Masinde Masinde

IoT

Implementing a Sensor-Tag to Embedded Linux Platform

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

19 December 2016

Helsinki Metropolia University of Applied Sciences

Author	Masinde Mtesigwa Masinde
Title	Implementing a Sensor-Tag to Embedded Linux Platform
Number of Pages	31 pages + 1 appendix
Date	19 December 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Embedded Software Engineering
Instructor	Sami Sainio, Supervisor

The purpose of the final year project was to create a connection between a Texas Instrument sensor tag and an embedded Linux platform, and implement a standalone application connected to the internet to upload sensor data to the cloud. Raspberry Pi 2 was the device used as the embedded Linux platform. The application was built using IBM Bluemix cloud and USB BLE (Bluetooth Low Energy) for connection between Raspberry Pi 2 and the sensor tag. The Wi-Fi Dongle was used for internet connection between the Raspberry Pi 2 and the IBM Bluemix cloud. The data of Texas Instrument Sensor Tag CC2541 was visualized in real time on the cloud. The sensors on the tag include IR temperature, humidity, barometer, accelerometer and magnetometer sensors. The theoretical part of the work explores the sensor implementation and cloud services.

The IoT application of this project can be used by the open source community for further development of similar open source projects. They can make related applications using the wireless sensors in embedded Linux platforms. Special attention should be paid to choosing the devices to make sure that they fit the project. A successful project can be achieved through solving and avoiding the drawbacks presented in this study. The suggested solution is to use a different development board, which has differt architecture supported by Cloud foundry.

Keywords	TI CC2541, Raspberry Pi 2, Embedded Linux, IBM Bluemix, BLE

Contents

Appendices

List of Abbreviations

AWS	Amazon Web Services
BLE	Bluetooth Low Energy
HCI	Host Controller Interface
IaaS	Infrastructure as a Service
IBM	International Business Machine
IP	Internet Protocol
GAP	Generic Access Profile
GATT	Generic Attribute Profile
LAN	Local Area Network
M2M	Machine to Machine
MCU	Multipoint Control Unit
MQTT	Message Queuing Telemetry Transport
NoSQL	non-SQL
PaaS	Platform as a Service
PC	Personal computer
ScPP	Scan Parameters Profile
ScPS	Scan Parameters Service
SSH	Secure shell
SQL	Structured Query Language
TIP	Time Profile
USB	Universal Serial Bus

1 Introduction

The purpose of this thesis is to explore embedded systems and the sensors used in embedded systems. The aim is to describe how a sensor tag was implemented on an embedded Linux platform for the open source community. The objective was to create a stand-alone application that uploads sensor data on the cloud.

In recent years, embedded Linux devices have gained economic advantage competing with the traditional embedded devices in the market. This includes some telecommunication technologies with the Linux platform. In addition, health service devices, transport technologies and household items such as TVs, microwaves and many more are often operating under embedded Linux.

Linux technology has become popular due to its high performance, stability and deterministic nature. Most Linux technologies are open source which attracts many software developers to contribute to various Linux projects. The vastness of ideas from different users makes the embedded Linux and Linux technologies to grow fast. This is due to Open Source and General Public License (GPL) which give the developers freedom to share the source codes to the public. The users or developers have the freedom of modifying the source code and reusing the code for advanced features or as a stepping stone for another project. [1.]

Thus, the research question for the final year project discussed in this thesis was

> How to connect Texas Instrument wireless sensors to embedded Linux platform and send the data into the cloud in real time.

The main objective of the project was to implement an application which can display time series data in real time on the cloud using an embedded Linux platform as the gateway.

This thesis is in six sections. The first section is the Introduction to the topic. Section 2 describes the theoretical background of the embedded Linux platform Raspberry Pi 2, Texas Instrument Sensor Tag CC2541, Bluetooth dongle and Wi-Fi dongle. Section 3

describes application design and project implementation. Section 4 describes application testing. Section 5 describes results and discussion. Lastly, section 6 presents the conclusions and recommendations.

2 Theoretical Background of the Project Devices

The final year project used Raspberry Pi 2 embedded Linux platform and Texas Instrument Sensor Tag TI CC2541. The following subsections describe these two devices, Wi-Fi Dongle, Bluetooth Dongle and their features in detail.

2.1 TI CC2541 Sensor Tag

The TI CC 2541 sensor tag is focused on wireless sensor applications, and the tag is often used by the mobile phone application developers. Texas Instrument used BLE technology is developing a sensor tag hardware platform, whereby all available sensors are on a single board for quick evaluation and demonstration. The sensor drivers are on a GATT server. The server contains all the main services for every sensor available on the BLE stack. [2.] Figure 1 below shows the TI CC2541 sensor tag used in the final year project.

Figure 1. TI CC2541 Sensor Tag. Reprinted from Texas Instrument [2].

2.1.1 Types of Sensors on the Sensor Tag

The TI CC2541 Sensor Tag has five sensors which can be viewed on the development board. Figure 2 shows the sensors which were used in the project.

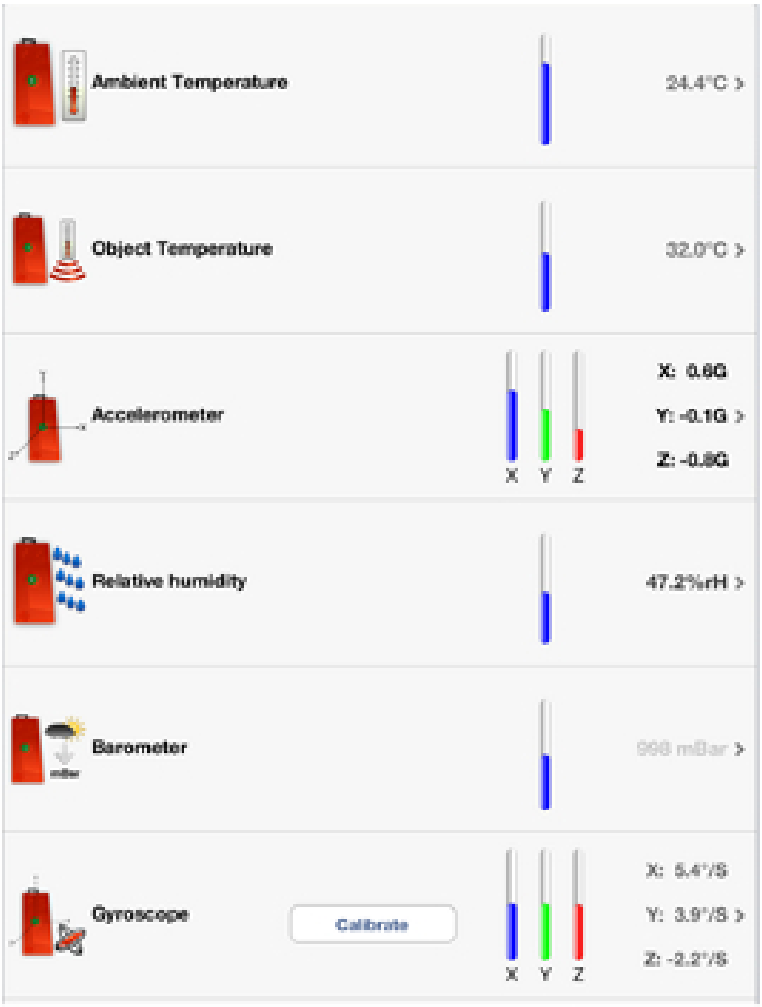

Figure 2. Sensors in the final year project. Reprinted from Texas Instrument [2].

