Lab 06 - MQTT/ COAP IoT Communication Protocol - Tinker CAD Arduino

Name of student (Batch No / Roll No)

Divyang Bagla (D2/PD 33)

Performance of Experiment	Journal Submission	Total Marks	Remarks	Instructor Sign

Aim: To demonstrate MQTT/COAP protocols using message broker to subscribe and publish sensor data using Raspberry Pi/ ESP8266 boards / Beagle board/ TinkerCAD Arduino Uno.

Objectives:
1. To understand how sensor data will get published and subscribed
2. To learn various clients and brokers available for implementation

Journal content- Theory and frequently asked questions and experiment
1. What is MQTT and 2. What are the principles of MQTT
3. Explain MQTT protocol theory
4. List different cloud brokers. Also describe different ways of using Mosquitto broker.

Lab Sketch : Prepare labeled Lab sketch with TinkerCAD
Program Code
Conclusions

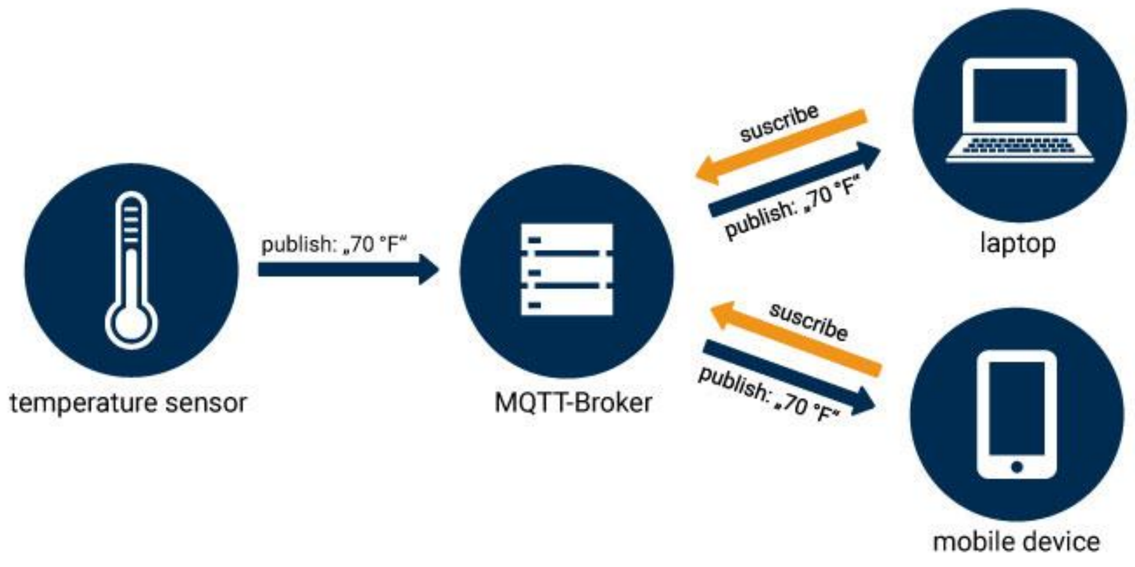
1. What is MQTT ?

MQTT is a publish/subscribe protocol that allows edge-of-network devices to publish to a broker. Clients connect to this broker, which then mediates communication between the two devices. Each device can subscribe, or register, to particular topics. When another client publishes a message on a subscribed topic, the broker forwards the message to any client that has subscribed.

MQTT is bidirectional, and maintains stateful session awareness. If an edge-of-network device loses connectivity, all subscribed clients will be notified with the "Last Will and Testament" feature of the MQTT server so that any authorized client in the system can publish a new value back to the edge-of-network device, maintaining bidirectional connectivity.

The lightweightness and efficiency of MQTT makes it possible to significantly increase the amount of data being monitored or controlled. Prior to the invention of MQTT, approximately 80% of data was being left at remote locations, even though various lines of business could have used this data to make smarter decisions. Now MQTT makes it possible to collect, transmit, and analyze more of the data being collected.

