to the service. The gateway still maintains a network address translation table to translate between the node addresses and also an additional mechanism to map the services to the associated nodes; the services are determined from the URI. Since this approach is also a per node approach the target is implicit. Unlike the node per node approach, the interfaces are specific and there exists multiple interfaces per node. Thus, the gateway consists of functional blocks which correspond to the services offered in the networks. The type of functionality that is offered is calling and subscribing where in processes running on one node can invoke routines present on other nodes and can also subscribe to messages/ events of other nodes [11].

Use Case i) Discovery

![Sequence Diagram: Service per Node Approach- Discovery Use Case]
Use Case ii) Switching

Fig 13. Sequence Diagram: Service per Node Approach - Switching Use Case
Use Case iii) Multicast

Fig 14. Sequence Diagram: Service per Node Approach - Multicast Use Case
With this gateway architecture option, the gateway will maintain a node-service pair table that maps an IP address - IoT endpoint pair to a ZigBee network address - ZigBee endpoint pair. The table thus helps to route an IoT message to the right ZigBee node and ZigBee endpoint. The gateway also maintains another table to map a source IP address of an IoT node to a suitable ZigBee source endpoint value that is required in a ZigBee frame. For this, the source IoT node needs to send its IP address as part of the CoAP request sent to the gateway. Also, a typical ZigBee packet would have transaction information in the form of the sequence number, APS counter and transaction sequence number all of which have the purpose of uniquely identifying a ZigBee packet. Because of this, an IoT node needs to maintain counter type variables to keep track of such transaction information. Also, each gateway functional block is associated with a specific node, endpoint and a cluster. Thus a gateway functional block is exposed as a node-service pair which presents as an overhead to the gateway. However, this is acceptable since the gateway is a relatively more computationally powerful entity. The approach reduces the resource requirement on IoT nodes which enables the IoT nodes to communicate with the gateway as they would do with any other resource constrained IoT node. This architecture option does not increase the complexity of the gateway with increase in network nodes; when the number of nodes increases, the gateway does not need any extra parameters to facilitate the mapping from one network to the other. Thus it scales well with increase in services.

3. Service per Service Approach

![Fig 15. Service per Service Approach](image)
In this approach, the gateway service offered is per service and also the functionality that is offered is an interface to the service. Since the approach is a per service approach, the target is explicit i.e. the target node has to be explicitly specified. Thus, an additional interface for node discovery and an interface for discovering which services are supported by which nodes is essential. The interfaces are specific and there exists one interface per service. Thus, the gateway consists of functional blocks such that each block corresponds to one particular service. The type of functionality that is offered is calling and subscribing.

Use Case i) Discovery

![Sequence Diagram: Service per Service Approach- Discovery Use Case]

Use Case ii) Switching

![Sequence Diagram: Service per Service Approach- Switching Use Case]
Use Case iii) Multicast

Considering the discovery use case, it is not the ZigBee nodes that answer to the discovery request but it is the gateway that responds. There is only one response that is received. For Switching use case, the request is sent to the gateway. The destination endpoint value in the request is defined by the gateway (e.g. 100 as in figure 17). The payload contains the source IP address of the sending IoT node and destination ZigBee node as arguments. For the multicast use case, group creation is difficult; there is no service for group creation and hence it has to be created. Then, through the group creation service, end points can be added by calling the group service URIs three times as shown in figure 18. However, group control is simple as shown in the figure.

With this gateway architecture option, a sending IoT node needs to explicitly specify the target node since the gateway service offered is per service and also the interface level exposed is at a service level. Due to this, the gateway functional block that receives the request from the IoT device communicates it to the ZigBee node since the ZigBee node would have subscribed to the gateway functional block’s service. This
eases the service discovery in a network. However, the correspondence between the gateway service offered and node presence is lost and thus nodes can be discovered by querying the service through an additional API. There will be a variation in the number of commands sent by the IoT nodes all the time and they need to keep track of status the nodes, such as the services provided by the nodes etc. Although the gateway keeps track of this kind of information, the IoT nodes also need to know it since they need to specify it in the URI. This adds more complexity for the IoT nodes than that exits with the Service per Node option. The Service per Service gateway architecture option scales well because the gateway will just have functional blocks equal to the number of service types in the network but as mentioned before, IoT nodes need to be adapted to discover nodes and specify target of commands explicitly which implies change in behavior protocol of IoT nodes.

4. Node per Network Approach

![Fig 19. Node per Network Approach](image-url)
In this approach, the gateway service offered is for an entire network and the interface level that is exposed is at the node level. Since the approach is a per network approach, the target is explicit i.e., the target nodes need to be specified through a scheme such as a numbering scheme. Thus, an additional interface for node discovery is essential. The interface offered is single and fixed. Thus, the gateway consists of only one functional block which corresponds to an entire network. Since this approach offers a node interface, the type of functionality offered is message passing [11].

Use Case i) Discovery

![Sequence Diagram: Node per Network Approach - Discovery Use Case](image)

Use Case ii) Switching

![Sequence Diagram: Node per Network Approach - Switching Use Case](image)
Use Case iii) Multicast

With this gateway architecture, the gateway has a network node number table that contains node numbers and their addresses. The table is used by the gateway for node discovery. This architecture option is scalable since there exists only one gateway functional block for an entire network, irrespective of the number of nodes in the network. However, an additional interface such as the network node table is essential for node
discovery. The node per network architecture option is similar to the node per node architecture but it has more drawbacks since it is a per network approach and the key drawback is that of explicit addressing; the IoT nodes need to explicitly specify the target nodes which makes the IoT nodes complex. Thus, this architecture option is not suitable for our project.

5. Service per Network Approach

In this approach, the gateway service offered is for an entire network and the interface level that is exposed is at a service level. Thus the target is explicit and interfaces for both node discovery and node-service mapping are essential. The interfaces provided are specific and there exists one interface per service, with an associated node discovery interface and a node-service mapping interface. The type of functionality that
is offered is calling and subscribing [11]. This gateway architecture option of Service per Network is similar to the Service/Service option but the difference is that there is a node-service map table.

**Use Case i) Discovery**

![Sequence Diagram: Service per Network Approach - Discovery Use Case](image)

**Use Case ii) Switching**

![Sequence Diagram: Service per Network Approach - Switching Use Case](image)
Use Case iii) Multicast

**Fig 26. Sequence Diagram: Service per Network Approach- Multicast Use Case**
For the discovery use case, queries are created by the gateway beforehand to find the nodes and services in the ZigBee network. Also, Number table is built based on the discovery information and it is used in the other two use cases of the project - Switching and Multicast. With regard to the IoT/CoAP response that is received by the IoT node- Node 1, there exists only one service instance of a particular type. E.g.: there exits only one service instance of type OnOff when Node 1 tries to discover the OnOff service. For the Switching use case, the IoT request uses the service type in the URI and the node number is part of the payload. The node number is derived from the node number table that is built in the discovery process. For the multicast use case, in the group creation stage, the IoT request that is sent is addressed to the gateway and node IDs are included as parameters in the payload. This is because nodes do not have IP addresses in this architecture option and also there exists one instance of Add Group service on the gateway. In the Group control stage, the OnOff service that is used in the Switching use case is used but the group Id is used as target. So, the group Id and the group address are included as payload parameters.

With this gateway architecture, the gateway has to maintain a node-service map, only if it needs to provide fast feedback on error/valid requests. Apart from this, the gateway has to have a mechanism such as a network node mapping table as used by the node per network gateway architecture. Although this architecture facilitates easy service discovery, it comes with an overhead of additional interfaces required for node discovery and node service mapping. The overhead also arises due to changes in function calling in IoT nodes due to extra parameters. Also, the IoT nodes need to know which node provides which services. Besides this, the gateway has to translate many parameters such as destination endpoint, sequence no., and payload which is challenging.

6. Network per Network Approach

Fig 27. Network per Network Approach
In this approach, the gateway service offered is per network and the interface that is exposed is an interface to the entire network. Since this approach exposes an entire network as one gateway service, the target is explicit. This provides an API (Application Programming Interface) for manipulating and accessing the network as a whole. The API provided has two parts; the first part is a loader interface to add, remove, load and unload programs and subscriptions into a network. The second part of the API is application specific and exposes only those parts of services which are required to interact with the application as a whole such that, all the interfaces of the network are hidden. Thus, the interface offered is fixed and specific, and the type of functionality that is offered is of loader and application type [11].

With this gateway architecture option, the loader interface would be essentially a script running on the gateway that expects a whole ZigBee packet and it will determine what is to be done with the ZigBee packet. So, the loader is expected to dynamically load services into the network. But that is not possible in a ZigBee network since all the functions in a ZigBee network are prescribed. E.g. if a ZigBee network contains only luminaries, it cannot be requested to control a HVAC. Thus, a loader does not work. So, with just the Application Specific blocks remaining, it becomes a service per service architecture. Also, with this architecture option, there is no good interaction between the IoT and the ZigBee worlds as the gateway needs to be provided with the whole application that has to be run on the ZigBee network. What is expected for our project is a single interface that executes a single or a set of ZigBee commands. Hence, this gateway architecture option is not applicable for our project. In view of this, the message sequence diagrams of the project use cases are dropped for this gateway architecture option.

4.2 Design

From the analysis of the six gateway architecture options in the above paragraphs, it can be concluded that the potential architecture options for our project are: node per node, service per node and service per service. The other three architecture options: node per network, service per network and the network per network are limited by impediments. These ‘per network’ architecture options imply giving a single interface for the entire network. This means the entire functioning of that network has to be understood; to discover, send all messages etc. Thus, these three options are worse than even the node per node option which provides just message passing. The options also do not make it possible for the gateway to inform the sending IoT node about the count of the ZigBee nodes in the ZigBee network.

In the service per node approach, although the resource requirement on the gateway is increased whereas the resource requirement on the IoT nodes is reduced. This is acceptable since the gateway is relatively more computationally powerful device when compared to the IoT nodes and thus can cope up with the increase in resource requirement. When the number of nodes in the network goes up, the complexity of the gateway does not increase. The gateway has no scalability concerns with the increase in services as it scales well. The service per node approach resembles the closest to a standard IoT communication with the least overhead on the gateway. The overhead arises since the gateway functional blocks have to be represented as a node-service pair. But as mentioned in the above paragraphs, this is acceptable since the gateway has a higher computational power as compared to nodes in the networks. The service per node approach also preserves the correspondence between the node presence and the gateway service presence. This makes the service per node approach the most suitable gateway architecture for the IoT gateway. The design of the gateway based on this architecture is presented below.
Fig 31. Service per Node Approach Design
4.2.1 Discovery

![Diagram showing the process of Device Discovery and Service Discovery using CoAP packets and ZigBee packets.]

**Fig 32. Sequence Diagram: Service per Node Approach - Discovery Use Case**
This use case considers the scenario of node 1 sending a discovery request to find out all the OnOff type devices; those that contain the OnOff cluster.

From figure 32, node 1 sends a broadcast GET CoAP request to the gateway. The request is sent to the single, well-known/core Discovery interface functional block on the gateway and hence in response, there is one corresponding response received by node 1. The request query contains broadcast IP address- ff03://fd, the target- URI “/.well-known/core” and a query part containing the URN (Uniform Resource Name), the ZigBee entry point resource- /zcl and the cluster id: 6 corresponding to the OnOff cluster. The well-known relative URI is a default entry point for requesting the links about resources hosted by a server. The gateway performs a translation on the received request by performing a lookup in the Node-Service pair lookup table to determine the corresponding ZigBee network broadcast address. The gateway then sends a Discovery request through 4 ZigBee packets to the ZigBee network. Discovery in the ZigBee network is achieved through the four ZigBee discovery commands as described in section 4.1.3. Thus, there is a one-to-four commands correspondence between the IoT discovery request and the ZigBee discovery request. The key relevant parameters determined through the four ZigBee discovery request and response commands include: neighbor table list, active endpoints and the ZigBee Simple Descriptor. The neighbor table list provides the network addresses of all ZigBee devices (Device Discovery). The Active Endpoints and the Simple Descriptor together determine the device type (Service Discovery). Knowing the Active Endpoints enables to send Simple Descriptor requests to the Active Endpoints. The ZigBee Simple Descriptor determines the device profile id and cluster list. In our application profile, all the devices have the profile id: 49246, indicating a ZigBee Light Link (ZLL) profile. Since the discovery request in our scenario tries to find out devices having the OnOff cluster, the ZigBee devices that contain the cluster 0x0006 (OnOff Cluster) alone respond to the request, which thus achieves service discovery.

There are two possible options for the discovery request sent by an IoT node: option #1- the gateway responds and option #2- the ZigBee nodes: A, B and C respond. The option #2 of the individual ZigBee nodes responding is a theoretical option and is possible if the nodes have their own IP addresses. But in our project setup, there is only one IP addressable device, which is the gateway (one PC/ host). Therefore, option #1 has been chosen as the design choice for our project. Thus, the gateway responds to the discovery request sent by an IoT node. The discovery response contains the field values: status, network addresses of interest, active endpoints and the simple descriptor- profile id and matching cluster ids. If a match is found for the discovery request, then the status field in the response is set to SUCCESS and the network address of interest field is set to the address of the matching ZigBee node(s). Also, the parameter- match length field in the response is set to the number of simple descriptors that matched the criteria. This match list field will contain an ascending list of the endpoints on which the simple descriptor matched the criteria. For the return/ ACK communication, the gateway determines the destination IoT node which sent the discovery request by performing a node-service pair table lookup. It translates the ZigBee discovery/ match descriptor responses to corresponding IoT (CoAP) acknowledgements- CoAP URIs by adding a suitable CoAP response code. The translated CoAP ACK frame is sent to the Fairhair IoT Node 1 and this confirms that the discovery operation has been successful. Additional details of the actual implementation of the Discovery use case are described in the next chapter- Chapter 5. Implementation.
4.2.2 Switching

This use case considers the scenario of a switch on node 1 turning ON an OnOff lamp on node A. From figure 33, node 1 sends a POST coap request to the gateway which is received by the gateway functional block node A’ OnOff. The IoT/CoAP request contains the destination IP address of the gateway functional block, source address of node 1, the destination endpoint id- e/1, the server side cluster- s/6 and the attribute value for turning ON a lamp- c/1. Since node A’ OnOff is of IoT type, it interprets the CoAP message readily. But, to send the received request across to the actual destination ZigBee node, it performs a translation on the received request. It performs a look up in the Node-Service pair look up tables to find the destination ZigBee endpoint and cluster. The tables mainly map between IP addresses and ZigBee network addresses. Since a ZigBee application frame is required to have the values- Frame Control field, Sequence number and Source endpoints, these are added by the gateway to produce the ZigBee packet, which is sent by node 1’ which acts as a virtual sender. Since the aforementioned values that are added by the gateway keep changing with parameters like message instance, operation to be performed and endpoints, the gateway keeps track of those.