The sensors used for this final year project include

- IR temperature Sensor (ambient and objective)
- Humidity Sensor (relative and humidity temperature)
- Barometer Sensor (pressure and temperature)
- Accelerometer, 3 axis
- Magnetometer, 3 axis
- Gyroscope, axis

2.1.2 Operations

The TI CC 2541 sensor tag advertises its frequency of 100 m/s[2]. Connection is established on the receiving device (Raspberry Pi 2 in this project) and the sensors are configured to provide the measurement data. The operation can be listed as follows

- Scan and find the sensor tag
- Establish connection
- Make service discovery
- Write and read the data

2.1.3 GAP

GAP, Generic Access Profile, is the Bluetooth low energy (BLE) protocol which defines the client and server actions and performance of two BLE devices. These actions and performance include the discovery of the device, establishing and termination of the connection link between the sensor tag and Raspberry Pi, initiating security features and device configuration. Figure 3 shows the possible device states. [1.]

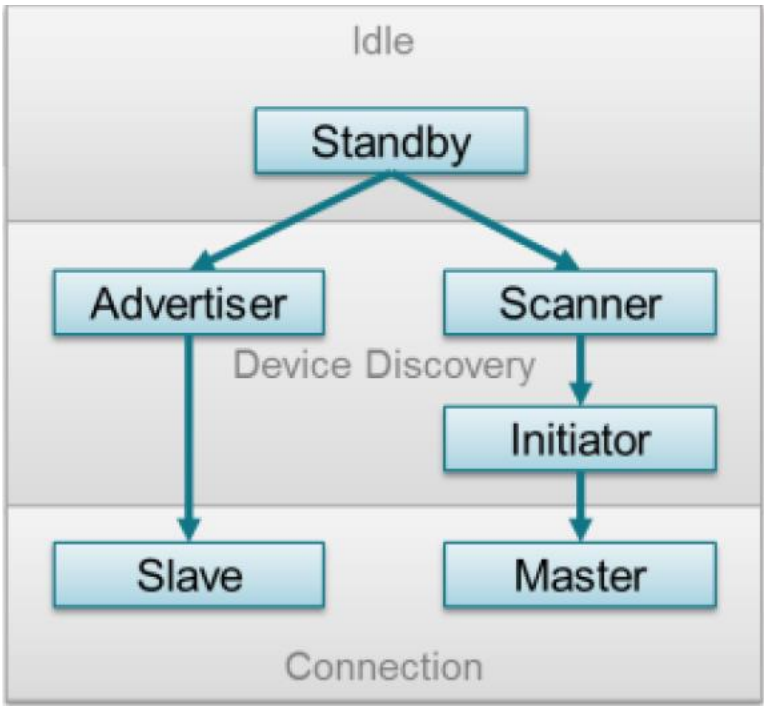

Figure 3 GAP State diagram. Reprinted from Texas Instrument [2].

2.1.4 GATT

Texas Instrument designed the GATT (Generic Attribute Profile) layer for the low energy Bluetooth protocol stack. This protocol enables the client and the server to connect. When connected, they exchange data using services and characteristics. The server stores data from the sensors written by GATT. The client requests the data so that it can be viewed and converted to human readable form by the client [2.]. Figure 4 shows the request and response between the client and the server [2.]

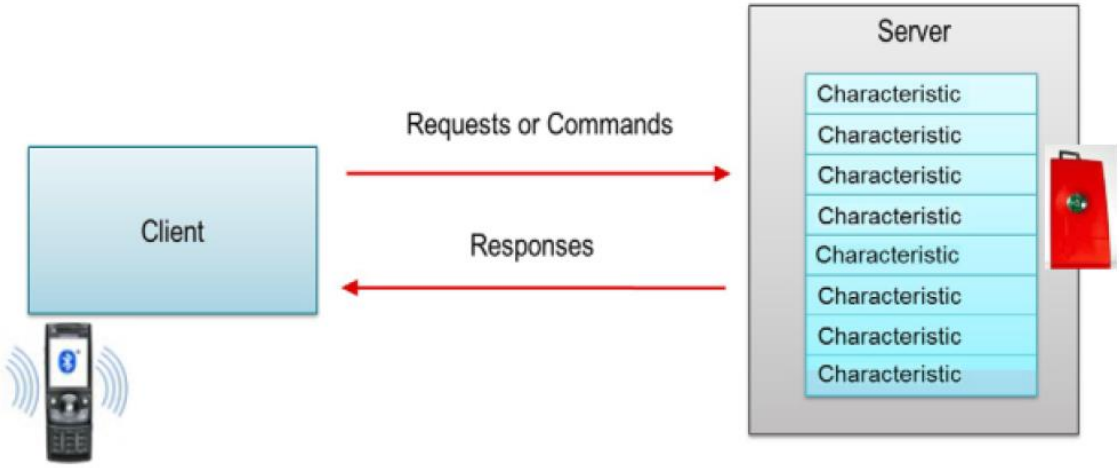

Figure 4. GATT Client and Server. Reprinted from Texas Instrument [2].

2.1.5 Services and Characteristics

GATT services in BLE are based on objects called profiles, services and characteristics. A profile is the simple collection of ready defined services by the Bluetooth vendors. The services include ScPP (Scan Parameters Profile) which defines how the client can write its data to a server and how the server can request the data from the client [3.]. Figure 5 shows the service and characteristics.

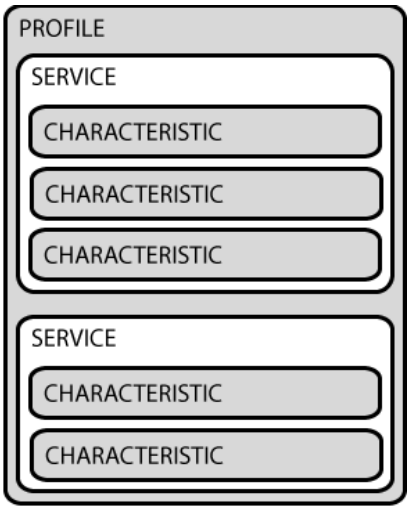

Figure 5. Service and characteristics. Reprinted from Texas Instrument [2].

2.2 BLE Technology on TI CC2541 Sensor Tag

BLE is a low power solution for controlling and monitoring application technology which was released by Bluetooth Interest Group. It provides attractive energy performance that makes it particularly suitable for portable, battery-driven electronic devices. [3]

Sensor tag initiating mode is known as a scanner or initiator, which scans periodically advertising channels to get information of other devices. On the other device which is Raspberry Pi 2 in this final year project, where the BLE device is operating in advertising mode which is known as advertiser, it periodically transmits advertising information in three channels. When there is connection with the initiator they will establish a connection. [3.]

2.3 Raspberry Pi 2

The project required an embedded Linux board for the development of the application. The embedded Linux board which was chosen was Raspberry Pi 2 model-B. The reason for choosing this Raspberry Pi 2 model B was due to its performance power. The Pi has four processors which draw high power resulting to high performance. The type of processor is ARMv7 which is multicore processor. The Raspberry Pi 2 also has 1 GB of ram which makes it even more powerful. [4.] Figure 6 shows the Raspberry Pi 2-Model B which was used in the project.

Figure 6. Raspberry Pi 2-model B. Reprinted from Adafruit [4].

2.4 Wireless Dongle for Internet Connection

To have connection between the development board and the cloud and to install necessary drivers, there is supposed to be network connection. A wireless dongle was the choice for this project because it helps in moving the development board from one place to another during the testing. The dongle which was chosen was TP-Link which has capability of transmitting 300Mbps. [5]. Figure 7 shows the wireless dongle used in the project.

Figure 7. TP-Link wireless dongle. Reprinted from TP-Link [5].