Unlike the usual poll/response model of many protocols, which tend to unnecessarily saturate data connections with unchanging data, MQTT's publish/subscribe model maximizes the available bandwidth.



2. What are the principles of MQTT

MQTT was designed for low-bandwidth, high latency networks in the late 1990s/early 2000s. As a result, the designers made a number of key choices which influenced the way it "looks and feels".

1. Simplicity, simplicity, simplicity! Don't add too many "bells and whistles" but provide a solid building block which can easily be integrated into other solutions. Be simple to implement.
2. Publish/subscribe messaging. Useful for most sensor applications, and enables devices to come online and publish "stuff" that hasn't been previously known about or predefined.
3. Zero administration (or as close as possible). Behave sensibly in response to unexpected actions and enable applications to "just work" e.g. dynamically create topics when needed.
4. Minimize the on-the-wire footprint. Add an absolute minimum of data overhead to any message. Be lightweight and bandwidth efficient.
5. Expect and cater for frequent network disruption (for low bandwidth, high latency, unreliable, high cost-to-run networks)... -> Last Will and Testament
6. Continuous session awareness -> Last Will and Testament
7. Expect that client applications may have very limited processing resources available.
8. Provide traditional messaging qualities of service where the environment allows. Provide "quality of service"
9. Data agnostic. Don't mandate content formats, remain flexible.

3. Explain MQTT protocol theory

MQTT (Message Queuing Telemetry Transport) is an ISO standard publish-subscribe-based messaging protocol. It works on top of the TCP/IP protocol. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The publish-subscribe messaging pattern requires a message broker.
An MQTT system consists of clients communicating with a server, often called a "broker". A client may be either a publisher of information or a subscriber. Each client can connect to the broker.

Information is organized in a hierarchy of topics. When a publisher has a new item of data to distribute, it sends a control message with the data to the connected broker. The broker then distributes the information to any clients that have subscribed to that topic. The publisher does not need to have any data on the number or locations of subscribers, and subscribers in turn do not have to be configured with any data about the publishers. If a broker receives a topic for which there are no current subscribers, it will discard the topic unless the publisher indicates that the topic is to be retained.

This allows new subscribers to a topic to receive the most current value rather than waiting for the next update from a publisher.
When a publishing client first connects to the broker, it can set up a default message to be sent to subscribers if the broker detects that the publishing client has unexpectedly disconnected from the broker.
Clients only interact with a broker, but a system may contain several broker servers that exchange data based on their current subscribers' topics. MQTT relies on the TCP protocol for data transmission. A variant, MQTT-SN, is used over other transports such as UDP or Bluetooth.

Message Types:

<u>Connect</u>

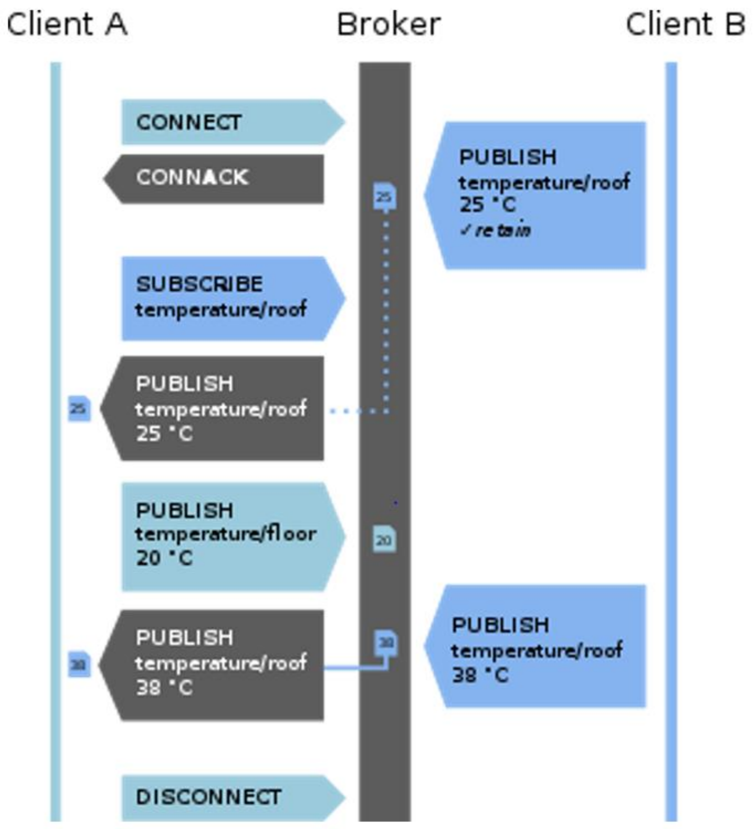
Waits for a connection to be established with the server and creates a link between the nodes.

