Practical IoT (Internet of Things)

BSCIS – DCIS, PIEAS

Lab 04: Using MQTT Protocol in IoT

Objective

- Understand the fundamentals of MQTT—a lightweight messaging protocol designed for IoT.
- Learn the publish/subscribe communication pattern.
- Configure ESP32 / WeMos D1 Mini (ESP8266) to connect to a public MQTT broker (e.g., test.mosquitto.org / broker.hivemq.com / iot.intellihouse.com.pk).
- Publish sensor/button data to a topic (Collaborate with another group).
- Subscribe to a topic and control actuators based on received data (Collaborate with another group).
- Implement code that publishes messages to a topic and subscribes to another topic to receive messages (single device as publisher and subscriber).
- Gain hands-on experience with IoT messaging, debugging, and integrating sensor/actuator control through MQTT.

Required Components

- WeMos D1 Mini (ESP8266) / ESP32 – Optionally 2 boards (one each for publish and subscribe)
- LED (any color) plus a 220Ω current-limiting resistor
- One push button with optional pullup resistor
- Breadboard and Jumper Wires
- USB cable for programming
- Reliable WiFi connection with internet (if using public broker). Internet is not required if using local hosted broker.
- (Optional) MQTT client tool (e.g., MQTT Explorer, MQTT.fx or an online MQTT dashboard) for testing

Background

What is MQTT?

- MQTT (Message Queuing Telemetry Transport) is a lightweight, publish/subscribe messaging protocol.
- It is designed for connections with remote locations where network bandwidth is limited.
- The protocol uses a broker to relay messages between publishers and subscribers.

Publish/Subscribe Model

- Publisher: Sends messages to a topic.
- Subscriber: Listens for messages on a topic.

- Broker: Manages topics and routes messages from publishers to subscribers.

Public MQTT Broker

- For this lab, we will use a public MQTT broker such as **test.mosquitto.org**. No authentication is needed, which simplifies initial testing. Default MQTT broker port is 1883.

Part 1: Setting Up the Environment

Step 1: Install Required Libraries

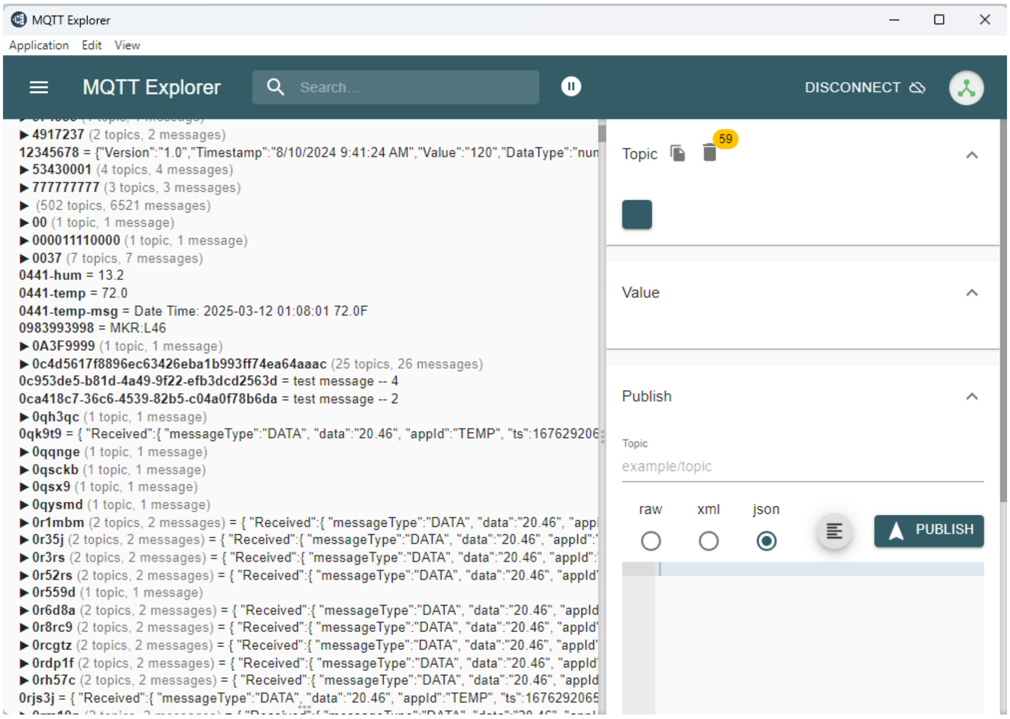
1. Open the Arduino IDE.
2. Go to Sketch → Include Library → Manage Libraries...
3. Install the PubSubClient library by Nick O'Leary.
4. (Optional) Update the ESP8266 board package if necessary.

