Detailed Report on ESP32 Over-the-Air (OTA) Programming

Introduction

Over-the-air (OTA) programming is a powerful feature that allows you to update the firmware of an ESP32 device wirelessly over a Wi-Fi network. This eliminates the need for a physical connection (e.g., USB) to the device, making it ideal for remote or hard-to-access installations. OTA is particularly useful in IoT applications, where devices are deployed in large numbers and need to be updated frequently.

This report provides a detailed explanation of how to implement OTA programming on an ESP32, specifically for an air quality monitoring system that uses DHT22 and MQ135 sensors. The report covers the following topics:

1. **Overview of OTA Programming**
2. **Types of OTA Updates**
3. **Implementation Steps**
4. **Code Explanation**
5. **Verification and Troubleshooting**
6. **Best Practices**

1. Overview of OTA Programming

What is OTA Programming?

OTA programming allows you to upload new firmware to an ESP32 device over a Wi-Fi network. This is particularly useful when the device is deployed in a remote location or when you need to update multiple devices simultaneously.

Key Advantages of OTA

- **Remote Updates**: No need for physical access to the device.
- **Scalability**: Update multiple devices on the same network from a central location.
- **Convenience**: Easily push updates without disrupting the device's operation.

Key Considerations

- **Initial Setup**: The first firmware upload must be done via a serial connection (USB) to enable OTA functionality.
- **OTA Routine**: Every subsequent firmware update must include the OTA code to allow future updates.
- **Network Stability**: OTA updates require a stable Wi-Fi connection to avoid corruption during the update process.

2. Types of OTA Updates

There are two main ways to implement OTA updates on the ESP32:

1. **Basic OTA (Arduino IDE)**

- Updates are delivered using the Arduino IDE.
- The ESP32 must be connected to the same Wi-Fi network as the development PC.
- Ideal for small-scale projects or during development.

2. Web Updater OTA

- Updates are delivered via a web browser.
- The ESP32 runs a web server that allows you to upload new firmware through a web interface.
- Suitable for larger deployments or when you want to provide a user-friendly update mechanism.

This report focuses on **Basic OTA** using the Arduino IDE.

3. Implementation Steps

Step 1: Install Required Libraries

To enable OTA functionality, you need the `***ArduinoOTA***` library, which is included with the ESP32 board package in the Arduino IDE. Ensure that you have the ESP32 board package installed. Or else install the ArduinoOTA library.

Step 2: Modify Your Code to Support OTA

Integrate OTA functionality into your existing code. This involves:

- Including the `**ArduinoOTA**` library.
- Setting up OTA event handlers (e.g., start, progress, end, error).
- Initializing OTA in the `**setup()**` function.
- Handling OTA updates in the `**loop()**` function.

Step 3: Upload the Initial Firmware via USB

The first (The **BASIC_OTA.ino** code) firmware upload must be done via a serial connection (USB) to enable OTA functionality. This initial sketch must include the OTA code.

Step 4: Upload New Firmware Wirelessly

Once OTA is enabled, you can upload new firmware wirelessly using the Arduino IDE. The ESP32 will appear as a network port in the Arduino IDE.

4. Code Explanation

Key Components of the Code

1. **Include Required Libraries**

```
#include <WiFi.h>
#include <FirebaseESP32.h>
#include "addons/TokenHelper.h"
#include "addons/RTDBHelper.h"
#include <DHT.h>
#include <time.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
```

   - **WiFi.h**: Enables Wi-Fi connectivity.
   - **FirebaseESP32.h**: Facilitates communication with Firebase.
   - **DHT.h**: Interfaces with the DHT22 temperature and humidity sensor.
   - **ArduinoOTA.h**: Enables OTA functionality.

2. **Define Sensor Pins and Variables**

```
#define DHTPIN 26       // GPIO pin connected to DHT22
#define DHTTYPE DHT22   // DHT22 sensor type
DHT dht(DHTPIN, DHTTYPE);  // Initialize DHT sensor
float temperature = 0;  // Variable to store
temperature
float humidity = 0;   // Variable to store humidity
```

   - **DHTPIN**: GPIO pin connected to the DHT22 sensor.
   - **DHTTYPE**: Specifies the type of DHT sensor (DHT22 in this case).

