Know and Understand MQTT protocol. Then implement and build a project to subscribe and publish on the public server of test.mosquitto.org.

**Contents**

- Introduction
- Step 1 - Getting Started
- Step 2 - Get PCBs for Your Projects Manufactured
- Step 3 - MQTT Broker
- Step 4 - Mosquitto Platform
- Step 5 - Publisher (ESP32)
- Step 6 - Subscriber (Windows PC)
- Comments

**Introduction**

MQTT stands for Message-Queue-Telemetry-Transport, is a *publish/subscribe* protocol for *machine-to-machine* communication. This simple protocol, is easy to implement for any client. Termed as the **Pub** and **Sub**, both are used for same purpose but with different methods.

**Materials**

- Mosquitto_MQTT_-_IoT_Platform_Series_mosquitto_mqtt_using_pubsub_client_library.ino

**Tools**
Step 1 - Getting Started

Here, there are 2 sections - **Publish and Subscribe**. And then there is a middleman - **Broker**. Let us see in depth

- IoT Devices play the role to collect sensor data and send to the cloud (broker). While **PC / Server / Mobile devices** play the role to monitor and receive the sensor data to be viewed - Here, **IoT Device** is a **Publisher**, and **PC Devices** are **Subscriber**.

**[EXAMPLE]** When a **user1** publishes an image on social media, then only the **user2** subscribed to **user1** can view/receive the image. Here, the **user1** is the **PUBLISHER**, **user2** is the **SUBSCRIBER**, and the **user1’s account** is the **BROKER**.

- According to the above analogy, the image that is published is the data, that was transferred from **user1** to **user2**. And that is the exact scenario in an MQTT Pub/Sub model.
- We have a more secure layer! To make sure the data is shared through a specific path, we call that ‘topic’. When **user1** publishes data or topic, the subscriber automatically receives if already connected to the broker. Hence, the LOW latency.

---

Step 2 - Get PCBs for Your Projects Manufactured

You must check out **PCBWAY** for ordering PCBs online for cheap!

You get 10 good-quality PCBs manufactured and shipped to your doorstep for cheap. You will also get a discount on shipping on your first order. Upload your Gerber files onto **PCBWAY** to get them manufactured with good quality and quick turnaround time.

**PCBWay** now could provide a complete product solution, from design to enclosure production. Check out their online Gerber viewer function. With reward points, you can get free stuff from their gift shop.
Step 3 - MQTT Broker

Whenever there is a pub-sub model used as a message communication protocol, we require a broker that can transfer the information in the required device. This can be done by sending the message under correct topic.

Let us understand this -

- A Broker is a runtime server (continuously running service), which can have 2 types of clients - Publisher (seller) & Subscriber (buyer)
- For instance, when a seller sells a product through a broker to a buyer, then it is using the Broker’s service to reach & find a secured buyer.
- Similarly, when publisher publishes a piece of information, the data reaches to the subscriber through the Broker.
- The broker is responsible for having specific storage space where it can expect data from the publisher to store temporarily and then send to the subscriber.
- In the pub-sub MQTT, clients talk to each other through an MQTT broker.
- There are many MQTT Brokers in the market. It is even possible to create our own broker, or use an open-source broker ‘paho’.
- For the current project, we shall first understand the mechanism and then watch a trial movement of data on Mosquitto MQTT Broker.
Step 4 - Mosquitto Platform

Now that we understand how MQTT works, let us use a cloud MQTT service and send data across the internet. In this article, we’ll be using Mosquitto MQTT - test.mosquitto.org

Under the Server section, we can see different ports provide feature-separated servers. These servers act like channels for sharing data over the cloud. Let us understand it first -

- **MQTT Broker Port (default: 1883):** This is the standard port used for MQTT communication. MQTT clients use it to connect to the Mosquitto broker and publish/subscribe to topics. It operates over TCP.