<u>Disconnect</u>

Waits for the MQTT client to finish any work it must do, and for the <u>TCP/IP</u> session to disconnect.

<u>Publish</u>

Returns immediately to the application thread after passing the request to the MQTT client.



Publish/Subscribe

The MQTT protocol is based on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to a broker and subscribe to topics that they are interested in. Clients also connect to the broker and publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. The broker and MQTT act as a simple, common interface for everything to connect to. This means that you if you have clients that dump subscribed messages to a database, to Twitter, Cosm or even a simple text file, then it becomes very simple to add new sensors or other data input to a database, Twitter or so on.

Topics/Subscriptions

Messages in MQTT are published on topics. There is no need to configure a topic, publishing on it is enough. Topics are treated as a hierarchy, using a slash (/) as a separator. This allows sensible arrangement of common themes to be created, much in the same way as a filesystem.

4. List different cloud brokers. Also describe different ways of using Mosquitto broker.

Different Cloud Brokers :

- *AWS Service Broker*
- *IBM MultiCloud Management Service*
- *Cloudmore*
- *Jamcracker Cloud Services Storage*
- *Boomi*

Different ways of using Mosquitto broker

Mosquitto is an open source message broker (or server) that implements MQTT protocols. With its good community support, documentation, and ease of installation it has become one of the most popular MQTT brokers. Mosquitto is an open source iot.eclipse.org project. It implements the MQTT protocol versions 3.1 and 3.1.1. For more details please refer to http://mosquitto.org/.

a. Prerequisites

- An Ubuntu 16.04 server with root access
- Open port TCP:1883 on firewall (Check if port is available or not)

b. Mosquitto Broker Installation

Step One: Install Mosquitto

Update Ubuntu's package list and install the latest Mosquitto Broker available from it

```
$sudo apt-get update
$sudo apt-get install mosquito
```

The Mosquitto service will start after installation.

Step Two:

Install MQTT clients

```
$sudo apt-get install mosquitto-clients
```

Mosquitto clients help us easily test MQTT through a command line utility. We will use two command windows, one to subscribe to a topic named "test" and one to publish a message to it.

Topics are labels used by the broker to filter messages for each connected client. A client program subscribed to a topic "Home1/BedroomTemp" will only listen to messages published to the same topic by other clients.

Subscribe to topic "test"

```
mosquitto_sub -t "test"
```

Mosquito_sub is a subscribe client we installed in the previous command. Here we are specifying "-t" followed by a topic name.

Publish a message to topic "test"

Login to the terminal as a second instance and publish a message to the "test" topic.

```
$mosquitto_pub -m "message from mosquitto_pub client" -t "test"
```

Here the additional parameter "–m" is followed by the message we want to publish. Hit "Enter" and you should see a message from mosquitto_pub client displayed in other terminal where mosquito_sub client is running.

MQTT is the protocol of choice for M2M and IoT Applications. However, when it comes to selecting the MQTT broker, most of the times we resort to Cloud based Brokers.

Having a local MQTT Broker may have many advantages over Cloud based Brokers, like Security, Flexibility, Reliability, Low Latency, Cost Effectiveness, better QoS implementation etc.

