Internet Access Possibilities in Bluetooth Low Energy Sensors

Joakim Klasman
ael10jkl@student.lu.se

u-blox
Östra Varvsgatan 4, 211 75 Malmö

Advisor: Jens Andersson

February 22, 2017

Printed in Sweden
E-huset, Lund, 2017

Abstract

The purpose of this Master Thesis is to evaluate different solutions that enable Internet access to Bluetooth Low Energy sensor networks as well as provide an example implementation using one of the solutions evaluated. Three distinct solutions are observed from a set of attributes and applied to three different use cases of Bluetooth Low Energy sensor networks. The solutions evaluated are the HTTP Proxy Service, the GATT REST API, and 6LoWPAN applied to Bluetooth Low Energy devices. The new Bluetooth Mesh technology is also introduced and assessed for how it may affect the future of the three evaluated solutions and the Bluetooth enabled Internet of Things paradigm as a whole. A simplified version of the HTTP Proxy Service is implemented in a **u-blox** ODIN-W2 gateway device allowing HTTP messages to be sent via Bluetooth Low Energy. All three solutions are shown to have discrete means of controlling the flow of information in the networks, as well as varying in both complexity and security, proving each solution to excel for one of the three use cases assessed.

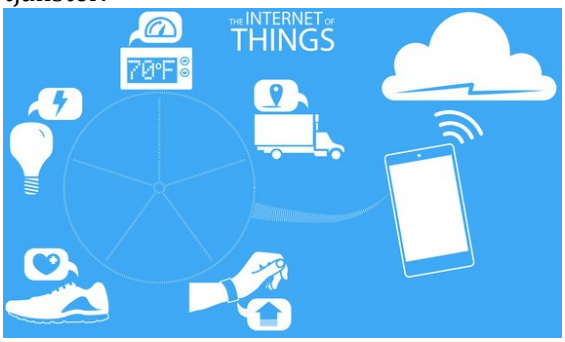
Acknowledgements

I would like to thank my supervisor at Lund University, Jens Andersson for aiding me in the planning and structuring of my work as well as providing me with valuable feedback during the course of the thesis. I would also like to express my gratitude to Mats Andersson, my supervisor at **u-blox** who contributed with valuable insight and direction as the work progressed and Andreas Larsson who shared his expertise of the **u-blox** modules and proved invaluable to the implementation stage of the thesis.

Internet of Things för Bluetooth sensor nätverk

Joakim Klasman
Institutionen för Elektro- och Informationsteknik,
Lunds Universitet

”Internet of Things”, eller ”sakernas internet” som det kallas på svenska, är ett uttryck fått stor uppmärksamhet de senaste åren. I dagens informationssamhälle skapas ständigt nya anslutningar och utvecklingsområden, där Internet of Things är ett av de senaste stora genombrotten. Internet of Things innebär förenklat att man kopplar upp vardagsföremål till internet och därmed utökar deras funktionalitet och användbarhet. Man kan till exempel koppla upp lampor och termostater för att enkelt och trådlöst reglera temperatur och belysning hemma, allting styrt från en hemsida eller mobil app. Ett annat vanligt förekommande användningsområde är stegräknare och pulsband som kan skicka information till olika fitness-appar och tjänster.



För att åstadkomma detta finns en mängd olika kommunikationstekniker, varav ett av dem är Bluetooth. Bluetooth är ett trådlöst kommunikationsprotokoll som kan arbeta under väldigt energisnåla omständigheter. Många känner nog igen Bluetooth via sina mobiltelefoner, då det ofta används för trådlös uppspelning av ljud via t.ex. trådlösa headset eller högtalare. Bluetooth kan användas för att skicka all tänkbar sorts information, och kan därför appliceras på en stor mängd områden.

Examensarbetet som utfördes undersökte just hur man kan åstadkomma denna uppkoppling av vardagsföremål till internet med hjälp av Bluetooth. Arbetet undersökte tre olika tekniker som finns tillgängliga idag; 6LoWPAN, GATT REST API och HTTP Proxy Service. Alla tre tekniker har vissa likheter men skiljer sig i framförallt två områden: hur information skickas och tas emot, och vilket transportprotokoll som används för att skicka information ut på internet.

Just hur informationen skickas och tas emot är en väldigt viktig aspekt av lösningen eftersom det bestämmer vem som styr flödet av informationen. Under arbetet fann man att alla tre tekniker lämpades bäst för var sitt användningsområde. 6LoWPAN är bäst lämpat för när alla inblandad parter ska kunna styra informationen, medan HTTP Proxy Service är optimalt då den uppkopplade saken ska vara i kontroll, och GATT REST API är att föredra om en användare själv ska kunna styra informationsflödet.

När det kommer till själva transporteringen av informationen ut på internet använde sig GATT REST API och HTTP Proxy Service av HTTP för att skicka informationen. HTTP är det vanliga sättet för information att skickas över internet, och används t.ex. när man hämtar hem en hemsida. 6LoWPAN har även stöd för IPv6, vilket är internets indentifikationssystem. Detta medför att 6LoWPAN klarar av att erbjuda mer funktionalitet, så som ökad säkerhet vid transporten av information.

En ny Bluetooth teknik vid namn Bluetooth Mesh som just nu är under utveckling undersöktes även och visar på stor potential att konkurrera med de nuvarande existerande teknikerna.

För att ge ett mer konkret exempel gjordes även en implementation av HTTP Proxy Service som kunde visa hur en av lösningarna fungerar i praktiken. Förhoppningen är att arbetet kan bidra till att underlätta val av tekniker vid utveckling av Internet of Things lösningar som använder sig av Bluetooth.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This thesis explores and evaluates current and under development Bluetooth Low Energy (BLE) solutions within the Internet of Things (IoT) domain. Taking into account earlier developments and the present state of Bluetooth enabled IoT, three current IoT solutions are introduced, as well as the latest emerging BLE technology currently under development by the Bluetooth Special Interest Group (Bluetooth SIG). How and for which criteria they are evaluated is explained along with an account of the scope for this thesis.

1.1 Background

With today's increasing desire to stay connected combined with the rapid development of the IoT paradigm there is an ever growing interest in finding new ways to connect objects and circulate information in low energy devices. There has recently been a surge in the number of technologies one can use to make these connections. By researching these new technologies and their possibles areas of use this thesis intends to make a contribution to the present wave of technological development within connectivity and information transmission. The Internet of Things is growing rapidly and is expected to compromise 26 billion connected devices by 2020[1]. These devices are expected to be present in a wide range of areas such as home, health and vehicular automation[2].

Bluetooth Low Energy is one of the recent technologies that has had a key role in the research and development of the Internet of Things. Its low cost and low energy consumption as well as the widespread adoption of Bluetooth in today's computers and handheld devices makes it a very promising solution. There have been a number of studies evaluating performance aspects of BLE such as latency[3] and energy consumption[4], which are unquestionably relevant but do not offer much support in terms of how data is sent and received. An integral part of the IoT domain is the functionality of directing Internet traffic to and from low energy devices. There is however not much research performed in the different types of Internet access solutions that the BLE standard entails. Studies have been conducted that explore possible solutions for the IoT domain using BLE such as the GATT REST API[5], but the evaluation as well as comparison aspects of the solutions are omitted. There have also been studies that evaluate single solutions such as 6LoWPAN,

but do so in a broader sense and do not specifically target BLE-based networks[6]. Studies have also been conducted that compare IoT networking solutions such as NFC and WiFi with BLE[7], however the scope again is very broad and does not offer much support in terms of actual implementation. This thesis intends to give a comprehensive view of the different possible IoT solutions available for the BLE standard.

1.2 Problem Description

With the emergence of the Bluetooth Low Energy standard and the rapid increase in demand for information transmission in low energy devices, a number of new tools and technologies have materialized. These technologies vary in complexity and operation but share the ability to grant Internet access to BLE enabled devices. Due to the relative youth of many of these technologies, evaluations of them are limited. This thesis investigates what feasible solutions for enabling Internet access in BLE devices these new technologies present, and how effective the different solutions are. The thesis also expands on for which circumstances one solution is more beneficial than another and what the security implications involved are. Characteristics such as range and strength pertaining to the different solutions are also discussed. It also addresses these topics with the intention of giving a coherent account of what different solutions are available for facilitating Internet connectivity in the IoT domain over Bluetooth Low Energy.

1.3 Methodology

The thesis focuses on three current solutions for granting Internet acccess in BLE sensors. The solutions are: IPv6 over Low power Wireless Personal Area Networks (6LoWPAN), Generic Attribute Profile Representational State Transfer (GATT REST API), and the HTTP Proxy Service (HPS). The architecture behind every solution is studied to gain an understanding of each of their respective properties. The main properties taken into consideration are security, ease of implementation, scalability and usability. The solutions are then evaluated for a number of different use cases to give a more practical interpretation of the solution domain. The evaluation is performed by theoretically applying the solutions to the use cases defined in section 1.3.2, and analyzing how well they are able to fulfill the use case requirements.

The future of the different solutions is also taken into consideration with regard to the new emerging Bluetooth IoT solution currently under development by the Bluetooth SIG; Bluetooth Mesh. Finally an implementation of one of the solutions is developed and applied to one of the use cases.

1.3.1 Attributes

The solutions evaluated are researched with emphasis on the following attributes. The purpose of focusing on these attributes is to provide a structured and relevant

base for comparing the solutions with each other and do derive strengths and weaknesses of each solution.

Security
The security attribute is evaluated by analyzing what possible security measures are available for the solution. The relative strength of the security measures are examined, looking at aspects such as whether or not end-to-end encryption is possible and what different encryption techniques are available.

Implementation
The implementation attribute refers to how much effort is required to implement the solution. Aspects that are evaluated are how much effort is required to set up the solution, how much knowledge is required to perform the set up and what the requirements for the solution are.

Scalability
The scalability attribute expresses how open the solution is to scaling. Aspects that are evaluated are how devices are added to the system, how much effort is required to add new functionality to the gateway and/or connected devices, what different network topologies are supported and whether or not the solution supports meshing.

Usability
The usability attribute concerns how easily operated, adaptable and manageable a solution is. Aspects that are evaluated are what possible areas of use the solution has, how well does the solution adapt to different situations, and what restrictions the solution has.

1.3.2 Use Cases

The thesis defines three use cases that each solution is applied to. The use cases all consist of a gateway implementing the solution, and one or more BLE enabled devices connected to the gateway. The gateway is assumed to be connected to an external power supply, as well as being BLE enabled and having a stable Internet connection.

Single Sensor Push
The Single Sensor Push use case consists of a single BLE enabled sensor connected to the gateway. In this use case the BLE sensor sends information to the gateway, which then posts the information to an Internet service such as a Website. The BLE sensor is assumed to send information to the gateway at regular intervals, with external events also triggering the sensor to send information to the gateway. A practical example is a temperature sensor sending the measured temperature at a set interval, and where a warning message is sent when the temperature reaches a certain value.

Single Sensor Pull
The Single Sensor Push use case consists of a single BLE enabled sensor connected to the gateway. In this use case an external service such as a

Fig. 1.1: Single Sensor Push Use Case

Website or mobile application is used to request information from the sensor via an Internet connection to the gateway. The external service should be able to request data from the sensor at any time. The external service should also be able to post information to the sensor. A practical example is a BLE enabled lightbulb connected to a gateway accessible through a mobile application. Through the mobile application, a user is able to request the status of the lightbulb (on/off) and change the status of the lightbulb.

Fig. 1.2: Single Sensor Pull Use Case

Multiple Sensor Network

The Multiple Sensor Network use case consists of multiple BLE enabled sensors connected to the gateway. In this use case a network of multiple sensors is controlled through an external service. The external service should be able to send and request data to and from the sensors. The sensors should also be able to push data to the service as well as be able to send data triggered through external events. A practical example is a home automation setting where a number of sensors are connected simultaneously and can interact with each other through the gateway and can be controlled through a mobile application.

