# MQTT Protocol Implementation Using ESP32 and HiveMQ Cloud

IIT Ropar Minor in AI

May 29, 2025

## Contents

# 1 Introduction to MQTT Protocol

## 1.1 Overview and Historical Background

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol developed by IBM in 1999, specifically designed for transmitting sensor data over low bandwidth and unreliable networks. The protocol is approximately eight times lighter than HTTP, making it ideal for IoT applications where efficiency and resource conservation are paramount.

## 1.2 Key Characteristics

- Lightweight design optimized for low-power, low-speed internet connections
- Widely adopted in popular applications including WhatsApp, Amazon Alexa, and Uber
- Utilizes a Publish-Subscribe model rather than the traditional client-server approach
- Enhanced security through broker-mediated communication
- Efficient operation on unreliable networks with high latency

# 2 MQTT Architecture and Components

## 2.1 Core Components

The MQTT architecture consists of four fundamental components that work together to enable efficient message transmission:

### 2.1.1 Publisher

The publisher is an entity responsible for sending data to the network. In IoT contexts, this typically represents sensors or other data sources that generate information for transmission.

### 2.1.2 Subscriber

The subscriber is an entity that receives and processes data from the network. These components consume the information published by publishers and typically perform actions based on received data.

### 2.1.3 Broker

The broker serves as a middleman server that manages message flow between publishers and subscribers. It handles message routing, storage, and delivery according to specified quality of service requirements.

### 2.1.4 Topic

Topics represent the subject or channel through which publishers send messages and subscribers listen for information. Topics follow a hierarchical structure that enables organized message routing.

### 2.2 Architectural Analogy

To better understand MQTT architecture, consider the following business analogy:

- **Seller** = Publisher (data source)
- **Buyer** = Subscriber (data consumer)
- **Agent** = Broker (message facilitator)

## 3 MQTT Features and Quality of Service

### 3.1 Network Reliability Features

MQTT is specifically designed to work effectively with unreliable networks and high latency conditions. The protocol includes several features that ensure robust communication in challenging network environments.

### 3.2 Quality of Service Levels

MQTT implements three distinct Quality of Service (QoS) levels to provide flexibility in message delivery guarantees:

#### 3.2.1 QoS 0: At Most Once

This level provides best-effort delivery with no guarantee of message arrival. Messages are delivered at most once, with the possibility of message loss but no duplication.

#### 3.2.2 QoS 1: At Least Once

This level guarantees message delivery but may result in message duplication. The system ensures that messages arrive at least once at their destination.

#### 3.2.3 QoS 2: Exactly Once

This level provides the highest guarantee, ensuring that messages are delivered exactly once without loss or duplication.

### 3.3 Additional Features

- Always-connected architecture with configurable keep-alive intervals
- Security support through username/password authentication
- SSL/TLS encryption capabilities for secure communication
- Efficient bandwidth utilization for resource-constrained devices

## 4 Hardware and Software Setup

### 4.1 Hardware Components

The practical implementation utilizes the following hardware components:

#### 4.1.1 Microcontroller

ESP32 microcontroller implemented in Wokwi simulator environment for development and testing purposes.

#### 4.1.2 Sensors

DHT22 sensor for measuring temperature and humidity with high accuracy and reliability.

#### 4.1.3 Output Devices

- LED for visual indication
- Buzzer for audio alerts
- Relay module for controlling external devices

### 4.2 Software Requirements

The implementation requires several essential libraries:

#### 4.2.1 Required Libraries

- `WiFi.h` - Enables internet connectivity for ESP32
- `PubSubClient.h` - Provides MQTT communication capabilities
- `DHT sensor library` - Facilitates DHT22 sensor data reading

#### 4.2.2 MQTT Broker Selection

HiveMQ Cloud serves as the primary broker for demonstration purposes. Alternative brokers include Thingspeak and Mosquitto, each offering different features and capabilities.

## 5 MQTT Publisher Implementation

### 5.1 Hardware Configuration

The publisher implementation involves connecting an LED to the ESP32 microcontroller with the anode connected to pin 2 and the cathode connected to ground.