Step Three:

Test Mosquitto MQTT Broker with MQTT Client:

For testing you can use any MQTT Client. However, if you have Python 2.7 Installed on your machine, you can test it with following sample Python scripts. To execute these Scripts, you must have Paho MQTT Client installed on your machine.

You can install it with pip command –

```
$pip install paho-mqtt
```

Once Paho Client Library is installed, you can execute following Python scripts (Don't forget to change "MQTT_BROKER" IP Address) –

- Publisher.py (Code is available at the end for reference)
- Subscriber.py (Code is available at the end for reference)

c. Uninstall Mosquitto MQTT Broker:

To uninstall Mosquitto you can use following command –

```
$sudo apt-get purge mosquitto
```

If you want to completely remove Mosquitto with its associated configuration files, use following command –

```
$sudo apt-get --purge remove mosquito
```

d. Test MQTT implementation using Mosquitto

MQTT using an Existing MQTT Broker on the Internet/cloud based on Paho Python

Several MQTT brokers are available on the Internet for use as test brokers. This guide uses the one at http://test.mosquitto.org.

Follow the steps in this section to use the Paho Python libraries that you installed earlier to setup a test topic called mytopic at http://test.mosquitto.org for MQTT message subscription and publication.

Perform these steps on your Development Computer.

1. Connect the Development Computer to the Internet.
2. Open a new Linux terminal window.
3. Go to the Paho examples directory:
   ```
   cd org.eclipse.paho.mqtt.python-1.1/examples
   ```
4. Modify the Subscriber.py script to change the mqttc.connect and mqttc.subscribe lines:
   ```
   mqttc.connect("test.mosquito.org",1883, 60)
       mqttc.subscribe("mytopic", 0)
   ```
   In this script, the following apply:
   - mqttc.connect
     - test.mosquitto.org: The Internet broker URL.
     - 1883: Network port used for MQTT messages.
     - 60: Timeout in seconds.

        - mqttc.subscribe
            - mytopic: Name of your new MQTT topic. The topic is established automatically on the broker if it does not already exist.
            - 0: Quality of Service.

5. Run the sub.py script:

```
python subscriber.py
```

Subscribing to a new topic called mytopic on the test.mosquitto.org broker automatically establishes that topic. Now, when the Gateway (or any other computer) publishes an MQTT message to mytopic on the test.mosquitto.org broker, the message will be sent to the Linux terminal where the sub.py script is running. Continue to Gateway Setup.

- Publisher.py (Code is available at the end for reference)
- Subscriber.py (Code is available at the end for reference)

Set up an MQTT Broker on a Local Network Using Mosquitto

Use the steps in this section if your goal is to use your Development Computer as a local network test broker using the open source application called mosquitto.

Note: For details about mosquitto use and syntax, see http://mosquitto.org, or use <mosquitto_command> --help, where <mosquitto_command> is the command that you need help with.

