Bluetooth Low Energy Based CoAP Communication in IoT

----- CoAPNonIP: An Architecture Grants CoAP in Wireless Personal Area Network

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

by

NAN CHEN

PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis. Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldsen Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

ABSTRACT

In recent years, the development of smart devices has brought the Internet of things (IoT) become a popular topic. In IoT, the Constrained Application Protocol (CoAP) is a well-known protocol used in constrained nodes and constrained networks. CoAP aims to work in an IP-based network where a socket is available. However, there are many constrained devices using different scenario to transfer data. For example, in Bluetooth Low Energy (BLE) device use MAC address as an identifier and use Generic Attribute Profile (GATT) to transfer data. Therefore, how to overcome those barriers of communication technologies to support CoAP is a meaningful research field.

There are several approaches to overcome those barriers. For example, a new hardware component can be added to make those device support TCP/IP protocol stacks. Then those devices can easily implement CoAP. On the other hand, an application layer architecture can be added upon existing communication technologies to support CoAP. Considering to minimize the changes of underlying communication infrastructure, the second approach can achieve the goal with less cost.

The objective of this research is to propose an architecture that grants CoAP in different Non-IP based communication technologies. Meanwhile, the research will take Bluetooth Low Energy as an example to explore how to overcome limitations of underline communication technology.

ACKNOWLEDGEMENTS

I would like to express my very great appreciation to my supervisor Professor Ralph Deters. Under his supervision, I have gained great skills in the past two years. The past two years was a wonderful and memorable journey of my life. Under Professor Ralph Deters's guide, I not only achieved marks in academic but also increase skills in programming.

I would also like to extend my thanks to all colleagues in MADMAC lab. Their support helped me to work through the tough time at the first year and granted me confidence to overcome problems.

At last, I want to thank the support of my parents and family members. Without their support, I can not study and work smoothly in the past two years.

TABLE OF CONTENTS

PERMISSION TO USE i
ABSTRACT ii
ACKNOWLEDGEMENTS iii
TABLE OF CONTENTS iv
LIST OF FIGURES vii
LIST OF TABLES viii
LIST OF ABBREVIATION ix
CHAPTER 1 INTRODUCTION 1
CHAPTER 2 PROBLEM DEFINITION 2
2.1 How to unify communication language between Non-IP and IP based technologies? 2
2.2 How to overcome limitations of standard BLE communication? 2
2.2.1 Size limitation of a BLE request 3
2.2.2 Chaos server-side callback 3
2.3 How to serve multiple applications as a background service? 3
2.4 How to provide common interface to support different technologies? 3
2.5 Research Goal 4
CHAPTER 3 LITERATURE REVIEW 5
3.1 IoT 5
3.2 Personal Cloud 7
3.3 SOAP and REST 8
3.3.1 SOAP 8
3.3.2 REST 9
3.3.3 REST VS SOAP 11
3.3.4 REST VS SOAP in mobile app 12
3.3.5 Conclusion 12
3.4 MQTT and CoAP 13
3.4.1 MQTT 13
3.4.2 CoAP 14

3.4.3 COAP VS MQTT 18
3.4.4 Conclusion 18
3.5 Bluetooth 19
3.5.1 Classic Bluetooth 20
3.5.2 Bluetooth Low Energy 21
3.5.3 Classic Bluetooth vs Bluetooth Low Energy 25
3.5.4 Conclusion 25
3.6 CAP Theorem 26
CHAPTER 4 DESIGN AND ARCHITECTURE 28
4.1 Application layer 29
4.1.1 Process component 29
4.1.2 Communication component 31
4.1.3 Conclusion 32
4.2 Network layer 32
4.3 Communication Mechanism 33
4.4 Packet Format 35
4.5 Virtual Resource 36
4.6 Detail Design 37
4.6.1 Application Layer 37
4.6.2 Network Layer 39
4.7 Solutions for proposed problems 44
4.7.2 Size limitation of a BLE request 44
4.7.3 Chaos server-side callback 44
4.7.4 Serve multiple applications 45
4.7.5 Provide common interface to support wireless communication protocols 45
CHAPTER 5 IMPLEMENTATION 46
CHAPTER 6 EXPERIMENT DESIGN 48
6.1 Goals of Experiment 48
6.2 Experiment Setup 49
6.3 Details 49
6.3.1 Minimum Data with Interval 49
6.3.2 Multiple Packets 53

6.3.3 Round Trip 55
6.3.4 Multiple Apps 57
6.4 Conclusion 58
CHAPTER 7 SUMMARY AND CONTRIBUTION 60
CHAPTER 8 FUTURE WORKS 62
8.1 Data Rate 62
8.2 Availability and Partition Tolerance 62
8.3 Cross-Platform and Underlying Technology 62
8.4 Security 62
CHAPTER 9 REFERENCE 64

LIST OF FIGURES

Figure 3.1 Trend of devices vs people (Barrett, 2012) 5
Figure 3.2 SOAP message structure example (Mitra & Lafon, 2007) 9
Figure 3.3 MQTT format 13
Figure 3.4 CoAP format (Shelby, Z., Hartke, K., & Bormann, C., 2014) 15
Figure 3.5 From Web Applications to IoT Nodes (Shelby, 2014) 17
Figure 3.6 Topology of Classic Bluetooth (SIG, 2010) 20
Figure 3.7 Topology of Classic Bluetooth (Torvmark, 2013) 21
Figure 3.8 GATT format (SIG, 2010) 22
Figure 3.9 GATT format (SIG, 2010) 23
Figure 4.1 Layers of proposed architecture 28
Figure 4.2 Application layer structure 29
Figure 4.3 Process component flowchart 30
Figure 4.4 Communication mechanism 34
Figure 4.5 Packet Structure 36
Figure 4.6 Virtual Resource Mechanism 37
Figure 4.7 Life cycle of network service 42
Figure 4.8 Life cycle of CoAP communication in proposed architecture 43
Figure 5.1 Implementation screenshot 46
Figure 6.1 Results of sending headers with different interval 50
Figure 6.2 Result of sending headers with no interval 51
Figure 6.3 Result of sending headers with 150m interval 52
Figure 6.4 Result of sending headers with 200m interval 52
Figure 6.5 Results of sending multiple packets 54
Figure 6.6 Result of sending 12-byte payload (2 packets) 54
Figure 6.7 Result of sending 28-byte payload (3 packets) 55
Figure 6.8 First set of 4-bytes round trip 56
Figure 6.9 Second set of 4-bytes round trip 56
Figure 6.10 Third set of 4-bytes round trip 57
Figure 6.11 Result of Multi-App Experiment 58

LIST OF TABLES

Table 6.1 Experiment Goals 48
Table 6.2 Device specification.....49

LIST OF ABBREVIATION

IoT	Internet of Things
CoAP	Constrained Application Protocol
WPAN	Wireless Personal Area Network
MQTT	MQ Telemetry Transport
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
BLE	Bluetooth Low Energy
SIG	Special Interest Group
NFC	Near Field Communication
IPSP	Internet Protocol Support Profile
PA	Partition Tolerance and Availability
PC	Partition Tolerance and Consistency
IP	Internet Protocol
IPV6	Internet Protocol Version 6
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
MAC	Media Access Control
GATT	Generic Attribute Profile
MTU	Maximum Transmission Unit
L2CAP	Logical Link Control and Adaptation

CHAPTER 1 INTRODUCTION

The boom of the smartphone market and the development of wearable devices are introducing low energy sensors and personal hubs (phone, tablet or other portable devices) with increasing rate. In this trend, we find that the role of the smartphone has shifted from a single function device to an integrated personal data hub. On the other hand, with the development of tiny sensors, more and more small devices are installed to collect data as the new edge of Internet. Since more and more people have multiple smart devices to interact with each other, the communication between personal hubs and edge devices becomes increasingly important.

There are multiple technologies to support short range wireless data communication like BLE and NFC. However, different technologies have the different restrictions on data transfer. Although, technologies like MQTT and COAP are available to talk with different machines running in IP cluster, IP address is an essential requirement in the current implementation. Therefore, there is always a protocol transfer to consume resources through WPAN technologies like BLE and NFC. There is still a great room to improve in merge edge nodes into IoT. For example, Johanna proposed a hardware solution to install IPv6 in BLE devices by adopting 6lowpan (Isomaki, M., Nieminen, J., Gomez, C., Shelby, Z., Savolainen, T., & Patil, B., 2011) . However, there are no efforts being made on implementing a software layer upon existing architecture of BLE.

In this research, I will take BLE as an example to show how to merge lightweight Non-IP based wireless communication technologies into IoT context by using the proposed architecture. The remaining sections of the thesis are constructed as follows:

- Chapter 2: Discuss what problems need to be solved in the research.
- Chapter 3: Review related work.
- Chapter 4: Discuss proposed architecture in details.
- Chapter 5: Explain key implementations.

- Chapter 6: Discuss experiments for performance test.

CHAPTER 2 PROBLEM DEFINITION

As mentioned above, the research aim is to propose an architecture to accelerate the process of merging NonIP based technologies into IoT. To achieve this goal, we need to develop a lightweight protocol to carry information as well as a set of mechanisms to guarantee data consistency. Since one or more apps in one device need to communicate, the architecture also needs to run as a background service.

In short, we need to answer the following key questions:

1. How to unify communication language between Non-IP and IP-based technologies?

2. How to overcome limitations of standard BLE communication?

3. How to serve multiple applications as a background service?

4. How to provide a common interface to support different technologies?

2.1 How to unify communication language between Non-IP and IP based technologies?

The concept of IoT is firmly associated with IPv6. It tags things by assigning a unique address (usually IPv6) to each of them. However, in the real world, not all things tagged following the same rule (IPv6). For Example, BLE uses MAC address to identify devices. To unify communication language between Non-IP and IP-based devices, the following two things are necessary: 1. A software layer identifier. 2. A protocol which is independent of IP address and MAC address

2.2 How to overcome limitations of standard BLE communication?

Since BLE is designed for low bandwidth data transfer, some limitations need to overcome, in order to merging it with existing IP-based IoT networks.

2.2.1 Size limitation of a BLE request

In a standard BLE communication, GATT profile is a must, in which the payload of a characteristic is limited. According to Bluetooth4.0 specification (SIG, 2010). "All L2CAP implementations shall support a minimum MTU (maximum transmission unit) of 48 octets over the ACL-U logical link and 23 octets over the LE-U logical link". According to my experiment, the available payload size of a characteristic is 20 bytes. However, in IoT, one packet can easily extend 20 bytes. Therefore, the architecture should be able to cut long byte array messages into short sub packages and assemble them at remote side. We need a simple protocol, as well as relevant pack and unpack mechanism to solve this problem,

2.2.2 Chaos server-side callback

In BLE, a server may serve multiple clients. Unlike classic Bluetooth where each connected client can create its own socket to maintain communication with the server, BLE server handles all requests for a certain operation in one callback function. In brief, one callback function may be triggered by different clients. Therefore, the proposed architecture must provide a mechanism to handle messages from different clients in one function.

2.3 How to serve multiple applications as a background service?

The consumer of a specific request may come from different applications in one device, which means the communication between provider and consumer are not at device level but application level. Thus, the architecture must be able to identify a certain piece of message come from which application at which device.

2.4 How to provide common interface to support different technologies?

Beside BLE, other Non-IP based wireless communication technologies have the same requirement to take advantages CoAP. We can support those technologies by proposing an abstract interface.

2.5 Research Goal

The goal of the research is to propose architecture to accelerate the merging of WPAN technologies into IoT.

It is composed of the following three sub-goals.

Goal 1. Propose a solution for identifying Non-IP based devices.

Goal 2. Put forward a protocol and architecture to support CoAP communication in BLE.

Goal 3. Design the architecture to support multiple apps as a background service.

Goal 4. Provide abstract interface to support other WPAN technologies.

CHAPTER 3 LITERATURE REVIEW

3.1 IoT

In recent years a new concept named "IoT" is popular. IoT is a concept of making real world things become available through The Internet. It is happening and will fundamentally change our world. A background of this technological trend is the increasing number of smart devices. According to an analysis of IoT from Cisco in 2011 (Evans, 2011), by 2010, the number of connected devices has exceeded the population of human beings. In 2020, 50 billion devices will be connected by IoT.

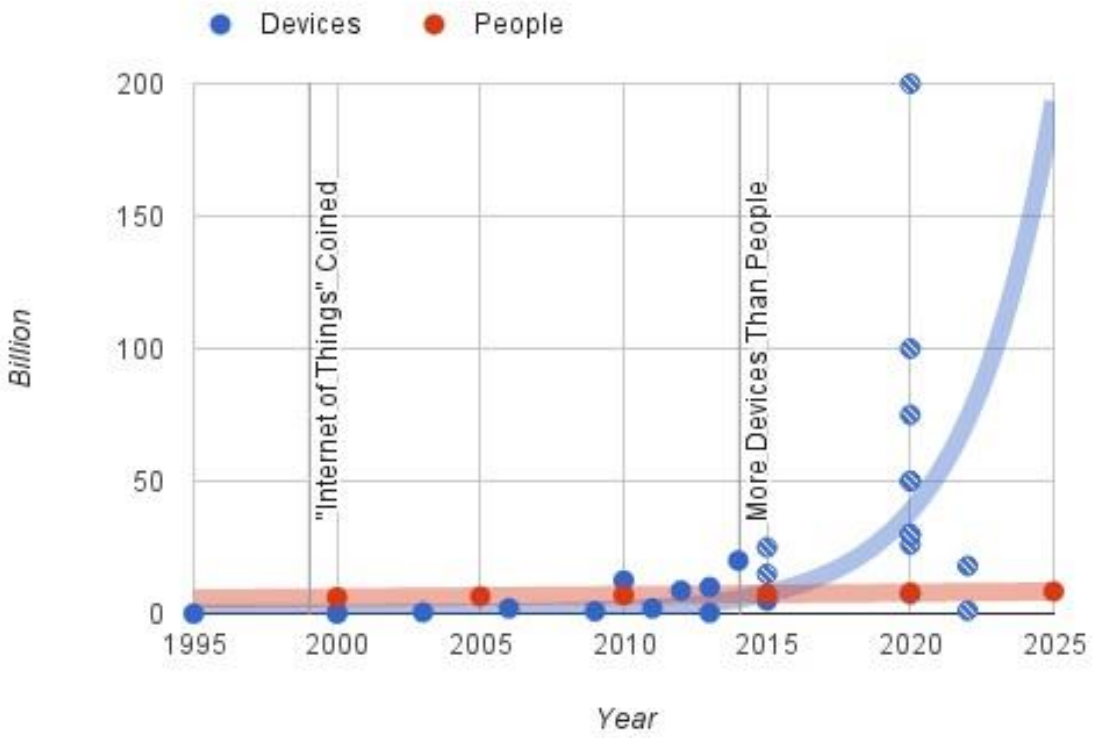

Figure 3.1 Trend of devices vs people (Barrett, 2012)

As shown above, recent research from Philip N. Howard has demonstrated this trend. Between 2011 and 2015 the connected device has experienced exponential growth.

The concept of IoT is not only a simple extension of Internet. It introduces a way to make physical devices become available in the virtual world. In order to make a real object become a "thing" on the internet, according to Dr. John Barrett (Barrett, 2012), there are several essential steps to convert an ordinary object in daily life into a smart object in IoT.