### 5.2 Publisher Code Structure

The publisher code implements the following functionality:

```
1
2  #include <WiFi.h>
3  #include "PubSubClient.h"
4  const char* ssid = "Wokwi-GUEST";
5  const char* password = "";
6  const char* mqttServer = "broker.hivemq.com";
7  int port = 1883;
8  String stMac;
9  char mac[50];
10 char clientId[50];
11 WiFiClient espClient;
12 PubSubClient client(espClient);
13 const int ledPin = 2;
14 void setup() {
```

```
Serial.begin(115200);
randomSeed(analogRead(0));
delay(10);
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);
wifiConnect();
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
Serial.println(WiFi.macAddress());
stMac = WiFi.macAddress();
stMac.replace(":", "_");
Serial.println(stMac);
client.setServer(mqttServer, port);
client.setCallback(callback);
pinMode(ledPin, OUTPUT);
}
void wifiConnect() {
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
}
void mqttReconnect() {
while (!client.connected()) {
Serial.print("Attempting MQTT connection...");
long r = random(1000);
sprintf(clientId, "clientId-%ld", r);
if (client.connect(clientId)) {
Serial.print(clientId);
Serial.println(" connected");
client.subscribe("topicName/led");
} else {
Serial.print("failed, rc=");
Serial.print(client.state());
Serial.println(" try again in 5 seconds");
delay(5000);
}
}
}
void callback(char* topic, byte* message, unsigned int length) {
Serial.print("Message arrived on topic: ");
Serial.print(topic);
Serial.print(". Message: ");
String stMessage;
for (int i = 0; i < length; i++) {
Serial.print((char)message[i]);
stMessage += (char)message[i];
}
Serial.println();
if (String(topic) == "topicName/led") {
Serial.print("Changing output to ");
if(stMessage == "on"){
Serial.println("on");
digitalWrite(ledPin, HIGH);
}
else if(stMessage == "off"){
Serial.println("off");
digitalWrite(ledPin, LOW);
```

```
}
}
}
void loop() {
delay(10);
if (!client.connected()) {
 mqttReconnect();
}
client.loop();
}
```

Listing 1: MQTT Publisher Code Structure

### 5.3 Key Implementation Features

- Wi-Fi connection establishment using predefined SSID and password
- Connection to HiveMQ broker using broker URL and port 1883
- Random MQTT client ID generation for connection uniqueness
- Message publishing to specific topics for device control
- Robust reconnection logic for handling connection failures

## 6 MQTT Subscriber Implementation

### 6.1 Sensor Integration

The subscriber implementation incorporates DHT22 sensor connectivity with the same ESP32 board used in the publisher configuration.

### 6.2 Subscriber Code Structure

The subscriber functionality includes comprehensive sensor data handling and device control capabilities:

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

#define DHTPIN 15        // DHT22 data pin connected to GPIO 15 #define DHTTYPE
    DHT22    // DHT 22 (AM2302)
DHT dht(DHTPIN, DHTTYPE);
 const char* ssid = "Wokwi-GUEST"; const char* password = "";

const char* mqttServer = "broker.hivemq.com";  // //host:
broker.hivemq.com     //port:8884 //  client name :testClient123 //Publish topic
    :topicName/led int port = 1883;                                  // port

String stMac; char mac[50]; char clientId[50];

WiFiClient espClient;
PubSubClient client(espClient);
 const int ledPin = 2;

void setup() {   Serial.begin(115200);   dht.begin();  // Initialize DHT sensor
      randomSeed(analogRead(0));   delay(10);
  Serial.println();
  Serial.print("Connecting to ");   Serial.println(ssid);   wifiConnect();
  Serial.println("");
```

