WIA 3003 Academic Project 2

Academic Session 2019/2020, Semester 1

Energy Monitoring System for Wireless Sensor Network Device

Tan Li Han

WIC160063

Supervisor: Dr. Tey Kok Soon

Table of Content

1.0 Abstract

With the advancement in technology especially in Internet of Things (IoT). Wireless Sensor Network (WSN) devices has become more and more popular with its specific functionality and low deployment cost. However, the deployment of WSN devices also poses some problems and limitations, for instance, providing continuous service. This problem has become one of the main challenges in ensuring the reliability of WSN device. In order to overcome the problem, many improvement and research has been done, ranging from optimizing the transmission protocol, determining optimum idle time for device, researching energy harvesting method and more. Energy harvesting is one of the popular solutions for ensuring continuous power supply to the wireless devices. However, to develop an energy harvesting system, the discharge rate of the battery supply and a detailed information on the power consumption of the system is needed. This project aims to monitor the power consumption of wireless sensor network device with Arduino MKR1000 is chosen for the sensor node development. This module is mainly focus on develop back-end for the wireless sensor network device. This include develop the database to store data, data transmission between the sensor node and the server, encryption of the data and display of the data.

2.0 Introduction

Internet of Things (IoT) has brought along a lot of transformation in our daily life. The increasing growth of the mobile devices and development in the area of communication, cloud computing, embedded systems have made the IoT concept more relevant (Shete, R., 2017). With the number of devices being connected to the internet increasing exponentially each year, it is expected that the number of devices connected to the internet to reach 30.73 billion according to statista.com(n.d.).

Wireless Sensor Network (WSN) devices is one big part of IoT as it is important for big data collection. Generally, WSN can be described as a single network, which controls the unity and the environment, aiding communicates between human, machines such as computers and to its surrounding environment while WSN device is the node within the network. A node usually are made up of 4 basic components which consist of a sensor unit to sense the parameter needed, a processing unit to carry out the algorithm and processing raw data, a transceiver unit to send and receive data to other node or to the database and finally a power unit that supplies power to the node. There are also some additional units depending on the specific WSN device functionality (Akhtar, F., 2015). The relation of the units is roughly shown in Figure 1. Each unit is capable of having and handling several sub-modules as well depending on the processing power and the source of power supply.

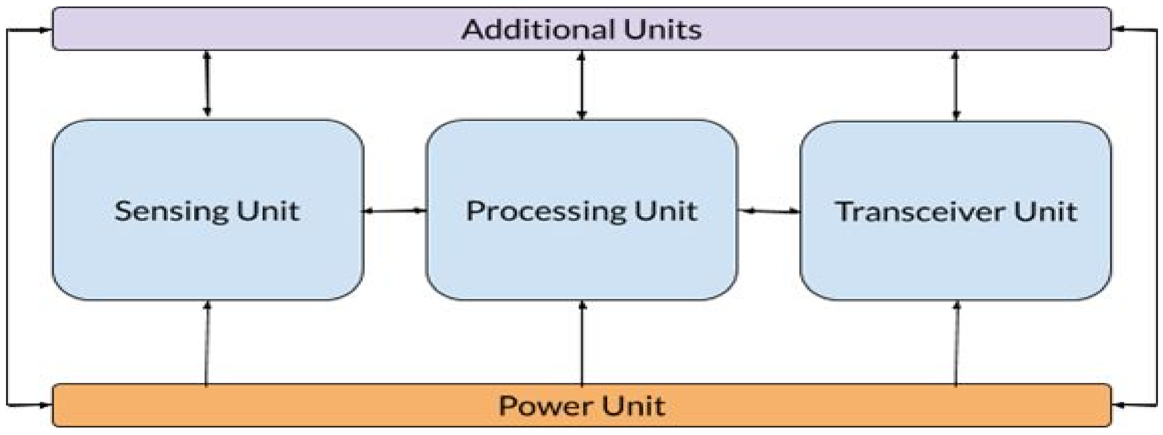


Figure 2.1: Basic component of a sensor unit

With only those components, WSN device usually share common characteristic such as it is small in size, it bears low cost and have low power usage, WSN device are also able to communicate to each other in short range. By being a low cost and small device, each node is only capable of carrying limited power source which eventually leads to limited service time. Therefore, power management, or energy management in WSN device is very crucial because with WSN device it is often the case where replacing its battery (power supply) is either too tiresome task due to its vast dispersity or it is simple not possible (Akhtar, F., 2015). For example, a WSN sensor node used to monitor volcano activity where the condition is too dangerous to undergo battery replacement. Although there are several methods to achieve energy efficiency, the recent technology trend in energy harvesting serves as a great way to prolong WSN devices' lifetime. Thus, energy harvesting is a promising approach for the emerging IoT and also to WSN system.

In light of that, to create an optimized energy harvesting system catered for a specific device, the energy consumption of the device is crucial for determining the charging rate for each device. In order to achieve that we proposed to develop an energy monitoring system for environment monitoring device. The environment monitoring device is capable of sensing environment parameter such as temperature, humidity, and dust density while the energy consumed by the system is measured using an ACS712 current sensor. The data measured will then be sent to database for further process and analysis.

3.0 Objectives

The project objectives are:

a) To develop an environment monitoring system by using MKR1000 as microcontroller.

b) To monitor power usage of environment monitoring system in idle and active state.

c) To store and display sensor retrieved data in database for analysis.

The module objectives are:

a) To improve the data security when transfer form MKR1000 to server.

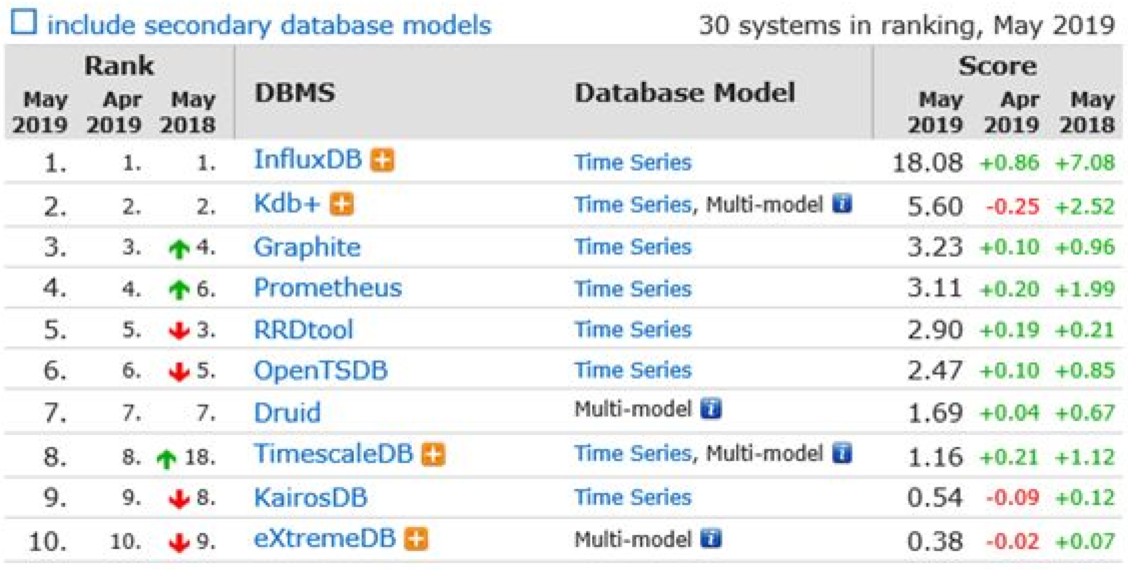
b) To manage the data in database and visualisation of the data.

c) To create a mechanism of data transmission between MKR1000 and database.

4.0 Literature Review

4.1 Choosing of the database

In this project, time series database is chosen as the database. Time series database is optimized for time-stamped data and build specifically for handling metrics, events or measurements that are time stamp.



