Chapter 9
Smart City Based on MQTT Using Wireless Sensors

Monika Bharatbhai Patel
Charotar University of Science and Technology, India

Chintan Bhatt
Charotar University of Science and Technology, India

Hamed Vahdat-Nejad
University of Birjand, Iran

Hardik B. Patel
Epsilon Electronics, India

ABSTRACT

The internet of things can involve a huge number of connected devices and sensors for the betterment of our lives and businesses. Sensors are the main part of IoT. The main target of this chapter is to develop an IoT-based information observing system for specific areas like home, cities, industries, hospitals, etc. In this system, the environmental data of different elements, for example, temperature, humidity, pressure, should screen and get a redesign with a particular time interval. The authors use Raspberry Pi 3 and MQTT to observe information over a remote area and get an update with it anyplace in the world. They transmit the environmental data to the cloud server sent by Raspberry Pi 3. There, the authors can monitor data in both modes (online and offline).

DOI: 10.4018/978-1-5225-3805-9.ch009

INTRODUCTION

The Internet of Things (IoT), which is based on Machine to Machine (M2M) communication, can possibly change our life. It is based on client-server architecture as well as publish/subscribe transport protocol. Message Queuing Telemetry Transport (MQTT) is a lightweight open source protocol, which is easily implemented in IoT-based applications. It makes it perfect to use an IoT application, implemented system, and M2M devices (Atzori, lera & Morabito, 2010).

The main parts of IoT are sensors and the MQTT protocol. By various sensors we can easily get environmental data and can process them. Using this data, we can make our life more comfortable.

Right now these elements are encountering particular necessities that identify striking topics, such as business and work, economic, energy and water, open security, environment, healthcare, education and open administrations, all of which are, in some frame or another, progressively encouraged and empowered by ICT. Simultaneously, the most recent turbulent worldwide monetary downturn is progressively setting weight on urban areas to cut spending plans. It brings about malicious impacts not just on the upkeep and update of current ICT framework and offices, but additionally on future advancement policies. Notwithstanding, the idea of a "keen city", additionally known as astute city, data city, advanced city, e-city and virtual city, has been recognized just like a commendable case of a reaction to address the present and future complex difficulties of expanding asset proficiency, lessening emissions, providing sustainable human services administrations for maturing populations, empowering youth and incorporating minorities (Shah & Bhatt, 2014).

Smart sensors are the major aspect of the IoT (Bhayani, Patel & Bhatt, 2016). Once you have real-time data of various environments, then actions can be taken easily into account based on that Wi-Fi based smart connected sensors. With the help of the smart sensors, Wi-Fi module, MQTT protocol and IoT-based various applications, we are making systems that could be used for monitoring sensors data like temperature, humidity, pressures, and altitude of selected areas where the system is placed.

The IoT devices can collaborate with each other without or with human intercession. The IoT involves the things that are implanted in frameworks, and it can possibly change our reality with the assistance of them. The IoT is an assuming part for brilliant urban areas, shrewd frameworks, keen wellbeing checkings, and savvy garments and so on. Each one of these advances of IoT will change the living way of human beings. (Bhatt, Dey & Ashour, 2017)

MQTT

IoT is a keen system, which associates all things to the web with the final goal of trading data with concurred conventions. In this way, anybody can get to anything, whenever and from anyplace. In the IoT system, things or items are remotely associated with shrewd sensors. (Singh, Rajan, Shivraj, & Balamuralidhar, 2015)

MQTT (Message Queue Telemetry Transport) is a lightweight messaging protocol. It is a broker-based publish/subscribe messaging protocol designed to be simple, open and straightforward to implement. In 1999, Dr.Andy Stanford-Clark from IBM and Arlen Nipper from Arcom invented the MQTT protocol. In MQTT protocol the session layer is used for IoT (Garcia, Espada & Midgar, 2014). The IoT devices are inter-connected with each other using the light-weight MQTT communication. The IoT applications typically use C, Python, Java as well as an MQTT scripting language (Andy Banks, 2012).

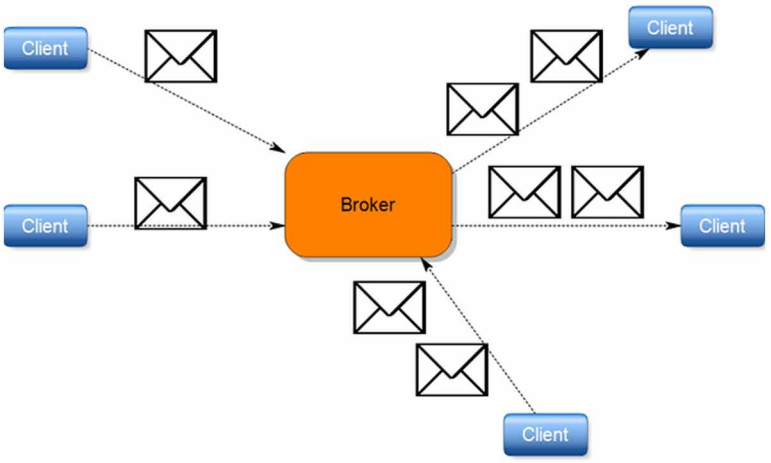
MQTT Broker controls the distribution of information (Gibbons, 2017). It stores, forwards, filters and prioritizes public requests from the publisher to the subscriber clients. Client switch between publisher and subscriber depends on the desired functionalities. There are many MQTT brokers available and they are different in their features; they can be set as per our requirements. MQTT brokers are Web sphere MQ, MQTT.js, Apache ActiveMQ, IBM Message Sight, Solaee Message Routers, Apache Apollo, Mosquitto, JoramMQ, HiveMQ, mosca, etc. For some of them, we can also implement additional features (Collina, Bartolucci, Vanelli-Coralli, & Corazza, 2014).

