CoAPNonIP- An Architecture Support Non-IP based protocol in IoT

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

Abstract

Acknowledgements

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

In boom of smart phone market and the development of wearable devices are introducing low energy sensor and personal hub (phone, tablet or other portable devices) with increasing rate. In this trend, we found that the role of smart phone has shift 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. Therefore, the communication between personal hubs and edge detectors becomes important.

For example, at the backyard of a house, there are sensors which can get humidity data. Those sensors are smart devices where Wi-Fi communication is not supported and data can only be providing in a short range. In this case, we can design a small app for phone. When user is near the field. the phone can automatically connect history data of humidity from multiple sensors and store those data on device or deliver it to remote server.

There are some popular technologies to support short range wireless data communication like Bluetooth Low Energy, NFC and etc. However, another problem is how can we unify our language when talking with different technologies. Although, technologies like MQTT and COAP have provided solutions based on TCP and UDP respectively. However, IP address is a must for both of them. Therefore, how to consume resources of from different message channel like Bluetooth and NFC becomes a problem. In 2011, Johanna ** proposed a IPV6 based solution to connect BLE devices to the internet. In their solution, compressed IPV6 must be embed in order to support COAP protocol. In addition, the strategy of transmit IPv6 packets over NFC is also available on 6lo working group. However, at present those IPv6 solutions are not widely adopted. It may because the price of adopt IP based communication is too high for small sensor.

With the blooming of wearable device market, more and more BLE devices are involved in our daily life. New rising wireless communication technologies like BLE have strong requirement to involving in existing IP based IoT networks. However, it is hard to merge those Non-IP based technologies into existing networks because of their own characteristics. In this research we will take BLE as an representative to show how to merge lightweight Non-IP based wireless communication technologies into IoT context.

In this research we will take BLE as an representative to show how to merge lightweight Non-IP based wireless communication technologies into IoT context.

CHAPTER 2 PROBLEM DEFINITION

2.1 How to merge Non-IP based device into IoT?

The concept of IoT are firmly associate with IPv6. It tagging 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 use MAC address to identify devices. In order to support communications between different tagged, following two things are must. 1. A identifier beyond the concepts like IP address or MAC address. 2. A protocol which is independent from IP address or MAC address.

2.2 How to overcome limitations of standard BLE communication?

Since BLE is designed for low intensity data transfer, some limitations need to overcome for 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 must, in which the payload of a characteristic is limited. According to Bluetooth4.0 specification**. "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". For example, according to experiment, the max size of payload of a write characteristic request 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 it 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 communicate with multiple clients. Unlike classic Bluetooth where each connected client can create its socket to maintain communication, 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?

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 wireless communication protocols

Beside BLE, there are other Non-IP based wireless communication protocols have the requirement to take advantages of IoT protocol. If a higher level interface is proposed, we can further support those protocols.

2.5 Research Goal

The goal of the research is to propose a complete solution for merge Non-IP based wireless communication protocols like BLE into existing IP based IoT networks. The main goal consists of following three sub goals.

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

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

Goal 3. Provide strong accessibility in BLE.

CHAPTER 3 LITERATURE REVIEW

3.1 Cloud Computing

The glooming of user-centric cloud services is not a coincidence. It emerges under the condition of mature cloud computing infrastructure and the revolution of mobile phone market.
Cloud computing is a new technology, but it has a close relation with grid computing. Ian et al. **analyzed features of cloud computing and grid computing and concluded that even though grid computing backbones cloud computing as inner infrastructure. Cloud computing has many exclusive features. According to their arguments, the cloud computing is wrapped by web 2.0 technology and aims to provide public services. They also believe that the majority of differences between grid computing and cloud computing are mainly in security and model. Rajkumar et al. ** studied the relationship between cluster computing, grid computing and cloud computing. It implicates that the cloud computing is the most open scenario, which is shaping the world by providing computing abilities in a convenient way. In 2010, Michael et al. ** measured the cloud computing in details and proposed top 10 obstacles and opportunities of clouding, which shows a big picture to readers. Interestingly, both papers mentioned about the "pay as you go" business model and the bottleneck of data transfer, which due to the core value of cloud computing (provide utility computing services).
Generally, people prefer to classify cloud services into three categories, which are IaaS (infrastructure as a service), PaaS (platform as a service) and SaaS (software as a service).

- IaaS is the base infrastructure of PaaS and IaaS. It allows users to buy computing abilities and storages from service providers. For example, Amazon EC2 provides three ways to pay for purchasing services (On-Demand Instances, Reserved Instances and Spot Instances).
- PaaS is application-oriented service. It provides APIs and running environment to users who only need to upload applications to service provide before everything is deployed automatically. For example, Google app engine provides a complete solution for cloud side services.

- SaaS typically is a special kind of application, which is deployed and implemented at cloud side. User can access to it by using any kinds of clients with web browsers.

In practice, most cloud computing providers do not provide their computing ability in a single way. As it has more customize rooms than SaaS and less setting works than IaaS, the PaaS has gained attention of giant companies. Besides Google, both Amazon and Microsoft have released their PaaSs (AWS Elastic Beanstalk and Azure).
In summary, the basic infrastructure of cloud computing services becomes more and more mature. Anyone who purchases computing ability from IaaS or PaaS can play the role of third party provider. Therefore, an increasing number of middle and small business owners implement their applications on cloud computing platforms. This trend is linking the world and slashing the cost of business. On the other hand, since more and more data are exposed. The protection of privacy has been widely concerned. In order to deliver personal data safely, a user-centric concept (Personal Cloud) has been proposed.

3.2 IoT

In recent years a new concept named "IoT" is popular. IoT (internet of things) is a concept of real world embedded in electronic world. It is happening and will fundamentally change our world. In order to make real object become a "thing" in internet. There are several essential steps to convert an ordinary object in daily life into a smart object in IoT. (**reference TED speech)

First, a unique identifier to tag a thing. Every asset in the IoT have 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 big limitation of its capacity. With 32-bits space, IPv4 has 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 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 object or called things in IoT must implement a way to communicate. According to different transmission medium, we have to implement different mechanism to communicate. Today, we mainly use microwave and twisted-pair as the front end medium.

Third, give object senses. In real world, an object can be identified by smell, feeling, color and so on. It is also true in digital world. by collect different data of an object, people can know changes of an object or its surrounding environment. Therefore, we need implement different kinds of sensors to an object to make it become "smart". By implement sensors, we get 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 it to a remote object through internet, the remote side may have further requirement to modify some values of a smart object through Internet. Therefore, we need remote controller to support modify operations.
After above four procedures, we can make sure an object becomes smart. As an smart object, it can involve in data communication of IoT.

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

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

3.3 Personal Cloud

Personal cloud is a new term under the concept of private cloud. With the increasing number of personal devices, more and more people have their own personal cloud. In fact, there are some business model available for years. For example, Seagate proposed a personal cloud solution (**

reference Seagate definition of personal cloud) which aims to provide a centralised media library where user can access to their data from anywhere. The model is consisting of software on different platforms and a central server. User can access their digital assets on different platforms through different software interface but their data are backed up in one physical device under the control of user.

In another aspect, with the increasing number of personal mobile devices (like smart phone, pad, wearable devices), more and more digital assets are distributed stored in different devices. For most of the time, those devices are not all in the radiation of a Wi-Fi network. In this case, a centralized server is not an elegant solution for a mobile cloud which are connected by short range wireless communication technologies(SRWMC). The following are two important facts of a SRWMC.

First, the data communication between mobile devices are relatively less intensive than pc to pc communication. It is decided by the computing power of mobile devices.

Second, the network state is constantly change from time to time. It is decided by the nature of short range wireless communication.

Taking the above two into considerable, a REST approaches, seems gain more momentum for distributed personal cloud solution. due to it’s lightweight and close relationship with web inspired protocols.

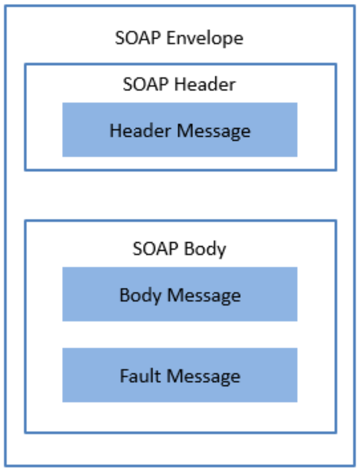
In the next section, we will discuss advantages and disadvantages of REST by comparing it with SOAP.

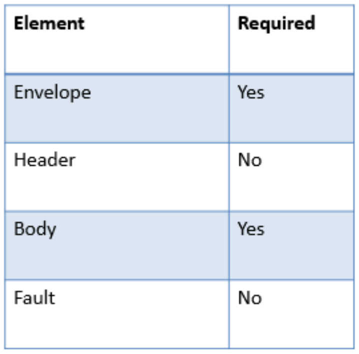
3.4 SOAP and REST

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

3.4.1 SOAP

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





Element	Required
Envelope	Yes
Header	No
Body	Yes
Fault	No

As shown above, there are four types of element in SOAP. In Envelope element, user defines xmlns:soap namespace and soap:encodingStyle attribute. Header is an optional element where defines how to process SOAP message. mustUnderstand, actor and encodingStyle attributes are available. SOAP Body element contains actual messages. Meanwhile, it also may have a child element named as Fault where error message can be handled.

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.

3.4.2 REST

REST stands for representational state transfer. It is a popular style in design web application. In REST, each available resource at server side can be visited through a consistent path (URI). 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 whatisrest.com (**), 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. At client side, developer need to concern how to maintain user interface and state. On the other hand, server need to concern about how to store data.
- Stateless: Stateless is the most important feature of RESTful design. It indicates each request from client contains all information a server need to know and all session state data should send to client after each request.
- Cache: A RESTful design should support both cacheable and un-cacheable mode. In cacheable mode user can save latest response from server for further usage. In un-cacheable mode, every request from client will send to server.
- Uniform Interface: It is fundamental characteristic of a REST service which has guaranteed each request can independently get an individual resource.
- Layered system: A REST-based solution may consist of multiple layers. Communication between service provider and consumer are independent events which can not be affected by changes of other layers.
- Code-On-Demand: It is an optional constraint. client can update its code indecently from server.

RESTful design makes web application back to the original purpose of HTTP.
According to Martin Fowler (**), there are three steps toward glory of REST (from traditional HTTP web service to a RESTful service).

- Level 1. Resource: Developer 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 misuse them.
- Level 3. Hypermedia Controls: Grant discoverability to response. By adopting hyperlinks in response. Web server can show user what resource is available as his or her next operating object.

3.4.3 REST VS SOAP

It is common to see REST and SOAP are put together to make comparison. Theoretically, SOAP is a protocol and should not be compared with REST. However, those two terms defined two popular ways to design web service. In SOAP, all request and response must follow XML standard in the body.

One highlight characteristic of SOAP is POST is the only one verb used in request. Meanwhile, things in REST are straightforward because operations are limited within four verbs and each URL point to one resource. The following section will discuss those two technology from different aspects.

- Complexity: Protocol: In SOAP, it adopts XML as transfer format which means the complexity of serialize and de-serialize is certain. In contrast, according to difference scenario, REST can adopt different protocol to transfer data. Although JSON is the most popular one but it is not the only option. In general, JSON is more concise than XML which means it can represent same amount of information with less words. In other words, to say, it consumes less bandwidth than XML. On the other hand, In SOAP one URL may consists of a bunch of business logic and operations are all customized in a POST request which makes it become a black box for user. In contract, operations in REST are limited by verbs of HTTP, which are simple and efficient. Since one resource mapping one URL in REST, user can easily know target and operation.

- Scalability: The popularity of SOAP was back to the time when IT service are only available for giant company. It designed to fulfill a particular task which makes codes are hard to reuse and hard to change logics of an existing web service. On the other hand, REST mapping one URL to one resource where complex logic is breakdown into multiple simple operations. The structure naturally increased the scalability of a system.