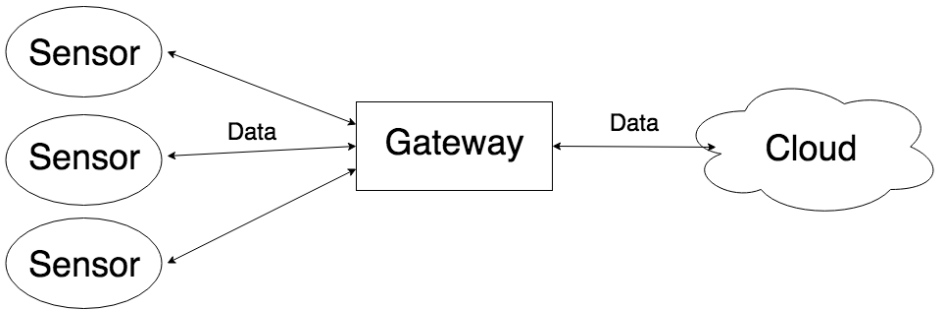


Fig. 1.3: Multiple Sensor Network Use Case

1.4 Limitations

There are a number of different methods and IoT solutions available that are able to grant Internet access to BLE sensors. Due to the scope of this study three different solutions have been chosen. These solutions were chosen in part because they are deemed most relevant and also because they all conform to established standards; 6LoWPAN is developed by the Internet Engineering Task Force, while the GATT REST API and HPS are maintained by the Bluetooth SIG.

1.4.1 Solution Criteria

The study focuses specifically on solutions utilizing the Bluetooth Low Energy standard and therefore does not choose to take into consideration Internet of Things solutions such as Zigbee, Z-wave and Thread which all operate according to different wireless technology standards such as IEEE 802.15.4 in the case of Zigbee and Thread. Also, solutions that are tied to a certain ecosystems such as Apple Homekit are not evaluated. The desired solutions are intended to be as general as possible, allowing the use of different ecosystems within the implementation. Due to these criteria, ecosystem-locked solutions such as Apple Homekit are not desiderate.

1.4.2 Attribute Criteria

The study focuses on evaluating the following attributes: security, ease of implementation, scalability and usability (defined in section 1.3.1). The study is oriented to evaluate the different solutions from a practical perspective and therefore focuses primarily on these four attributes. Power consumption is not taken into consideration as all communication is done over Bluetooth Low Energy and the gateway where the solutions is implemented is assumed to be connected to a power source. These attributes are chosen because they highlight the differences between the respective solutions. The different solutions use different modes of operations which affect aspects such as what security features are available, how devices connect and communicate within the sensor network, and how the sensor networks can be accessed over the Internet, whereas aspects such as power consumption and range generally are more influenced by the device hardware.

1.4.3 Use Case Criteria

The use cases are intended to be as broad as possible, offering application examples in multiple areas. The use cases do not define specific usage areas, but instead focus on differences in how data is transmitted within the sensor network. The reason for this is to have the use cases applicable for a vast number of relevant real life scenarios and therefore no specific information about the type of sensors or gateways is given.

Chapter 2
Theory

The study focuses on a predetermined set of solutions which utilize the Bluetooth Low Energy standard for granting Internet access to BLE sensors which form the theoretical base of the study. The Bluetooth Low Energy standard and the chosen set of solutions are described in this chapter.

2.1 Bluetooth Low Energy

Bluetooth Low Energy is developed by the Bluetooth SIG and emerged in 2010 with the release of the Bluetooth Core Specification Version 4.0. It is an expansion of classic Bluetooth but designed for minimal power consumption. BLE has many similiarities with classic Bluetooth but is generally considered a seperate technology with a slightly different method of operating. While most information transmission technologies tend to gravitate towards increasing speeds, BLE takes a different approach by focusing on minimal power consuption and minimal cost, where high-rate data transfers become a trade-off. BLE is intended to be able to connect devices that historically have not featured any wireless technology. Because of the low data transfer rates in BLE, many use cases associated with classic Bluetooth cannot be applied to BLE. Some examples of these use cases are audio streaming to Bluetooth enabled headsets and speakers, and file transfers.

2.1.1 Device Types

There two different types of BLE devices, single-mode and dual-mode. Single-mode devices are only able to support BLE communication, while dual-mode devices are able to communicate both over classic Bluetooth and BLE. Most modern computers, tablets and phones are examples of dual-mode devices. Single-mode devices generally comprise of devices with low energy consumption and computing power such as BLE enabled sensors and fitness bands.

2.1.2 Architecture

The BLE architecture is split into three parts: Application, Host and Controller. The Controller is the lowest level of the architecture and is responsible for sending, receiving and interpreting radio signals. On top of the Controller is the Host, which

is a software stack that defines the intercommunication between BLE devices. It is in this layer that the BLE services and attributes are defined, as well as how and with who devices are able to connect. The uppermost layer is the Application, which uses the Host layer to enable different use cases. This study focuses on the upper layers of the architecture, namely the Application layer and the upper parts of the Host; the Generic Access Profile (GAP) and the Generic Attribute Profile (GATT). Fig. 2.1 shows the different layers and their respective parts.

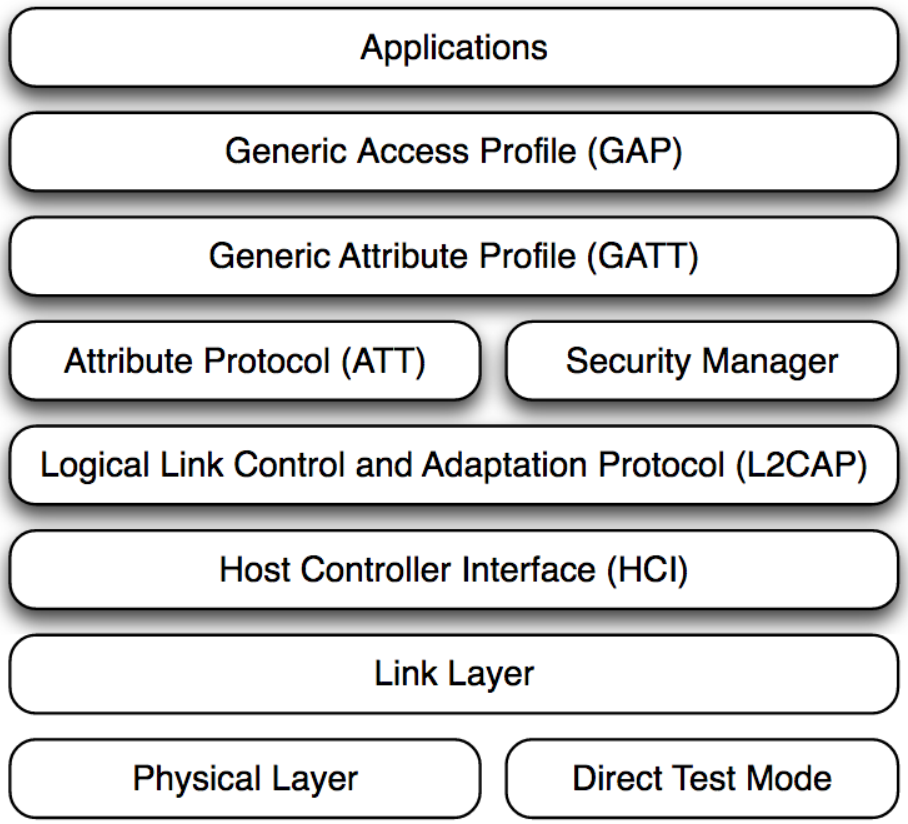


Fig. 2.1: The Bluetooth Low Energy Architecture

2.1.3 Generic Access Profile

The Generic Access Profile (GAP) controls the connections and advertising in BLE. It is through GAP that devices are made discoverable to other devices. GAP is also responsible for establishing connections between two devices, and determines how two devices interact with each other. Within GAP, devices are divided into two different device types: Peripheral and Central.

Peripheral

Peripheral devices are small low power devices that connect to a more com-

plex and powerful central device. Peripheral devices are generally single-mode devices due to their low power consumption.

Central
Central devices act as a gateway for the peripheral devices and often maintain connections to multiple peripheral devices simultaneously. Central devices are generally dual-mode devices, but can be single-mode as well.

The Peripheral devices communicate to the Central devices by sending advertising packets at set intervals known as Advertising Data payloads. This process is known as Broadcasting. The Central devices within range receive the advertising packets and can in return request more data from the Peripheral device returning a Scan Response Request. The peripheral device then responds by sending the requested additional data in a Scan Response Data payload. Through this process, a connection between the devices can be established, enabling GATT services.

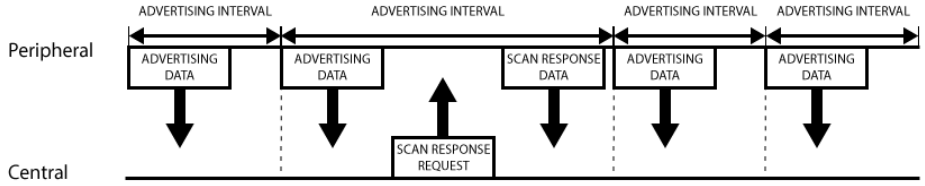


Fig. 2.2: Generic Access Profile Advertising

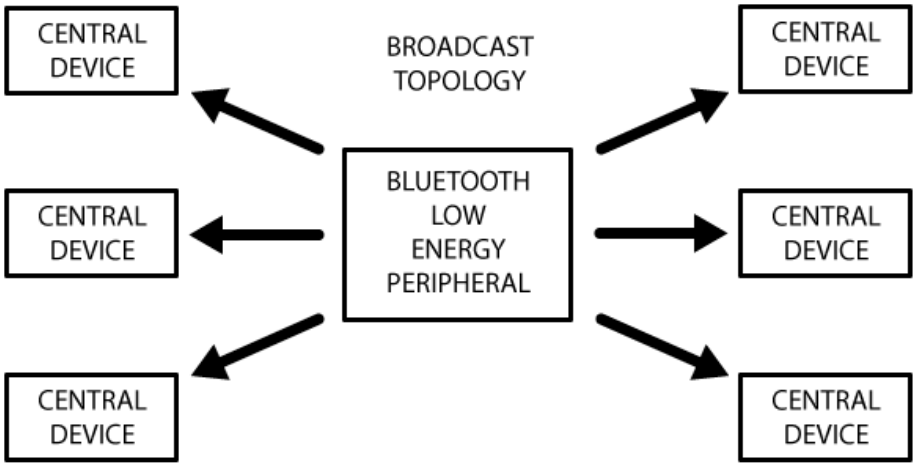


Fig. 2.3: Generic Access Profile Broadcasting

2.1.4 Generic Attribute Profile

The Generic Attribute Profile is responsible for managing data transfers between connected devices. It is built upon the Attribute Protocol (ATT), which stores the

attribute data used by GATT identified through unique handles. The attributes also contain a type identifying the data stored in the attribute and the data value of the attribute. GATT describes these attributes and how they are used. The main attributes described by GATT are Characteristics, Services and their relationships.

Within GATT the Peripheral device is known as the GATT server and holds all the ATT data as well as the attribute definitions. The Central device is known as the GATT client. The GATT client acts as the Master device and initiates data transactions between the client and the server by sending data requests to the server containing the data. The server acting as Slave then responds by sending the requested data.

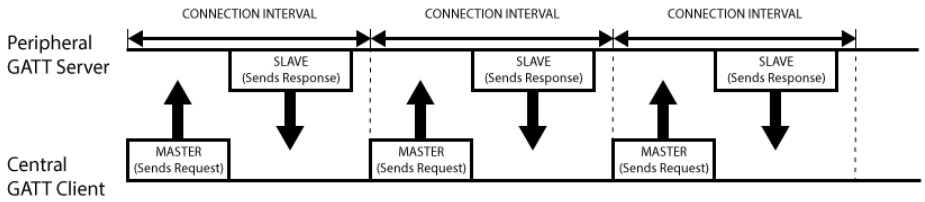


Fig. 2.4: Generic Attribute Profile Transaction

2.1.5 Application layer

The Application layer is the uppermost layer of the BLE architecture. It is in this layer that the actual use cases for the BLE devices are defined. Within the Application layer the main attribute specifications defined in the GATT profile are used; Services and Characteristics. The attribute specifications are labeled by Universally Unique Identifiers (UUIDs). Another important part of the application layer is a concept known as Profiles.