MQTT Client and MQTT Broker

MQTT client can act as a publisher and subscriber or sometimes both. MQTT library running on the MQTT client is any micro controller device that could be connected to an MQTT broker over any network. MQTT client libraries are available in a variety of programming languages like C, C++, C#, Android, Arduino, iOS, Go, .NET, JavaScript, Java, and etc. Using MQTT protocol client implementation is straight forward and really reduces to the embodiment. For this purpose, MQTT is suitable for small devices ("Sensor, Types of sensor,").

MQTT broker can handle hundreds of connected MQTT clients, concurrently. The broker is responsible for receiving MQTT client messages, it then filters them and decides who is interested in and forwards them, accordingly. MQTT clients are authenticated and authorized by MQTT broker. The MQTT broker is extensible; it easily incorporates some integration, authorization and authentication in the system backend ("Sensor, Types of sensor,").

Figure 1. MQTT Broker and Client



MQTT Connection

MQTT protocol is based on TCP/IP and client-broker scheme. The message is wrapped inside the TCP packet. It uses port number 1883 for MQTT. In MQTT no client can directly connect to another client, but the connection is always between one client and the broker. The MQTT client sends a CONNECT message to the broker for initiating the connection. The MQTT broker sends the CONNACK message as the response and also sends the status code. The broker keeps the connection alive till the client sends disconnect message to broker or the client loses the connection ("Sensor, Types of sensor,").

Each MQTT client has its unique identification number. The broker provides the client with its own unique id "Client ID" and by this id, manages the client connection and exchanges messages. The broker is answerable for managing messages persistence so that it can send them to the clients that were temporarily disconnected; this feature is called "retain message", which is important on unreliable networks with fragile connections.

When a client does not need to send any message or it does not receive messages for a long time, it must periodically send a "keep-alive" message to the broker to keep the connection alive; otherwise the broker terminates the connection after a timeout. The entire architecture is based on TCP/IP, so that each message exchanged between clients is wrapped inside TCP segments.

For connection between client and broker, there is MQTT connection command:

Figure 2. MQTT Broker and Client Connection

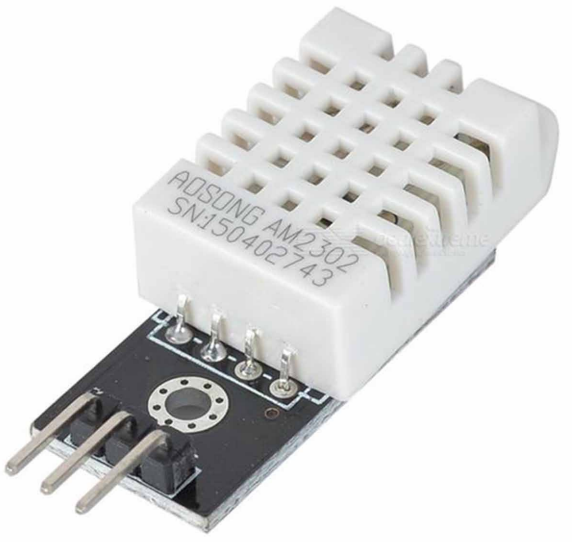
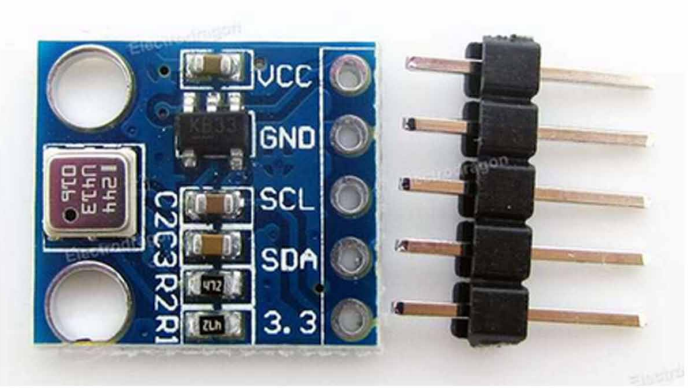


Figure 3. MQTT Publisher and Subscriber connection




```
MqttClient client =new MqttClient("tcp:// brokerid:portnum",
MqttClient.generateClientId(),
                 new  MemoryPersistence());
client.connect();
client.disconnect();
, where
           brokerid=IP address of broker,
           portnum=Broker port number(default 1883)
```

MQTT Publish/Subscribe Architecture

MQTT is based on "publish/subscribe" and "topic". A client can act as a publisher, subscriber or both. If it acts as a publisher, it publishes a message on a topic; if it acts as a subscriber, it subscribes to a specific topic. From this specific topic, it can receive the entire messages that are published on the topic (Locke, 2010).