- Cache: REST is naturally suit catching. As mentioned above, the last step towards REST is Hypermedia Controls where all possible next step is provided by hyperlink. In this way, the client side can easily be preloaded next step before user actually execute an operation. On the other hand, some resources like images and text are mapping into a URL. Those resources are easy to cache. In SOAP, we always see things are wrapped together and return to user where resources are not designed can be visited at anytime. This characteristic makes cache becomes much harder in SOAP.

- Security: Because of WS-Security, many people believe SOAP is more secure than REST. However, WS-Security is only a solution of SOAP's transport independency. In REST, HTTP or HTTPS are underlying transport layer protocol where build-in security solutions are available. Therefore, we can not say which protocol is better in terms of security.

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 more characteristics need to be compared between those two technology. After elaborate effort, Cesare come to a conclusion (**) (https://scholar.google.com/scholar?q=related:vCTjpGkWG0AJ:scholar.google.com/&hl=en&as_sdt=0,5):" REST scores better with respect to flexibility and control, but re-quires a lot of low-level coding; WS-* provides better tool support
and programming interface convenience, but introduces a depen-
dency on vendors and open source projects."

3.4.4 REST VS SOAP in mobile app

In recent years the emerging of mobile applications in smart device required more and more web services to support client from web and smart device at the same time. Although the REST design style is proposed with HTTP, we did not use it much in old system. Back to late 1990s and early 2000s the web server was expensive for a single user or a small business owner. At that time SOAP are popular. Not only because it is good at handling complex logic in one entrance, but also it is simple to implement. In recent years, more and more web services need to be short time connection with less data communication thus it become popular.

For 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 mobile app which need network connection through wireless communication channel (common ways for mobile phone connect to Internet are Data Plan and Wi-Fi where connection are fragile). In light weight mobile app REST should weight more. On the other hand, SOAP is not a bad option where mobile apps are used to serve web application for complex business logic (e.g. bank application and tax return application with existing backbone of SOAP services).

3.4.5 Conclusion

REST and SOAP are two popular design patterns for web service. From the analysis above, it is clear that REST is simple and clear. In REST, each URI has clear meaning and actions defines operations associate with a URI in a request. On the other hand, in regular, there are more logic in a single SOAP page. The XML makes the SOAP can handle complex logic but also increase the cost of response a request.

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

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

3.5 MQTT and CoAP

MQTT and CoAP are two popular protocol in IoT. The proposed architecture is designed for CoAP. In this section we put MQTT and CoAP together to discuss features of CoAP.

3.5.1 MQTT

MQTT stands for MQ Telemetry Transport. it is a lightweight publish-subscribe protocol running on TCP/IP protocol. Because of lightweight, it suits M2M (Machine to Machine), WSN (Wireless Sensor Networks) and IoT.

3.5.1.1 Packet Format

According to 3.1.1 specification, MQTT packet has two bytes fixed header. Therefore, the minimum size of a MQTT packet is two bytes. The following chart shows how a MQTT packet looks like.

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

As shown above, the first byte of MQTT contains two parts. From bits 7-4, it indicates the type of packet. Sixteen options are available in this section. From bits 3-0, it sets flags for different message type. Start from second byte, the packet will indicate the remaining length of the message, the minimum size of it is one byte, the maximum size of it is four bytes. After the specify remaining size of a packet, user can add optional headers to a packet. For example, 2 bytes packet identifier can be added in the optional header. After optional header, the remaining section of a packet is payload.

3.5.1.2 Feature

Publisher & subscriber(pub/sub): In this model, publisher and subscriber do not know about the existence of one another. Instead, a role called broker will gather all published messages from publisher and accordingly send them to subscribers after filter. In brief, server plays role of broker and client plays role of publisher or subscriber.

QoS: MQTT support three levels of QoS. At level 0, a message will not get acknowledged by receiver or resend by sender. At level 1, a message will be guaranteed to be delivered at least once. Therefore, a message will might be received twice by subscriber until an acknowledgement is received by sender. At level 2, a message will be guaranteed to be processed by receiver. In this case, receiver will store message and ready to resend of it have not receive PUBREC before timeout. Meanwhile, at receiver side it will store a reference of received packet's id to prevent processor from processing same message twice.

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

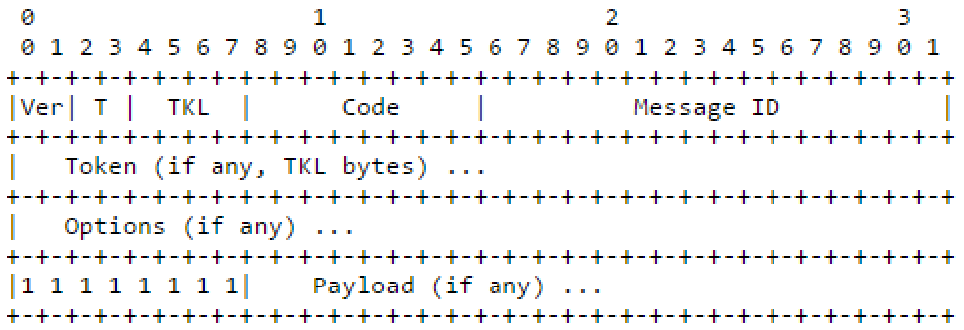
In conclusion, the MQTT is a well designed protocol for lightweight bandwidth, the logic of it is simple. And all communication of client is based on message delivery from broker.

3.5.2 CoAP

CoAP is short for Constrained protocol. It is based on REST model and designed for constrained networks. According to stable reference RFC 7252, UDP is the default transport protocol of it. Meanwhile, it could also be used over other transports such as TCP. As a HTTP like protocol, the raise of CoAP offers an elegant solution for low bandwidth communication.

3.5.2.1 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.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: Message Format

(**https://tools.ietf.org/html/rfc7252)

As shown above, the header of a CoAP message consists of five type message.

The first 2 bits of a message shows CoAP version of the message.

The following 2 bits indicates type of the message (4 available values).
TKL stands for Token Length (4 bits), it indicates the length of token field.
The second byte only contains one element (Code) which indicates operation of the message. There are two types of code. One is called method codes which consists of 4 verbs in HTTP (GET, POST, PUT and DELETE). The other is response code, it represents state of a response. For example, CREATED, DELETED, VALID and etc.
The third byte and fourth byte is assigned to Message ID. a message id is unique for a piece of message.
Besides the four-bytes header, token and options are additional information which are neither part of header or payload. The size of token ranges from to 0 to 8 bytes. it is used to match request and response. CoAP also support some metadata in HTTP. In the Options section, user can define Etag, Max-Age, Content-Format and etc. Those options make it is easy to translate between CoAP message and HTTP message.
Unlike MQTT where a remaining length is defined to specify the length of optional header and payload size. In CoAP, 1-byte flag is used to indicate the beginning of payload. When a CoAP message first time get a 0xFF, a processor knows the payload start from the next byte.

3.5.2.2 Feature

There are several highlight features for CoAP.

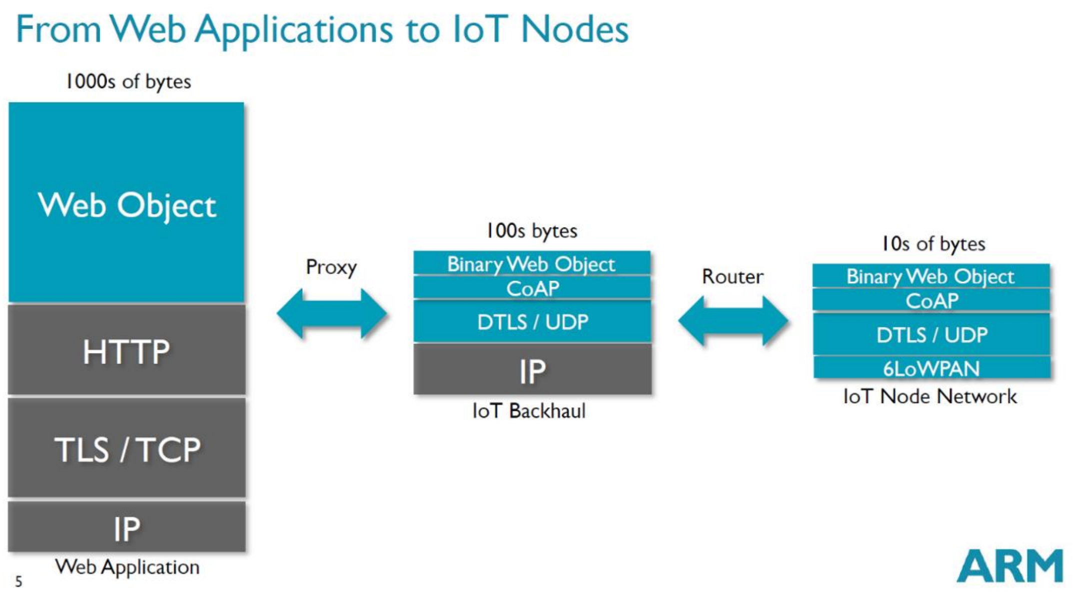
- URI support: CoAP follows a URI scheme which is similar as the one in http.The format of CoAP looks like this: coap-URI ="coaps://" host [":" port] path-abempty ["?" query].

- Similar features to HTTP: Operations in CoAP are based on 4 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 requesting this address, client can get all available resources of a server. The response format of "/.well-known/core" has been well designed.

- Observation: CoAP support data observation as an extra option besides traditional send and receive. At Server side, developer can add observable resource to observer list. Then, client can register resource observer, after which clients can 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 network under the consideration of collaborating with exists network. As shown in the following chart, we expect the CoAP handle medium or small size messages to become a bridge between fat web services and low payload networks.



The chart above also shows the current status of IoT network in terms of payload size.
For web applications where payload size relatively high is the backbone of IoT network. The majority of web applications adopt HTTP to transfer data where IP cluster are underlying protocols. In general, the payload size should beyond 1000 bytes since the size of TCP is 65,535bytes and 65,507bytes for UDP.

Between giant network and IoT nodes, there may have routers which will act as an interpreter between CoAP and HTTP. As shown above, attributes in CoAP and HTTP are similar, which minimized works in 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, the far end nodes of a IoT networks should consists of many tiny devices with low bandwidth network. For example, sensors in a room, iBeacon in mall, and remote start component in a car.

In conclusion, CoAP provides a solution to merge low payload network into the existing HTTP based network. Apparently, it will accelerate the communication between tiny personal network and cloud side service providers, which will convenience data access.

3.5.3 COAP VS MQTT

Although both CoAP and MQTT can be used as IoT protocols, they are 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 at multiple communications between clients. Meanwhile, with smaller header MQTT is more efficient than CoAP when carry small amount of data. On the other hand, since it is based on subscribe and data push model. It expects tiny information in payload. In MQTT, 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 quality communication.

CoAP is designed for RESTful architecture where more metadata are supported. Therefore, a CoAP message can easily been transferred into a HTTP message. One the other hand, the minimum size of a CoAP message is 4-bytes which makes it can also have been 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 in a packet, which means developer need to implement their own strategy for quality service. It also true in terms of cache, 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 solution.

In conclusion, the MQTT aims to multiple clients' communication with a central broker. The CoAP focuses on one to one REST communication. The CoAP is more flexible and more close to HTTP protocol. CoAP is more friendly to developers where less effort is required and more flexibility are guaranteed.

3.5.4 Conclusion

We compared two most popular technologies in IoT According to analysis above we can come to the conclusion that CoAP is more close to Internet than MQTT. The MQTT has some nice features to guarantee communication quality but it is designed for small device only. Therefore, if we 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 merge NonIP based devices into IoT. In this case the cost of language transfer should weight more than others. In this case, CoAP suit more to solve the problem.

In the next chapter, the paper will discuss strength and weakness of Bluetooth and Bluetooth Low Energy.

3.6 Bluetooth

Bluetooth is a technology 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 need frequently send small amount of data. It is invented by telecom vendor Ericsson in 1994. Originally is was designed as an alternative technology of RS-232 (a serial port standard).