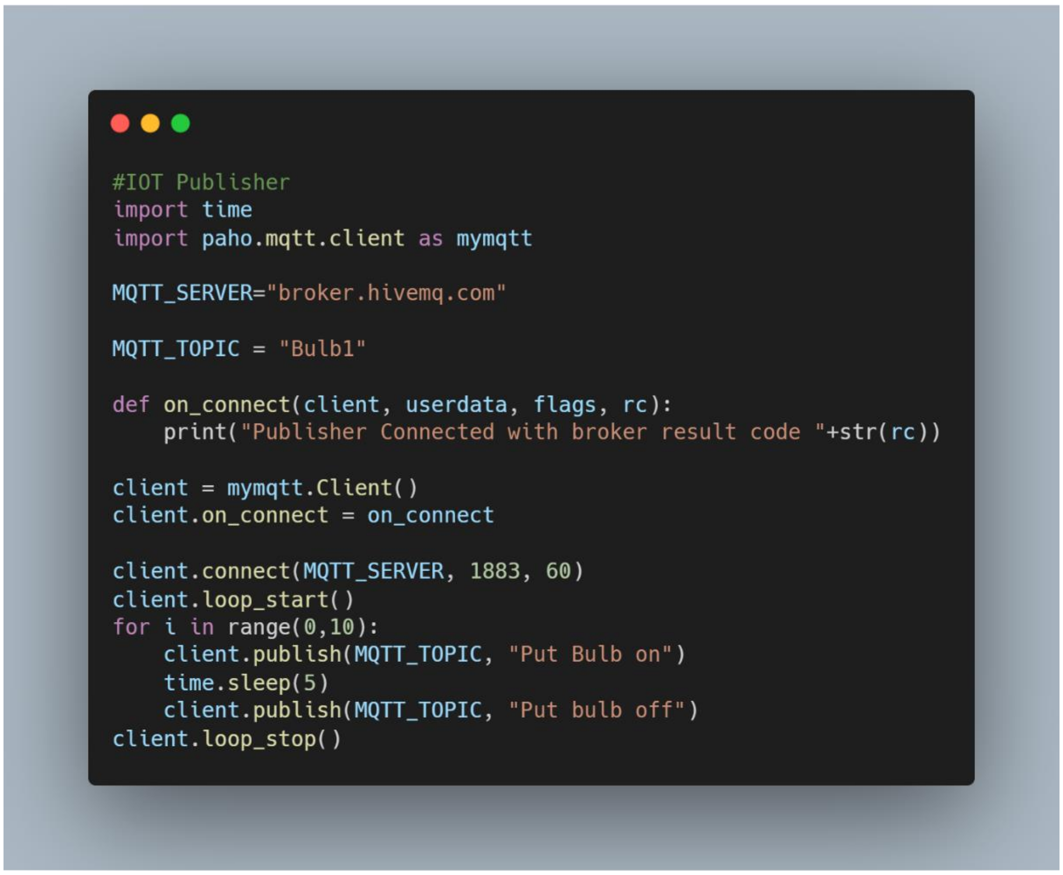
To start a broker process on the Development Computer:

1. Open a new Linux terminal window.
   The broker will run in this window. You can still receive MQTT messages on this Development Computer. The MQTT messages will be received in a different terminal window.
2. Install mosquitto if you do not already have it installed.
   ```
   sudo apt-get install mosquitto
   ```
3. Create a topic and subscribe to it with the mosquitto_sub command.
   ```
   mosquitto_sub -d -h localhost -t mytopic
   ```
   In this script, the following apply:
   - -d: Enable debug messages.
   - -h localhost: An alias to the development computer's local loopback IP address (typically 127.0.0.1).
   - -t mytopic: Name of the MQTT topic. The topic is established automatically on the broker if it does not already exist.

4. Check that the Development Computer is connected to the same network as the Gateway. Subscribing to a new topic called mytopic on the local host automatically starts the local broker and establishes that topic for other subscribers. Now, when the Gateway publishes an MQTT message to mytopic at the Development Computer's IP address, the message will be sent to the Linux terminal where the mosquitto_sub command is running.