First, a unique identifier to tag a thing. Every asset in the IoT has a tag to uniquely identify itself. In this context. IPv6 is the solution for the first step because of its huge volume. Besides IPv6, IPv4 and MAC are also widely being used. However, IPv4 has a fatal limitation of its capacity. With 32-bits space, IPv4 has only 4,294,967,296 addresses which will be used up soon. In contrast, the IPv6 has 128-bits space available. Similarly, the MAC address is designed to identify a network interface on the physical network level. It adopts 48-bits address space which is better than IPv4. However, the capacity of MAC is still not enough to tag all objects on earth. Therefore, IPv6 is the best solution we have.

Second, the ability to communicate. In order to communicate with the internet, smart objects or called things in IoT must implement a way to communicate. Therefore, different transmission media need different communication mechanisms. Today, we mainly use microwave and twisted-pair as the front end media.

Third, give object senses. In the real world, an object can be identified by smell, feeling, color and so on. It is also true in the digital world. By collecting different data of an object, people can know changes of an object or its surrounding environment. Therefore, people need to implement different kinds of sensors on an object to make it become "smart". By implementing sensors, we acquire desired data of an object to share on the internet.

Fourth, a controller at remote side. Since we can get data from sensors and deliver them to a remote object through the internet, the remote side may have requirements to modify some values of a smart object through Internet. Therefore, people need a remote controller to achieve remote control.

After above four procedures, we can make sure that an object becomes smart. As a smart object, it can take part in data communication of IoT.

The highlight of IoT is the concept of gathering data from the machine and transmitting data between with pre-programmed procedures, which means less manual operations are required in IoT. In this way, it liberates more productive forces from input data to The Internet to more valuable work.
As a basic concept of ¨smart homes¨ and ¨wearable devices¨, the IoT is a promised coming future. It will make a great influence on our daily life.

In the section, we will discuss the concept of personal cloud which helps us define the edge of proposed architecture.

3.2 Personal Cloud

The personal cloud is a combination of the private cloud and the public cloud. According to Sang-Ho Na, Jun-Young Park and Eui-Nam Huh (Na,S.,Park,J.&Huh,E., 2010), “The Personal Cloud describes a user-centric model of Cloud computing where an individual's personal content and services are available anytime and anywhere, from whatever device they choose to access it.”.

In fact, there are business models available for years. For example, Seagate proposed a personal cloud solution (Seagate, 2016) which aims to provide a centralized media library where a user can access to their data from anywhere. The model consists of software on different platforms and a central server. People can access their digital assets on different platforms through different interfaces but their data are backed up in one physical device.

In recent years, with the increasing number of personal mobile devices (like smartphones, pads or wearable devices), more and more digital assets are distributed on different devices. For most of the time, those devices are not all in the radiation range of a Wi-Fi network. In this case, those scattered personal devices need to connect to a center node or a bridge device to reach the

Internet. Therefore, there are two important facts of a short-range wireless communication technology.

First, the data communication between mobile devices is relatively less intensive than PC to PC communications. It depends on the computing power of mobile devices.

Second, the network state is constantly changing from time to time. It is determined by the nature of short-range wireless communication.

After examining the context of IoT, we found that more and more smart devices are available for a single person. Those smart devices may be placed in a particular place or taken by their owners as portable devices. In order to let those devices working together to serve us, I would like to explore.

In the next section, we will discuss two popular design styles for the machine to machine communication: SOAP and REST.

3.3 SOAP and REST

In general, SOAP is a protocol which was popular in the late 1990s. REST is a design style which has been proposed with HTTP but is not popular until recent years. Technically, SOAP and REST are two different kinds of concept and not directly comparable. Here we put them together just because each of them stands for a style of design.

3.3.1 SOAP

SOAP is the abbreviation for Simple Object Access Protocol. It is a widely adopted protocol for data exchanging among web applications. Both SMTP and HTTP are notable transport protocols which support SOAP well. The format of SOAP message is based on Extensible Markup Language (XML).

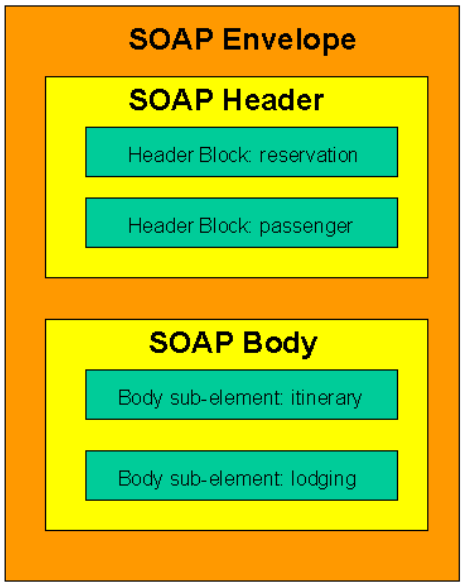

Figure 3.2 SOAP message structure example (Mitra & Lafon, 2007)

As shown above, a SOAP envelope consists of two sub-elements: Header and Body. The Header contains metadata which are optional. The Body contains the payload.

The benefit of SOAP is obvious. It allows internet communication between programs with the support of different transport protocols. Meanwhile, because it follows post/response mechanism in HTTP, it can easily pass through firewalls and proxies.

3.3.2 REST

REST stands for representational state transfer. It is proposed by Fielding, Roy Thomas in 2000. In REST, each available resource at server side can be visited through a consistent path (URI). The client can request different operations (Create, Read, Update, and Delete) through standard HTTP verbs (POST, GET, PUT, and DELETE). Although operations in REST are defined by HTTP verbs, REST is not bound with web service.

According to the author (Fielding, 2000), the advantage of REST is obvious because it "emphasizes scalability of component interactions, the generality of interfaces, independent

deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems". However, those features not just bring benefits but also limitations as well. According to whatisrest.com (Arcitura, 2016), there are six constraints for a REST architecture.

- Client-Server: The most fundamental constraint is Client-Server design style in which concerns of communication are separated. Client side needs to concern how to maintain user interface and state. On the other hand, a server needs to concern about how to store data.
- Stateless: Stateless is the most important feature of the RESTful design. It indicates that each request from client contains all information a server needs to know, and all session state data should be sent to a client after each request.
- Cache: A RESTful design should support cacheable mode by which people can save the latest response for further usage.
- Uniform Interface: It is the fundamental characteristic of a REST service which guarantees each request can independently get an individual resource.
- Layered system: An REST-based solution may consist of multiple layers. Communication between service providers and consumers are independent events which can not be affected by changes in other layers.
- Code-On-Demand: It is an optional constraint. A client can update its codes independently from a server.

The RESTful design makes web application back to the original purpose of HTTP. Since there are many existing systems adopting SOAP design style, martin Fowler (Fowler, 2010) proposed three ways towards REST (from traditional HTTP web service to a RESTful service).

- Level 1. Identify every resource: Developers need to design URI to make sure resources and URIs are one to one mapped.
- Level 2. HTTP Verbs: Use appropriate HTTP verbs under different situations instead of using POST only.
- Level 3. Hypermedia Controls: Grant discoverability to response. By adopting hyperlinks in response, users can always get links for next possible operation.

3.3.3 REST VS SOAP

It is common to see REST and SOAP are put together to make a comparison. Theoretically, SOAP is a protocol. It should not be compared with REST. However, those two terms define two popular ways to design services. According to Cesare (Pautasso, C., Zimmermann, O., & Leymann, F., 2008.April), “REST is well suited for basic, ad hoc integration scenarios, WS-* is more flexible and addresses advanced quality of service requirements commonly occurring in enterprise computing”.
In SOAP, all requests and responses must follow XML standard as well as use the verb: POST. In REST, all four verbs are available and each URLs map to resources. The following section will discuss those two technologies in detail.

- Complexity: SOAP adopts XML as transfer format, which means the complexity of serializing and de-serializing is certain. In contrast, according to difference scenario, REST can adopt different protocols to transfer data. For example, comparing with XML, JSON can store more information with less words. On the other hand, in SOAP, operations are all sealed in a POST request, which makes it become a black box for users. In contrast, operations in REST are defined by four HTTP verbs. The user can easily know target resource and action of a request.

- Scalability: SOAP was popular when IT services were only available for giant companies. It is designed to fulfill a particular task, which makes codes are hard to scale. On the other hand, REST mapping one URL to one resource. It distributes complexity of a single page into multiple simple pages. The characteristic naturally increases the scalability of a system.

- Cache: REST is naturally suit caching. As mentioned above, at the last step towards REST, possible operations of next request are included in the return. In this way, the client-side can easily to load and store result before the user makes a request. In SOAP, we always see resources are wrapped together which makes the next operation of a user is hard to predict. This characteristic makes cache becomes much harder in SOAP.

- Security: Because of WS-Security, many people believe SOAP is more secure than REST. However, it is not true. According to Flanders' research(Flanders.J, 2009)," the WS-* arena certainly has more standards than the RESTful arena (and this will probably always continue to be the case), but there are efforts to support federated security in the world of REST. OpenID is one such effort".

In conclusion, we can not simply say SOAP is better than REST or REST is better than SOAP. In order to make a right architectural decision for the proposed architecture, more characteristics need to be compared.

3.3.4 REST VS SOAP in mobile app

For a mobile app, where internet connection is fragile (nature of wireless connection). One single communication between client and server are expected to be short and light. As mentioned above, compared with SOAP, REST is more lightweight, flexible and controllable. Those characteristics meet the requirement of the wireless connection between mobile devices and the internet. On the other hand, SOAP is not a bad option when mobile apps are used to serve web application for complex business logic.

3.3.5 Conclusion

REST and SOAP are two popular design patterns. From the analysis above, it is obvious that REST is more simple and clear. In REST, each URI has clear meaning and operations are limited within POST, DELETE, UPDATE and GET. On the other hand, in regular, there are more logics in a single page of a SOAP architecture. The XML makes the SOAP can handle complex logic but also increases complexity.

In summary, REST weights more than SOAP in lightweight communication. It is especially true at the edge of the network where lightweight payload is required.

After the comparison, I chose REST as the design style of proposed architecture. Since REST is chosen, I turn to explore popular protocols based on REST.

In the next chapter, we will discuss advantages and disadvantages of CoAP by comparing with MQTT.

3.4 MQTT and CoAP

MQTT and CoAP are two popular protocols in IoT. In this section, we put MQTT and CoAP together to discuss features of CoAP.

3.4.1 MQTT

MQTT stands for MQ Telemetry Transport. It is a lightweight publish-subscribe protocol running on TCP/IP protocol. According to its official website, (OASIS Technical Committee, 2014) “MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement.”

3.4.1.1 Packet Format

According to 3.1.1 specification (OASIS Technical Committee, 2014), MQTT packet has two bytes fixed header. Therefore, the minimum size of an MQTT packet is two bytes. The following chart shows how an MQTT packet looks like according to the description of the official document.

Bit	7	6	5	4	3	2	1	0
Byte1	Packet Type				Flag			
Byte2…	Remaining Length							
Unknown Size	Optional Header							
Unknown Size	Payload							

Figure 3.3 MQTT format

As shown above, the first byte of MQTT consists of two parts. From bits 7-4, it indicates the packet type. From bits 3-0, it sets flags. Since the second byte, it indicates the remaining length of the message, the minimum size of this section one byte, the maximum size of this section is

four bytes. After specifying remaining size of a packet, the developer can add optional headers to a packet. After the optional header, the remaining section of a packet is payload.

3.4.1.2 Feature

Publisher and subscriber (pub/sub): In this model, publisher and subscriber do not know about the existence of each other. Instead, a broker will gather all published messages from a publisher and accordingly send them to subscribers after filtering. In other words, there is a central server between a publisher and a subscriber.

QoS: MQTT supports three levels of QoS. At level 0, a message will not be acknowledged or resend by a sender. At level 1, a message will be guaranteed to be delivered at least once. At level 2, a message will be guaranteed to be processed by a receiver. In this case, the sender will store a message and be ready to resend of it. Meanwhile, the receiver stores a reference of a received packet's id to prevent processing same message twice.

Last Will and Testament(LWT): The client can register a LWT when a connection is initialized with a broker. If a client has disconnected from a broker, the broker will send LWT to all clients subscribed the "lastWillTopic".

In conclusion, the MQTT is a well-designed protocol for lightweight bandwidth, the logic of it is simple where all communication between clients is based on message transition of a broker.

3.4.2 CoAP

CoAP is short for Constrained protocol. It is based on REST model and designed for constrained networks. According to RFC 7252 (Shelby, Z., Hartke, K., & Bormann, C., 2014), UDP is the default transport protocol for it. Meanwhile, it could also be used over other transports such as TCP. CoAP offers an elegant solution for low bandwidth communication with HTTP-like mechanisms.

3.4.2.1 Packet Format

CoAP has a four-bytes header which includes information of "version", "token", "length of variable-length token field", "message ID" and code of "message type".

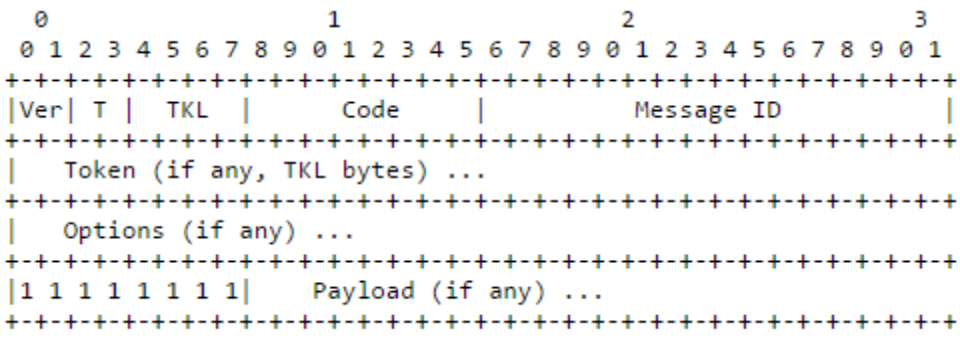

Figure 3.4 CoAP format (Shelby, Z., Hartke, K., & Bormann, C., 2014)

As shown above, the header of a CoAP message consists of five parts. The first 2 bits of a message shows CoAP version of the message. The following 2 bits indicates message type (4 options available). TKL stands for Token Length (4 bits), it indicates the length of the token field.

The second byte is Code which indicates operation of the message. There are two types of Code. One is called method code which consists of 4 verbs in HTTP (GET, POST, PUT and DELETE). The other is response code. It returns the result of a request. For example, CREATED, DELETED, VALID and etc.

The third byte and fourth byte are Message ID.

Besides the four-bytes header, token and options are additional information which is neither part of header or payload. The size of token ranges from to 0 to 8 bytes. It is used to match the request and the response. CoAP also supports some metadata in HTTP. In the Options section, a user can define parameters like in HTTP header. For example, Etag, Max-Age, Content-Format, etc. Those options simplified the process to translate between a CoAP message and a HTTP message.
Unlike MQTT where the remaining length is defined to indicate the length of the optional header and payload size. In CoAP, a 1-byte flag is used before the payload. When a processor first time encounter a 0xFF, it knows the payload start from the next byte.

3.4.2.2 Feature

CoAP has several features. I listed the most interesting four as the following:

- URI support: CoAP follows a URI scheme which is similar as the one in HTTP.

- Similar features to HTTP: Operations in CoAP are based on four HTTP verbs (GET, POST, PUT, and DELETE). Metadata and URI are available as options in CoAP. Those two characteristic makes it is easy to convert CoAP message to a HTTP message.

- Resource discovery: There is a special URI in CoAP defined as "/.well-known/core". By sending request to this URI, a client can get all available resources of a server.

- Observation: CoAP supports data observation as an extra option besides traditional send and receive. At Server side, a developer can add an observable resource to observer list. Then, a client can register resource observer to get data change notifications.