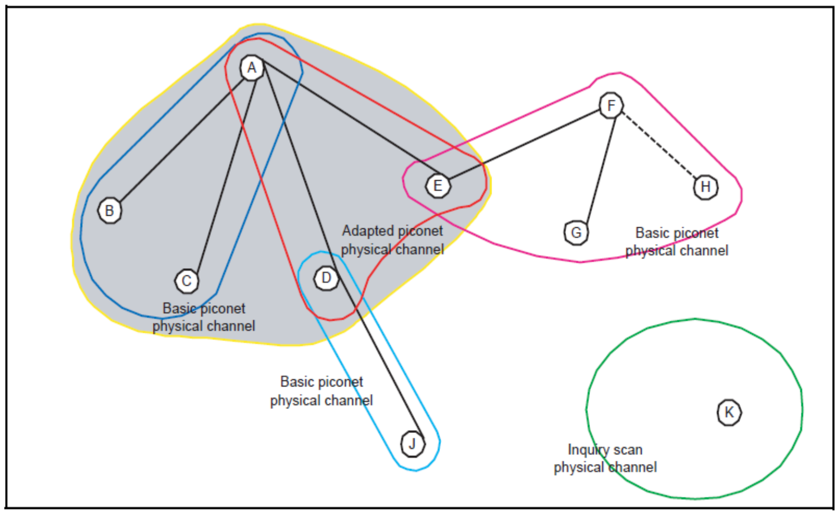
So far, Bluetooth has four types. They are Bluetooth, Bluetooth EDR, Bluetooth HS and Bluetooth low energy. This paper mainly focuses on Bluetooth low energy which is available since 2010.

Bluetooth focus on short range and low payload wireless communication. Since Wi-Fi takes great advantages in the field of high speed data transfer, Bluetooth HS has been proved a failed product. As a consequence, SIG focus more on lightweight and low energy communication. In 4.0 version or Bluetooth, it proposed "Bluetooth Low Energy" to take up the market of low energy wireless communication. With the merging of wearable devices. The BLE has been widely adopted by sensor producers. In the remaining section of the chapter, we will discuss the differences between class Bluetooth and Bluetooth low energy. Further, we will explore merging protocols to figure out how to taking advantages of Bluetooth low energy.

3.6.1 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) and one master can have up to seven slaves. Communication through Classic Bluetooth is based on socket which means user can send any size of package through the channel. In this way, the Classic Bluetooth is more flexible but less energy efficiency.

As shown in the following figure (a reference from Bluetooth4.0 specification), 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 never 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 slave. The device E play a role of slave in both the piconet of A and F. The K is an isolated advertising node.

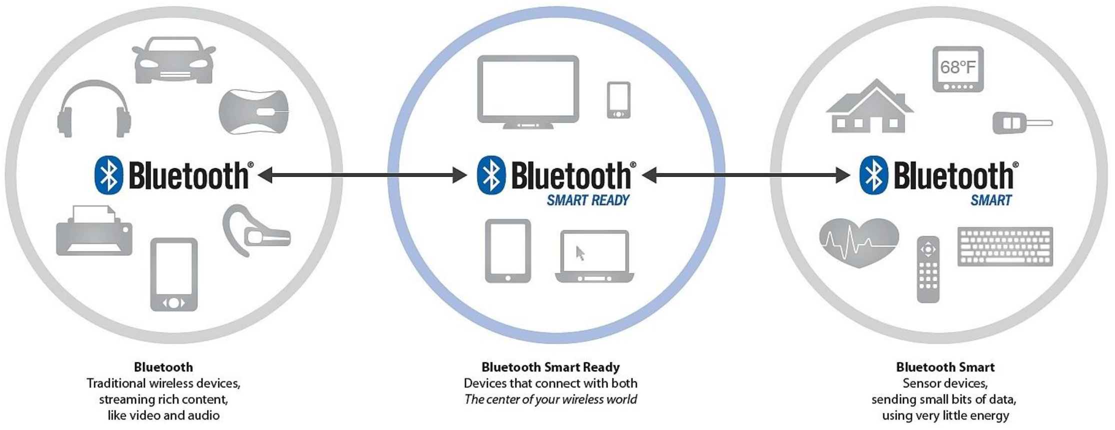

A
B
C
D
E
F
G
H
J
K
Adapted piconet physical channel
Basic piconet physical channel
Basic piconet physical channel
Basic piconet physical channel
Inquiry scan physical channel

As explained above, the classic Bluetooth has widely been adopted in wireless communication. However, there is a big disadvantage of it. In order to increase data rate, the class Bluetooth will send streaming data in channel. 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 gradually made it overlap the functionality of Bluetooth. After the attempt of increasing data rate in version 3.0. The SIG turn to develop low energy version of Bluetooth. In the following section we will discuss Blueooth4.0 (Introduced BLE).

In conclusion, the class Bluetooth is a success product. However, it can not make 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.

3.6.2 Bluetooth Low Energy

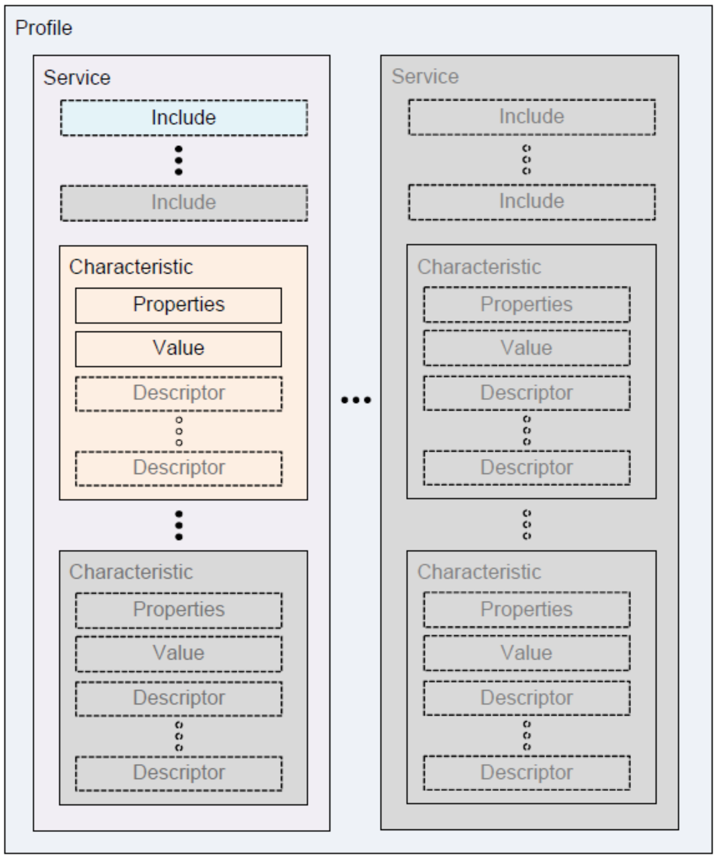
The concept of Bluetooth smart in introduced since Bluetooth 4.0. The highlight of this version is the adoption of Bluetooth low energy.


Bluetooth
68°F
Bluetooth SMART READY
Bluetooth SMART
Bluetooth
Traditional wireless devices, streaming rich content, like video and audio
Bluetooth Smart Ready
Devices that connect with both
The center of your wireless world
Bluetooth Smart
Sensor devices, sending small bits of data, using very little energy

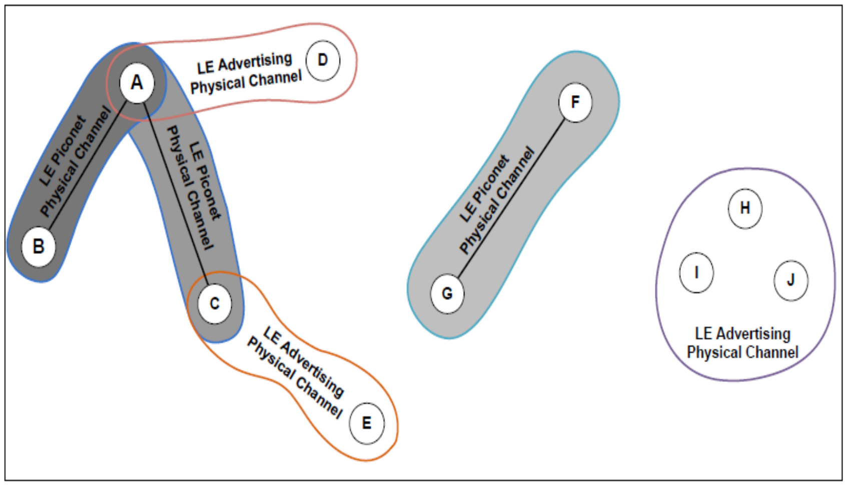
As shown above, so far, there are three kinds of Bluetooth devices ("Bluetooth", "Bluetooth Smart Ready" and "Bluetooth Smart"). Devices with "Bluetooth" logo means it is only support Classic Bluetooth connection. If a Bluetooth device can both communicate with Classic Bluetooth device and communicate with BLE device, it is named as " Bluetooth Smart Ready ". For those latest devices which only support BLE based communication 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", which is a socket based communication. Communication between BLE devices are based on GATT (Generic Attribute Profile) and ATT (Attribute Protocol). According to Bluetooth4.0 specification: "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 property, value and descriptor. The BLE client can read, write or monitor value changes of a characteristic.



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


A
B
C
D
E
F
G
H
I
J
LE Piconet Physical Channel
LE Piconet Physical Channel
LE Piconet Physical Channel
LE Advertising Physical Channel
LE Advertising Physical Channel
LE Advertising Physical Channel

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:

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

According to Suke Jawanda who is chief marketing officer of Bluetooth SIG: "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 (**https://www.**bluetooth**.org/ja-jp/Documents/**Bluetooth**4-2FAQ.pdf), the latest version has improved BLE in three aspects: IoT capability, security and speed.
In terms of IoT capability, now, BLE device can directly participate in IoT network by adopting 6LowPAN and connect with router supporting Bluetooth Smart Internet Gateway.
In terms of 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.
In terms of 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.
With the development of BLE there are more and more new protocols are defined to support BLE. In the following section, we will discuss the iBeacon and 6LowPAN which have fundamentally change the use of BLE.
iBeacon. iBeacon is a protocol proposed by Apple. According to BLE's advertising standard, user can set up to 20-bytes payload. Apple proposed a protocol to format advertising data. 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 seen by BLE smart ready devices in scanning status.

6LowPAN. 6LowPAN is short for IPv6 over Low Power Wireless Personal Area Networks. So far, the latest version of BLE (Ver. 4.2) has proposed IPSP (Internet Protocol Support Profile) to support IPv6 based communication between Bluetooth Smart Sensors and the cloud. This means a remote device can Directly control a device by using IPv6 to target the device. The profile enables the use of UDP/TCP/other IP stacks based on IPv6 and 6LoWPAN. We should expect a device with more memory and processing power it it adopts IPSP. Although, the IPSP has been adopted by Bluetooth SIG, it is not widely implemented yet. (**http://www.bluetooth.com/Pages/Press-Releases-Detail.aspx?ItemID=220)

Those two technologies have proposed solutions in two important use case of BLE.

- First, taking advantages of BLE's short range radiation, 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 things.
- Second, making BLE devices as much visible as possible in IoT. The 6LowPAN grants IPv6 address to BLE devices to achieve point to point control through cloud. The benefit of the solution is making communication become straightforward. Because of 6LowPAN, less intermediate devices are needed between remote Cloud and IoT nodes. However, the point to point communication may bring concerns about management and security. In terms of management, a remote device may send a request to get a virtual resource which will depends on one or more edge nodes. For example, a remote request from a house owner is rising the temperature of his or her house to 24c. The request may need to be interpreted into two or more request to all air condition in the house to cool different rooms. In this case, point to point communication is not suitable. Instead, a central controller is required. On the other hand, allow a remote device directly control a node in

IoT is not safe. If every personal cloud can maintain a router as a central server to deal with communication with other cloud is a far more elegant design.

