Design and Implementation of an Open Beaconing Architecture for Internet of Things

by

ARJUN KUMAR BHASKAR SHETTY

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2015

Acknowledgements

I would like to thank my supervising professor Dr. Yonghe Liu for the opportunity to work with him in, in my interested field of research. He has been a strong motivation and has constantly supported me during the course of master's. I also wish to thank my academic advisor Dr. Ramirez Elmasri for his guidance. Also I would mention Dr. Chengai Li and Dr. Dimitrious Zikos for taking interest in my work and making time to be in my thesis committee. I would like to mention my friends and family who have financially supported me to pursue my dreams, I am indebted to them.

April 14,2015

Abstract

Design and Implementation of an Open Beaconing Architecture for

Internet of Things

Arjun Kumar Bhaskar Shetty, MS

The University of Texas at Arlington, 2015

Supervising Professor: Yonghe Liu

The power of the Internet is penetrating into the physical world with ever increasing speed and expanding reach under the general concept of Internet of Things. Extensive sets of efforts have been devoted toward establishing an interconnected network of smart devices in a self-organized manner. Nowadays, using low power beacons to announce the presences of physical objects has gained tremendous momentum exemplified by iBeacon from Apple. Unfortunately, schemes like iBeacon use abstract codes to denote a physical entity, limiting it to be only understandable by those with knowledge of it beforehand.

In this thesis, we develop an open architecture where a beacon will announce a URL representing the presence of the corresponding object in the proximity and a pointer for related information. By employing URL instead of an abstract code as in iBeacon, the architecture provides a flexible and powerful way of representing physical entities in an open manner. We implement the URL beacon broadcasting based on Bluetooth Low Energy using Texas Instruments' Sensortag nodes. We further implement a ranking scheme for the URL received by a smart device from the beacons around by based on a variety of metrics including signal strength, URL content, and user history.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction to Beacon and BLE

The term iBeacon and Beacon are often used interchangeably. iBeacon is the name for Apple's technology standard, which allows Mobile Apps (running on both iOS and Android devices) to listen for signals from beacons in the physical world and react accordingly. In essence, iBeacon technology allows Mobile Apps to understand their position[1] on a micro-local scale, and deliver hyper-contextual content to users based on location. The underlying communication technology is Bluetooth Low Energy.

Bluetooth Low Energy is a wireless personal area network technology used for transmitting data over short distances. As the name implies, it's designed for low energy consumption and cost, while maintaining a communication range similar to that of its predecessor, Classic Bluetooth. Power Consumption: Bluetooth LE, as the name hints, has low energy requirements. It can last up to 3 years on a single coin cell battery. Lower Cost: BLE is 60-80% cheaper than traditional Bluetooth. Application: BLE is ideal for simple applications requiring small periodic transfers of data. Classic Bluetooth is preferred for more complex applications requiring consistent communication and more data throughput.

BLE communication consists primarily of "Advertisements", or small packets of data, broadcast at a regular interval by Beacons or other BLE enabled devices via radio waves. BLE Advertising is a one-way communication method. Beacons that want to be "discovered" can broadcast, or "Advertise" self-contained packets of data in set intervals. These packets are meant to be collected by devices like smartphones, where they can be used for a variety of smartphone applications to trigger things like push messages, app actions, and prompts.

Figure 1-1 Typical BLE Beacon

In Figure 1-1 we can see the BLE enabled beacon. This beacon is being manufactured by various numbers of vendor now. We can obtain a beacon now at a cost as low as $20 and in future as the usage increases the cost of these beacon will go down drastically making it more and more feasible for different establishment to setup and utilize in the way they wish to do it. For thesis we have used TI sensor tag, which you can see below in figure 1-2. TI sensor tag is a beacon with various sensors, which will be useful for different application.

1.1 Bluetooth Low enery

Bluetooth Low Energy hit the market in 2011 as Bluetooth 4.0. When talking about Bluetooth Low Energy vs. Bluetooth, the key difference is in BLE's low power consumption. Below in table 1-1 we have listed the comparision of the two Blutooth technology.

Table 1-1 Classic Blutooth vs BLE Comparision

	Classic Bluetooth technolgy	BLE
Data Payload throughput	2Mbps	~100kbps
Robustness	Strong	Strong
Range	Up to 1000m	Up to 250m
Local system density	Strong	Strong
Large scale network	Weak	Good
Low latency	Strong	Strong
Connection set-up spee	Weak	Strong
Power consumption	Good	Very Strong
Cost	Good	Strong

In several key aspects, Bluetooth low energy technology is a totally new technology. For instance, the technology features very efficient discovery and connection set-up, short data packages, and asymmetric design for small devices. As with Classic Bluetooth technology, Bluetooth low energy technology is based on a master connected to a number of slaves. However, in Bluetooth low energy technology the number of slaves can be very large; how large depends on the implementation and available memory. The new advertising functionality makes it possible for a slave to announce that it has something to transmit to other devices that are scanning. Advertising messages can also include an event or a measurement value.