From the highlights introduced above, we know that CoAP is a compatible protocol designed for IoT. Although it is efficient, it not aims to replace HTTP. It is designed to support constrained networks under the consideration of merging them into exists networks. As shown in the following chart, we expect the CoAP handle medium or small size messages as a bridge between fat web services and low payload networks.

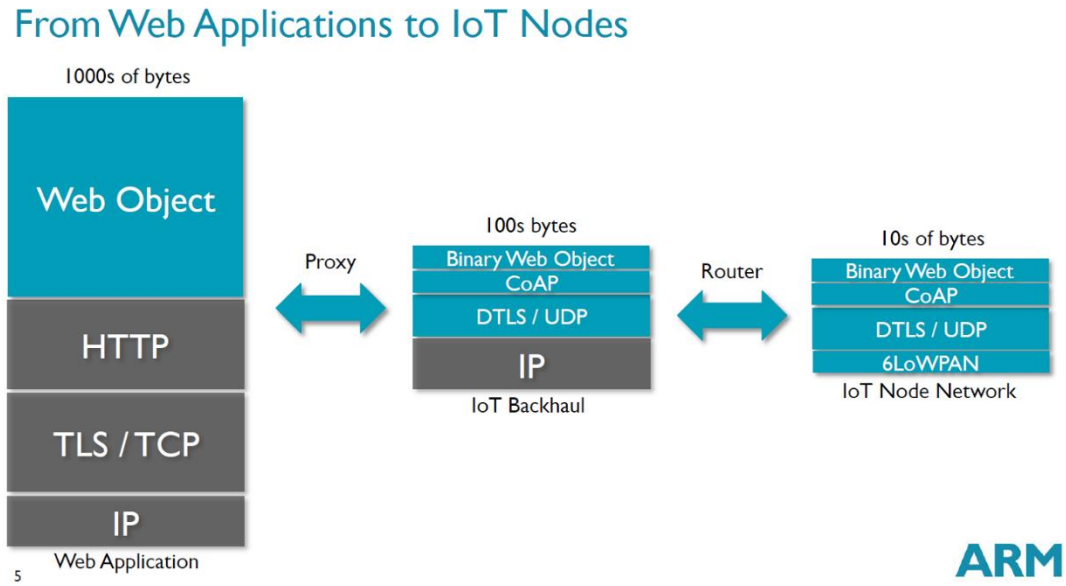

Figure 3.5 From Web Applications to IoT Nodes (Shelby, 2014)

The chart above also shows the current status of IoT network regarding payload size.
For web applications where payload size relatively high is the backbone of IoT network. The majority of web applications adopt HTTP. In general, the payload size should beyond 1000 bytes since the maximum size of TCP is 65,535bytes and 65,507bytes for UDP.

Between giant network and IoT nodes, there are routers which act as interpreters between CoAP services and HTTP services. As shown above, attributes in CoAP and HTTP are similar, which minimized works in a proxy. On the other hand, Since CoAP is designed for lightweight communication, it is can be transferred in channels with lower bandwidth (around 100s bytes). Further, at the far end nodes of IoT networks, there are many tiny devices with low bandwidth network. For example, sensors in a room, iBeacon in a small store, and remote start component in a car.

In conclusion, CoAP provides a solution for merge low payload network into the existing HTTP-based network. Apparently, it accelerates the communication between personal networks and the cloud.

3.4.3 COAP VS MQTT

Although both CoAP and MQTT can be used as IoT protocols, they focus on different aspect of data transfer.

In MQTT, it uses the concept of client and broker to replace the concept of server and client. Since the central hub (broker) is used. It is good for multiple communications between smart machines. Meanwhile, with smaller header, MQTT is more efficient than CoAP in transferring small payload. On the other hand, since it is based on subscription and push model. It expects little data in the payload. In addition, the implementation of "last will statement" makes the protocol has better performance when it is running in intermittent connectivity. Further, the support of three level QoS makes it has a clear solution for reliable communication.

CoAP is designed for HTTP-like communication. A CoAP message can easily be transferred into a HTTP message. One the other hand, the minimum size of a CoAP message is 4-bytes which makes it can also be implemented in low bandwidth environment. In CoAP, there is not a standard to guarantee quality service, but it supports "if-match," "if-none-match" and "accept" as options, which support developers to implement their own strategies for quality service. Similarly, regarding data access mechanism, as mentioned above, the last will statement is implemented in MQTT. However, in CoAP, developers have to take advantages of supported "Etag" and "Max-Age" options to implement their own solutions.

In conclusion, the MQTT aims to support multiple clients' communication within a small network. The CoAP focuses on REST communication and the communication between small networks and the Internet. The CoAP is more flexible and closer to HTTP protocol. In addition, it is friendly to developers. It requires less effort to transfer a HTTP service into low bandwidth network.

3.4.4 Conclusion

In the above section, I compared two most popular technologies in IoT. According to the analysis, I can come to the conclusion that CoAP is closer to the Internet than MQTT. The

MQTT has some nice features to guarantee communication quality, but it is designed for small area network where different terms and mechanism are used other than HTTP. Therefore, if I design an architecture where CoAP is adopted, messages between devices can easily be transferred into a HTTP format, which means the new network can merge to existing Internet with less effort. As mentioned in the problem definition, our goal is access process of merging NonIP based devices into IoT. In this case, the cost of language transfer should weight more. Therefore, I decide to choose CoAP.

Since I decided to use BLE as an example to test the proposed architecture, I need to review associate works of BLE. In the next chapter, the paper will discuss strength and weakness of Bluetooth and Bluetooth Low Energy.

3.5 Bluetooth

Bluetooth is a standard for low bandwidth wireless communication. It is maintained by the Bluetooth Special Interest Group (SIG). The latest version of Bluetooth is 4.2. The BLE has widely been implemented in smart devices like tablet, smartphone, and wearable devices. It is designed for networks with low data payload but needs frequent small data transfer. It is invented by telecom vendor Ericsson in 1994. Originally is was designed as an alternative technology of RS-232 (a serial port standard).

So far, Bluetooth consists of four technologies: Bluetooth, Bluetooth EDR, Bluetooth HS and Bluetooth low energy. Since Wi-Fi takes great advantages in the field of high-speed data transfer, SIG focuses more on lightweight and low energy communication. In 4.0 version, it proposed "Bluetooth Low Energy" to take up the market of low energy wireless communication. With the merging of wearable devices, BLE has been widely used.

In the remaining section of the chapter, I will discuss the differences between class Bluetooth and Bluetooth low energy. Further, I will explore relative works of BLE.

3.5.1 Classic Bluetooth

As mentioned above, Classic Bluetooth is a reference of Bluetooth 2.1 +EDR/4.0. It is designed for streaming data transfer. The data rate of it can reach 3Mbps. It adopts Standard Bluetooth Profiles (SPP, DUN and PAN) where one master device can have up to seven slaves. Communication through Classic Bluetooth is based on data streaming. In this way, the Classic Bluetooth has better data rate but less energy efficiency.

As shown in the following figure (SIG, 2010), in a BR/EDR piconet, two or more devices occupy same physical channel. Messages in a physical channel are synchronized by a common clock and hopping sequence. A Bluetooth can not be a master of more than one piconet, but it may belong to two or more piconet. In the following figure, the device A is a master of a piconet with device B, C, D and E as slaves. Meanwhile, the device D is a master of another piconet with J as the slave. The device E play a role of slave in both the piconet of A and F. The K is an isolated advertising node.

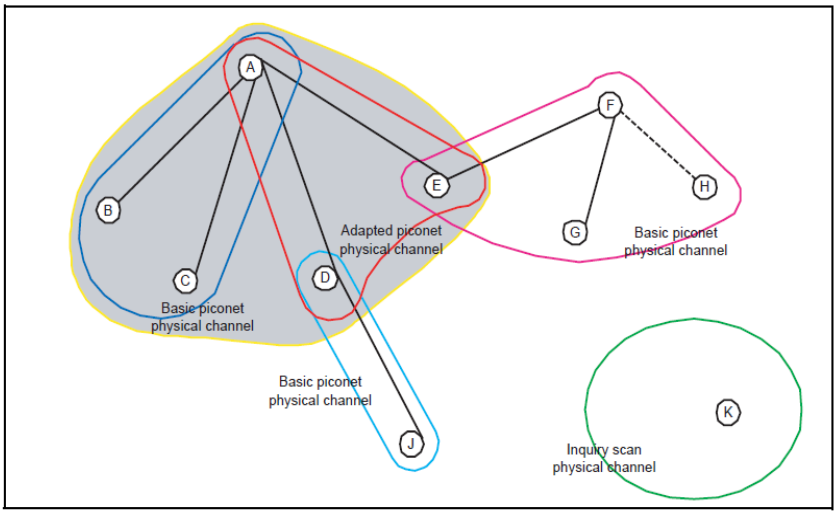

Figure 3.6 Topology of Classic Bluetooth (SIG, 2010)

As explained above, the classic Bluetooth has widely been adopted in wireless communication. However, it has a natural disadvantage. Since wireless communication is naturally fragile, the streaming data can be interrupted at any time. Moreover, in a single piconet network, one master device can be connected to up to seven slave devices. Those limitations have influenced the development of it. On the other hand, in 1997, Wi-Fi is introduced. In 1999, the data rate of Wi-

Fi reached 11Mbit/s. With the development of Wi-Fi, it gradually overlaps the use cases of Bluetooth. After the attempt of increasing data rate in version 3.0, the SIG turns to develop low-energy version of Bluetooth.

In conclusion, the class Bluetooth is a success product. However, it can not make a breakthrough in increase data rate to facing challenges of Wi-Fi. The future of Bluetooth has turned out to focusing on support smart devices by adopting BLE. In the following section, we will discuss BLE.

3.5.2 Bluetooth Low Energy

The concept of Bluetooth Smart is introduced since Bluetooth 4.0.

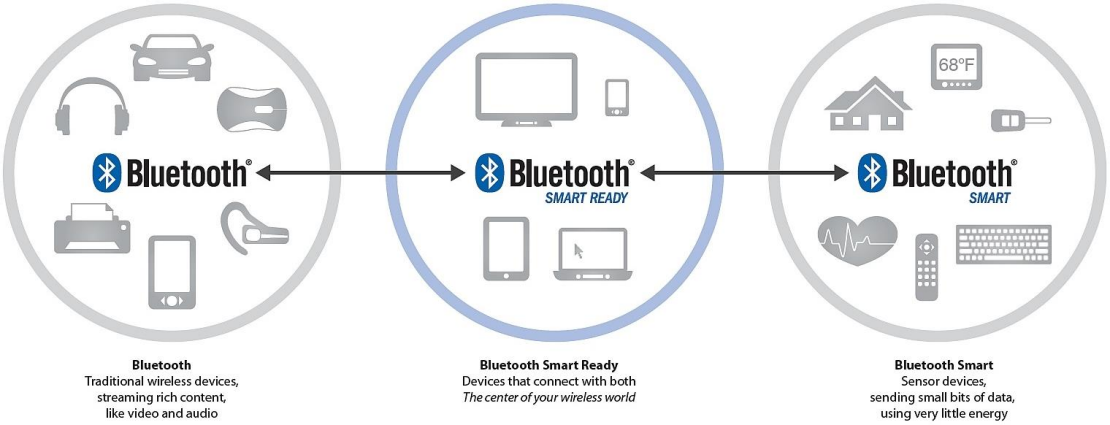

Figure 3.7 Topology of Classic Bluetooth (Torvmark, 2013)

As shown above, so far, there are three kinds of Bluetooth devices ("Bluetooth", "Bluetooth Smart Ready" and "Bluetooth Smart"). Devices with "Bluetooth" logo means it is only supports Classic Bluetooth connection. If a Bluetooth device can both communicate with Classic Bluetooth devices and BLE device, it is named as " Bluetooth Smart Ready ". At last, those only support BLE are called "Bluetooth Smart".

According to the official website of Bluetooth, the new low energy technology of Bluetooth support short data packages with the speed of 1Mbps, in addition, it supports connection setup and data transformer as low as 3ms. Moreover, its propagating range can reach 100 meters.

As shown below, unlike "Classic Bluetooth, communication between BLE devices is based on GATT (Generic Attribute Profile) and ATT (Attribute Protocol). According to Bluetooth4.0 specification (SIG, 2010): "The GATT server sends responses to requests and when configured, sends indication and notifications asynchronously to the GATT client when specified events occur on the GATT server. " As shown in the following figure, a GATT profile main contains one or more services. Each service act as a folder to contains a set of characteristics to store data. For a characteristic, there are three types of attributes: Properties, Value and Descriptor. The BLE client can read, write or monitor value of a characteristic.

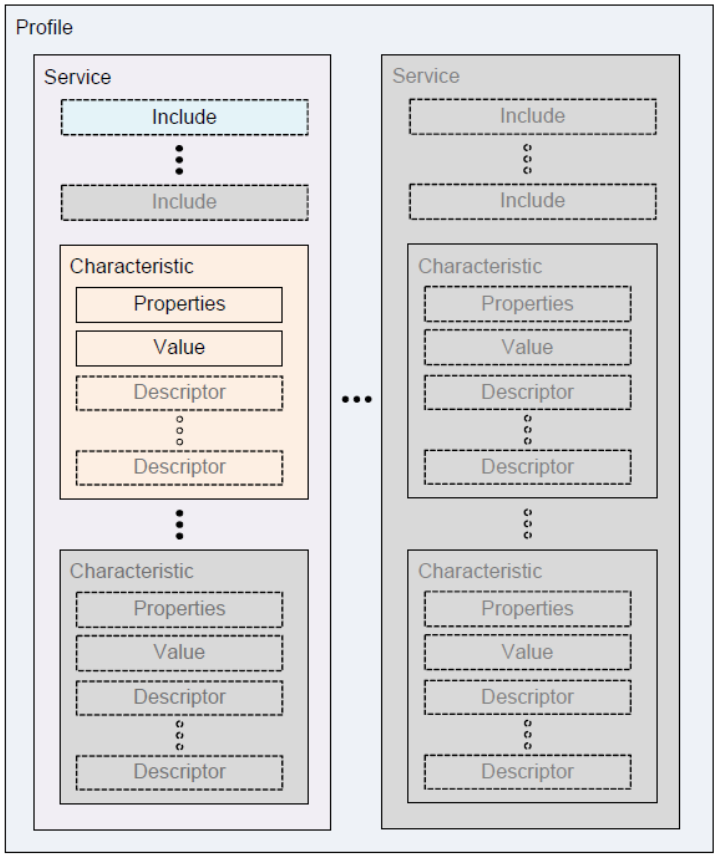

Figure 3.8 GATT format (SIG, 2010)

As shown below, there is a great difference between Classic Bluetooth and Bluetooth Low Energy in terms of topology. In BLE, one physical channel consists of two devices, which means each slave communicate with a master in a separate physical channel.

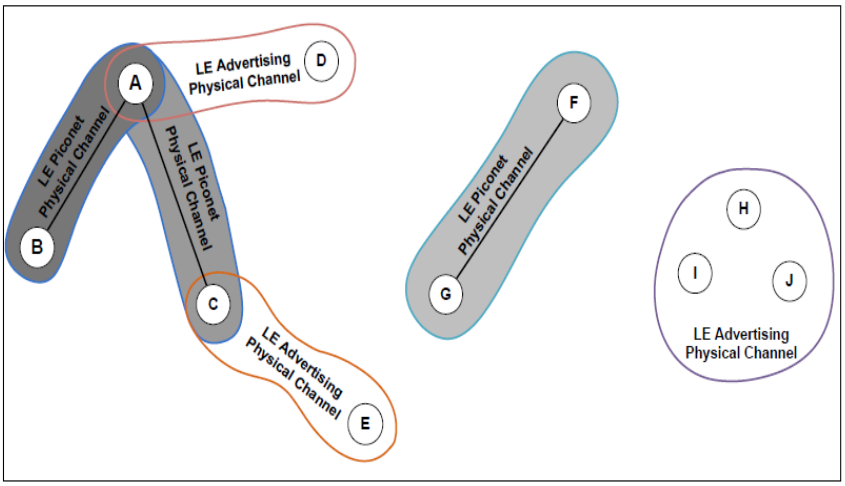

Figure 3.9 GATT format (SIG, 2010)

So far, there are two more versions available for Bluetooth low energy. The first update was available since 2013 (called Bluetooth 4.1). The second update was available since 2014 (called Bluetooth 4.2).

