Module 7 - MQTT, HTTP, WIFI

7.1 Introduction: The IoT Communication Stack

For an IoT device to be useful, it needs to communicate. This communication happens in layers, much like a conversation. You need to have connectivity, then you need a common language to request things (web communication), and sometimes a specialized shorthand (messaging).

- **Connectivity (Wi-Fi)**: Wi-Fi allows your ESP32 to connect to a local network, giving it a path to the internet or other local devices. This module will cover how to configure the ESP32 as both a client that connects to a router and as an access point that creates its own network.
- **Web Communication (HTTP/HTTPS)**: HTTP allows your device to request data from and send data to web servers and APIs (Application Programming Interfaces). This is essential for tasks like fetching weather data, logging information to a cloud database, or being controlled by a web dashboard. HTTPS is the secure version, encrypting the conversation to protect your data.
- **Messaging (MQTT/MQTTS)**: While HTTP is powerful, it can be inefficient for the constant, small data packets typical of IoT sensors. MQTT is a lightweight, efficient protocol designed specifically for this purpose. It uses a publish/subscribe model, which is perfect for real-time telemetry and control. MQTTS is its secure counterpart.

7.2 Local Network Connectivity with Wi-Fi

Wi-Fi is a technology based on the **IEEE 802.11 standards** that enables wireless data exchange. The ESP32 supports common standards like 802.11b, 802.11g, and 802.11n, operating in the 2.4 GHz frequency band.

ESP32 Wi-Fi Modes

1. **Station Mode (STA)**: The ESP32 acts as a client, connecting to an existing access point (AP) like your home or university router. This is the most common mode for devices that need internet access.
2. **Access Point Mode (AP)**: The ESP32 creates its own Wi-Fi network, allowing other devices (like your phone or laptop) to connect directly to it. This is useful for initial device configuration or creating isolated local networks.
3. **STA + AP Mode**: The ESP32 can do both simultaneously, connecting to one network while also providing its own. This allows it to act as a range extender or a bridge between networks.

Example

```
#include <WiFi.h>

// --- Replace with your network credentials ---
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
// ---------------------------------------

void setup() {
  Serial.begin(115200); // Allow serial to initialize

  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  // Start the Wi-Fi connection
  WiFi.begin(ssid, password);

  // Wait for the connection to complete
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected successfully!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  // The main work is done in setup for this example.
  // In a real application, you would do your networking tasks here.
  delay(10000);
}
```

7.3 Web Communication with HTTP/HTTPS

HTTP

HTTP (Hypertext Transfer Protocol) is the foundation of data communication on the World Wide Web. It operates on a request-response model.

- Client (ESP32): The client make a request to the server for a resource, like a webpage or data.
- Server (A Web Server): The server takes your request, process it, and send the processed request back to you as a response.

HTTPS

HTTPS is simply HTTP Secure. It adds a layer of SSL/TLS encryption to the conversation. This prevents anyone from eavesdropping on the data exchanged between the client and server, which is crucial for protecting sensitive information like passwords or personal data.

Example

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>

// --- Replace with your network credentials ---
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
// ---------------------------------------

// The URL of the API we want to request data from
const char* api_url = "http://jsonplaceholder.typicode.com/posts/1";

void setup() {
  Serial.begin(115200);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  // Check if we are connected to WiFi before proceeding
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    Serial.println("Making HTTP GET request...");
    http.begin(api_url); // Initialize the HTTP request

    int httpCode = http.GET(); // Send the GET request

    if (httpCode > 0) { // Check if the request was successful
      Serial.printf("HTTP Response code: %d\n", httpCode);

      if (httpCode == HTTP_CODE_OK) {
        String payload = http.getString(); // Get the response payload as a string

        // --- Parse the JSON response ---
        JsonDocument doc;
        DeserializationError error = deserializeJson(doc, payload);

        if (error) {
          Serial.print("deserializeJson() failed: ");
          Serial.println(error.c_str());
        }
        else {
          // Extract the value associated with the "id" and "title" key

          int postId = doc["id"];
          const char* title = doc["title"];

          Serial.printf("Post ID: %d\n", postId);
          Serial.printf("Title: %s\n", title);
```