There are also differences in software structure. In Bluetooth low energy technology all parameters have a state that is accessed using the attribute protocol. Attributes are represented as characteristics that describe signal value, presentation format, client configuration, etc. The definitions of these attributes and characteristics along with their use make it possible to build numerous basic services and profiles like proximity, battery, automation I/O, building automation, lighting, fitness, and medical devices. All these nuances are needed to make the implementation seamless and compatible between devices from different manufacturers.

The use of low power consumption and coin cell battery operation also has its limitations. Data transfer rates with Classic Bluetooth technology using enhanced data rate (Bluetooth v2.1 + EDR) [6]can exceed 2 Mb/s (actual payload), but practical transfer rates for Bluetooth low energy technology are below 100 kb/s (actual payload of roughly 1/20). Therefore, streaming Bluetooth low energy connections will lose a great deal of the huge potential power savings as the utilization approaches continuous transmission.

Here is a summary of key BLE terms and concepts:

Generic Attribute Profile (GATT)—The GATT profile is a general specification for sending and receiving short pieces of data known as "attributes" over a BLE link. All current Low Energy application profiles are based on GATT.

The Bluetooth SIG defines many profiles for Low Energy devices. A profile is a specification for how a device works in a particular application. Note that a device can implement more than one profile. For example, a device could contain a heart rate monitor and a battery level detector.

Attribute Protocol (ATT)—GATT is built on top of the Attribute Protocol (ATT). This is also referred to as GATT/ATT. ATT is optimized to run on BLE devices. To this end, it uses as few bytes as possible. Each attribute is uniquely identified by a Universally Unique Identifier (UUID), which is a standardized 128-bit format for a string ID used to uniquely identify information. The *attributes* transported by ATT are formatted as *characteristics* and *services*.

Characteristic—A characteristic contains a single value and 0-n descriptors that describe the characteristic's value. A characteristic can be thought of as a type, analogous to a class.

Descriptor—Descriptors are defined attributes that describe a characteristic value. For example, a descriptor might specify a human-readable description, an acceptable range for a characteristic's value, or a unit of measure that is specific to a characteristic's value.

Service—A service is a collection of characteristics. For example, you could have a service called "Heart Rate Monitor" that includes characteristics such as "heart rate measurement." You can find a list of existing GATT-based profiles and services on bluetooth.org.

NFC cannot deliver the experiences that can be achieved with the use of BLE and the beacons, mainly because it is a short-range technology only. This is an area in wich BLE is unique and has no competitors, but BLE can replace NFC in certain tasks in which NFC seems to fit very well, as for instance pairing devices “automagically”, or any other kind of interaction involving low consumption and proximity. NFC will still be present in those circumstances where the use of a unpowered chip is necessary, such as cards. Before BLE entered into the picture, Bluetooth and NFC were addressing different and clearly defined purposes in mobile wireless communications: NFC was establishing itself as a low-power technology for short distance operations with low data traffic, while Bluetooth was being used (and is still used) extensively in the field of communications with other devices over short distances, with increased data traffic but with a higher energy consumption.

1.2 TI Sensor Tag

As shown in Figure 1-2 we use the sensor tag for working. The CC2541 [3] SensorTag is the first Bluetooth Smart development kit focused on wireless sensor applications and it is the only development kit targeted for smart phone app developers. The SensorTag [2]can be used as

reference design and development platform for a variety of smart phone accessories. Sensortag contains these sensors IR temperature sensor, Humidity Sensor, Pressure Sensor, Accelerometer, Gyroscope, Magnetometer. These sensors collect the data from the surrounding and transmit it through Bluetooth Low Energy to a Client (Smart Device). The design guide to this sensor tag can be found on the net [4][5]. It is a open source which make it more practical for the common man to use it.

Chapter 2

The Concept and Specifications

This is an effort to extend the core superpower of the web - the URL - to everyday physical objects. Our core premise is that you should be able to walk up to any “smart” physical object (e.g. a vending machine, a poster, a toy, a bus stop, a rental car) and interact with it without first downloading an app. The user experience of smart objects should be much like links in a web browser: i.e., just tap and use. At its base, the Physical Web is a discovery service: a smart object broadcasts relevant URLs that any nearby device can receive. This simple capability can unlock exciting new ways to interact with the Web. The URL is the fundamental building block of the web, giving remarkable flexibility of expression. It can be: a web page with just a tiny paragraph of info, a fully interactive web page, a deep link into a native application.

As the number of smart objects is going to explode, both in our homes and in public spaces. Much like the web, there is going to be a long tail of interactivity for smart objects. But the overhead of installing an app for each one just doesn’t scale. We need a system that lets you walk up and use a device with just a tap. This isn’t about replacing native apps; it’s about allowing interaction for the times when native apps just aren’t practical. Must be an open standard that everyone can use. This can’t be a product that is locked down by a single company. Like many web specifications, this is an open source design that is being released early so everyone can experiment.

This involves creating an open ecosystem where smart devices can broadcast URLs into the area around them. Any nearby display such as a phone or tablet can then see these URLs and offer them up to the user. It mirrors the basic behavior we have today with a search engine. The user requests a list of what's nearby. A ranked list of URLs is shown. The user picks one. The URL is opened in a full screen browser window, as shown below in figure 2-1.