☐ include secondary database models — 30 systems in ranking, May 2019

Rank			DBMS	Database Model	Score		
May 2019	Apr 2019	May 2018			May 2019	Apr 2019	May 2018
1.	1.	1.	InfluxDB	Time Series	18.08	+0.86	+7.08
2.	2.	2.	Kdb+	Time Series, Multi-model	5.60	-0.25	+2.52
3.	3.	↑4.	Graphite	Time Series	3.23	+0.10	+0.96
4.	4.	↑6.	Prometheus	Time Series	3.11	+0.20	+1.99
5.	5.	↓3.	RRDtool	Time Series	2.90	+0.19	+0.21
6.	6.	↓5.	OpenTSDB	Time Series	2.47	+0.10	+0.85
7.	7.	7.	Druid	Multi-model	1.69	+0.04	+0.67
8.	8.	↑18.	TimescaleDB	Time Series, Multi-model	1.16	+0.21	+1.12
9.	9.	↓8.	KairosDB	Time Series	0.54	-0.09	+0.12
10.	10.	↓9.	eXtremeDB	Multi-model	0.38	-0.02	+0.07

Figure 4.1: DB-Engines Time Series Database Ranking

DB-Engines is an independent website which rank database based on search engine popularity, social media mentions, number of jobs offers, and technical discussion frequency. According to DB-Engines (Figure 4.1), InfluxDB is the top time series database among 30 databases at May 2019. InfluxDB is chosen as the project database.

4.2 Message Queuing Telemetry Transport (MQTT)

MQTT is an open pub/sub protocol first developed in 1999. MQTT protocol consists of two types of network entities which are message broker and a number of clients. Broker is a server that receives all messages from the clients and then routes the messages to other clients. At client side subscriber is components that are interested in consuming certain information register their interest while publisher is the component that produce certain information do so by publishing their information.

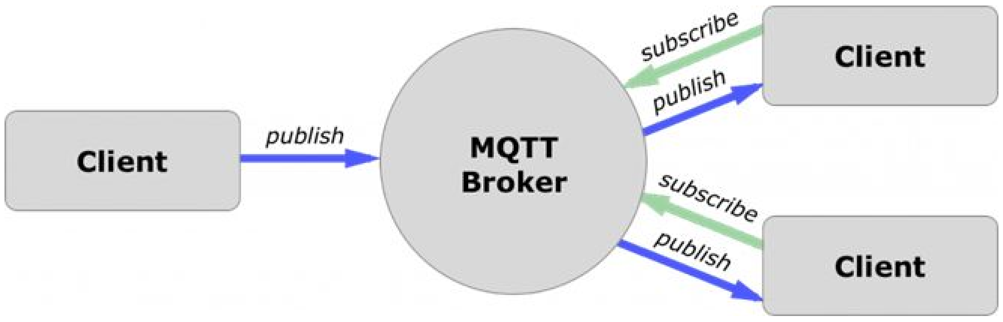


Figure 4.2: MQTT dataflow

Figure 4.2 show how MQTT work. Client use the publish function to send message to specific topic then broker send the message to client which subscribed to the topic.

4.3 Transport Layer Security (TLS)

TLS is cryptographic protocols designed to provide communications security over a computer network. Aim to provide privacy and data integrity. TLS connection is initiated using a sequence known as TLS Handshake. TLS handshake establish a cypher suit for each communication session. By using public key cryptography TLS able to set the matching session keys overs unencrypted channel. Figure 4.3 show how the TLS connection establish.

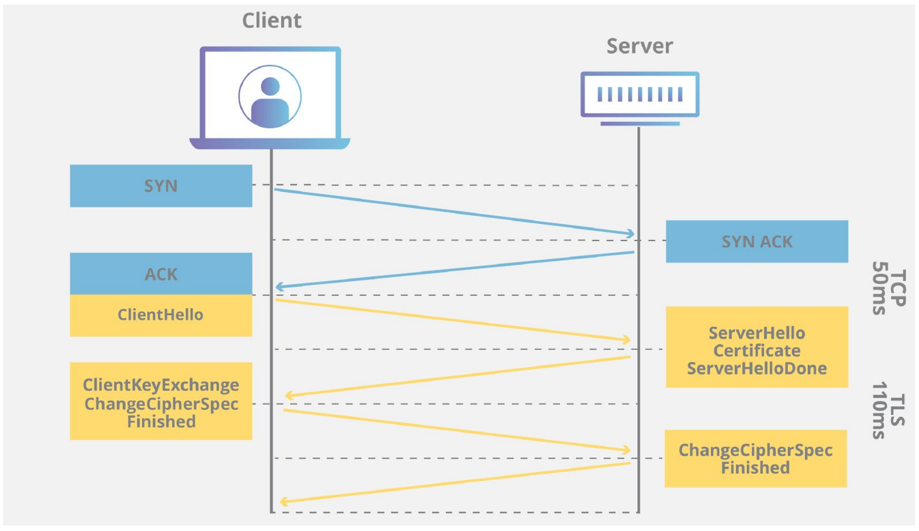


Figure 4.3: TLS Connection

5.0 Problem Statement

The problem statements of the project are:

a) Limited service time of battery powered WSN device.

b) Unable to monitor energy consumption of WSN device in an easy way.

c) Manual measuring of current is required without using sensor.

d) Lack of information to develop energy harvesting system for MKR1000 as WSN device.

The problem statements of the module are:

a) Handle multiple device connection and data transmission.

b) Data transferred is not secure.

c) Raw data in database is hard to analyse.

6.0 Research Methodology

6.1 Understand the system and server

In this project, the collected data are process and store in a server that its operating system is Ubuntu 18.04.2 LTS. It important to know how to operate a Linux operating system by only using command-line interface (CLI) without graphical user interface (GUI). Familiar with the command also help in operating the server.

6.2 Database

InfluxDB is chosen because it is designed to work with time-series data. Although SQL databases also can handle time-series data but not created strictly for that purpose. InfluxDB supported many programming languages such as Java, JavaScript, PHP and many more. The core of InfluxDB is a custom-build storage engine called the Time-Structured Merge (TSM) Tree, which is optimized for time series data. InfluxDB is perfect for custom monitoring and metrics collection, real-time analytics, plus IoT and sensor data workloads.

6.3 Message Queuing Telemetry Transport (MQTT)

MQTT is designed for devices which run on low power and low bandwidth. It is a lightweight pub/sub messaging protocol which any devices can subscribe to particular topic. With its lightweight behavior, it able to longer battery life. Pub/sub mechanism allow the messages to be send to many clients at one time. MQTT also provide 3 types of QoS for the data transfer which QoS 0 is at most once, QoS 1 is at least once and QoS 2 is exactly once. MQTT also support TLS connection for the data security and data integration. Authorization is also provided by creating username and password.

7.0 System Analysis and Design

7.1 System Architecture

The system is separated into front-end and back-end with the front-end being on the senor side while back-end being the server side. A simple design of the whole system is shown in figure below.

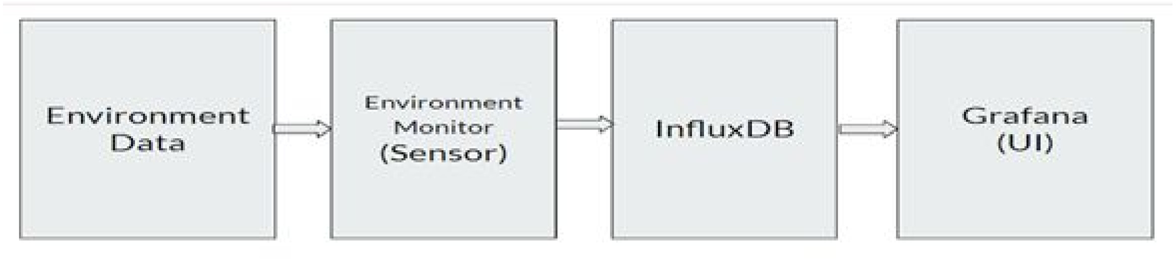


Figure 7.1: Block design of environment monitoring system

1. **Environment Monitor (Sensor Modules)**

Environment Monitor is the working module in detecting ambient environment parameters and sending it to the database. The sensors involved in this project are: DHT22 (Temperature and Humidity Sensor), GP2Y1014AU0F (Optical Dust Sensor) and CTSR 0.6P (current sensor) while the microcontroller is MKR1000 with low-power WiFi enabled. It is easy to connect to the Internet and communicate with the database.

2. **InfluxDB**

InfluxDB is the database of the system, it is a time-series based database which is advantage for the environment monitoring system due to its automation timestamp inclusion. With the inclusion of timestamp on each packet received by InfluxDB, we are able to keep track of the situation of the sensors whether it is functioning or not.