3.6.3 Classic Bluetooth vs Bluetooth Low Energy

Based on above introduction, we can come to the conclusion that Classic Bluetooth and Bluetooth Low Energy target different market. For Classic Bluetooth, it is a desirable solution when higher data rate is required and more power are available. For Bluetooth Low Energy, it is designed for devices with less power than Classic Bluetooth devices. BLE is a better option when small amounts of data need to be transferred frequently. In addition, the BLE has some features make it different from other short range wireless technologies. For example, according to official website (http://www.bluetooth.com/Pages/low-energy-tech-info.aspx), BLE optimized the connect mechanism which makes it possible to setup connection to as low as 3ms. (http://www.medicalelectronicsdesign.com/article/bluetooth-low-energy-vs-classic-bluetooth-choose-best-wireless-technology-your-application)

In the next chapter, the paper will discuss associated protocols which can provide a global view when design a new protocol for the proposed architecture.

3.6.4 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 big changes for BLE are IPv6 capability, the packet size increased and security level upgraded. Those improvements meet the development of IoT. Developers need Bluetooth become more compatible with existing network and can transfer more data in 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, we can see there are two innovation direction for BLE smart device. The first direction is cheap and simple sensor. This kind of sensor have 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. Them will constantly report two data to local hub. In this case, each sensor should be as cheap as possible because of big quantity. On the other hand, sensors in smart watch are different, where many sensors are integrated in one device and 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 in second scenario. Now, developers are more comfortable to develop a medium size app for integrated sensors in BLE environment.

3.7 CAP Theorem

The CAP stands for consistency, availability and partition tolerance. It is an important concept in any design of a distributed architecture. We explain it in details as following.

- Consistency: Consistency means only users an operation of users can only be fully executed or dropped. For example, in a system which is lack of consistency, user might get 100 dollars in an account with 50 dollars by sending request multiple times.
- Availability: Availability means 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 different copies in different node. Since the whole data assets are available for each node, 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 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 to different partition. Thus, if connection lost for a node, other node can change data without worrying about the issue of inconsistency when the lost node backs to the network. However, in this scenario, users cannot access data in the lost node.

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

3.8 Data Security

In many use case of short range data communication, we assuming no hackers exists between two devices. However, with the increasing numbers of sensors, security of short range data transfer is more and more considered by designer. In order to provide reliable data communication, data security must be take into account. In the following paragraphs, we will explore popular technology which are used in data encryption.

3.8.1 Symmetric encryption

Before 1976, people use same pattern to encrypt data. Procedures are following:

1. A use a standard to encrypt data.
2. B use same standard to decrypt data.

Because same key are used at both sides, this way for secure data is called Symmetric encryption. Although, this symmetric encryption is efficient and straightforward, it is hard to update key. It is always a headache to transfer key in channel.

3.8.2 Asymmetric encryption

In 1976, Whitfield Diffie and Martin Hellman proposed a way to decrypt ciphertext, which is called Diffie-Hellman key exchange algorithm. Their research made public understand that encrypt and decrypt can use different key. They only need to have relationship. In this way, we no longer need to transfer key in communication. This new pattern of encrypting data is called asymmetric encryption. Procedures are following:

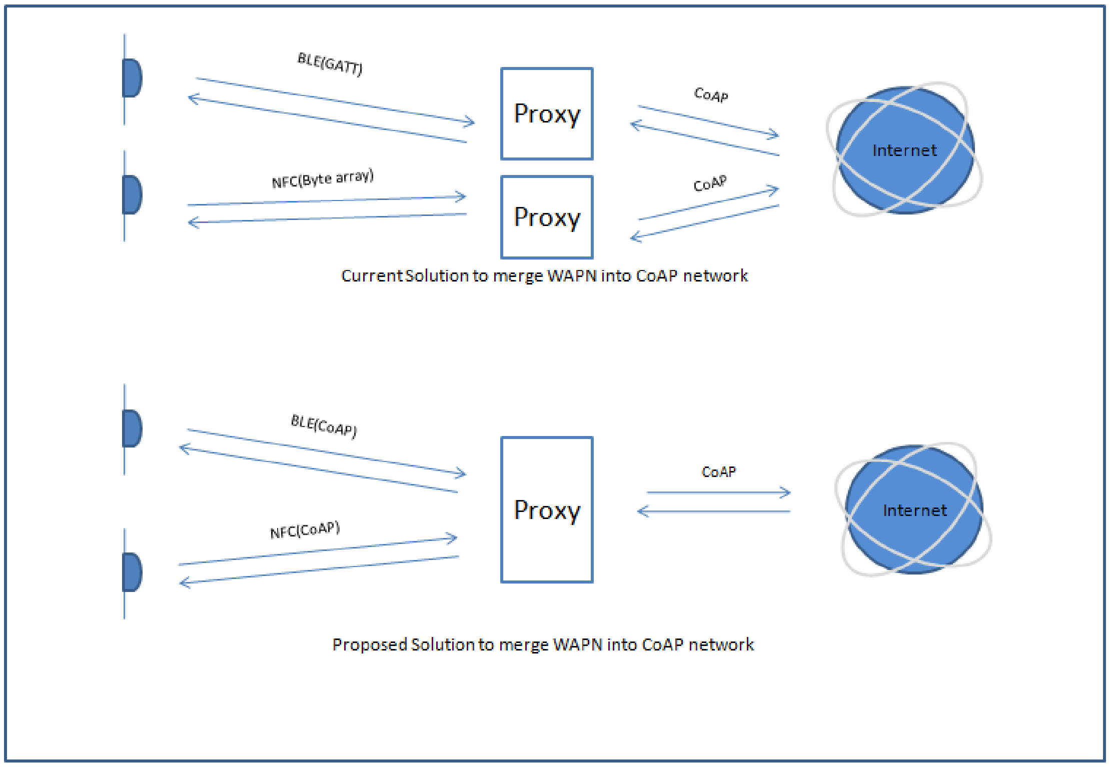
1. B generate public key and private key. The public key can be get by anyone but private key is secret to others.
2. A get B’s public key to encrypt data and send data to B.
3. B use its private key to decrypt encrypted data from A.

In this pattern, public key is used by senders and private key is used by receivers. Since public key is not designed to be secret. It can easily be transferred through internet.

In the proposed architecture, a build-in mechanism for data security should be implemented since data security is a general concern in data communication and original CoAP standard have not make effort on it.

CHAPTER 4 DESIGN AND ARCHITECTURE

As discussed above, the bandwidth of WAPN keeps increase for supporting large data transfer. Therefore, the fragile network of sensors which play a role as edge of IoT will become more and more reliable. Under this prediction, we expect the CoAP should be supported in those networks. In this research our main goal is propose a suitable solution to support CoAP in WPAN.

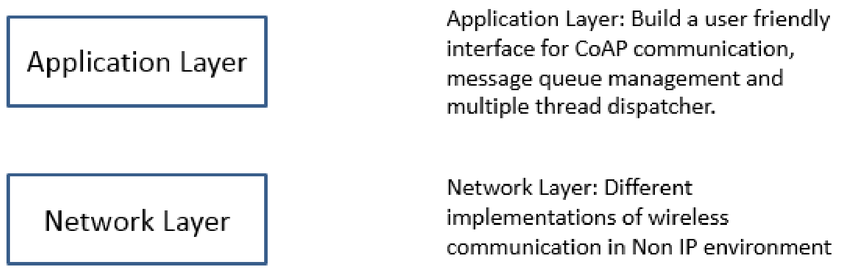

BLE(GATT)
Proxy
CoAP
Internet
NFC(Byte array)
Proxy
CoAP
Current Solution to merge WAPN into CoAP network
BLE(CoAP)
Proxy
CoAP
Internet
NFC(CoAP)
Proposed Solution to merge WAPN into CoAP network

As shown above, currently the 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 own standard 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 same protocol to transfer data, the proxy can easily be integrated. In order to achieve proposed network structure there are three major things need to be solved.

1. Use same standard to uniquely identify a device.
2. Need at least two layers to separate operations and detail implementation of communication.
3. Customize CoAP protocol to make it support Non-IP environment.

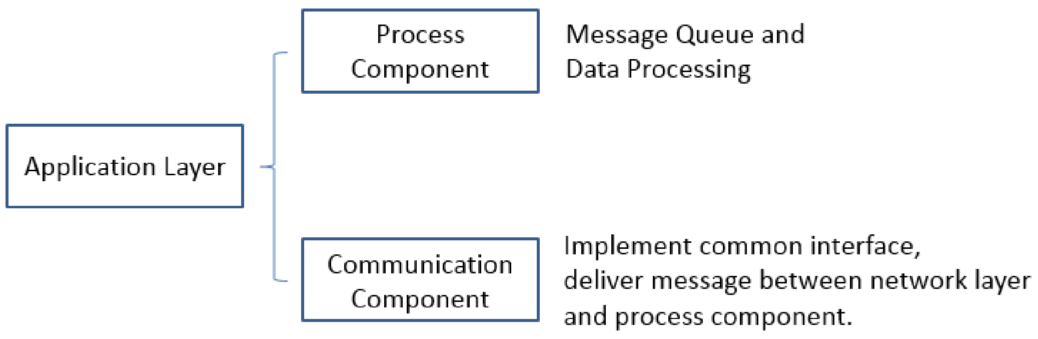
In the remaining section of the chapter, we will introduce our solution: CoAPNonIP architecture.

The proposed CoAPNonIP architecture consists of application layer and network layer. The Application Layer focus on message management and message deliver.


Application Layer
Application Layer: Build a user friendly interface for CoAP communication, message queue management and multiple thread dispatcher.
Network Layer
Network Layer: Different implementations of wireless communication in Non IP environment

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.


Process Component
Message Queue and Data Processing
Application Layer
Communication Component
Implement common interface, deliver message between network layer and process component.

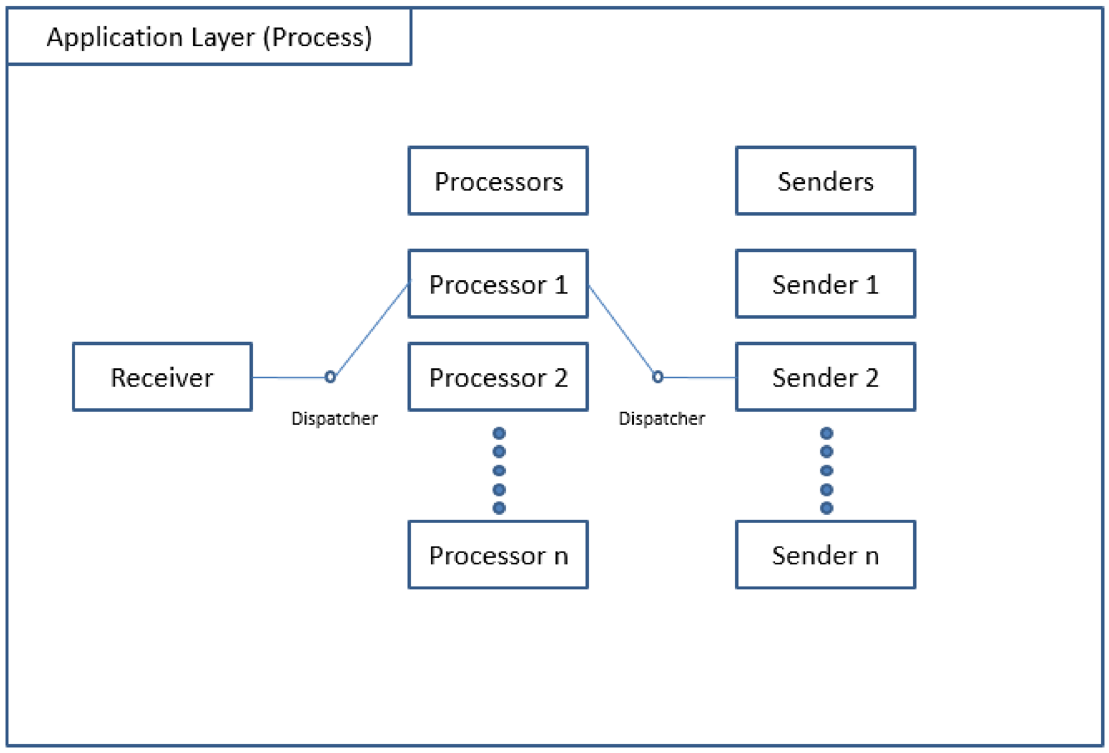
As shown above, in order to grant capability of supporting different protocols of WPAN (wireless personal area network) the architecture and provide a robust management mechanism, we implemented two components in application layer. The process component aims to provide queue management for send and receive messages. The communication layer aims to provide 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 where data are packet to byte array and send to other device. It consists of multiple thread to handle CoAP messages.
- Processors: The processors consist of multiple thread where specific CoAP request can be processed in 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 user try to send a request, he or she can specify a customized callback function to handle responses of the request which will be register in a callback map for management.
- Default Receive Handler: If user not specify 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. In order to increase 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. Moreover, auto cache for virtual resources are defined at this layer. As a supplement for resource mechanism in CoAP. virtual resources will try to get data from one or more devices at remote side where auto cache is available. In this way, user can easily gather data from multiple devices.