Step 2: Configure Your WiFi Credentials

- Prepare your WiFi SSID and password; these will be used in the code. You can also use your mobile hotspot for the purpose.

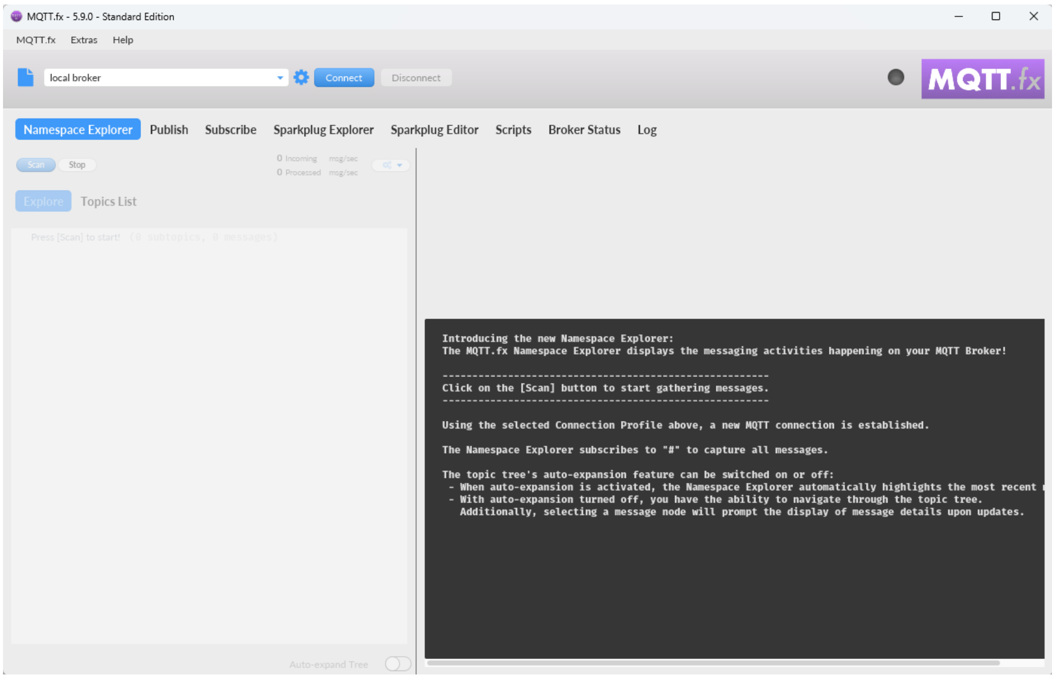
Step 3: Install Client Software for visualization (Optional)