Fig 33. Sequence Diagram: Service per Node Approach- Switching Use Case
The gateway performs translation using look up tables and determines the destination ZigBee short address-address of node A, the endpoint id- 1 and the cluster id- 6. ZigBee Short address is a 16-bit address that is assigned to a node when it joins the ZigBee network. Thus, along with these parameters, the ZigBee packet contains the aforementioned parameters listed in the above paragraph: frame control field- 01 that corresponds to a cluster specific, client-to-server and a default response type of communication. It also indicates that the frame type is a command frame and that the frame will be delivered in a normal unicast delivery mode. The ZigBee packet also contains the transaction sequence number- A1, and the command id- 01 which corresponds to ON command. The transaction sequence number is an eight bits field value and specifies an identification number for a ZigBee transaction so that the response command frame can be related to the request frame. An eight bit counter value can be maintained and copied into this field. For every command sent, this counter needs to be incremented by one. When the counter reaches a value of 0xFF, the next command can restart the counter from 0x00. When a device has sent multiple commands, the transaction sequence number field can be used to match incoming responses to the multiple commands issued by the device. Once the node 1’ gateway functional block receives the ZigBee packet sent from the node A’ gateway block, it delivers the received packet to the OnOff cluster on the endpoint 1 of the ZigBee node A which turns the lamp ON.

The translation process in the Service per Node gateway architecture based design is achieved mainly through two design lookup tables: Design Lookup Table #1 and Design Lookup Table #2. The former maps destination IP addresses to destination ZigBee network addresses and the later maps source IP addresses to source endpoints. There exist two options for the design lookup table #1. These design table options are provided in tables 2 and 3. In option #1, there is a mapping of the IP addresses of Fairhair IoT side gateway functional blocks to ZigBee network addresses of ZigBee nodes in the ZigBee network. E.g. the IP address of the Fairhair IoT side gateway functional block A’ maps to ZigBee network address of the ZigBee node A. In Option #2, there is a mapping of the Gateway IP address-IoT Endpoint combination to a ZigBee network address-ZigBee endpoint combination. E.g. the Gateway IP address-IoT endpoint 1 combination maps to the ZigBee network address of Node A-ZigBee Endpoint 10 combination. Although option #1 is more correct, option #2 has been chosen for the design and implementation since the project setup has only one IP addressable device- the gateway (PC/ host). In the Design Lookup table #2 shown in table 4, there is a mapping between the IP addresses of IoT nodes (source IP addresses) and the source ZigBee endpoint values that are added by the gateway to create the ZigBee message. E.g. the IP address of source IoT node- node 1 maps to a unique ZigBee source endpoint value=11.

<table>
<thead>
<tr>
<th>[IP Addr of Gateway Functional Block]</th>
<th>[ZigBee NWK Addr of ZigBee Node]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[IP Addr of Block A’]</td>
<td>[ZigBee NWK Addr Node A]</td>
</tr>
<tr>
<td>[IP Addr of Block B’]</td>
<td>[ZigBee NWK Addr Node B]</td>
</tr>
<tr>
<td>[IP Addr of Block C’]</td>
<td>[ZigBee NWK Addr Node C]</td>
</tr>
</tbody>
</table>
For the Switching use case of the project, the translation of an IoT message to a ZigBee message through the Design lookup tables #1 and #2 can be well understood by from figure 34, which uses the same IoT and ZigBee messages used in the message sequence diagram of figure 33. In figure 34, the parameters in color re part of the translation process where as the non-colored parameter values are added directly by the gateway to create the destination ZigBee message.
The ZigBee node A in turn sends a ZigBee acknowledgement to the gateway functional block node 1’ along with the same sequence number that is used in the request frame. With a node-service pair table lookup and the sequence number, the node 1’ determines the destination IoT node. The sequence number is used to map a request to a response. It then translates the acknowledgement (ACK) frame from the ZigBee format to the IoT frame format (CoAP) by adding a suitable CoAP response code. The translated ACK frame is sent to the gateway functional block node A’ OnOff, which in turn delivers the ACK frame to the Fairhair IoT node 1 and this confirms that the switching operation has been successful.
4.2.3 Multicast

![Sequence Diagram]

**Fig 35. Sequence Diagram: Service per Node Approach- Multicast Use Case**
This use case considers the scenario of a switch on node 1 turning on the OnOff lamps in a Group G1. From figure 34, the multicast communication consists of two parts: group creation and group control. In the group creation phase, node 1 adds similar endpoints (in this case, the end points are similar since they all contain the OnOff cluster) to a group-group 1. This is done by sending unicast requests to the gateway. So, if A and B are two IoT side gateway functional blocks with different IP addresses, each of them would be sent a unicast request to their unique IP addresses. But, this holds true only for the analysis part where IP addresses of A and B are considered differently. For the design part, because of implementation constraints, IP address of A is equal to that of B and that is equal to the IP address of the gateway. The request URL contains a POST CoAP command with parameters: gateway functional block IP address, the ZigBee entry point resource/zcl, the endpoint id (e.g. A.1), the cluster id corresponding to the groups (0x0004), the add group command id (0x0). The parameter 0 in the JSON payload in the request URL corresponds to the group id and 1 corresponds to the group name. The request is sent to the gateway functional block node A’ OnOff, which performs a Node Service pair table lookup and forwards it to the gateway functional block node 1’ OnOff to translate it to a ZigBee discovery request. The translated ZigBee discovery request contains the ZigBee network address of node A and the GroupAdd(G1) command. The GroupAdd command with the argument G1 creates a new group G1 and adds the endpoint 1 of node A to G1. The node 1 then sends similar CoAP requests to add endpoints of node B and node C (containing the OnOff clusters) to the group G1. These requests are translated to corresponding ZigBee requests as explained above. But, the ZigBee requests now contain the command- GroupsCommandAddGroupRequestSend(G1) instead of the command- GroupAdd(G1). This is because the group G1 has already been created. Thus the former command is associated with the functionality of adding destination endpoints to the group G1, which is specified as the payload of the command. The forwarding of group creation commands to the actual nodes is an implementation choice.

Group control, as explained in the standard IoT communication deals with simultaneous control of group members- similar endpoints through a single multicast request. From figure 34, node 1 sends out a multicast request to turn ON all the group members. The request URL contains a POST CoAP command containing the multicast IP address, the ZigBee entry point resource /zcl, group id(1), OnOff cluster id (0x0006), command id (0) and the On command parameter (1). The group request is indicated by the character ‘g’ in the URL. As, in the case of group creation, the IoT/CoAP multicast request is translated to the corresponding ZigBee multicast request through the node-service pair table lookup and forwarding of the request to the ZigBee network side gateway block. The ZigBee multicast request frame contains the network address of Group=1, destination endpoint=A.1, cluster id=6, the source endpoint and the payload=01A101. Since a ZigBee application frame is required to have the values- Frame Control field, Sequence number and Source endpoints, these are added by the gateway functional block node A’ before it forwards the request query to ZigBee network side gateway block node 1’ to produce the ZigBee packet. Since the aforementioned values that are added by the gateway keep changing with parameters like message instance, operation to be performed and endpoints, the gateway keeps track of those- sequence number and source endpoint. The payload 01A101 contains three parts: Frame Control field=01 indicating a cluster specific communication, transaction sequence number= A1, and the command id= 01 which corresponds to ON command. Once the node 1’ OnOff gateway functional block receives the translated ZigBee request sent from the node A’ gateway block, it delivers it to the OnOff cluster on the endpoints (members) of group G1 which turns on the lamps on all the member nodes of group G1.
In response to the request sent by the ZigBee network side gateway functional block node 1’ OnOff, it receives a unicast ZigBee acknowledgement from group members i.e. endpoints A, B, C1 and C2, which contain the OnOff cluster. The acknowledgement frames are received with a transaction sequence number that is same as the one used in the request frame. With the node–service pair lookup table and the transaction sequence number, the node 1’ OnOff determines the destination IoT node. But, CoAP multicast allows suppression of ACK since it is typical to not send them, as it is unknown how many devices listen on the multicast.
Chapter 5. Implementation

Service per node is the architecture that has been chosen for the IoT gateway after analyzing multiple other options and its design was presented in the previous chapter. This chapter deals with the implementation of the design. It describes the implementation tools used, the implementation options considered and the implementation workflow.

5.1 Implementation Tools

5.1.1 ZCLIP Tool: For Fairhair IoT Network Implementation

Only one implementation option was considered and selected for implementing the Fairhair IoT network since the Fairhair Alliance and its specifications are new. The Fairhair IoT network has been implemented using Philips’ in-house tool called the ZCLIP (ZigBee Cluster over IP) tool. The ZCLIP tool structure, as shown in figure 1 contains three key elements: A test framework, a ZCLIP Client instance and a ZCLIP Server instance. The test framework has an HTML interface which is described later in figure 2. Each ZCLIP Client/Server instance consists of ZCL endpoint(s) and cluster(s) which in turn contain command(s) and attribute(s). The ZCLIP client and Server instances communicate with each other through CoAP, and with the HTML interface through a Command Line Interface (CLI).

The ZCLIP tool contains an HTML client (controller) which facilitates the control of both the ZCLIP Client and Server. The HTML interface of the tool is as shown in figure 2.
The following are the key parts of the ZCLIP tool’s HTML Interface [50]:

1. **Control Client Configuration**: sets the configuration for the control client which includes options such as:
   a. JSON input field for inputting JSON text
   b. Action radio buttons for choosing CoAP actions: GET/PUT/POST/DELETE
   c. Client and server ports
   d. IP address field, to which client sends commands
   e. Resource: The ZCL resource URI
   f. Filter: an optional query string, applied to the URI
   g. Send command button: To send commands to the server

2. **Control Client Output**: It is the raw output of the control client.

3. **Server Output**: It is the output of the server when started.

4. **Server/client configuration**: It is a configuration option to configure the port, endpoints and clusters for the client and server.

5. **Client output**: It is the output of the client when started. It displays output (information) when the client is initialized. It is different from the Control Client Output window since the Control Client Output is used to display CoAP responses received by the client.

6. **Tests**: These are the tests that can be run to test the communication between the ZCLIP Client and Server, with different CoAP commands and different combinations of ZCLIP endpoints and clusters. These tests are just automated client request sequences (Test scripts: java script); the same test could be executed by configuring the control client manually.
5.1.2 HSE Tool: For ZigBee Network Implementation

The implementation option that was selected for implementing the ZigBee Legacy network was another tool used within Philips, called Hue System Emulator (HSE). It was decided not to use a test setup consisting of real ZigBee devices because of multiple reasons: Use of real ZigBee devices would require a ZigBee dongle with drivers which can integrate easily with the ZCLIP application but is not readily available. It would also require support in the test application for bootstrapping a ZigBee network or any other alternate option to do that. Although the option of using real ZigBee devices would enable performance measurements on the solution, it wouldn’t have provided more insights in terms of functionally validating the design because the HSE tool also verifies the functional correctness of the ZigBee messages.

The HSE tool is mainly developed for testing the Philips wireless lighting system - the Philips HUE [52]. The tool structure, as shown in figure 3 contains: a dashboard block, the emulator block and a proxy block. The HSE tool simulates a ZigBee network containing different types of ZigBee devices such as sensors, luminaires such as On/Off luminaire, and level control luminaire. These form the ZigBee nodes of the ZigBee legacy network. An external application, such as a gateway can be interfaced with the HSE tool through its ZigBee Light Link (ZLL) interface [51].

![Diagram of HSE Tool Structure]

Fig 3. Hue System Emulator (HSE) Tool Structure

The HSE tool HTML interface is the HSE Dashboard and is as shown in figure 4. It shows the different commissioned ZigBee luminaires and the Hue Bridge. ‘Commissioning’ is used in Lighting to refer to the binding of sensors, switches and other control devices to luminaires and other output devices. An XML file may define some bindings (like groups) but is primarily used to define which ZigBee devices (quantity and type) are present in the virtual network. The commissioned luminaires are also shown through a CLI window when the HSE tool is started. This is as shown in figure 5. Clicking on a particular ZigBee luminaire displays its various properties and values such as state, type and address.
Fig 4. HUE System Emulator (HSE) Tool HTML Interface - HSE Dashboards

Fig 5. HUE System Emulator (HSE) Command Line Interface
5.2 Implementation Details

The implementation details can be discussed under two headings: implementation overview and implementation workflow. The implementation overview describes the implementation setup and the implementation workflow describes the implementation of the gateway application.