Characteristic
: Characteristics are unique specific portions of data containing the values and permissions for the different features of the peripheral device. The Bluetooth SIG has defined a set of standard characteristics which are free to use, but it is also possible to define new characteristics designed specifically for a certain device or software. Characteristics are the main point of interaction between the peripheral and central device. It is through Characteristics that values can be read from and written to the peripheral device. Characteristics can also contain descriptors, which provide metadata to the client.

Service
: Services are the container for Characteristics and are used to divide data into logical entitites. Services come in two types: Primary and Secondary. Primary Services provide the main functionality of the device as seen by the user. Secondary Services are used by Primary Services to add increased

functionality to the Primary Service. A Secondary Service must be accessed through a Primary Service.

Profile

Profiles describe the actual use cases incorporating two or more BLE devices. Within the Profile the used Services are described as well as how the devices within the Profile discover and connect with each other. In other words, the Profile describes how the devices interact with each other within the use case.

The Application layer architecture and an example of a BLE use case is shown in Fig. 2.5 and 2.6 respectively.

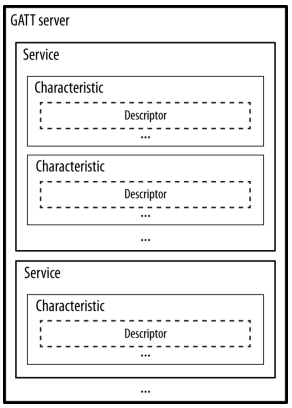


Fig. 2.5: Application layer architecture

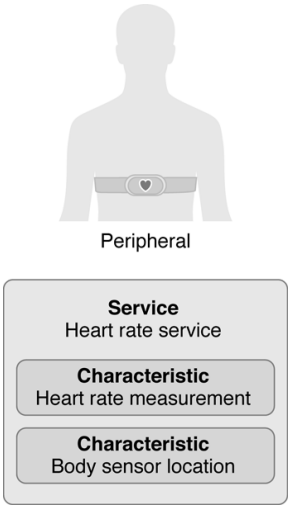


Fig. 2.6: Example of a Heart rate monitor BLE device

2.2 GATT REST API

The GATT REST API is an Application Programming Interface (API) used to facilitate the handling of data transmitted between BLE devices. The API implements the Representational State Transfer (REST) architecture, which is used to read and write to and from BLE devices using the GATT profile (see section 2.1.4). The GATT REST API is developed and maintained by the Bluetooth SIG.

2.2.1 Representational State Transfer

Representational State Transfer (REST) is a software architecture, which is used for designing network applications. REST relies on the Hypertext Transfer Protocol (HTTP) to read and write data between workstations using a stateless client/server architecture. Data is accessed using HTTP requests to simple Uniform Resource Identifiers (URIs) known as Resources, which reference to the data on a server implementing REST architecture. These requests operate using the same HTTP requests used when retrieving Websites, namely GET, POST, PUT and DELETE.

2.2.2 Accessing GATT through REST

To access the GATT services from a BLE device over REST, a gateway is needed that acts as the server and implements the GATT REST API. The gateway manages connections to peripheral devices using the GAP REST API (see section 2.2.3). The GATT REST API enables users to read and write data to and from peripheral BLE devices connected to the gateway. To access the data on a connected peripheral device, the appropriate URI Resource for the peripheral is required. The desired action for the data is defined by the type of HTTP request. The HTTP request is sent to the URI of the data the user wishes to access. The structure of the Resource URIs in the GATT REST API follow the GATT architecture (see section 2.1.5) and are described below:

Nodes
: The Nodes Resource contains the data for all the available connected devices. The nodes can either be requested to see which nodes are available, or a specific node can be accessed using its id.

Services
: The Services Resource lists all the available GATT Services pertaining to a specific device. To access the Services Resource a Node Resource must first be specified.

Characteristics
: The Characteristics Resource represents the GATT Characteristics within a GATT Service. To access the Characteristics Resource, a Service Resource must first be specified.

The HTTP example below demonstrates how to access the value of a characteristic from a connected peripheral device:

```
GET http://<gateway>/gatt/nodes/<node>/services/<service>/
characteristics/<characteristic>/value
```

In the example the HTTP command GET is sent to retrieve information from a device, where `<gateway>` is the address of the gateway, `<node>` is the identifier of the peripheral device connected to the gateway, `<service>` is the name of a service on the peripheral device, and `<characteristic>` is the name of the characteristic containing the value the user wishes to access.

GAP REST API

The GAP REST API is an Application Programming Interface (API) used in conjunction with the GATT REST API and is responsible for administering connections between a gateway and BLE devices using BLE's GAP profile. The GAP REST API also follows the REST architecture and allows for discovering, connecting and disconnecting to and from BLE enabled devices within range of the gateway. The API only uses the Nodes Resource for managing connections. The GAP REST API is simpler than its GATT counterpart and only supports the HTTP commands GET and PUT.

Enabled Nodes
: BLE devices that have established a connection with the gateway are known as Enabled Nodes. The gateway automatically reconnects to Enabled Nodes and will try to establish a connection to all Enabled Nodes when possible.

The HTTP example below demonstrates how to add a BLE device to the list of Enabled Nodes:

```
PUT http://<gateway>/gap/nodes/<node>?connect=1
```

In the example the HTTP command PUT is sent in order to initiate a connection with a BLE device, where `<gateway>` is the address of the gateway, `<node>` is the identifier of a peripheral device in range, and setting the connect option to 1 requests the gateway to connect to the peripheral device.

2.3 6LoWPAN

IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) is a protocol definition maintained by the Internet Engineering Task Force (IETF). The goal of the 6LoWPAN standard is to facilitate transmission of Internet Protocol version 6 (IPv6) packets over Low Power Wireless Networks (LoWPAN). 6LoWPAN is originally intended for IEEE 802.15.4 based networks, but its techniques can also be applied to Bluetooth Low Energy[16].

2.3.1 Internet Protocol version 6

Internet Protocol version 6 is an Internet Protocol (IP) developed by the IETF responsible for identification of computers and networks on the Internet. It expands

upon the earlier Internet Protocol version 4 (IPv4) and emerged as a response to the rapid growth of the Internet. The main improvements IPv6 brings are: extended address space, autoconfiguration of IP addresses, simplification of the header format, and improved support for options and extensions. While all four of these improvements are welcome, the most crucial is the extended address space, allowing an address format consisting of 128 bits instead of the IPv4's 32 bits. This solves the main issue with IPv4, namely running out of available IP addresses, which in turn plays an important part in the Internet of Things paradigm due to the influx of Internet connected devices.

2.3.2 Low Power Wireless Personal Area Networks

Low Power Wireless Personal Area Networks (LoWPAN) refers to networks consisting of devices that are distinguished by their short range, low data transfer rate, low power and low cost. The devices generally include sensors, which allows for physical real-world applications. LoWPANs have the following key characteristics:

Low Power
The majority of devices within the networks are generally battery operated, which gives the need for low power consumption within the devices in order to maintain extended operability.

Low Cost
Devices are commonly low powered sensors with very limited computing power and as such tend to be associated with a low cost.

Small packet size
As a consequence of the limited computing power and power consumption within the networks, the size of the data packets that are transmitted are also limited. 6LoWPAN supports compressing IPv6 headers into smaller frames which aids in maintaining smaller packet sizes.

Low bandwidth
LoWPANs are designed for transmitting small amounts of data at minimal cost and therefore widely operate on low bandwidth.

Energy Conservation
In order to conserve energy and maintain their low power consumption, devices often sleep for long periods of time where they are unable to communicate.

Topologies
LoWPAN supports star and mesh networking topologies.

2.3.3 Constrained Application Protocol

The Constrained Application Protocol (CoAP) is a Web transfer protocol designed for constrained low power networks such as 6LoWPAN. The intended use of the protocol is for Machine-2-Machine (M2M) communication in application areas such as healthcare, energy conservation and automation. 6LoWPAN supports fragmentation of IPv6 packets into smaller frames in order to keep packet sizes small but

this also entails reduction in package delivery probability. By keeping the message overhead small CoAP reduces the need for fragmentation. CoAP is built on the REST model (see section 2.2.1) and operates on a request/response interaction like that of HTTP. CoAP is by default transfered over the User Datagram Protocol (UDP) but has support for transfers over other protocols as well such as the Transmission Control Protocol (TCP).

2.3.4 6LoWPAN with Bluetooth Low Energy

Due to the similiarities between BLE and 6LoWPAN, many of the 6LoWPAN standards can be applied to BLE as well in order to allow for transportation of IPv6 packets over BLE. This is possible in part due to the release of the Internet Protocol Support Profile (IPSP) by the Bluetooth SIG, which allows for establishment of a link layer connection for transporting IPv6 packets. In order to transport IPv6 packets, BLE version 4.1 and ISPS version 1.0 are required. Originally a major difference between 6LoWPAN for IEEE 802.15.4 and 6LoWPAN applied to BLE was within network topology, more specifically BLE only supporting star topology, but recent developments have been made allowing for transmitting IPv6 packets over BLE to also support mesh topology[17].

In order to incorporate support for IPv6 in BLE some additions need to be made in the BLE stack, namely having an IPv6 stack working in parallel to the GATT stack in the host layer (see Fig. 2.7).

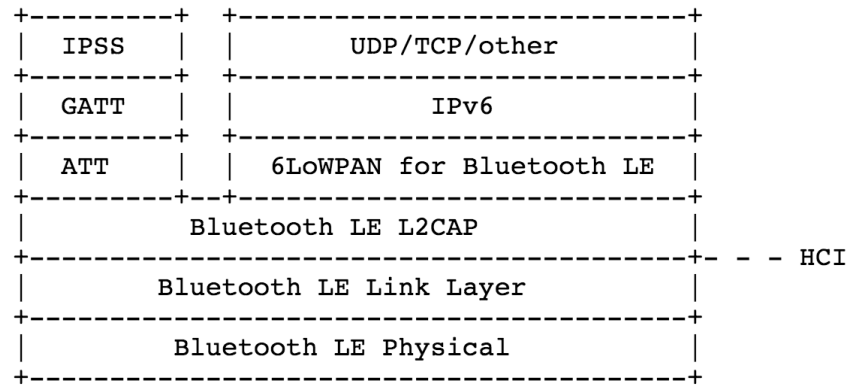


Fig. 2.7: 6LoWPAN on BLE stack

The GATT stack is necessary for discovering nodes that support ISPS. UDP and TCP are viable upper layer protocols but other transport protocols are also possible. The 6LoWPAN layer is adapted specifically for transmission of IPv6 packets over BLE. The IPSP is responsible for setting up a BLE connection adhering to 6LoWPAN standards.

2.4 HTTP Proxy Service

The HTTP Proxy Service (HPS) is a specification defined by the Bluetooth SIG that allows for peripheral BLE devices to communicate with Web services through a gateway device. The HPS uses the GATT profile to create and dispatch HTTP requests and read responses. Because peripheral BLE devices often work in constrained environments, peripheral devices implementing HPS need to restrict the header and body sizes of the HTTP requests in order to be able to process them. The header and body need to fit within the maximum size for GATT characteristics.

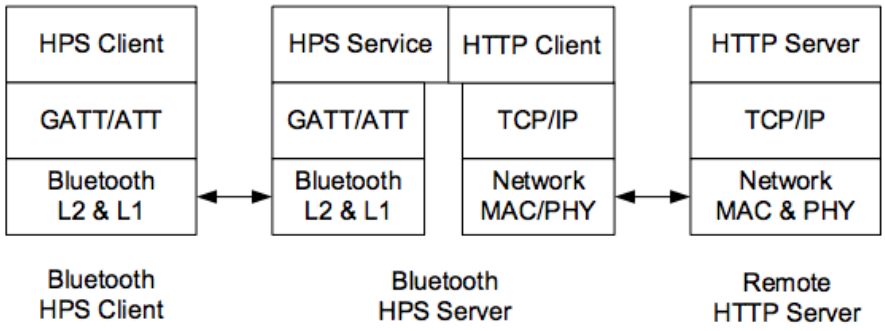


Fig. 2.8: HTTP Proxy Service Architecture

2.4.1 Service Definition

The HPS is a primary service with no other service dependencies. The HPS consists of the following characteristics:

URI
: The URI characteristic defines a Uniform Resource Identifier (URI) used for identifying HTTP requests. This characteristic must be written before a HTTP is request executed.