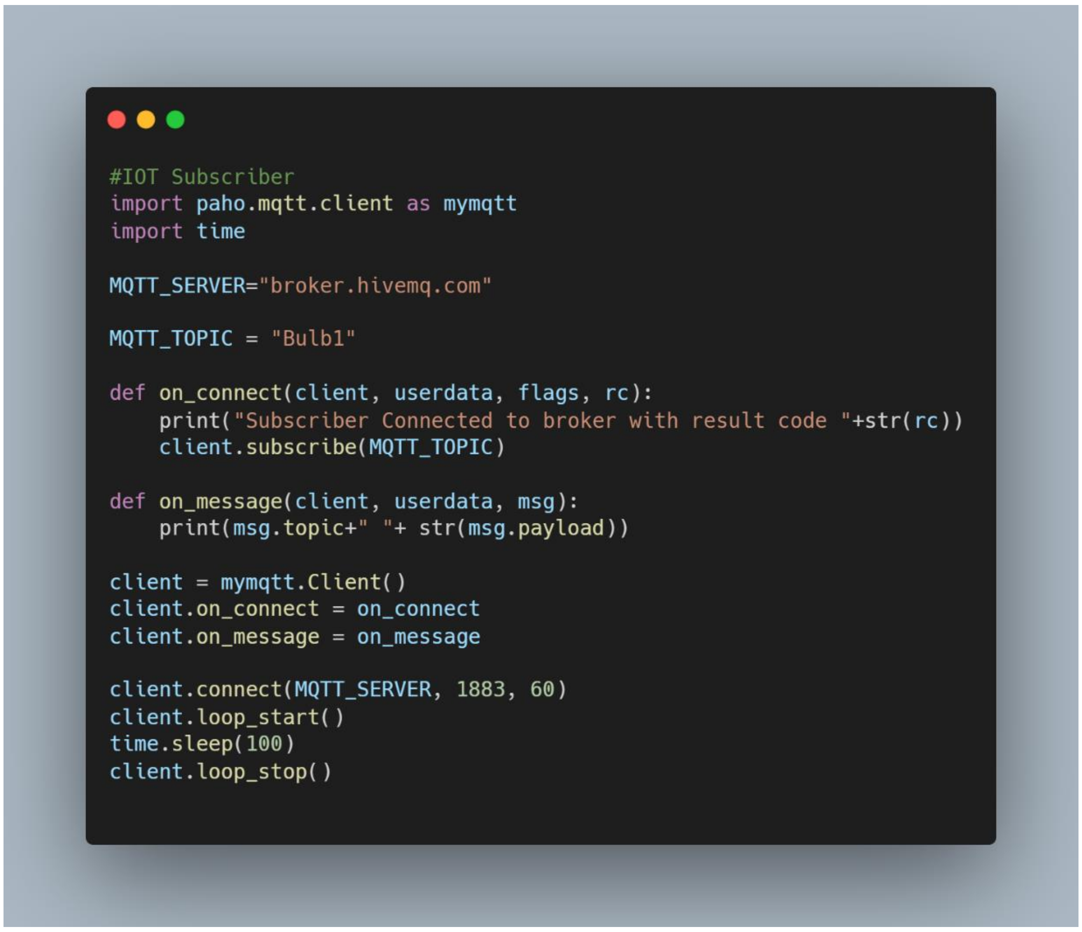
   Subscriber and publisher code: -

Publisher Code BY Divyang Bagla(PD 33)