2.5 Bluetooth Dongle for Device Connections

The connection between Raspberry Pi 2 and TI CC2541 requires a USB Bluetooth which has the capability to connect to low energy devices. Ubiquitous computing devices are often small in size and they are power constrained, which makes it impossible to be connected directly to the internet. In the project, two Bluetooth dongles were chosen. The first one, USB Mini Bluetooth v2.1dongle failed because it did not have the low energy connection capability. [6.] Figure 8 shows USB Mini Bluetooth v2.1 dongle.

Figure 8. USB Mini Bluetooth v2.1 dongle. Reprinted from Konig [6].

Asus USB Bluetooth was the second Bluetooth dongle trialled to connect to the sensor tag. It was a trial because not all Bluetooth version 4 have the capability of connecting to BLE devices. This Bluetooth dongle was successfully able to connect to the sensor tag [8.]. Figure 9 shows ASUS USB-BT400 which was used.

Figure 9. USB-BT400 dongle. Reprinted from ASUS [7].

3 Application Design and Project Implementation

The final year project application was designed using the following devices and softwares:

- Raspberry Pi 2 as the development board and as the gateway to the cloud
- Wireless dongle for connection between the development board and the cloud services
- Bluetooth dongle for connection between the development board and the Sensor Tag
- TI CC2541 sensor tag as a source of sensor data
- Cloud service for data visualization and computing
- MQTT for machine to machine connection
- Node.JS for application building
- IBM Bluemix IoT Sensor Tag repository

Fgure 10 below shows the project application prototype and all the technologies involved.

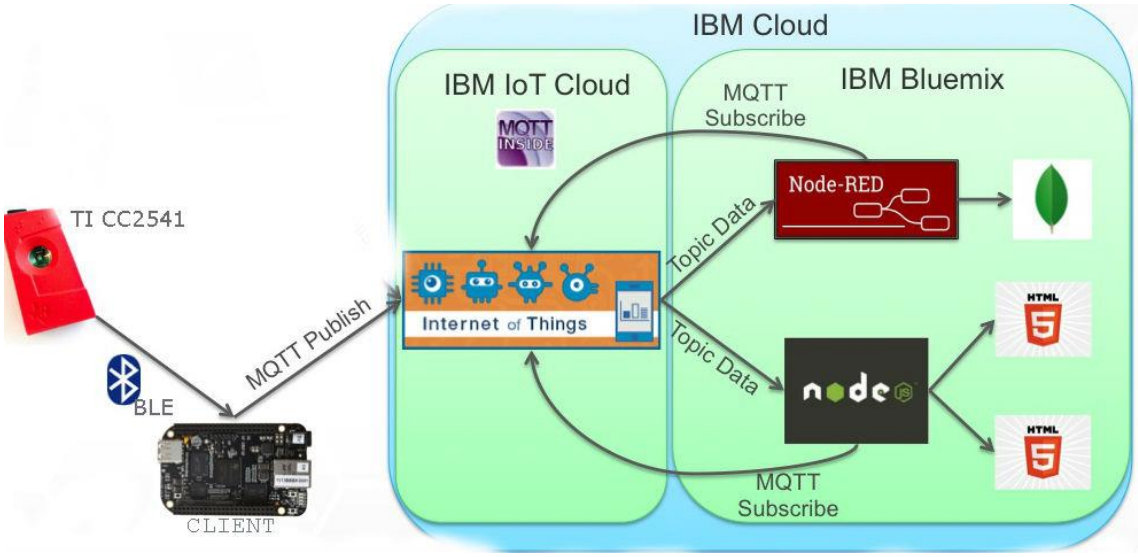

Figure 10. Project prototype. Modified from IBM [9].

Figure 11 shows the components which were used in the final year project, which are Raspberry Pi 2, TI CC2541, a wireless dongle, BLE dongle and a power bank.

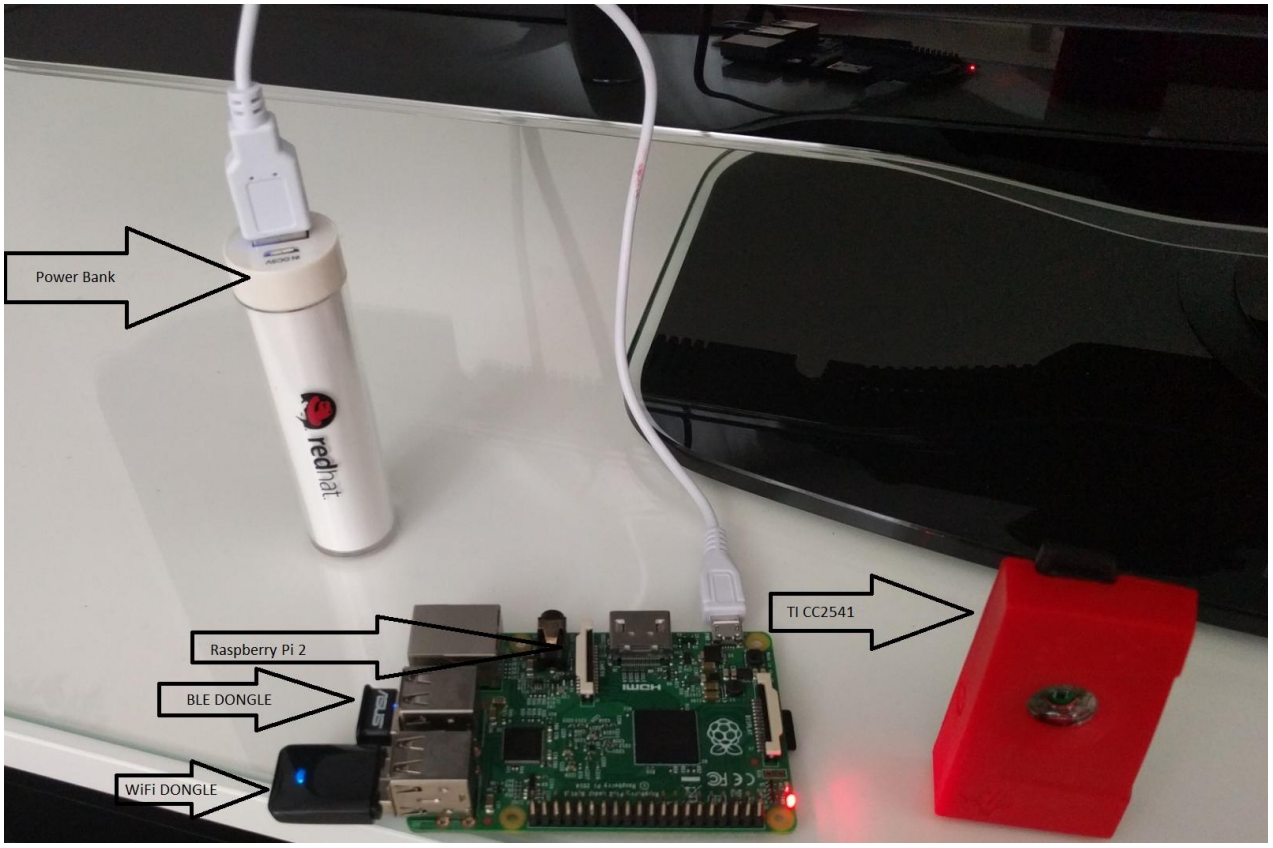

Figure 11. Components used in this project.

3.1 Raspberry Pi 2 Setup

Debian Jessie was installed using the SD card in order to setup the project environment on the Raspberry Pi 2. After that, BLE drivers were installed to enable the BLE dongle to work. The next step was to test the BLE drivers. BLE USB was inserted to the development board and configured up for listening to advertising channels. Figure 12 shows the command used to power up the BLE USB, also the command for scanning the BLE devices in the nearby areas.

```
pi@raspberrypi:~ $ sudo hciconfig hci0 down
pi@raspberrypi:~ $ sudo hciconfig hci0 up
pi@raspberrypi:~ $ sudo hcitool lescan
LE Scan ...
88:C6:26:7E:C0:6D
88:C6:26:7E:C0:6D (unknown)
78:A5:04:19:77:DB (unknown)
78:A5:04:19:77:DB SensorTag
D0:03:4B:1D:1B:CD (unknown)
D0:03:4B:1D:1B:CD (unknown)
```

Figure 12. Command for BLE USB powering up and scanning for advertised channels.