1. Download one of the MQTT client software.
    - MQTT Explorer (https://mqtt-explorer.com/)
    - MQTT.fx (https://www.softblade.de/) – Need to get trial license key
2. Install the downloaded software.



```
▶ 4917237 (2 topics, 2 messages)
12345678 = {"Version":"1.0","Timestamp":"8/10/2024 9:41:24 AM","Value":"120","DataType":"nun
▶ 53430001 (4 topics, 4 messages)
▶ 777777777 (3 topics, 3 messages)
▶  (502 topics, 6521 messages)
▶ 00 (1 topic, 1 message)
▶ 000011110000 (1 topic, 1 message)
▶ 0037 (7 topics, 7 messages)
0441-hum = 13.2
0441-temp = 72.0
0441-temp-msg = Date Time: 2025-03-12 01:08:01 72.0F
0983993998 = MKR:L46
▶ 0A3F9999 (1 topic, 1 message)
▶ 0c4d5617f8896ec63426eba1b993ff74ea64aaac (25 topics, 26 messages)
0c953de5-b81d-4a49-9f22-efb3dcd2563d = test message -- 4
0ca418c7-36c6-4539-82b5-c04a0f78b6da = test message -- 2
▶ 0qh3qc (1 topic, 1 message)
0qk9t9 = { "Received":{ "messageType":"DATA", "data":"20.46", "appld":"TEMP", "ts":167629206
▶ 0qqnge (1 topic, 1 message)
▶ 0qsckb (1 topic, 1 message)
▶ 0qsx9 (1 topic, 1 message)
▶ 0qysmd (1 topic, 1 message)
▶ 0r1mbm (2 topics, 2 messages) = { "Received":{ "messageType":"DATA", "data":"20.46", "appl
▶ 0r35j (2 topics, 2 messages) = { "Received":{ "messageType":"DATA", "data":"20.46", "appld":
▶ 0r3rs (2 topics, 2 messages) = { "Received":{ "messageType":"DATA", "data":"20.46", "appld":
▶ 0r52rs (2 topics, 2 messages) = { "Received":{ "messageType":"DATA", "data":"20.46", "appld"
▶ 0r559d (1 topic, 1 message)
▶ 0r6d8a (2 topics, 2 messages) = { "Received":{ "messageType":"DATA", "data":"20.46", "appld
▶ 0r8rc9 (2 topics, 2 messages) = { "Received":{ "messageType":"DATA", "data":"20.46", "appld"
▶ 0rcgtz (2 topics, 2 messages) = { "Received":{ "messageType":"DATA", "data":"20.46", "appld"
▶ 0rdp1f (2 topics, 2 messages) = { "Received":{ "messageType":"DATA", "data":"20.46", "appld"
▶ 0rh57c (2 topics, 2 messages) = { "Received":{ "messageType":"DATA", "data":"20.46", "appld
0rjs3j = { "Received":{ "messageType":"DATA", "data":"20.46", "appld":"TEMP", "ts":1676292065
```

Screenshot of MQTT Explorer



```
Introducing the new Namespace Explorer:
The MQTT.fx Namespace Explorer displays the messaging activities happening on your MQTT Broker!

-------------------------------------------------------
Click on the [Scan] button to start gathering messages.
-------------------------------------------------------

Using the selected Connection Profile above, a new MQTT connection is established.

The Namespace Explorer subscribes to "#" to capture all messages.

The topic tree's auto-expansion feature can be switched on or off:
 - When auto-expansion is activated, the Namespace Explorer automatically highlights the most recent
 - With auto-expansion turned off, you have the ability to navigate through the topic tree.
   Additionally, selecting a message node will prompt the display of message details upon updates.
```

Screenshot of MQTT.fx

Part 2: Configure the MQTT Publisher (Button)

Program the publisher ESP8266 to:

- Connect to Wi-Fi.
- Connect to the MQTT broker.
- Read the button state.
- Publish `ON`/`OFF` to the topic `pieas/iotlab/4/group/[group_id]/button` when the button is pressed.

Code Template:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Wi-Fi Credentials
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

// MQTT Broker
const char* mqtt_server = "test.mosquitto.org"; // Check Broker
const int mqtt_port = 1883;
const char* topic = "pieas/iotlab/4/group/1/button"; // Check Group No.