3. Grafana

Grafana is the data visualization platform which enables us to analyse the raw data received in InfluxDB. It has easy to configure graphical user interface (GUI) which ease the process of plotting graph and charts for the data received.

7.2 Process Flow

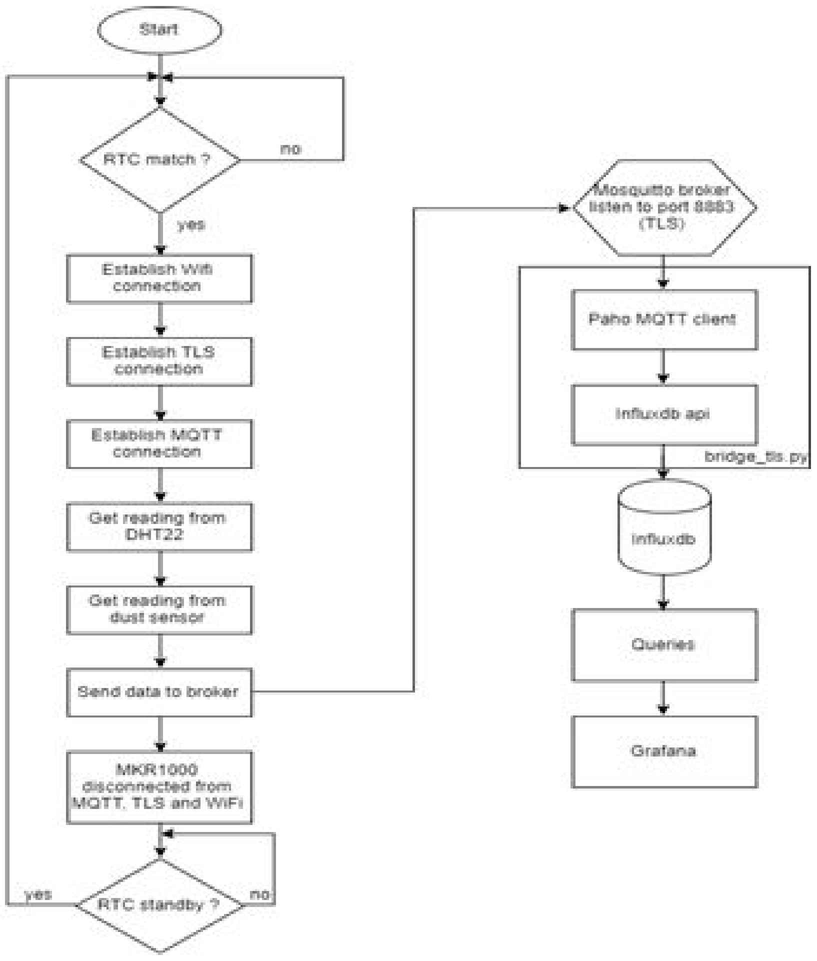


Figure 7.2: Process flow

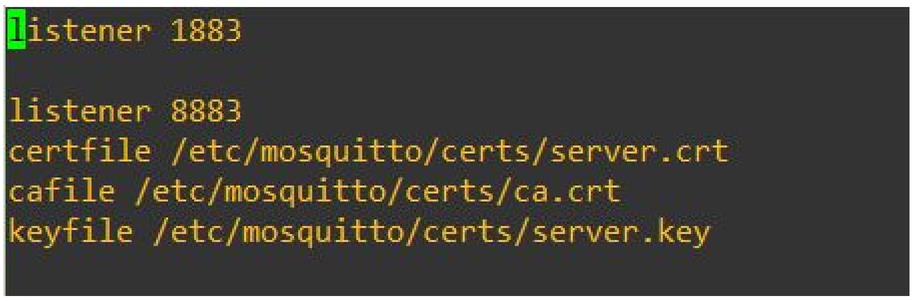
The process flow start from set up the RTC in MKR1000 and wait for the alarm time to reach and wake up then Wifi connection, TLS connection and MQTT connection are established. After all connection established then it will take the reading from all sensor and send data to broker. After that, MKR1000 will disconnect all the connection and wait to the alarm time to reach and go to standby mode. MKR1000 will constantly repeat wake up, establish connection, get reading from sensor, send data to broker and go to standby mode.

For dataflow from MKR1000 to server, MKR1000 need to establish the TLS with Mosquitto broker port 8883 before sending the data. Then program named bridge.py or bridge_tls.py inside the server will receive the data and send to database. The data then is queried by Grafana and displayed.

8.0 System Development

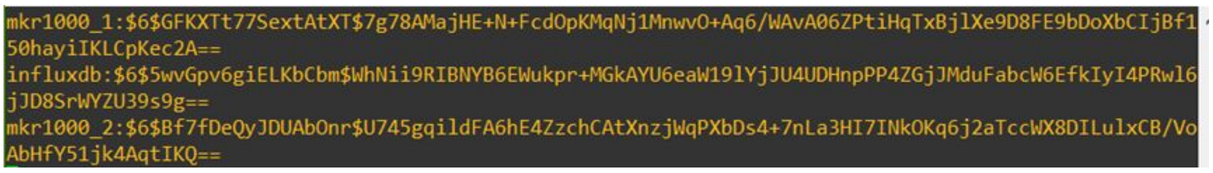
8.1 Connection Between Server and MKR1000

MQTT protocol is used for the connection between server and MKR1000 and the connection is secure by TLS. MQTT broker used in this project is Eclipse Mosquitto. Broker is installed and had been configured with port 1883 as non secure port and port 8883 as secure port. The certificate for TLS is created using openssl. Username and password also created for authorising client connection to broker. The MKR1000 connection is using QoS 1 which the data will received at least once mean duplicate data may occur.



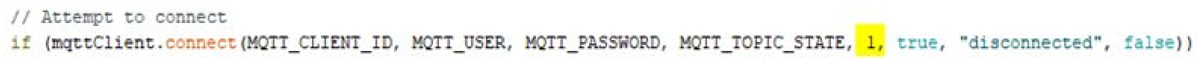
```
listener 1883

listener 8883
certfile /etc/mosquitto/certs/server.crt
cafile /etc/mosquitto/certs/ca.crt
keyfile /etc/mosquitto/certs/server.key
```

Figure 8.1: MQTT port configuration



```
mkr1000_1:$6$GFKXTt77SextAtXT$7g78AMajHE+N+FcdOpKMqNj1MnwvO+Aq6/WAvA06ZPtiHqTxBjlXe9D8FE9bDoXbCIjBf1
50hayiIKLCpKec2A==
influxdb:$6$5wvGpv6giELKbCbm$WhNii9RIBNYB6EWukpr+MGkAYU6eaW19lYjJU4UDHnpPP4ZGjJMduFabcW6EfkIyI4PRwl6
jJD8SrWYZU39s9g==
mkr1000_2:$6$Bf7fDeQyJDUAbOnr$U745gqildFA6hE4ZzchCAtXnzjWqPXbDs4+7nLa3HI7INkOKq6j2aTccWX8DILulxCB/Vo
AbHfY51jk4AqtIKQ==
```

Figure 8.2: MQTT username and password

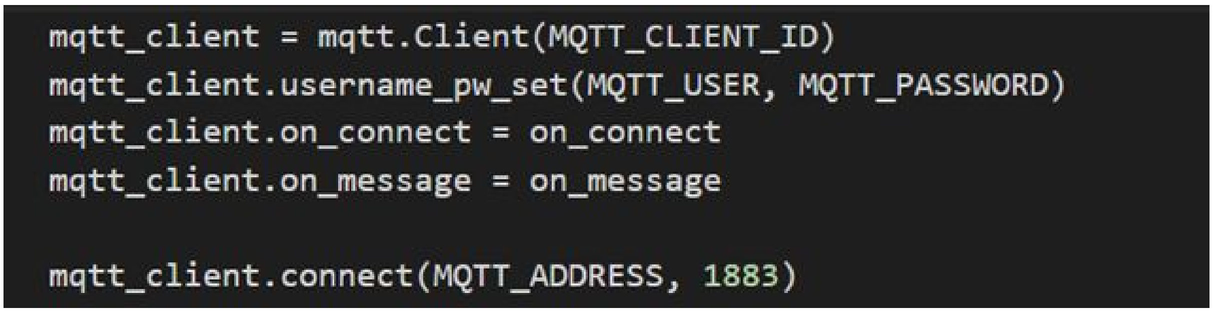


```
// Attempt to connect
if (mqttClient.connect(MQTT_CLIENT_ID, MQTT_USER, MQTT_PASSWORD, MQTT_TOPIC_STATE, 1, true, "disconnected", false))
```