Bluetooth 4.1. There are four main changes in version 4.1 (SIG, 2013):

1. Solve interference: Bluetooth and LTE were interfering each other. In the version, Bluetooth try to avoid the interference of LTE.
2. Flexible Connections: Bluetooth 4.1 allows manufacturers to customize reconnection timeout intervals, which helps to reduce power consumption.
3. Multiple roles: Devices can act as hub and end point at the same time.

According to Suke Jawanda who is the chief marketing officer of Bluetooth SIG (SIG, 2013): "We updated the Bluetooth specification to address this projected growth, making changes to give developers more control in assigning a role to their product, limiting interference with other

wireless technologies, and allowing Bluetooth Smart products to exchange data faster and maintain connections with less manual intervention,"

Bluetooth 4.2. The latest version of Bluetooth is 4.2. According to FAQ document of Bluetooth (SIG, 2014), the latest version has improved BLE in three aspects: IoT capability, security, and speed.

Regarding IoT capability, a BLE device can directly participate in IoT network by adopting 6LowPAN and connect with router supporting Bluetooth Smart Internet Gateway.

Regarding security, they introduced LE Privacy 1.2 to prevent Bluetooth smart device being tracked by untrusted devices. Moreover, it uses FIPS-compliant encryption to secure data transfer.

Regarding speed, the SIG claimed the new patch will make BLE 2.5 times faster and capacity of packet will be 10 times larger than previous versions.

iBeacon. iBeacon is a protocol proposed by Apple. It defines the use of BLE in goods' information transfer. According to BLE's advertising standard, user can set up to 20-bytes payload. Apple proposed the protocol to format advertising data (iOS, 2014). The 20-bytes has been divided into three parts: 16-bytes UUID, 2-bytes major value, and 2-bytes minor value. The protocol is designed for business owners who want to push ids of their products to people who are nearby the store. The technology taking advantages of BLE's advertising mechanism which can guarantee those 20 bytes can always be visited by BLE smart ready devices when scanning status.

Taking advantages of BLE's short radiation range, the iBeacon can push data notification to a BLE smart ready device. It aims to use BLE as a tag of real products and use phone as a personal hub to retrieve information.

6LowPAN. 6LowPAN is short for IPv6 over Low-Power Wireless Personal Area Networks. So far, the latest version of BLE (SIG, 2014) has proposed 6LoWPAN implementation and Bluetooth Smart Internet Gateways to support IPv6 based communication. This means a remote device can directly control a BLE sensor by IPv6.

3.5.3 Classic Bluetooth vs Bluetooth Low Energy

Based on above introduction, we can come to a conclusion that Classic Bluetooth and Bluetooth Low Energy target different markets. For Classic Bluetooth, it is a desirable solution when higher data rate is required and more power is available. For Bluetooth Low Energy, it is designed for devices with less power than Classic Bluetooth devices. BLE is a better option when a small amount of data need to be transferred frequently. Also, the BLE has some features make it different from other short-range wireless technologies.

3.5.4 Conclusion

So far, the BLE is widely adopted in smart devices, which has brought it to an important position in the market. With the adoption of IPv6, it can join in the IoT with less effort. Meanwhile, with more and more extra protocols are developed for it (e.g. beacon). The ecosystem of it will become more and more robust. The latest changes for BLE are IPv6's capability, packet size and security level. Those improvements meet the development of IoT. Developers need Bluetooth become more compatible with existing network and can transfer more data in a more secure way. With the development of IoT, we should also expect the density of sensors will increase, and the computing power of sensors will increase.

So far, there are two innovation directions for BLE devices.

The first direction towards cheaper and simpler. This kind of sensor has simple tasks to collect limited kinds of data. For example, in a large farm land, farmer need to manage temperature and humidity indifferent spot. In this scenario, many sensors will involve in the network. They will constantly report two data to the local hub. In this case, each sensor need to be as cheap as possible.

The other direction is integrating sensors in smart devices, where different kinds of data need to be processed at the same time. Therefore, this kind device is battery constraint and bandwidth constraint. The recent updates of BLE improved its performance. Now, developers are more comfortable to develop a medium size app for integrated sensors.

In the next section, I will discuss the CAP theorem which becomes a principle throughout design and implementation of proposed architecture.

3.6 CAP Theorem

The CAP stands for Consistency, Availability, and Partition-tolerance. It is an important concept in any design of a distributed architecture. It was proposed by Brewer in 2000. In 2002, Seth Gilbert and Nancy Lynch proved (Gilbert, S., Lynch, N., 2002) the theorem and pointed out "It is impossible to achieve all three." The following are details of those three concepts.

- Consistency: Consistency means only users an operation of users can only be fully executed or dropped. For example, in a consistency constrained system, a user might get 100 dollars in an account with 50 dollars by sending request multiple times.
- Availability: Availability means a user can be served at any time. For example, in a high availability system, it can be accessed by users at any time even when it is updating or maintaining.
- Partition Tolerance: A system with high partition tolerance can still serve users if connections between two nodes are lost.

Since the intermittent connectivity is a given condition, mobile applications must achieve high partition tolerance. Therefore, the options for a mobile application are limited between PC (Partition Tolerance and Consistency) and PA (Partition Tolerance and Availability).

In the context of PA (Partition Tolerance and availability), the system and database are separated into different nodes to guarantee both Partition Tolerance and availability since data have copies on a different node. Since the whole data assets are available for each node, a user can access any data they want when one or more nodes are not available. However, in this scenario, consistency cannot be guaranteed since changes made by the user can synchronize with the lost node.

In the context of PC (Partition Tolerance and Consistency), if a system's Consistency is guaranteed while the system needs partition tolerance, the system cannot have high availability since the best scenario is to separate data. When a node lost connection, other nodes can change data without worrying about the issue of inconsistency when the lost node backs to the network. However, in this scenario, users can not access data in the lost node.

The proposed architecture aims to guarantee high accessibility. Even if a connection between devices are not available (can happen at any time for intermittent connectivity), a request should still get a response. Therefore, the system must choose PA as its design principles.

CHAPTER 4 DESIGN AND ARCHITECTURE

With the development of smart devices, the fragile network of sensors becomes more and more reliable. Under this context, we expect the CoAP should be supported in those networks. In this research, our main goal is to propose a suitable solution to support CoAP in Non-IP based WPAN.

Currently, implementations of CoAP only support IP based communication which is a natural barrier for WPAN. Technology like BLE (below v4.2) and NFC do not support IP and have their standards to transfer data. However, with the increase of bandwidth in WPAN, the CoAP communication mechanism should be implemented in those technologies for seamless communication. On the other hand, if all WPAN use the same protocol to transfer data, the proxy can easily be integrated.

In the remaining section of the chapter, we will introduce our solution: CoAPNonIP architecture.

As shown below, the proposed CoAPNonIP architecture consists of an application layer and a network layer. The Application Layer focus on message management and message delivery.

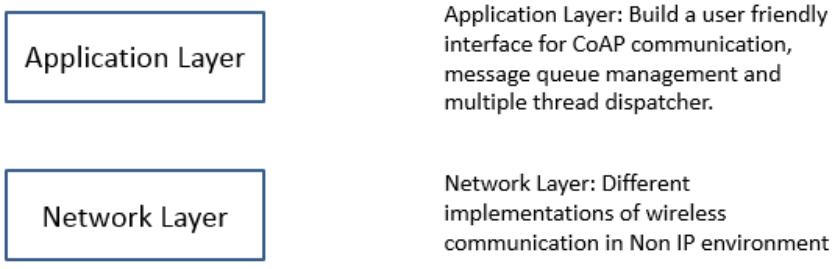

Figure 4.1 Layers of proposed architecture

4.1 Application layer

Since the aim of the project is to support CoAP protocol in NonIP based environment, we adopt an application layer to manage data and provide a user-friendly interface for CoAP developers.

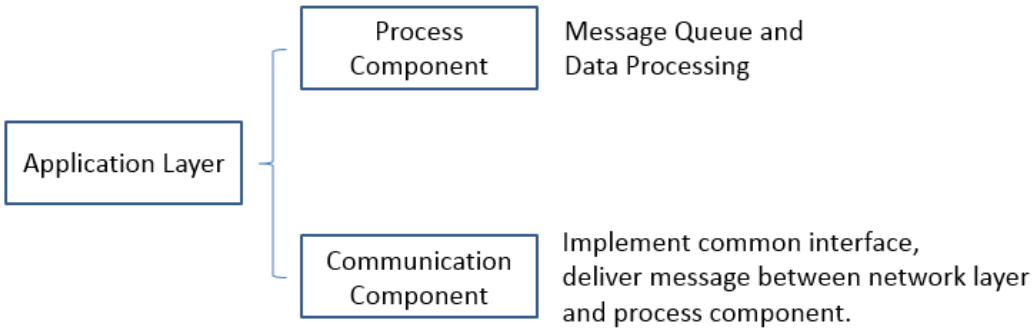

Figure 4.2 Application layer structure

As shown above, in order to grant capability of supporting different protocols of WPAN (wireless personal area network) the architecture and provide a robust management mechanism, we implemented two components in the application layer. The process component aims to provide queue management for send and receive messages. The communication layer aims to provide a common interface for message delivery between process component and network layer.

4.1.1 Process component

Process component aims to provide message management hub where message queue and cache are implemented to improve performances. In the process component, there are some important concepts.

- Receiver: Receiver is the access point of received data.
- Sender: Sender is the place to convert objects to a byte array and send to the network layer. It consists of multiple threads to handle CoAP messages.
- Processors: The processors consist of multiple threads where specific CoAP request can be processed in a predefined way.

- Resource: Resource is a concept in CoAP. It represents an available data at server side. A resource management tool is provided in the application layer.
- Callback Map: When a user tries to send a request, he or she can specify a customized callback function to handle responses to the request which will be registered in a callback map for management.
- Default Receive Handler: If user not specifies a callback function for a request message. A default response handler will be triggered when a response of the request come.

The main task of process component is managing resources and handling messages. To increase the capability of the system, we grant the architecture the ability to define multiple threads to process data. In detail, Users can define one or more processors to process received data and one or more senders to send data to lower layer.

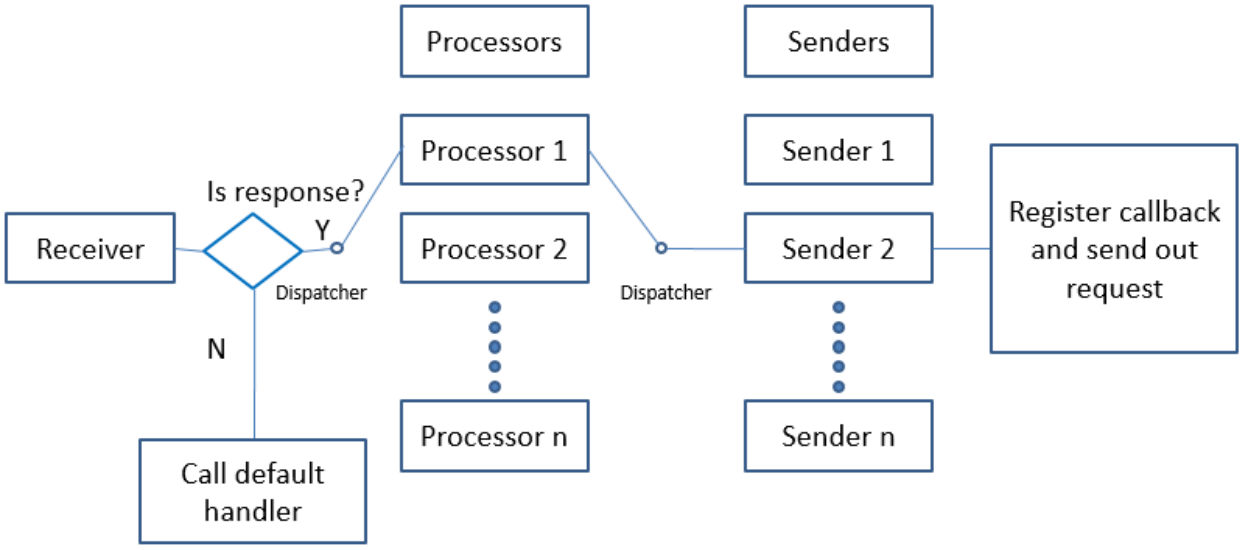

Figure 4.3 Process component flowchart

As shown above, there are three important roles in application layer: Receiver, Processors, and Senders. When a receiver gets messages from the lower network layer, it will trigger a callback which may be defined by a user or the default one. If it needs to process the received message, it will run a dispatcher to decide in which processor to process the data. After processing the data,

a response may need to send through Senders. Then, the message will be pushed into a sender thread to send the data.

4.1.2 Communication component

Message receive and send in process component are based on function calls at communication component. The communication component implemented a general interface where following actions are defined.

- Broadcast: As a server (group owner) in wireless p2p communication, a device needs to broadcast itself to make sure other devices can find it. In this state, we say the device is discoverable. Regarding BLE's implementation, the broadcast method will try to advertise itself as a BLE advertiser. If any connection is created, it shifts to a BLE peripheral device.
- Search Peers: If a device plays the role of client, it needs to search nearby broadcast signals to find other nodes. Regarding BLE's implementation, the method will trigger a search operation. If a searcher finds a broadcaster and creates a connection with it, the searcher becomes a central device.
- Get Nodes A node needs to know available devices in its network to decide destinations of a message. Regarding BLE's implementation, the method will return all connected devices.
- Send Data: Every node can send data to one or multiple target devices. Regarding BLE's implementation, the method will deliver destinations and byte array of a message to network layer for further process.
- Receive Data: Every node needs to receive data from remote devices. Regarding BLE's implementation, the method will receive data through call back function.

The communication component will be implemented according to the technology used at the network layer, which grants the architecture's reusability. Meanwhile, since the interface only consists of five functions, it guarantees low coupling.

4.1.3 Conclusion

From descriptions above we know that the application layer consists of two layers. The process component proposed a multiple thread mechanism to manage messages. It grants more capacity to the whole solution. Meanwhile, by supporting default handler and customized handler, it defines a standard procedure to handle CoAP message.

The communication component defines an interface for five common functions. For different technologies, there are different ways to implement those functions. However, the define of those functions set standard actions for communication. In practice, based on those five basic function, developers can customize more action to meet a specific requirement. In this way, this component provides great flexibility as well as set basic rules to programs.
To sum up, the application layer introduces two layers to handle messages, support multiple communication technologies and provide the capability.

4.2 Network layer

In the proposed architecture, the network layer is the underlying component for communication. The main task of network layer gets data from a remote device or upper layer, process message, deliver processed data to remote device or upper layer. For different protocols, it can be implemented in different ways.

Under the context of BLE， we create a micro service will run at background. It receives messages from an upper layer and chops message into multiple 20-bytes packets. Meanwhile, it retrieves data from remote devices, assemble them into CoAP message and deliver those message into the upper layer.