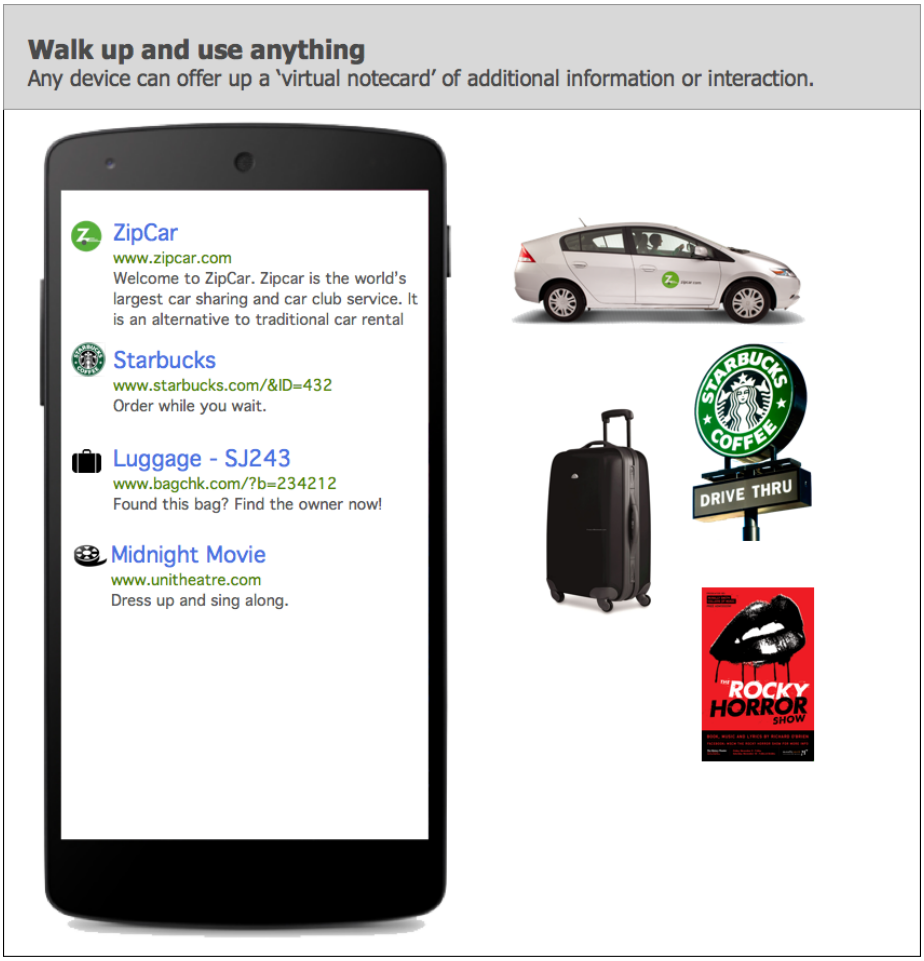

Figure 2-1 Physical Web Example

A core principle of this system is no proactive notifications. The user will only see a list of nearby devices when they ask. If your phone were to be buzzing constantly as you walked through the mall, it would be very frustrating. Push notifications in general are too easily abused. Of course, the user can opt-in to notifications, we are just saying that by default, the user must ask to see anything nearby.

In addition, we only scan when the screen is on: there is no scanning that goes on when the phone is in your pocket. This is consistent with our 'no interruptions' goal but it also has a large positive impact on power usage. Using this app should have very little impact on your phone's battery life.

At first, the nearby smart devices will be small but if we're successful, there will be many to choose from and that raises an important UX issue. This is where ranking comes in. Today, we are perfectly happy typing "tennis" into a search engine and getting millions of results back, we trust that the first 10 are the best ones. The same applies here. The phone agent can sort by both signal strength as well as personal preference and history, among many other possible factors. Clearly there is lots of work to be done here. We don't want to minimize this task, but we feel that this simple signal strength ranking can get us very far for the first few versions of this project.

URLs broadcast in the clear so anyone can see them. This is by design. That is why we're initially suggesting this to be used in public spaces. This does raise issues for home use where it would be possible for neighbors to intercept beacons. However, one of the big advantages of URLs is that there are so many ways they can be used to increase their security: The URL could be obfuscated (e.g. using a non-branded domain), The web page could require a login, A rotating token on the beacon would constantly change the URL, The URL could reference an IP address that is only accessible when connected to a local network. One of the big values of URLs is that they are so flexible and encourage this further evolution.

With any system, there will be people that try to exploit it. There will likely be many solutions around this problem but again, we can start with the web. Search engines today have this issue and are fairly effective and displaying the correct web sites in your search results. That same approach would apply here. Combine that with historical results of who clicks on what and it's possible to build a fairly robust ranking model and only show the proper devices.

The value of a URL is that it is a known part of the web, very flexible, and most importantly, decentralized. URLs allow anyone to play and no central server to be the bottleneck. This is one of the core principles of the web and critical to keep alive. That being said, we completely expect others to experiment with a url+ID model that goes through their server (e.g. safeurls.com/?id=12345). That is perfectly acceptable and to be encouraged. Systems like that are likely to provide much better security and vetting of sites. But that is the beauty of URLs, there can be as many of these as you'd like and the system still works seamlessly.

The Physical Web's primary value is to enable a device to place at users' fingertips anything from a tiny piece of location-based information to a full blown web app.