- **MQTT Broker SSL/TLS Port (default: 8883):** This is the secure version of the MQTT broker port. It uses SSL/TLS encryption to provide secure communication between MQTT clients and the Mosquitto broker. Clients connect to this port to establish a secure connection.

- **WebSocket Port (default: 9001):** Mosquitto also supports MQTT over WebSockets, allowing MQTT clients to connect to the broker using the WebSocket protocol. The WebSocket port is used for WebSocket-based MQTT communication.

- **WebSocket SSL/TLS Port (default: 9443):** This is the secure WebSocket port used for encrypted WebSocket-based MQTT communication. It provides a secure connection using SSL/TLS encryption.

We shall be using the **1883** port to send data and monitor. As we know, MQTT has 3 services - Publisher, Broker, and Subscriber. In this case, mosquitto MQTT Cloud is already playing the role of a broker. Now, we'll be using the **ESP32 Dev Board**, which has a WiFi chip and is able to connect to the Internet, playing the role of a Publisher for sharing its temperature and humidity data from the sensor.

On the other hand, we shall use the PC to view this data as a **Subscriber**. This will enable us to fully understand the working principle of the MQTT protocol used in IoT Communication between devices.

Step 5 - Publisher (ESP32)

```
#define <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "XXXXXXXXXX";
const char* password = "XXXXXXXXXX";
const char* mqtt_server = "test.mosquitto.org";
```

The provided code is an Arduino sketch that uses the ESP32 WiFi module and the PubSubClient library to connect to an MQTT broker and publish temperature and humidity data. Let's break down the code step by step:

1. **Include necessary libraries:**

   ```
   #include <WiFi.h>
   #include <PubSubClient.h>
   ```

2. **Define WiFi & MQTT Server variables:**

   ```
   const char* ssid = "XXXXXXXXXX";
   const char* password = "XXXXXXXXXX";
   const char* mqtt_server = "test.mosquitto.org";
   ```

   These variables store the SSID (network name) and password for the WiFi network you want to connect to. The variable holds the IP address or hostname of the MQTT broker.
3. Declare global variables and objects:

```c
WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;
float temperature = 0;
float humidity = 0;
```

Here, a WiFi client object () and an MQTT client object () are declared. The variable stores the timestamp of the last message, and the is a character array for message storage. The , , and variables are used to hold the respective sensor values.

4. Setup function:

```c
void setup()
{
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);
}
```

The function is called once at the start of the program. It initializes the serial communication, sets up the WiFi connection, configures the MQTT server and port, and sets the callback function to handle incoming messages.

5. WiFi setup function:

```c
void setup_wifi()
{
    // ...
}
```

The function handles the connection to the WiFi network using the provided SSID and password. It waits until the connection is established and prints the local IP address to the serial monitor.

6. MQTT callback function:

```c
void callback(char* topic, byte* message, unsigned int length)
{
    // ...
}
```

This function is called when a message is received from the MQTT broker. It prints the received message along with the corresponding topic.

7. MQTT reconnection function:

```c
void reconnect()
{
    // ...
}
```

The function is responsible for reconnecting to the MQTT broker if the connection is lost. It attempts to connect to the broker using a randomly generated client ID. If the connection is successful, it prints a success message. Otherwise, it waits for 5 seconds before retrying.

8. Main loop:
void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();
    long now = millis();
    if (now - lastMsg > 2000) {
        lastMsg = now;
        sendData();
    }
}

loop() setup() client.loop() void sendData() {
    // ...
}

The function is responsible for publishing temperature and humidity data to specific MQTT topics. It generates random values for temperature and humidity, converts them to strings, and publishes them along with the corresponding topic.

9. Publish sensor data function:

- Publish a gap message:

  client.publish("schoolofiot/gap", "----------");

  This line publishes a message consisting of a series of dashes (--) to the MQTT topic "schoolofiot/gap". It is used to indicate a separation or gap between different sets of data.