There are two roles in communication: Server is the device who advertise itself to create a connection. The client is the device who search devices to create a connection. Communication between the application layer and the network layer is achieved through message broadcast.

At the application level, a developer can define one or more instance to handle messages, which may happen in a different position of a program or different applications.
The network layer logic of a device can be implemented in a server service or client service. If a device decides to search other devices' signals, it will start a client service. Otherwise, it will broadcast itself through server service. Since the communication between network layer and application layer is through broadcast, network layer can support multiple instances of the application layer, which multiple apps can share the service of the network layer.

If the device plays a role as a server, it will create a thread to loop message queue for send out a message and an instance to receive callback functions. Meanwhile, it will define two characteristics for receive and send data respectively. Those characteristics will be accessible by clients through advertisement.

On the other hand, if a device decides to play a role as a client, it will try to search nearby server. If a new server is available, it will try to create a connection with the server. Therefore, a client service needs to maintain one or more callback instances and send thread. In the following section we will discuss the structure of client side and server side in details.

4.3 Communication Mechanism

As introduced in the literature review section, the BLE is not a stream based communication technology. Instead, a BLE connection is open and close periodically. Messages only send out in short time frames when connection are available. There are two ways to send a message in BLE: One is to send a message sequentially, which will wait for a signal from a remote device before send the next message. The other is send message none sequentially where messages are sending without wait response from the remote device. In the proposed architecture, we send and receive message sequentially. The following chart shows how BLE client and server communicate with each other.

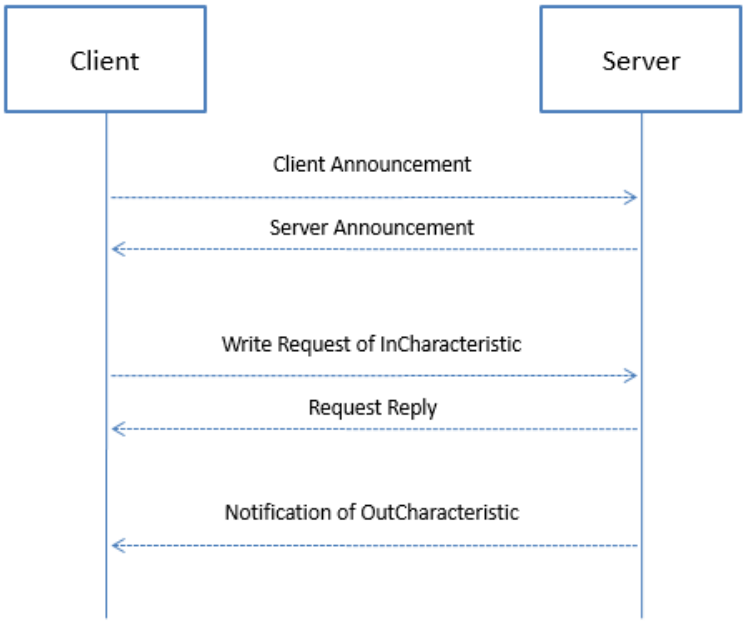

Figure 4.4 Communication mechanism

As shown above, once a connection between devices is created, the client side will send one or more announcement to the server side. Meanwhile, the server side will also send an announcement to the client. Through either announcement, destination device can get app id and user id of the source device, which makes destination device can filter messages. Once announcement information exchanged, client side can write characteristics at server side to send a message. Meanwhile, server side can send a message to the client through indication.

I introduced announcement as a “shake hands” mechanism before communication. Unlike AppID and UserID in normal messages, AppID and UserID in an announcement are values of the source device. In this way, before sending any information, both sides can know roles of the remote device. On the other hand, AppID and UserID in normal messages are values of the target device. Since the architecture uses AppID and UserID fields in two different ways, it can overcome the chaos of message received at the server side as well as act like unique ID to filter messages for different applications.

The proposed communication mechanism is based on two characteristics' operations: "InCharacteristic" and "OutCharacteristic". To provide reliable data transfer, every request from one device to another need to wait for a reply before sending the next message packet. As explained above, data communication between two BLE devices is based on data change in characteristic. So the proposed architecture taking advantages of the architecture by using two characters to send and receive messages for a BLE server. One the other hand, the architecture expects messages in the communication channel are sequential by waiting return signals of characteristics write request to make sure sealed CoAP message pieces can be assembled at destination.

4.4 Packet Format

As shown below, in the proposed architecture, we designed a protocol based on the 20-bytes size of a BLE message. The protocol consists of two parts: 4-bytes header and 16-bytes payload.

The first 2 bits of a packet indicates the type of it. Currently, there are three types available: announcement, continue packet and end packet. 11 is reserved for future use. Announcement message is a special type of message which only has 4-bytes header. It is used to let another device which user and app are sending the request. 01 indicates a packet is a continued packet, and the service should expect more packet to assemble a CoAP message.

The following 14 bits are application ID where numbers are used to indicate which app are sending the request. The arrange of the number is from 1 to 16383.

After AppID, the UserID is used to indicate which user are sending data. The arrange of it is from 1 to 65535.

So far, we use first 4-bytes of a packet as a header. Therefore, we still have 16 bytes as payload to carry messages. Since CoAP message also has a 4-bytes header, the actual payload of a CoAP message in each packet is 12 bytes.

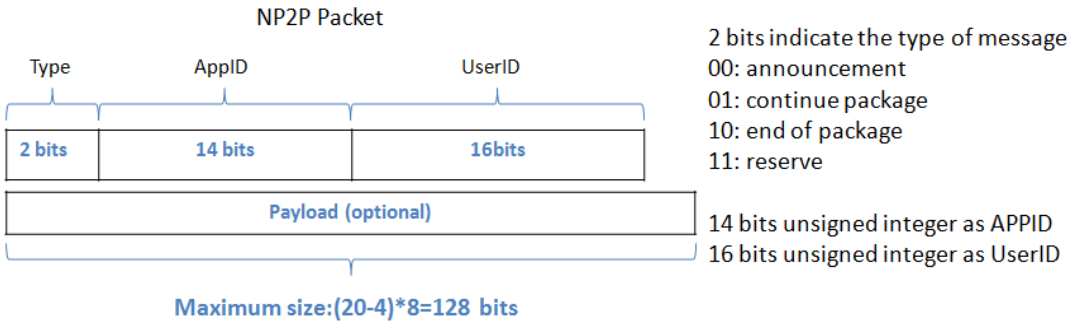

Figure 4.5 Packet Structure

4.5 Virtual Resource

In standard CoAP protocol, all request and response are based on a resource on a device. In this scenario, resources are binding on one device. However, in many cases, the one mechanism needs to gather same data from different devices for minimum errors of sensors. Moreover, the data collect behavior may have a timeout property to guarantee performance. Since more and more cheap sensors are available in the market, we can safely say this need will increase.
In the proposed architecture, we implemented a virtual resource mechanism to meet the need of multi-sampling.

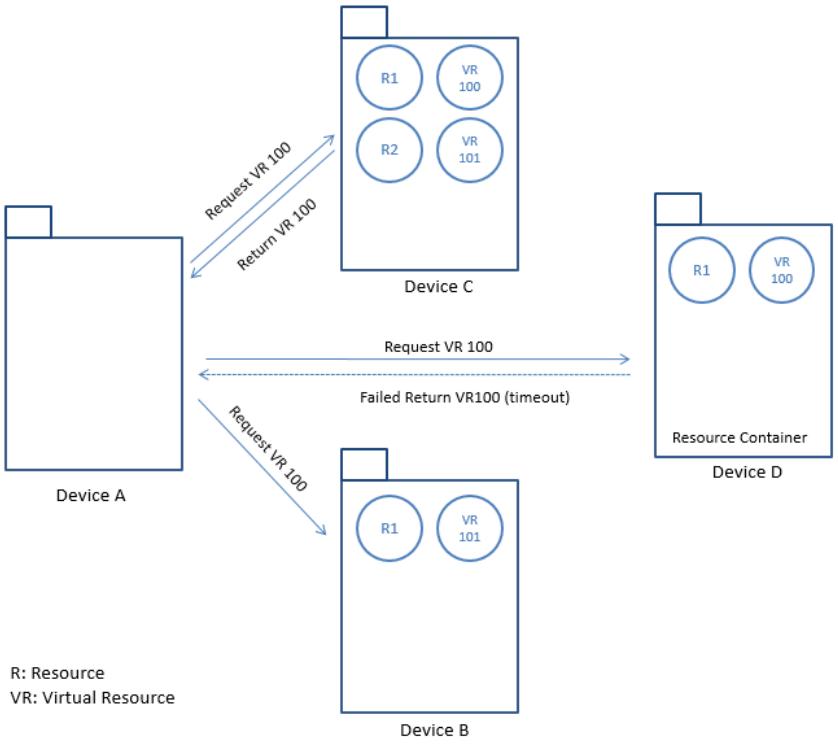

Figure 4.6 Virtual Resource Mechanism

As shown in the above chart, in the view of resources, all involved devices are resource containers. The device A broadcast a request (request the value of virtual resource 100) to all connected devices. In device B, it does not respond the request because resource 100 is not available. The device C successfully return the value of resource 100. The device D have resource 100 but it did not respond in time. Therefore, the device A will receive one return.

The virtual resource is achieved by introducing a timer to wait for return values from connected devices.

4.6 Detail Design

4.6.1 Application Layer

As mentioned in previous chapter, the application layer of proposed architecture consists of process component and network component.

Me and XiaoDan.Li designed the application layer and implemented its process component.

We will discuss the details design of those two in the following section.

4.6.1.1 Process Component

The process component defines how the user communicates with underlying architecture. As mentioned above, the application focuses on behaviors' define and management, which means instead of taking care about how to send out a message or how to prune into an appropriate format. It focuses on loading balance as well as define of actions and target. There are five basic functions:

1. InitReceiver: In this function user defines actions when received messages are delivered into the application layer. If you do not want to customize it, the architecture will send a message to a default data handler.
2. InitProcessor: By implement this function, a user can create numbers of threads to handle received CoAP request. Those threads will execute predefined actions of a specific resource.
3. InitSender: Since multiple threads are available to process a request, the architecture also can create one or more thread to send out messages.
4. SetDefaultResponseHandler: To handle requests which do not appear on the map. The user needs to provide a default action for them.
5. Run: After complete all necessary procedures, the user can start the CoAP service with different parameters. Since the "pairing" processor are required by most wireless communication protocols, we defined four different roles to start-up a service.
   1. BroadCaster: The device will advertise itself to wait for an connect request.
   2. Seeker: The device will search nearby broadcasters to create a connection with it.
   3. Auto: The device will try to find an available broadcaster in limited time. If no broadcaster found in time, it will turn to broadcast mode to broadcast itself.
   4. Mix: The service play will broadcast itself as well as available search broadcaster. Due to the performance issue, the current implementation of BLE does not support this role yet.

4.6.1.2 Communication Component

As mentioned above, in the network components of the application layer, we defined three callback functions and six abstract functions. Developers can customize the behavior of those function to support different communication technologies. Regarding callback function. The architecture defined three events: peer found, peer lost, and data receive. Developers can either handle the events at communication level or expose it to process component. Regarding functions, they defined actions for broadcast, search peers, sniff peers, get nodes, send data and receive data.

- BoradCast: Initialize a device as a broadcaster and begin to advertise itself.
- SearchPeers: Initialize a device as peer seeker. Since searching peers is a battery-intensive activity, the search will stop if no broadcaster is found in certain time.
- SniffPeers: Initialize a device as a peer seeker. If no device is found in certain period, it will shift role to a broadcaster.
- GetNodes: Return all connected devices. It is used to help the user to determine data destination and network status.
- SendData: Send data to one or more destination.
- SetRecveDataFunc: Catch event of data received.

4.6.2 Network Layer

As mentioned above, in the proposed architecture, a device can either play as a client or a server. A client or a server seal its communication mechanism in a service. In this paper, service means an independently running component which can provide data to the body of a system. The independence of a service makes it possible to support one or more program at the same time.

In the following section, we will discuss the detail design of client and server separately.

4.6.2.1 Client Side

Service. As mentioned above, the client side runs independently as a background service. It serves one or more app through message broadcast. The service can automatically search nearby available servers and connect with it. Whenever a client service gets an available server through

search, it will create a new thread to communicate with remote side. The client service will maintain each communication thread in a list of threads. A communication thread in a client will not only listen to events from the remote side but also maintain a message queue to send (through a send thread) in its channel.

Message Processing. As shown below, a client service will listen to two kinds of messages from the upper layer: announcement and data. Those two kinds of messages will be assembled into different data formats for communicating in BLE channel. Since the size of a CoAP message can change easily over 20 bytes, the system may need to cut messages into multiple packets which will be transferred to the communication channel and assembled at destination. After receiving any send request, it will pack information into target format and put it into sending message queue. On the other hand, the client service will open Bluetooth adapter to scan available server. When a server is found, it will create a thread to connect with it and create three handlers: OnMessageReady, OnReceiveAnnouncement and OnLostConnection. The OnMessageReady will be triggered when a complete CoAP is received. The OnReceiveAnncouncement will be triggered when an announcement is received. (announcement is a special type of message, which is designed for information exchange when a connection is initialized). Since we force client and server to exchange announcement whenever they establish a connection, the architecture also regards it as a signal of connection created. The OnLostConnection is triggered when a connection is lost. Those three handlers handled connection ready, connection lost and message available three basic messages of a communication channel. If any of them is triggered, the client server will broadcast message to the upper class.

Regarding sending and receiving messages, when messages need to be delivered, the client side will write data to the InCharacteristic. It will listen the OutCharacteristic to receive data from server side.

In terms of message type, the first two bytes of the proposed protocol indicate the type of a message. As mentioned above, there are three types of messages. As soon as get a message from the remote side, the receiver will retrieve its type. If the message type is 0 (an announcement), the receiver will construct an object with information of remote UserID, remote AppID, and

remote mac address. Further, the receiver will trigger announcement received event where constructed object is the parameter. If a message type is 1 (a continue packet), the receiver will add received the message to a hashmap where the combined string of UserID and AppID is the key. The hashmap maintains messages received from the connected device. If the message type is 2 (the end packet of a CoAP message), the receiver will add the message to the hashmap, combine existing message pieces into an object and trigger a message receive event.

4.6.2.2 Server Side

Service. As mentioned above, the server side runs independently as a background service. It serves one or more app through message broadcast. A server will automatically broadcast itself. Remote clients and discover it and try to connect with it. Once a connection created, connected devices can read and write a value to the server. Unlike client service, a server service only has one thread for send message because all messages from clients go to one function call, which is restricted by the implementation BLE. Therefore, the system will only create one send thread for data notification from server to client.

Message Processing. Similarly, the server-side listens send an announcement and send message requests from an upper layer. When new request received by the service. It will process the data and push data packets to a queue. A send thread manages the queue and constantly send notifications.

Regarding sending and receiving messages, the server service listens to data changes in InCharacteristic as a data receiver. Meanwhile, when data need to be send, it will change the value of OutCharacteristic and send a notification to the client side.

Regarding message type. The server side also supports three types of message which has been described earlier. Once message received, the server-side need to parse it into an object and trigger a message receive event.