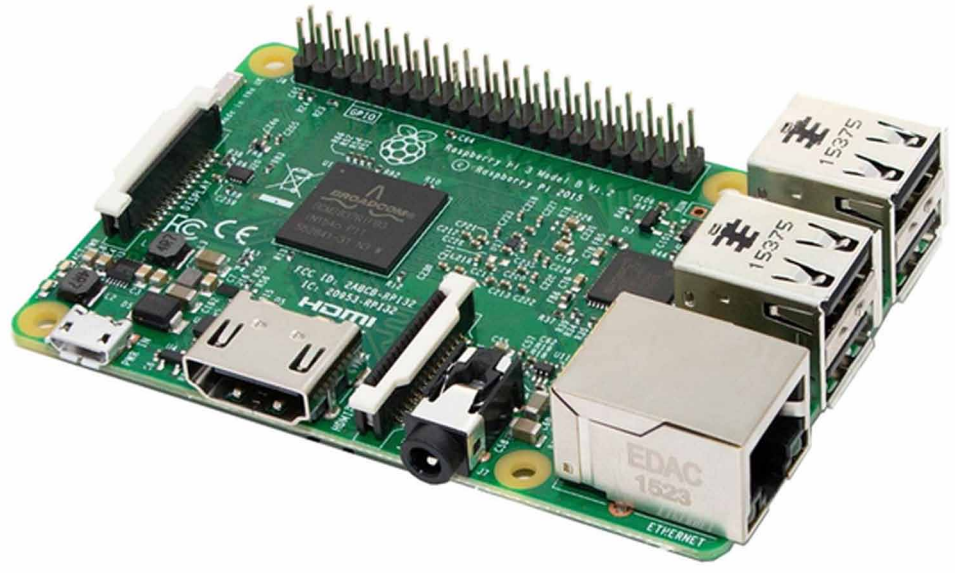
MQTT protocol has a message queue mechanism; and all subscriptions to the topics are managed by the MQTT broker. The broker has to transfer the message from the publishing client to those who are subscribed to the topic. MQTT allows one-to-one and one-to-many dispersion and the subscriber does not need to worry about the subscribers (Locke, 2010).

The following is the MQTT publish command for publishing a message:

```
client.connect();
client.publish(subscribetopic,message);
 where
           subscribetopic= Name of the subscribed topic
```

In the above example, Client A publishes a message on the specific topic "current_temp", and other clients Client B and Client C who have subscribed on that topic name "current_temp" and through MQTT broker, receive the message.

Figure 4. MQTT Publish architecture



Client B and Client C also publish messages on the same topic, so at this time they become a publisher and client A becomes receiver.

A subscription can be durable or non-durable. In the durable mode, the broker transfers a message to a subscribed client immediately, (if they are connected) or it stores the message till the subscriber connects. In the non-durable mode, if the subscriber is not connected, it loses that message.

The following is the MQTT subscribe command for the subscribe topic:

```
client.connect();
client.subscribe(subscribeTopic,QoS);
client.unsubscribe(subcribeTopic);
, where
          subcribeTopic=Topic name for subscription
          QoS=Quality of service
```

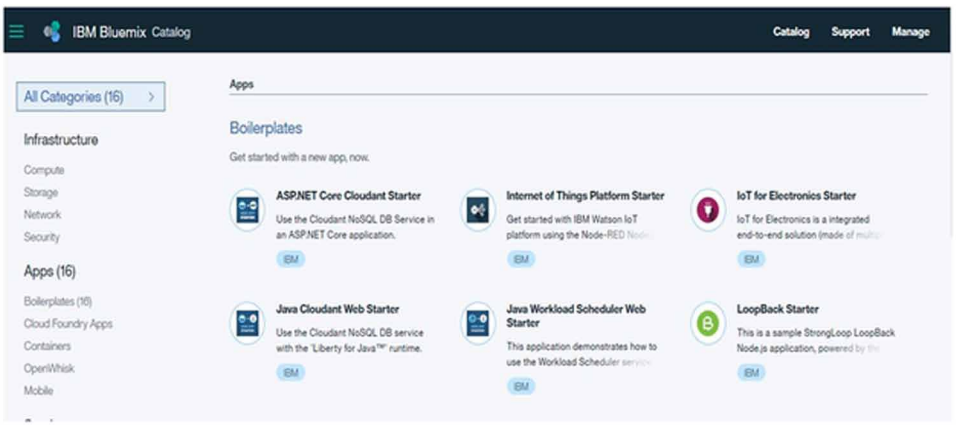
Quality of Services (QoS) of MQTT

QoS is an agreement between the sender and receiver of a message. It gives surety for delivering messages. The quality of services of a publication is an attribute of MqttMessage (Luzuriaga et al., 2015).

There are 3 levels of QoS in MQTT.

1. QoS 0
2. QoS 1
3. OoS 2

Figure 5. MQTT Subscription architecture



QoS 0: At Most Once

In this level, the message is delivered at most once, so it may not be delivered at all. Its delivery across the network is not acknowledged. At time of disconnection, the message might be lost. if the server fails, it receives the publication but it might be discarded, depending on the server. It is called "fire and forget" ("Sensors: Different Types of Sensors,").

There is the MQTT subscription command qos-0 to connect broker:

```
client.connect();
client.subscribe(subscribeTopic,qos-0);
, where
          subcribeTopic=Topic name for subscribe
          qos-0=Quality of service level
```

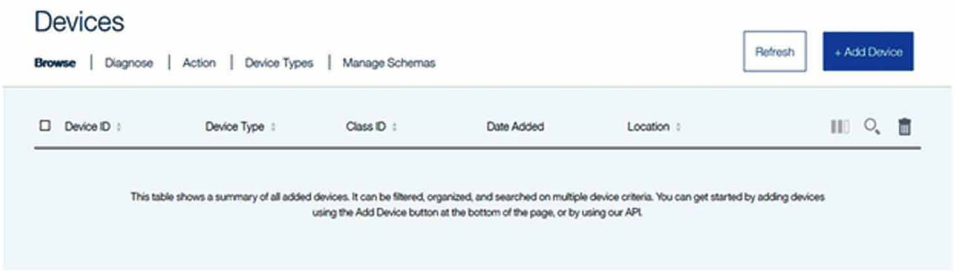
Use of QoS 0

- At the time of stable connection between sender and receiver or when there is wired connection between them, one can use this level of QoS.
- When data is not much important, loss of data is not important or message quality is not needed, this QoS level might be used.

QoS 1

It guarantees that the receiver gets messages successfully. In this case, the message can be delivered more than once. Sender stores the message until it gets PUBACK as acknowledgement from the receiver. PUBLISH and PUBACK are adapted by their

Figure 6. MQTT Quality of Service level 0



packet identifier that is placed in the packet. If PUBACK is not received in some fix amount of time, the sender resends the PUBLISH message. When receiver receives the PUBLISH message, it immediately replies PUBACK as acknowledgement to the sender.