```
22   Serial.println("WiFi connected");   Serial.println("IP address: ");
23   Serial.println(WiFi.localIP());   Serial.println(WiFi.macAddress());   stMac =
       WiFi.macAddress();   stMac.replace(":", "_");   Serial.println(stMac);
     client.setServer(mqttServer, port);   client.setCallback(callback);
     pinMode(ledPin, OUTPUT);
24 }
25 void wifiConnect() {
26   WiFi.mode(WIFI_STA);   WiFi.begin(ssid, password);   while (WiFi.status() !=
     WL_CONNECTED) {     delay(500);
27     Serial.print(".");
28   }
29 }  void mqttReconnect() {   while (!client.connected()) {
30     Serial.print("Attempting MQTT connection...");
31     long r = random(1000);     sprintf(clientId, "clientId-%ld", r);     if (
     client.connect(clientId)) {       Serial.print(clientId);       Serial.
     println(" connected");       client.subscribe("topicName/led");  //
     Subscribing to control LED
32     } else {
33       Serial.print("failed, rc=");
34       Serial.print(client.state());
35       Serial.println(" try again in 5 seconds");       delay(5000);
36     }
37   }
38 }
39 void callback(char* topic, byte* message, unsigned int length) {
40   Serial.print("Message arrived on topic: ");
41   Serial.print(topic);
42   Serial.print(". Message: ");   String stMessage;
43   for (int i = 0; i < length; i++) {     Serial.print((char)message[i]);
       stMessage += (char)message[i];
44   }
45   Serial.println();
46   if (String(topic) == "topicName/led") {     Serial.print("Changing output to "
     );     if (stMessage == "on") {       Serial.println("on");
     digitalWrite(ledPin, HIGH);
47     } else if (stMessage == "off") {       Serial.println("off");
     digitalWrite(ledPin, LOW);
48     }
49   }
50 }  void loop() {   if (!client.connected()) {     mqttReconnect();
51   }
52   client.loop();
53
54   // Read and publish DHT22 data every 5 seconds   static unsigned long lastTime
       = 0;   if (millis() - lastTime > 5000) {     float temp = dht.
     readTemperature();     float hum = dht.readHumidity();
55
56     if (isnan(temp) || isnan(hum)) {
57       Serial.println("Failed to read from DHT sensor!");
58     } else {       char payload[60];
59       sprintf(payload, "Temperature: %.2f C, Humidity: %.2f %%", temp, hum);
         Serial.println(payload);       client.publish("topicName/sensor", payload
     );  // Publishing to HiveMQ     }      lastTime = millis();
60   }
61 }
```

Listing 2: MQTT Subscriber Code Structure

### 6.3 Subscriber Functionality

- Continuous listening to sensor data topics
- Temperature and humidity reading every 5 seconds

- Data validation to ensure sensor reading accuracy
- Publishing sensor data to HiveMQ broker
- Receiving and processing commands for peripheral control

# 7 HiveMQ Cloud Integration

## 7.1 Platform Features

HiveMQ Cloud provides a comprehensive web interface and WebSocket clients for MQTT communication, enabling multiple simultaneous user connections and real-time message exchange.

## 7.2 Security Implementation

- Username and password authentication mechanisms
- SSL/TLS encryption for secure data transmission
- Port configuration options (1883 for standard, 8883/8884 for secure connections)

## 7.3 Multi-Device Control

The platform demonstrates the capability to control multiple devices simultaneously through shared topics, enabling coordinated operation of LEDs, buzzers, and relay modules.

# 8 Practical Assignments

## 8.1 RGB LED Control Implementation

Design and implement an RGB LED control system using MQTT with three separate topics:

- `LED/Red` - Controls red color intensity
- `LED/Green` - Controls green color intensity
- `LED/Blue` - Controls blue color intensity

## 8.2 Alternative Sensor Integration

Expand the system capabilities by incorporating additional sensors:

- MQ gas sensor for air quality monitoring
- Ultrasonic sensor for distance measurement
- MPU6050 for motion and orientation detection

## 8.3 Dynamic LED Control

Implement dynamic LED blinking speed control based on received MQTT commands, allowing real-time adjustment of visual indicators.

# 9 Important Considerations

## 9.1 Protocol Advantages

MQTT demonstrates exceptional suitability for IoT applications due to its low bandwidth requirements and reliable message delivery mechanisms. The protocol's efficiency makes it ideal for resource-constrained environments.

## 9.2 Topic Management

MQTT topics follow a hierarchical structure and require consistent configuration between publisher and subscriber implementations to ensure proper message routing.

## 9.3 Network Dependencies

Simulation performance depends on internet stability, and users may experience delays due to network conditions or server load. Understanding these limitations is crucial for real-world implementation planning.

## 9.4 Quality of Service Selection

The choice of appropriate QoS levels provides flexibility in balancing delivery guarantees against network overhead, allowing optimization for specific application requirements.

# 10 Conclusion

This comprehensive overview covers the fundamental concepts, practical implementation steps, and important protocol features of MQTT implementation using ESP32 microcontrollers and HiveMQ Cloud services. The combination of theoretical understanding and hands-on coding experience provides a solid foundation for developing robust IoT communication systems.

The practical assignments offer opportunities to extend the basic concepts and explore advanced implementations, encouraging deeper understanding of MQTT protocol capabilities and IoT system design principles.