5.2.1 Implementation Overview

![Project Implementation Diagram]

_Fig 6. Project Implementation Diagram_

Figure 6 shows the project implementation setup that mainly contains three key blocks: The Fairhair IoT network, the Gateway and the ZigBee network. The Fairhair IoT network is formed by the ZCLIP tool’s client block which is associated with a HTML UI. The ZigBee legacy network is made up of the HSE tool which contains a set of ZigBee nodes (a network bridge and bulbs) and is associated with a HTML UI. The gateway consists of a gateway application that acts as a server to the ZCLIP tool’s client block, and as a client to the HSE tool. The ZCLIP client sends CoAP messages to the ZCLIP server; the gateway application in turn interacts with the ZCLIP server using a Command Line Interface (CLI). The gateway application communicates CoAP messages and these messages are sniffed using the network packet analyzer tool- Wireshark [53]. To link this statement to figure 3, the gateway application actually communicates through a socket with the Hue Proxy. The Hue proxy then sends the ZigBee messages to the virtual ZigBee nodes and the ZigBee (ZigBee Cluster Library (ZCL) [54]) messages are sniffed by Wireshark.
5.2.2 Implementation Workflow: Gateway Application

The implementation work flow is based on the network configuration diagram given in Chapter 4. The problem description can be narrowed down to consider a specific implementation application scenario as shown in figure 7 and figure 8, to describe the implementation details of the project’s use cases. The application scenario that is considered contains one light switch IoT device on the Fairhair IoT network side which is identified as FH IoT Device #1. The ZigBee legacy network contains four legacy ZigBee devices: A ZigBee network bridge, a Hue bridge identified as ZigBee Device#1, and three ZigBee bulbs, identified as ZigBee Device #2, ZigBee Device #3 and ZigBee Device #4. The ZigBee devices #2 and #4 contain only the On/Off ZigBee cluster whereas the ZigBee device #3 contains an On/Off cluster and a Level Control cluster. In other words, the ZigBee devices #2 and #4 are On/Off light bulbs whereas ZigBee device #3 is a level control light bulb. The communication between the Fairhair IoT device and the ZigBee legacy devices is facilitated through the IoT gateway, which is from here on referred to as ‘gateway’.

The application scenario consists of the following use cases and these can be described in a sequential order as shown in the message sequence diagrams of figure 7 and figure 8:

1. **Gateway Initialization**
   In this step, the Gateway searches for all the ZigBee devices and services present in the ZigBee network and creates the corresponding URIs that could be used by an IoT device in the Fairhair IoT network to access the ZigBee devices and services.

2. **Discovery**
   In this step, the Fairhair IoT Device #1 sends a request to the Gateway, with an objective to find ZigBee clusters (i.e. services instead of devices) of type: On/Off. This functionality involves two parts: Device Discovery and Service Discovery. Device Discovery involves determining the network addresses of the ZigBee devices. Service Discovery involves determining the profile ID and cluster IDs of the ZigBee devices.

3. **Switching**
   In this step, the Fairhair IoT Device #1 sends a request to the Gateway to first ‘toggle’ a specific endpoint on ZigBee Device #2. With this, the ZigBee Device #2 goes from its default state ‘ON’ to ‘OFF’. Then, the IoT Device #1 sends a turn ON request to the Gateway to turn ‘ON’ the ZigBee Device #2. This is followed by a turn OFF request, which turns the state of the ZigBee device#2 to ‘OFF’.

4. **Multicast**
   This step involves two parts: Group Creation and Group Control. In group creation, the Fairhair IoT Device #1 first sends a request to the Gateway to add ZigBee Device #2 to a new group, with group ID: 1 and group name: Group1. Also, in a multicast scenario, specific endpoints on devices are added to a group instead of the device as a whole being added to the group. We do not see a difference in our application scenario because each bulb has only one endpoint and thus adding an endpoint to a group implies adding the device itself to the group. Once the ZigBee Device #2 is added to Group1, the IoT device #1 then sends a request to add ZigBee Device #3 to the Group1. Once the ZigBee devices #2 and #3 are added to Group1, the IoT Device #1 sends a ‘toggle’ request
to Group1 which toggles the Group1 members- ZigBee Device #2 and ZigBee Device #3 simultaneously. This forms the Group Control part of the Multicast functionality.
I. Gateway Initialization: Search for all devices and services

- [Connection,GetAddress] - Get Bridge Address
  [Connection:GetAddress, i=11:22:33:44:00:00:00:00, s=0x0001.0]
  [Zdp,SendGetTypeInfoReq, s=0x0001.0, 0] - Get Neighbor table of bridge
  [Zdp,ReceivedGetTypeInfoRsp, 2, s=0x0001.0, 0, 8, 0, 1, 0, 2, 238, 46:4D:2B:19:29:89:DB:AA, l=11:22:33:44:00:01:00:00, s=0x0002.0, 1, 1, 1, 0, 2, 0, 2, 238, 46:4D:2B:19:29:89:DB:AA, l=11:22:33:44:00:01:00:01, s=0x0003.0, 1, 1, 0, 2, 0, 2, 238]

- [Zdp,SendActiveEndpointReq, s=0x0002.0, 0, 2, 11, 242] - Get Active Endpoints of Device #2
- [Zdp,ReceivedActiveEndpointRsp, 3, s=0x0002.0, 0, 2, 11, 242] - Get Active Endpoints of Device #3
- [Zdp,ReceivedSimpleDescRsp, 6, s=0x0002.0, 0, 2, 11, 242] - Get Simple Descriptor of Active Endpoint 11 of Device #2
- [Zdp,ReceivedSimpleDescRsp, 7, s=0x0003.0, 0, 2, 11, 242] - Get Simple Descriptor of Active Endpoint 11 of Device #3
- [Zdp,ReceivedSimpleDescRsp, 8, s=0x0004.0, 0, 2, 11, 242] - Get Simple Descriptor of Active Endpoint 11 of Device #4

Create ZCLIP URIs

2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/1/s6>;rt=urn:zcl:c:6.s)
2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/2/s6>;rt=urn:zcl:c:6.s)
2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/3/s6>;rt=urn:zcl:c:6.s)
2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/1/s8>;rt=urn:zcl:c:8.s)
2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/2/s8>;rt=urn:zcl:c:8.s)
2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/3/s8>;rt=urn:zcl:c:8.s)

II. Use Case 1: Discovery - Search for OnOff Devices

CON MID: 10729, GET coap://[ff03::fd]/.well-known/core?rt=urn:zcl:c:6.s - Search for all devices with OnOff Cluster (6.s)

2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/1/s6>;rt=urn:zcl:c:6.s)
2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/2/s6>;rt=urn:zcl:c:6.s)
2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/3/s6>;rt=urn:zcl:c:6.s)
2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/1/s8>;rt=urn:zcl:c:8.s)
2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/2/s8>;rt=urn:zcl:c:8.s)
2.05 Content (Content-Format:application/link-format(40))(<coap://[::1]/zcl/e/3/s8>;rt=urn:zcl:c:8.s)

Fig 7. Implementation Application Scenario - Part 1
III. Use Case 2. Switching: Toggle, On, Off

CON MID: 16972, POST coap://[:1]/zcl/e/1/s/6/c/2 - URI to toggle ZigBee Device #2, through Gateway

\[\text{Zcl, S, S=0x0002.11,6,01A102,64}\] - ZigBee Command to toggle ZigBee Device #2

ACK MID: 16972, 2.04 Changed

ACK MID: 64499, 2.04 Changed

ACK MID: 3681, 2.04 Changed

IV. Use Case 3. Multicast: Group Creation and Group Control

A. Group Creation

CON MID: 58927, POST coap://[:1]/zcl/e/1/s/4/c/0 {0:1,1:Group1} - URI to add ZigBee Device #2 to 'Group 1', through Gateway

\[\text{Zcl, S, S=0x0002.11,4,01A100 0100 06 47726F757031,64}\] - ZigBee Command to add ZigBee Device #2 to Group 1

ACK MID: 58927, 2.04 Changed {0:1,1:Group1}

CON MID: 51982, POST coap://[:1]/zcl/e/2/s/4/c/0 {0:1,1:Group1} - URI to add ZigBee Device #3 to 'Group 1', through Gateway

\[\text{Zcl, S, S=0x0003.11,4,01A100 0100 06 47726F757031,64}\] - ZigBee Command to add ZigBee Device #3 to Group 1

ACK MID: 51982, 2.04 Changed {0:1,1:Group1}

B. Group Control

CON MID: 20867, POST coap://[:1]/zcl/g/1/s/6/c/2 - URI to toggle Group 1, through Gateway

\[\text{Zcl, S, G=0x0001,6,01A102,64}\] - ZigBee Command to toggle Group 1

ACK MID: 16972, 2.04 Changed

ACK MID: 64499, 2.04 Changed

ACK MID: 3681, 2.04 Changed

Zcl: Default Response (Command: Add Group, Status: Success, Group ID: 0x0001, Group Name: Group1), A1

Zcl: Default Response (Command: Add Group, Status: Success, Group ID: 0x0001, Group Name: Group1), A1

Zcl: Default Response (Command: Add Group, Status: Success, Group ID: 0x0001, Group Name: Group1), A1

Fig 8. Implementation Application Scenario- Part 2
5.2.3 The IoT Gateway Translation Algorithm

The translation between the Fairhair IoT network and the ZigBee legacy network messages is essentially a translation between the ZCLIP tool and HSE tool messages respectively. It is done through the IoT Gateway application that uses a translation algorithm which is described in this section.

![Fig 9. The Network Message Translation Process](image)

The translation algorithm of the overall IoT gateway application is presented as a flowchart in figure 10. The gateway application first reads in the CoAP message from the ZCLIP tool. The gateway application receives the CoAP message as a serialized message (a string) from the ZCLIP server. The message is split using a string delimiter to extract key CoAP message parameters. These parameters include: CoAP action method, destination address, source address, destination endpoint and command payload. The gateway application checks the CoAP action method. If the CoAP action method is a ‘GET’ method, there exists two possibilities; the message could be for either Gateway Initialization or for Discovery. To determine between these two options, the gateway application checks if the CoAP message has a query/ filter part. If the message does not have the query part, the gateway application calls the Gateway Initialization module else it calls the Discovery module and passes the query part to it. If the CoAP action method is a ‘POST’ method, the gateway application checks the destination endpoint parameter. If the destination endpoint is not a single endpoint, it calls the Multicast module directly since the destination would be a group. But if the destination endpoint is a single endpoint, it checks for the cluster ID. If the cluster ID is 6 (On/Off cluster), then the gateway application concludes that the CoAP message request is for a Switching operation and thus calls the Switching module. If the cluster ID is 4 (Group cluster), it calls the Multicast module. The different modules of the gateway application- Gateway Initialization, Discovery, Switching and Multicast perform their respective tasks and use a socket to communicate their respective translated messages to the HSE tool. The socket also receives suitable acknowledgement messages from the HSE tool and in turn communicates those to the ZCLIP tool.
Fig 10. Overall Gateway Application Flowchart
The Gateway Initialization module sends a set of ZigBee discovery commands to the ZigBee network to discover all the devices and services available in the ZigBee network. Since the details of these ZigBee discovery commands is already covered under section 4.3 of chapter 4, this section will focus on the implementation of those commands. The ZigBee discovery commands are essentially API calls that are sent using the Socket module of the gateway application. They are sent by the gateway application to the Hue proxy. The Hue proxy processes these API calls and sends the corresponding ZigBee messages on the virtual ZigBee network. These API calls are sent in a sequence such that the ZigBee command/ API call that is sent is based on the response that is received for the previous ZigBee command/ API call. The following paragraph covers the semantics of the messages sent and received during the initialization process. The message semantics that are relevant for this project are alone covered and are described in the order in which they appear in the message.

The Gateway Initialization module can be described using figure 8. The module first sends the ZigBee API call- [Connection, GetAddress] to get the network address of the ZigBee network bridge (HUE bridge). In
response, it receives the response- 

\[\text{[Connection, GetAddress, L=11:22:33:44:00:00:00:00, S=0x0001.0]}\]

Here, the parameters L and S are the ZigBee long and short address of the ZigBee network bridge (HUE Bridge). Based on this response, the Gateway initialization module sends the ZigBee API call- 

\[\text{[Zdp,SendMgmtLqiReq,S=0x0001.0]}\] to get the neighbor table of the bridge. Here, ZDP stands for ZigBee Device Profile which indicates that the API call is based on the ZigBee Cluster Library (ZCL) command. SendMgmtLqiReq indicates that the API call sends the ZCL command- Mgmt_Lqi_req, S=0x0001.0 is the destination address and is that of the network bridge. 0 is the starting index of the neighbor table which indicates that the neighbor table request sent has to get the neighbor table entries from the index of 0. The neighbor table response is- 

\[\text{[ZDP, ReceivedMgmtLqiRsp,2, S=0x0001.0,0,3,0,2,46:4D:2B:19:29:89:DB:AA,L=11:22:33:44:00:00:00:00,S=0x0002.0,1,1,0,2,0,2,38,46:4D:2B:19:29:89:DB:AA,L=11:22:33:44:00:01:00:01,S=0x0003.0,1,1,0,2,0,2,238].}\]

ReceivedMgmtLqiRsp indicates that the received ZCL command is Mgmt_Lqi_rsp. 2 is the sequence number, S=0x0001.0 is the source address of the bridge, 3 is the number of neighbor table entries, 0 is the start index. The parameters L and S are long (IEEE address) and short (network address) addresses respectively of the devices in the neighbor table. So, the response received contains three devices with network addresses 0x0002.0, 0x0003.0 and 0x0004.0. The Gateway Initialization module then sends the ZigBee API call- 

\[\text{[Zdp,SendActiveEndPointReq,S=0x0002.0,2]}\] to get the active endpoints on ZigBee device #2. SendActiveEndPointReq indicates that the API call sends the ZCL command- Active_EP_req, S=0x0002.0 is the network address of the destination device (ZigBee Device #2). The response received for the active endpoint request is- 

\[\text{[Zdp, ReceivedActiveEndPointRsp,3,S=0x0002.0,0,2,2,11,242].}\]

Here, ReceivedActiveEndPointRsp indicates that the received ZCL command is Active_EP_rsp, 3 is the sequence number, S=0x0002.0 is the source address, 2 is the active endpoint count, 11 and 242 are the active endpoints. The gateway initialization module sends active endpoint requests to ZigBee devices with network address s=0x0003. 0 (ZigBee device #3) and s=0x0004.0 (ZigBee device #4) and discovers the active endpoints on those devices. After discovering the endpoints, the Gateway Initialization module sends the ZigBee API call- 

\[\text{[Zdp,SendSimpleDescReq,S=0x0004.0,4,11]}\] to get the simple descriptor on the active endpoint of 11 of ZigBee device #4. SendSimpleDescReq indicates that the API call sends the ZCL command Simple_Desc_req and 11 is the active endpoint. The response received for the simple descriptor request is- 

\[\text{[Zdp, ReceivedSimpleDescRsp,8, S=0x0004.0,0,4,26,11,49246,0,2,0,8,0,3,4,5,6,8,4096,64513,1,25].}\]

Here, ReceivedSimpleDescRsp indicates that the received ZCL command is Simple_Desc_Rsp, 8 is the sequence number, s=0x0004.0 is the source address, 11 is the endpoint, 49246 (C05E) is the profile ID that corresponds to ZigBee Light Link (ZLL), 8 is the number of ‘In’ clusters (Input clusters), [0,3,4,5,6 (On/Off), 8(Level Control),4096,64513] are the ‘In’ clusters, 1 is the number of ‘Out’ clusters and 25 is the out cluster. The gateway initialization module sends simple descriptor requests to the active endpoints of ZigBee devices with network address s=0x0002. 0 (ZigBee device #2) and s=0x0003.0 (ZigBee device #3) and discovers the profile ID and clusters on those devices.