At the time of redelivering message, the duplicate (DUP) flag is set by broker or client. Receiver sends PUBACK for DUP flag ("Sensors: Different Types of Sensors,").

There is the MQTT subscription command qos-0:

```
client.connect();
client.subscribe(subscribeTopic,qos-1);
, where
          subcribeTopic=Topic name for subscribing
          qos-1=Quality of service
```

Use of QoS 1

- OoS 1 is typically used when you have to get each message and your utilization case can deal with copies. The QoS-1 ensures the message touches base at least once.
- Obviously, the application must endure copies and process them appropriately. QoS 1 is a great deal quick in conveying messages.

QoS 2

It ensures that every message is received just once by the counterpart. It is the most secure and furthermore the slowest QoS level. Between the sender and receiver, it provides two flags.

It replies PUBREL and stores PUBREC. The receiver discards all packets that are stored and replies PUBCOMP when getting PUBREL ("Sensors: Different Types of Sensors,").

Figure 7. MQTT Quality of Service level 1

There is the MQTT subscription command qos:

```
client.connect();
client.subscribe(subscribeTopic,qos-2);
, Where
          subcribeTopic=Topic name for subscribing
          qos-2=Quality of service
```

Use of QoS 2

- When you need every message once use this QoS level, but duplication may harm your application.
- In this QoS level, one must take care of the overhead, because it takes longer to be completed.

Types of MQTT Control Packet

MQTT control Packets are shown in table 1.

MQTT Features

1. TCP/IP port 1883 is used for MQTT.
2. The use of TCP/IP to provide basic network connectivity.

Figure 8. MQTT Quality of Service level 2

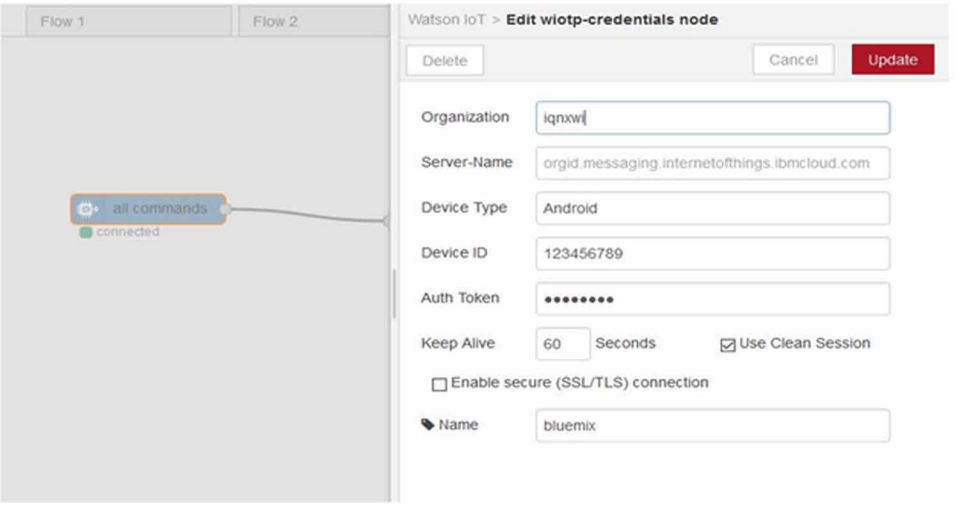


Table 1. MQTT control Packet

Packet Name	Packet Value	Flag Direction	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client → Server	Client request to connect to Server
CONNACK	2	Server → Client	Connect acknowledgment
PUBLISH	3	Client → Server Or Server → Client	Publish message
PUBACK	4	Client → Server Or Server → Client	Publish acknowledgment
PUBREC	5	Client → Server Or Server → Client	Publish received
PUBTREL	6	Client → Server Or Server → Client	Publish release
PUBCOMP	7	Client → Server Or Server → Client	Publish complete
SUBSCRIBE	8	Client → Server	Client subscribe request
SUBACK	9	Server → Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client → Server	Unsubscribe request
UNSUBACK	11	Server → Client	Unsubscribe acknowledgment
PINGREQ	12	Client → Server	PING request
PINRESP	13	Server → Client	PING response
DISCONNECT	14	Client → Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

3. A small transport overhead and protocol exchange minimized to reduce network traffic.
4. A mechanism to notify interested parties of an abnormal disconnection of a client using the Last Will and Testament feature.

PROBLEM SOLUTION

The MQTT is lightweight. The degree of exchanging messages in an application relies upon the way an MQTT broker and client are composed. When building a system with the help of sensors, Raspberry Pi 3, MQTT, and IoT based different applications and executing it, issues happen.

Without MQTT and IoT establishment and connection, it will be difficult to interfacing with client and server as well as machines that are utilized for M2M correspondence. MQTT is particularly used for installed devices as well as M2M and IoT availability protocol.