1.A dog collar could let you call a service to find the owner.

2.A bus could tell you its next stop.

3.A city rent-a-bike service could let you sign up on the spot.

4.A home appliance could offer an interactive tutorial.

5.An industrial robot could display diagnostic information.

6.A mall could offer a map.

Each of these examples, taken by itself, is modestly useful. Taken as a whole, however, they imply a vast "long tail" where anything can offer information and utility. Native apps are great for high frequency usage. The web enables people to walk up and use something once with hardly any friction. Our current URL broadcast method involves a Bluetooth broadcast from each beacon. The user's phone gathers this info without connecting to the beacon. This ensures the user is invisible to all beacons, meaning a user can't be tracked simply by walking past a broadcasting beacon.

Chapter 3

Design and open beacon architecture

The current design uses Bluetooth Low Energy (BLE) devices that broadcast the URL in the advertising packet. The sole job of the device is to broadcast the URL to the surrounding area. The reason is to accommodate potentially the worst-case scenario of a large number of devices in an area with many people. A broadcast only approach avoids an N-squared problem of every user connecting to every device. By making each device broadcast constantly, any number of devices can just pick up the information passively with little conflict. This has another advantage in that it means that a user can walk through a space and leave no trace: the broadcasting devices have no idea who is listening. The current prototype broadcasts once every second, striking a balance between user response time and battery life. There is nothing stopping a device from broadcasting faster if it wishes.

The URL is stored in the advertising packet of a BLE device. The packet identifies itself with a 16-bit Service UUID of 0xFED8 which indicates the beacon is a URIBeacon. This small size of the advertising packet does not leave a lot of room for the text of the URL. This is one of tradeoffs that comes from avoiding any connections to the beacon (to reduce tracking and avoid congestion). URLs are encoded so common patterns like 'http://www.' and '.com' can be compressed into a single character. This is very similar to NDEF compression in NFC. In addition, we expect initial testers will use either short domains or a URL shortener. Both the Android and iOS apps do this automatically when a URL is typed in that is too long to fit.

The current client is an application, rather than a system service or integrated part of the operating system, to prove out the technology. If you open the app, it lists the nearby beacons that it can see, sorted by signal strength. Note the signal strength is a very iffy metric as there many reasons why it can vary. However, if you are standing in front of device and the next device is more

than five feet away, it tends to work out well in practice. This is the primary reason we include a TX_POWER byte in the advertising packet so it's possible to calculate signal loss and rank different strength beacons.

The client lists the meta information from the URL: TITLE, DESCRIPTION, URL, and FAVICON. These can be pulled at the time the URL is received. Alternately, there is a simple caching proxy server that speeds up this process.

The server receives a request from the client with all the URLs found and returns JSON listing the URLs' metadata. The current prototype collects no user data, only returning the cached information. However, alternative implementations could keep track of user choice and use that to help change the ranking within the client. The server isn't required as part of Physical Web, but it greatly simplifies the work on the client side and improves responsiveness and quality of results.

The system expects URLs to point to a web page, which offers up the metadata described above. This somewhat limits the URLs, as they must always point to a valid HTML page. This is likely a significant limitation to URLs that want to link to native applications.

As more devices are found, the importance of ranking the devices becomes more valuable. Sorting only by signal strength is a good start, but the server could do a much better job in two ways. The first would be to track which URLs are clicked as that implies value, so more frequently used URLs could be ranked higher. In addition, the server could track personal use, so if you tended to pick the same device at work, it might rank the device higher as well.

At this point, the URL is broadcast as plain text, so we must be careful recommending this for personal use (as neighbors could know what devices you have in your home). However, there are many possible solutions. Turn down the transmit power so the range is quite small. Use obfuscated URLs, Web authentication mechanisms, such as cookies, A rotating key at the end of a dynamic URL, Use mDNS on a wifi network. The advantage of URLs is that there are several

standard, well-understood solutions. We expect discussion on this topic to continue and more robust solutions to be proposed.

The UriBeacon design allows a Bluetooth Low Energy (BLE) device to periodically transmit an ADV packet that contains a URI. This technology enables any device, place or thing to become a beacon through physical association. As a consequence UriBeacon enables any object in the physical world to transmit a URI over a short distance (typically 20-30 feet), and is an enabling technology to support the Physical Web paradigm. By providing discovery, search, and control (using web pages) based on situated proximity, the Physical Web can be thought of as an on-ramp to the Internet-of-things(IoT).

The UriBeacon consists of two basic data types in an Advertising Data (AD) block: <<Service UUID>> and <<Service Data>. Both of these data types utilize a 16-bit Universally Unique Identifier (UUID) allocated by the Bluetooth standards body for this purpose.

Table 3-1 Assigned UUID for URL Beacon

Assigned Number	Service Name	Service Description
0xFED8	Uniform Resource Identifier Service	The Uniform Resource Identifier (Uri) Service provides a way for a Bluetooth Low Energy device to discover nearby identifiers for resources over a network, for example, provides a way for a user to discover a Url advertised by a Bluetooth Low Energy peripheral and then download it on their smartphone.