- Read and publish temperature data:

  temperature = random(30, 40);
  char tempString[8];
  dtostrf(temperature, 1, 2, tempString);
  Serial.print("Temperature: ");
  Serial.println(tempString);
  String tempdata = "Temperature: " + String(tempString);
  client.publish("schoolofiot/temperature", tempdata.c_str());

These lines generate a random temperature value between 30 and 40 degrees, store it in the variable, and use \texttt{\texttt{dtostrf}} function to convert decimal point data to String.

\texttt{\texttt{dtostrf}}(floatValue, minStringWidth, numAfterDecimal, charBuffer_to_store_string); This function takes four parameters to convert double into an ASCII value stored inside string:

1. floatValue: The first parameter that takes the float value we want to convert into a string.

2. minStringWidth: This is the second parameter that defines the minimum field width of the output string.

3. numAfterDecimal: The third parameter is precision which describes the number of digits after the decimal point.

4. charBuffer: Final argument is where the string will be stored. This is a kind of char array having a defined size.
The temperature value is then printed to the serial monitor and concatenated with the string "Temperature: " The resulting string is stored in the `client.publish()` function.

- **Read and publish humidity data:**

```c
humidity = random(60, 70);
char humString[8];
dtostrf(humidity, 1, 2, humString);
Serial.print("Humidity: ");
Serial.println(humString);

"mString humdata = "Humidity: " + String(humString); client.publish("schoolofiot/humidity", humdata.c_str());
```

These lines generate a random humidity value between 60 and 70 percent, store it in the variable. Overall, the function generates random temperature and humidity values, converts them to strings, and publishes them to specific MQTT topics for further processing or monitoring.

**Final Code can be found in the Code section**

But to confirm this, we also need to read the data from the other side - **Subscriber**.
Step 6 - Subscriber (Windows PC)

To set up the Subscriber on PC, we need to install Mosquitto MQTT Application. This application can create a broker, publisher & subscriber - all sections.

To install Mosquitto MQTT on your PC from the official website and make changes to the configuration file for listener 1883 and allow anonymous connections, you can follow these steps:

1. Download and Install Mosquitto:
   - Go to the official Mosquitto website (https://mosquitto.org/).
   - Navigate to the "Downloads" section.
   - Choose the appropriate installer for your operating system (Windows x64 in this case) and download it.
   - Install the application in desired location.

2. Edit Configuration File:
   - Open the installation directory where Mosquitto is installed.
   - Locate the file (usually found in the main directory).
   - Open in a text editor of your choice. Add the below 2 lines -
     ```
     listener 1883
     allow_anonymous true
     ```
   - It should look somewhat like this -
   - We can uncomment ad make changes in the file as well, but adding only 2 lines on the top is more simple and noticeable.

3. Run Mosquitto Subscriber:
   - We can run the Mosquitto broker and then subscribe to the topic we desire. But running directly the subscriber is best in our case.
   - Open the installation directory where the mosquitto.exe along with mosquitto.conf is present.
   - Run the PowerShell/CMD terminal from within the directory. For windows, open the directory > Press shift + right-mouse-button(right-click), and we'd see options for running a terminal like powershell.
   - On the terminal, enter below command -
     ```
     mosquitto_sub -h test.mosquitto.org -t "schoolofiot/#"
     ```
   - In the above command, if you noticed, I did not subscribe to a specific topic. As per the topics we published (from ESP32), like "schoolofiot/gap", "schoolofiot/temperature" or "schoolofiot/humidity".
     - The reason is, gap, temperature & humidity comes under the general topic of schoolofiot level. So, to access/view any data published as a sub-level of schoolofiot, we can use '#'.
     - Apart from this, in case we need to subscribe to a specific topic (like temperature), we can use command like this -
       ```
       mosquitto_sub -h test.mosquitto.org -t "schoolofiot/temperature"
       ```
   - Therefore, no matter what name is put under the general topic, we can subscribe to it and view all of them together.

Hurray! 🎉

We have learned another IoT Platform - Mosquitto MQTT (By Eclipse)