MOSQUITTO

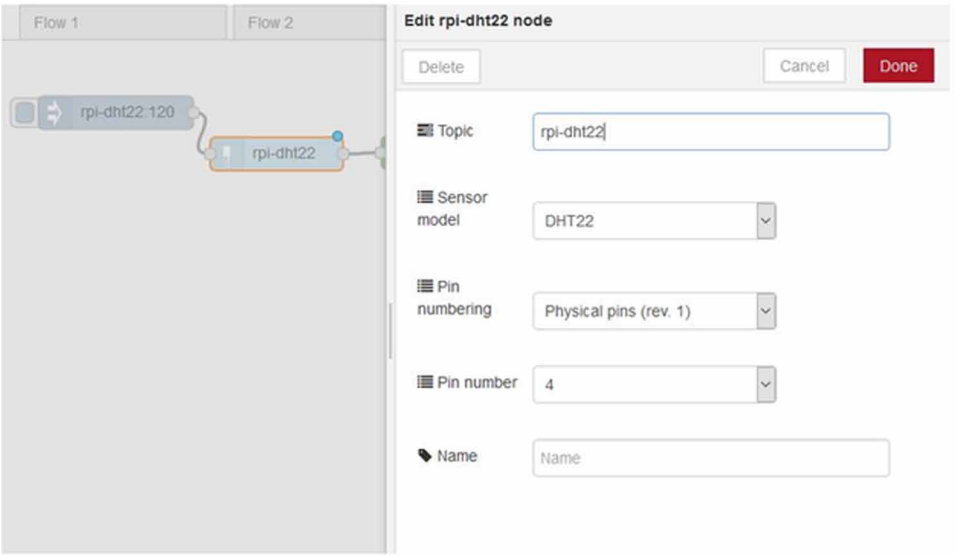
MQTT is a broker-based public and subscription TCP/IP messaging protocol. It is designed for connections with remote locations, where a small code footprint is required or the network bandwidth is limited. Mosquitto is an open-source message brokering service that uses the MQTT protocol for sending and receiving messages. This broker is based on open source software that implements MQTT v 3.1 and v3.1.1.

Mosquitto Installation

For Broker

- The Raspberry Pi normal "apt-get" archives do not contain the latest version of the Mosquitto software. Do the followings:

Figure 9. Device with MQTT broker



```
sudo wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.
key
sudo apt-key add mosquitto-repo.gpg.key
cd /etc/apt/sources.list.d/
sudo wget http://repo.mosquitto.org/debian/mosquitto-jessie.
list
sudo apt-get update
sudo apt-get install mosquito
```

- Install the three parts of the Mosquitto, properly:

```
mosquitto – the MQTT broker
mosquitto-clients – command line clients, very useful in
debugging
python-mosquitto – the Python language bindings
sudo apt-get install mosquitto mosquitto-clients python-
mosquitto
```

- As is the case with most packages from Debian, the broker is immediately started. Since we have to configure it first, stop it:

```
sudo /etc/init.d/mosquitto stop
```

- Before using Mosquitto, we need to set up the configuration file. The configuration file is located at /etc/mosquitto.

```
sudo nano /etc/mosquitto/mosquitto.conf
```

- Next add the next six lines:

```
 log_type error
log_type warning
log_type notice
log_type information
connection_messages true
log_timestamp true
```

- Now start the mosquitto server:

```
sudo /etc/init.d/mosquitto start
```

- Broker subscribe a specific topic

```
mosquitto_sub -d -t hello/world
```

- Broker publishes a message on a specific topic

```
mosquitto_pub -d -t hello/world -m "Hello"
```

For Clients

- Install mosquitto client:

```
sudo apt-get install mosquitto-client
```

- Client subscribes a specific topic

```
            mosquitto_sub -h <broker ip> -p 1883 -v -t 'topic'
-u <user id> -P <password
```

- Client publishes a message on a specific topic

```
            mosquitto_pub -h <broker ip> -p 1883 -t 'topic' -u
<user id> -P <password> -m "Message"
```

NODE-RED

Node-RED is a product tools created by IBM for implementing aggregated equipment gadgets, APIs and online administrations as a component of the Internet of Things. Node RED gives a program-based stream manager, which can be utilized to make JavaScript capacities. Components of applications can be spared or shared for re-utilization. The runtime is based on Node.js. The streams made in Node-RED are put away utilizing JSON.

Node-RED is a tool for implementing the Internet of Things in new and intriguing ways, supporting equipment gadgets, APIs, and online administrations. It is based on top of Node.js and exploits the colossal hub module environment to provide an instrument that is equipped for coordinating a wide range of frameworks. Furthermore, its lightweight nature makes it perfect to keep running at the edge of the system, for example, on the Raspberry Pi, and other hack-accommodating stages.

Node-RED contains the accompanying Watson IoT hub that helps users to unite devices, gateways and applications to Watson IoT Platform and make IoT arrangements, rapidly.

Node-Red Installation on Raspberry Pi

- First install its library:

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/
raspbian-deb-package/master/resources/update-nodejs-and-
nodered)
```

- Update nodejs

```
update-nodejs-and-nodered
```

- Go to .node-red folder and rebuild npm.

```
cd ~/.node-red
npm rebuild
```

- Goto .node-red folder.

```
cd ~/.node-red
npm ls --depth=0
```

- Start node-red.

