Aditya Vishe BT22CIV091

Experiment No. 10

Transfer of information to and from an ESP32 using MQTT protocol.

Aim

To establish communication between an ESP32 and an MQTT broker to transfer information (data) to and from the ESP32 using the MQTT protocol.

Apparatus

- ESP32 microcontroller
- LM35 temperature sensor
- WiFi network
- Jumper wires
- Breadboard (optional)
- Computer with Arduino IDE for programming

Theory

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol for low-bandwidth and high-latency networks. It uses a publish-subscribe model, enabling efficient communication between devices.

Key components of MQTT:

Broker: Central server managing communication (e.g., Mosquitto).
Publisher: Device sending data to the broker.
Subscriber: Device receiving data from the broker.
Topics: Channels for data transfer.
In this experiment:

The ESP32 acts as a publisher to send data (e.g., sensor readings) and as a subscriber to receive commands (e.g., LED control).
The broker facilitates communication between devices.

Procedure

1. Set Up the Environment
   Install Arduino IDE.
   Add ESP32 board support to the IDE.
   Install the PubSubClient library.
2. Configure Wi-Fi and MQTT
   Update the ssid and password variables with your Wi-Fi credentials.
   Set the broker details (broker: broker.emqx.io, topic: emqx/esp32). 3. Upload the Code
   Connect the ESP32 to your computer.
   In the Arduino IDE, select the appropriate board and port.
   Upload the code to the ESP32.
4. Monitor Serial Output
   Open the Serial Monitor in Arduino IDE with a baud rate of 115200.
   Observe the connection status for Wi-Fi and MQTT, published messages, and updates to subscribe topics
5. Test MQTT Communication
   Use an MQTT client (e.g., MQTT Explorer) to publish messages to the topic emqx/esp32. Check the ESP32's response on the Serial Monitor.
6. Debugging
   Verify Wi-Fi and MQTT credentials.
   Check network connectivity.
   Ensure the MQTT broker is accessible if any issues occur.

CODE-

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

// WiFi
const char *ssid = "Emb_Lab"; // Enter your
Wi-Fi name const char *password =
"emb@1234";  // Enter Wi-Fi password

// MQTT Broker
const char *mqtt_broker =
"broker.emqx.io"; const
char *topic =
"emqx/esp32"; const char
*mqtt_username = "emqx";
const char *mqtt_password
= "public"; const int
mqtt_port = 1883;
```

```
WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
   // Set software serial baud to 115200;
   Serial.begin(115200);
   // Connecting to a WiFi network
WiFi.begin(ssid, password);
   while (WiFi.status() !=
WL_CONNECTED) {        delay(500);
      Serial.println("Connecting to WiFi..");
   }
   Serial.println("Connected to the Wi-Fi network");
   //connecting to a mqtt broker
client.setServer(mqtt_broker, mqtt_port);

client.setCallback(callback
);    while
(!client.connected()) {
String client_id = "esp32-
client-";        client_id +=
String(WiFi.macAddress())
;
      Serial.printf("The client %s connects to the public MQTT broker\n",
client_id.c_str());        if (client.connect(client_id.c_str(), mqtt_username,
mqtt_password)) {
         Serial.println("Public EMQX MQTT broker connected");
      } else {
         Serial.print("failed
with state ");
Serial.print(client.state());
delay(2000);
      }
   }
   // Publish and subscribe
   client.publish(topic, "Hi, I'm ESP32 ^^");
client.subscribe(topic);
}

void callback(char *topic, byte *payload, unsigned int length) {
   Serial.print("Message arrived in topic: ");

Serial.println(topi
c);
```

```
Serial.print("Mes
sage:");     for (int
i = 0; i < length;
i++) {
Serial.print((char)
payload[i]);
    }
    Serial.println();
    Serial.println("-----------------------");
}

void
loop() {
client.lo
op();
}
```

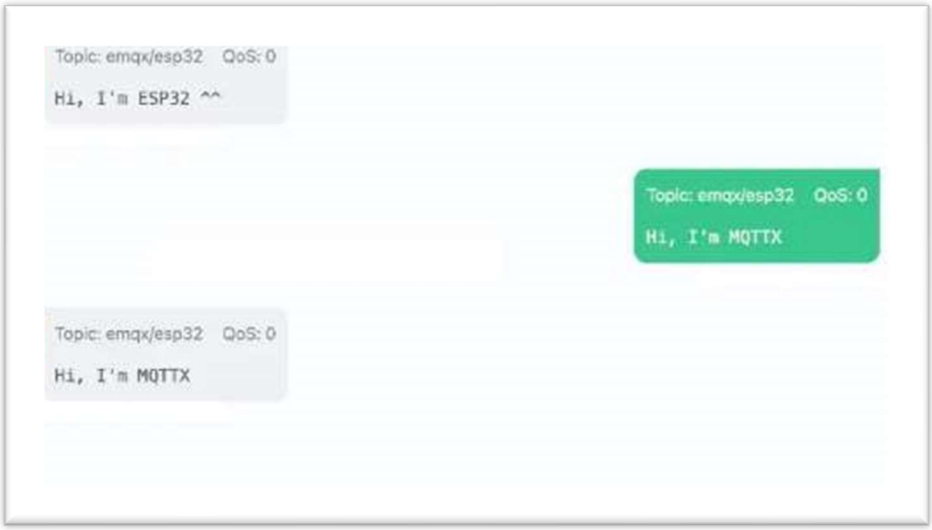
Observations

The experiment successfully demonstrated reliable communication between the ESP32 and the MQTT broker. The ESP32 connected to the Wi-Fi network and broker, publishing sensor data (e.g., temperature) to a specific topic at regular intervals. The data was promptly displayed on the subscribed MQTT client, validating the publish functionality. Commands sent to the ESP32 via a subscribed topic were received and executed correctly, such as toggling an LED.
The communication was low-latency, ensuring real-time updates and immediate responses to commands. Despite occasional delays due to Wi-Fi instability, the system recovered seamlessly from brief network interruptions. Data integrity was maintained throughout the process, and the ESP32 managed resources efficiently, confirming its suitability for MQTT-based IoT applications.

Conclusion

The experiment highlighted the efficiency of the MQTT protocol in enabling real-time, bidirectional communication. The ESP32 successfully published sensor data and subscribed to commands, showcasing its potential for IoT applications. MQTT's scalability and robustness were evident, with the system resuming communication after minor disruptions.
Future improvements could include implementing encryption for secure communication and using advanced MQTT features like retained messages and QoS for better reliability. The experiment reinforced MQTT's suitability for applications like smart home automation and industrial monitoring, requiring efficient and reliable communication.

Result:


Topic: emqx/esp32 QoS: 0
Hi, I'm ESP32 ^^
Topic: emqx/esp32 QoS: 0
Hi, I'm MQTTX
Topic: emqx/esp32 QoS: 0
Hi, I'm MQTTX