As shown above, there are three important roles in application layer: Receiver, Processors and Senders. When a receiver get message from lower network layer, it will trigger a callback which may be defined by user or a default one. If it need to process the received message, it will run a dispatcher to decide in which processor to process the data. After process 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 need to broadcast itself to make sure other devices can find it. In this state, we say the device is discoverable. In terms of BLE's implementation, the broadcast method will try to advertising itself as a BLE advertiser. If any connection be 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. In terms of BLE's implementation, the method will trigger a search operation. If a searcher finds a broadcaster and create a connection with it, the searcher becomes a central device.
- Get Nodes: A node need to know available devices in its network to decide destinations of message. In terms of BLE's implementation, the method will return all connected devices.
- Send Data: Every node has the ability to send data to one or multiple target devices. In terms of BLE's implementation, the method will deliver destinations and byte array of message to network layer for further process.
- Receive Data: Every node need to receive data from remote devices. In terms of BLE's implementation, the method will receive data through call back function.

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

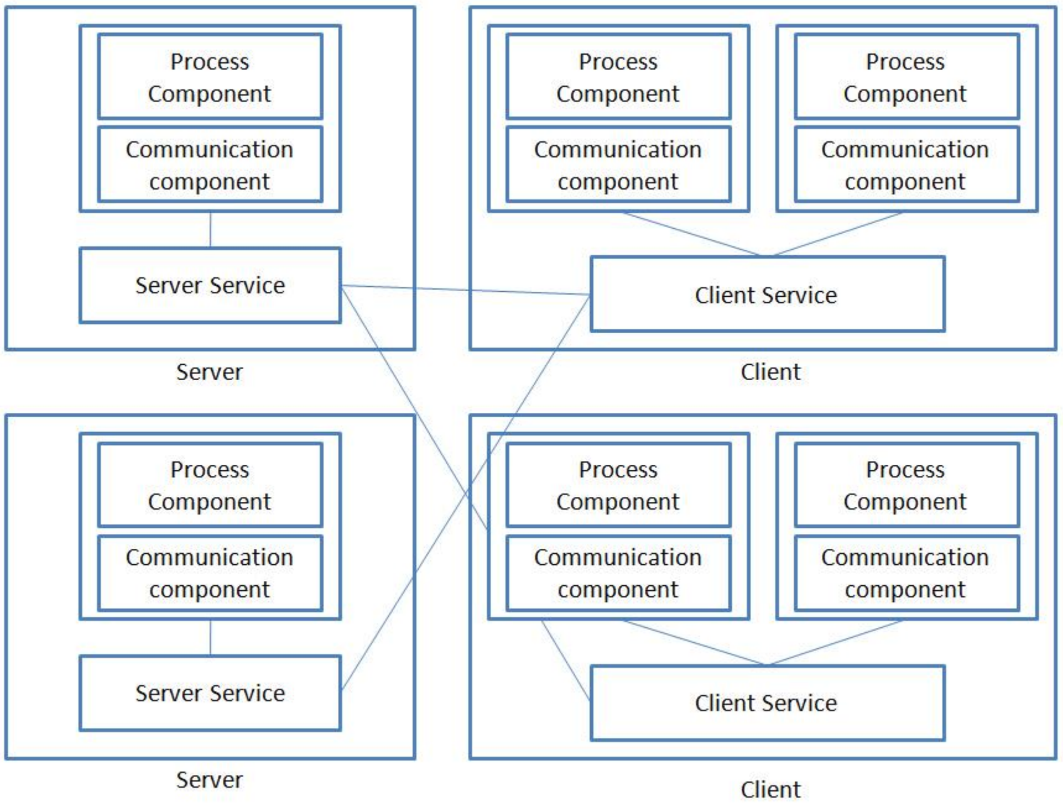
4.1.3 Conclusion

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 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 technology and provide capability.

4.2 Network layer

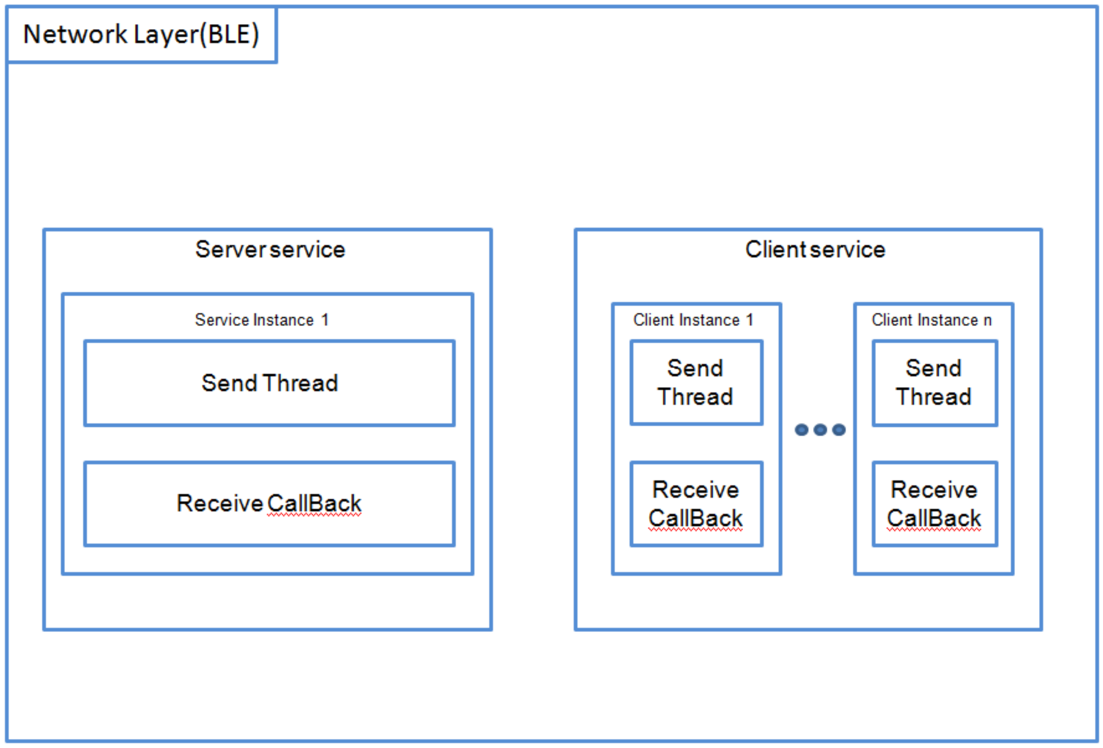
In the proposed architecture, network layer is the base component for communication.
The main task of network layer is get data from remote device or upper layer, process message, deliver processed data to remote device or upper layer.
For different protocols, it can be implemented in different ways.
Under the context of BLE， we create a micro service will run at background. It receives messages from upper layer and chop message into multiple 20-bytes packets. Meanwhile, it retrieves data from remote devices, assemble them into CoAP message and deliver those message into upper layer.



The chart above is shows how communication is organized by the proposed architecture. Server is the device who advertise itself to create connection. Client is the device who search devices to create connection. Communication between application layer and network layer is achieved through message broadcast.

At application level, user can define one or more instance to handle messages, which may happen in 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 application layer, which multiple apps can share the service of network layer.



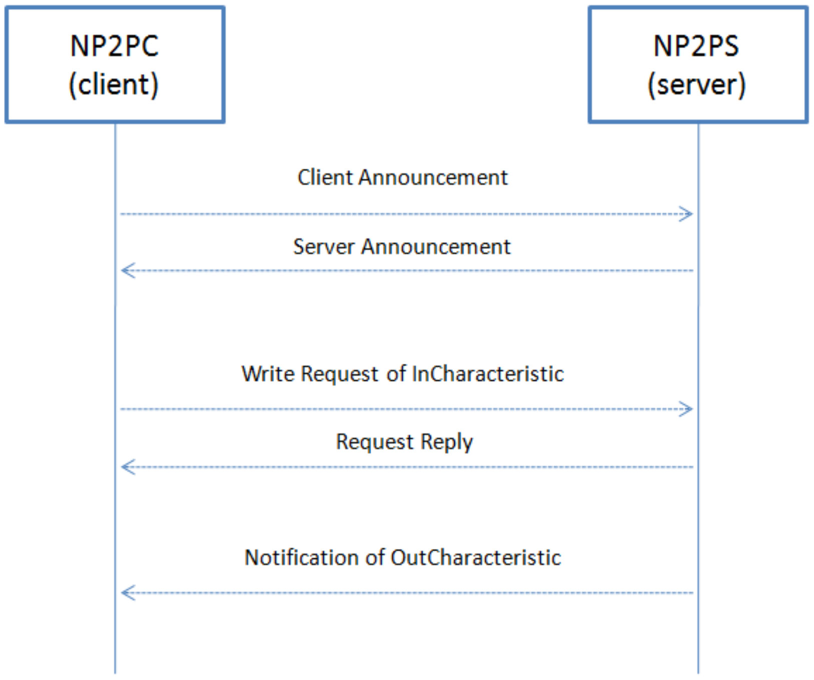
As shown above, since two roles are available for a device, the network layer of BLE consists of server service and client service.

If the device will play a role as a server, it will create a thread to loop message queue for send out 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 need 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, BLE connection is open and close periodically. Messages only send out in short time frames when connection are available. We also known that there are two ways to send message in BLE: One is send message sequentialize, which will wait signal from remote device before send the next message. The other is send message none sequentially where messages are send not without wait response from 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.



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

We 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 source device. In this way, before send any information both sides can know roles of remote device. On the other hand, AppID and UserID in normal messages are values of target device. Since the architecture use AppID and UserID fields in two different ways, it can overcome the chaos of message received at server side as well as act like unique ID for different application.

The proposed communication is based on two characteristics’ operations: InCharacteristic and OutCharacteristic.In order to provide reliable data transfer, every request from one device to another need to wait reply before sending the next message packet. As explained above, data communication between two BLE device are based on data change in characteristic. So we taking advantages of the architecture by using two characters to send and receive messages for a BLE server. One the other hand, we expect message in the communication channel is 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 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 other device know which user and app are sending 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 a numbers are used to indicate which app are sending request. The arrange of the number is from 1 to 16383.

After AppID, a packet must use 16 bits to indicate which user are sending request. 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.

