

Transmitting data over WiFi using MQTT

MQTT (Message Queuing Telemetry Transport) has gained a lot of prominence in the context of IoT devices. It is a protocol that runs generally over TCP/IP. Instead of the server–client model that we saw for HTTP, MQTT uses the broker–client model. Wikipedia defines MQTT brokers and clients as –

An MQTT broker is a server that receives all messages from the clients and then routes the messages to the appropriate destination clients. An MQTT client is any device (from a micro controller up to a full–fledged server) that runs an MQTT library and connects to an MQTT broker over a network.

Think of the broker as a service like Medium. The topics would be the Medium publications, and the clients would be the Medium users. A user (client) can post to a publication, and another user (client) who has subscribed to that publication (topic) would be told that a new post is available for reading. By now, you would have understood a major difference between HTTP and MQTT. In HTTP, your messages are directly sent to the intended server and you even get an acknowledgment in the form of status codes. In MQTT, you just send messages to the broker in the hope that your intended server(s) will take it from there. Several features of MQTT turn out to be a boon if you are resource–constrained. They are listed below –

- With MQTT, header overheads are very short and throughput is high. This helps save time and also battery.
- MQTT sends information as a byte array instead of the text format. This makes the message lightweight.
- Because MQTT isn't dependent on the response from the server, the client is independent and can go to sleep (conserve battery) as soon as it has transmitted the message.

These are just some of the points which have resulted in the popularity of MQTT. You can get a more detailed comparison between MQTT and HTTP [here](#).

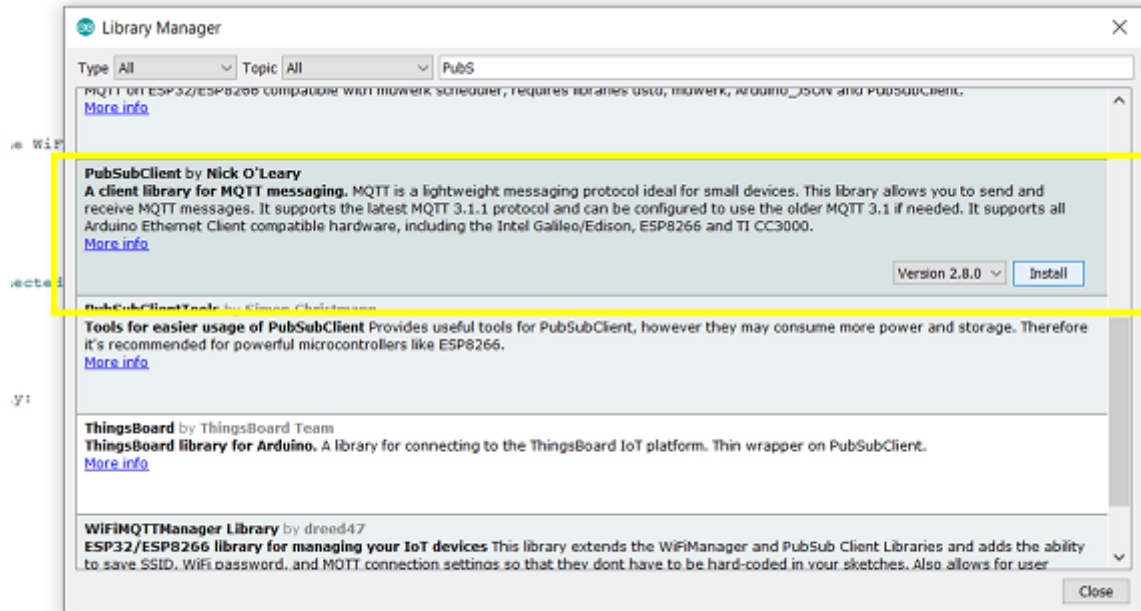
Code Walkthrough

In general, testing MQTT requires you to sign up for a free/ paid account with a broker. AWS IoT and Azure IoT are very popular platforms providing MQTT broker services, but they come with a lengthy signup and configuration process. Luckily, there is a free broker service from HiveMQ which can be used for testing MQTT without any signup or configuration. It is ideal for those of you who are new to MQTT and just want to get your hands dirty, and also lets you focus more on the firmware of ESP32. Therefore, that is the broker we will be using for this chapter. Of cou

because it is a free service, there will be limitations. You can't share sensitive information, because all your messages are public, anyone can subscribe to your topics. For testing purposes, of course, these limitations won't matter.