Figure 13 shows the connection testing between TI CC2541 sensor tag and Raspberry Pi 2.

```
pi@raspberrypi:~ $ sudo gatttool -b 78:A5:04:19:77:DB -I
[78:A5:04:19:77:DB][LE]> connect
Attempting to connect to 78:A5:04:19:77:DB
Connection successful
[78:A5:04:19:77:DB][LE]>
```

Figure 13. Command for testing connection between Raspberry Pi and TI CC2541.

3.2 Cloud Vendor Selection

There are different types of cloud service providers ranging from open source to private ones. The cloud providers which were considered during the project were Microsoft Azure, AWS and Rack Space among others. The cloud provider used in the final year project was IBM Bluemix.

IBM Bluemix was chosen because it supports open source projects for the Internet of Things. For example a project called NodRed, which is used for the Internet of Things, is an open source project. [9.]

3.3 IBM Bluemix

IBM Bluemix is IBM's solution for the cloud. Bluemix is a platform as a service solution, as well as Infrastructure as a Service. In PaaS (Platform as a Service) Bluemix it is possible to develop, build, test, deploy, run and manage the application in the cloud. In IaaS (Infrastucture as a Service) the hardware, storage and network can be managed on the cloud.

Bluemix supports building of the application using private infrastructure. This is possible only when the virtualization layer used is Bluemix. There are various types of virtualization layers which stay on top of Bluemix infrastructure. Bluemix supports those and make the applications deployable. These include openstack virtual machines and Docker containers which are machines and Cloud Foundry. [10.]

Cloud Foundry was used in this project. Cloud Foundry is a platform as a service. Cloud foundry is open source software which allows developers to code in multiple languages such as Java, Node.js Go, PHP and Ruby. [11.]

The procedures in the final year project for using the IBM Bluemix were creating an account, choosing the IoT starter platform, filling in the necessary credentials such as name of the application, organisation name, choosing the region where the services are available, creating an application, registering the gateway device, adding the registered device to the application and finally testing the device.

3.3.1 Creating the Account in Bluemix

The Bluemix account can be crated on the Bluemix website www.ibm.com/Bluemix. [10.] After creating the account in Bluemix, the next step was to create the application via Bluemix boilerplate. In general a boilerplate is a template that includes an application, its associated runtime environment and predefined services for a domain. [10.] Figure 14 below shows the boilerplate with Bluemix services.

Figure 14 Boilerplate. Reprinted from IBM Bluemix [10].

3.3.2 Internet of Things

In the catalogue, the Internet of Things starter was chosen because the application is based on the Internet of Things. One of the services in the boilerplate was called Cloudant NoSQL database which is suitable for the application in storing the real-time data from the sensors.

The Internet of Things platform which is the hub of all things in IBM IoT was one of the services which were included in the boilerplate. It is where all the setup and management of all connected devices can be done. In the Internet of Things is the application can access the live and historical data.

Another service which was available in the boilerplate was SDK for Node.js which is used for building the server side JavaScript application. [10.]. Figure 15 below shows the boiler plate and the services which were available

Figure 15. Services available in the boilerplate. Reprinted from IBM Bluemix [10].

3.3.3 Creating an Application

The name of the created application was ThesisIoT. The hostname was chosen automatically by Bluemix after filling in the application name (which can later be changed). Organization ID also is set automatically when the application is created. The organisation ID for this project was aa7asp. The organization type was Bluemix-free and it was chosen because it offers various services compared to others. The geographic location chosen was US-SOUTH, due to the number of services which are available in the region compared to other regions.

The application is built using Node-RED. Node-RED is a powerful visual editor from IBM used for building IoT applications. [11]. Node-RED is built on top of Node.JS. Figure 16 below shows the application ready for Node-RED editing.

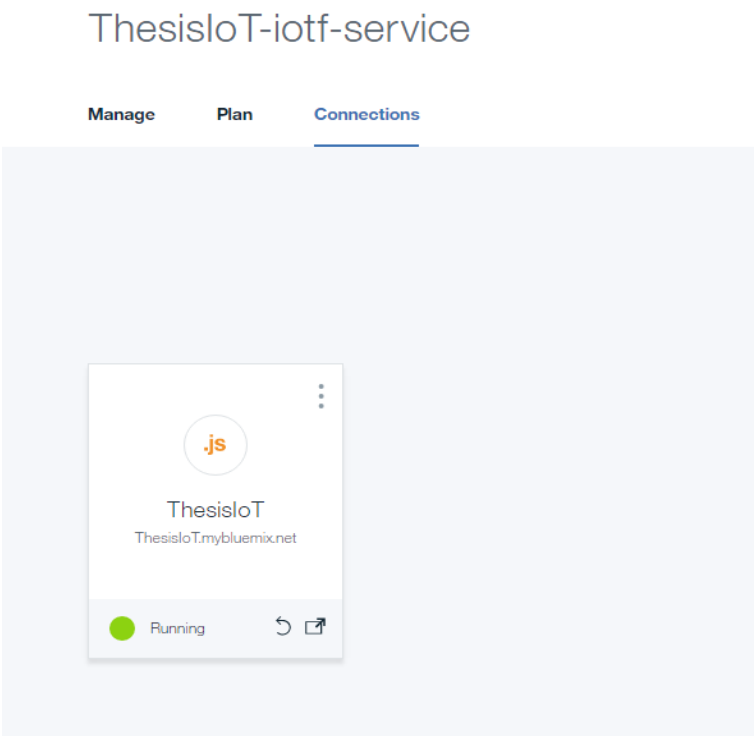

Figure 16. ThesisIoT application.

3.4 Device Registration

The device is added to IBM Watson IoT after the application has been created. In creating the device type there are two options: device type and gateway type. This application uses device type. The next step is to provide the name of the device (for this application it was Raspberry) and provide description of the device.

When the device has been created, the next step is to add the device by providing the device ID which is the Mac address of the Raspberry Pi 2. The device ID is written in a way that all colons are deleted and without space. The next step is to provide a password which is a token between the device and Bluemix for security reasons and for secure connection. After providing the device information, the next step is to copy the device information to the Raspberry Pi 2 in the folder where the application is located. The device information is copied into a file called `configure.properties`. Figure 17 below shows the device information after it has been created and added to IBM Watson IoT.

Figure 17. Device information.

3.4.1 Performance Monitoring

When the application is running, Bluemix auto-create performance monitoring messages are in Json format. The messages help to monitor the application life and help to troubleshoot when problems occur. Listing 1 below shows application performance and availability of the application.

```
{
  "iotf-service": [
    {
      "credentials": {
        "iotCredentialsIdentifier": "xxxxxxxx",
        "mqtt_host":"aa7asp.messaging.inter-
netofthings.ibmcloud.com",
        "mqtt_u_port": 1883,
        "mqtt_s_port": 8883,
        "http_host":"aa7asp.inter-
netofthings.ibmcloud.com",
        "org": "aa7asp",
        "apiKey": "xxxxxxxx",
        "apiToken": "xxxxxxxx"
      },
```

```
      "syslog_drain_url": null,
      "label": "iotf-service",
      "provider": null,
      "plan": "iotf-service-free",
      "name": "ThesisIoT-iotf-service",
      "tags": [
        "internet_of_things",
        "Internet of Things",
        "ibm_created",
        "ibm_dedicated_public"
      ]
    }
  ]
}
```


Listing 1. Performance monitoring.