3.1 Beacon advertising data

The format for the Advertising Data that includes the <<Service UUID>> and <<Service Data>> basic types is shown in in Figure 3.1 and Table 3.2.

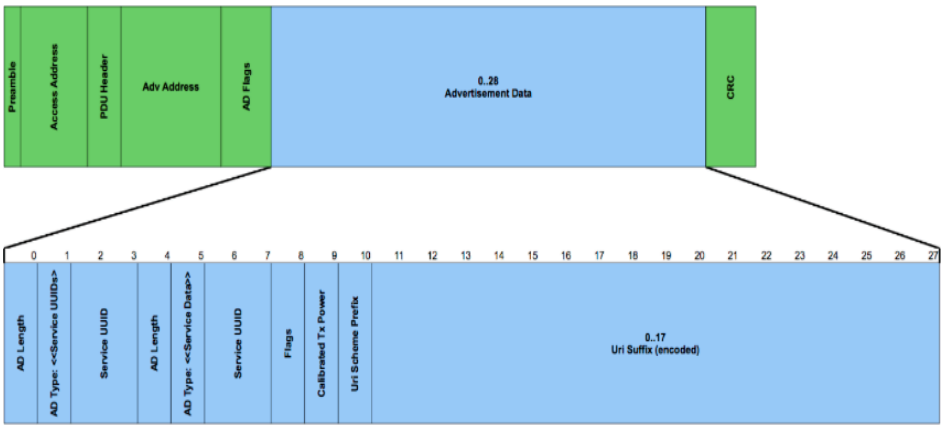

Figure 3-1 Beacon Advertisement data format

The Uri Scheme Prefix byte defines the identifier scheme, an optional prefix and how the remainder of the Uri is encoded as seen below in table 3-2.

Table 3-2 Beacon Scheme Prefix

Decimal	Hex	Expansion
0	0x00	http://www.
1	0x01	https://www.
2	0x02	http://
3	0x03	https://

Chapter 4

Implementation on TI Sensor Tag

Software developed using the BLE software development kit consists of five major sections: the OSAL, HAL, the BLE Protocol Stack, profiles, and the application. The BLE protocol stack is provided as object code, while the OSAL and HAL code is provided as full source. In addition, three GAP profiles (peripheral role, central role, and peripheral bond manager) are provided, as well as several sample GATT profiles and applications.

4.1 Operating System Abstraction Layer (OSAL)

The BLE protocol stack, the profiles, and all applications are all built around the Operating System Abstraction Layer (OSAL). The OSAL is not an actual operating system (OS) in the traditional sense, but rather a control loop that allows software to setup the execution of events. For each layer of software that requires this type of control, a task identifier (ID) must be created, a task initialization routine must be defined and added to the OSAL initialization, and an event processing routine must be defined. Optionally, a message processing routine may be defined as well. Several layers of the BLE stack, for example, are OSAL tasks, with the LL being the highest priority (since it has very strict timing requirements).

In addition to task management, the OSAL provides additional services such as message passing, memory management, and timers. All OSAL code is provided as full source.

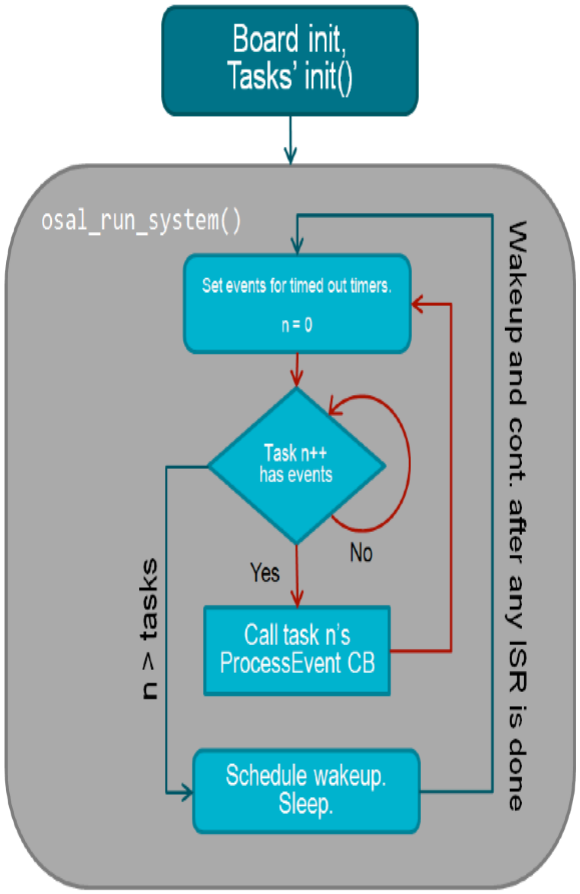

Figure 4-1 OSAL Task Loop

4.1.1 Task Initialization

In order to use the OSAL, at the end of the main function there should be a call to osal_start_system. Each layer of software that is using the OSAL must have an initialization routine that is called from the function osalInitTasks. Within this function, the initialization routine for every layer of software is called. As each task initialization routine is called, an 8-bit "task ID" value is assigned to the task. Note that when creating an application, it is very important that it be added to the end of the list, such that it has a higher task ID than the others. This is because the priority of

tasks is determined by the task ID, with a lower value meaning higher priority. It is important that the protocol stack tasks have the highest priority in order to function properly. Such is the case with the SimpleBLEPeripheral application: its initialization function is SimpleBLEPeripheral_Init, and it has the highest task ID and therefore the lowest priority.