HTTP Headers
: The HTTP Headers characteristic defines the headers to be sent with the HTTP requests and responses. The value in the characteristic should correspond to the general-header, request-header and entity-header fields for HTTP requests/responses.

HTTP Entity Body
: The HTTP Entity Body characteristic contains the payload of the HTTP message. The value in the characteristic corresponds to the HTTP message-body field of a HTTP message. When a HTTP request is sent, the payload of the message must first be written to the characteristic. When a HTTP message is recieved, the message-body of the recieved HTTP message is written to the HTTP Entity Body characteristic.

HTTP Control Point
The HTTP Control Point characteristic is responsible for initiating the transmission of a HTTP request from the gateway implementing HPS, and also acting as HTTP client. The characteristic is a one octet long value that indicates which type of HTTP request to be sent. Supported HTTP requests are: GET, HEAD, POST, PUT, DELETE. It is also used to cancel a current HTTP request. Only one HTTP request can be active at a time.

HTTP Status Code
The HTTP Status Code characteristic holds the Status-Code from the last received HTTP response message. It also contains a Data Status bit field in the form of one octet, which indicates the status of the data received.

HTTPS Security
The HTTPS Security characteristic allows for using HTTPS when sending requests. The characteristic contains the known authenticity of the HTTP Server certificate for the URI.

Values are read from the peripheral devices and transfered to the HPS characteristics. This allows for the gateway to be completely generic, where any peripheral device can connect to the gateway and send data over HTTP without needing to configure the gateway. However, the peripheral devices must be configured to send the necessary data and in the correct format. In other words, the peripheral devices are responsible for formatting the HTTP body and managing HTTP headers, HTTP request type and destination URI.

2.4.2 Client/Server interaction

The necessary HTTP message characteristic values are sent from the peripheral device to the HPS implementing gateway, then the control point characteristic is sent to initate the HTTP request. The gateway compiles the characteristics into a complete HTTP request, which it then sends to the intended recipient. The gateway notifies the peripheral device of the status of the HTTP request and finally, if required, the response message is read and processed by the peripheral.

2.5 Bluetooth Mesh

Bluetooth Mesh is new Bluetooth technology currently under development by the Bluetooth SIG. The main idea behind Bluetooth Mesh is to enable interoperable mesh networking for BLE enabled networks. The solution enables messages to be sent from one device to one or more other devices by relaying the messages between nodes in the network.

2.5.1 Mesh Networking

Mesh networking is a network topology consisting of several nodes where each node can relay data for the entire network and the nodes thereby work together to distribute data across the network. Mesh networking have two methods of relaying

data, either through Flooding or through Routing. Initially Bluetooth Mesh only supports flooding based mesh networks with routing based networks arriving with a later release[18].

Flooding

Flooding based networks use broadcast channels to transmit messages to other nodes which in turn relay messages by retransmitting them, thereby extending the range of the original message. Flood based networks "flood" the network with messages by dispatching a message to each node within range which in turn continue to dispatch the message to all respective nodes within range. Each node acts as a transmitter and also as a receiver, where the nodes try to relay every message to every node on the network except the node which dispatched the original message.

Routing

Routing based networks transmit messages along a directed path rather of nodes, where each node receiving the message decides the which node to direct the message to next in order for the message to reach its destination. How to direct messages is based on routing tables. The complexity of managing routing tables results in routing based mesh networks to be less dynamic and harder to implement than flooding based networks while offering better scalability.

2.5.2 Addressing

Smart Mesh is designed to support device numbers in the thousands. In order to accomodate this, Bluetooth Mesh addresses use an address space of 15 bits. Multiple devices can also be joined under a single address known as a group address which uses an address space of 14 bits. Bluetooth Mesh also supports two special addresses, a broadcast address that reaches all devices and an unassigned address used to signify no address is configured for a given device.

2.5.3 Security Model

Bluetooth Mesh uses two seperate keys to secure the integrity of messages. The first key is an application key which is used to provide confidentiality and authentication of application data sent between devices. The second key is a network key which is used to protect messages sent within the network. The keys are shared between devices. Any device that has access to a key can send and receive messages using that key. The reason for splitting the keys between application and network is to allow application data to be transmitted via intermediary devices and allowing them to authenticate the messages when they are relayed without the relaying devices being able to read or alter the application data sent.

2.5.4 Device Provisioning

Device provisioning in Bluetooth Mesh works differently to the standard point-to-point bonding in BLE. Three bearers are used in Bluetooth Mesh in order to provision devices. They are: Provision over Advertising, Provision over GATT and Provision over Network. Provision over Advertising is the mandatory form of provisioning used to connect to new devices, with Provision over GATT offering legacy support for BLE devices and Provision over Networking enabling remote provisioning to distant preinstalled devices in the mesh network. Existing GATT services do not function over Bluetooth Mesh.

2.5.5 Protocol Stack

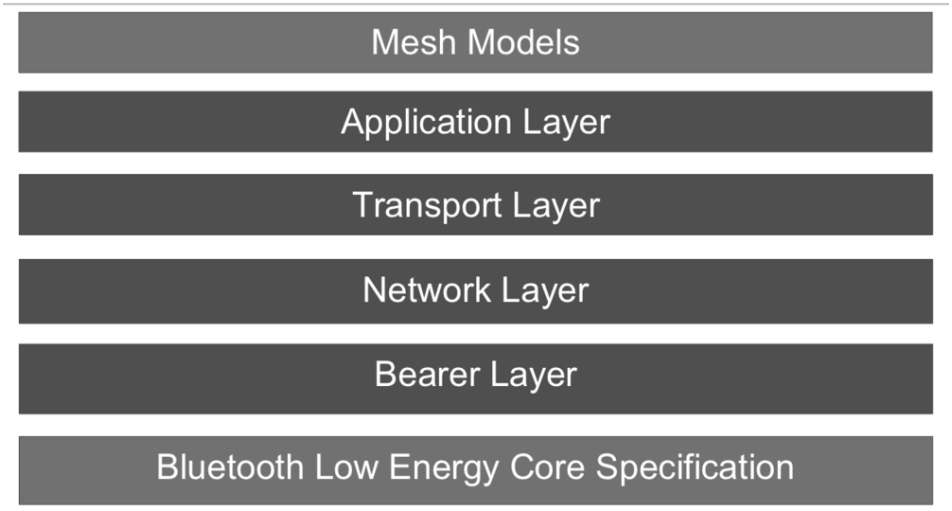


Fig. 2.9: Bluetooth Mesh Protocol Stack

Application Layer

The application is responsible for defining how higher level applications use the transport layer. It is also in this layer that application operation codes, parameters, states and behaviors are defined.

Transport Layer

The transport layer defines encryption and authentication procedures of dispatched and received messages, ensuring security and confidentiality.

Network Layer

The network layer specifies how addressing of transport messages is performed for nodes within the network. The format of messages is also defined in this layer as well as whether or not to relay messages as well as encrypting node addresses to provide confidentiality.

Bearer Layer

The bearer layer defines how messages are transported between nodes within the network.

2.5.6 Architecture

Models

The basic functionality of a node is described as a collection of models. A model determines the states and operations possible for a node, as well as messages and data used by the node and any associated behavior.

Addressing

In order to be able to identify nodes within the network each node is assigned a unique address when it is provisioned on the network. Nodes may also be allocated the broadcast address, the unassigned address and subscription and group addresses.

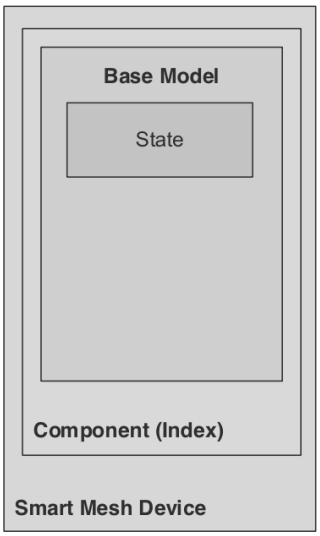


Fig. 2.10: Bluetooth Mesh Node Architecture

Components

A component is a set of non-overlapping models exposed in a node. A non-overlapping model is defined as a model that requires a set of operation codes no other models within the same component require. A node needs to consist of at least one component. Components are needed since some nodes may require a model to be implemented multiple times as the node may need to expose multiple instances of a model. By seperating the models into components multiple instances of the same model can avoid conflicting.

Publish/Subscribe

The messages transmitted over Bluetooth Mesh networks follow a publish-subscribe paradigm. This means that nodes that generate messages will publish data and nodes wanting to use the data subscribe to the publishing nodes. The address a node sends a message to is a publish address. Each model within a node has a single publish address and each model within a node can subscribe to one or more addresses. A node can have any number of subscriptions. This allows for nodes to respond to messages published to different addresses.

Messages

All messages dispatched in the network are done so from a single unique either node or component address and directed to a destination address that can be either a node, component, group or broadcast address (i.e. all nodes). Messages are encrypted using two separate keys to ensure privacy (see section 2.5.3).

2.5.7 Topology

Nodes within the Bluetooth Mesh network operate either as Edge or Relay nodes.

Edge Node

Edge nodes are able to transmit and receive messages but cannot forward them to other nodes. In other words, edge nodes are only capable of creating or consuming messages and can only use existing relay infrastructure to transmit and receive messages. They do not actively contribute to the mesh network with the benefit of functioning under greater resource constraints.

Relay Node

Relay nodes are capabe of transmitting, receiving as well as forwarding messages in the network. By being able to forward messages, relay nodes help to extend the overall range of the network. The relayed messages are authenticated using network keys. Stationary devices with a secure power supply are optimal for this role.

Chapter 3
Discussion

The different solutions are researched in regard to their history and characteristics as well as their possible methods of operation, areas of use and viability in terms of future developments within the realm of IoT. All solutions are observed from the same set of attributes, facilitating a juxtapositioned comparison to determine their respective strengths and weaknesses. Finally the solutions are assessed in relation to the new Bluetooth Mesh standard.

3.1 GATT REST API

The GATT REST API provides a solution based on the well established REST architecture used by many Web applications today. It allows for peripheral devices connected to a gateway device with the GATT REST API implemented to be accessed through simple URI's much like a Website is accessed through a Web browser. This provides a simple way to remotely access and pass information to and from peripheral devices in a BLE network. Basing the flow of information on accessing the devices through URIs does however limit the solution in terms of how responsive the solution can be to events and changes within the BLE network.

3.1.1 Implementation Considerations

The GATT REST API allows for communication with the gateway either over HTTP or HTTP Secure (HTTPS). When HTTPS is used BLE's link encryption is implemented on the connection between the gateway and peripheral. If HTTP is used no encryption is required on the link between the gateway and peripheral. While HTTPS is a widely used and accepted standard it is not without issues[9]. It is therefore important to be aware of the weaknesses of HTTPS if security is a vital component of the use case. BLE's link encryption is also widely used but has been shown to be suspectible to compromisation of both confidentiality and integrity[10]. Consequently security can be compromised in both the communication between the user and the gateway and between the peripheral device and the gateway. If the data to be transmitted is of sensitive nature it is therefore advisably to look to alternative solutions, preferably that offer end-to-end encryption.

The GATT REST API is defined by the Bluetooth SIG and follows the REST-

ful architecture. Using the GATT REST API puts a number of demands on the gateway, namely it needs to be able to function as a HTTP server. While there exist available GATT REST APIs for some gateway devices such as the Raspberry Pi[19], for many gateway devices the API has yet to be developed. This results in either needing to develop the GATT REST API for the current gateway device, or using a gateway device where the GATT REST API is already implemented.

On the client side some form of application must be implemented to consume the RESTful services in the gateway. An example of a common client-side implementation would be a mobile application or a Website which sends HTTP requests to the gateway resources. Most modern programming languages and major platforms support consuming RESTful Web services such as Python, Javascript, Android, iOS, and C#[11][12][13] [14][15].