Figure 8.3: MQTT QoS 1

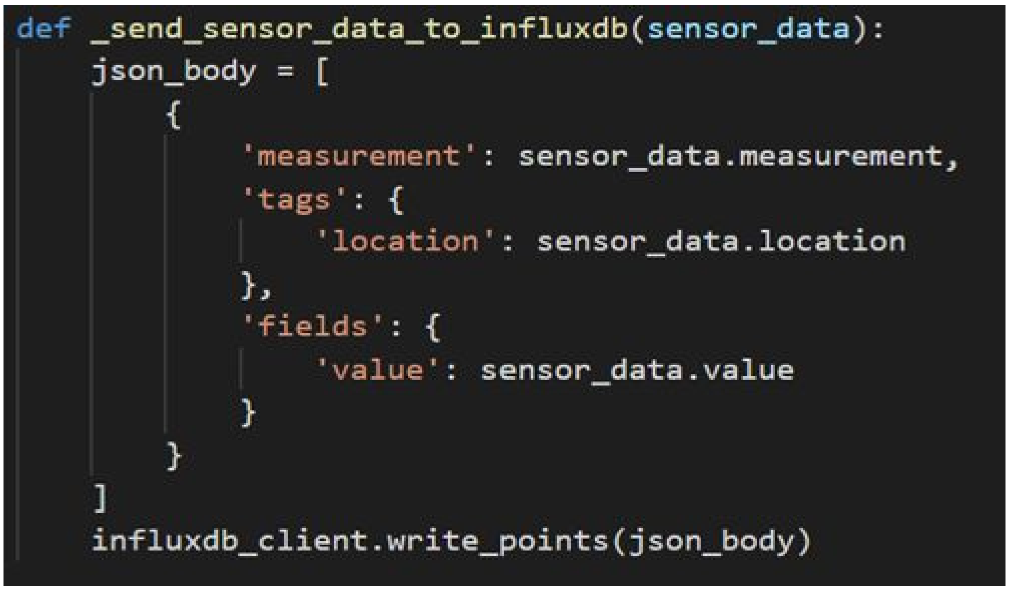
8.2 Data Forwarder

Programs named bridge.py and bridge_tls.py created to forward data from broker to database inside the server. The difference between these two programs are bridge_tls.py is connect to broker using secure port while bridge.py using non secure port. The program created using Python programming language, Influxdb api and Eclipse Paho Python Client (MQTT). Eclipse Paho Python Client used to connect to broker and subscribed to the topics to receive data and used Influxdb api to send data to database. Only one of the programs needed to run at one time because both function are same. “nohup python3 bridge.py&” used to run the program at background.



```
mqtt_client = mqtt.Client(MQTT_CLIENT_ID)
mqtt_client.username_pw_set(MQTT_USER, MQTT_PASSWORD)
mqtt_client.on_connect = on_connect
mqtt_client.on_message = on_message

mqtt_client.connect(MQTT_ADDRESS, 1883)
```

Figure 8.4: Eclipse Paho Python Client (MQTT)

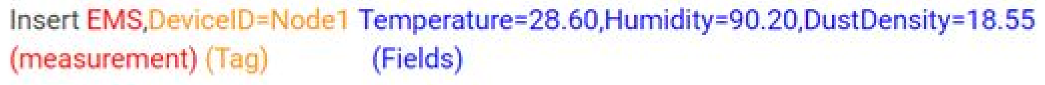


```
def _send_sensor_data_to_influxdb(sensor_data):
    json_body = [
        {
            'measurement': sensor_data.measurement,
            'tags': {
                'location': sensor_data.location
            },
            'fields': {
                'value': sensor_data.value
            }
        }
    ]
    influxdb_client.write_points(json_body)
```

Figure 8.5: Influxdb api send data

8.3 Database

InfluxDB is installed. InfluxDB used Influx Query Language(InfluxQL) which is SQL-like Query Language.

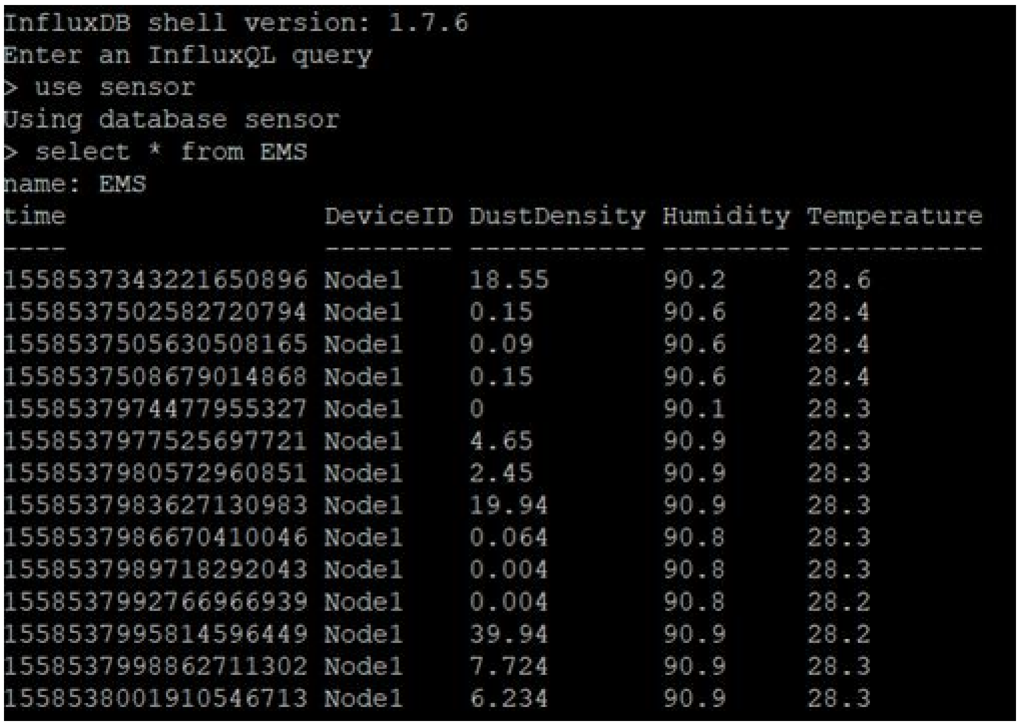


```
Insert EMS,DeviceID=Node1 Temperature=28.60,Humidity=90.20,DustDensity=18.55
(measurement) (Tag)         (Fields)
```

Figure 8.6: InfluxDB line protocol

- An InfluxDB measurement (EMS) is similar to an SQL database table.
- InfluxDB tags (DeviceID) are like indexed columns in an SQL database.
- InfluxDB fields (DustDensity, Humidity) are like unindexed columns in an SQL database.

Figure 8.6 show the example of the InfluxDB query.