WiFiClient espClient;
PubSubClient client(espClient);

// Button Setup
const int buttonPin = D2;
bool lastButtonState = LOW;
```

```
void setup() {
  Serial.begin(115200);
  pinMode(buttonPin, INPUT);
  setupWiFi();
  client.setServer(mqtt_server, mqtt_port);
}

void setupWiFi() {
  delay(10);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi Connected");
}

void reconnect() {
  while (!client.connected()) {
    if (client.connect("p.iot.0.567")) {  // Use a random unique device id.
      Serial.println("MQTT Connected");
    } else {
      Serial.println("Trying to connect MQTT ... ");
      delay(5000);
    }
  }
}

void loop() {
  if (!client.connected()) reconnect();
  client.loop();

  bool buttonState = digitalRead(buttonPin);
  if (buttonState != lastButtonState) {
    if (buttonState == HIGH) {
      Serial.println("Button is ON");
      client.publish(topic, "ON");
    } else {
      Serial.println("Button is OFF");
      client.publish(topic, "OFF");
    }
    lastButtonState = buttonState;
  }
  delay(50);
}
```

Code Explanation

WiFi Connection:

- The ***setup_wifi()*** function connects the ESP8266 to your WiFi network using the provided SSID and password.

MQTT Connection:

- The ***reconnect()*** function attempts to connect to the MQTT broker (test.mosquitto.org) and subscribes to the topic "iot/led".

Publishing Messages:

- In the ***loop()***, a message is published to the "iot/test" topic every 10 seconds.

Maintaining the Connection:

- The ***client.loop()*** call processes incoming messages and maintains the connection.

Part 3: Configure the MQTT Subscriber (LED)

Program the subscriber ESP8266 to:

- Connect to Wi-Fi.
- Connect to the MQTT broker.
- Subscribe to the topic `pieas/iotlab/4/group/[group_id]/button`
- Turn the LED `ON/OFF` based on received messages.

Code Template:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Wi-Fi Credentials (same as publisher)
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

// MQTT Broker
const char* mqtt_server = "test.mosquitto.org"; // Check Broker
const int mqtt_port = 1883;
const char* topic = "pieas/iotlab/4/group/1/button"; // Match publisher

WiFiClient espClient;
PubSubClient client(espClient);

// LED Setup
const int ledPin = D4;

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH); // Initially LED is OFF
  setupWiFi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback); // Handle incoming messages
}

void setupWiFi() {
  delay(10);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi Connected");
}
```

```
void reconnect() {
  while (!client.connected()) {
    if (client.connect("p.iot.0.568")) {  // Use a random unique device id.
      Serial.println("MQTT Connected");
    client.subscribe(topic);    // Additional Line to subscribe topic
    } else {
      Serial.println("Trying to connect MQTT ... ");
      delay(5000);
    }
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  //Print received message on Serial Console
  Serial.print("Got Message: ");
  Serial.print(topic);
  Serial.print(" > ");
  Serial.println(message);

  //Set LED state based on the received Message
  if (message == "ON") {
    digitalWrite(ledPin, LOW);
  } else if (message == "OFF") {
    digitalWrite(ledPin, HIGH);
  }
}

void loop() {
  if (!client.connected()) reconnect();
  client.subscribe(topic);
  client.loop();
}
```

Code Explanation

WiFi Connection:

- The ***setup_wifi()*** function connects the ESP8266 to your WiFi network using the provided SSID and password.

MQTT Connection:

- The ***reconnect()*** function attempts to connect to the MQTT broker (test.mosquitto.org) and subscribes to the topic `pieas/iotlab/4/group/[group_id]/button`.

Callback Function:

- The ***callback()*** function processes incoming MQTT messages on subscribed topics and prints them to the Serial Monitor.

Maintaining the Connection:

- The ***client.loop()*** call processes incoming messages and maintains the connection.
- We also need to re-subscribe to all the topic after device loss connection.