The Gateway Initialization then computes the IoT endpoint value through the function- ZigBeeToIoT Table, which implements the ZigBee to IoT Table shown in Table 1. The table maps a ZigBee Network address-ZigBee endpoint combination to an IoT Endpoint value. E.g. The combination 0x0002 and 11 maps to the IoT endpoint e/1.
Table 1. ZigBee to IoT Table

<table>
<thead>
<tr>
<th>ZigBee NWK Addr of Node, ZigBee Endpoint</th>
<th>IoT Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0002, 11</td>
<td>e/1</td>
</tr>
<tr>
<td>0x0003, 11</td>
<td>e/2</td>
</tr>
<tr>
<td>0x0004, 11</td>
<td>e/3</td>
</tr>
</tbody>
</table>

Using the IoT endpoints and the cluster list the Gateway Initialization module creates the URIs. E.g. the URI corresponding to the On/Off service provided by endpoint 11 on ZigBee device #4 is `coap://[::1]/zcl/e/3/s6`, where e/3 is the IoT endpoint and s6 represents the On/Off cluster, present on the server side i.e. a ZigBee device. URIs corresponding to other ZigBee devices and their endpoints are created in a similar way. Once all the URIs are created, the gateway initialization module prints those on the CLI. The URIs can be used by the IoT device to access the devices and services present in the ZigBee network, in same way as it does in an IoT network.

In a real world situation, the Gateway Initialization module needs to be automatically executed when the gateway is powered up but in this project the Gateway Initialization module is triggered by clicking a link.
**ii. Discovery Module Operation**

![Discovery Module Flowchart]

**Fig 12. Discovery Module Flowchart**
Figure 12 assumes that each discovery request will repeat all steps included in the gateway initialization process. Alternatively, an annotated table of all devices and services (clusters) discovered in the ZigBee network can be maintained and it can be used to quickly respond to discovery requests for particular cluster types. Although this implies that the table consistency needs to be maintained (e.g.: gateway periodically needs to check whether the devices are still available), but this might be a more efficient solution overall.

From the networks configuration diagram shown in Chapter 4, there exists two options for implementing the discovery module:

**Option A**

In this option, a table as shown in Table 2 is maintained. This table maps a combination of IP address and IoT endpoint to a particular ZigBee NWK address. Thus, a discovery request for discovering all devices/services results in three answers, corresponding to the three ZigBee network addresses.

<table>
<thead>
<tr>
<th>IP Addr, IoT Endpoint</th>
<th>ZigBee NWK Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Addr A, e/1</td>
<td>ZigBee Addr A</td>
</tr>
<tr>
<td>IP Addr B, e/1</td>
<td>ZigBee Addr B</td>
</tr>
<tr>
<td>IP Addr C, e/1</td>
<td>ZigBee Addr C</td>
</tr>
</tbody>
</table>

**Option B**

In this option, a table as shown in Table 3 is maintained. This table maps an IoT endpoint to a particular ZigBee endpoint. In this approach, all the requests are addressed to one IP address- IP address of the gateway. Thus, a discovery request for discovering all devices/services will result in getting five answers, corresponding to the five ZigBee endpoints in the ZigBee network.

<table>
<thead>
<tr>
<th>IoT Endpoint</th>
<th>ZigBee Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>e/1</td>
<td>A.1</td>
</tr>
<tr>
<td>e/2</td>
<td>B.1</td>
</tr>
<tr>
<td>e/3</td>
<td>B.2</td>
</tr>
<tr>
<td>e/4</td>
<td>C.1</td>
</tr>
<tr>
<td>e/5</td>
<td>C.2</td>
</tr>
</tbody>
</table>
To summarize the descriptions of the above two options; in option A, the gateway generates a separate discovery response for each matching ZigBee device where as in option B, the gateway generates one single discovery response specifying all matching clusters, for any devices/ endpoints. The Discovery module in this project is implemented using the Option B since we have only one IP addressable device, which is the Gateway and it only listens on one IP address and ZigBee devices are all mapped to that one IP address.

The Discovery module first reads in the Cluster ID from the input CoAP message and stores it. Assuming that the Discovery is performed after a considerable time after the gateway is initialized, the Discovery module performs all the key Discovery steps performed by the Gateway Initialization module explained in the above section; Get the address of the network bridge, get the neighbor table of the network bridge, get the active endpoints of ZigBee devices and get the simple descriptors of devices.

The Discovery module then checks if the cluster ID that it reads is present in the cluster list of any of the Simple Descriptor responses that it has obtained. It does this by passing the Cluster ID as one of the parameters to the Socket module along with the socket message. Then the cluster ID is compared with the input cluster list of all the simple Descriptor responses received. If the cluster ID is not present in any of the Simple Descriptor responses, it prints out to the CLI that the cluster (service) is not available.

If the Cluster ID read in by the Discovery module is present in any of the Simple Descriptor responses received by the Discovery module, it stores the network address and active endpoint. Using these parameters, the Discovery module computes the IoT endpoint using the ZigBee to IoT table shown in Table 1. With the IoT endpoints and the input cluster ID, the Discovery module creates the URIs and sends those to the IoT device through the Socket module and those are displayed on the Command Line Interface (CLI). Thus, the URIs that are created correspond specifically to the devices that host the requested service (cluster) only and not to all the devices/ services present in the network. This distinguishes the Discovery module’s implementation from that of the Gateway Initialization module.

The following illustration uses messages from figure 7 to summarize the implementation of the Discovery module. It considers the example of IoT device sending a discovery request to the gateway to discover all the On/Off devices in the ZigBee network.
If the Discovery operation is immediately performed after the Gateway Initialization process, the Discovery module would directly check if the requested service/cluster is present in any of the URIs created by the Gateway Initialization process and accordingly send the selected URIs to the IoT device. This is illustrated in figure 7. However, in order to provide a more detailed understanding of the Discovery process, this shortened scenario is not implemented in this project.
ii. Switching Module Operation

The Switching module first parses the simple message constructs and parameters of the input CoAP message directly into ZigBee message constructs and parameters, which are then stored. The simple CoAP message constructs include: “CoAP”, port number value and “zcl”. The Switching module then extracts and stores the DST IoT Endpoint and Cluster ID from the CoAP message. The module maintains a table shown in Table 4, which maps DST IoT Endpoints to ZigBee Addresses and ZigBee Endpoints. E.g.: From Table 4, IoT endpoint ‘e/1’ maps to ZigBee address 0x0002 and ZigBee endpoint 11. Through Table 4, the Switching module computes the destination ZigBee address and destination ZigBee endpoint for each of
the DST IoT endpoints that it has extracted. It then formats the extracted CoAP message Cluster ID to create ZigBee message Cluster ID. The module then computes the ZigBee source endpoint value using a mapping shown in Table 5. This table maps the source IP addresses of Fairhair IoT devices to unique ZigBee source endpoint values, along with a separate transaction sequence number value assigned for each IoT device. E.g.: From Table 5, the IoT device with the IP address- 3ffe:1900:4545:3:200:f8ff:fe21:67c1 is assigned a ZigBee source endpoint value of 61, along with a transaction sequence number 0xA1. Separate transaction sequence numbers are maintained for each of the IoT devices to be able to discard messages that are duplicates or are not fresh. The source endpoint is mapped to IP address to be able to identify the IoT client targeted in a response. This implementation makes use of a single IoT device. Thus, the third and fourth rows of the table are for illustration only.

<table>
<thead>
<tr>
<th>IoT Endpoint</th>
<th>ZigBee NWK Addr of Node, ZigBee Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>e/1</td>
<td>0x0002, 11</td>
</tr>
<tr>
<td>e/2</td>
<td>0x0003, 11</td>
</tr>
<tr>
<td>e/3</td>
<td>0x0004, 11</td>
</tr>
</tbody>
</table>

The ZigBee message payload consists of three parameters: The Frame Control field, the transaction sequence number and the command ID. The Frame control field is fixed for this implementation since, this project mainly implements scenarios of request messages going from the Fairhair IoT network to the ZigBee network, and not the other way around. The fixed value of Frame Control field is: ‘01’ which indicates that the direction of communication is from the client to server. The Transaction Sequence Number value is derived from Table 5 as described in the above paragraph. The ZigBee Command ID value is computed by the Switching module. It computes the command Id by formatting the Command ID in the CoAP message. E.g.: The CoAP Command ID ‘2’, which corresponds to the toggle operation is translated to the ZigBee Command ID ‘02’ by the Switching module.

Once all the ZigBee message parameters are determined, the Switching module re-arranges those to create the required ZigBee message. This message is sent to the ZigBee network by using the Socket module which is described under section 6 of the chapter. In response to the ZigBee request message sent, the socket module receives a suitable ZigBee ACK (Acknowledgement), which in turn is translated back to a CoAP.
ACK. For a successful translation, the CoAP ACK received is: ‘2.04: Changed’, where 2.04 is the CoAP response code and ‘Changed’ indicates that the resource has been modified. As a last step, the CoAP ACK is printed on to the CLI (Command Line Interface). The ACK is also captured through the Wireshark tool.

The following illustration uses messages from figure 8 to summarize the implementation of the Switching module. It considers the example of IoT device sending a toggle request to ZigBee Device #2, through the gateway.

![Diagram of CoAP communication](image)

**CON MID: 16972, POST coap://[::1]/zcl/e/1/s/6/c/2**

[Zcl,S,S=0x0002.11,6,01A102,64]

Zcl: Default Response (Command: Toggle, Status: Success), A1

**ACK MID: 16972, 2.04 Changed**

Fig 15. Switching Module Operation

(Continued.)
iii. Multicast Module Operation

The Multicast module, similar to the Switching module first parses the simple message constructs and parameters of the input CoAP message directly into ZigBee message constructs and parameters, which are
then stored. The simple CoAP message constructs include: “CoAP”, port number value, “zcl” etc. This is followed by extraction and storage of the DST IoT Endpoint and Cluster ID from the CoAP message. The Multicast module then checks for the cluster ID in the CoAP message. This is done to differentiate a group creation request message from a group control message request; if the message request contains the cluster ID equal to 4, the Multicast module concludes that the message is a group creation request since cluster ID 4 indicates the ‘Groups’ cluster, which is associated with attributes and commands for group configuration. If the message request contains a cluster ID not equal to 4, then the Multicast module concludes that the request is a group control request.

For a group creation request, the Multicast module computes the destination ZigBee address and ZigBee endpoint using Table 4, as done by the Switching module. It then formats the extracted CoAP message Cluster ID to create ZigBee message Cluster ID. The module then computes the ZigBee source endpoint value using Table 5, as done by the Switching module. The ZigBee message payload in the case of group creation consists of the group ID and group payload parameters, apart from the other parameters - the Frame Control field, the transaction sequence number and the command ID. These three parameter values are computed in the same way as they are computed by the Switching module, which is explained in the above paragraphs. The only difference is, the CoAP command ID of ‘0’ is translated to ‘00’ in the translated ZigBee command ID and the value corresponds to ‘Add Group’ operation. For computing the group ID part of the ZigBee payload, the Multicast module uses the function- createGroupIdPayload. This function takes in the group ID from the CoAP message and returns group ID (in the correct format) of the ZigBee payload. It does this using the table shown in Table 6. E.g.: CoAP group ID ‘1’ maps to ZigBee group ID 0100. This is because the ZigBee group Id needs to be represented with four digits and more importantly, the representation needs to be in ‘big endian’ format. Thus, 1 translates to 0100 and not 0001. Since this implementation makes use of a single group, with group id = 1, the third and fourth rows in Table 6 are only for illustration.

<table>
<thead>
<tr>
<th>IoT Group ID</th>
<th>ZigBee Group ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0100</td>
</tr>
<tr>
<td>2</td>
<td>0200</td>
</tr>
<tr>
<td>3</td>
<td>0300</td>
</tr>
</tbody>
</table>

For creating the Group name part of the ZigBee payload, the Multicast module makes use of the function- createGroupNamePayload. This function reads in the group name from the CoAP message and returns the group name for the ZigBee message. The group name payload created by the createGroupNamePayload function, for the ZigBee message starts with ‘0’, followed by length of the group name and hex equivalent of the group name. The hex equivalent value is computed using a dedicated function- strToHex. E.g. if the group name in the CoAP message is ‘group1’, the ZigBee message equivalent of it is: 06 47726F757031 where 6 is the length of the string and 47726F757031 is the hex equivalent.
For a Group Control request, the Multicast module first computes the ZigBee network group address based on CoAP group ID, through a table shown in Table 7. This table is similar to Table 4 but here, the IoT endpoints are replaced by IoT group IDs and the ZigBee network address-ZigBee Endpoint combination is replaced with ZigBee network group address.