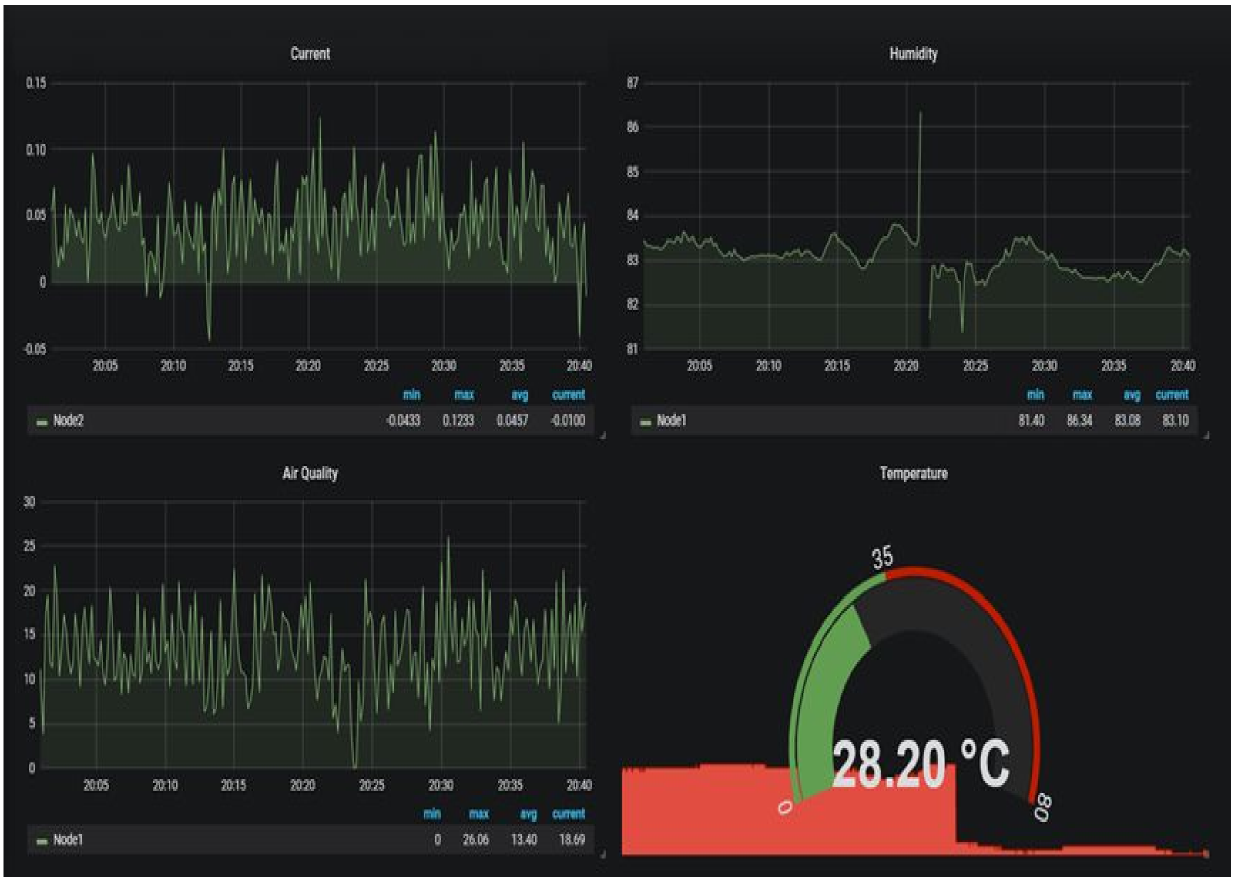


```
InfluxDB shell version: 1.7.6
Enter an InfluxQL query
> use sensor
Using database sensor
> select * from EMS
name: EMS
time                DeviceID DustDensity Humidity Temperature
----                -------- ----------- -------- -----------
1558537343221650896 Node1    18.55       90.2     28.6
1558537502582720794 Node1    0.15        90.6     28.4
1558537505630508165 Node1    0.09        90.6     28.4
1558537508679014868 Node1    0.15        90.6     28.4
1558537974477955327 Node1    0           90.1     28.3
1558537977525697721 Node1    4.65        90.9     28.3
1558537980572960851 Node1    2.45        90.9     28.3
1558537983627130983 Node1    19.94       90.9     28.3
1558537986670410046 Node1    0.064       90.8     28.3
1558537989718292043 Node1    0.004       90.8     28.3
1558537992766966939 Node1    0.004       90.8     28.2
1558537995814596449 Node1    39.94       90.9     28.2
1558537998862711302 Node1    7.724       90.9     28.3
1558538001910546713 Node1    6.234       90.9     28.3
```

Figure 8.7: Example of query

8.4 Display

The display of data is using Grafana. Grafana installed and integrated with Simple Mail Transfer Protocol (SMTP) for sending email for users registration and sending alerts. SMTP need to be installed and configure before intregrate with Grafana. Gmail used for the SMTP. Postfix installed to enable the SMTP function.