3.4.2 Performance Credentials

Bluemix IoT Watson auto-creates the performance monitoring application using Json. The messages help to monitor the life of the application. Listing 2 below shows the performance credentials of the application in Bluemix IoT Watson.


```
{
  "AvailabilityMonitoring": [
    {
      "credentials": {
        "pass": "xxxxxxxxxxxx",
        "id": "71972e89-cfa5-4ee2-a71b-9cbfc651a101",
        "url": "https://perfbroker.ng.bluemix.net/1.0/cre-
dentials/71972e89-cfa5-4ee2-a71b-9cbfc651a101"
      },
      "syslog_drain_url": null,
      "label": "AvailabilityMonitoring",
      "provider": null,
      "plan": "Base",
```

```
        "name": "performance-monitoring-auto",
        "tags": [
          "ibm_created",
          "ibm_beta",
          "bluemix_extensions",
          "dev_ops"
        ]
      }
    ]
  }
```

Listing 2. Performance credentials.

3.4.3 Database Credentials

When the application is created in Bluemix, the database auto-creates the credentials for the application ready for storing data in real time. Listing 3 below shows the database credentials.

```
{
  "cloudantNoSQLDB": [
    {
      "credentials": {
        "username":"14a1383f-10c3-4881-8aa3-2d5911d67306-blue-
    mix",
        "password": "xxxxxxxx",
        "host":    "14a1383f-10c3-4881-8aa3-2d5911d67306-blue-
    mix.cloudant.com",
        "port": 443,
        "url":"https://14a1383f-10c3-4881-8aa3-2d5911d67306-
    blue-
    mix:3a4288e4ec6855f7dd818964591284b3b238aa88960377d16fd68
    974429aab3a@14a1383f-10c3-4881-8aa3-2d5911d67306-blue-
    mix.cloudant.com"
      },
      "syslog_drain_url": null,
```

```
        "label": "cloudantNoSQLDB",
        "provider": null,
        "plan": "Lite",
        "name": "ThesisIoT-cloudantNoSQLDB",
        "tags": [
          "data_management",
          "ibm_created",
          "ibm_dedicated_public"
        ]
      }
    ]
  }
```

Listing 3. Database credentials.

3.5 Node JavaScript Installation on Raspberry Pi 2

For an application to be built, Node JavaScript is supposed to be installed on the development board. In the final year project Node JavaScript was installed to the Raspberry Pi 2.

3.6 MQTT

To have connection between Raspberry Pi 2 and Bluemix Watson IoT, a MQTT protocol was needed. MQTT is a Client Server publish/subscribe messaging transport protocol. It is a light weight application protocol which is used for communication machine to machine (M2M) communication and IoT. To make communication possible a network or the internet is required. In this application wireless dongle was used to ensure the availability of the network. [12.].

3.6.1 MQTT Installation

MQTT was installed on the client which is Raspberry Pi 2. The following command installs the MQTT on the client: `npm install mqtt -g`. NPM is used because the application is built on top of Node.JS. [12.]

3.6.2 MQTT Client

The client is a publisher or a subscriber of the messages. The same device can publish and subscribe at the same time. In this project the MQTT client is Raspberry Pi 2 which has a MQTT library installed and running. The client is connected to an MQTT broker which is Bluemix Watson IoT. The connection is done through TCP/IP through Wi-Fi dongle. Figure 18 below shows the MQTT broker and clients sending and receiving messages.

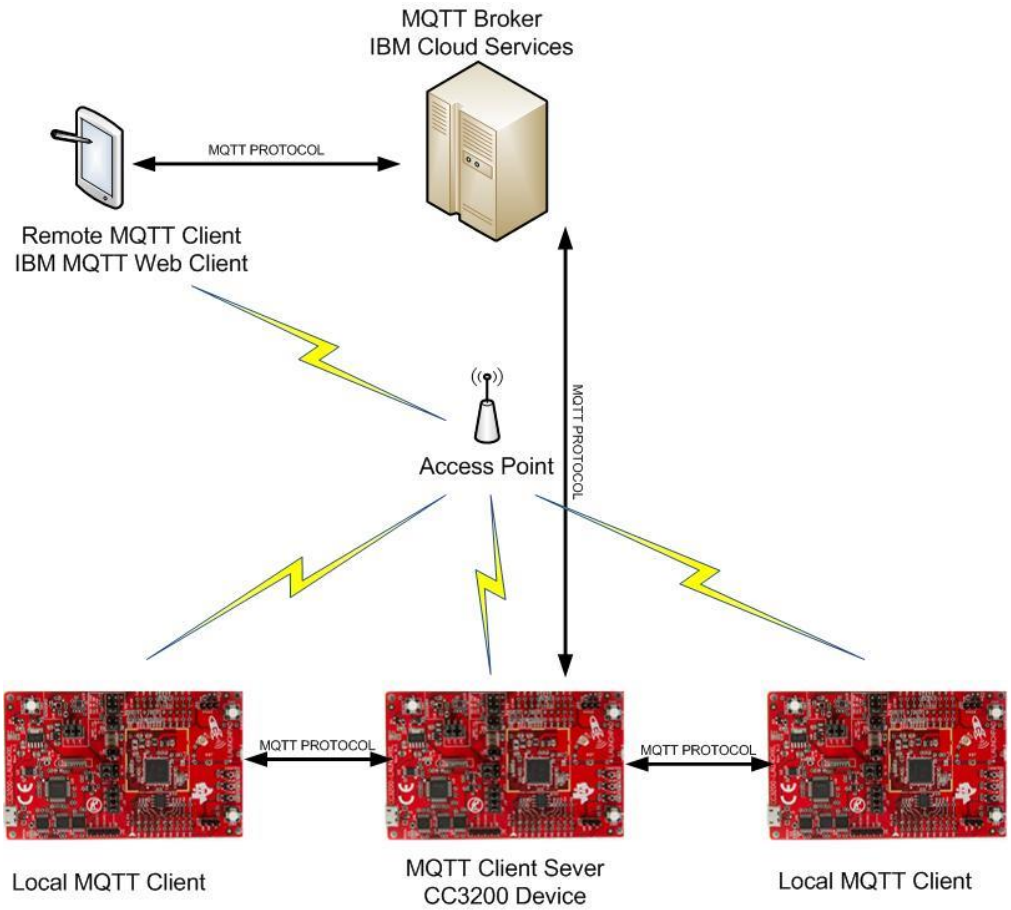

Figure 18. MQTT broker and clients. Reprinted from Texas Instrument [12].

3.6.3 MQTT Broker

The MQTT broker receives messages from the clients. In this project the broker is Bluemix Watson IoT. Bluemix Watson IoT receives all messages from the client which is Raspberry Pi 2.

Bluemix Watson IoT as the broker is responsible for the authentication and authorization of the client. The broker is the central hub, through which all the messages pass. Figure 19 below shows the MQTT broker receiving messages from the clients.