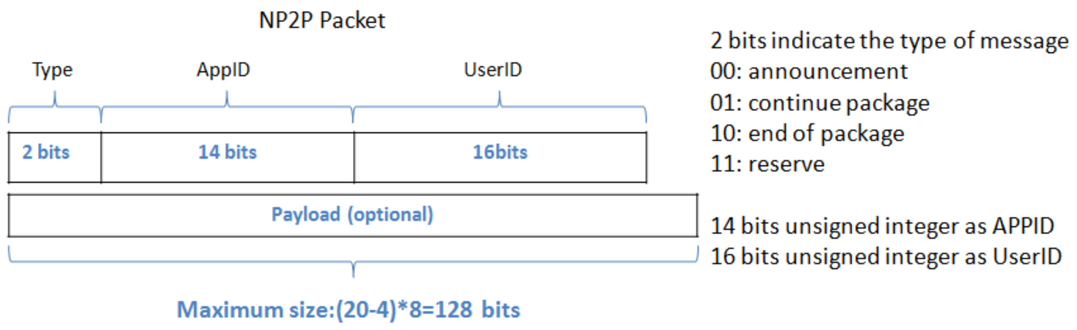

NP2P Packet
Type
AppID
UserID
2 bits
14 bits
16bits
Payload (optional)
Maximum size:(20-4)*8=128 bits
2 bits indicate the type of message
00: announcement
01: continue package
10: end of package
11: reserve
14 bits unsigned integer as APPID
16 bits unsigned integer as UserID

Figure 3: 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 sensor are merging 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.

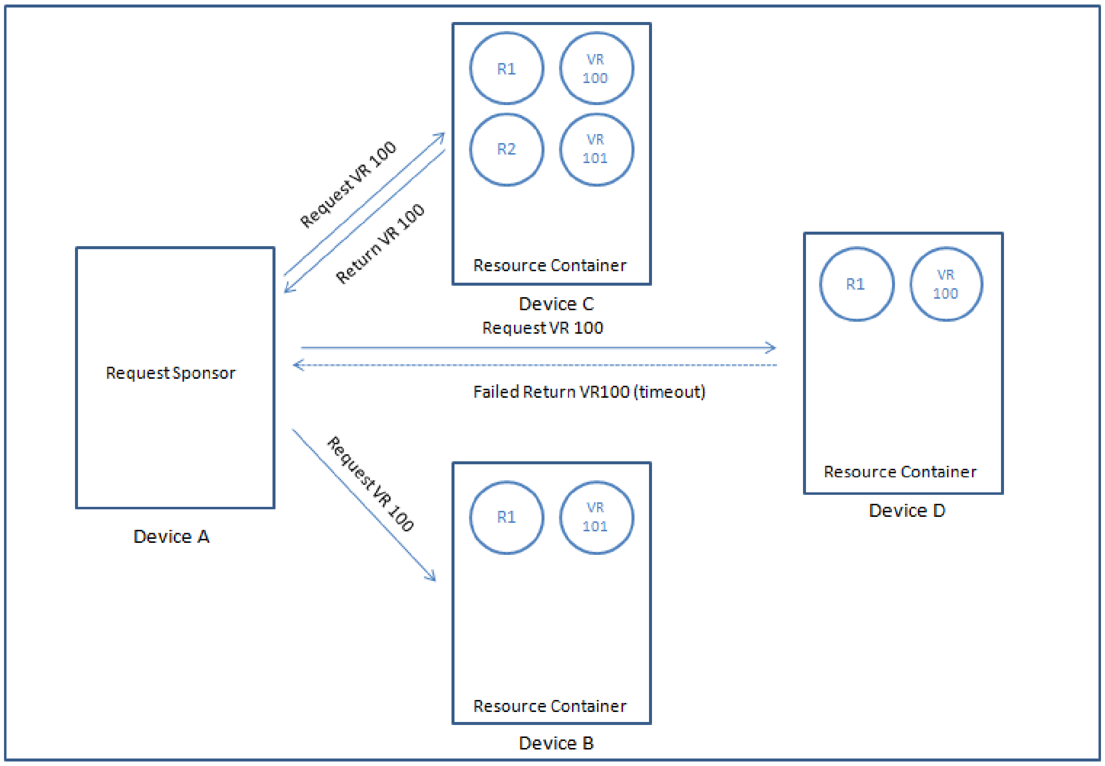


Figure 4: Virtual Resource Mechanism

As shown in the above page, 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 introduce a timer to execute operations after timeout and a hashmap to store key value pair of device id and response.

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.

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

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

4.6.1.1 Process Component

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

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 message into a default data handler.
2. InitProcessor: By implement this function, 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 request, the architecture also can create one or more thread to send out messages.
4. SetDefaultResponseHandler: In order to handle requests which are not appears in the map. User need 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 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 search available broadcaster. Due to performance issue, current implementation of BLE do not support this role yet.

4.6.1.2 Communication Component

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

- BoradCast: Initialize a device as broadcaster and begin to advertise itself.
- SearchPeers: Initialize a device as peer seeker. Since searching peers is an 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 user determined data package destination and acknowledge 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 provide one or more program at higher level, which brings great benefit to achieve cross application communication.

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 thread. A communication thread in a client will not only listen events from 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 two kinds of messages from upper layer: send announcement and send message. Those two kinds of messages will be assembled into different data format for communicate in BLE channel. Since the size of a CoAP message can easily over 20 bytes, the system may need to cut messages into multiple packets which will be transferred in communication channel and assembled at destination. After receive any send request, it will pack information into target format and put it into send 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, we also regard it as an 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 be triggered, the client server will broadcast message to upper class.

In terms of send and receive message, the client server listen notification from server side which pass data to client by change the value of OutCharacteristic. Meanwhile, when messages need to be delivered, the client side will write data to InCharacteristic.

In terms of message type, the first two bytes of the proposed protocol indicate type of a message. As mentioned above, there are three types of messages. As soon as get a message from remote side, the receiver will retrieve its type. If 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 message type is 1(a continue packet), the receiver will add received message to a hashmap where the combined string of UserID and AppID is the key. The hashmap maintain messages received from connected device. If message type is 2 (the end packet of a CoAP message), the receiver will add message to hashmap, combine existing message pieces into an object and trigger 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 value to 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 listen send announcement and send message requests from 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.

In terms of send and receive message, the server service listens data changes in InCharacteristic as a data receiver. Meanwhile, when data need to be send, it will change the value of OutCharacteristic and send notification to client side.

In terms of message type. The server side also support three types of message which has been described earlier. Once message received, the server side need to parse it to higher level through handlers and intent message.

4.6.2.3 Flowchart

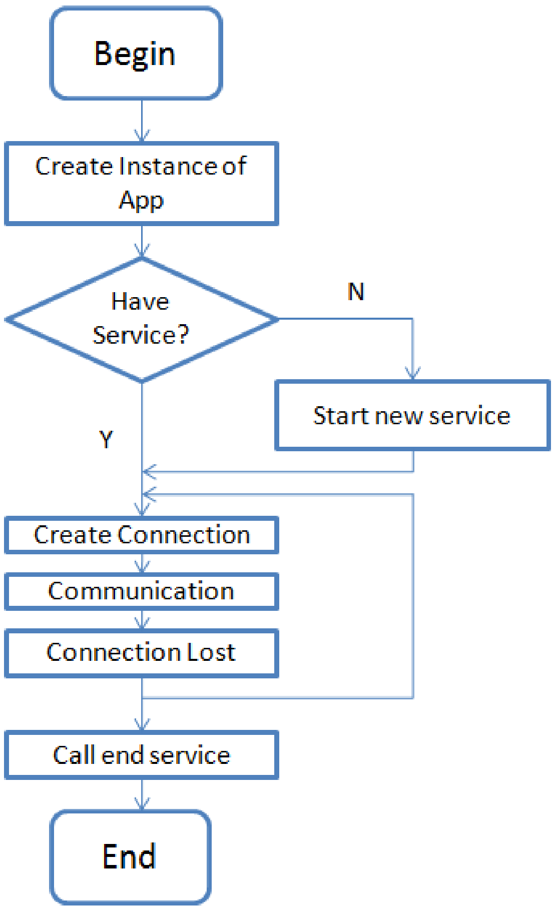


Figure 5: Life cycle of network service

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 object called App. When initialize the App, logics at application layer need to be specified. After call several functions to set the application layer (e.g. receiver thread, processor thread and default receiver), user need to call run function to check if a network layer service exists or not. If no service was running at background, a service will manage the lifecycle of a connection. The service will keep running unless user force to stop it.

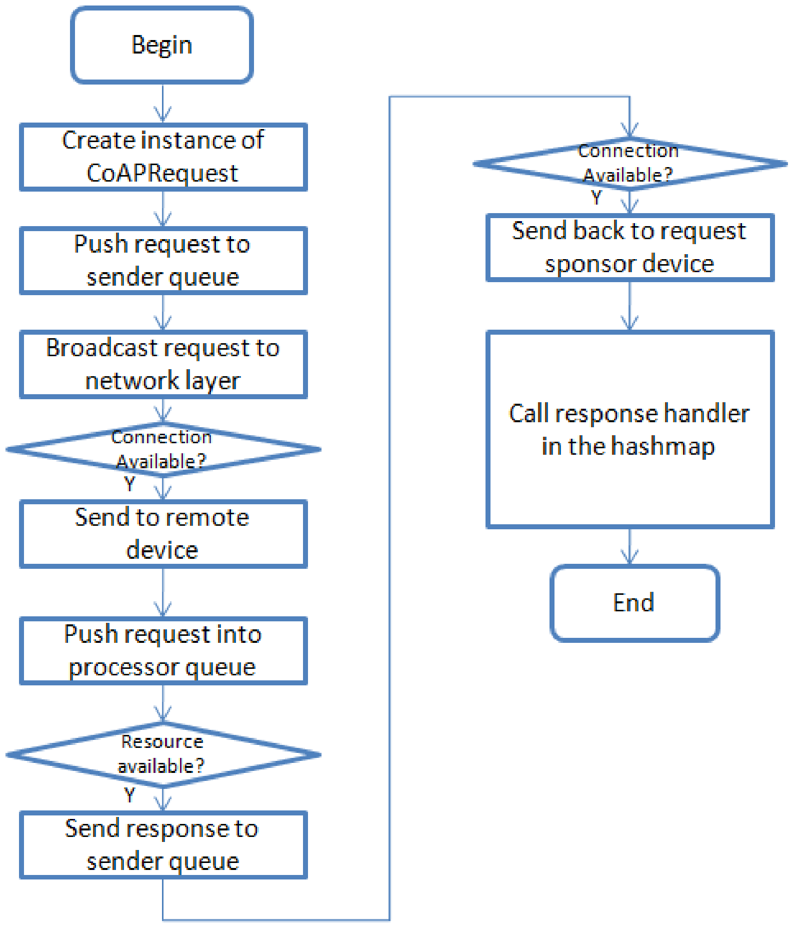

Begin
Create instance of CoAPRequest
Push request to sender queue
Broadcast request to network layer
Connection Available?
Y
Send to remote device
Push request into processor queue
Resource available?
Y
Send response to sender queue
Connection Available?
Y
Send back to request sponsor device
Call response handler in the hashmap
End

Figure 6: Life cycle of CoAP communication in proposed architecture

The flowchart shows the life cycle of a regular communication. Since the architecture is resource driven, a request needs to specify resource URL and targets. After create a request, user need to send it to the sender queue and register a handler for response in request hashmap. When a sender thread processing the request, it will send it to network layer service. In the service, the program will judge if a connection with target device is available and send message to the target. When a request is received, the network layer of remote device will throw it to upper layer. In the application layer, it will be pushed into processor queue. when a processor thread retrieves it from queue, it will judge whether target resource is available. If the resource is available, it will seal the value of it into a CoAP response and send it to request sponsor. 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 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 in two factors. One is find a way to overcome the limitation of BLE. The other is 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 physical layer, there may have either MAC address, IP address or other way to uniquely identify a device but they all need to use AppID and UserID to identify itself. One advantage of the design is the system can easily change devices with less effect to communication because we do not hardcode IP address to bind a device. Moreover, the adoption of AppID and UserID grants flexibility to the architecture where multiple communication technologies need to be supported at the same time.

4.7.2 Size limitation of a BLE request

We proposed an automatic packet cut and assemble 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 are solved at software level.