4.6.2.3 Flowchart

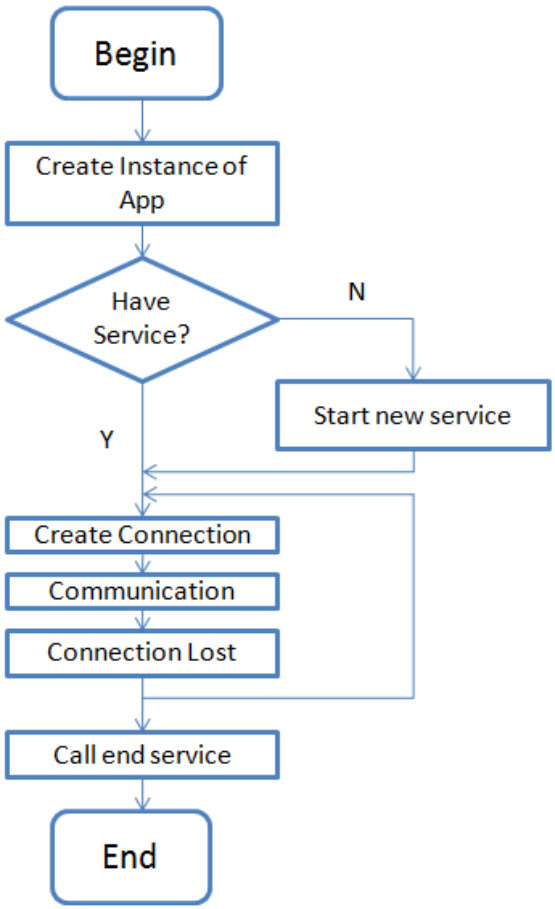

Figure 4.7 Life cycle of network service

The flowchart above shows lifecycle of a service. As shown above, whenever user need create a CoAPNonIP service, he or she needs to create an instance of an object called App. When initializing the App, logics at application layer need to be specified. After calling several functions to set the application layer (e.g. receiver thread, processor thread, and default receiver), users need to call the run function to check if a network layer service exists or not. If no service is running at background, a service will manage the lifecycle of a connection. The service will keep running unless user force to stop it.

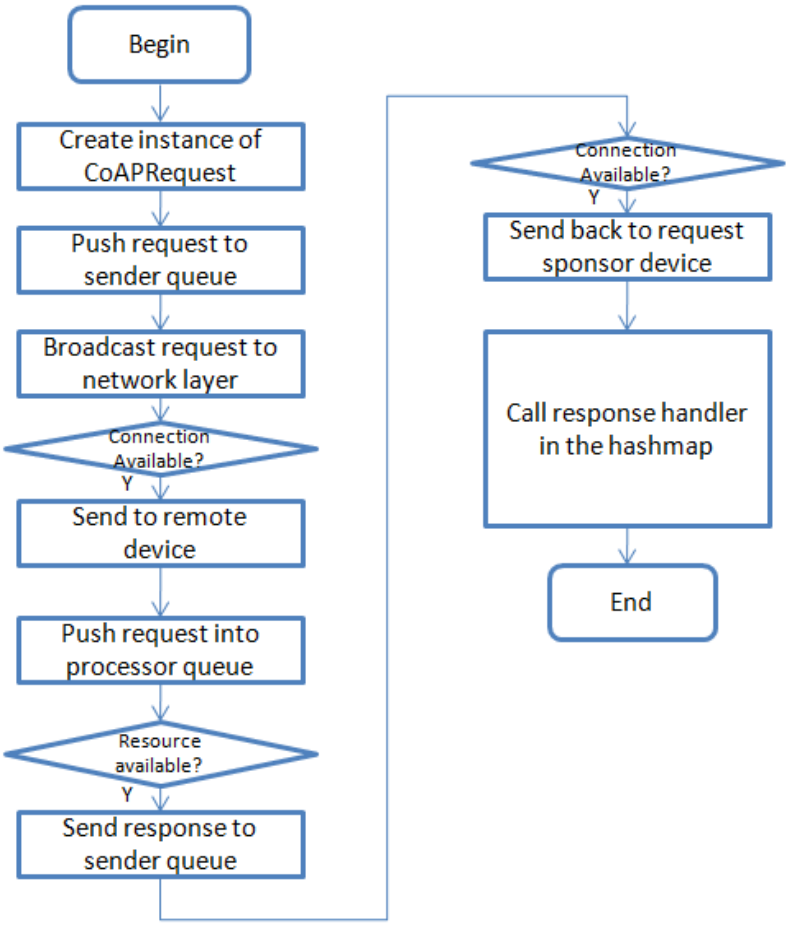

Figure 4.8 Life cycle of CoAP communication in proposed architecture

The flowchart shows the life cycle of communication. Since the architecture is resource driven, a request needs to specify resource URL and targets. After creating a request, users need to send it to the sender queue and register a handler for response in request hashmap. When a sender thread is processing the request, it will send it to network layer service. In the service, the program will judge if a connection with the target device is available and send the message to the target. When a request is received, the network layer of remote device will throw it to the upper layer. In the application layer, it will be pushed into processor queue. When a processor thread retrieves it from the queue, it will judge whether target resource is available. If the resource is available, it will generate a CoAP response and send it back. Finally, in the application layer of request sponsor, the program will try to find a handler in the hashmap (key value pair of message id and handler) to handle the response.

As explained earlier, the network layer of BLE defines how two devices can communicate in the proposed architecture. Both the design of communication mechanism and packet format involves two factors. One is to find a way to overcome the limitation of BLE. The other is to support multiple roles' communication. Since the BLE network is fragile, the design of communication mechanism and data format aims to support sequential communication. Because of sequential communication, it is easier to manage data packets under the context of detached CoAP message.

4.7 Solutions for proposed problems

4.7.1 Merge Non-IP based device into IoT

The proposed architecture adopted the concept of AppID and UserID. As mentioned above, an application can get a unique identifier by combining those two numbers together. In this way, we make the identifier become a concept beyond a physical device. In the physical layer, there may have either MAC address, IP address or another way to identify a device but they all need to use AppID and UserID to identify itself in the proposed architecture. One advantage of the design is the system can easily change devices with less effect on communication because the architecture does not bind a physical address to a device. Moreover, the adoption of AppID and UserID grants flexibility to the architecture if multiple communication technologies need to be supported at the same time.

4.7.2 Size limitation of a BLE request

I proposed an automatic packet cut and assembly mechanism. In our solution, all CoAP messages will be arranged into one or more 20-byte package (The proposed format has been described above). In this way, the size limitation of BLE is solved at the software level.

4.7.3 Chaos server-side callback

As mentioned above, the server side of BLE is chaos. A BLE server will receive different messages from different clients in one channel. Therefore, we use AppID, UserID and as the identifier to build a hashmap for each connected device. As mentioned in the design of packet.

The AppID and UserID information are contained in each packet. Therefore. The server side can always know where a message come from.

4.7.4 Serve multiple applications

With the development of tiny sensors. Engineers can set up more and more sensors in one device. This trend has been provided by smartphones and smartwatches. In this architecture, the user can declare role in applications. When a new message come, the network layer will simply broadcast message to the upper level with information of remote devices' AppID and UserID. The developer can select interested messages from a specific data provider. This design makes the architecture can serve one or more applications at the same time.

4.7.5 Provide common interface to support wireless communication protocols

As mentioned above, the proposed architecture consists of the application layer and the network layer, which separate the controller and the network behaviors. A developer can make the architecture support different communication technologies by overwriting communication component of the application layer and network layer. Meanwhile, since the front end functions are highly abstracted, application developer do not need to rewrite data management at the application layer.

CHAPTER 5 IMPLEMENTATION

To test the performance of proposed architecture. I made a test program to create a BLE connection between two devices and send package between them. The application is written in C# and compiled into native Android application in a cross-platform framework: Mono (maintained by Microsoft).

The procedure of implementation consists of two stages. In the first stage, I implemented the BLE infrastructure to support transfer CoAP message. Meanwhile, I was working with House Lee to design interface of the application layer. In the second stage, two apps were developed to do experiments. The first App is called “Major App.” It is designed to do first three experiment. The second app is called “Trigger App.” It is designed to play the role of the second sender in the fourth experiment.

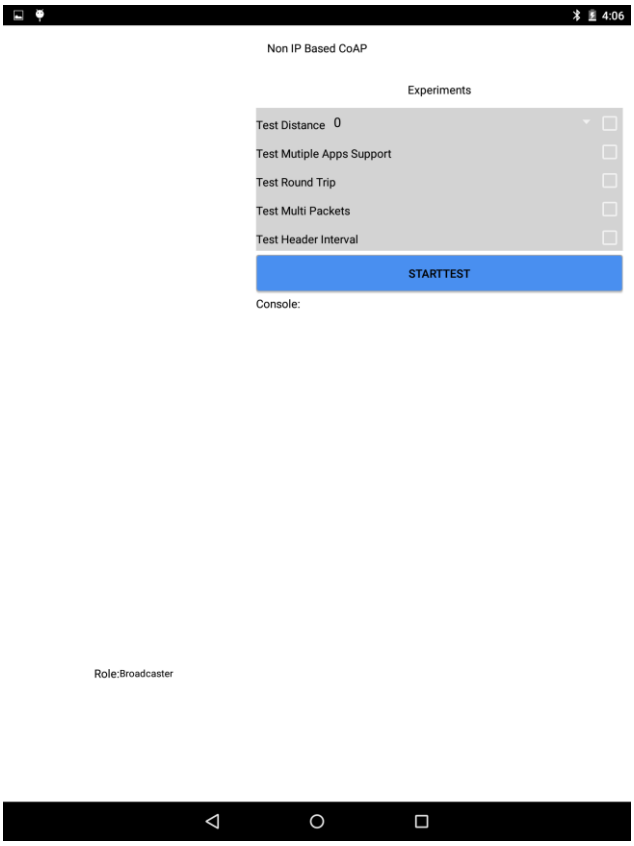

Figure 5.1 Implementation screenshot

As shown above, the UI of Major app consists of three sections. The left side contains a list of available devices and a role indicator. The top of the right side is experiment selection section with start test button. The remaining section of the right side is output console where experiment information is recorded. The UI of Trigger app is a lite version of that in Major app where experiment selection section is disabled. For those experiment apps, there is a producer thread generates designed packets and call send request function to send a message out.

CHAPTER 6 EXPERIMENT DESIGN

In this chapter, we will evaluate the performance of proposed architecture in BLE. As mentioned above the architecture is compatible with different technologies. In the implementation, I adopted BLE to make a prototype. So far, all experiments are under the context of BLE.

Although Bluetooth low energy has been widely tested and adopted, we should keep in mind that from Android device to Android device communication through BLE (peripheral communication) has not been widely tested. So far, only latest Android devices with Bluetooth4.1 hardware support peripheral mode.

6.1 Goals of Experiment

Experiments of the system focus on accessing the communications between mobile devices. I designed five experiments and listed them in the below table.

Table 6.1 Experiment Goals

Title	Goals
Minimum Data with Interval	Test performance of send CoAP header with different intervals.
Multiple Packets	Test performance of send different CoAP messages with different data size.
Round Trip	Test performance of round trip
Multiple Apps	Test performance of underlying service to support multiple apps

6.2 Experiment Setup

As explained earlier, the proposed architecture is designed for server-client communication. We use two Android devices to do the test. Since only latest hardware with Bluetooth v4.1 supports android to android Bluetooth communication, we used two Nexus 9 to test BLE implementation of the architecture.

Table 6.2 Device specification

Hardware	Details
OS	Android OS, v5.1.1(Lollipop)
CPU	Dual-core 2.3 GHz Denver
Memory	16GB/2GB RAM
Bluetooth	v4.1, A2DP, apt-X

Two devices run a test app (more details in implementation chapter), the app can connect two devices together through BLE. Then, one device can start to send messages.

6.3 Details

6.3.1 Minimum Data with Interval

Description

As mentioned in the chapter of design, we implemented BLE in sequential send mode which means messages are not sent out at the same time. Instead, each message needs to wait for a signal from the remote device before writing a new data to characteristic. In this way, the connection between devices become more stable.

Since the design of BLE aims to support lightweight data transfer, it is important to get its performance with the minimum payload. In this test, I send 4-bytesheader between two devices. I expect to find some patterns of time cost of sending data with delay.

Procedure

1. Start sample program scan and create a connection between two devices.
2. Send 4-bytesheader (100 times with 0,50,100,150,200,250,300,350,400,450ms respectively) with different intervals to remote device.
3. Record received time at sender side (get 99 sets of data).
4. Calculate transfer time according to time gap between message been received

Result and Analysis

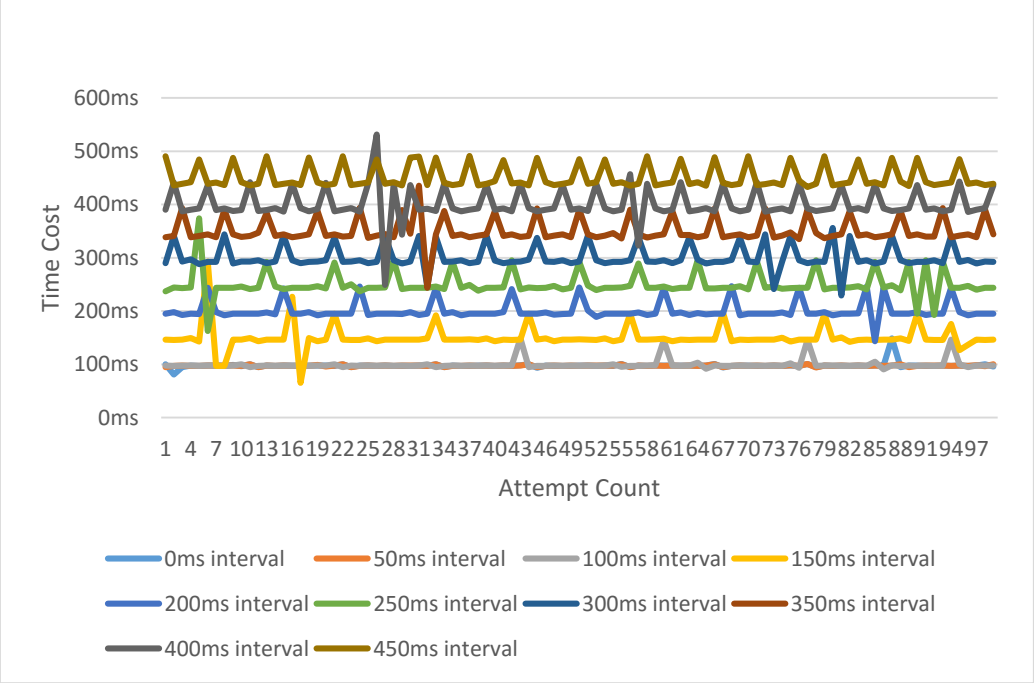

Figure 6.1 Results of sending headers with different interval

As shown above, ten series of data were recorded. The x-axis is the number of samples (99 samples for each series). The y-axis is the data transfer time plus waits time (unit is MS). From the above chart, we can get four interesting points.

First, the first three series have similar y value. The average time is around 100ms. On the other hand, since the series 4, the Y value increases uniformly by 50ms. The reason is when we sent a message to the message queue with interval 0ms, 50ms, and 100ms, the message does not need

to wait before it been sent because the system always needs to wait for a response signal of the previous message before sending a new one out. Since the time of sending a message and wait for response signal is around 100ms. The message the Y value of first three series are around 100ms.

Second, there is a heartbeat-like pattern in those series. The pattern always observed with the increasing of interval time. As shown in the blowing three charts. The heartbeat-like pattern is also visible with 150ms delay and 200 delays. The heartbeat-like pattern has similar amplitude (around 50ms). The pattern can be explained with energy saving strategy of BLE.