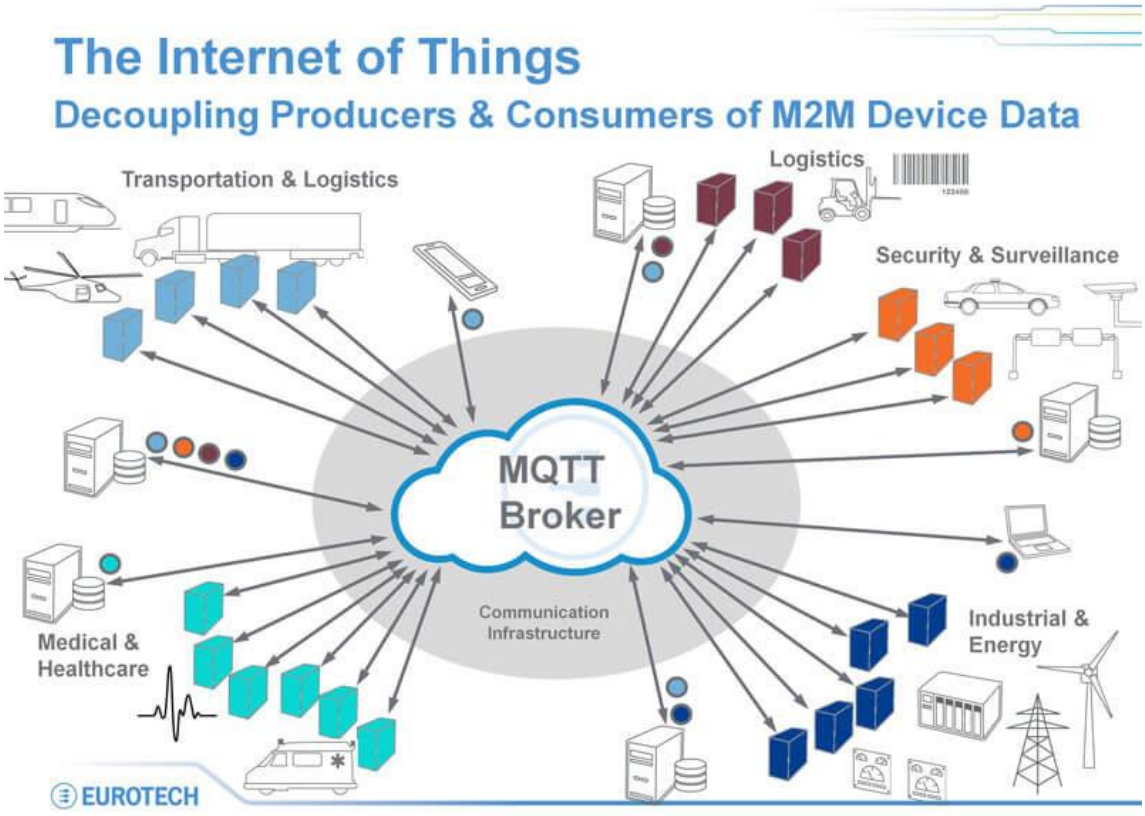

Figure 19. MQTT broker getting messages from the clients. Reprinted from Janakiram MSV [12].

3.7 IBM Bluemix IoT Sensor Tag Open Source Repository

Bluemix has the git repository in GitHub which has the source code for creating a connection between the Sensor Tag, Raspberry Pi 2 and the cloud. To get the source code from GitHub to Raspberry Pi 2, the following command was issued on the terminal

```
$ git clone git@github.com:IBM-Bluemix/iot-sensor-tag.git
```

The next step was to install Node.JS modules. After the installation of the node modules the next step was to create a config.properties file in the publish directory. The config.properties file contains the organization ID and other credentials which were created in the Blue-mix Watson IoT when registering the device. These credentials are for the connection between the client and the broker. Figure 20 show the files and the application in the publish directory.

Figure 20. The files and the application.

The application which was sensortag.js was changed to ThesisIoT since some of the sensors were only supported in the previous version of Bluemix Watson IoT and are not supported now. The original application had a lot of errors once executing. After the changes, the application started working. Figure 21 shows the client connected to the broker.

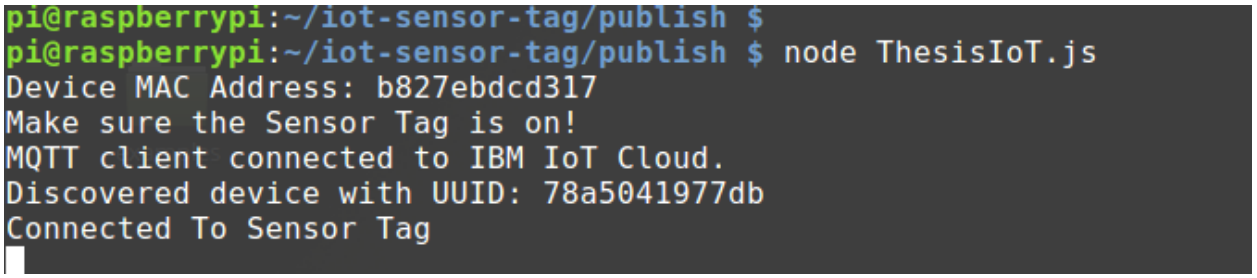

Figure 21. Client connected to the broker.

3.8 Application Deployment for Visualization

To deploy the IBM IoT Watson application as a standalone application for visualization, the installation of a cloud foundry command line in the client is required. After installing

cloud foundry command line, the next step is to install Bluemix command line in the client. Bluemix command line works on top of cloud foundry command line.

After several attempts to install cloud foundry in to Raspberry Pi 2 which runs an ARM processor, the installation failed. The failure was due to the lack of support for ARM architecture from cloud foundry organisation.

The goal of deploying the application as a standalone application for visualization was declined. The next solution was to use the graphs which are available in Watson IoT to visualize the sensor data.

4 Application Testing

The application which was created in the final year project was tested and the sensor data was visualized on the cloud. The first aim was to build two applications, one with NodeRed where messages will be sent from the sensor tag to be seen through the NodeRed[11] application, and the second one to make standalone application which will use cloudfoundry to send sensor data to the cloud for visualisation. The second aim failed because there is lack of support for arm architecture processors for Raspberry Pi 2.

The visualization of the application was built using the graphical cards of Watson IoT to view data in real time. After the application was built, the next step was to test it in different areas of a house such as living room, kitchen, bathroom and bed room. Figure 22 shows an example of visualized data from IBM Watson IoT Platform, giving information about the humidity (%), ambient temperature (°C) and air pressure (mbar) measured. Figure 23 shows how the object temperature is presented in the application.

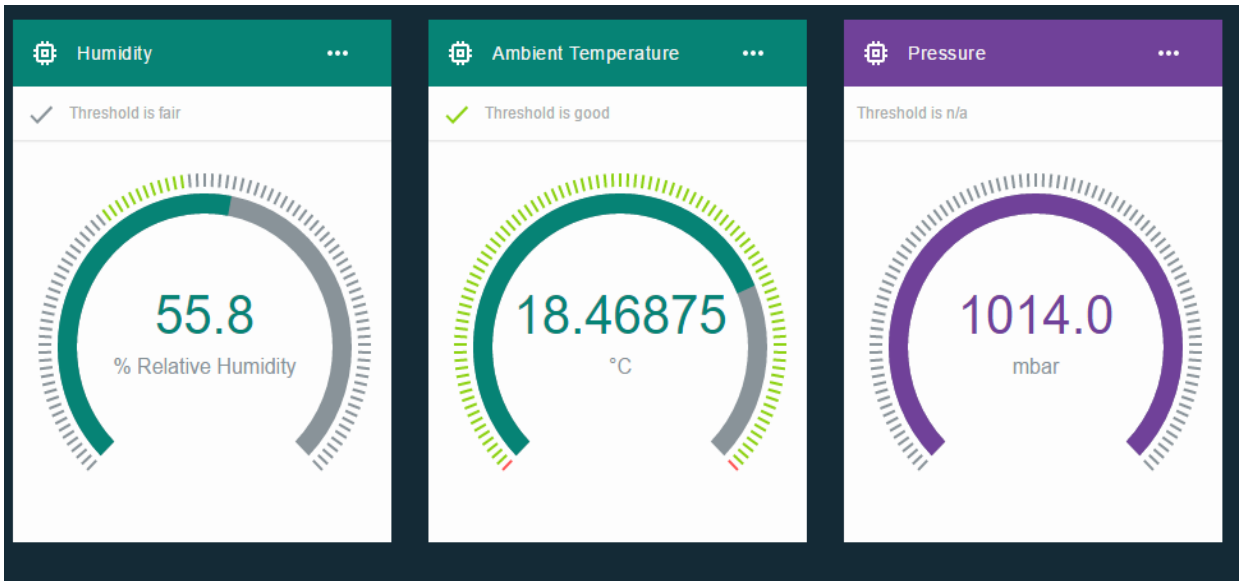

Figure 22. Humidity, ambient temperature and pressure.