Part 4: Both Publisher and Subscriber on Single Device

Program the publisher ESP8266 to:

- Connect to Wi-Fi.
- Connect to the MQTT broker.
- Subscribe to the topic `pieas/iotlab/4/group/[group_id]/button`
- Read the button state.
- Publish `ON/OFF` to the topic `pieas/iotlab/4/group/[group_id]/button` when the button is pressed.
- Turn the LED `ON/OFF` based on received messages.

Code Template:

```
// Wi-Fi Credentials (same as publisher)
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";

// MQTT Broker
const char* mqtt_server = "test.mosquitto.org"; // Check Broker
const int mqtt_port = 1883;
const char* topic = "pieas/iotlab/4/group/1/button"; // Match publisher

WiFiClient espClient;
PubSubClient client(espClient);

// Button Setup
const int buttonPin = D2;
bool lastButtonState = LOW;

// LED Setup
const int ledPin = D4;

void setup() {
  Serial.begin(115200);
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, HIGH); // Initially LED is OFF
  setupWiFi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback); // Handle incoming messages
}

void setupWiFi() {
  delay(10);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi Connected");
}
```

```
void reconnect() {
  while (!client.connected()) {
    if (client.connect("p.iot.0.569")) {  // Use a random unique device id.
      Serial.println("MQTT Connected");
        client.subscribe(topic);    // Additional Line to subscribe topic
    } else {
      Serial.println("Trying to connect MQTT ... ");
      delay(5000);
    }
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  //Print received message on Serial Console
  Serial.print("Got Message: ");
  Serial.print(topic);
  Serial.print(" > ");
  Serial.println(message);

  //Set LED state based on the received Message
  if (message == "ON") {
    digitalWrite(ledPin, LOW);
  } else if (message == "OFF") {
    digitalWrite(ledPin, HIGH);
  }
}

void loop() {
  if (!client.connected()) reconnect();
  client.loop();

  bool buttonState = digitalRead(buttonPin);
  if (buttonState != lastButtonState) {
    if (buttonState == HIGH) {
      Serial.println("Button is ON");
      client.publish(topic, "ON");
    } else {
      Serial.println("Button is OFF");
      client.publish(topic, "OFF");
    }
    lastButtonState = buttonState;
  }
  delay(50);
}
```

Advanced Tasks (Optional for Practice)

Add a Sensor:

- Modify the publisher to read a DHT11 sensor and publish temperature to topic: `"pieas/iotlab/4/group/[group_id]/temperature"`.

Two-Way Communication:

- Make both devices publish and subscribe (e.g., button press triggers LED on both devices).

Key Concepts to Discuss

MQTT Topics:

- Hierarchical structure (e.g., sensors/+/temperature for wildcard subscriptions).

QoS Levels:

- Experiment with QoS 1 (at least once delivery).

Retained Messages:

- Use retain=true to send the last known value to new subscribers.

Submission Requirements:

Submit a single PDF containing:

- **Your Arduino Code:** Submit Arduino sketches (.ino) for all cases.
- **Screenshot/Video:** Show the Serial Monitor with published and received MQTT messages.
- **Circuit Diagram:** Provide clear schematic diagrams (or photos) of your breadboard setup.
- **Short Description:**
  - The steps you followed.
  - How you verified the functionality.
  - Any challenges you encountered and how you addressed them.

Conclusion:

- This lab demonstrates fundamental actuator control using a WeMos D1 Mini.
- Students gain hands-on experience with relays for switching and PWM for LED dimming.
- The optional potentiometer integration adds an extra layer of interactivity, showcasing the dynamic nature of IoT applications.

Grading Rubric:

Criterion	Points	Description
WiFi & MQTT Connection	30	Successful connection to WiFi and MQTT broker, with proper code implementation.
Publishing Functionality	25	Regular publishing of messages to the designated topic.
Subscription Handling	25	Proper reception and handling of messages from subscribed topics.
Documentation & Report	20	Clear, concise documentation of the lab procedure, code explanation, and troubleshooting steps.
Total	100	