3.1.2 Application Considerations

The number of simultaneous connections a gateway can have is set to 7 active peripheral devices and 255 inactive peripheral devices, known as a piconet. In order to scale a solution using the GATT REST API it is therefore necessary to add more gateways to the network. Adding gateways to the solution increases the number connections possible in the network and can also increase the range of the overall network, which in turn results in the need to manage multiple gateway endpoints. This does not in itself pose a problem on the client side. The client simply needs to be aware which endpoint grants access to which peripheral device.

The GATT REST API implements the GATT architecture in its endpoint definitions which makes for straightforward endpoints coherent with the BLE attribute architecture. Due to the limited number of resource endpoints (see section 2.2.2) accessing data follows a comparatively simple hierarchy. A clear limitation with the GATT REST API however is the unidirectional form of communication. The GATT REST API supports requesting data from the peripheral device via the gateway and through the use of server-initiated updates. The peripheral device can send data to the gateway to be cached, but the GATT REST API does not define a method for sending the data from the peripheral all the way to the client. This limits the number of applicable use cases for the GATT REST API.

3.1.3 Summary

The GATT REST API is a straightforward solution for granting access to BLE devices over the Internet based on established architecture in the form of REST. The solution unfortunately only sees a limited number of applications due to its focus on client initiated communication. The solution also poses some security risks allowing for possible security issues in both the connection between the gateway and client, and between the peripheral devices and the gateway. Use cases involving a large number of gateways may also constitute a problem as it requires clients to manage a large number of endpoints. Scaling up or down the solution however is a relatively simple process merely involving adding or removing gateways and

informing the client of the new endpoint changes.

3.2 6LoWPAN

6LoWPAN is a set of characteristics defined by the IETF for how low power wireless networks can effectively transmit IPv6 packets in constrained environments. 6LoWPAN is not specific to BLE but can be applied to other wireless technologies as well such as WiFi and the IEEE 802.15.4 wireless standard. 6LoWPAN for BLE describes how the BLE stack can work together with an IP stack to transmit IPv6 packets between BLE enabled devices and the cloud. This is achieved by supplying each device in the BLE network with its own IP address, just as a computer or smartphone connected to the Internet would have.

3.2.1 Implementation Considerations

6LoWPAN can be applied to various types of networking solutions and therefore the security characteristics of 6LoWPAN depend on what the underlying networking technology consists of. In this thesis 6LoWPAN is evaluated for use with BLE and therefore much of the security properties are defined by BLE rather than 6LoWPAN in itself. 6LoWPAN does however offer one security benefit not present in the other solutions evaluted in the thesis. Due to 6LoWPAN focusing on sending and receving IPv6 packets adjusted for the constrained environments of the IoT domain, 6LoWPAN enables inter-device communication to support end-to-end encryption when sending and receving data. This increases the overall level of security of the solution by among other things greatly reducing the possibilites of man-in-the-middle attacks[21]. IETF recommends using BLE link layer encryption when transmitting IPv6 packets over BLE.

Implementing a 6LoWPAN solution for BLE sensor networks is not as straightforward as with the other solutions evaluted partly because unlike the other solutions which are specific to BLE, 6LoWPAN was originally designed for 802.15.4 networks. When applying 6LoWPAN to BLE networks there are several aspects that need to be considered such as which transport protocol to be used running on top of Ipv6 (TCP/UDP/other), configuring the link layer of BLE to the correct characteristics for transmitting IPv6 packets, configuring stateless address autoconfiguration, setting up appropriate neighbor discovery, and compressing headers according to correct formats[16]. This requires working in several layers of the BLE architecture making 6LoWPAN for BLE more comprehensive than for example setting up a HTTP Proxy Service or a GATT REST API which mainly work on the two uppermost layers of the BLE architecture.

3.2.2 Application Considerations

6LoWPAN was designed with the expectation of a large number of devices to be deployed and is therefore well adapted for scaling. Since 6LoWPAN is built on giving each device in the network its own IPv6 address this allows for easy connectivity to existing IP-based networks without the need for intermediate steps

to translate communication. The large address space of IPv6 makes it possible to support a very large number of devices and the maximum number of devices in a network rather relies on the underlying network architecture, in this case BLE[22]. Scaling a 6LoWPAN is simply a matter of connecting 6LoWPAN configured peripheral devices to 6LoWPAN configured gateways known as edge routers. Edge routers handle the data exchange between 6LoWPAN peripheral devices and the Internet, local data exchange within the 6LoWPAN network, and the generation and maintenance of the 6LoWPAN network.

6LoWPAN allows for peripheral devices to both push and pull data to and from the cloud and therefore is applicable to most use cases. 6LoWPAN is by far the solution that sees the most broad areas of application. Due to the fact that it is based the IP networking protocol providing all devices with their own IPv6 address, 6LoWPAN networks can easily be connected to current networking systems and architectures, whether it be through ethernet, Wifi, or 3G/4G[23].

3.2.3 Summary

While 6LoWPAN was originally designed for 802.15.4 networks its specifications can also be applied to BLE. 6LoWPAN is a very versatile solution offering capabilities for both peripheral devices and cloud services to control the flow of information within the sensor network. It also has beneficial security aspects as a result of the solutions focus on transfers of IPv6 packets. This allows for end-to-end encryption in the communication between devices as well as offering greater interoperability with existing systems also using IP architecture. The broad usuability of 6LoWPAN however comes at the cost of a quite extensive implementation configuration for both gateways and peripheral devices.

3.3 HTTP Proxy Service

The HTTP Proxy Service is based around the simple idea of having a BLE device act as a HTTP proxy client, as the name implies. A BLE service is simply configured to construct syntactically correct HTTP messages from information it recieves from other BLE devices, and send them through a HTTP client implemented in the same BLE device that contains the HTTP Proxy Service, in other words the gateway device. In this solution the peripheral devices who wish to send information are responsible for whether or not the correct information is sent and can be read by the intended receiver. The HTTP Proxy Service in itself simply converts the information received to a HTTP message with the appropriate structure and sends the message to the recipient specified by the peripheral device.

3.3.1 Implementation Considerations

The HTTP Proxy Service uses the GATT profile to send HTTP messages from a peripheral device via a gateway to a HTTP server. Through the HTTPS Security characteristic it has support for sending and receiving encrypted HTTPS messages allowing for greater confidentiality and integrity protection in the link between the

gateway and the HTTP server. The HPS extends the GATT profile and allows for the standard encryption options offered by BLE. As such it is suspectible to the same security concerns as the GATT REST API (see section 3.1.1) which also uses the same methods of transportation both between the peripheral and gateway, and between the gateway and end user.

The implementation of the HPS simply consists of creating a GATT service adhering to a set of specifications defined by the Bluetooth SIG. A clear advantage of this is the fact that the solution is built on the same platform as other BLE applications, namely GATT. Hence there is no need for the implementation of extended core functionality in the device as in 6LoWPAN, or development of an API like the GATT REST API in order to implement the solution. The gateway must however translate the data in the characteristics from the HPS to a syntactically correct HTTP message.

In order for the HTTP messages sent by the gateway to be read correctly the peripheral devices must be configured so that the data sent from the peripheral to the body characteristic of the HPS is readable by the HTTP server. This means the peripheral device must be configured to send its data as a valid HTTP body. How this is done varies depending on the peripheral device. This functionality can for example be hardcoded or by uploading some form of configuration to the peripheral device.

3.3.2 Application Considerations

A gateway implementing the HPS is intended to be completely generic and function with any peripheral device without any need for configuration in the gateway. Scaling the solution up from a gateway perspective is therefore simply a matter of adding HPS implementing gateways. As stated earlier, configuration for the HPS solution is almost entirely performed in the peripheral devices, which implicates that all peripheral devices must be configured to work with the HPS. Because the gateway solution is generic, any peripheral device can connect to any gateway with the same functionality. This allows for the solution to support peripheral devices being able to send data to an arbitrary gateway within range with the same end result.

The main function of the HPS solution is to allow peripheral devices to push data to the cloud, but considering that the HPS also has support for GET requests the HPS also supports polling data from a Web service. This allows for the HPS to be applicable to use cases requiring both pushing and polling of data to and from the peripheral devices. In the HPS solution all communication is initiated by the peripheral device, therefore when a user reads the data from the HTTP server the data is not retrieved directly from the peripheral device, but rather just the value of the latest data sent by the peripheral device. In other words, the values retrieved from the HTTP server may not always reflect the current values in the peripheral devices.

3.3.3 Summary

The HPS solution is built upon the same platform as other BLE device implementations, namely GATT Services and Characteristics. This allows for a comparably simple implementation using the same tools and techniques as when developing functionality for any BLE enabled device. While the solutions offers simple implementation, it comes at the cost of functionality. Since the solution only supports control of data flow from the peripheral device for many application areas the solution is not an alternative. A clear benefit of the solution however is its interoperability with different devices. A gateway implementing the HPS solution is generic, meaning the gateway itself does not require any configuration for connecting peripheral devices to transmit data to the cloud. All necessary implementation configuration is instead passed on to the peripheral devices, allowing for sensor networks based on the HPS solution to function in completely distinct ways while still having identical gateway implementations.

3.4 Comparison

Below shows an overall comparison of the attributes between the different solutions. The key aspects for each attribute is represented. The key aspects for each attribute are as follows:

Security What security protocols are available.

Implementation In which devices solutions need to be implemented.

Scalability What configuration is necessary when extending sensor network.

Usability How communication is initiated between actors in the network.

Solution	Security	Implementation	Scalability	Usability
GATT REST API	BLE Encryption, HTTPS	Gateway device only	Endpoints need to be managed for each gateway.	Client initiated communication only
6LoWPAN	BLE Encryption, HTTPS, End-to-End Encryption with IPv6	Peripheral and Gateway devices	Peripheral devices need to use same transport protocol as gateways	Client and Peripheral initiated communication
HTTP Proxy Service	BLE Encryption, HTTPS	Peripheral and gateway devices	Peripheral and gateway devices interchangeable	Peripheral initiated communication only

Table 3.1: Attribute Comparison of Solutions

3.5 Bluetooth Mesh

Bluetooth Mesh is a new Bluetooth standard under development by the Bluetooth SIG. The standard is designed with the idea of allowing a decentralized system of devices in contrast to solutions available today. However, Bluetooth Mesh is more than just a new network topology. It involves an entire new Bluetooth architecture specifically designed for the IoT domain.

Bluetooth Mesh Release

Bluetooth Mesh is to be released in several phases with each release phase adding to its functionality. The first phase focuses on the flooding-based mesh network and offers all the necessary functionality for Bluetooth Mesh such as message relaying, network security, and compatability with existing devices. The second phase will add support for routing-based mesh networking while also optimizing for power and reducing congestion of the advertising channel. The third phase will allow for larger packet sizes resulting in support for protocols such as IPv6. The fourth phase is expected to further reduce power consumption and advertising channel congestion[18].

Bluetooth Mesh Specification

Bluetooth Mesh, as the name implies, enables BLE devices to use mesh networking topology. The most obvious benefit of using mesh networking is the greatly extended possible range for a BLE network. Since the devices in the network can relay messages they don't require a Bluetooth connection with the intended receiving device. In effect this results in the fringe devices in the network determining the range of the entire network. Accomodating for both Flooding- and Routing-based mesh topologies further increases the flexibility as how the messages are relayed can be adapted to best suit the requirements of the network.

Bluetooth Mesh vs. Evaluated Solutions

By offering aspects such as a new form of security, multiple forms of device provisioning and support for an increased number of devices it could prove to be a worthy alternative to the currently available solutions. Further boosting the competitiveness of the Bluetooth Mesh standard is the fact that it will be supported by existing devices today, meaning current BLE sensor networks can implement the Bluetooth Mesh solution without the need for replacing their devices. Bluetooth Mesh will however not work with current GATT services, so there is still a certain effort of implementation that needs to be considered.