<table>
<thead>
<tr>
<th>CoAP Group ID</th>
<th>ZigBee NWK Group Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>g/1</td>
<td>0x0001</td>
</tr>
<tr>
<td>g/2</td>
<td>0x0002</td>
</tr>
<tr>
<td>g/3</td>
<td>0x0003</td>
</tr>
</tbody>
</table>

The subsequent processing of the CoAP message is done in the same way as is done in the Group Creation stage; the Multicast module formats the CoAP cluster ID to create the ZigBee message cluster ID. It then computes the ZigBee message payload and also determines the ZigBee source endpoint through Table 5.

As a common step to both group creation and group control, the Multicast module rearranges the different computed ZigBee message parameters in the right order to create the required ZigBee Multicast message. This message is then sent to the ZigBee network using the Socket module. For Group Creation message requests that are sent, the Socket module receives acknowledgements in the ZigBee frame format. These are translated and sent to the IoT device. E.g. from figure 8, for a Group Add request sent by the IoT device to add ZigBee device #2 to Group 1, the Socket module receives the ZigBee acknowledgement (ACK) that mainly contains the group ID and group name. This is later translated to a CoAP ACK. For a successful translation, the CoAP ACK received by the IoT device is: ‘2.04 Changed {0:1,1: Group1}’, where 2.04 is the CoAP response code and ‘Changed’ indicates that the resource has been modified, as in the Switching module. The CoAP ACK also contains the group ID and group name which confirm that the ZigBee device has been added to the requested group. The ACK is also captured through the Wireshark tool. The Socket module does not receive any acknowledgements for Group control requests because of the design which is described in Chapter 4.

The following illustration in figure 17 uses messages from figure 8 to summarize the implementation of the Multicast module. It considers the example of IoT device sending a ‘Group Add’ request to ZigBee Device #2 and then subsequently sending a ‘toggle’ group request to Group 1.
5.3. Gateway Communication Mechanism

The Gateway uses a TCP socket to communicate with the Hue Proxy. The socket module uses the destination socket address: 10.0.2.2 and port number: 60001 which correspond to the HUE System Emulator (HSE) tool. The socket is created using the `socket_create` function, with the function parameters - domain: AF_INET, type: SOCK_STREAM and protocol: SOL_TCP [55]. The translated ZigBee message is sent using the socket function - `socket_send` [56], with flag field value set to 0, which indicates that the data that is sent is the actual data that is being communicated. If the socket module is unable to send the data, it prints out suitable error messages by using the socket error functions - `sock_last_error` [57] and `socket_strerror` [58]. The socket module also writes the sent message to a text file - Testfile.txt. This is done to verify if the message translation by the gateway has been successful. The socket module then receives the message from the HSE tool using the `socket_recv` function [59], which uses a buffer size of 4000 bytes and a flag - MSG_PEEK. The MSG_PEEK flag indicates that the data is received from the beginning of the receive queue and a copy of it is retained in the queue. This flag type is chosen particularly because, in the case of the discovery use case, the gateway application receives multiple responses. If the socket module is unable to receive the response messages, it prints out suitable error messages using the aforementioned socket error functions. Lastly, the socket is not closed in order to maintain a continuous communication between the ZCLIP and HSE tools.
Chapter 6. Results and Discussion

This chapter presents the results of the implementation of project’s use cases. The implementation is done according to the implementation work flow described in the previous chapter. The results verify the correctness of implementation of the IoT Gateway’s functionality with respect to multiple network parameters such as source and destination addresses, endpoints, protocols and messages information. The results presented are supported with a discussion on key topics that are relevant to the project.

6.1 Result Terminologies

Before considering the actual results of the project’s use cases implementation, it is essential to consider the key network parameters used in describing the results. These network parameters are also the key sections displayed in the Wireshark tool which is largely used to demonstrate the results of the project. The following are the key network parameters:

1. **Source and Destination Addresses**
   Source address refers to the address of the device from which a message originates. Destination address refers to the address of the device where a message terminates. In the Wireshark on the ZCLIP tool side, these addresses are IP addresses. In the Wireshark on the HSE tool side, the Source and Destination addresses are ZigBee network addresses (Short addresses).

2. **Source and Destination Endpoints**
   The Source and Destination endpoints in the Fairhair IoT network side are CoAP endpoints. “A CoAP endpoint is a source or destination of a CoAP message” [28]. It is identified by an IP address and a port number [28]. The Source and Destination endpoints in the ZigBee network are ZigBee endpoints. A ZigBee endpoint represents an application running on a ZigBee node or device. A ZigBee endpoint is identified by a number which can range from 1 and 240 [60].

3. **Protocol**
   A protocol specifies the interaction between two or more communicating entities. The two key protocols considered in this project are CoAP (Constrained Application Protocol) [28] and the ZigBee protocol [39].

4. **Protocol-specific Parameters**
   Message information provides details of the message that is being communicated.

   a. **CoAP Message**
      The message information for a CoAP message includes the fields: message type, message ID, request/response methods and the URI.

      - Two types of CoAP messages are involved in this project: CON and ACK. ‘CON’ stands for Confirmable message and is a request message. The device which receives this type of message needs to respond with an acknowledgement. ‘ACK’ stands for Acknowledgement and is used to acknowledge a CON message. This message is sent by a device that received a request [28].
Formally the ACK may be piggybacked on the response message. If the response takes a longer time to generate, an ACK is often sent separately.

- Message ID is a 16-bit value and is used to detect duplicate messages. It specifies the transaction [28].

- The CoAP request/ response methods used in this project include: GET and POST. The GET method fetches a representation for the information that corresponds to the resource identified by the request URI. Upon success, a 2.05 (Content) or 2.03 (Valid) response code is obtained in the response message [28]. In this project, the GET method is used for implementing the discovery use case. The response message contains the response code 2.05. This is because the GET request used retrieves the endpoints and clusters (through the URIs) for accessing the ZigBee devices and services. The POST method requests for processing of the representation that is enclosed in the request. With the POST request, either a new resource will be created or the target will be updated. If the POST method succeeds and there is no creation of a new resource, the response contains the response code 2.04 (Changed) but if there is creation of a new resource, the response contains the response code 2.01 (Created) [28]. In this project, the POST method only updates the target and thus the POST method responses contain the response code 2.04 (changed) only.

- The CoAP message URI is used for identifying CoAP resources and for providing a means of locating the resource. It has a generic structure shown below [28]:

  Coap- URI= “coap://host[“:”port[/path-|abempty[“?”query]

  Host component is provided as an IP literal value of the CoAP server and this is the loopback address in this project since the Fairhair IoT device is hosted on the same IP interface as the gateway. The port subcomponent is the UDP port number at which the CoAP server is located. In this project this value is 5683, the default CoAP port. The path part of the URI identifies a resource within the scope of the host and the port. The <path-|abempty> is an absolute path and contains a sequence of path segments separated by ‘/’ character; it can be absent or empty. The query part further describes the resource. It is a sequence of arguments which are in the form of a ‘key=value’ pair separated by ‘&’ characters.

b. ZigBee Message
A ZigBee message includes the fields: message type, command and sequence number.

- In this project, all ZigBee messages are ZCL (ZigBee Cluster Library) [61] messages. The messages can be request or response. A request message is associated with a ZCL command and a sequence number. A response message is also associated with a sequence number which is equal to the sequence number included in its corresponding request message.

- Commands specify an action on a ZigBee cluster. They are identified by an 8-bit number and are either specific to a ZigBee cluster or are cross-cluster.
Sequence number is also an 8-bit field value. It is incremented by 1 for each new frame sent by the same source.

6.2 Results

An overview of the various results of the project’s use cases implementation is provided in table 1. The description column provides the key scenario(s) implemented for a particular use case. The result column provides the key outcomes of implementing the use case.

<table>
<thead>
<tr>
<th>Use case</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Gateway Initialization</td>
<td>Gateway application starts</td>
<td>URIs for accessing all ZigBee devices and services</td>
</tr>
<tr>
<td>2. Discovery</td>
<td>Discover all OnOff devices and services in ZigBee network</td>
<td>URIs for accessing On/Off service offered by ZigBee devices</td>
</tr>
<tr>
<td>3. Switching</td>
<td>a. Toggle</td>
<td>Access ZigBee devices and perform switching operations: Toggle, on and off</td>
</tr>
<tr>
<td></td>
<td>b. Turn on and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c. Turn off a ZigBee device</td>
<td></td>
</tr>
<tr>
<td>4. Multicast</td>
<td>a. Create a group of two OnOff ZigBee devices</td>
<td>Add ZigBee devices to a group</td>
</tr>
<tr>
<td></td>
<td>b. Toggle the ZigBee group</td>
<td>Control a ZigBee device group</td>
</tr>
</tbody>
</table>

The following sections describe the implementation results of the individual use cases in detail. The results are presented in the form of screenshots and the key sections of the screenshots that are relevant for this project are discussed.

6.2.1 Gateway Initialization

This section presents the results of implementation of the initialization phase of the gateway.

When the gateway starts up, it sends a set of four ZigBee commands to the ZigBee network to discover all the devices and services available in the ZigBee network. These commands are sent as API calls through a socket interface as described in Chapters 4 and 5. Figure 1 shows a collective Wireshark capture of all these commands, which includes both the request and response ZigBee messages. The Wireshark capture shows all the API calls, except the first API call: [Connection GetAddress] and its response: [Connection, GetAddress, L=11:22:33:44:00:00:00:00, S=0x0001.0]. This is because the first API call or command is executed on the ZigBee stack in the gateway itself, i.e., it is a local function call to get the gateway’s ZigBee address and not a message that is being sent over the network.
Figure 2 shows the second ZigBee discovery command—Link Quality Request that is sent to find out the network addresses of all the devices. This may or may not be all the device addresses in the entire network; if the command is used in a multi-hop network, the command needs to be sent again to the nodes discovered in the first response and this needs to be done iteratively. Since all the devices are connected to the network bridge—Hue bridge, the request message is sent to the Hue bridge and thus the source and destination of the command is the Hue bridge (Address: 0x0001). The ZigBee message detail section shows that the ZigBee Application Support Layer Data contains the source and destination endpoints as: 0. The endpoint 0 is a unique endpoint which always contains the ZigBee Device Object (ZDO). ZDO is an application that runs on endpoint 0 of all ZigBee devices. It is responsible for keeping track of the ZigBee device. It also provides an interface to the ZigBee Device Profile (ZDP). The ZDP is a special application profile that has a profile ID 0x0000 and is responsible for discovering, configuring and maintaining ZigBee devices and services in the network [62]. The ZigBee Device Profile in the figure contains the command Link Quality Request with sequence number: 0 and index: 0. Index refers to the starting index of the requested elements of the neighbor table [48]. Thus this request message is sent to discover the devices present at index 0 of the neighbor table.
Fig 2. HSE wireshark output: Link quality request message sent to Hue bridge, index=0

Figure 3 shows the second ZigBee discovery command’s response- Link Quality Response, that mainly contains the network addresses of all the devices. The network addresses obtained are in the form of neighbor table entries. Since the response is received by the Hue bridge (Address: 0x0001), the source and destination addresses are 0x0001. The response command- Link Quality Response is associated with a status: success, indicating that the request has been successfully processed. The ZigBee message detail section shows the neighbor table that mainly contains the network addresses of the devices that are connected to the Hue bridge. The neighbor table also contains other parameters along with the network addresses of the devices but those are not very relevant for this project. The neighbor table contains two entries with two corresponding network addresses: 0x0002 and 0x0003 at index: 0. These are the ZigBee network short addresses (16-bit) of the ZigBee device #2 and ZigBee device #3 respectively.
Figure 4 shows the second ZigBee discovery command - Link Quality Request, with index: 2 and it is sent to find out the network addresses of the devices present at index 2 of the neighbor table. Since the devices are connected to the network bridge - Hue bridge, the request message is sent to the Hue bridge and thus the source and destination of the command is the Hue bridge (Address: 0x0001). The index 2 indicates that the neighbor table needs to be retrieved twice. This is because the size of the neighbor table is too large for a single ZigBee packet, which is addressed by asking for table entries at specific indices. The ZigBee message detail section shows that the source and destination endpoints is: 0, which contains the ZigBee Device Object. The ZigBee Device Profile in the figure contains the command Link Quality Request with index: 2. Thus this request message is sent to discover the devices present at index 2 of the neighbor table.
Figure 5 shows the second ZigBee discovery command’s response- Link Quality Response, with index:2 and it mainly contains the network addresses of the devices present at index 2 of the neighbor table. The response command- Link Quality Response is associated with a status: success, indicating that the request has been successfully processed. The ZigBee message detail section shows the neighbor table entry with the network address 0x0004, which is the ZigBee network short address of (16-bit) of the ZigBee device #4. The ZigBee message detail section also shows other neighbor table related parameters: table size and table count. Table size refers the total number of neighbor table entries within the remote device- Hue bridge. Since three devices (ZigBee device #2, ZigBee device #3 and ZigBee device #4) are connected to the Hue bridge, total number of neighbor table entries is 3 as shown in figure 5. Table count refers to the total number of neighbor table entries within the current neighbor table list of entries. Since figure 5 shows that only one neighbor table entry at index 2 which is ZigBee device #4, the value of the table count parameter is 1 as shown in the figure.
Figure 6 shows the third ZigBee discovery command - Active Endpoint Request that is sent to find out the active ZigBee endpoints on the ZigBee device #2. The request message is sent to the ZigBee device #2 (Address: 0x0002) from the Hue bridge (Address: 0x0001). The ZigBee message detail section shows that the ZigBee Application Support Layer Data contains the source and destination endpoints as: 0. This is because the endpoint 0 contains the ZDO which provides an interface to the ZDP which is associated with the functionalities of discovering, configuring and maintaining ZigBee devices and services in the network as explained above. The ZigBee Device Profile in the figure contains the command - Active Endpoint Request, with Device field value: 0x0002. This indicates that the request is sent to the ZigBee Device #2, which has the network address 0x0002.
Figure 7 shows the third ZigBee discovery command’s response- Active Endpoint Response, that mainly provides all the active endpoints on the ZigBee device #2. The response is received by the Hue bridge (Address: 0x0001) from the ZigBee device #2 (Address: 0x0002). The response command: active endpoint response is associated with a status: success, indicating that the request has been successfully processed. The ZigBee message detail section shows the endpoint count and the active endpoint list. The endpoint count gives the count of active endpoints on the device. The active endpoint list provides a list of all the active endpoints present on the device. Since two active endpoints are present on the ZigBee device #2 (Address: 0x0002), the endpoint count is 2 and the endpoints (8-bit values) are 11 and 242 as shown in the figure.