The code can be found on GitHub

We will be using the PubSubClient library. You can install it from Tools -> Manage Libraries.



Once the library is installed, we include WiFi and PubSubClient libraries.

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

Next, we will define some constants. Remember to replace the WiFi credentials. The `mqttServer` and `mqttPort` are the once mandated by <http://www.mqtt-dashboard.com/>. The `mqtt_client_name`, `mqtt_pub_topic` and `mqtt_sub_topic` can be any strings of your choice. Just make sure that you do change their values. If multiple users copy the same code from this tutorial, you will receive a lot of messages from unknown clients when testing.

We also define the `WiFiClient` and `mqttClient` object. The `MQTTClient` object requires the network client as an argument. If you are using Ethernet, you would provide the Ethernet client as an argument. Since we are using WiFi, we have provided the WiFi client as an argument.

```
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

//The broker and port are provided by http://www.mqtt-dashboard.com/
char *mqttServer = "broker.hivemq.com";
int mqttPort = 1883;

//Replace these 3 with the strings of your choice
```

```

const char* mqtt_client_name = "ESPYS2111";
const char* mqtt_pub_topic = "/ys/testpub"; //The topic to which our client will publish
const char* mqtt_sub_topic = "/ys/testsub"; //The topic to which our client will subscribe

WiFiClient client;
PubSubClient mqttClient(client);

```

Next, we define the callback function. A callback function is an interrupt function. Every time a new message is received from a subscribed topic, this function will be triggered. It has three arguments– the topic from which the message was received, the message as a byte array, and the length of the message. You can do whatever you want to do with that message (store it in SPIFFS, send it to another topic, and so on). Here, we are just printing the topic and the message.

```

void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message received from: "); Serial.println(topic);
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
    }
    Serial.println();
    Serial.println();
}

```

In the setup, we connect to the WiFi like in every other sketch. The last two lines concern MQTT. We set the server and port for MQTT and also the callback function.

```

void setup() {
    // put your setup code here, to run once:
    Serial.begin(115200);
    WiFi.mode(WIFI_STA); //The WiFi is in station mode
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println(""); Serial.print("WiFi connected to: "); Serial.println(ssid);
    delay(2000);
    mqttClient.setServer(mqttServer, mqttPort);
    mqttClient.setCallback(callback);
}

```

Within the loop, we do the following:

- If the client is not connected to the broker, we connect it using our client name.
- Once connected, we also subscribe our client to the `mqtt_sub_topic`.
- We then publish a message to `mqtt_pub_topic`
- We then run the **`mqttClient.loop()`**. This `loop()` function should be called regularly. It maintains the connection of the client with the broker and also helps the client process incoming messages. If you don't have this **`mqttClient.loop()`** line, you will be able to publish to `mqtt_pub_topic`, but won't get messages from `mqtt_sub_topic`, because the incoming messages are processed only when this line is called.
- Finally, we wait for 5 seconds, before starting this cycle again.