```
node-red-start
```

SENSORS

Sensors are devices that are used for detecting and reacting to electrical or optical signals. They are able to convert any type of physical quantity to be measured into a signal that can be read, displayed, stored or used to control some other quantity. It converts physical parameters like temperature, humidity, pressure, blood, speed, etc. into a signal that can be measured, electronically.

There are several features for a sensor:

- Usually used for environmental measurements.
- Calibration changes over time.
- Sensors detect smallest changes.
- Reading varies in repeated measurements.

DHT22

DHT is the temperature and humidity sensor. It has digital sensor and has a low cost. It measures the surrounding air and capacitive humidity by using a thermostat and spits out a digital signal. It is simple to use; however requires wary wanting to get data.

It is the costlier version, which obviously has better specifications.DHT22 measuring temperature is between -40 to +125℃ with +-0.5degree accuracy while DHT11 measuring temperature is between 0 to 50 ℃ with +-2degree accuracy. DHT11 has less measuring range of humidity than DHT22. DHT22 measuring range is from 0 to 100% with 2-5% accuracy while DHT11 have 20 to 80% with 5% accuracy (Govindan & Azad, 2015).There are two specifications where the DHT11 is better than the DHT22. Both sensors have 3 to 5 volts and their max used yielded when measuring is 2.5mA. The DHT22 sampling rate is 0.5Hz one reading every two seconds and the DHT11 is 1Hz or one reading every second. DHT11 has a small size (Govindan & Azad, 2015).

4.2 BMP085

Barometric pressure and temperature are measured by BMP085. It is a low-cost sensing tool. This sensor is with PCB that has a 4.3V regulator, I^2C level shifter, and pull-up resistors. It is the successor of the SMD500.

Figure 10. DHT22 sensors

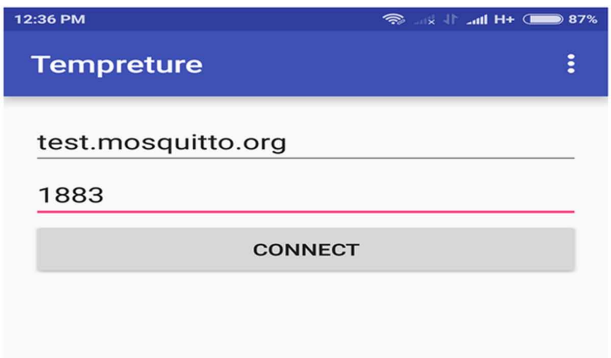
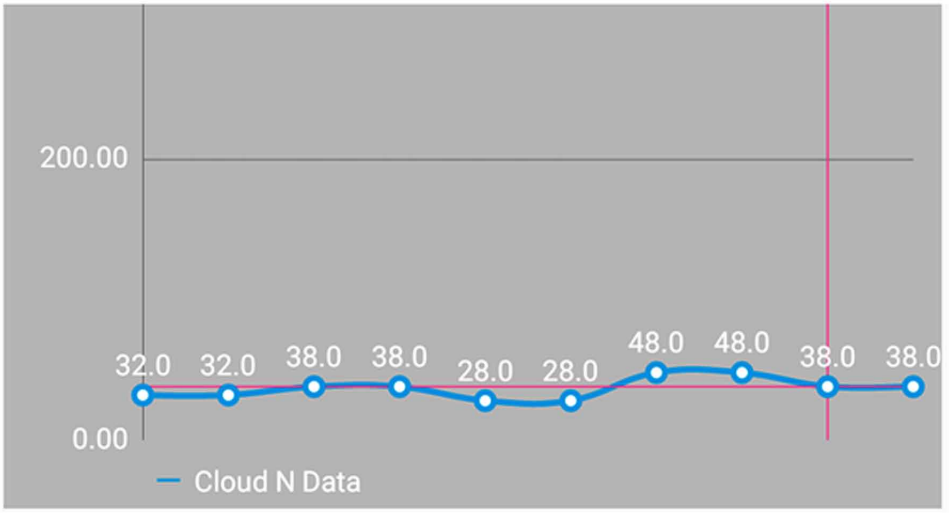


Figure 11. BMP085 sensors



The BMP085 sensor depends on piece-resistive MEMS innovation for EMC vigor, high precision and linearity and additionally long term dependability. It arrives in an ultra-thin, yet powerful 8-pin fired lead-less chip carrier (LCC) package. The BMP085 is intended to be associated straightforwardly to a miniaturized scale controller of a cell phone by means of the I2C bus.

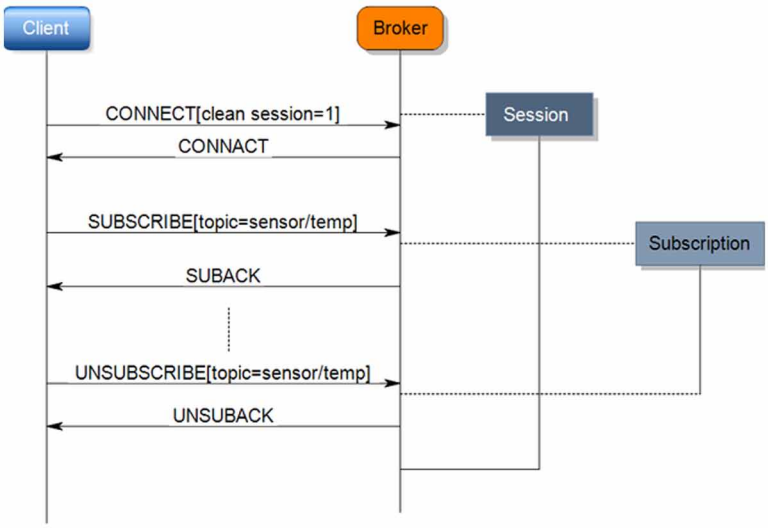
The BMP085 comes as completely aligned, prepared to utilize sensor module without the requirement for extra outer hardware. Weight and temperature information are given as 16 bit values by means of the I2C interface (Chen & Lin, 2014).

RASPBERRY PI 3

It was set up by the Raspberry Pi establishment in UK("RASPBERRY PI 3 MODEL B,"). It has been prepared for open utilization since 2012 with making a minimal effort instructive microcomputer for understudies and kids. The fundamental pure-posture of planning the Raspberry Pi board is to support learning, experimentation and advancement for school level understudies. The Raspberry Pi board is versatile and easy to use. A colossal fragment of this being driven by the versatile businesses. 98% of the cell phones were utilizing ARM innovation (Hunkeler, Truong, & Stanford-Clark, 2008).

The Raspberry Pi comes in two models; which are model A and B. The primary distinction between model A and B is the USB port. The model A board devours

Figure 12. Raspberry Pi 3 Mode B