4.1.2 Task Events and Event Processing

After the OSAL completes initialization, it runs the executive loop checking for task events. This loop can be found in the function osal_start_system in the file OSAL.c. Task events are implemented as a 16-bit variable (one for each task) where each bit corresponds to a unique event. The definition and use of these event flags is completely up to the application.

When the OSAL detects an event for a task, it will call that task's event processing routine. The layer must add its event processing routine to the table formed by the array of function pointers called tasksArr. You will notice that the order of the event processing routines in tasksArr is identical to the order of task ID's in the osalInitTasks function. This is required in order for events to be processed by the correct software layer.

In the case of the SimpleBLEPeripheral application, the function is called SimpleBLEPeripheral_ProcessEvent. Note that once the event is handled and if it is not removed from the event flag, the OSAL will continue to call the task's process event handler. As can be seen in the SimpleBLEPeripheral application function SimpleBLEPeripheral_ProcessEvent, after the START_DEVICE_EVT event occurs, it returns the 16-bit events variable with the SBP_START_DEVICE_EVT flag cleared.

It is possible for any layer of the software to set an OSAL event for any other layer, as well as for itself. The simplest way to set up an OSAL event is to use the osal_set_event function (prototype in OSAL.h), which immediately schedules a new event. With this function, you specify the task ID (of the task that will be processing the event) and the event flag as parameters.

Another way to set an OSAL event for any layer is to use the osal_start_timerEx function (prototype in OSAL_Timers.h). This function operates just like the osal_set_event function. You select task ID of the task that will be processing the event and the event flag as parameters; however for a third parameter in osal_start_timerEx you input a timeout value in milliseconds. The OSAL will set a timer, and the specified event will not get set until the timer expires.

4.1.3 Heap Manager

OSAL provides basic memory management functions. The osal_mem_alloc function serves as a basic memory allocation function similar to the standard C malloc function, taking a single parameter determining the number of bytes to allocate, and returning a void pointer. If no memory is available, a NULL pointer will be returned. Similarly, the osal_mem_free function works similar to the standard C free function, freeing up memory that was previously allocated using osal_mem_alloc.

The pre-processor define INT_HEAP_LEN is used to reserve memory for dynamic allocation. To see how much memory you typically need, you can set the pre-processor define OSALMEM_METRICS=TRUE in the project options. After a stress test of the application where you send as many messages, have as many clients as you will in the worst case, remembering to use bonding and encryption during the test if that's applicable, you can look at the value of the variable memMax in OSAL_Memory.c to see how much memory was ever allocated at the same time. This figure could be used as a guideline for lowering INT_HEAP_LEN if necessary, but thorough testing is needed, as the heap is used by the BLE stack.

4.1.4 OSAL Messages

OSAL also provides a system for different subsystems of the software to communicate with each other by sending or receiving messages. Messages can contain any type of data and can

be any size. To send an OSAL message, first the memory must be allocated by calling the osal_msg_allocate function, passing in the length of the message as the only parameter. A pointer to a buffer containing the allocated space will be returned (you do not need to use osal_mem_alloc when using osal_msg_allocate). If no memory is available, a NULL pointer will be returned. You can then copy the data into the buffer. To send the message, the osal_msg_send should be called, with the destination task for the message indicated as a parameter.

The OSAL will then signal the receiving task that a message is arriving by setting the SYS_EVENT_MSG flag for that task. This causes the receiving task's event handler function to be called. The receiving task can then retrieve the data by calling osal_msg_receive, and can process accordingly based on the data received. It is recommended that every OSAL task have a local message processing function (the simpleBLEPeripheral application's message processing function is simpleBLEPeripheral_ProcessOSALMsg) that decides what action to take based on the type of message received. Once the receiving task has completed processing the message, it must deallocate the memory using the function osal_msg_deallocate (you do not need to use osal_mem_free when using osal_msg_deallocate).

4.2 BLE Protocol Stack

The entire BLE protocol stack is provided as object code in a single library file (Texas Instruments does not provide the protocol stack source code as a matter of policy). The functionality of the GAP and GATT layers should be understood as they interact directly with the application and profiles.

4.2.1 Generic Access Profile (GAP)

The GAP layer of the BLE Protocol Stack is responsible for handling the device's access modes and procedures, including device discovery, link establishment, link termination, initiation of security features, and device configuration. This advertisement contains the device address, and can contain some additional data as well, such as the device name. The central device, upon receiving the advertisement, sends a "scan request" to the peripheral. The peripheral responds with a "scan response". This is the process of device discovery, in that the central device is now aware of the peripheral device, and knows that it can form a connection with it. The central device can then send out a request to establish a link with the peripheral device. A connection request contains a few connection parameters:

Connection Interval – In a BLE connection between two devices, a frequency-hopping scheme is used, in that the two devices each send and receive data from one another on a specific channel, then "meet" at a new channel (the link layer of the BLE stack handles the channel switching) at a specific amount of time later. This "meeting" where the two devices send and receive data is known as a "connection event". Even if there is no application data to be sent or received, the two devices will still exchange link layer data to maintain the connection. The connection interval is the amount of time between two connection events, in units of 1.25ms. The connection interval can range from a minimum value of 6 (7.5ms) to a maximum of 3200 (4.0s). Different applications may require different connection intervals. The advantage of having a very long connection interval is that significant power is saved, since the device can sleep most of the time between connection events. The disadvantage is that if a device has data that it needs to send, it must wait until the next connection event. The advantage of having a very short connection interval is that there is more opportunity for data to be sent or received, as the two

devices will connect more frequently. The disadvantage is that more power will be consumed, since the device is frequently waking up for connection events.