Both the GATT REST API and HTTP Proxy Service have the benefit of being maintained by the Bluetooth SIG which ensures support and compatability for BLE devices. This in contrast to 6LoWPAN which is originally not designed for BLE but rather supported through adaption of the 6LoWPAN model to fit BLE. Bluetooth Mesh joins the GATT REST API and HTTP Proxy in being maintained by the Bluetooth SIG and therefore also enjoy the benefits of guaranteed support

for BLE devices. The main limitation for both the GATT REST API and HTTP Proxy Service are their unilateral communication approach. They each emphasize one direction of communication, GATT REST API through polling data from peripheral devices while the HTTP Proxy Service focuses on pushing data from peripheral devices. Bluetooth Mesh, just like 6LoWPAN equally supports both methods of communication greatly broadening their applicable areas of use.

In regard to range, Bluetooth Mesh shows great promise to outshine the current solutions. With mesh networking, the range of the entire network can stretch far greater distances than with current BLE technology, which is constrained by the range of each device on the network. Bluetooth Mesh allows for devices to reach other devices far beyond their own BLE range of connectivity. Aspects such as security and implementation effort remain to be seen but the two layer security keys and expected support for IPv6 packets, which in effect results in end-to-end encryption, should not be understated. Assuming flooding-based mesh networking is used, scaling a Bluetooth Mesh solution is uncomplicated as nodes can easily be added, replaced and interchanged. If routing-based mesh networking is used however, scaling becomes a bit more complicated as routing tables need to be altered and updated. In this regard Bluetooth Mesh varies in its ability to scale and can in certain cases prove more complicated than existing solutions, and in other cases scaling the solution is easier than with existing solutions. Ultimately it is dependent on the type of mesh networking that is used.

Chapter 4
Use Case Application

This chapter looks at how the different solutions fare when applied to the three predefined use cases (see section 1.3.2). Each solution is theoretically applied to each use case and their respective possibilites and outcomes are discussed.

4.1 GATT REST API

The GATT REST API is focused on external queries to communicate with BLE devices in the BLE network. A REST interface is used to access data and trigger events in the devices within the network. This REST interface can for example be a mobile application controlled by a user.

4.1.1 Use Case 1: Single Sensor Push

The Single Sensor Push use case relies on the peripheral sensor to be responsible for pushing data to the client via the gateway. This use case is difficult to achieve using the GATT REST API due to the fact that the API does not support sending HTTP messages from the gateway to the client if the client has not inititially sent a request. The peripheral can only push data to the gateway via the Advertise and Notify functions, where the data is then cached in the gateway. For the client to recieve the data pushed to the gateway by the peripheral, it needs to send a request to the gateway to recieve the cached value. This mode of operation is very ineffective for this use case, the API simply does not support pushing the data all the way from the peripheral device to the client without action from the client.

4.1.2 Use Case 2: Single Sensor Pull

The Single Sensor Push use case relies on the client to be responsible for requesting data from the peripheral device via the gateway. The GATT REST API is designed with the intention of accessing data in a peripheral device via a gateway which is exactly what this use case describes. The API allows for requesting data from a peripheral device connected to the gateway through requests initiated by the client, with the client acting as master and the peripheral device acting as slave.

4.1.3 Use Case 3: Multiple Sensor Network

The Multiple Sensor Network use case relies on multiple peripheral devices and the client being able to send and recieve data to each other via the gateway. As stated earlier the GATT REST API does not have support for the peripheral device sending data to the client without a request for the data from the client. This allows for a limited application of the use case, where the client is not explicitly informed of data changes from peripheral devices and has to query changes itself. For many applications of the use case this is not optimal if for example the funtionality of certain peripheral devices are dependent on values from other devices. The client must regularly send queries for the necessary data from one device in order to push changes to a peripheral device dependent on this data, resulting in an inefficient and inaccurate solution.

4.1.4 Conclusion

The GATT REST API is optimized for applications where peripheral devices need to be controlled externally, whether to query them for data, or to trigger certain functionality. A system where non-critical data values are monitored such as keeping track of warehouse inventory is a relevant application. Because the solution focuses on client initiated communication, it does not serve as an ideal solution in use cases dependent on change detection in peripheral devices.

4.2 6LoWPAN

6LoWPAN offers the capability of both peripheral devices in the BLE network, and external actors outside the network (e.g. via the cloud) to initiate and control the flow of data for devices in the network. This broad communication approach offers a wide array of possible areas of use.

4.2.1 Use Case 1: Single Sensor Push

6LoWPAN is a viable solution for the Single Sensor Push use case since it allows for peripheral device to push data via a gateway to the cloud. 6LoWPAN is however designed for large networks with multiple devices and features a somewhat extensive implementation process and from a cost-benefit perspective although 6LoWPAN is a possible solution it may not be the best solution for this use case. 6LoWPAN allows for many features which could be seen as superfluous for such a simple use case. 6LoWPAN does however offer the security benefit of end-to-end encryption through IPv6 and could therefore be seen as a good option if security is of high priority.

4.2.2 Use Case 2: Single Sensor Pull

Applying 6LoWPAN to the Single Sensor Pull use case is in many was similiar to the previous use case, Single Sensor Push. 6LoWPAN allows for external cloud services to request data from a peripheral device via a gateway making it a feasible

solution, but implementing a 6LoWPAN solution for a use case with such limited features may not be the most effective way of action. The same security aspects apply to this use case as to the previous one, namely end-to-end encryption for data being sent in the network offering greater security than the other solutions addressed in the thesis. Hence 6LoWPAN is a favourable solution when security is of great concern, but for other situations the effort of implementing 6LoWPAN may not be worthwhile.

4.2.3 Use Case 3: Multiple Sensor Network

The Multiple Sensor Network use case is the most demanding of the use cases evaluated due to its need for both peripheral devices and cloud services ability to initiate communication and data transfers. 6LoWPAN allows for this form of communication, more specifically, 6LoWPAN was designed for this type of use case with multiple peripheral devices intercommunicating with each other and the cloud through a gateway. The IETF have released a set of examples of application spaces for 6LoWPAN which all adhere to the Multiple Sensor Network use case[25].

4.2.4 Conclusion

Because of 6LoWPAN's comprehensive functionality it serves as a valid solution to all the use cases considered in the thesis. The flexibility of 6LoWPAN allows it to thrive for most imaginable use cases as it allows system to be both client controlled and controlled by peripheral devices simultaneously. This makes for very reflexive application possibilites. 6LoWPAN is applicable to most imaginable use cases, as it is responsive both to external client requests and peripheral devices. Because of this flexibilty a 6LoWPAN solution is able to control complex systems where a device can react to the changes of other devices in the network.

4.3 HTTP Proxy Service

In the HTTP Proxy Service, focus is on the peripheral devices in the BLE network. It is the peripheral device that controls the flow of information, sending their data through a HPS implementing gateway which translates the data to valid HTTP messages and dispatches them to the specified recipient.

4.3.1 Use Case 1: Single Sensor Push

The Single Sensor Push use case is based on the peripheral device initiating data transmission which is exactly what the HPS solution is intended for. The HPS solution allows the peripheral device to send and request data to and from a Web service, making it a viable solution for this use case. Having the peripheral control all communication allows for the solution to accurately portray the values presented by the peripheral device. Due to the fact that the HPS also supports using the HTTP GET request, it is possible to read values from the HTTP server through the gateway, allowing for the peripheral device to respond to changes in

the HTTP server thereby enabling the peripheral to read values pushed by both the peripheral itself and other sources.

4.3.2 Use Case 2: Single Sensor Pull

The Single Sensor Pull uses case relies on the user to request data from the peripheral device when it is desired. This goes against the functionality of the HPS solution which only allows for the peripheral device to control the flow of information. While it is possible to achieve a working solution using the HPS, where the user simply retrieves the latest data pushed by the sensor, the solution will in many cases not give accurate measurements. To achieve accurate measurements the peripheral must push its data at frequent intervals which leads to greater power consumption and in many cases superfluous HTTP requests. Considering that the type of devices and networks involved greatly value energy efficiency and minimal data transfers the HPS is a possible, but ineffective solution to this use case.

4.3.3 Use Case 3: Multiple Sensor Network

The Multiple Sensor Network incorporates elements of both the previous use cases and as stated above the HPS solution is applicable to both of these use cases, albeit with varying degrees of efficiency. For pushing data from the peripheral device to an Internet service the HPS solution is ideal, however regarding polling data from the peripheral device through an Internet service, the solution relies on simply getting the latest pushed value from the peripheral device. This may work for systems not requiring great precision in the values retrieved from the peripheral device but for most applications this method of function leaves great room for error.

A major flaw in the HPS solution is the fact that only one HTTP request can be handled by the gateway at a time, which leaves great risk for traffic congestion and delayed messages. A multi-sensor network implementing the HPS solution has however one clear benefit, namely having the gateways be fully generic, allowing for any HPS-adapted peripheral to connect to arbitrary HPS gateway. This allows for peripheral devices to be paired to several gateways and transmit identical HTTP requests irregardless of which gateway they connect to.

4.3.4 Conclusion

The HTTP Proxy Service is in some ways the opposite to the GATT REST API, at least in terms of where in the network the control of communication lies. The HPS allows for the peripheral devices to control the flow of information. This limits its areas of use but still provides a solid set of possible applications. A system which requires a client to be responsive to changes in peripheral devices is a good example of a relevant application for the HTTP Proxy Service. Reversely, if a client were to require being able to communicate with peripheral devices on demand the HPS solution would be sub-par.

4.3.5 Comparison

The three solutions are shown to each fulfill the requirements for at least one use case, with 6LoWPAN proving to be applicable to all three use cases. Below shows an overall comparison of the different solutions applied to the three use cases.

Solution	Single Sensor Push	Single Sensor Pull	Multiple Sensor Network
GATT REST API	Possible through Advertise and Notify functions only	Fully Supported	Not Supported
6LoWPAN	Fully Supported	Fully Supported	Fully Supported
HTTP Proxy Service	Fully Supported	Possible through retrieving latest pushed value	Not Supported

Table 4.1: Use Case Comparison of Solutions

Chapter 5
Implementation

This chapter states the goals of the implementation as well as the reasoning behind electing a solution to implement. The hardware, as well as software, of the implementation is presented and the implementation process is detailed. Challenges and issues encountered during the implementation are also discussed.

5.1 Implementation Goals

When looking at the different implementation possibilites for the different solutions a few main concerns were central. One of the central themes was that the implementation should have some value to **u-blox**, preferably implementing the solution in one or more of their devices. Being able to fully utilize the resources and support available at **u-blox** was another motivator for using **u-blox** devices. Another important factor was how the implemented solution would be deemed to fare in the future, as implementing a technology or solution that was likely soon be considered obsolete or undesirable is not optimal in a domain as new and rapidly evolving as the IoT domain.

Optimally more than one solution would be implemented to give a better foundation for a more hands-on comparison between the different solutions. The required effort of implementing these solutions within the given timeframe did not make this a viable option.

5.2 Choice of Solution

All three solutions were considered for implementation, with 6LoWPAN and the HTTP Proxy Service deemed as generelly more favorable; 6LoWPAN for its wide areas of application and use of the TCP/IP protocol, the HTTP Proxy Service for its simplicity and being based on the same architecture as other BLE services. The GATT REST API was not chosen as it was not considered a solution that would see widespread adoption, in part due to the coming release of Bluetooth Mesh, and also due to 6LoWPAN offering the same functionality but with a wider set of features such as IPv6 addressing and improved security.

The HTTP Proxy Service was finally chosen as the solution to implement, one

of the main reason being its almost ingenious simplicity in using existing BLE functionality and a simple HTTP client to acheive an IoT BLE sensor network solution.

5.3 Gateway Device

The gateway device that was used to implement the HPS solution is a **u-blox** ODIN-W2 IoT gateway module. ODIN-W2 is a stand-alone multiradio module which supports concurrent dual-band wifi and dual-mode Bluetooth connections.