Fig 7. HSE wireshark output: Active endpoint response message, from ZigBee device 0x0002

Similar to figure 6, the figures 8 and 10 show the third ZigBee discovery command- Active Endpoint Request that is sent to find out the active ZigBee endpoints on ZigBee device #3 and ZigBee device #4 respectively. Similar to figure 7, the figures 9 and 11 show the third ZigBee discovery command’s response- Active Endpoint Response that is received and provides the active endpoints on ZigBee device #3 and ZigBee device #4 respectively. Thus figures: 7, 9 and 11 essentially show the gateway’s device discovery of all the ZigBee devices in the ZigBee network.
Fig 8. HSE wireshark output: Active endpoint request message, to ZigBee device 0x0003

<table>
<thead>
<tr>
<th>Frame 17: 30 bytes on wire (240 bits), 30 bytes captured (240 bits) on interface 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE 802.15.4 Data, Dst: 0x0003, Src: 0x0001</td>
</tr>
<tr>
<td>ZigBee Network Layer Data, Dst: 0x0003, Src: 0x0001</td>
</tr>
<tr>
<td>ZigBee Application Support Layer Data, Dst Endpt: 0, Src Endpt: 0</td>
</tr>
<tr>
<td>ZigBee Device Profile, Active Endpoint Request, Device: 0x0003</td>
</tr>
<tr>
<td>Sequence Number: 8</td>
</tr>
<tr>
<td>Device: 0x0003</td>
</tr>
</tbody>
</table>

Fig 9. HSE wireshark output: Active endpoint response message, from ZigBee device 0x0003

<table>
<thead>
<tr>
<th>Frame 18: 34 bytes on wire (272 bits), 34 bytes captured (272 bits) on interface 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE 802.15.4 Data, Dst: 0x0003, Src: 0x0003</td>
</tr>
<tr>
<td>ZigBee Network Layer Data, Dst: 0x0003, Src: 0x0003</td>
</tr>
<tr>
<td>ZigBee Application Support Layer Data, Dst Endpt: 0, Src Endpt: 0</td>
</tr>
<tr>
<td>ZigBee Device Profile, Active Endpoint Response, Device: 0x0003, Status: Success</td>
</tr>
<tr>
<td>Sequence Number: 8</td>
</tr>
<tr>
<td>Status: Success (0)</td>
</tr>
<tr>
<td>Device: 0x0003</td>
</tr>
<tr>
<td>Endpoint Count: 2</td>
</tr>
<tr>
<td>Active Endpoint List</td>
</tr>
<tr>
<td>Endpoint: 11</td>
</tr>
<tr>
<td>Endpoint: 242</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frame 19: 30 bytes on wire (240 bits), 30 bytes captured (240 bits) on interface 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE 802.15.4 Data, Dst: 0x0003, Src: 0x0003</td>
</tr>
<tr>
<td>ZigBee Network Layer Data, Dst: 0x0003, Src: 0x0003</td>
</tr>
<tr>
<td>ZigBee Application Support Layer Data, Dst Endpt: 0, Src Endpt: 0</td>
</tr>
<tr>
<td>ZigBee Device Profile, Active Endpoint Request, Device: 0x0003</td>
</tr>
<tr>
<td>Sequence Number: 8</td>
</tr>
<tr>
<td>Device: 0x0003</td>
</tr>
</tbody>
</table>

Page 114 of 151
Figure 12 shows the fourth ZigBee discovery command - Simple Descriptor Request that is sent to find out the clusters on the active endpoint 11 of ZigBee device #4. The Simple Descriptor request is also sent to the other active endpoint: endpoint 242 but this endpoint does not contain the clusters that are relevant for this project. Hence the description for the Simple Descriptor request sent to endpoint 242 is not considered. With regard to Simple Descriptor request sent to the active endpoint 11, the request message is sent to the ZigBee device #4 (Address: 0x0004) from the Hue bridge (Address: 0x0001). The ZigBee message detail section shows that the ZigBee Application Support Layer Data contains the source and destination endpoints as: 0. This is because the endpoint 0 contains the ZDO which provides an interface to the ZDP which is associated with the functionalities of discovering, configuring and maintaining ZigBee devices and services in the network as explained above. The ZigBee Device Profile in the figure contains the
command- Simple Descriptor Request, with Device field value: 0x0004 and endpoint value: 11. This indicates that the request is sent to endpoint 11 of the ZigBee Device #4 (Address: 0x0004).

Figure 13 shows the fourth ZigBee discovery command’s response- Simple Descriptor Response, that mainly contains the clusters present on the active endpoint 11 of the ZigBee device #4 (Address: 0x0004). The response is received by the Hue bridge (Address: 0x0001) from the ZigBee device #4 (Address: 0x0004). The response command: simple descriptor response is associated with a status: success, indicating that the request has been successfully processed. The ZigBee message detail section shows multiple simple descriptor parameters as indicated in the figure. Since the simple descriptor request is sent to discover the clusters on endpoint 11, the endpoint value is 11. The application profile that is supported on the endpoint 11 is ZigBee Light Link (ZLL) [63] and hence the profile ID value is: 0xc05e, which corresponds to the ZLL application profile. The ZigBee device #4 is a dimmable light and according to the ZLL application profile, the application device ID for a dimmable light is 0x0100 which is verified from the figure. The application version field is not relevant for this project. The Input cluster field specifies the count of the input clusters supported on the endpoint and its value is: 7. The input cluster list is a list of all the input clusters supported on the endpoint, for use during the service discovery. The clusters in the cluster list are identified by their cluster ID/numbers. Since, the ZigBee device#4 is a dimmable light, it contains the input clusters: Level Control (Cluster ID: 8), OnOff (Cluster ID: 6), and Groups (Cluster ID: 4). The other input clusters are not relevant for this project. Since the ZigBee device #4 is a server side device, it does not contain any output clusters and hence the output cluster count is: 0 in the figure. The concept of ZigBee...
input clusters and output clusters can be understood in detail from figure 3.2 of the ZigBee Cluster Library Specification document [61].

Similar to figure 12, the fourth ZigBee discovery command- Simple Descriptor Request is sent to find out the clusters on the active endpoint 11 of ZigBee device #2 and ZigBee device #3 respectively. Similar to figure 13, the figures 16 and 17 show the fourth ZigBee discovery command’s response- Simple Descriptor Response that is received and provides the input clusters on the active endpoint 11 of ZigBee device #2 and ZigBee device #3 respectively. Thus figures: 13, 16 and 17 essentially show the gateway’s service discovery of all the key ZigBee services available in the ZigBee network that are relevant to this project.

Figure 14 shows the URIs that are derived by the gateway after it completes its initialization phase. The URIs are received on the CLI of the ZCLIP tool and then extracted to a text file. The URIs enable Fairhair IoT devices to access all the devices and services available in the ZigBee legacy network, through the gateway. The first URI enables the FH IoT device #1 to access the ZigBee device #2 and its OnOff service. In the URI, coap indicates that a CoAP request is used to access the device/service, ‘[::1]’ indicates that the destination address used is a loopback address which is because the CoAP client and the CoAP server (gateway) are located on the same host, ‘zcl’ indicates the ZigBee entry to the target resource/ device, e/l refers to the destination IoT/CoAP endpoint 1. In this project, it has been chosen to map all devices to the same CoAP server, hence we get one long list of endpoints e/1, e/2, e/3 etc. One endpoint per ZigBee endpoint on any ZigBee device. Alternatively, devices could be modelled as separate CoAP endpoints.
Considering the URI again, s6 indicates the server side OnOff cluster which has a cluster ID 6. The section ‘rt=urn:zcl:c6.s’ indicates that the target device hosts a service that is defined by another protocol. The parameter ‘rt’ refers to the resource or service type, c refers to cluster and ‘.s’ indicates that the URI should be sent to a server side cluster, which in this case is the OnOff server side cluster indicated by the cluster ID: 6. The second and fourth URI responses have a similar structure as the first URI described above. The difference is that the second and fourth URI responses enable the FH IoT device #1 to access the ZigBee device #3 and ZigBee device #4 and their OnOff services respectively. The third URI response also has a similar structure as the other URI responses but the difference is that, it enables the FH IoT device #1 to access the ZigBee device #4 and its Level Control service.

Fig 14. ZCLIP tool CLI output: URIs to access all devices and services present in the ZigBee network, extracted to a text file

6.2.2 Discovery Use Case

This use case result presents the result of implementation of the Discovery use case. The scenario that is considered for implementation is the Fairhair IoT device #1 tries to discover all the ZigBee devices offering the OnOff service, through the gateway. In response, it gets the URIs to access all the ZigBee devices that offer the OnOff service, through the gateway.

Figure 15 shows a discovery CoAP request message that is sent by the Fairhair IoT device #1 to the gateway to discover all the ZigBee devices offering the OnOff service. Since the CoAP client (Fairhair IoT device#1) and the server (gateway application) are both on the same host, a loop back address is used. Thus, the source and destination addresses are “::1”. The CoAP request message is created as a CON message. A CON message defines that the request should be acknowledged. This is independent of a response, which is anyway generated for each unicasted request. The CoAP client gets the URIs from the target device in response to the request message it sent. With this it can access the devices and services in the ZigBee network. Since the request message is intended to get the content which is the URIs, a GET CoAP method is used. In the message URI, ‘.well-known/core’ indicates that the request is made to the ‘.well-known/core’ resource [33] with the query part specifying the OnOff cluster in the ZigBee namespace as described under section 6.2.1 of this chapter. The message does not contain any payload. The figure also shows the
acknowledgement message that is sent by the CoAP server (gateway) to the CoAP client (Fairhair IoT device #1). The CoAP acknowledgement message is an ACK message corresponding to the discovery request message, which is a CON type CoAP message. The ACK message has the same message ID: 10729, which is also the message ID contained in the discovery request CON message. This indicates that the ACK message corresponds to the appropriate CON message. The response code is: 2.05 Content which means that the request has been successful and the ACK message contains a content payload. The content payload of the ACK message contains the URIs for accessing all the ZigBee devices that offer OnOff service. These URIs are presented towards the end of this section.

![Fig 15. ZCLIP wireshark output: Discovery request message, sent to the gateway](image)

As discussed in Chapter 4 and 5, service discovery in a ZigBee network deals with finding the clusters (essentially services) present on the devices. Service discovery in a ZigBee network is done based on the profile IDs and the cluster IDs and not on the device IDs [62].

Based on the CoAP discovery message received, the gateway performs the same actions that it performs in its initialization phase; it sends the same four types of ZigBee discovery commands to the ZigBee network in the same sequence and receives appropriate responses as described in section 6.2.1. But, this approach seems inefficient if there is little dynamic behavior (devices leaving or joining) in the ZigBee network. Alternatively, a caching mechanism can be implemented in the gateway, where in the gateway would cache discovered devices and services. When a discovery request is received by the gateway, it could first check in its cache if the requested device or service is present. If it is present, the gateway can fetch the requested device or service else, it can perform the discovery process by sending the ZigBee discovery commands and then add the discovered device or service to its cache.

The figures 1 to 11 and the associated descriptions apply to the discovery use case implementation also, but only for the initial part of the discovery use case. Similar to the description provided for figures 12 and 13, the gateway sends simple descriptor requests to ZigBee device #2 and ZigBee device #3. The simple descriptor responses received by these devices are shown in figures 16 and 17 respectively. The simple
descriptor response received by ZigBee device #4, for the discovery use case is shown in figure 18. From figures 16, 17 and 18, it can be observed that the ZigBee devices #2, #3 and #4 respectively contain the input cluster: 6 (OnOff cluster) on their active endpoint 11. Based on this, the gateway creates the URIs shown in figure 19. These URIs enable the Fairhair IoT device #1 to access the only the OnOff services present in the ZigBee legacy network, through the gateway; the first URI enables the Fairhair IoT device #1 to access the OnOff service of ZigBee device #2, the second URI facilitates access to the OnOff service of ZigBee device #3 and the third URI facilitates access to the OnOff service of ZigBee device #4. The gateway does not create an URI to access the level control service of ZigBee device #4 since the discovery request message contains the resource/service type as OnOff (Cluster ID: 6) only. This distinguishes the discovery use case implementation from that of the gateway initialization where the gateway creates URIs to access all the services available in the ZigBee network. The description of the URIs’ structure is as described under section 6.1.

From figures 16, 17 and 18 it may be observed that the ZigBee devices #2, #3 and #4 have the profile ID: 0xc5e which corresponds to the ZLL profile. The application device IDs of the devices are: 0x0000, 0x0000 and 0x0100 respectively. According to the ZLL User Guide [63], this implies that the device #2 and device #3 are OnOff light devices and device #4 is a dimmable light device. However, it may be observed from figures 16 and 17 that the devices #2 and #3 contain the Level Control cluster (Cluster Id: 8) although they are OnOff devices. This is because, according to section 2.2.1 of the ZLL User Guide [63], the level control cluster is a part of an OnOff light device that has a ZLL profile. The ‘change level’ command of the Level Control cluster can be used to control an OnOff light; an increase in level switches ON the light and a decrease in level can switch it OFF [63]. For service discovery in this project, the gateway application only checks if a required cluster (service) is present in the cluster list of a device. If it is present, then it creates an URI to access that particular service. Thus if the gateway in this project received a discovery request message to find all the level control services in the ZigBee network, it would create URIs that include those that provide access to the level control services offered by OnOff light device #2 and #3. But, if the ZigBee devices had a different application profile ID such as: 0x0104 (Home Automation profile) and the gateway received a discovery request to find all the level control services, the gateway’s response would be different. The URIs created by the gateway in that case would provide access to the level control service provided by ZigBee device #4 only. This is because according to the ZigBee HA profile specification [64], the Level Control cluster is not present in an OnOff light device and is present only in a dimmable light device. Thus, service discovery in a ZigBee network is dependent on both the cluster IDs and the profile IDs.