less power and that does exclude an Ethernet port. The model B board incorporates an Ethernet port and is outlined in china. The Raspberry Pi accompanies an arrangement of open source advancements, i.e. correspondence and sight and sound web technologies. In the year 2014, the establishment of the Raspberry Pi (shown in table 2) board propelled the PC module, that bundles a model B Raspberry Pi board into the module for use as a piece of inserted frameworks, to energize their utilization ("Meteoric growth: The Internet of Things is scaling exponentially,").

APPLICATION

Step-1: Create IBM Bluemix service

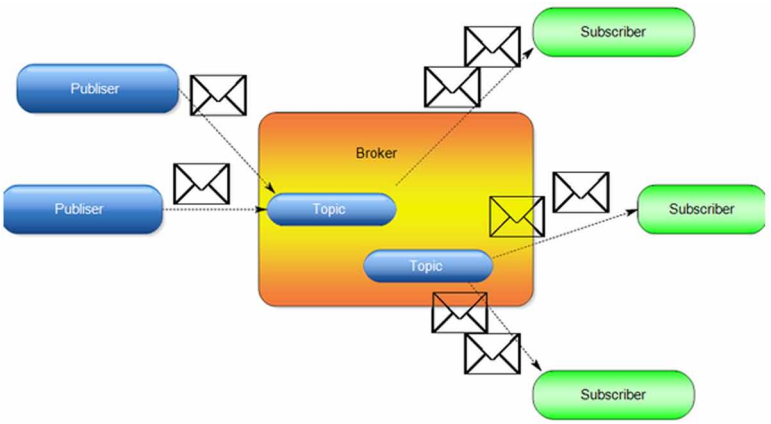
- Login to Bluemix using Bluemix account
- Click on Catalog in top
- Select Boilerplates section and click on Internet of things platform starter (Figure 13)

- Give the unique name of your app. I used "bluemixMobMqtt".
- Click on create and wait for the application to start.
  - For adding IoT services.
- Click on ADD A SERVICES
- Choose the Internet of things service in the Internet of things section.
- Click Create.

Table 2. Raspberry Pi 3 detail

Type	Model-B
Generation	3
Date of release	2016, February
Architecture	ARMv8-A (64/32-bit)
SoC	Boardcom BCM2837
CPU	1.2 GHz 64-bit quad-core ARM Cortex-A53
GPU	Boardcom VideoCore IV@250MHz,OpenGL ES 2.0, MPEG-2 and VC-1,1080p30 H.264/MPEG-4 AVC high-profile decoder and encoder
Memory	1 GB
USB 2.0 ports	4(on-board 5-port USB hub)
Video for input	Camera interface connector with 15-pin
Video for output	HDMI, composite video have 3.5mm TRRS jack, MIPI display interface-DSI for raw LCD panels
Audio input	I^2S for 2 board
Audio output	Analog via 3.5mm phone jack, HDMI for digital
Storage(on-board)	USB Boot Mode and Micro SDHC slot
Network(on-board)	Ethernet 10/100 Mbit/s, wireless 802.11n, Bluetooth 4.1
Low-level peripherals	17xGPIO plus the same specific function and HAT ID bus
Power rating	800 mA
Power source	5V via Micro USB
Size	85.60mm x 56.5mm
Weight	45g
Console	Micro-USB cable or optional GPIO power connector for serial cable

Figure 13. IBM Bluemix catalog and select Boilerplate



- Click Restage.
  - Launch the IoT services
- Navigate the application on overview and click Internet of things
- Click the Launch dashboard to open IoT console.
  - Register the devices
- Click Add Device on the Devices tab (Figure 14).
- Choose Create a Device type for the Device type option.
- Fill in the Device Type as “Android”.
- Fill in the Device ID with a unique id like “123456789”
- Click Continue
- Copy the information (Figure 15).

Step-2: Start Node-red

- Add Watson IoT node
- Configure with their information and insert the information (Figure 16).

Figure 14. IBM Watson Add Device

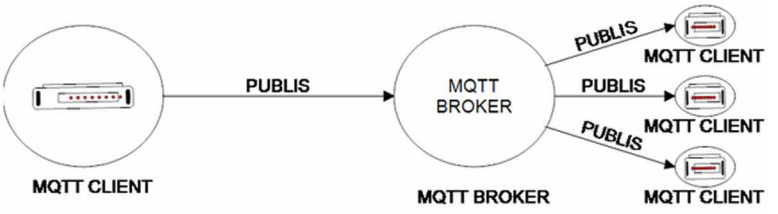


Figure 15. Copy information of the Device

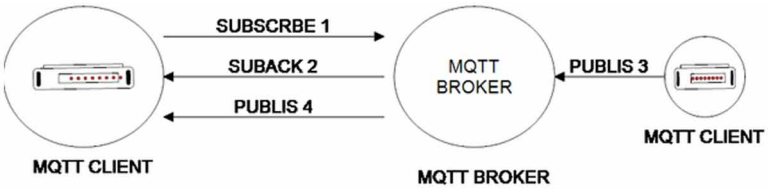
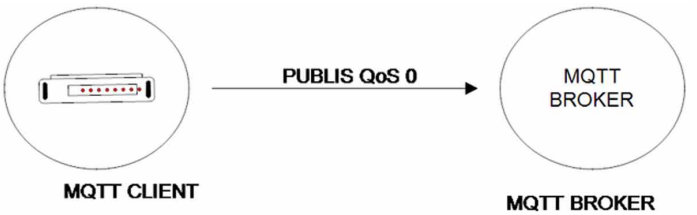


Figure 16. Configure Watson IoT node



- Add DHT22 node
- Configure this node (Figure 17)

Step-3: Make Android app

- This is configured with MQTT protocol.
- Enter Broker id and port address (Figure 18).

- *Make a graph of temperature (Figure 19).*

Figure 17. Configure DHT22 node

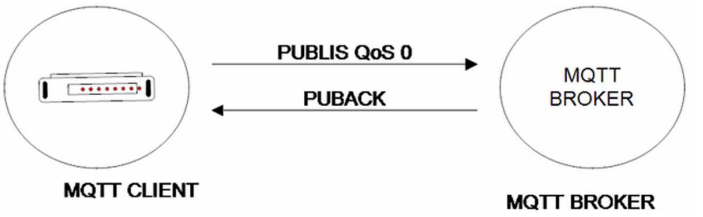


Figure 18. Connect with mosquito broker

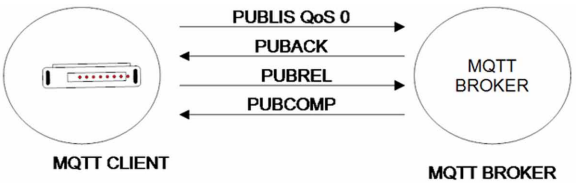
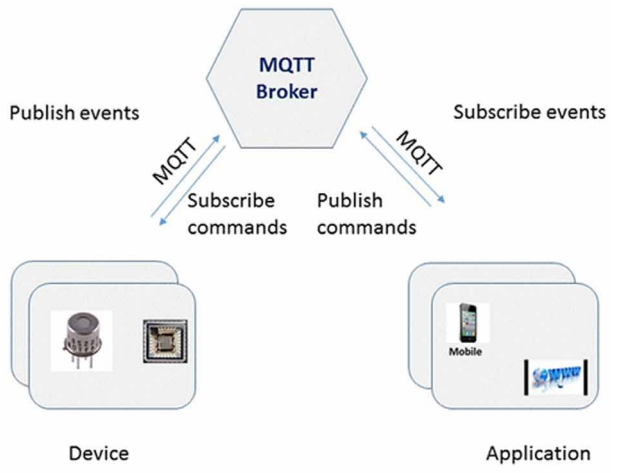


Figure 19. Graph of temperature



CONCLUSION

In this chapter, the designing and implementation of a Smart City based IoT applications have been described. With the help of sensors, DHT22 and BMP082, Wi-Fi Raspberry Pi 3 and MQTT, we can make systems that are used for transmitting environmental information such as humidity, temperature and pressure. The data is primarily sensed and sent by sensors to Raspberry Pi. Afterwards, with the help of Raspberry Pi and MQTT broker, we can monitor that data anywhere by the mobile device, computer system, and any embedded device that is subscribed.