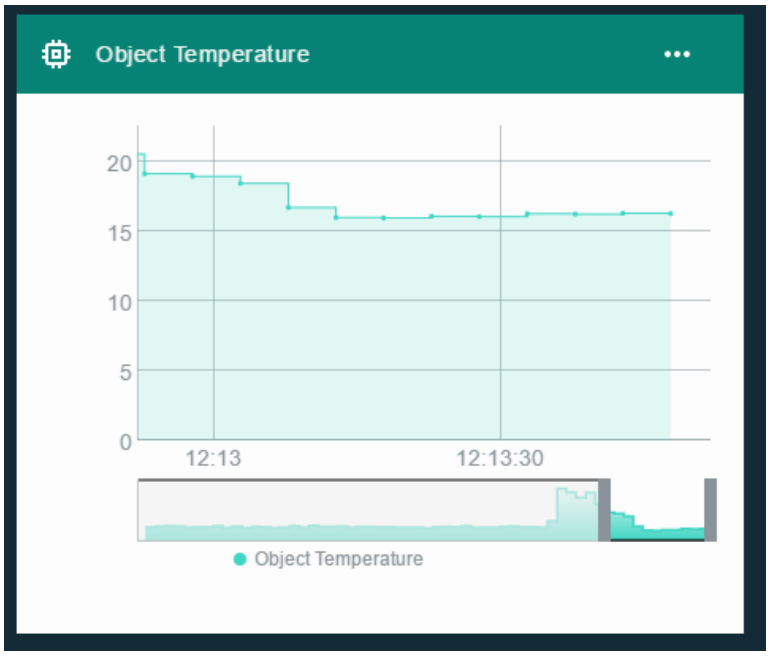

Figure 23. Object temperature.

The results of the measurements are presented in table 1.

Table 1 The results of application testing

	Temperature (°C)	Humidity (%)	Pressure (mbar)
Living room	21.9	41.7	1013.2
Kitchen	22.7	45.3	1013.2
Bathroom	18.5	55.8	1014.0
Bedroom	22.1	41.1	1013.1

5 Results and Discussion

This chapter describes the results, benefits and drawbacks of the final year project. The aim of the project was to make a standalone application which shows visualized sensor data on the cloud in real time. The standalon application was built, but it did not meet all the expectations.

5.1 Success

The final year project was successful because sensor datas were visualized in real time and in graphics. The sensor data that could not able to be visualized came from Gyro and Magnetometer. They could not able to be visualized because cloud foundry was not installed on Raspberry Pi 2 also on Watson IoT there are no graphic for the Gyro and Magnetometer sensors.

5.2 Drawbacks

The drawbacks in this project were Bluetooth Dongle, cloud foundry installation, customer support from IBM and MongoDB.

The first chosen Bluetooth Dongle needed for connection lacked the low energy technology for connection between the sensor tag and Raspberry Pi 2. Instead, another dongle was chosen that has the low energy device support.

To be able to make a standalone application for visualization, the cloud foundry virtualization layer was required in the client which is Raspberry Pi 2. The installation of the cloud foundry was not successful since Raspberry Pi 2 uses ARM processor architecture which has less support from the cloud foundry organisation. This was a huge drawback because some of the sensors could not be visualized in graphic. Such sensors were the Gyro and the Magnetometer.

During the project, there was a connection problem with the MQTT protocol in the IBM Bluemix Watson IoT and the client which is Raspberry Pi 2. The connection problem was sent to IBM in July 10, 2016 but the reply came in October 16 2016.

During the project various ways of implementing the final year project were used, MongoDB being one of the methods. Mongo DB was selected because of its ability to store real time data and the scalability of the database itself. After several attempts to install MongoDB, due to its insufficiency of support for ARM processor architecture, the database was dropped because Bluemix has its own database which is given for storing time-series data.

5.3 Suggestions

The project drawbacks met in the final year project have taught that through research or study of the components or technologies going to be used in a project before the project is started. Research will help to save time and will enable doing the project in an efficient way.

The drawbacks related to finding the right kind of devices, like the BLE USB Dongle. For example, that problem could be avoided by checking the data sheet of the USB Dongle if it supports LE (low energy) devices. The cloud foundry drawback could be avoided by making research on the development board architectures which cloud foundry supports.

6 Conclusion

The goal of the final year project was to connect sensor tag TI CC2541 to the cloud and make a standalone application where sensor data can be visualized in real time. A working application which shows visualized sensor data in real time over a network on the cloud was created.

The IoT application created in the final year project can be used for further development of similar open source projects. It can be achieved through solving the drawbacks or avoiding the drawbacks. The suggested solution is to use a different development board than Raspberry Pi 2, with different architecture supported by cloud foundry. However, because the ARM architecture has low support from cloud foundry, the chances of developing such as a final year project are limited.

References

1 The GNU General Public License v3.0 - GNU Project - Free Software Foundation. 2016. The GNU General Public License v3.0 - GNU Project - Free Software Foundation. [ONLINE] Available at: https://www.gnu.org/licenses/gpl-3.0.en.html. [Accessed 18 November 2016].

2 CC2541 SensorTag Development Kit - CC2541DK-SENSOR - TI Tool Folder (Obsolete). 2016. *CC2541 SensorTag Development Kit - CC2541DK-SENSOR - TI Tool Folder (Obsolete)*. [ONLINE] Available at: http://www.ti.com/tool/cc2541dk-sensor. [Accessed 01 April 2016].

3 Adafruit. 2014. *https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt*. [ONLINE] Available at: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt. [Accessed 22 June 2016].

4 Adafruit. 2015. *https://cdn-learn.adafruit.com*. [ONLINE] Available at: https://cdn-learn.adafruit.com. [Accessed 1 December 2016].

5 TP-LINK . 2015 [ONLINE] Available at: http://www.tp-link.com/en/products/details/cat-11_TL- WN823N.html [Accessed 9 November 2015]

6 Konig Electronics. 2015. [ONLINE] Available at: http://www.konigelectronic.com/en_us/computer/net working/550426277. [Accessed 9 November 2015].

7 *USB-BT400 | Networking | ASUS Global*. [ONLINE] Available at: https://www.asus.com/Networking/USBBT400/. [Accessed 15 May 2016].

8 GitHub. 2016. *iot-sensor-tag/sensor-tag.jpg at master · IBM-Bluemix/iot-sensor-tag · GitHub*. [ONLINE] Available at: https://github.com/IBM-Bluemix/iot-sensor-tag/blob/master/subscribe/public/sensor-tag.jpg. [Accessed 8 November 2016].

9 IBM Bluemix - Next-Generation Cloud App Development Platform. 2016. *IBM Bluemix - Next-Generation Cloud App Development Platform*. [ONLINE] Available at: https://console.ng.bluemix.net/. [Accessed 2 November 2016].

10 Cloud Foundry. 2016. *Cloud Foundry | The Industry Standard for Cloud Applications*. [ONLINE] Available at: https://www.cloudfoundry.org/. [Accessed 24 November 2016].

11 developerWorks Open. 2016. *developerWorks Open | Node-RED*. [ONLINE] Available at: https://developer.ibm.com/open/node-red/. [Accessed 18 May 2016].