Slave Latency – This parameter gives the slave (peripheral) device the option of skipping a number of connection events. This gives the peripheral device some flexibility, in that if it does not have any data to send it can choose to skip connection events and stay asleep, thus providing some power savings. The decision is up to the peripheral device. The slave latency value represents the maximum number of events that can be skipped. It can range from a minimum value of 0 (meaning that no connection events can be skipped) to a maximum of 499; however the maximum value must not make the effective connection interval (see below) greater than 16.0s.

Supervision Timeout – This is the maximum amount of time between two successful connection events. If this amount of time passes without a successful connection event, the device is to consider the connection lost, and return to an unconnected state. This parameter value is represented in units of 10ms. The supervision timeout value can range from a minimum of 10 (100ms) to 3200 (32.0s). In addition, the timeout must be larger than the effective connection interval (explained below). The “effective connection interval” is equal to the amount of time between two connection events, assuming that the slave skips the maximum number of possible events if slave latency is allowed (the effective connection interval is equal to the actual connection interval if slave latency is set to zero). It can be calculated using the formula:

Effective Connection Interval = (Connection Interval) * (1 + (Slave Latency))

Connection Interval: 80 (100ms)

Slave Latency: 4

Effective Connection Interval: (100ms) * (1 + 4) = 500ms

This tells us that in a situation in which no data is being sent from the slave to the master, the slave will only transmit during a connection event once every 500ms.

Chapter 5

Survey of Optimum Connection Interval and Data length

To get the right amount of data lenth and connection interval for our URL transmission. I had to manually test the devices with 3 different beacons and varying parameters, finally to obtain the right value .

First I did a survey for the connection interval against the Battery life. I took 3 fesh Lithium batteries which are used for TI Sensortag and had it installed in those beacons and then Started the Survey by having a Connection interval of 1600ms. The results of that test and and the other following test are ploted in the Chart 5-1 below. Here I have taken fresh batteries for every experiment. So finally I concluded that a Connection interval of 16ms would be optimum for our usage and purpose.

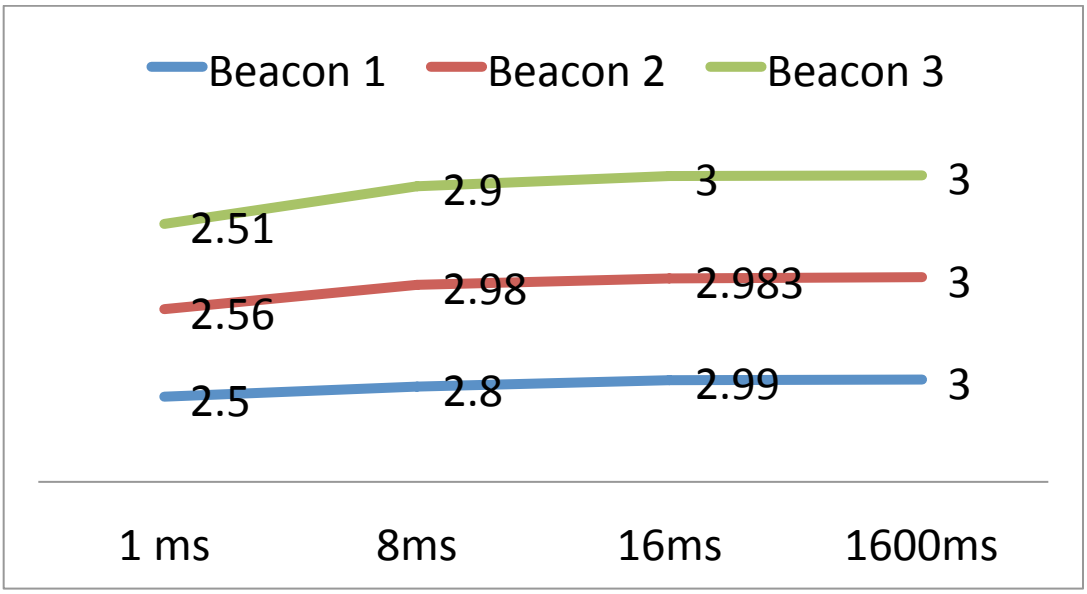

Figure 5-1 Connection Interval vs Battery Life

Secondly a survey for the data length against the Battery life. I took 3 fesh Lithium batteries which are used for TI Sensortag and had it installed in those beacons and then Started the Survey by having a data Length 4Bytes. The results of that test and and the other following test are ploted in the Chart 5-2 below. Here I have taken fresh batteries for every experiment. So finally I concluded that a Data length is not a major factor for the battery life.