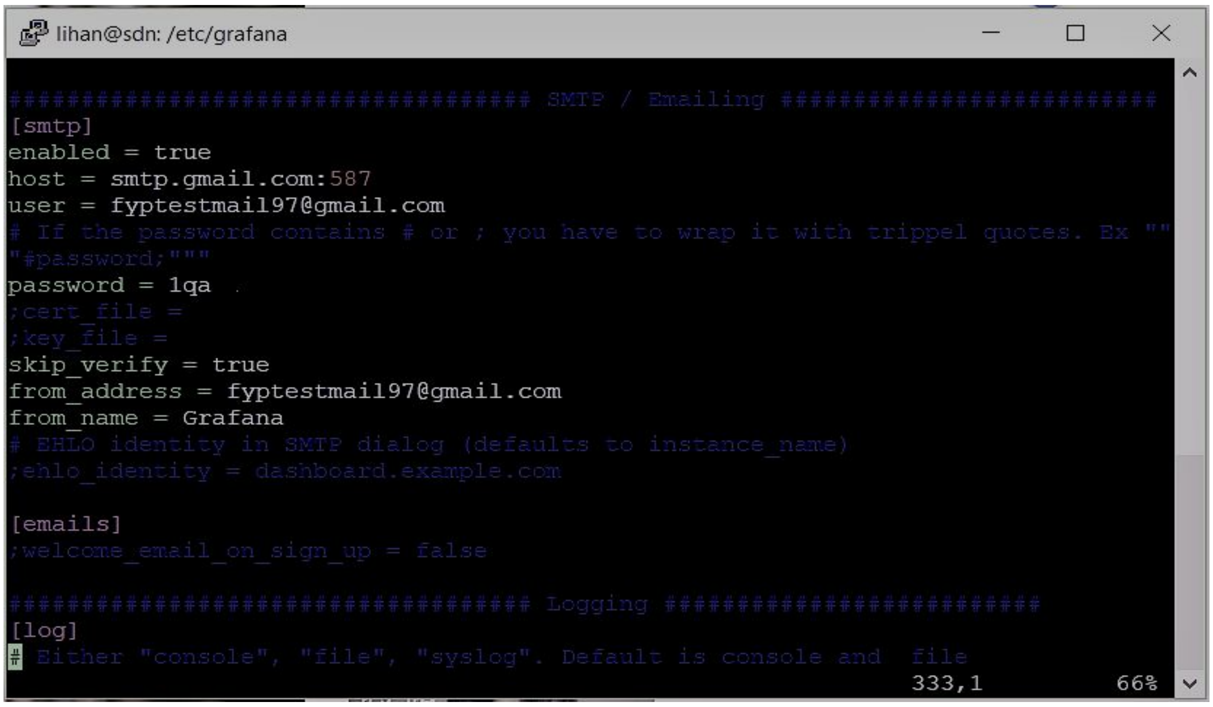


	min	max	avg	current
Node2	-0.0433	0.1233	0.0457	-0.0100

	min	max	avg	current
Node1	81.40	86.34	83.08	83.10

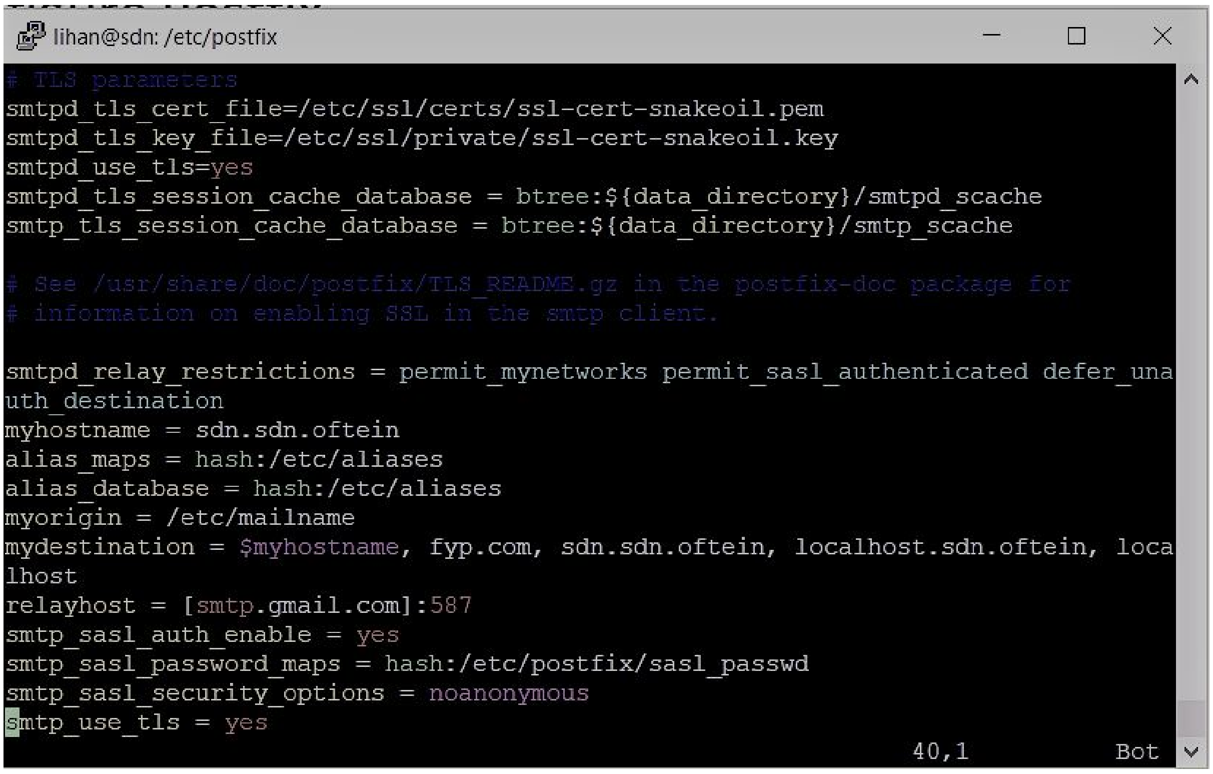
	min	max	avg	current
Node1	0	26.06	13.40	18.69

Figure 8.8: Display of Grafana



```
#################################### SMTP / Emailing ##########################
[smtp]
enabled = true
host = smtp.gmail.com:587
user = fyptestmail97@gmail.com
# If the password contains # or ; you have to wrap it with trippel quotes. Ex ""
"#password;"""
password = 1qa
;cert_file =
;key_file =
skip_verify = true
from_address = fyptestmail97@gmail.com
from_name = Grafana
# EHLO identity in SMTP dialog (defaults to instance_name)
;ehlo_identity = dashboard.example.com

[emails]
;welcome_email_on_sign_up = false

#################################### Logging ##########################
[log]
# Either "console", "file", "syslog". Default is console and  file
                                                            333,1         66%
```

Figure 8.9: Grafana SMTP configuration



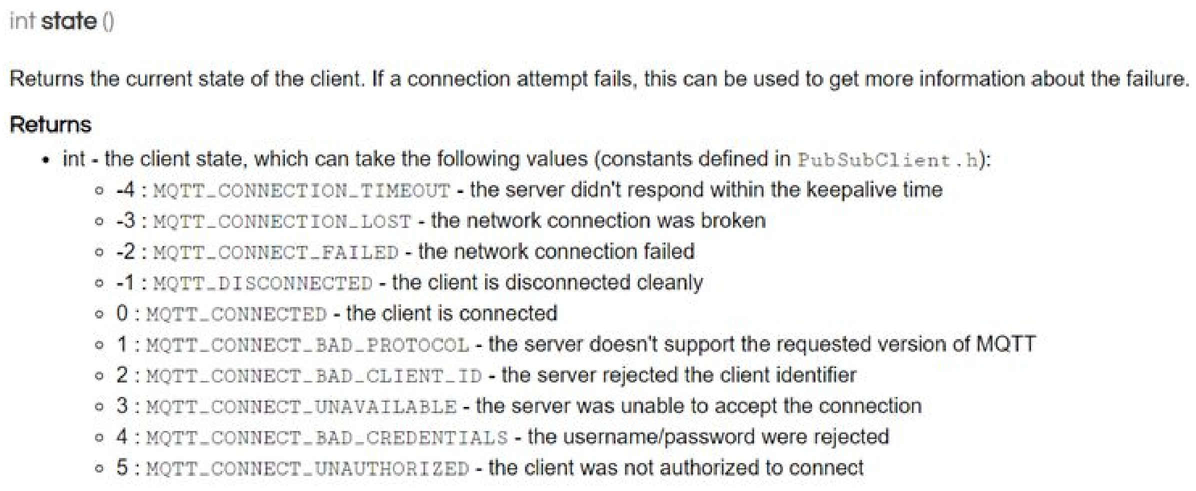
```
# TLS parameters
smtpd_tls_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
smtpd_tls_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
smtpd_use_tls=yes
smtpd_tls_session_cache_database = btree:${data_directory}/smtpd_scache
smtp_tls_session_cache_database = btree:${data_directory}/smtp_scache

# See /usr/share/doc/postfix/TLS_README.gz in the postfix-doc package for
# information on enabling SSL in the smtp client.

smtpd_relay_restrictions = permit_mynetworks permit_sasl_authenticated defer_una
uth_destination
myhostname = sdn.sdn.oftein
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
myorigin = /etc/mailname
mydestination = $myhostname, fyp.com, sdn.sdn.oftein, localhost.sdn.oftein, loca
lhost
relayhost = [smtp.gmail.com]:587
smtp_sasl_auth_enable = yes
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
smtp_sasl_security_options = noanonymous
smtp_use_tls = yes
                                                            40,1          Bot
```

Figure 8.10: Postfix configuration

9.0 System Evaluation

9.1 MQTT connection from MKR1000 to Server

Figure 9.1 show the error code for the PubSubClient.h library that used in MKR1000 for MQTT connection it can used to troubleshoot the connection. Figure 9.2 show the error for the incorrect password used. Figure 9.3 show the TLS connection error without correct certificate.



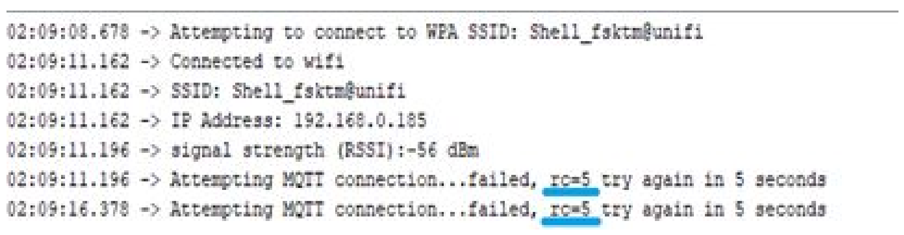
int **state** ()

Returns the current state of the client. If a connection attempt fails, this can be used to get more information about the failure.

Returns

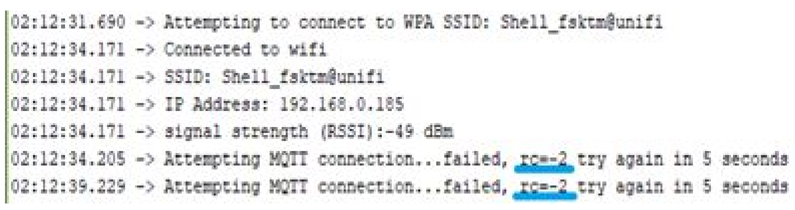
- int - the client state, which can take the following values (constants defined in `PubSubClient.h`):
  - -4 : `MQTT_CONNECTION_TIMEOUT` - the server didn't respond within the keepalive time
  - -3 : `MQTT_CONNECTION_LOST` - the network connection was broken
  - -2 : `MQTT_CONNECT_FAILED` - the network connection failed
  - -1 : `MQTT_DISCONNECTED` - the client is disconnected cleanly
  - 0 : `MQTT_CONNECTED` - the client is connected
  - 1 : `MQTT_CONNECT_BAD_PROTOCOL` - the server doesn't support the requested version of MQTT
  - 2 : `MQTT_CONNECT_BAD_CLIENT_ID` - the server rejected the client identifier
  - 3 : `MQTT_CONNECT_UNAVAILABLE` - the server was unable to accept the connection
  - 4 : `MQTT_CONNECT_BAD_CREDENTIALS` - the username/password were rejected
  - 5 : `MQTT_CONNECT_UNAUTHORIZED` - the client was not authorized to connect