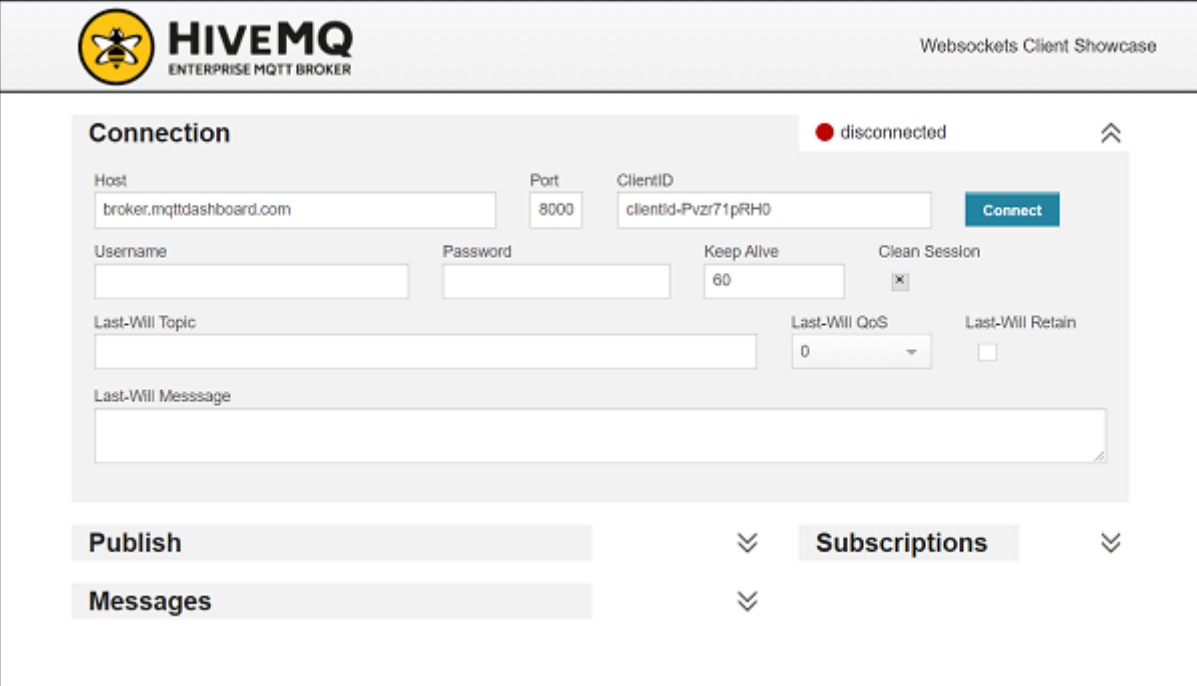
```
void loop() {  
    // put your main code here, to run repeatedly:  
    if (!mqttClient.connected()){  
        while (!mqttClient.connected()){  
            if(mqttClient.connect(mqtt_client_name)){  
                Serial.println("MQTT Connected!");  
                mqttClient.subscribe(mqtt_sub_topic);  
            }  
            else{  
                Serial.print(".");  
            }  
        }  
    }  
    mqttClient.publish(mqtt_pub_topic, "TestMsg");  
    Serial.println("Message published");  
    mqttClient.loop();  
    delay(5000);  
}
```

Testing the Code

In order to test the above code, you need to go to www.hivemq.com

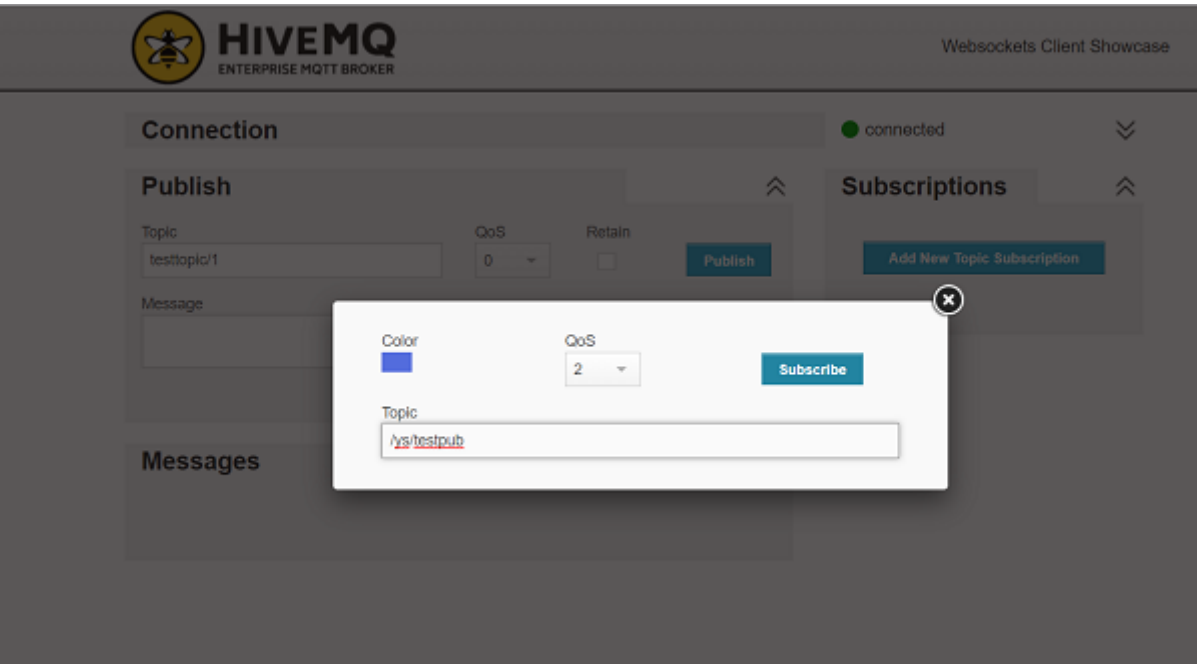
Follow these steps once you are on that webpage –