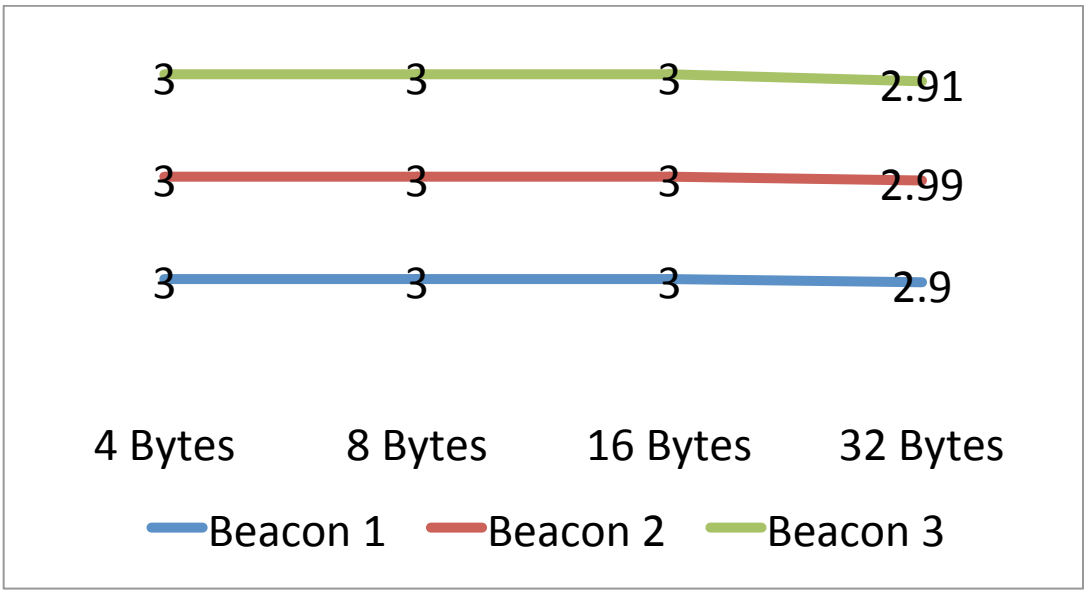

Figure 5-1 Data length vs Battery Life

Chapter 6

Conclusion

This concept will be a major boost in terms of user interaction. A retail shop however small or big can use it without much of overhead expenses. It can transmit personalized deals based on the products available in the retail store and can advertise its products just by placing a beacon and pushing some of the its offers.

We can use small hardware devices or tags to implement it and it is a open source. Security not much in this field but security can be implemented with some web authentication mechanism or by using obfuscated URL.Also you can use the mDNS method for this, Only people logged into your wifi can see the mDNS URLs. This means that in an appartment, your neighbors can't see your devices.

6.1 Possible Uses

Parking meters can pay in the cloud using the phone's internet connection. Any store, no matter how small, can offer an online experience when you walk in. Industrial equipment can offer diagnostics. We can use it in healthcare system to monitor the heart rate of the Patient. In muesuem for a better interaction and to provide the history of painting or any artifacts displayed with a personilsed beacontransmitting the required data to the mobile device.

6.2 URL Short

With the help of URL shot technique you can store and send any url with what so ever length but in a shortened format format making it work with the utmost efficiency. This technique has been embeded into the application. On transmiting of the URL the application will fetch for the

information from the web. It will also display the Favicon, Title, Description along with the URL itself.

There is still lot of research work left on this regard, as this technology is in its initial stage and yet to be discovered and well know to the public. BLE has the potential to grow and be implemented in almost all electronic device.

References

[1] http://www.ibeacon.com/apples-ibeacon-future-mobile-shopping/

[2] http://www.ti.com/tool/cc2541dk-sensor

[3] http://www.ti.com/product/cc2541

[4] http://www.ti.com/tool/cc2541sensortag-rd

[5] http://www.ti.com/lit/df/swrr125/swrr125.pdf

[6] Bh. Krishnamachari, "Networking wireless Sensors" Cambridge University Press 2005

[7] https://developer.android.com/guide/topics/connectivity/bluetooth-le.html

[8] http://www.ti.com/lit/ug/swru271f/swru271f.pdf

[9] Bh. Krishnamachari, "Networking wireless Sensors" Cambridge University Press 2005

[10] A. Akyildiz, "A survey on sensor networks," IEEE Commun. Mag., vol. 40, pp. 102-114, Aug. 2002.

[11] Waltengenus Dargie and Christian Poellabauer, "Fundamentals of Wireless Sensors and Networks 2010 John Wiley and Sons Ltd

[12] D. Niculescu, "Positioning in ad hoc sensor networks," IEEE Network, vol. 18, pp. 24-29,Jul 2000

Biographical Information

Arjun Kumar Bhaskar Shetty was born in Karnataka, India in 1987. He successfully completed his Bachelor of Engineering in Computer Science in Vishveshwaraya Institute of Technology, India in 2010. He enrolled for master's program in computer science at the University of Texas at Arlington after working for two years in AT&T, Bengaluru. His areas of interest are mainly focused on Computer network and as Network Analyst. He has also worked on the Android development and database related projects.