Figure 9.1: Error code for the MQTT client



```
02:09:08.678 -> Attempting to connect to WPA SSID: Shell_fsktm@unifi
02:09:11.162 -> Connected to wifi
02:09:11.162 -> SSID: Shell_fsktm@unifi
02:09:11.162 -> IP Address: 192.168.0.185
02:09:11.196 -> signal strength (RSSI):-56 dBm
02:09:11.196 -> Attempting MQTT connection...failed, rc=5 try again in 5 seconds
02:09:16.378 -> Attempting MQTT connection...failed, rc=5 try again in 5 seconds
```

Figure 9.2: Error code for incorrect password

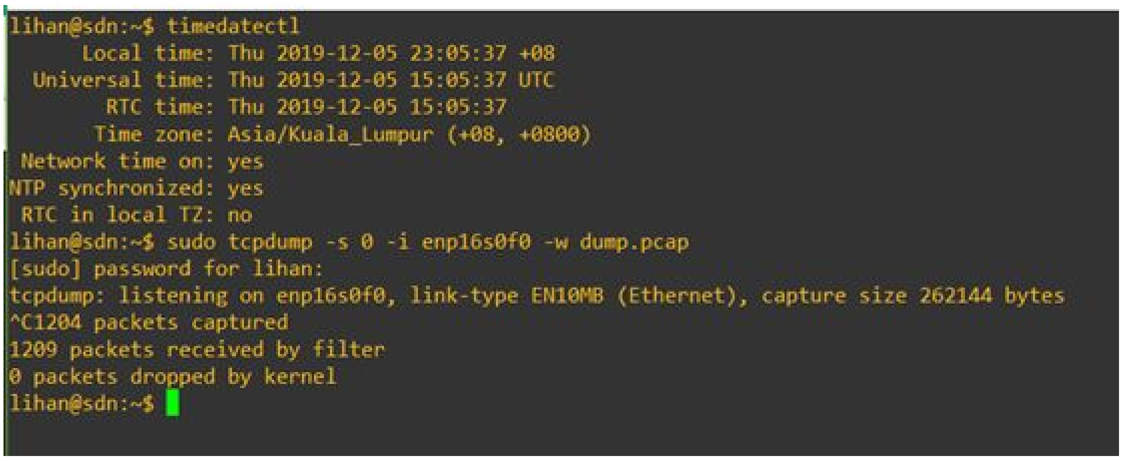


```
02:12:31.690 -> Attempting to connect to WPA SSID: Shell_fsktm@unifi
02:12:34.171 -> Connected to wifi
02:12:34.171 -> SSID: Shell_fsktm@unifi
02:12:34.171 -> IP Address: 192.168.0.185
02:12:34.171 -> signal strength (RSSI):-49 dBm
02:12:34.205 -> Attempting MQTT connection...failed, rc=-2 try again in 5 seconds
02:12:39.229 -> Attempting MQTT connection...failed, rc=-2 try again in 5 seconds
```

Figure 9.3: Error code for incorrect certificate

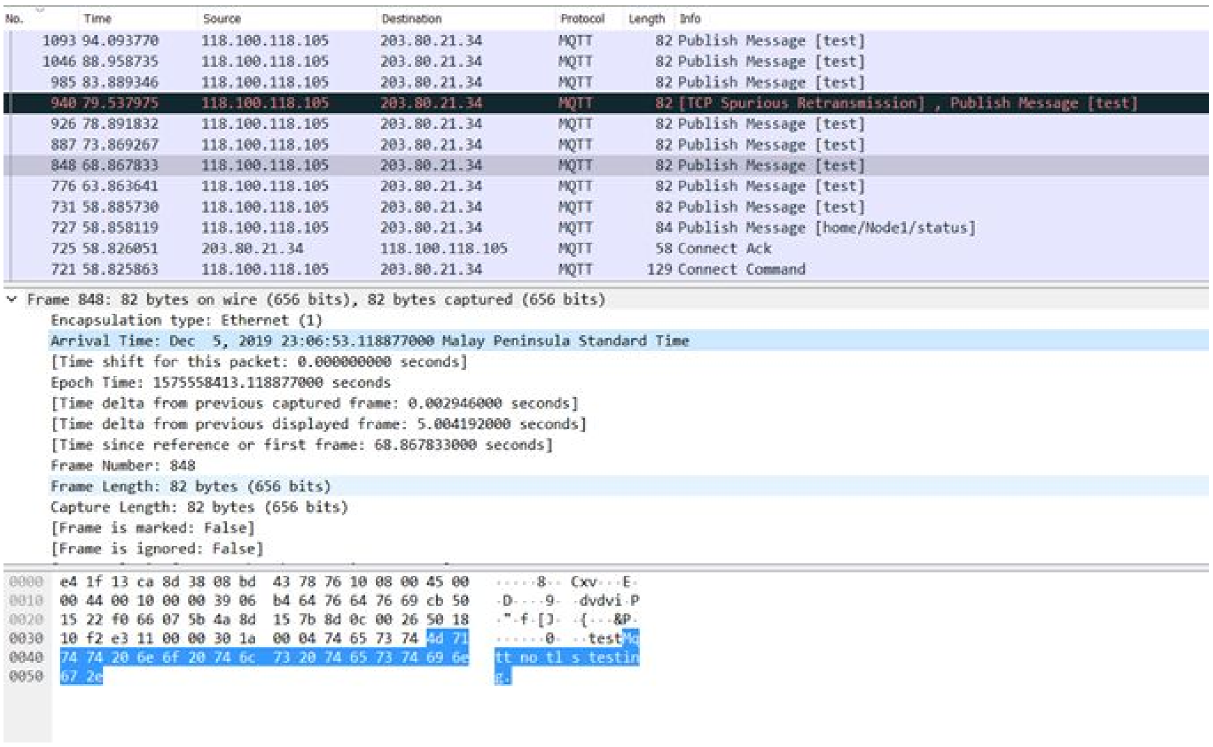
9.2 TLS connection testing

The TLS connection testing is done by using tcpdump to capture packet at the server interface then analyse the pcap file using wireshark. Figure 9.4 show that tcpdump captured the interface enp16s0f0 which is the interface that configured public ip and receiving the data send from MKR1000. Figure 9.5 and 9.6 show the comparison between the connection with and without TLS. The plain text data will be able to read from the packet that without TLS connection while the data in the packet with TLS connection only show encrypted data.



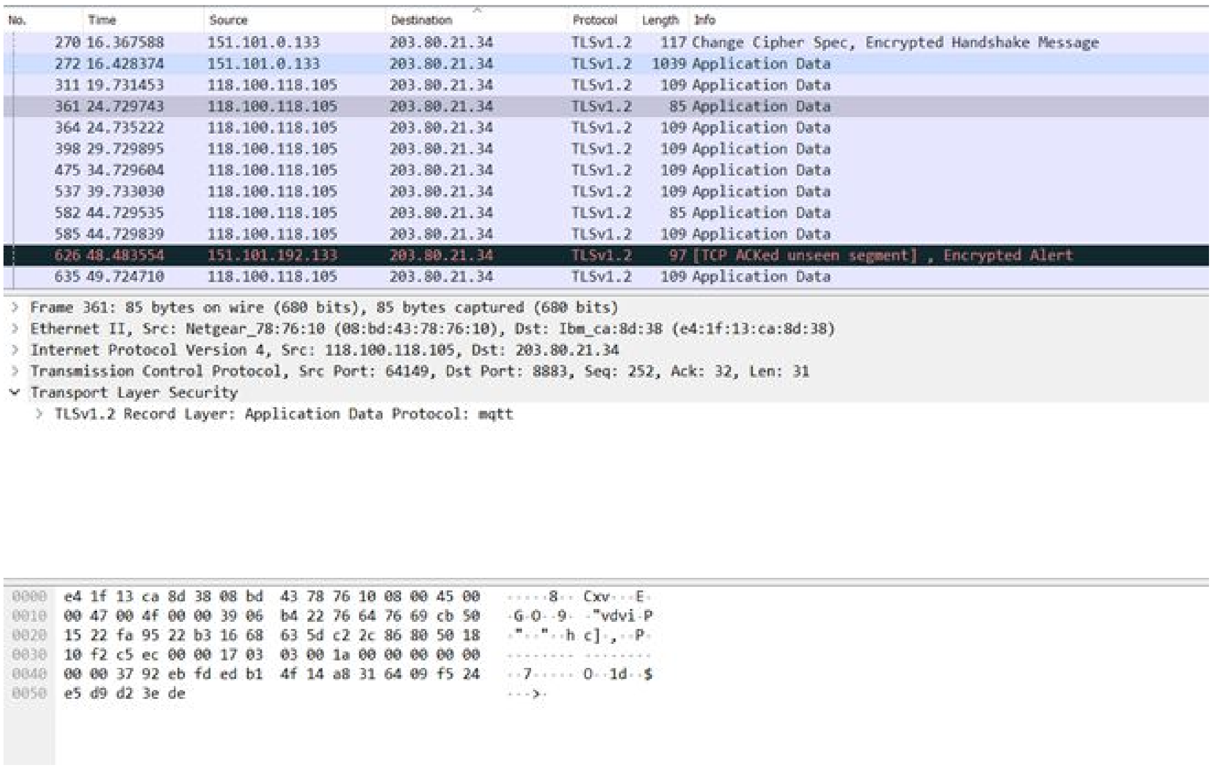
```
lihan@sdn:~$ timedatectl
      Local time: Thu 2019-12-05 23:05:37 +08
  Universal time: Thu 2019-12-05 15:05:37 UTC
        RTC time: Thu 2019-12-05 15:05:37
       Time zone: Asia/Kuala_Lumpur (+08, +0800)
 Network time on: yes
NTP synchronized: yes
 RTC in local TZ: no
lihan@sdn:~$ sudo tcpdump -s 0 -i enp16s0f0 -w dump.pcap
[sudo] password for lihan:
tcpdump: listening on enp16s0f0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C1204 packets captured
1209 packets received by filter
0 packets dropped by kernel
lihan@sdn:~$
```