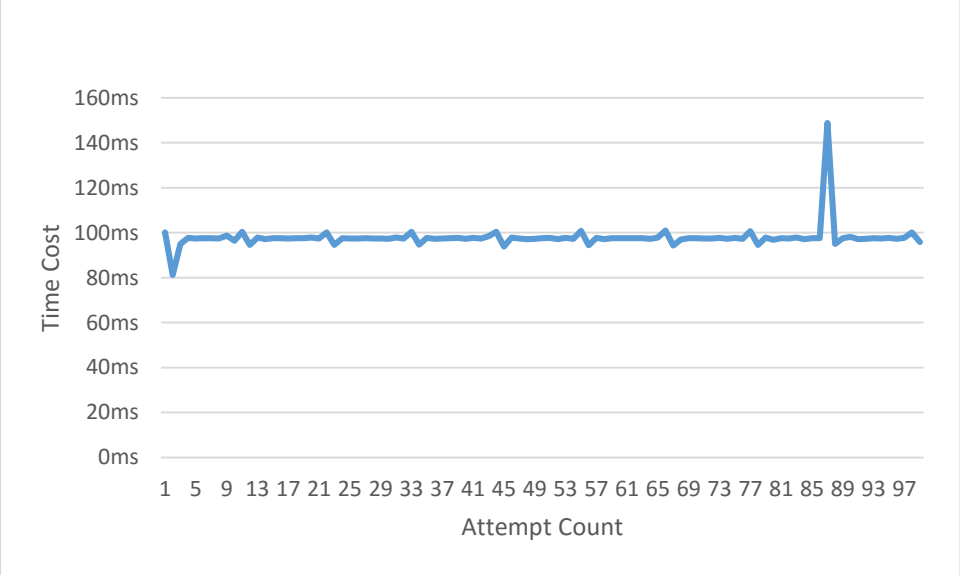

Figure 6.2 Result of sending headers with no interval

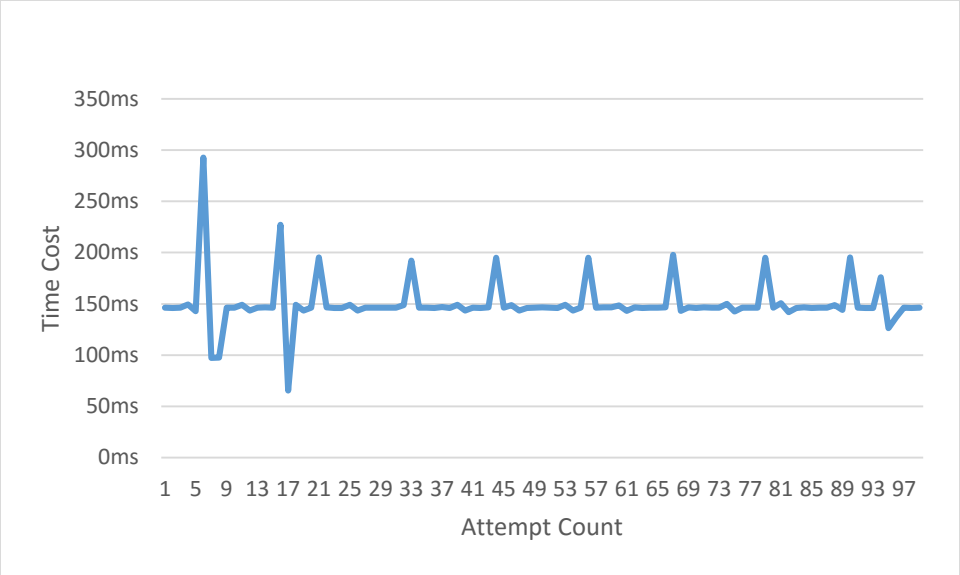

Figure 6.3 Result of sending headers with 150m interval

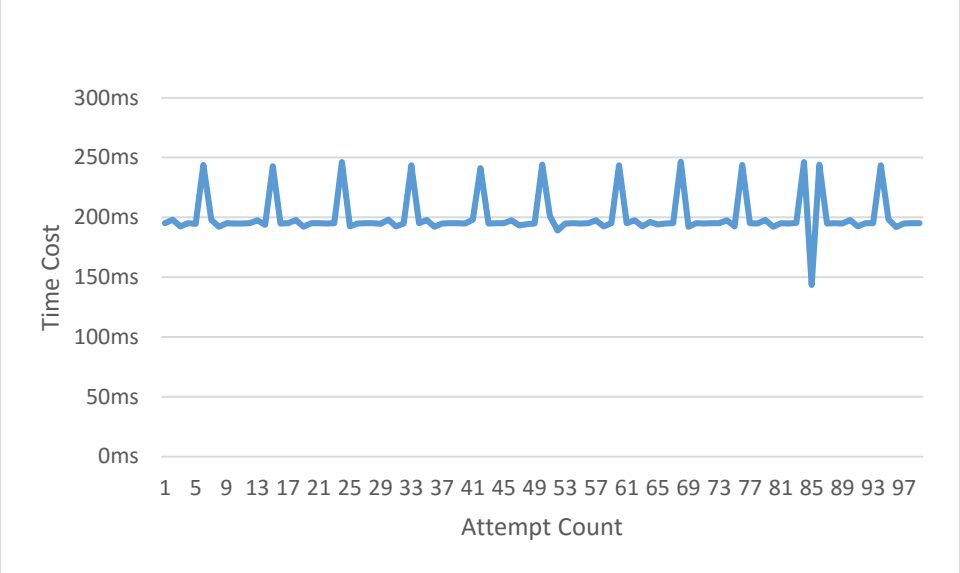

Figure 6.4 Result of sending headers with 200m interval

Third, as shown in the above, there are not only randomly spread fluctuations but also have random amplitude. This pattern can be explained with a known issue with Android Lollipop's BLE implementation or context switch problem.

Last, from those three charts, we know that those random fluctuations become a regular pattern when the system sends data with delay. This phenomenon can be explained by communication interval of BLE.

To further investigate the behavior of the system. We design another set of test focusing on size change of BLE packet.

6.3.2 Multiple Packets

Description

From the first experiment, we found three interesting patterns. We give out possible explanations for the last two. However, we need further experiments to investigate reasons behind them. As explained earlier, we send data in BLE by write and read data in characteristics in each packet we can maximally send 20-byte data (in other words, each packet can send 20-byte data).
In this experiment, we send multiple BLE packets of data by control the payload of CoAP message. We expect to find out whether the data pattern of experiment one still exists. Meanwhile, we expect more evidence to support our guess of a data pattern in the previous experiment.

Procedure

1.Start sample program scan and create a connection between two devices.
2.Send CoAP message with payload 4 (1 packet), 12 (2 packets),28 (3 packets), 44 (4 packets), 60(5 packets), 76 (6 packets) and 92(7 packets) 100 times respectively (0 interval time).
3.Record receive time when received a response from the connected device.
4.Calculate transfer time according to the time gap between message been received.

Result and Analysis

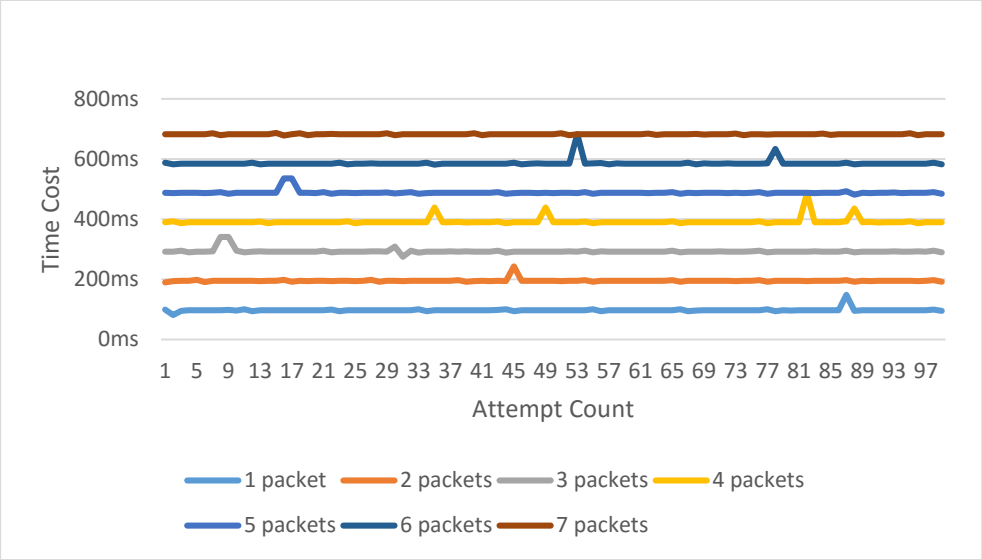

Figure 6.5 Results of sending multiple packets

As shown above, we get 7 series of data to show the performance of the architecture with the increasing size of BLE packets. From the summary chart, we can get following information:

1. With the increasing size of messages, transfer time linearly increases from 100ms to 700ms.
2. The heartbeat-like pattern still exists.
3. The random fluctuation observed in previous experiment still exists in the chart.

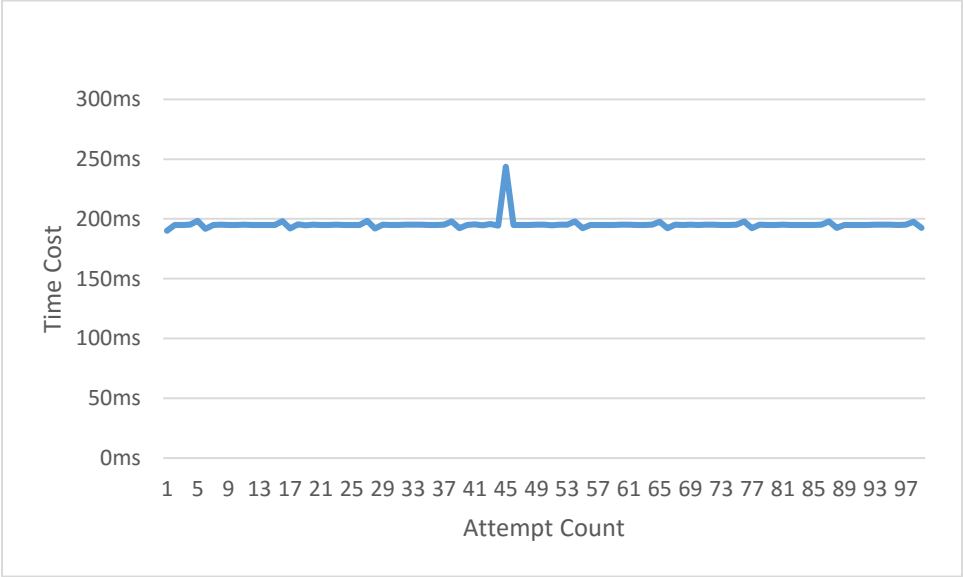

Figure 6.6 Result of sending 12-byte payload (2 packets)

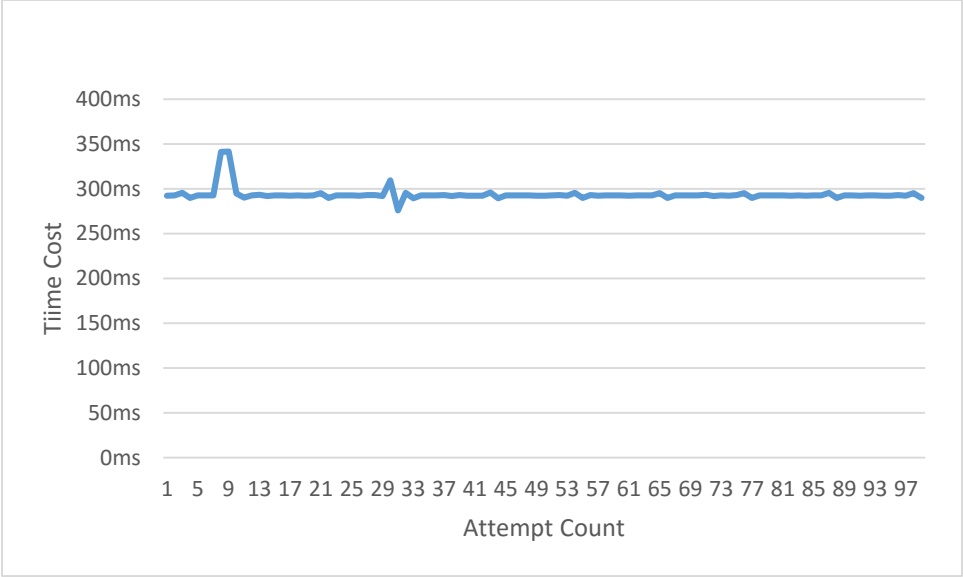

Figure 6.7 Result of sending 28-byte payload (3 packets)

As shown above, both random fluctuation and heartbeat-like pattern exist in those two charts. Compared with detected random fluctuations in experiment one, it consists the pattern of Figure 8 where random fluctuations are not predictable and the amplitude below 50ms. On the other hand, it shows the delay of data will influence the random fluctuation.

The experiment supports our explanation of the random fluctuation and heartbeat-like pattern.

6.3.3 Round Trip

Description

In order to test the performance of real-time data communication between two devices, we designed round trip experiment where the receiver of 4-bytes header request always sends a response to the sender.

Procedure

1. Start sample program scan and create a connection between two devices.
2. Send 4-bytes header 100 times (0 interval time).
3. Record received time at sender side (get 99 sets of data).

4. Calculate round trip time according to the time gap between message been received.
5. Repeat 2 to 4 steps three times.