12 The New Stack. 2016. *Get to Know MQTT: The Messaging Protocol for the Internet of Things - The New Stack*. [ONLINE] Available at: http://thenewstack.io/mqtt-protocol-iot/. [Accessed 20 November 2016].

Source code for connecting sensor client and the broker

```
/*
*Original author: IBM
*Modified:    Masinde Mtesigwa Masinde
*/



var SensorTag = require('sensortag');
var mqtt = require('mqtt');
var url = require('url');
var macUtil = require('getmac');
var properties = require('properties');
var connected = false;

properties.parse('./config.properties', {path: true}, func-
tion(err, cfg) {
  if (err) {
    console.error('A file named config.properties containing the
device registration from the IBM IoT Cloud is missing.');
    console.error('The file must contain the following proper-
ties: org, type, id, auth-token.');
    throw e;
  }
  macUtil.getMac(function(err, macAddress) {
    if (err) throw err;
    var deviceId = macAddress.replace(/:/gi, '');
    console.log('Device MAC Address: ' + deviceId);

    if(cfg.id != deviceId) {
        console.warn('The device MAC address does not match the
ID in the configuration file.');
    }

    var clientId = ['d', cfg.org, cfg.type, cfg.id].join(':');
```

```
    var client = mqtt.connect("mqtts://" + cfg.org + '.messag-
ing.internetofthings.ibmcloud.com:8883',
      {
        "clientId" : clientId,
        "keepalive" : 30,
        "username" : "use-token-auth",
        "password" : cfg['auth-token']
      });
    client.on('connect', function() {
          console.log('MQTT client connected to IBM IoT
Cloud.');
    });
    client.on('error', function(err) {
          console.log('client error' + err);
          process.exit(1);
    });
    client.on('close', function() {
          console.log('client closed');
          process.exit(1);
    });
    monitorSensorTag(client);
  });
   monitorSensorTag(client);
  });
});

function monitorSensorTag(client) {
  console.log('Make sure the Sensor Tag is on!');

n monitorSensorTag(client) {
  console.log('Make sure the Sensor Tag is on!');

  SensorTag.discover(function(device){
        console.log('Discovered device with UUID: ' + de-
vice['uuid']);
```

```
        device.connect(function(){
          connected = true;
          console.log('Connected To Sensor Tag');
          device.discoverServicesAndCharacteristics(func-
tion(callback){
            //getDeviceInfo();
                initAirSensors();
                initAccelAndGyro();
                initKeys();
          });
        });

        device.on('disconnect', function(onDisconnect) {
          connected = false;
          client.end();
          console.log('Device disconnected.');
        });

        function getDeviceInfo() {
          device.readDeviceName(function(callback) {
            console.log('readDeviceName: '+callback);
          });
          device.readSystemId(function(callback) {
            console.log('readSystemId: '+callback);
          });
          device.readSerialNumber(function(callback) {
                console.log('readSerialNumber: '+callback);
          });
          device.readFirmwareRevision(function(callback) {
            console.log('readFirmwareRevision: '+callback);
          });
          device.readHardwareRevision(function(callback) {
            console.log('readHardwareRevision: '+callback);
          });
          device.readSoftwareRevision(function(callback) {
                console.log('readSoftwareRevision: '+callback);
```

```
          });
          device.readManufacturerName(function(callback) {
                console.log('readManufacturerName: '+callback);
          });
        }

        function initKeys() {
          device.notifySimpleKey(function(left, right) {
          });
        };
 SensorTag.discover(function(device){
          console.log('Discovered device with UUID: ' + de
vice['uuid']);
        device.connect(function(){
          connected = true;
          console.log('Connected To Sensor Tag');
          device.discoverServicesAndCharacteristics(func-
tion(callback){
       //getDeviceInfo();
         initAirSensors();
          initAccelAndGyro();
          initKeys();
          });
        });

        device.on('disconnect', function(onDisconnect) {
          connected = false;
          client.end();
          console.log('Device disconnected.');
        });

        function getDeviceInfo() {
          device.readDeviceName(function(callback) {
            console.log('readDeviceName: '+callback);
          });
          device.readSystemId(function(callback) {
```

```
    console.log('readSystemId: '+callback);
  });
  device.readSerialNumber(function(callback) {
        console.log('readSerialNumber: '+callback);
  });
  device.readFirmwareRevision(function(callback) {
    console.log('readFirmwareRevision: '+callback);
  });
  device.readHardwareRevision(function(callback) {
    console.log('readHardwareRevision: '+callback);
  });
  device.readSoftwareRevision(function(callback) {
        console.log('readSoftwareRevision: '+callback);
  });
  device.readManufacturerName(function(callback) {
        console.log('readManufacturerName: '+callback);
  });
}
function initKeys() {
  device.notifySimpleKey(function(left, right) {
  });
};

function initAccelAndGyro() {
  device.enableAccelerometer();
  device.notifyAccelerometer(function(){});
  device.enableGyroscope();
  device.notifyGyroscope(function(){});
  device.enableMagnetometer();
  device.notifyMagnetometer(function(){});
};
device.on('gyroscopeChange', function(x, y, z) {
  var data = {
           "d": {
             "Tag": "TI Sensor Tag",
             "gyroX" : x,
```

```
                    "gyroY" : y,
                    "gyroZ" : z
                  }
                };
          client.publish('iot-2/evt/gyro/fmt/json',
JSON.stringify(data), function() {
      });
        });

        device.on('accelerometerChange', function(x, y, z) {
          var data = {
                   "d": {
                     "Tag": "TI Sensor Tag",
                     "accelX" : x,
                     "accelY" : y,
                     "accelZ" : z
                   }
                 };
          client.publish('iot-2/evt/accel/fmt/json',
JSON.stringify(data), function() {
      });
        });

        device.on('magnetometerChange', function(x, y, z) {
          var data = {
                   "d": {
                     "Tag": "TI Sensor Tag",
                     "magX" : x,
                     "magY" : y,
                     "magZ" : z
                   }
                 };
client.publish('iot-2/evt/mag/fmt/json', JSON.stringify(data),
function() {});
 });
var previousClick = {"left" : false, "right" : false};
```

```
     device.on('simpleKeyChange', function(left, right) {
        var data = {
                    "d": {
                      "Tag": "TI SensorTag",
                      "left" : false,
                      "right" : false
                     }
                   };
      if(!previousClick.left && !previousClick.right) {
        previousClick.left = left;
        previousClick.right = right;
        return;
      }
      if(previousClick.right && previousClick.left && !left &&
!right) {
        data.d.right = true;
        data.d.left = true;
      }
      if(previousClick.left && !left) {
        data.d.left = true;
      }
      if(previousClick.right && !right) {
  data.d.right = true;
      }
      previousClick.left = false;
      previousClick.right = false;

client.publish('iot-2/evt/click/fmt/json', JSON.stringify(data),
function() {});
 });
 function initAirSensors() {
device.enableIrTemperature(function(err)
{
if (err) throw err;});
 device.enableHumidity(function(err)
{
```

```
if (err) throw err;});
 device.enableBarometricPressure(function(err)
 {
 if(err) throw err;}
     var intervalId = setInterval(function() {
    if(!connected) {
       clearInterval(intervalId);
           return;
             }
   device.readBarometricPressure(function(error, pressure) {
device.readHumidity(function(error, temperature, humidity) {
device.readIrTemperature(function(error, objectTemperature,
 ambientTemperature)
 {
 var data = {"d": {"Tag": "TI CC 2541 Sensor Tag", "pressure" :
pressure,"humidity" : humidity,
"objtemp" : objectTemperature,"ambientTemp" : ambientTempera-
ture, "temp" : temperature,}
};
client.publish('iot-2/evt/air/fmt/json', JSON.stringify(data),
function() {                                                  });
 });
 });
  });
 }, 5000);
}
 });
};
```