4.7.3 Chaos server-side callback

As mentioned above, the server side of BLE are chaos. A BLE server will receive message different client in one channel. Therefore, we use AppID, UserID and as unique identifier to build a hashmap for each connected device. As mentioned in the design of packet. The AppID and UserID are must 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 devices. This trend has been provided by smart phones. In this architecture, user can declare different roles at application level. When a new message come, the network layer will simply broadcast message to upper level with information of remote devices’ AppID and UserID. User can select interested message from specific data provider. This design makes the architecture can serve one or more applications at the same time. In detail, the proposed architecture used an abstract concept called “device”, which contains a string as a unique identifier. In the proposed solution AppID and UserID are combined. However, developer can customize the object to contain other information as an identifier.

4.7.5 Provide common interface to support wireless communication protocols

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

CHAPTER 5 IMPLEMENTATION

In order to test performance of proposed architecture. I made a test program to create Bluetooth 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 IDE: Xamarin. There are two main thread involved in data communication. The producer thread generates random packet and push them into send queue. The send thread constantly monitors the send queue and send message out through BLE. In the test application, the implementation of network layer is based on the latest Android peripheral API of BLE (In Android API 5.1.1).

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 technology. In implementation we 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 5 experiments and list them in the below table.

Experiment	Purpose
Minimum Data Transfer	Test the performance of proposed architecture in light load data communication.
Multiple Packets	Test performance of the architecture with increasing load
Round trip	Test time consumption of retrieve data.

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 support android to android Bluetooth communication, we used two Nexus 9 to test BLE implementation of the architecture.

HTC Nexus 9 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 message for different experiment.

6.3 Details

According to implementations above, there are three key factors to influence the result.

1. Measured method: As explained earlier, because of time sync problem, we get data transfer time by measure the time gap between two request.
2. BLE build-in mechanism: Different manufacturer have different way to implement BLE. Some pre-set parameters can influence BLE's performance in different ways.
3. Message queue mechanism: In send thread of BLE implementation, a message queue will be checked in a while loop to determine whether to send a message out. In current implementation, the message queue will be locked when the size of it larger than 0 which means other messages need wait for all messages in queue been sent before they can be added to the queue.

6.3.1 Minimum Data Transfer Performance

Description. Since the design of BLE aims to support lightweight data transfer, it is important to get its performance with minimum payload. In this test, I send 4-bytesheader between two devices to get performance of proposed architecture in light load communication.

As mentioned in implementation, we implemented BLE in sequential send mode which means messages are not send out at the same time, instead, each message need to wait a signal from remote device before write a new data to characteristic. In this way, connection between devices become more stable. Moreover, this mechanism make it is possible to get transmit time by record response received time of each request.

In this Experiment, we tested performance of the architecture with light payload. Moreover, we expect to find some patterns of data to analysis.

Experiment step.

1. Start sample program scan and create 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.

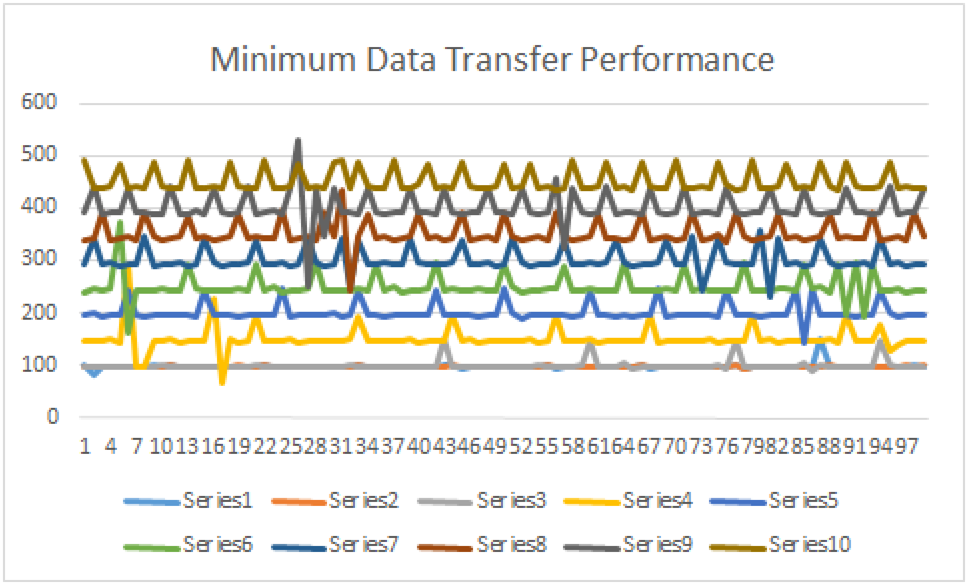


Figure 7: Summary of first experiment

As shown above,10 series of data are get. the x axis is the number of sample (99 samples for each series). the y axis is the data transfer time plus wait 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 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 need to wait a response signal of previous message before send a new one out. Since the time of send a message and wait 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 visible with the increasing of interval time. As shown in the blowing figure 8, 9 and 10. The heartbeat-like pattern still visible with 150ms delay and 200 delay. The heartbeat-like pattern have similar amplitude (around 50ms). The pattern can be explained with energy saving strategy of BLE.

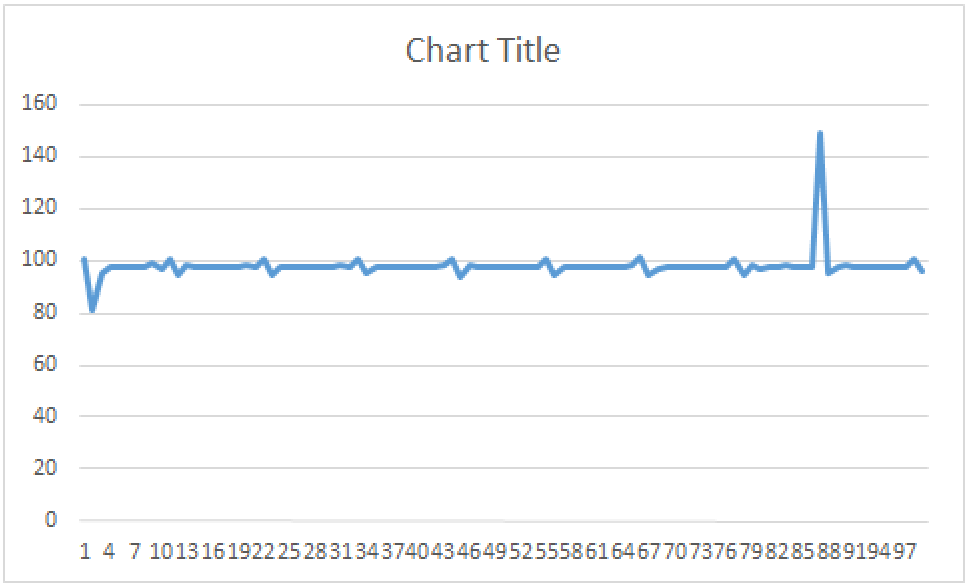


Figure 8: header no delay

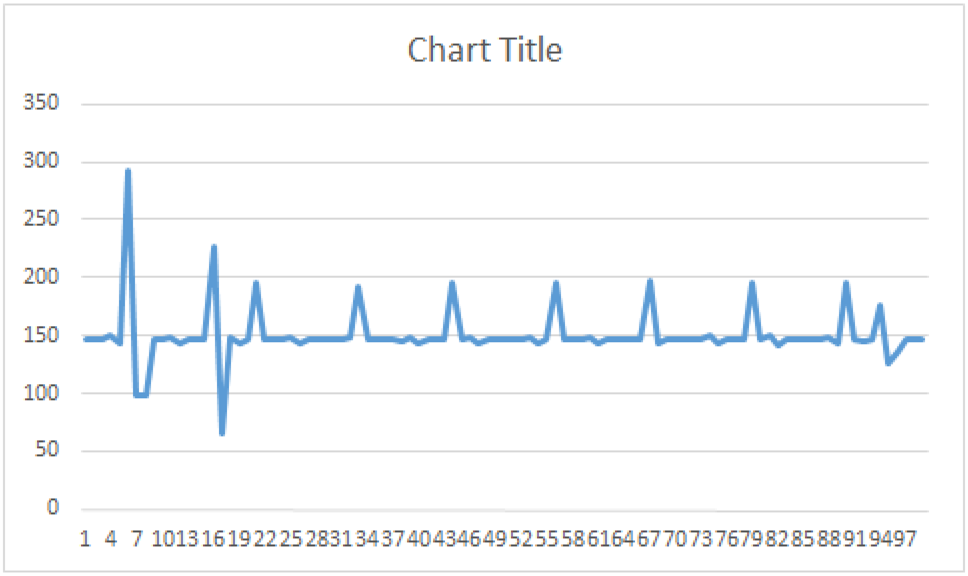


Figure 9: 4-bytes header 150ms delay

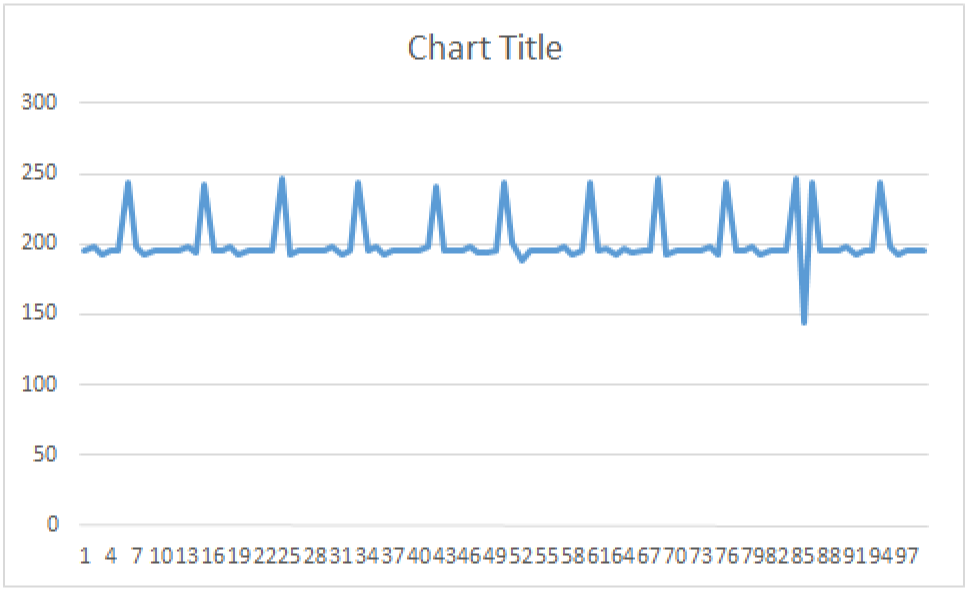


Figure 10: 4-bytes header 200ms delay

Third, the above charts show some random fluctuations. those fluctuations are not only randomly spread in time dimension but also have random amplitude. This pattern can be explained with known issue with Android Lollipop's BLE implementation or context switch problem.

Last, from figure 9 and 10 we know those random fluctuations becomes regular pattern when the system sends data with delay. This phenomenon can be explained by communication interval of BLE.

In order to further investigate the behavior of the system. we design an other set of test focusing on size change of BLE packet.

6.3.2 Multiple Packets

Description. From first experiment, we found three interesting pattern. We give out possible explanations for the last two. But we need further experiment 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 exist. Meanwhile, we expect more evidence to support our guess of data pattern in experiment.

Experiment step.

1.Start sample program scan and create connection between two devices.

2.Send CoAP message with payload 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 response from connected device.

4.Calculate transfer time according to time gap between message been received.

Result and Analysis.