Result and Analysis

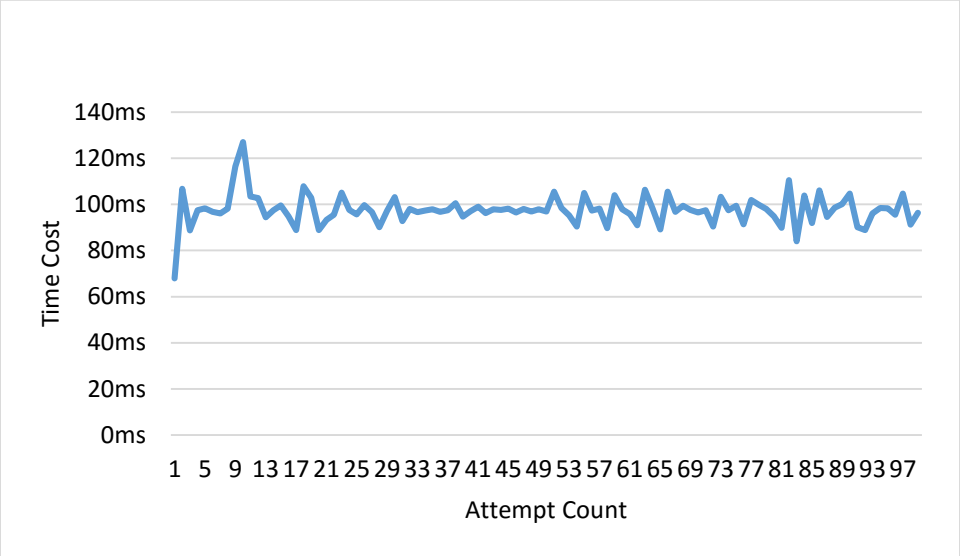

Figure 6.8 First set of 4-bytes round trip

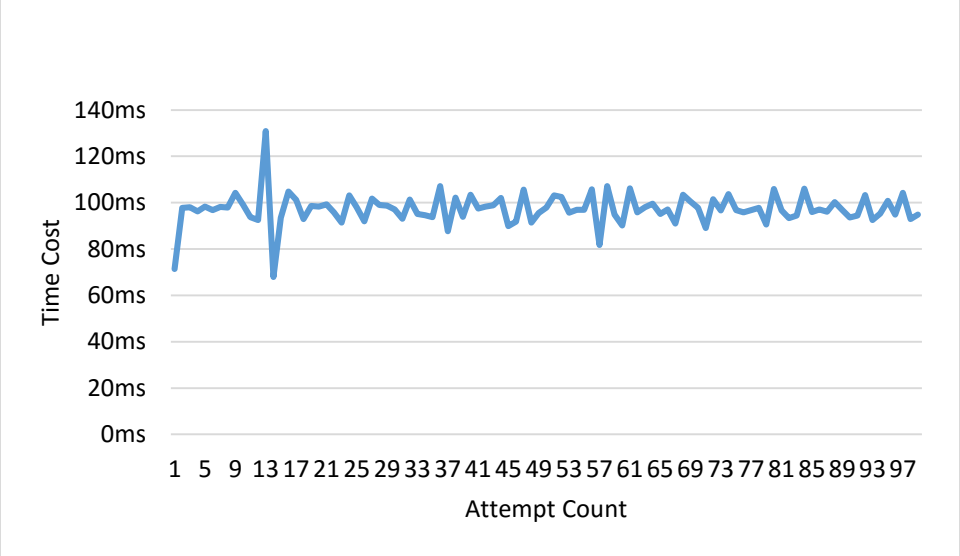

Figure 6.9 Second set of 4-bytes round trip

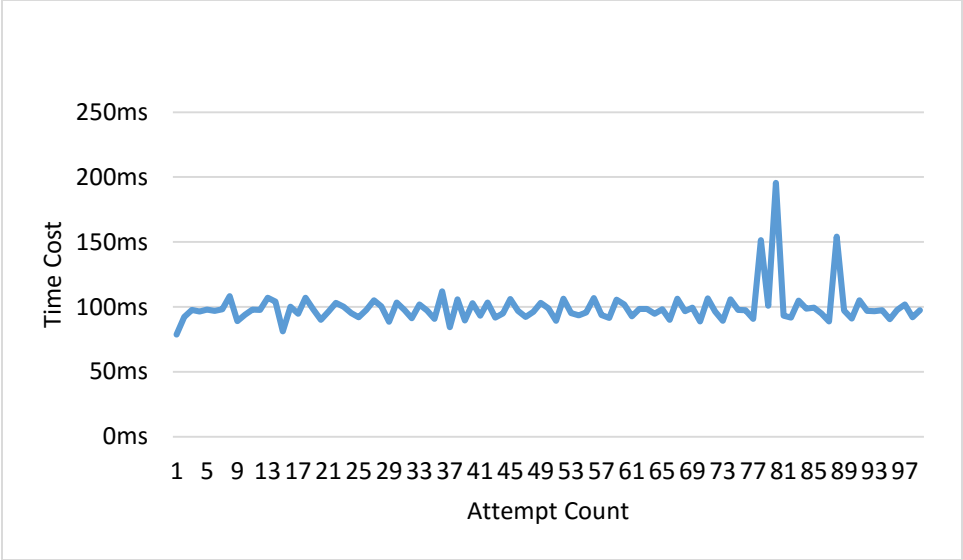

Figure 6.10 Third set of 4-bytes round trip

In the first experiment, we examined one-way time consumption of 4-bytes header CoAP message. The data we get from that experiment shows a pattern obviously. However, according to the charts above, we can not find any pattern for the round trip experiment. Compared those two experiments, I find that the curve fluctuates greatly around 100ms. Meanwhile, some great fluctuation occurs from time to time. The greater fluctuation of the curve may do to channel interference caused by complex reasons. On the other hand, I believe the random fluctuation here is caused by Android Lollipop's BLE implementation.

According to the round trip experiment and the minimum payload experiment, we can come to a conclusion that the default communication interval of BLE in Android should be 100ms. On the other hand, the average time consumption of one-way communication and round trip communication are close in the proposed architecture.

6.3.4 Multiple Apps

Description

The experiment aims to test the performance of underlying network service. In this experiment, two applications on the same device try to send 100 CoAP headers to remote device at the same time. The sender side will record total transfer time to calculate average time.

Procedure

1. Start both the “Major APP” and the “Trigger App”.
2. Click start test button at either app to start test.
3. Record start time and end time at sender side.
4. Repeat 2-3 steps 10 times.

Result and Analysis

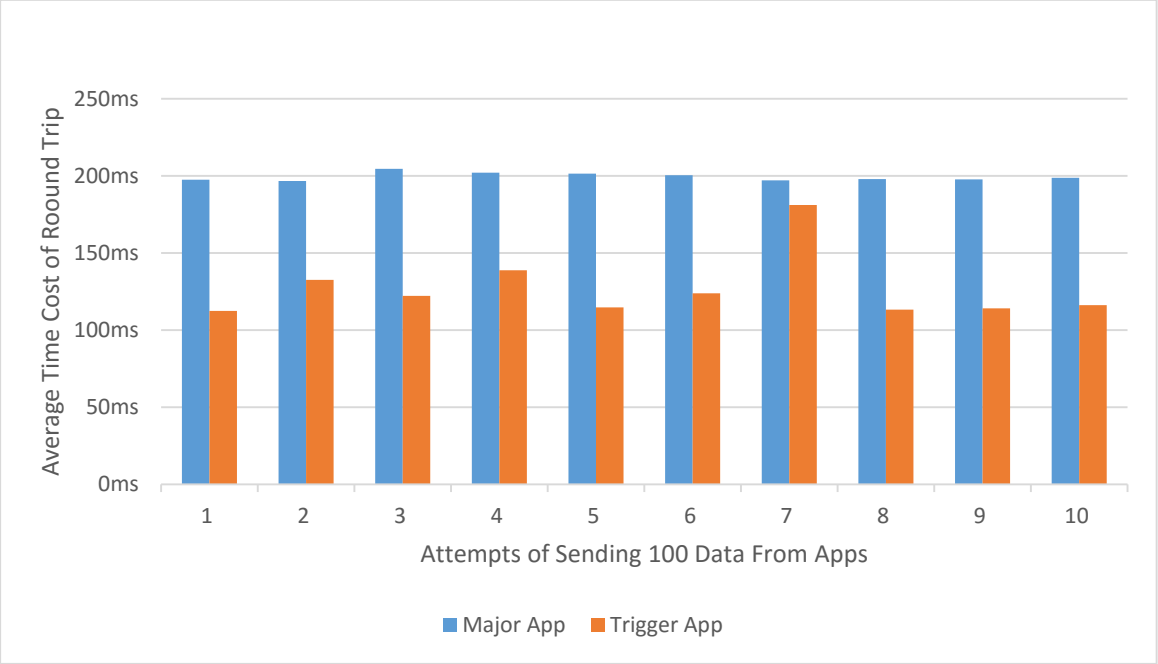

Figure 6.11 Result of Multi-App Experiment

As mentioned above, there are two apps use underlying service for CoAP communication. The Major app has a listener listening to the event of start test from the "Trigger App". After 10 times repeat, there are 10 sets of data available. As shown above, the average time cost of round trip in the "Major App" is always higher than the "Trigger App". Meanwhile, values of the "Trigger App" are fluctuant. This result due to the competition of one communication channel. Before the "Major App" gets broadcast, the "Trigger App" can push request into send thread without interrupt. Therefore, the trigger always can finish task earlier.

6.4 Conclusion

Based on the analysis of experiment results, we enhanced understanding of implementations of Android's BLE. Meanwhile, we obtained more data about the performance of proposed architecture. In previous experiments, we tested our architecture's performance in lightweight and medium weight payload communication. According to above analysis we get the following conclusion:

1. The minimum transfer time of proposed architecture is around 100ms. In single communication channel (one characteristic is used to transfer data), we have 16 bytes available to transfer CoAP message (more details in pocket design subchapter). Therefore, in current architecture, we have 10*16 byte/s data rate.
2. "Connection interval" is a parameter in BLE. It determines how often a central will ask data from peripheral. Since the data is specified by device's implementation and can be any value between 7.5ms and 4s. It is an important factor to affect data transfer. It is most likely to be the reason of constant fluctuation when we send data with different latency.
3. There is a considerable amount of bad performance reports about Android lollipop's implementation of BLE. The Android V5.0 (Lollipop) is the first Android version to support peripheral mode in Android device. It is not surprising to get bad performance from the new feature. We expect the experiment result can be improved on new Android device with latest API and BLE hardware.
4. Because a single message queue and send thread are used to send data in underlying architecture, the competition of send messages will influence the send time of a message.

CHAPTER 7 SUMMARY AND CONTRIBUTION

With the development of IoT, more and more small sensors are involved in data collecting. However, different protocols and hardware limitations make communications between devices becomes more and more difficult. To overcome those limitations, the research proposed architecture to grant CoAP in BLE and the potential to support other WPAN technologies.

In terms of design and architecture, the proposed architecture consists of two layers. The application layer provides an interface to send and receive CoAP message as well as message management. The network layer implements underlying logics of assembling and disassemble CoAP message on the top of Bluetooth Low Energy's GATT service. The architecture uses two parameters (user id and app id) to identify different communication instance. In order make the architecture works as a background service to serve one or more Apps in one physical device, special packet format and communication principles are designed for the proposed architecture.

In terms of implementation, the I adopt Mono to develop the proposed architecture. Mono is a software platform developed by Xamarin for cross-platform application development in C#. It is adopted to develop the proposed architecture. There are two stages in the implementation. In the first stage, I developed the underlying network layer for Bluetooth Low Energy for Android and involved in the definition of the application layer. In the second stage, developed two test program for experiments. They are major app and trigger app. The trigger app only used in the last experiment to test the performance of background running service when it serves multiple apps.

In order to test the performance of the proposed architecture, I proposed four experiments. Those experiments revealed cost and limitations of the proposed architecture on the top of BLE's GATT service. The first experiment shows when sending a message with interval messages may have consistent fluctuation. The second experiment shows the latency of sending a message

increase linearly with the increasing size of BLE packets. The third experiment shows there is no significate of time cost difference between send one-way and round trip messages in the proposed architecture. Meanwhile, there are random fluctuations. The fourth experiment shows competitions happens when multiple apps try to send messages at the same time.

In conclusion, the proposed architecture works on BLE. Although, there are some limitations and restrictions it achieved to send CoAP message in BLE and serve multiple APPs as a background running service.

CHAPTER 8 FUTURE WORKS

To further improve the performance of the proposed architecture. The architecture is expected to be improved in the following aspects.

8.1 Data Rate

As shown in experiment, the average time to send out a CoAP header cost 100ms in the current implementation.

8.2 Availability and Partition Tolerance

In order to achieve maximum availability and partition tolerance, the current implementation lost connection event triggers a reconnect action. However, the auto-reconnect is not enough. In the further development, I expect the system can listen to nearby paired devices and publish a connection whenever they are close enough.

On the other hand, data cache mode is also expected to be implemented to guarantee data availability.

8.3 Cross-Platform and Underlying Technology

During implementation, I choose to implement the proposed architecture in Android platform where a developer has more control over the Bluetooth adapter and background service. Other platforms like iOS and Windows phone should be explored.

As mentioned above, the proposed architecture is designed to have the potential to support multiple underlying communication technologies. In the current implementation, it only supports BLE as its underlying technology. However, many other smart devices are not supported BLE. For example, NFC and Classic Bluetooth. In further research, I will explore the possibility to implement those technologies as network layer of proposed architecture.

8.4 Security

Currently, all communication by the implementation is not encrypted. In further research, the proposed architecture can improve its security by implement symmetric algorithm or asymmetric algorithm.

CHAPTER 9 REFERENCE

Arcitura. (2016, April 14). *REST Constraints*. Retrieved April 14, 2016, from Arcitura.com: http://whatisrest.com/rest_constraints/index

Barrett, J. (2012, October 9). *The Internet of Things*. Retrieved April 13, 2016, from tedxtalks.ted.com: http://tedxtalks.ted.com/video/The-Internet-of-Things-Dr-John

Evans, D. (2011, April). *The Internet of Things - How the Next Evolution of the Internet.* Retrieved April 13, 2016, from http://www.cisco.com: http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Irvine: UNIVERSITY OF CALIFORNIA.

Flanders.J. (2009, July). *Service Station - More On REST*. Retrieved April 22, 2016, from msdn.microsoft.com: https://msdn.microsoft.com/en-us/magazine/dd942839.aspx#id0070007

Fowler, M. (2010, March 18). *Richardson Maturity Model: steps toward the glory of REST*. Retrieved April 16, 2016, from martinfowler.com: http://martinfowler.com/articles/richardsonMaturityModel.html

Gilbert, S., Lynch, N. (2002, June). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News, 33*(2), pp. 51-59.

iOS. (2014, June 2). *Getting Started with iBeacon*. Retrieved April 18, 2016, from developer.apple.com: https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf

Isomaki, M., Nieminen, J., Gomez, C., Shelby, Z., Savolainen, T., & Patil, B. (2011, April 19). *Transmission of IPv6 Packets over BLUETOOTH Low Energy*. Retrieved April 12, 2016, from tools.ietf.org: https://tools.ietf.org/html/draft-ietf-6lowpan-btle-00

Mitra, N., & Lafon, Y. (Eds.). (2007, April 27). *SOAP Version 1.2 Part 0: Primer (Second Edition).* Retrieved Apr 26, 2016, from www.w3.org: https://www.w3.org/TR/soap12-part0/

Na,S.,Park,J.&Huh,E. (2010). Personal Cloud Computing Security Framework. *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific* (pp. 671 - 675). Hangzhou: IEEE.

OASIS Technical Committee. (2014, 0ctober 29). *MQTT Version 3.1.1*. (A. Banks , & R. Gupta, Editors) Retrieved April 17, 2016, from docs.oasis-open.org: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

Pautasso, C., Zimmermann, O., & Leymann, F. (2008.April). Restful web services vs. big'web services: making the right architectural decision. *Proceedings of the 17th international conference on World Wide Web* (pp. 805-814). New York: ACM.

Seagate. (2016). *What Is a Personal Cloud?* Retrieved April 13, 2016, from www.seagate.com: http://www.seagate.com/ca/en/do-more/what-is-personal-cloud-master-dm/

Shelby, Z. (2014, April 30). *ARM CoAP Tutorial*. Retrieved April 17, 2016, from slideshare.net: http://www.slideshare.net/zdshelby/coap-tutorial

Shelby, Z., Hartke, K., & Bormann, C. (2014, June). *The Constrained Application Protocol*. Retrieved April 17, 2016, from tools.ietf.org: https://tools.ietf.org/html/draft-ietf-core-coap-18

SIG. (2010, June 30). *BLUETOOTH SPECIFICATION Version 4.0.* Retrieved April 13, 2016, from www.bluetooth.com: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=229737

SIG. (2013, December 4). *Updated Bluetooth® 4.1 Extends the Foundation of Bluetooth Technology for the Internet of Things*. (Bluetooth SIG) Retrieved April 17, 2016, from bluetooth.com: https://www.bluetooth.com/news/pressreleases/2013/12/04/updated-bluetooth4-1-extends-the-foundation-of-bluetooth-technology-for-the-internet-of-things

SIG. (2014, December). *BLUETOOTH ® CORE SPECIFICATION 4.2 FREQUENTLY ASKED QUESTIONS*. Retrieved April 18, 2016, from bluetooth.org: https://www.bluetooth.org/ja-jp/Documents/Bluetooth4-2FAQ.pdf),

Torvmark, K. H. (2013, January 29). *Three flavors of Bluetooth: Which one to choose?* Retrieved April 17, 2016, from edn.com: http://www.edn.com/Home/PrintView?contentItemId=4405960