(Continued.)
Fig 16. HSE wireshark output: Simple descriptor response message, from ZigBee device 0x0002

Fig 17. HSE wireshark output: Simple descriptor response message, from ZigBee device 0x0003

Fig 18. HSE wireshark output: Simple descriptor response message, from ZigBee device 0x0004 (for discovery use case)
Fig 19. ZCLIP tool CLI output: URIs to access all OnOff devices and services present in the ZigBee network, extracted to a text file

Figure 20 shows the HSE tool’s dashboard output. It shows that the ZigBee device #2 (Address: 0x0002) is an OnOff light device. This is indicated by the property and value section of the HSE dashboard, which shows that the `state.address` property value is ‘2’ and that the `type` property value is On/Off light.

Fig 20. HSE dashboard output: ZigBee Device 0x0002, with its properties and values

Similar to figure 20, figures 21 and 22 show the HSE tool’s dashboard output. They show that ZigBee device #3 (Address: 0x0003) is an OnOff light device and that ZigBee device #4 (Address:0x0004) is a dimmable light device respectively. These are indicated by the property and value section of the HSE dashboard, which shows that the `state.address` property value is ‘3’ and that the `type` property value is On/Off light for ZigBee device #3 and that the `state.address` property value is ‘4’ and that the `type` property value is dimmable light for ZigBee device #4.
6.2.3 Switching Use Case

This use case result presents the result of implementation of three operations of the switching use case: toggle, on and off.

Figure 23 shows the ‘toggle’ CoAP request message that is sent by the Fairhair IoT device #1 to the gateway to toggle ZigBee device 0x0002. Since the CoAP client (Fairhair IoT device#1) and the server (gateway application) are both on the same host, a loop back address is used. Thus, the source and destination addresses are “::1”. The CoAP request message is created as a CON message because then the client can be certain the request was either received or he was notified of the delivery failure. Since the request message is intended to toggle the state of the target device, a POST method is used. In the message URI, ‘zcl’ indicates that the target resource is a ZigBee resource, e/1 indicates the destination CoAP endpoint is 1, s6 indicates the server side OnOff cluster which has a cluster ID 6, and c/2 indicates the command id 2 which corresponds to toggle operation. The message does not contain any payload.

(Continued.)
Figure 24 shows the translated ZigBee request message for the toggle operation. The message is sent by the Hue bridge (Address: 0x0001) to the ZigBee device #2 (Address: 0x0002). The message is a ZCL message, directed for the server side OnOff cluster and contains the command- ‘toggle’. The message is associated with a sequence number 161, which is added by the gateway. The ZigBee message detail section shows that the ZigBee Application Support Layer Data contains the destination endpoint:11 and the source endpoint:64. These values are also computed and added by the gateway. The ZigBee Cluster Library frame contains the Frame Control Field which has the value 0x01, indicating that it is cluster specific which means that the ZCL command depends on the cluster number. The command ID is 0x02 which corresponds to the ‘toggle’ command, in the OnOff cluster.
Figure 25 shows the HSE tool’s dashboard output. It shows that the state of ZigBee device 0x0002 has been toggled to Off state. This is further verified by the property and value section of the HSE dashboard, which shows that the `state.on` property value is set to ‘false’.

Fig 25. HSE dashboard output: ZigBee device 0x0002 toggled from ON to OFF state
Similar to figures 23, 24 and 25, the figures 26, 27 and 28 show the ZCLIP Wireshark output, HSE Wireshark output and HSE dashboard output respectively for the turn ON operation on ZigBee device #2 (Address: 0x0002).

Fig 26. ZCLIP wireshark output: ‘ON’ request CoAP message sent to gateway, to turn ON ZigBee device 0x0002

(Continued.)
Fig 27. HSE wireshark output: ‘ON’ request ZigBee message sent to ZigBee device 0x0002

Fig 28. HSE dashboard output: ZigBee device 0x0002 turned to ‘ON’ state

The figures 29, 30 and 31 show the ZCLIP Wireshark output, HSE Wireshark output and HSE dashboard outputs respectively for the turn OFF operation on ZigBee device #2 (Address: 0x0002).
Fig 29. ZCLIP wireshark output: ‘OFF’ request CoAP message sent to gateway, to turn OFF ZigBee device 0x0002

Fig 30. HSE wireshark output: ‘OFF’ request ZigBee message sent to ZigBee device 0x0002
Figure 32 shows the ZigBee acknowledgement message corresponding to the turn OFF operation. The message is sent by the ZigBee device #2 (Address: 0x0002) to the Hue bridge (Address: 0x0001). This message is also a ZCL message but contains the command- ‘default response’. The default response command is used for indicating success or failure of a request message. The acknowledgement message is associated with the same sequence number 161, which is also associated with the request message. This ensures that the acknowledgement message corresponds to the appropriate request message. The ZigBee Application Support Layer data shows that the destination endpoint is 11 and the source endpoint is 64. This also confirms that the acknowledgement message is sent from and delivered to the correct endpoints.

The ZigBee Cluster Library frame contains the Frame Control Field which has the value 0x18, indicating that it is profile wide which means that the command is independent of the cluster number. The ZCL frame command is 0x0b, which corresponds to the ‘default response’ command. The ZCL frame contains another command, with command id: 0x00 which corresponds to the Off command. This indicates that the default response command is sent in response to an Off request message. The ZCL frame’s status field has the status code value 0x00, indicating ‘success’ or OK which indicates that the ZigBee device #2 has successfully processed the Off request message.
Figure 33 shows the CoAP acknowledgement message corresponding to the turn OFF operation. The message is sent by the CoAP server (gateway) to the CoAP client (Fairhair IoT device #1). Similar to the CoAP request message, since the source and the destination of the CoAP message are on the same host, a loopback address is used. Thus the source and destination addresses are “::1”. The CoAP acknowledgement message is an ACK message corresponding to the Off request message, which was a CON type CoAP message. The ACK message has the same message ID: 64499, which is also the message ID contained in the Off request CON message. This indicates that the ACK message corresponds to the appropriate CON message. The response code is: 2.04 changed which means that the request has been successful and the target resource of the request message has been changed. The acknowledgement message does not contain any payload, similar to its corresponding request CON message.
6.2.4 Multicast Use Case

This use case result presents the result of implementation of the two stages of the multicast use case: group creation/addition and group control.

Figure 34 shows the ‘group add’ CoAP request message that is sent by the Fairhair IoT device #1 to the gateway to add ZigBee device 0x0002 to a new group- Group1. The source and destination addresses are “::1” because of the argument provided in section 6.2.1. The CoAP message is created as a CON message so that the CoAP client knows about the addition of the target device to the group and can subsequently send the group control command. Since the request message is intended to add the target device to a group, a POST method is used. In the message URI, ‘zcl’ indicates that the target resource is a ZigBee resource, e/1 refers to the destination endpoint 1, s4 refers to the server side Groups cluster which has a cluster ID 4, and c/0 indicates the command id 0 which corresponds to group add operation. The request message does contain a payload. The payload is in CBOR data format and thus the gateway application receives the CoAP request with a CBOR payload, as seen in the figure. The CBOR payload contains two entries identified by indexes 0 and 1. The first entry identified with index 0 is an unsigned integer value ‘1’ and it refers to the group ID. The second entry identified with index 1 is a text string ‘Group1’ and it refers to the group name.
Fig 34. ZCLIP wireshark output: ‘Group add’ request CoAP message sent to gateway, to add ZigBee endpoint 11 of ZigBee device 0x0002 to Group1

Multicast in a ZigBee network is facilitated through the Groups cluster (Cluster ID: 0x0004). The Groups cluster facilitates the management of group addressing in ZigBee. In the group addressing scheme, an endpoint can be member of a group consisting of endpoints from one or more devices. A group is associated with a 16-bit group ID or address and a group name. The group ID and the endpoints of a group are maintained in a table - Group table on a device. If a message is sent to a group, the group table on the devices is used to decide the endpoints that should receive the message.

Figure 35 shows the translated ZigBee request for the group add operation. The message is sent by the Hue bridge (Address: 0x0001) to the ZigBee device #2 (Address: 0x0002). The message is a ZCL message, directed at the endpoint: endpoint 11 (located on device #2) and it contains the command- ‘Add Group’, which adds an endpoint to the group. The message is associated with the sequence number 161, which is added by the gateway. The ZigBee message detail section shows that the ZigBee Application Support Layer Data contains the destination endpoint: 11 and the source endpoint: 64. The destination endpoint: 11 indicates that the group add request essentially adds the endpoint 11 of ZigBee device #2 to
the group when the request is received by ZigBee device #2. The source and destination endpoint values are added by the gateway. The ZigBee Cluster Library frame contains the Frame Control Field which has the value 0x01, indicating that it is cluster specific. The command is 0x00 which corresponds to the ‘Add Group’ command, in the Groups cluster. The ZigBee request message also contains a payload part, which includes: the group ID: 0x0001, the group name: Group1 and length: 6. Length refers to the string length of the group name, which is 6 characters for Group1.

Figure 35 shows the HSE tool’s dashboard output. It shows that ZigBee device 0x0002 has been added to the group, with group ID 1. This is indicated by the property and value section of the HSE dashboard which shows that the groups11.1 property has the value 1, which is the group ID. The figure also shows that the device 0x0002 has a property- groups11.0 which has a value 6334. This means that the device is also member of another group, which has a group ID of 6334. But, this group with ID 6334 is not considered for description in this project.
Similar to figures 34, 35 and 36 the figures 37, 38 and 39 show the ZCLIP Wireshark output, HSE Wireshark output and HSE dashboard output respectively for the group add operation for adding the ZigBee endpoint 11 of ZigBee device #3 (Address: 0x0003) to the group- Group1.
Fig 37. ZCLIP wireshark output: ‘Group add’ request CoAP message sent to gateway, to add ZigBee endpoint 11 of ZigBee device 0x0003 to Group1

(Continued.)
Fig 38. HSE wireshark output: ‘Group add’ request ZigBee message sent to ZigBee device 0x0003, to add its ZigBee endpoint 11 to Group1

Fig 39. HSE dashboard output: ZigBee device 0x0003 added to group1
Figure 40 shows the ZigBee acknowledgement message corresponding to the Group Add ZigBee request message. The message is sent by the ZigBee device #3 (Address: 0x0003) to the Hue bridge (Address: 0x0001). This message is also a ZCL message but contains the command- ‘Add Group Response’, which is used for indicating success or failure of the requested group addition. The acknowledgement message is associated with the same sequence number 161, which is also associated with the request message. This ensures that the acknowledgement message corresponds to the appropriate request message. The ZigBee Application Support Layer data shows that the destination endpoint is 11 and the source endpoint is 64, which confirms that the acknowledgement is delivered from the correct endpoint. The ZigBee Cluster Library frame contains the Frame Control field which has the value 0x19, indicating that the Add Group Response command is cluster specific i.e. the command is dependent on the cluster number. The ZCL frame command is 0x00 which corresponds to the ‘Add Group Response’ command which is sent by the Groups cluster server of the ZigBee device #3 (Address: 0x0003). The ZCL frame contains a payload field containing two payload parameters: group status and group ID. The group status field is an 8-bit enumerated value and has the value 0x00, which indicating success. The group ID is a 16-bit unsigned integer value and has the value 0x0001, which is same as the group ID field value in the received Add Group command. The payload parameter values indicate that the ZigBee device #3 has successfully processed the Add Group request message.

Fig 40. HSE Wireshark output: ZigBee device 0x0003 sending an ACK response ZigBee message to the gateway, confirming that its endpoint (ZigBee endpoint 11) is added to Group1
Figure 41 shows the CoAP acknowledgement message corresponding to the Group Add operation for adding ZigBee device #3 (ZigBee endpoint 11) to the group - Group1. The message is sent by the CoAP server (gateway) to the CoAP client (Fairhair IoT device #1). Similar to the CoAP request message, a loopback address is used. Thus the source and destination addresses are “::1”. The CoAP acknowledgement message is an ACK message corresponding to the Add Group request message, which was a CON type CoAP message. The ACK message has the same message ID: 51982, which is also the message ID contained in the corresponding request CON message. The response code is: 2.04 changed which means that the request has been successful and the target resource has been changed i.e. the endpoint has been added to the group. The acknowledgement message does contain a payload and the payload is a representation of the action result. Thus the ACK message payload is different from that of the CON message; it differs in one payload parameter - status. The acknowledgement payload is in the CBOR data format and contains two entries identified by indexes 0 and 1. The first entry identified by index 0 is an unsigned integer value ‘0’ and it refers to the status: success. The second entry with index 1 is an unsigned integer value ‘1’ and it refers to the group ID.

Fig 41. ZCLIP Wireshark output: Fairhair IoT device #1 receiving an ACK CoAP response message, confirming that ZigBee device 0x0003 (ZigBee endpoint 11) is added to Group1
Figure 42 shows the ‘toggle’ CoAP request message that is sent by the Fairhair IoT device #1 to the gateway to toggle the devices of Group1. The source and destination addresses are “::1” since a loopback address is used. The CoAP request message is created as a CON message but the acknowledgements of a group control message are suppressed. Since the request message is intended to toggle the state of all devices of the group, a POST method is used. In the message URI, g/1 refers to the destination group ID 1, s6 refers to the server side OnOff cluster which has a cluster ID 6, and c/2 indicates the command id 2 which corresponds to toggle operation. The message does not contain any payload.