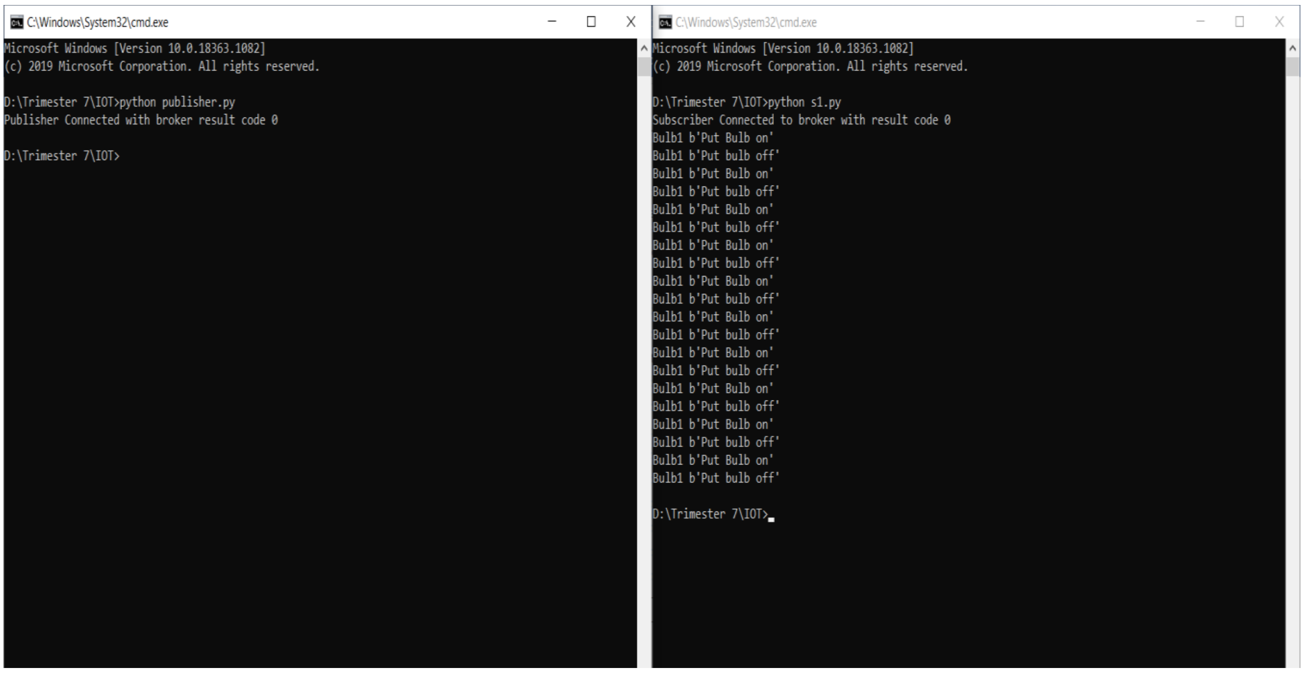
```python
#IOT Publisher
import time
import paho.mqtt.client as mymqtt

MQTT_SERVER="broker.hivemq.com"

MQTT_TOPIC = "Bulb1"

def on_connect(client, userdata, flags, rc):
    print("Publisher Connected with broker result code "+str(rc))

client = mymqtt.Client()
client.on_connect = on_connect

client.connect(MQTT_SERVER, 1883, 60)
client.loop_start()
for i in range(0,10):
    client.publish(MQTT_TOPIC, "Put Bulb on")
    time.sleep(5)
    client.publish(MQTT_TOPIC, "Put bulb off")
client.loop_stop()
```

Subscriber Code BY Divyang Bagla(PD 33)



```python
#IOT Subscriber
import paho.mqtt.client as mymqtt
import time

MQTT_SERVER="broker.hivemq.com"

MQTT_TOPIC = "Bulb1"

def on_connect(client, userdata, flags, rc):
    print("Subscriber Connected to broker with result code "+str(rc))
    client.subscribe(MQTT_TOPIC)

def on_message(client, userdata, msg):
    print(msg.topic+" "+ str(msg.payload))

client = mymqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect(MQTT_SERVER, 1883, 60)
client.loop_start()
time.sleep(100)
client.loop_stop()
```

OUPUT :-



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\Trimester 7\IOT>python publisher.py
Publisher Connected with broker result code 0

D:\Trimester 7\IOT>
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\Trimester 7\IOT>python s1.py
Subscriber Connected to broker with result code 0
Bulb1 b'Put Bulb on'
Bulb1 b'Put bulb off'
Bulb1 b'Put Bulb on'
Bulb1 b'Put bulb off'
Bulb1 b'Put Bulb on'
Bulb1 b'Put bulb off'
Bulb1 b'Put Bulb on'
Bulb1 b'Put bulb off'
Bulb1 b'Put Bulb on'
Bulb1 b'Put bulb off'
Bulb1 b'Put Bulb on'
Bulb1 b'Put bulb off'
Bulb1 b'Put Bulb on'
Bulb1 b'Put bulb off'
Bulb1 b'Put Bulb on'
Bulb1 b'Put bulb off'
Bulb1 b'Put Bulb on'
Bulb1 b'Put bulb off'
Bulb1 b'Put Bulb on'
Bulb1 b'Put bulb off'

D:\Trimester 7\IOT>
```

Conclusion

Thus we have studied how to use light weight messaging protocol so that sensors can publish and subscribe in over internet system and can communicate like one to one, one to many or many to many in order to disseminate the information for further processing.