Figure 9.4 Capturing packet



```
No.    Time        Source           Destination      Protocol  Length  Info
 1093 94.093770   118.100.118.105  203.80.21.34     MQTT        82 Publish Message [test]
 1046 88.958735   118.100.118.105  203.80.21.34     MQTT        82 Publish Message [test]
  985 83.889346   118.100.118.105  203.80.21.34     MQTT        82 Publish Message [test]
  940 79.537975   118.100.118.105  203.80.21.34     MQTT        82 [TCP Spurious Retransmission] , Publish Message [test]
  926 78.891832   118.100.118.105  203.80.21.34     MQTT        82 Publish Message [test]
  887 73.869267   118.100.118.105  203.80.21.34     MQTT        82 Publish Message [test]
  848 68.867833   118.100.118.105  203.80.21.34     MQTT        82 Publish Message [test]
  776 63.863641   118.100.118.105  203.80.21.34     MQTT        82 Publish Message [test]
  731 58.885730   118.100.118.105  203.80.21.34     MQTT        82 Publish Message [test]
  727 58.858119   118.100.118.105  203.80.21.34     MQTT        84 Publish Message [home/Node1/status]
  725 58.826051   203.80.21.34     118.100.118.105  MQTT        58 Connect Ack
  721 58.825863   118.100.118.105  203.80.21.34     MQTT       129 Connect Command

v Frame 848: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Dec  5, 2019 23:06:53.118877000 Malay Peninsula Standard Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1575558413.118877000 seconds
    [Time delta from previous captured frame: 0.002946000 seconds]
    [Time delta from previous displayed frame: 5.004192000 seconds]
    [Time since reference or first frame: 68.867833000 seconds]
    Frame Number: 848
    Frame Length: 82 bytes (656 bits)
    Capture Length: 82 bytes (656 bits)
    [Frame is marked: False]
    [Frame is ignored: False]

0000  e4 1f 13 ca 8d 38 08 bd  43 78 76 10 08 00 45 00   ·····8·· Cxv···E·
0010  00 44 00 10 00 00 39 06  b4 64 76 64 76 69 cb 50   ·D····9· ·dvdvi·P
0020  15 22 f0 66 07 5b 4a 8d  15 7b 8d 0c 00 26 50 18   ·"·f·[J· ·{···&P·
0030  10 f2 e3 11 00 00 30 1a  00 04 74 65 73 74 4d 71   ······0· ··testMq
0040  74 74 20 6e 6f 20 74 6c  73 20 74 65 73 74 69 6e   tt no tl s testin
0050  67 2e                                              g.
```

Figure 9.5: Without TLS connection



```
No.    Time        Source           Destination      Protocol  Length  Info
  270 16.367588   151.101.0.133    203.80.21.34     TLSv1.2    117 Change Cipher Spec, Encrypted Handshake Message
  272 16.428374   151.101.0.133    203.80.21.34     TLSv1.2   1039 Application Data
  311 19.731453   118.100.118.105  203.80.21.34     TLSv1.2    109 Application Data
  361 24.729743   118.100.118.105  203.80.21.34     TLSv1.2     85 Application Data
  364 24.735222   118.100.118.105  203.80.21.34     TLSv1.2    109 Application Data
  398 29.729895   118.100.118.105  203.80.21.34     TLSv1.2    109 Application Data
  475 34.729604   118.100.118.105  203.80.21.34     TLSv1.2    109 Application Data
  537 39.733030   118.100.118.105  203.80.21.34     TLSv1.2    109 Application Data
  582 44.729535   118.100.118.105  203.80.21.34     TLSv1.2     85 Application Data
  585 44.729839   118.100.118.105  203.80.21.34     TLSv1.2    109 Application Data
  626 48.483554   151.101.192.133  203.80.21.34     TLSv1.2     97 [TCP ACKed unseen segment] , Encrypted Alert
  635 49.724710   118.100.118.105  203.80.21.34     TLSv1.2    109 Application Data

> Frame 361: 85 bytes on wire (680 bits), 85 bytes captured (680 bits)
> Ethernet II, Src: Netgear_78:76:10 (08:bd:43:78:76:10), Dst: Ibm_ca:8d:38 (e4:1f:13:ca:8d:38)
> Internet Protocol Version 4, Src: 118.100.118.105, Dst: 203.80.21.34
> Transmission Control Protocol, Src Port: 64149, Dst Port: 8883, Seq: 252, Ack: 32, Len: 31
v Transport Layer Security
  > TLSv1.2 Record Layer: Application Data Protocol: mqtt

0000  e4 1f 13 ca 8d 38 08 bd  43 78 76 10 08 00 45 00   ·····8·· Cxv···E·
0010  00 47 00 4f 00 00 39 06  b4 22 76 64 76 69 cb 50   ·G·O··9· ·"vdvi·P
0020  15 22 fa 95 22 b3 16 68  63 5d c2 2c 86 80 50 18   ·"··"··h c]·,··P·
0030  10 f2 c5 ec 00 00 17 03  03 00 1a 00 00 00 00 00   ········ ········
0040  00 00 37 92 eb fd ed b1  4f 14 a8 31 64 09 f5 24   ··7····· O··1d··$
0050  e5 d9 d2 3e de                                     ···>·
```

Figure 9.6: With TLS Connection

10.0 Conclusion

By developing the back-end, we are able to store and process the raw data collected by the sensor node. The first objective is achieved by using the Transport Layer Security (TLS) between microcontroller and server the certificate is created by using openssl. Second objective is achieved by using Grafana to query data from database and display the data. Third objective is achieved by creating a bridge using Python, Influxdb api and Paho Mqtt client to receive the data and send to the database.

11.0 Reference

Akhtar, F., & Rehmani, M. H. (2015). Energy replenishment using renewable and traditional energy resources for sustainable wireless sensor network s: A review. Renewable and Sustainable Energy Review,45, 769-784. doi:10.1016/j.rser.2015.02.021

Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., & Wright, T. (2006). Transport Layer Security (TLS) Extensions. doi: 10.17487/rfc4366

DB-Engines. Db-engines ranking, 2019.

Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008). MQTT-S A publish/subscribe protocol for Wireless Sensor Networks. 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE 08). doi: 10.1109/comswa.2008.4554519

InfluxDB 1.7 documentation. (n.d.). Retrieved from https://docs.influxdata.com/influxdb/v1.7/

IoT: Number of connected devices worldwide 2012-2025. (n.d.). Retrieved from https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

Time Series Database (TSDB) Explained. (n.d.). Retrieved from https://www.influxdata.com/time-series-database/