Figure 43 shows the translated ZigBee request message for the group toggle operation. The message is sent by the Hue bridge (Address:0x0001) as a broadcast message to the ZigBee network. If a ZigBee device’s
group table contains the group ID present in the broadcast message, then the request message is processed by the device. The request message is a ZCL message, directed for the server side OnOff clusters of the group’s devices/ endpoints. The message contains the command ‘toggle’. The message is associated with a sequence number 161, which is added by the gateway. The ZigBee message detail section shows that the ZigBee Application Support Layer Data contains the destination group ID: 0x0001 and the source endpoint: 64. The group ID value is derived from the input CoAP message whereas the source endpoint value is added by the gateway. The ZigBee cluster Library frame contains the Frame Control field which has the value 0x01, indicating that it is cluster-specific. The command ID is 0x02, which corresponds to the ‘toggle’ command in the OnOff cluster. The group control operation does not generate acknowledgement messages in the ZigBee network and which hence cannot be seen in the figure 43. One possible approach to check for the result of a group control request message is by reading the status of group member devices individually.

Fig 43. HSE Wireshark output: Group1 receiving a ‘toggle’ request

Figures 44 and 45 show the HSE tool’s dashboard outputs. Fig 44 shows that the state of the ZigBee device 0x0002 which is a member of the group- Group1 (group ID:1) has been toggled toggled to Off state. This is
further verified by the property and value section of the HSE dashboard, which shows that the state.on property has the value ‘false’. Also, the groups11.1 property has the value 1 which confirms that the device is a member of the group with group ID 1. Similarly, figure 45 confirms that the ZigBee device 0x0003 is toggled to its Off state and that it is a member of the group with group ID 1.

![HSE dashboard output: ZigBee device 0x0002 of Group1 toggled to OFF state](image1)

![HSE dashboard output: ZigBee device 0x0003 of Group1 toggled to OFF state](image2)
Chapter 7. Conclusion

This chapter concludes the entire project work. The first part of the chapter lists the key conclusions that can be made based on the analysis carried out on the gateway architectures, implementation workflow and the implementation results presented in the previous chapter. The second part covers the key challenges and issues faced during the project and the last part proposes future prospects for the project.

7.1 Conclusion

Following are the important conclusions that can be made based on Chapters 4, 5 and 6:

- All the three use cases proposed in the project’s problem description statement: discovery, switching and multicast use case have been implemented successfully through the IoT gateway.

- The IoT gateway solution developed in this project demonstrates that the interoperability problem between a Fairhair IoT network and a ZigBee legacy network can be solved by the use of a suitably designed IoT gateway. Since the IoT network used in this project is according to the Fairhair alliance specifications, the project verifies the Fairhair alliance specifications with respect to discovery, switching and multicast scenarios.

- Out of the potential architectures for an IoT gateway that links a Fairhair IoT network to a ZigBee legacy network, the service per node type architecture was found to be the most suitable architecture. This is because of two key reasons: firstly, the service per node architecture reduces the resource requirement on the IoT nodes such as: memory, processing power etc. and enables them to communicate with the gateway just as they would do with any other resource constrained node. The second reason is, the architecture does not increase the complexity of the gateway with increase in the number of network nodes i.e. the gateway does not require additional parameters for translating messages between networks when the number of nodes increases and thus provides good scalability even with increase in number of services.

- The IoT gateway was designed and implemented according to the Service per Node architecture. The IoT gateway translation algorithm used by the IoT gateway provides systematic mapping between CoAP URIs (structured URIs) and ZigBee devices + endpoints. The mapping is provided through the use of a set of lookup tables. In certain cases, where a direct mapping is not possible, such as in the case of ZigBee source endpoints, the IoT gateway adds/ creates appropriate values based on the input CoAP message parameters such as source IoT node IP address and facilitates correct translation between IoT/CoAP messages and ZigBee messages.

- Discovery in a Fairhair IoT network involves a single CoAP discovery request message sent to a target IoT device. But discovery in a ZigBee network is a relatively complex process and involves
a set of commands being sent to devices in the network. Thus, for the discovery use case implementation, the IoT gateway implements a correspondence between one CoAP discovery request and many ZigBee discovery request messages. A more efficient solution could have been made here, but this assumes knowledge on the dynamics in the ZigBee network.

- Switching in a Fairhair IoT network involves sending of unicast CoAP switching request message to a target device and subsequent receiving of suitable acknowledgement messages. In a ZigBee network also, switching is a simple process and involves sending of unicast ZigBee switching request message to a destination device and subsequent receiving of a suitable acknowledgement message. Thus, for the switching use case implementation, the IoT gateway implements a correspondence between one CoAP switching message and one ZigBee switching message along with the acknowledgement messages in both the networks. Switching is a use case which was selected to be representative of many unicast service interactions. The switching use case considered in this project consists of a light switch in a Fairhair IoT network sending switching requests (On/Off/ Toggle) to a light bulb in a ZigBee network. These switching requests and devices demonstrate the essence of a unicast service interaction: a device ‘A’ changes the state of a device ‘B’ and receives an acknowledgement message for the same. Thus, the switching use case considered in this project applies to a broader set of unicast service interactions.

- Multicast is a two-step process in both a Fairhair IoT network and a ZigBee legacy network and involves: group creation and group control. Group creation phase in both the networks involves sending of unicast messages to a target destination device and subsequent receiving of suitable acknowledgement messages. Thus, the IoT gateway implements a correspondence between one CoAP group add request and a group add ZigBee request message along with the acknowledgement messages in both the networks. Group control phase in both the networks involves sending of a broadcast message, to which the member devices of the group alone respond. However, in both the networks there is no acknowledgement received for a group control message as they are suppressed. Thus, the IoT gateway implements a correspondence between one CoAP group control broadcast request message and one ZigBee group control broadcast message request message, with no acknowledgements.

7.2 Challenges

Although all the three use cases proposed in the project’s problem description have been met, there were a few challenges faced during the implementation of the IoT gateway for the aforementioned project’s use cases.

The first key challenge encountered was related to the ZigBee source endpoint. A ZigBee message contains the ZigBee source endpoint value as a mandatory parameter. But, the input CoAP message contains only destination CoAP endpoint and the CoAP URI which can be easily mapped to a suitable ZigBee endpoint value by using a lookup table. To overcome the challenge, it was decided to use the source IP address of the sending IoT node to derive a suitable value for ZigBee source endpoint. With this, the gateway needs to store one table entry (one int variable) per IP client (IoT device) that interacts with the ZigBee network during the entire lifetime of the gateway. When the gateway is replaced, this
information somehow would need to be loaded again on the new gateway. This however brings the big advantage that the IoT devices do not need to specify another parameter (source endpoint) in their commands which is meaningless in the context of the IoT network.

The second key challenge faced was with the discovery use case implementation. The challenge with the discovery use case was the input CoAP message cannot be translated through the use of lookup tables as is done for the switching and multicast use cases. This is because the input CoAP message only contains the cluster ID (service) of interest in the query part of the GET request CoAP URI and the message contains only one destination CoAP endpoint. But, for implementing the discovery on the ZigBee network side, four type of ZigBee discovery commands need to be sent to the whole ZigBee network. This issue was overcome by extracting the cluster ID from the input CoAP message and using it as a parameter to identify the input CoAP request message as a discovery request message. The cluster ID was further used along with the four type of ZigBee discovery commands to perform discovery in the ZigBee network.

A few issues remain unresolved in this project owing to time constraints. Firstly, the project is a proof of concept and not a fully functional gateway. The transaction sequence number value in the translated ZigBee message does not increment. After debugging, it was found that this is a PHP cookie related issue. The CoAP message that is used by the gateway for translation to a ZigBee network is the CoAP message that is sent from the ZCLIP tool’s client block and not the one that is received by the ZCLIP tool’s server block. This is not an issue in this project since only one ZCLIP tool’s client (Fairhair IoT device #1) is used and hence both the messages are the same. But, this could cause issues if more than one ZCLIP client (Fairhair IoT device) is used. It was found that the issue could be resolved by integrating the gateway application’s source code with that of the ZCLIP tool’s server block.

7.3 Future Prospects

- This project considers only three use cases: discovery, switching, and multicast interaction for implementation in the IoT gateway. Other use cases, such as those involving the concepts of binding, caching and event reporting can be considered for implementation. One of the most relevant use case that could be explored is that involving caching mechanism (i.e. adding caching mechanism to the gateway) because adding a caching mechanism would make the discovery functionality of the gateway more efficient as discussed in Chapter 6. But, this would also bring new challenges such as efficient management of memory and other computational requirements of the gateway.

- Some more analysis can be carried out around multicast. This could involve aspects such as: what would happen when a gateway is defective and is replaced. The groups which were created via the gateway in the ZigBee network still exist in the ZigBee network. But, the new gateway will have no URI to enable interaction with these groups. Approaches to resolve these kind of issues can thus be taken up under the next steps of this project.

- The project can be expanded to consider more ZigBee device profiles such as home automation (profile ID: 0x0104), commercial building automation (profile ID: 0x0105), personal home
and healthcare (profile ID: 0x0108) and industrial plant monitoring (profile ID: 0x0101). The project can also consider ZigBee devices with other device IDs such as color light (device ID: 0x0200), extended color light (device ID: 0x0210) and color temperature light (device ID: 0x0220) which contain additional clusters apart from those used in this project. This would help in extending the scope of the gateway to networks containing multiple type of devices and services, thereby increasing the extensibility of the gateway.

- In this project, ZigBee is used as the legacy network both for analysis and design, and subsequent implementation. Other legacy networks such as BACnet and KNX can be taken up in place of ZigBee.

- The current gateway application is an independent application. It can be integrated with the source code of the ZCLIP tool to facilitate for scaling up this project.

- As described in chapter 5, this project uses Philips in-house simulation tools for implementing both the Fairhair IoT network and the ZigBee legacy network. The project can be extended to carry out the implementation on a suitable hardware platform such as Thread [65]. This would validate the results of this project in a hardware environment. It would also enable assessment of performance measures like delay of commands.

- In general, with the IoT technology already making its way into our daily lives, solutions like the IoT gateway developed in this project seem to have a promising future and high demand in realizing ambitious IoT based ideas such as connected lighting, connected healthcare, smart buildings and smart cities.

- Evaluation of performance aspects can be considered. Measuring performance of a virtual “set of nodes” is not that representative, but aspects like the scalability of the solution, its robustness etc. can be explored. E.g., what if there would be two gateways (implemented as described in this thesis), connecting Fairhair nodes to the same legacy ZigBee network. Would that work? Would the two gateways be redundant (be copies of one another)?

- IoT gateway solutions try to solve the interoperability problem between IoT networks and legacy networks and are thus an important step towards realizing the vision of ‘seamless connectivity’ in the world of Internet of Things. Just as the notion of gateways, which was introduced by the landmark research paper [66] of Vint Cerf and Bob Kahn in 1974 triggered the Internet technology, IoT gateways might also play a key role in the success of the Internet of Things technology.
References

1. Fairhair Alliance. Available at: http://www.fairhair-alliance.org/
2. Thread Group. Available at: http://www.threadgroup.org/
3. AllSeen Alliance. Available at: https://allseenalliance.org/
6. Philips Lighting. Adaptive street lighting. Available at: https://www.youtube.com/watch?v=_v6IjcLGDE
9. D Dee, M Verschoor, T Zotti, “Fairhair: Interoperable IoT services for interoperable building automation and lighting control ecosystems” Available at: https://www.google.nl/webhp?sourceid=chromeinstant&ion=1&espv=2&ie=UTF8&q=fairhair%3A+Interoperable+IoT+servicess+for+major+building+automation+and+lighting+control+eco+systems
27. Fairhair Resource Model Document: Fairhair Core Specification Draft 0.1r1


30. The Internet Engineering Task Force (IETF). Available at: https://www.ietf.org/
31. Constrained RESTful Environments(core). Available at: https://datatracker.ietf.org/wg/core/charter/
32. Fairhair Discovery Document: Service Discovery in Fairhair Draft 0.5

38. BACnet. Available at: https://en.wikipedia.org/wiki/BACnet
39. ZigBee. Available at: http://www.zigbee.org/
41. ZigBee Cluster Library (ZCL) Available at: http://www.zigbee.org/download/standards-zigbee-cluster-library/
42. [Doc] ZigBee Application Architecture Revision 12, Draft Version 1.0

46. [Doc] ZigBee 2004 Specification: ZigBee Document 053474r06

47. Microchip. ZigBee Frame Overview. Available at: http://www.microchip.com/forums/download.axd?file=0;193947


50. [Doc] Philips ZCLIP tool documentation.

51. ZigBee Light Link. Available at: http://www.zigbee.org/zigbee-for-developers/applicationstandards/zigbee-light-link/

52. Philips Hue. Available at: http://www2.meethue.com/nl-nl/

53. Wireshark tool. Available at: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html


65. [Online] https://threadgroup.org/About

Appendix

A. Abbreviations

1. PDA- Personal Digital Assistant
2. IoT- Internet of Things
3. ICT- Information and Communication Technology
4. IP- Internet Protocol
5. HTTP- Hypertext Transfer Protocol
6. URI- Uniform Resource Identifier
7. JSON- JavaScript Object Notation
8. CBOR- Concise Binary Object Notation
9. REST- Representational State Transfer
10. IP- Internet Protocol
11. OSAS- Open Service Architecture for Sensors
12. UPnP- Universal Plug and Play
13. HVAC- Heating, Ventilating and Air Conditioning
14. CWS- Condenser Water Supply
15. ZDO- ZigBee Device Object
16. ZDP- ZigBee Device Profile
17. BACnet- Building Automation and Control networks
18. MAC- Media Access and Control
19. PDU-Protocol Data Unit
20. DD- Device Discovery
21. OD- Object Discovery
22. SD- Service Discovery
23. DHCP- Dynamic Host Configuration Protocol
24. AF- Application Framework
25. APS- Application Support Sublayer
26. FH- Fairhair
27. KVP- Key Value Pair ACK- Acknowledgement
28. CLI- Command Line Interface
29. API- Application Programming Interface
30. HSE- HUE System Emulator
31. ZCLIP- ZigBee CLuster over IP
32. URI- Universal Resource Identifier
33. ZCL- ZigBee Cluster Library
34. ZLL- ZigBee Light Link
35. URN- Uniform Resource Name