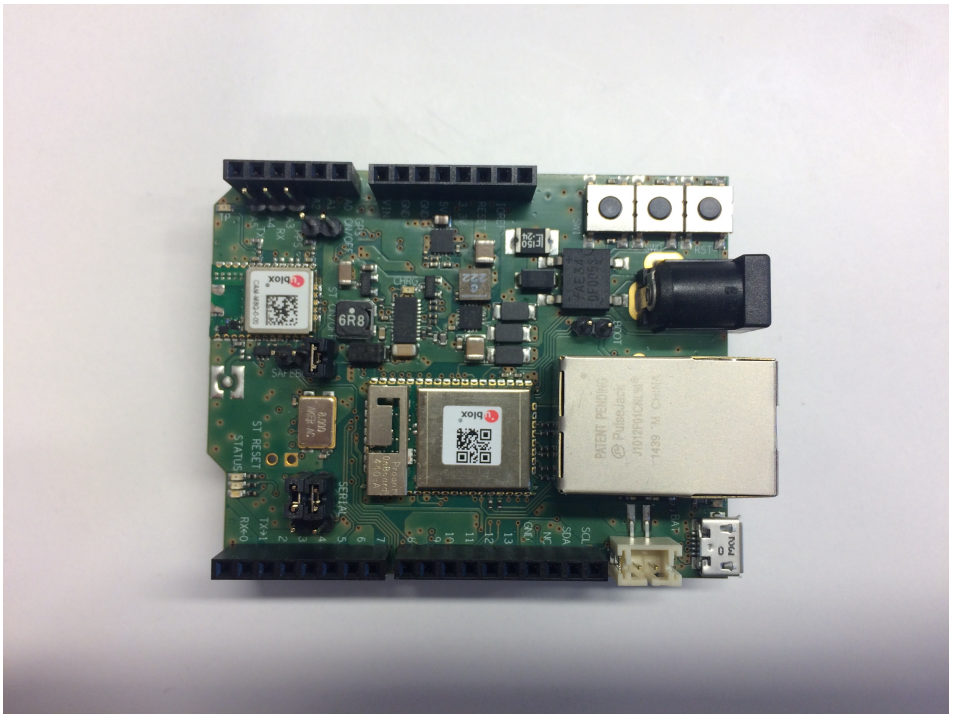


Fig. 5.1: u-blox ODIN-W2 gateway module

5.4 Peripheral Device

Originally a **u-blox** NINA-B1 module was intended to be used as the peripheral device used to send data to the gateway. The purpose of the periphal device was to have a HPS adapted GATT service in the that would send the device data correctly formatted for transmitting as a HTTP request. This idea was however later discarded in order to emphasize the development of the actual gateway module and instead a ConnectBlue OBS421 module was used to emulate the functionality of the NINA-B1 module. The OBS421 is a Bluetooth Serial Port Adapter allowing for sending Bluetooth GATT commands programmatically from a terminal

window.

Fig. 5.2: ConnectBlue OBS421 module

5.5 Software & Tools

The solution was written using the C programming language and achieved through the use of the following implementation tools:

ARM mbed
: ARM mbed is a platform aimed at the development of IoT system applications. The ARM mbed platform provides an operating system as well as cloud services and developer tools facilitating the creation and deployment of IoT solutions. The HPS solution was developed using the tools and operating system provided by the ARM mbed platform.

Yotta
: Yotta is a build tool associated with the ARM mbed platform. Yotta allows for building embedded software solutions from seperate software modules for use on the ARM mbed platform. By using Yotta the solution was possible to be built as a binary with the necessary software modules provided by **u-blox** together with the code defining the HPS solution.

lwIP
: lwIP is a lightweight TCP/IP implementation used mainly for embedded

systems applications with a focus to minimize resource usage while allowing for full TCP/IP functionality. lwIP was used to create and manage the HTTP client needed to send and recieve HTTP messages to and from the gateway device as well as manage IP configuration.

BLE stack
The BLE stack contained in the module was used to create the HPS service which handles the dataflow over Bluetooth for the solution. The BLE stack enabled the creation of a GATT service with the necessary characteristics and functionality required for the HPS solution as defined by the Bluetooth SIG.

Wifi driver
The Wifi driver controls the Wifi part of the gateway module and was used to initiate and manage the wifi connection of the gateway device allowing the device to connect to the Internet.

5.6 Implementation Process

The HPS solution was developed in three steps. Primarily a GATT service was created in the gateway, adhering to the specifications for a HTTP Proxy Service as defined by the Bluetooth SIG[20], then an Internet connection was set up for the gateway to enable communication outside of the BLE sensor network. The final step was implementing a HTTP client in the gateway and setting up reading and writing of data between the HTTP client and the GATT service. This allowed for the gateway device to receive GATT commands from peripheral devices, translate data received through the GATT commands to a syntactically correct HTTP message and send the HTTP message to the destined recipient.

5.6.1 The GATT Service

A skeleton of a GATT service was provided by **u-blox** for the ODIN-W2 module which was used as the code base for the solution. This skeleton allowed for the specification of necessary GATT commands and characteristics as well as their properties. The GATT service was designed according to the white paper specifications from Bluetooth SIG, which describes the necessary characteristics and their properties for the service(see section 2.4). The data fields required for sending a correct HTTP message are: destination URI, Headers, Body, and Control Point which initiates the HTTP request and specifies the type of request. These fields are each represented by their own characteristic in the service to which peripheral devices can write. The URI, Headers, and Body characteristics have a maximum size of 512 byte in order to contain the longer string values they represent, while the Control Point characteristic is simply a 8 bit integer to represent the different HTTP request types.

Writing to Characteristics

GATT commands can only send data packets of maximum 22 bytes, which means that when writing to the URI, Headers, and Body characteristics regular write commands will often not suffice. Therefore a Long Write command had to be implemented which called upon the write command several times, while keeping track of the offset for each write so that all data was transmitted. To achieve this each write command was written to a temporary field until there was no data left to write, and the written data was then transfered to the correct characteristic field. A state attribute was used to control the writing of data to the correct characteristic, with each Long Write command triggering a change in state.

Constructing HTTP requests

A control of the write order was also implemented in order to be able to correctly construct HTTP messages and initate the sending of the HTTP message. Writing to the Control Point characteristic triggers the formatting of the HTTP message from the characteristic fields. After the message has been formatted the HTTP client is called to initiate the sending of the HTTP request with the formatted message sent as a parameter.

5.6.2 Wifi Connection and the HTTP Client

With the HPS implemented, the remaining stages were setting up a Wifi connection on the gateway device, and implementing a HTTP client to dispatch the HTTP messages constructed by the service.

Wifi Connection

A code skeleton for configuring and calling the wifi driver was also provided by **u-blox** in order to facilitate the connection process for the gateway. The specifications for the wifi network to connect to are declared in a separate configuration file. The wifi driver is automatically started when the gateway is booted and will attempt to connect to the wifi network specified in the configuration file.

The HTTP Client

The final stage of the implementation was constructing a HTTP client responsible for managing the actual communication with the cloud. The first step of the HTTP client was to implement Domain Name System(DNS) lookup for for the destination address. DNS is an integral part of the Internet infrastructure and is used to locate the specific IP address from a given Uniform Resource Locator(URL). The IP address received through the DNS lookup is then connected to using the Transmission Control Protocol(TCP) and the HTTP message is sent and the response HTTP message is processed.

5.6.3 Sending information from a peripheral device to the cloud

When the HPS, wifi connection and HTTP client are all implemented and configured the gateway is open to send data from peripheral devices connected over BLE to the gateway. In this implementation a peripheral device was emulated using GATT commands. To send a HTTP message, the peripheral device writes to the URI, Header and Body characteristics in the HPS gateway. Finally the peripheral device writes to the Control Point characteristic denoting what type of request the HTTP message should be. Writing to the Control Point characteristic also triggers the formatting of the HTTP message, and passes the HTTP message on to the HTTP client which in turn sets up a connection to the URI contained in the URI characteristic in the HPS, and consequently dispatches the HTTP message.

5.7 Development Problems and Implementation Issues

The development of the HPS gateway was not without its challenges. Primarily attaining the write funtionality for characteristiccs larger than 22 bytes(i.e. the Headers, Body and URI characteristics) was quite arduous. As it turned out there was some functionality missing in the Bluetooth stack for the Odin W2 module which needed to be added by **u-blox** in order to achieve the Long Write functionality. There were also some issues regarding Wifi connectivity for the Odin W2 module. The module seemed to have trouble maintaining a simultaneous Bluetooth and Wifi connection, where the HPS gateway would reboot when an attempt to connect to a peripheral BLE device was made after a Wifi connection had been established. The issue seemed to vary in frequency depending on which Wifi router the module connected to. Together with **u-blox** the issue was debugged and found to be triggered in the drivers of the gateway. The issue was therefore deemed outside the scope of the implementation for rectifying.

5.8 The HPS solution in practice

The HPS solution is built on the peripheral devices controlling the flow of information, with the gateway device acting as a proxy. As stated in earlier (see section 3.3) the HPS solution is best suited for use cases oriented towards peripheral devices pushing data to an external service. Applying the HPS solution to scenarios where data needs to be distributed from an external service to peripheral devices or where control of the flow of information needs to be outside the sensor network is inexpedient. The HPS solution works best for systems requiring monitoring of data within a sensor network, rather than actively controlling the sensors and network data externally.

Chapter 6
Discussion and Conclusions

The thesis takes a quite superficial approach to the different Bluetooth-oriented IoT solutions encompassed within the thesis. The intention is not to give a fullbodied implementation guide but rather to give an idea of what solutions are available when implementing a Bluetooth-based IoT sensor network and in what different situations which solutions are beneficial and in which they are not.

6.1 Solutions

The three solutions addressed in this thesis all serve to solve the problem of granting Internet access to low energy Bluetooth devices with the help of a gateway device, but operate using different methods and specifications. Because of the differences between them they prove to excel under different situations and circumstances.

6.1.1 GATT REST API

The GATT REST API solution operates through cloud services requesting data from low energy devices via the gateway, limiting its applicable use areas. The solution is optimal for situations where functionality is not dependent on changes in the values of the sensor devices, but rather focus on retrieving data from the sensor devices on demand through cloud services. Because of the solution's specific functionality it should not be considered a standard solution for Bluetooth IoT networks, but rather as a solution oriented to a specific set of use cases which focus on retrieving data from the sensor network.

A benefit of using the GATT REST API is that it is built upon the RESTful architectural style (see section 2.2.1) which is supported by many current frameworks and systems. Integrating a GATT REST service with existing software as well as new software does therefore not require significant effort as it uses a well established Web service architecture. This can be useful for situations when for example an IoT sensor network needs to be added to an already existing software solution. An example of this would be a sensor network that is able to monitor inventory stock in a warehouse that needs to be integrated into the Web page for a vendor. The Web page would then simply have to call the RESTful service

for the network to get a current update on inventory status. A situation where the GATT REST API would be an inadequate solution is where events or actions would need to be triggered when certain events occur or when a certain value is reached. Because the GATT REST API revolves around actors outside the sensor network to intiate communication, the response accuracy would be subpar.

When it comes to factors such as security and scaling these attributes are more dependent on the BLE architecture than the GATT REST API. Therefore with issues such as security in mind, if the standard BLE security is deemed unsatisfactory one should consider either implementing a different network architecture or implementing a different BLE IoT solution which offers added security features such as 6LoWPAN.

6.1.2 6LoWPAN

6LoWPAN is both the most powerful and most advanced of the solutions studied. It is built upon a set of standards maintained by the IETF and envelopes multiple layers of the sensor network. Due to this multi-layer functionality it involves more advanced configurations than the HPS or GATT REST API solutions, which only operate on the uppermost layers of the BLE architecture. 6LoWPAN is able to accomodate for both sensor devices and external services and actors to initiate and control the flow of communication. This allows for the most expansive and versatile set of applicable use cases. 6LoWPAN's ability to distribute each network node with its own IP address allows for end-to-end encryption of IPv6 packets, which increases the overall security of the network as well as making it unproblematic to conjoin the sensor network with other IP implementing networks.

As a result of the more complex underlying configurations of 6LoWPAN, when implementing a BLE IoT solution, it is worth examining whether the necessary functionality can be achieved using simpler solutions such as the GATT REST API or the HPS solution. As these solutions only require configuration of the uppermost network layers, the requirements and implementation efforts are less significant.