```
        }
      }
    } else {
      Serial.printf("HTTP GET request failed, error: %s\n",
http.errorToString(httpCode).c_str());
    }

    http.end(); // Free resources
  } else {
    Serial.println("WiFi not connected.");
  }

  // Wait 30 seconds before the next request
  delay(30000);
}
```

7.4 Efficient IoT Messaging with MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for constrained devices and unreliable networks, making it perfect for IoT. Instead of the request-response model, MQTT uses a publish/subscribe (pub/sub) model.

MQTT Components

- **Broker**: A central server that acts as a intermediary, receiving messages from publishers and distributing them to interested subscribers
- **Client (Publisher)**: A device (like an ESP32 with a sensor) that publishes messages (e.g., temperature readings) to a specific "topic" on the broker. It doesn't know or care who reads the message.
- **Client (Subscriber)**: Another device or application (like a mobile app or a server) that subscribes to that same topic. The broker automatically forwards any message published to that topic to all subscribers.

MQTT Concepts

- **Topic**: A hierarchical string that acts as a channel for messages (e.g., home/livingroom/lamp).
- **Quality of Service (QoS)**:
  - **QoS 0 (At most once)**: The message is sent once ("fire and forget"). It's fast but offers no delivery confirmation.
  - **QoS 1 (At least once)**: The message is guaranteed to be delivered, but it might arrive more than once.
  - **QoS 2 (Exactly once)**: The most reliable level, guaranteeing the message is delivered exactly one time. It is the slowest due to a more complex handshake.

MQTTS

Just like HTTPS, MQTTS is MQTT secured with TLS encryption to protect data confidentiality and integrity.

Example

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

// --- Replace with your network credentials ---
const char* ssid = "YOUR_WIFI_SSID";
```

```
const char* password = "YOUR_WIFI_PASSWORD";
// ---------------------------------------

// --- MQTT Broker Configuration ---
const char* mqtt_server = "broker.hivemq.com";
const int mqtt_port = 1883;

// --- Topics ---
// Topic to publish messages to
const char* publish_topic = "Digilab/Modul7/status";
// Topic to subscribe to for incoming messages
const char* subscribe_topic = "Digilab/Modul7/command";

WiFiClient wifiClient;
PubSubClient mqttClient(wifiClient);

// This function is called whenever a message arrives on a subscribed topic
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String message;
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }
  Serial.println(message);

  // Example: Turn on a built-in LED if the message is "ON"
  if (message == "ON") {
    Serial.println("Turning on lamp");
    digitalWrite(LED_BUILTIN, HIGH);
  } else if (message == "OFF") {
    Serial.println("Turning off lamp");
    digitalWrite(LED_BUILTIN, LOW);
  }
}

void reconnect() {
  // Loop until we're reconnected
  while (!mqttClient.connected()) {
    Serial.print("Attempting MQTT connection...");
```

```
    // Create a random client ID
    String clientId = "ESP32Client-";
    clientId += String(random(0xffff), HEX);

    // Attempt to connect
    if (mqttClient.connect(clientId.c_str())) {
      Serial.println("connected");
      // Subscribe to the command topic upon connection
      mqttClient.subscribe(subscribe_topic);
    } else {
      Serial.print("failed, rc=");
      Serial.print(mqttClient.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}


void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(115200);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected!");

  // Configure MQTT client
  mqttClient.setServer(mqtt_server, mqtt_port);
  mqttClient.setCallback(callback);
}

void loop() {
  // Ensure the MQTT client is connected
  if (!mqttClient.connected()) {
    reconnect();
```

```
  }
  mqttClient.loop(); // This allows the client to process incoming messages

  // --- Publish a message every 10 seconds ---
  static unsigned long lastMsg = 0;
  unsigned long now = millis();
  if (now - lastMsg > 10000) {
    lastMsg = now;

    char msg[50];
    snprintf(msg, 50, "Uptime: %lu seconds", millis() / 1000);

    Serial.printf("Publishing message to %s: %s\n", publish_topic, msg);
    mqttClient.publish(publish_topic, msg);
  }
}
```