- Click Connect



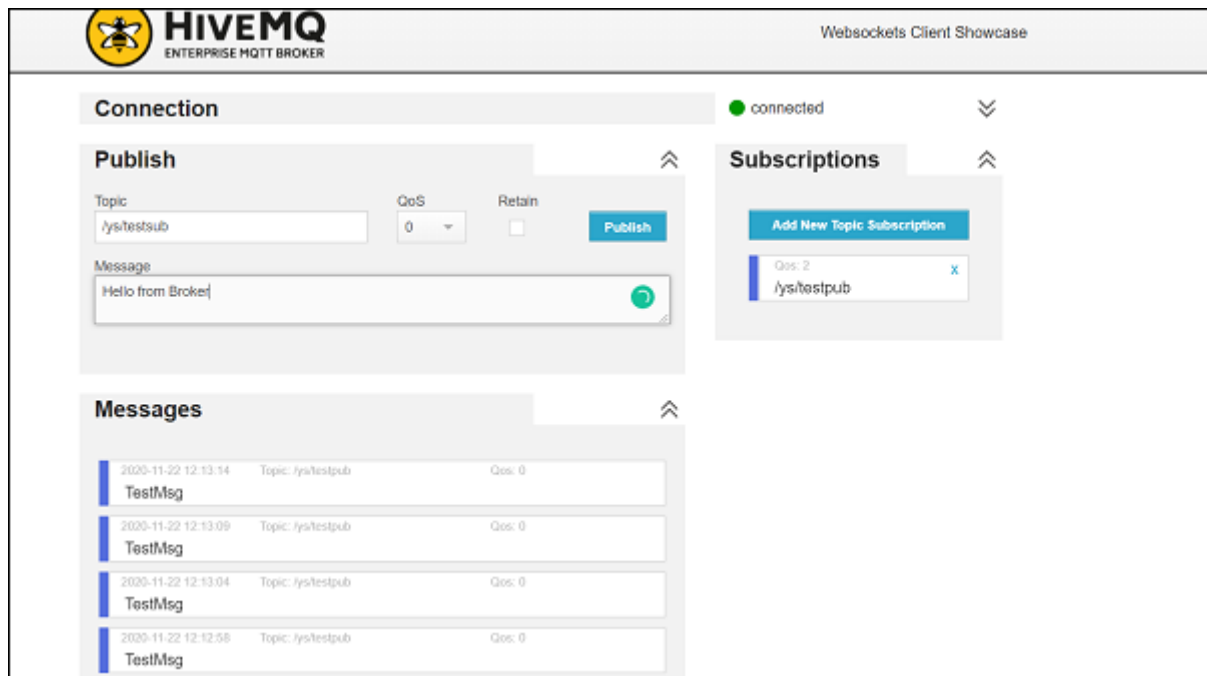
The image shows the Hivemq Websockets Client Showcase interface. At the top, there is a logo for Hivemq (a bee) and the text "HIVEMQ ENTERPRISE MQTT BROKER". To the right, it says "Websockets Client Showcase". The main section is titled "Connection" and shows a status of "disconnected" with a red dot. Below this, there are input fields for "Host" (broker.mqttdashboard.com), "Port" (8000), and "ClientID" (clientid-Pvzr71pRH0). There is a "Connect" button. Below these are fields for "Username", "Password", "Keep Alive" (60), "Clean Session" (checked), "Last-Will Topic", "Last-Will QoS" (0), "Last-Will Retain" (unchecked), and "Last-Will Message". At the bottom, there are tabs for "Publish", "Subscriptions", and "Messages".