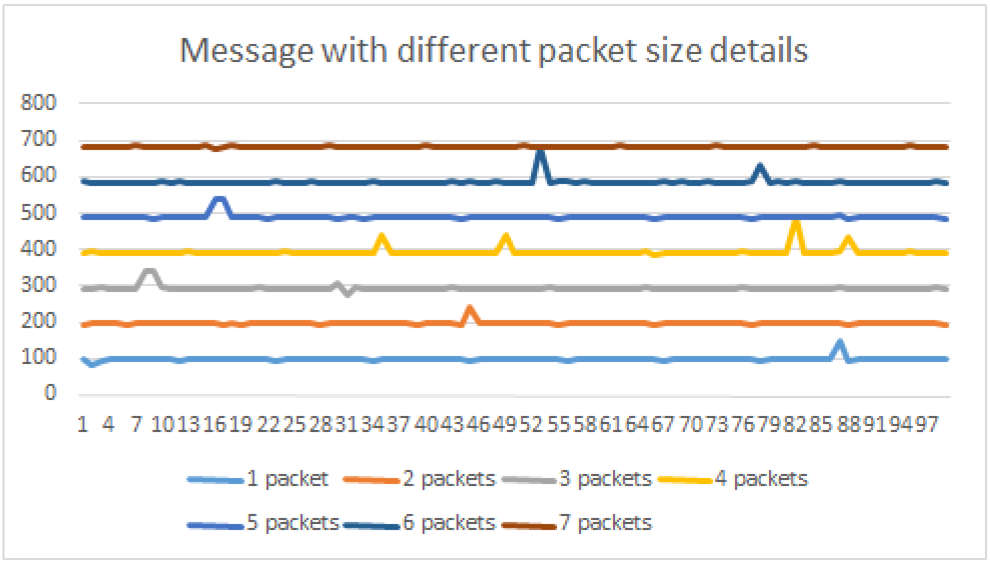


Figure 11: Summary of second experiment

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

1. With the increasing size of message, 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.

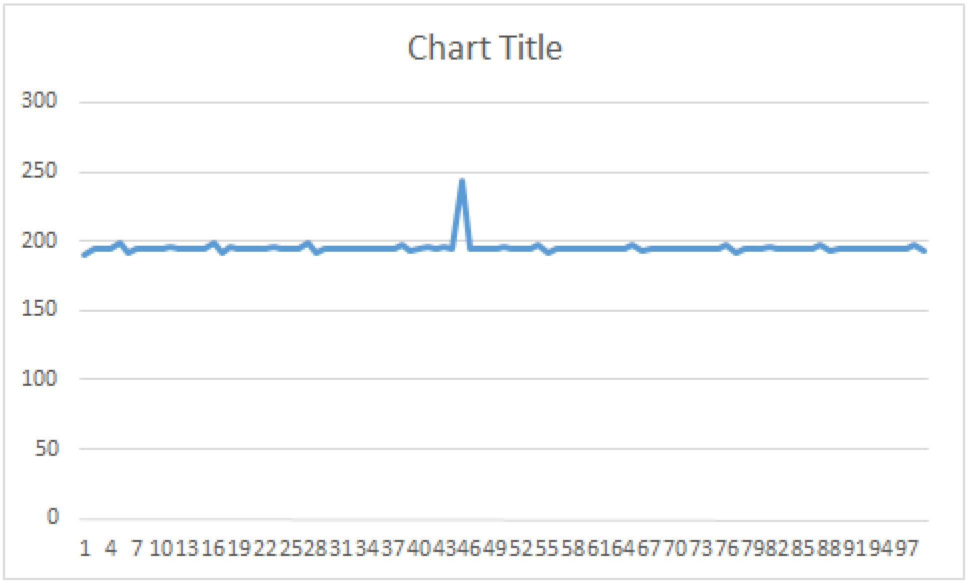


Figure 12: 12-byte payload (2 packets)

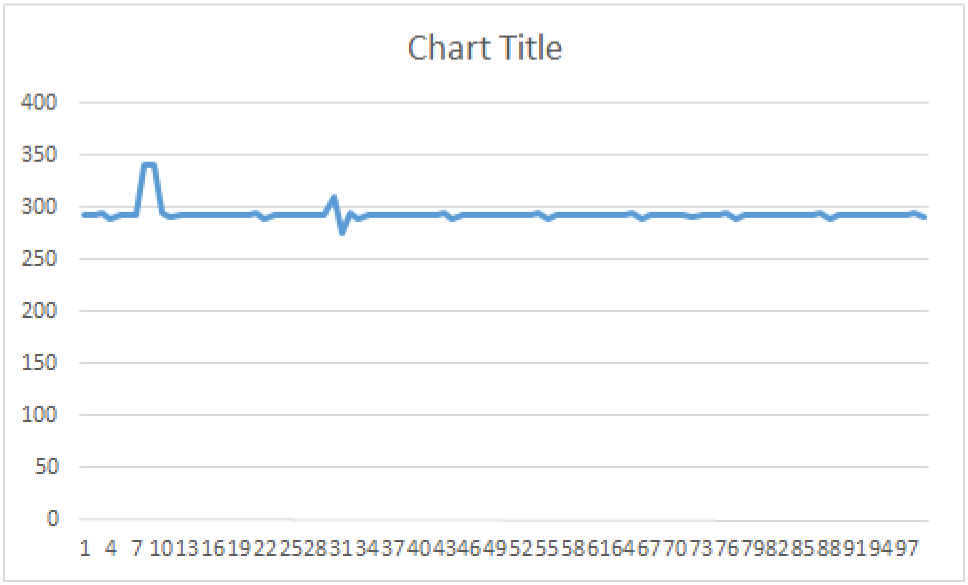


Figure 13: 28-byte payload (3 packets)

As shown above, both random fluctuation and heartbeat-like pattern exists in Figure 12 and 13. 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 receiver of 4-bytes header request always sends a response to sender.

Experiment step

1. Start sample program scan and create 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 time gap between message been received.
5. Repeat 2 to 4 three times.

Result and Analysis

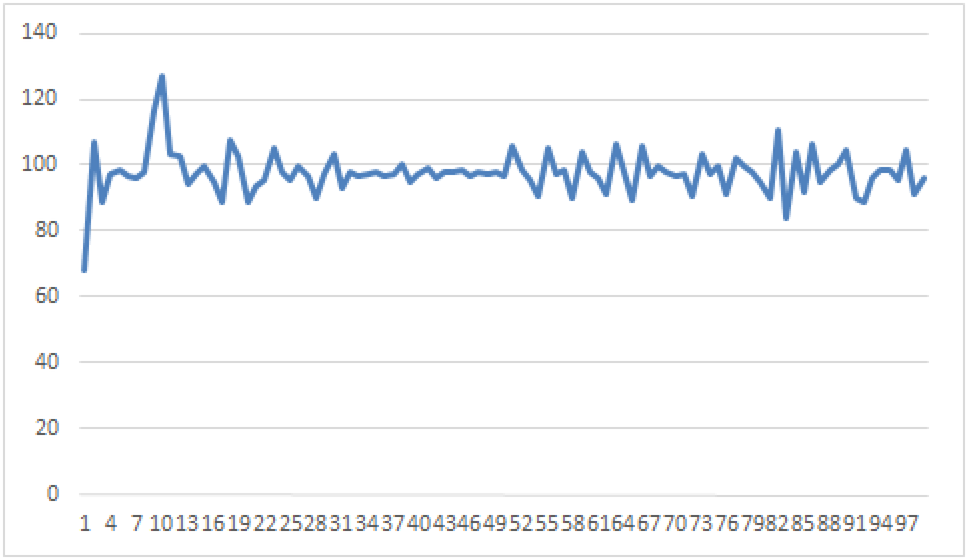


Figure 14: first set of 4-bytes round trip

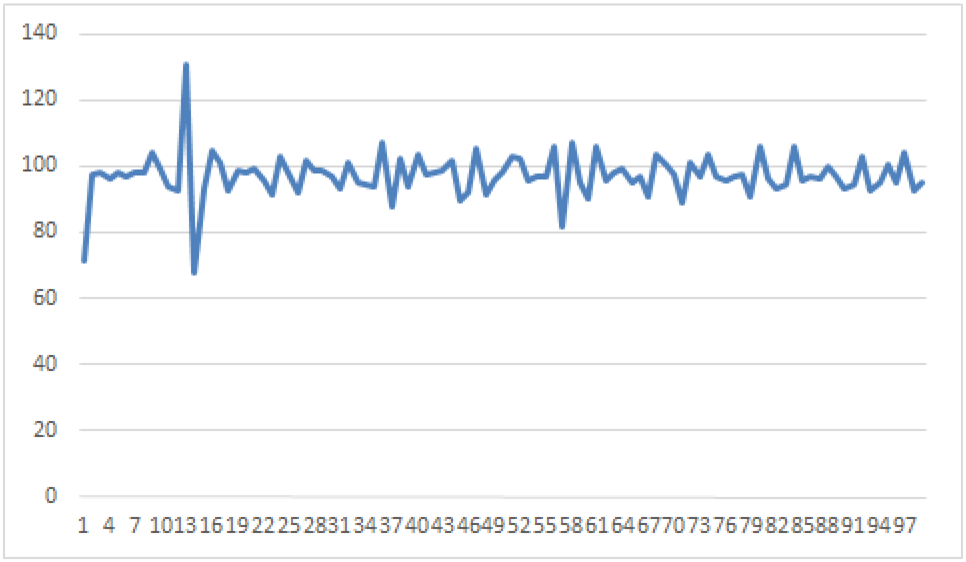


Figure 15: second set of 4-bytes round trip

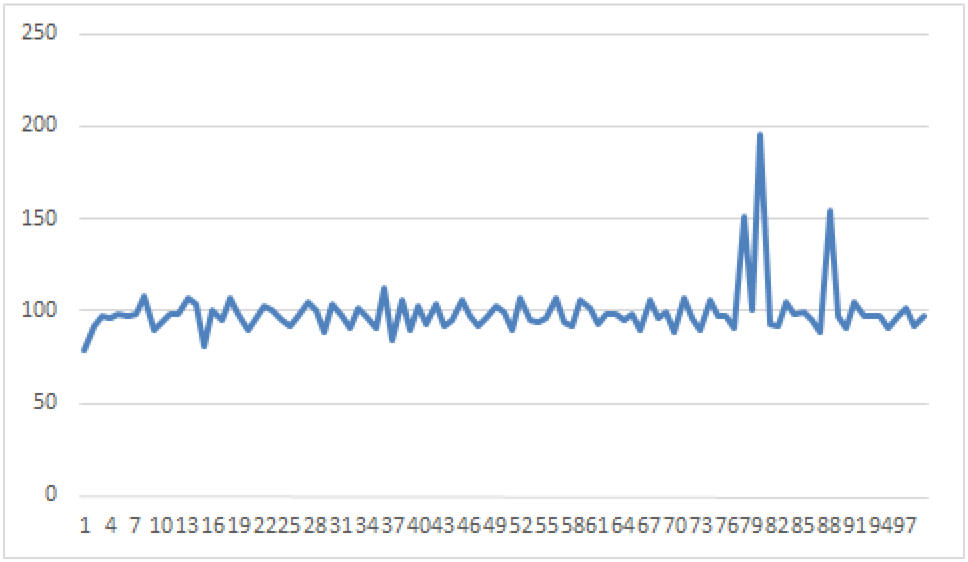


Figure 16: third set of 4-bytes round trip

In the first experiment, we examined one-way time consumption of 4-bytes header CoAP message. The data we get from that experiment shows obvious pattern. However, according to the charts above we can not find any pattern for the round trip experiment. Compared those two experiments, we can 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, we 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.

6.4 Conclusion

Based on the analysis of experiment results, we enhanced understanding of implementations of Android’s BLE. Meanwhile, we obtained more data about performance of proposed architecture. In above experiments, we tested our architecture’s performance in lightweight and medium weight payload communication which are common sense of using CoAP. According to above analysis we get following information:

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 currently 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 most like to be the reason of constant fluctuation when we send data with different latency.
3. There is 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 on android device. It is not surprise to see bad performance from the new feature. We expect the experiment result can be improved on new Android device with latest API and BLE hardware.