The complexity of 6LoWPAN and its expansive functionality allows for a wide myriad of application areas. Advanced sensor networks requiring intercommunication between sensor nodes in the network are great examples of use cases where 6LoWPAN excels. Due to 6LoWPAN's support for meshing as well as its flexible communication inititation it is able to accomodate for such advanced networks whose functionality is dependent on sensor devices intercommunicating with each other. For example, a sensor network controls a set of pressure valves where one type of sensor is responsible for opening and closing the valves while another type of sensor reads the current pressure. A 6LoWPAN network would be able to allow for the valve sensors to be aware of the pressure values from the pressure sensors, and open and close the valves accordingly. Both types of sensors would be able to send their current values to a cloud service that allows monitoring of the current pressure and the state of the valves, as well as allowing for the ability to manually

change the state of the valves.

While it can seem like 6LoWPAN is the most optimal solution as it enables by far the most functionality as well as increased security features and added intercommunication capabilites in the form of meshing, it is worth considering that many aspects need to be configured on multiple layers of the network in order to implement 6LoWPAN. 6LoWPANs are not necessarily interchangable either, for example if a 6LoWPAN gateway is configured to use UDP in the transport layer, how will it react if a 6LoWPAN node configured for using TCP is added to the network?

6.1.3 HTTP Proxy Service

The HTTP Proxy Service, like the GATT REST API, is oriented towards a specific method of operation, which one could say is the opposite of the GATT REST API. The HTTP Proxy Service is based on the sensor devices controlling the flow of communication in the sensor network. Also, as with the GATT REST API, this limits the application of the solution to a set of specific use cases. Here however, the use cases are based on one or more sensors controlling the network, specifying when to send data to the cloud and what to send. The HPS has support for the HTTP command GET which allows it to retrieve data from external services and actors, which it turn increases its areas of application. It does not however, change the fact that it is still only the sensor devices that can initiate communication.

The HPS solution is implemented just as one would implement any other BLE functionality, using GATT Services (see section 2.1.5). This allows for sensor devices to not require any added attributes or extended functionality as all the required implementation needs are contained within BLE. The gateway device simply needs support for implementing a HTTP client apart from the standard BLE functionality. These minimal requirements make the HPS a versatile and relatively simple solution. Adding to its versatility is the fact that the HPS gateway is generic, meaning sensor devices adapted for the HPS solution can connect to any HPS implementing gateway device without any additional configuration required. For simpler use cases fitting into the mode of operation of the HPS or situations where a solution is needed to be implemented relatively prompt, the HPS is ideal. Use cases where the sensor network is used exclusively for monitoring are a good example of an ideal application area for the HPS solution. The sensors can push data at set intervals or when certain events are triggered which can then be observed by external services and actors. A temperature sensor could for example push the temperature value when it passes a certain limit to a mobile application which in turn triggers a notification informing users of the temperature breaking the limit.

The simplicity of the HPS can in many cases be a draw back as it does not leave much room for expanding the functionality of a sensor network. Because the HPS solution is comparably simple to implement, it can serve as a first draft for a IoT solution, when a solution needs to be set up and running quickly, giving a tem-

porary solution while a more complex and powerful solution such as 6LoWPAN is developed.

6.1.4 Bluetooth Mesh and the Future of BLE Enabled IoT

With the approaching release of Bluetooth Mesh, the future of the evaluated solutions is uncertain. The solutions in their current form offer their respective benefits and drawbacks. Many of the shortcomings of the GATT REST API and HTTP Proxy Service, such as unilateral communication and lack of end-to-end encryption, are adressed with Bluetooth Mesh. It remains to be seen how well the Bluetooth Mesh standard performs and how much of an implementation effort will be involved for adopting a Bluetooth Mesh solution. One can however assume that if Bluetooth Mesh delivers on what is being promised, the GATT REST API, and maybe even also the HTTP Proxy Service will see themselves replaced in favor of Bluetooth Mesh. The HTTP Proxy Service may still be preferable in certain situations considering its relative simplicity.

Bluetooth Mesh also challenges 6LoWPAN as it offers the same breadth of functionality as 6LoWPAN but with improved networking capabilities. Bluetooth Mesh also has the benefit of being maintained by the Bluetooth SIG and as such does not require implementing separate technology in order to apply the solution, which is the case with 6LoWPAN. Bluetooth Mesh is expected to work on current BLE enabled devices which greatly facilitates its implementation.

Considering Bluetooth Mesh is designed with the intention of being an all round BLE IoT solution it shows great promise for the BLE IoT domain as it attempts to solve many of the issues with BLE IoT seen today. Time will tell if Bluetooth Mesh manages to become the standard IoT solution for BLE enabled networks.

6.2 Implementation

The implementation, although successful, is far from complete. The functionality of the implementation in its current state is very limited and could not be applied to a production environment. While the implementation allows for sensor devices to communicate over HTTP via the gateway many necessary aspects of the HPS solution are yet to be implemented such as canceling ongoing HTTP requests, processing of status-codes in the HPS server, and the HPS security characteristic allowing for HTTPS messages to be sent.

Issues with the wifi connectivity also limit overall functionality making the implementation in its current state unreliable. The issue was traced to the wifi driver of the gateway module and resolving the issue most likely lies outside of the HPS implementation.

The implementation however shows that it is relatively elementary to implement the HPS solution in a gateway, as it is implemented as one would implement any

other BLE service, for which there is a plethory of information at hand. Implementing the HPS solution in peripheral devices however proved to be a bit more complicated than originally thought. The fact that the peripheral device intended to be used for the solution required a different set of tools than the gateway device complicated the implementation as implementing the peripheral device required learning a completely new set of tools and frameworks. Luckily the module used for testing the HPS gateway could also be used to emulate a peripheral device.

6.3 Final Thoughts

The thesis study evaluates a set of possible IoT solutions for Bluetooth-based sensor networks and shows that the different solutions are optimal for different use cases and under different circumstances. This is in itself not a suprising result, it is natural that solutions built on different architectures and standards are applied with varying degrees of effectiveness to different use cases. An interesting conclusion however is that each of the different solutions match one of the three use cases applied in the thesis. One could argue that the use cases were chosen to match one of the evaluated solutions, this is however not the case as the use cases were chosen early in the process before the solutions had been thoroughly researched.

The thesis only touches the surface of the specifics for each of the solutions and many aspects are not addressed. Focus was consciously put on how the solutions solved the issue of granting Internet access to BLE sensors, and how data was transmitted within the network. Important aspects not taken into consideration but which nonetheless are of great significance are range, power consumption and future developments. A more in depth evaluation of security would also be of great benefit as the security aspects considered in the thesis are very superficial.

6.4 Future Work

IoT is a field of technology which is evolving at a very fast pace and therefore it is important but also demanding to be up to date. Following the development of the paradigm and its associated and available solutions is key to providing a relevant account of available options for enabling IoT through the use of BLE devices.

6.4.1 Solutions

With the release of Bluetooth Mesh it would be interesting to perform a proper comparison of Bluetooth Mesh with the currently evaluated solutions to see how they fare against this new architecture. Could Bluetooth Mesh make the other solutions obsolete?

Delving deeper into the different aspects of the solutions as well as aspects not yet addressed such as range and power consumption would further improve the evaluation of the solutions and give better indications for under which circumstances the

different solutions are optimal such as in situations with very constrained power consumption requirements, or where range and reliability are key.

6.4.2 Implementation

The obvious next step in the implementation would be adding a HPS enabled peripheral device to the implementation, as was originally intended. To make the implementation more interesting something more creative could be done with the actual HTTP messages sent through the gateway, such as posting a Twitter update. If the gateway is ever to be used in an environment outside of academics the rest of the HPS attributes and functionality would need to be implemented such as handling of error codes, canceling ongoing HTTP communication and implementing support for HTTPS messages.

References

[1] Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020, http://www.gartner.com/newsroom/id/2636073, 2013 (visited on 10/12/16)

[2] Mike Kavis, Don't Underestimate The Impact of the Internet of Things, http://www.forbes.com/sites/mikekavis/2014/07/21/dont-underestimate-the-impact-of-the-internet-of-things/2/#5aa9b79a587d, 2014 (visited on 10/12/16)

[3] Cho, Keuchul and Park, Woojin and Hong, Moonki and Park, Gisu and Cho, Wooseong and Seo, Jihoon and Han, Kijun, Analysis of Latency Performance of Bluetooth Low Energy (BLE) Networks, Sensors, 15(1):59, 2015

[4] Gomez, Carles and Oller, Joaquim and Paradells, Josep, Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology, Sensors, 15(9):11734, 2012

[5] Dr. Cuno Pfister, Bluetooth Low Energy for the last 30 Meters of the Internet of Things, Wireless Congress, 2014

[6] Aiman J. Albarakati, Junaid Qayyum, Dr. Khalid A. Fakeeh, A Survey on 6LowPAN & its Future Research Challenges, International Journal of Computer Science and Mobile Computing, Volume 3, 2015

[7] Ida Sabel, Internet of Things & Communication Protocol, Jönköping University, School of Engineering, 2015

[8] Robin Heydon, Bluetooth Low Energy Developer's Handbook, Prentice Hall, 2012

[9] Jeremy Clark, Paul C. van Oorschot, SoK: SSL and HTTPS:Revisiting past challenges and evaluating certificate trust model enhancements, IEEE Symposium on Security and Privacy, 2013

[10] Mike Ryan, Bluetooth: With Low Energy comes Low Security, USENIX Workshop on Offensive Technologies, 2013

[11] Spring, Spring for Android, http://projects.spring.io/spring-android/, 2014 (visited on 10/12/16)

[12] Spring, Consuming a RESTful Web Service from iOS, https://spring.io/guides/gs/consuming-rest-ios/, 2014 (visited on 10/12/16)

[13] Mike Wasson, Calling a Web API From a .NET Client in ASP.NET Web API 2 (C#), http://www.asp.net/web-api/overview/advanced/calling-a-web-api-from-a-net-client, 2014 (visited on 10/12/16)

[14] Dr .M .Elkstein, Using REST in Python, http://rest.elkstein.org/2008/02/using-rest-in-python.html, 2008 (visited on 10/12/16)

[15] Dr. M. Elkstein, Using REST in Javascript, http://rest.elkstein.org/2008/02/using-rest-in-javascript.html, 2008 (visited on 10/12/16)

[16] J. Nieminen T. Savolainen M. Isomaki B. Patil Z. Shelby & C. Gomez, IPv6 over Bluetooth Low Energy, Internet Engineering Task Force(IETF), 2015

[17] C. Gomez, S. Darroudi, T. Savolainen, IPv6 over Bluetooth Low Energy Mesh Networks, Internet Engineering Task Force(IETF), 2015

[18] Bluetooth SIG, Bluetooth Mesh White Paper, Bluetooth SIG 2016

[19] Bluetooth Special Interest Group, Gateway Smart Starter Kit, https://www.bluetooth.com/develop-with-bluetooth/developer-resources-tools/gateway, 2015 (visited on 10/12/16)

[20] Bluetooth Special Interest Group, HTTP Proxy Service White Paper, https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=308344&_ga=1.19823679.661477216.1468331256, 2015 (visited on 10/12/16)

[21] LinkLabs, 3 Reasons Why IPv6 Is Important for the Internet of Things, http://www.link-labs.com/why-ipv6-is-important-for-internet-of-things/, 2015 (visited on 10/12/16)

[22] N. Kushalnagar, G. Montenegro, C. Schumacher, IPv6 over Low-Power Wireless Personal Area Networks, Internet Engineering Task Force(IETF), 2007

[23] Jonas Olsson, 6LoWPAN Demystified, Texas Instruments, http://www.ti.com.cn/cn/lit/wp/swry013/swry013.pdf, 2014 (visited on 10/12/16)

[24] Bluetooth SIG, Bluetooth technology to gain longer range faster speed mesh networking in 2016, https://www.bluetooth.com/news/pressreleases/2015/11/11/bluetooth-technology-to-gain-longer-range-faster-speed-mesh-networking-in-2016, 2015 (visited on 10/12/16)

[25] E. Kim D. Kaspar N. Chevrollier JP. Vasseur, Design and Application Spaces for 6LoWPANs, Internet Engineering Task Force(IETF), 2011

[26] Rick Walker, Smart Mesh for the Smart Home, http://cwbackoffice.co.uk/docs/Rick%20Walker.pdf, 2015 (visited on 10/12/16)