- Click on Add New Topic Subscription and enter the name of the topic to which your ESP32 will publish (/ys/testpub in this case)

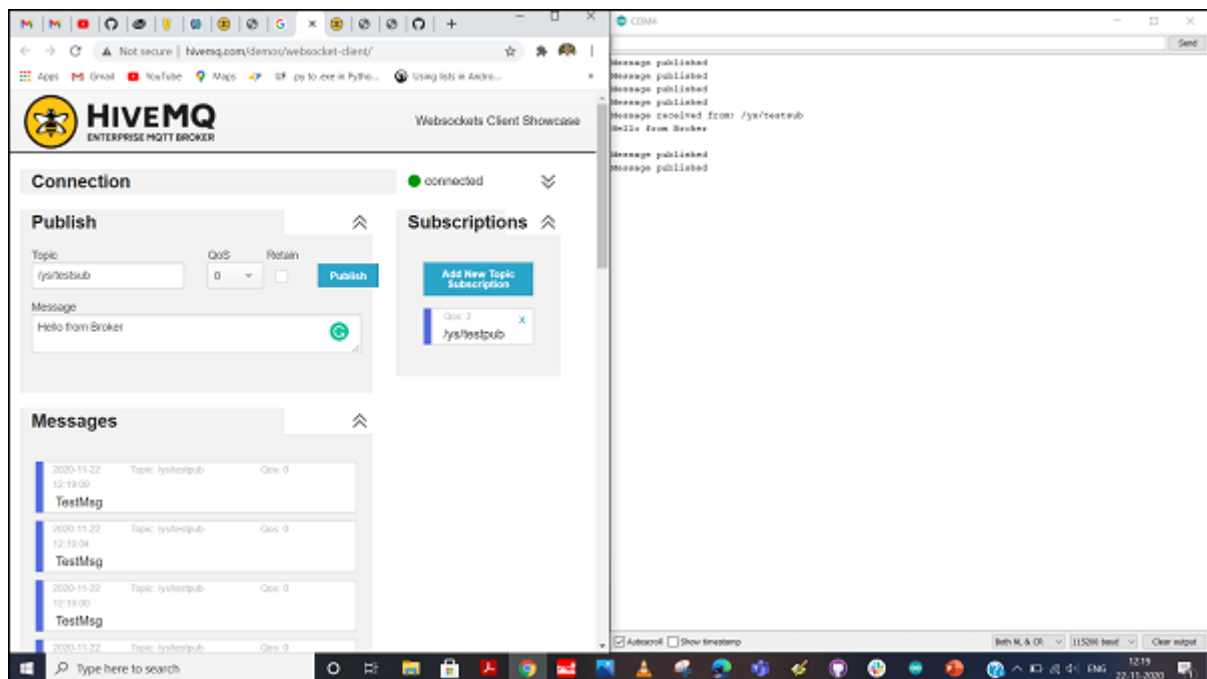


The image shows the Hivemq Websockets Client Showcase interface with the "Subscriptions" tab selected. The status is now "connected" with a green dot. A modal dialog is open for adding a new subscription. It has a "Color" dropdown (blue), a "QoS" dropdown (2), and a "Subscribe" button. Below this, there is a "Topic" input field containing "/ys/testpub". The background interface shows the "Publish" tab with fields for "Topic" (testtopic/1), "QoS" (0), "Retain" (unchecked), and a "Publish" button. The "Messages" tab is also visible at the bottom.

- Once you flash your ESP32, you will start receiving messages on that topic every 5 seconds.



- Next, to test reception of message on ESP32, enter the name of the topic your ESP32 is subscribed to (ys/testsub in this case), then type a message in the message box and click publish. You should see the message on the Serial Monitor.



Congratulations!! You've tested both publish and subscribe using MQTT on ESP32.

References

- MQTT - Wikipedia
- ESP-MQTT