REFERENCES

Andy Banks, D. L. (2012). *An introduction to the MQTT transport for Mobile Messaging & M2M*. Retrieved from https://developer.ibm.com/messaging/2012/12/05/introduction-mqtt-transport-mobile-messaging-m2m/

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787–2805. doi:10.1016/j.comnet.2010.05.010

Bhatt, C., Dey, N., & Ashour, A. (2017). *Internet of Things and Big Data Technologies for Next Generation Healthcare*. Springer. doi:10.1007/978-3-319-49736-5

Bhayani, M., Patel, M., & Bhatt, C. (n.d.). *Proceedings of the International Congress on Information and Communication Technology.* Retrieved from https://link.springer.com/chapter/10.1007/978-981-10-0767-5_37 doi:https://doi.org/10.1007/978-981-10-0767-5_37

Chen, H. W., & Lin, F. J. (2014). *Converging MQTT resources in ETSI standards based M2M platform.* Paper presented at the Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE. doi:10.1109/iThings.2014.52

Collina, M., Bartolucci, M., Vanelli-Coralli, A., & Corazza, G. E. (2014). *Internet of Things application layer protocol analysis over error and delay prone links.* Paper presented at the Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC), 2014 7th. doi:10.1109/ASMS-SPSC.2014.6934573

GibbonsD. (2017). *MQTT*. Retrieved from https://github.com/mqtt/mqtt.github.io/wiki/server-support

González García, C., García-Bustelo, C. P., Espada, J. P., & Cueva-Fernandez, G. (2014). Midgar: Generation of heterogeneous objects interconnecting applications. A Domain Specific Language proposal for Internet of Things scenarios. *Computer Networks*, *64*, 143–158. doi:10.1016/j.comnet.2014.02.010

Govindan, K., & Azad, A. P. (2015). *End-to-end service assurance in IoT MQTT-SN*. Paper presented at the Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE. doi:10.1109/CCNC.2015.7157991

Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008). *MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks*. Paper presented at the Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on.

Locke, D. (2010). *MQ Telemetry Transport (MQTT) V3.1 Protocol Specification*. Retrieved from http://www.ibm.com/developerworks/library/ws-mqtt/

Luzuriaga, J. E., Cano, J. C., Calafate, C., Manzoni, P., Perez, M., & Boronat, P. (2015). *Handling mobility in IoT applications using the MQTT protocol.* Paper presented at the Internet Technologies and Applications (ITA). doi:10.1109/ITechA.2015.7317403

Meteoric growth: The Internet of Things is scaling exponentially. (2017). Retrieved from https://www.internet-der-dinge.de/en.html

Raspberry Pi 3 Model B. (2017). Retrieved from https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Sensor, Types of Sensor. (2017). Retrieved from https://www.electrical4u.com/sensor-types-of-sensor/

Sensors: Different Types of Sensors. (2017). Retrieved from http://www.engineersgarage.com/articles/sensors

Shah, T., & Bhatt, C. M. (2014, April). The Internet of things: Technologies, Communications and Computing. *CSI Communications*, *38*(1), 7.

Singh, M., Rajan, M., Shivraj, V., & Balamuralidhar, P. (2015). *Secure MQTT for internet of things (IoT).* Paper presented at the Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on.

Tan, L. (2010). Future Internet: The Internet of Things. In *2010 3rd Int. Conf. Adv. Comput. Theory Eng*. IEEE. doi:10.1109/ICACTE.2010.5579543