3. **Wi-Fi Configuration**

```
#define WIFI_SSID "HUAWEI-2.4G"       // Wi-Fi SSID
```

```
#define  WIFI_PASSWORD  "malikza01"           //  Wi-Fi
password

void Wifi_Init() {
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    Serial.print("Connecting to Wi-Fi");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(300);
    }
    Serial.println();
    Serial.print("Connected with IP: ");
    Serial.println(WiFi.localIP());
}
```

- Connects the ESP32 to the specified Wi-Fi network.
- Prints the IP address once connected.

4. OTA Setup

```
void setupOTA() {

ArduinoOTA.setHostname("esp32-air-quality-monitor");
// Set a hostname for OTA

    // Define OTA event handlers
    ArduinoOTA.onStart([]() {
        Serial.println("OTA Update Starting...");
    });

    ArduinoOTA.onEnd([]() {
        Serial.println("\nOTA Update Complete!");
    });

    ArduinoOTA.onProgress([](unsigned int progress,
unsigned int total) {
        Serial.printf("Progress: %u%%\r", (progress /
(total / 100)));
    });

    ArduinoOTA.onError([](ota_error_t error) {
        Serial.printf("Error[%u]: ", error);
        if (error == OTA_AUTH_ERROR)
Serial.println("Auth Failed");
```

```
            else if (error == OTA_BEGIN_ERROR)
    Serial.println("Begin Failed");
            else if (error == OTA_CONNECT_ERROR)
    Serial.println("Connect Failed");
            else if (error == OTA_RECEIVE_ERROR)
    Serial.println("Receive Failed");
            else if (error == OTA_END_ERROR)
    Serial.println("End Failed");
        });

        ArduinoOTA.begin();  // Start OTA service
        Serial.println("OTA Initialized");
    }
```

- Configures OTA with event handlers for start, progress, end, and error.
- Starts the OTA service.

5. **Read Sensor Data**

```
    void updateSensorReadings() {
        temperature = dht.readTemperature();
        humidity = dht.readHumidity();
        if (isnan(temperature) || isnan(humidity)) {
            Serial.println("Failed to read from DHT
    sensor!");
            return;
        }

        Serial.printf("Temperature: %.2f°C\n",
    temperature);
        Serial.printf("Humidity: %.2f%%\n", humidity);
    }
```

- Reads temperature and humidity from the DHT22 sensor.
- Validates the readings and prints them to the Serial Monitor.

6. **Sensor Data Upload**

```
void uploadSensorData() {
      if (millis() - elapsedMillis > update_interval &&
isAuthenticated && Firebase.ready()) {
        elapsedMillis = millis();
        updateSensorReadings();
        FirebaseJson sensorData;
        sensorData.set("temperature", temperature);
```

```
        sensorData.set("humidity", humidity);
                                    sensorData.set("timestamp",
getFormattedTimestamp());
                  Firebase.setJSON(fbdo,  nodePath.c_str(),
sensorData);
    }
}
```

- Reads sensor data and uploads it to Firebase

7. **Main Setup and Loop**

```
void setup() {
    Serial.begin(115200);
    Wifi_Init();
    firebase_init();
    dht.begin();
    pinMode(MQ135_PIN, INPUT);
    setupOTA();
}

void loop() {
    ArduinoOTA.handle();
    uploadSensorData();
    delay(1000);
}
```

- Initializes Wi-Fi, Firebase, sensors, and OTA.
- Handles OTA updates and uploads sensor data in the loop.

5. Verification and Troubleshooting

Verifying OTA Functionality

- **Initial Upload**: Upload the code via USB to enable OTA.
- **Wireless Upload**: Modify the code (e.g., add new sensor functionality) and upload it wirelessly using the Arduino IDE.
- **Check Serial Monitor**: Verify that the OTA update process is logged in the Serial Monitor.

Troubleshooting

- **ESP32 Port Not Visible**: Ensure the ESP32 and your PC are on the same Wi-Fi network. Use the `ping` command to check connectivity.
- **Request Timeout**: Disable Windows Defender, firewall, or antivirus software that may block the connection.
- **OTA Update Fails**: Ensure the ESP32 has a stable Wi-Fi connection and sufficient power.

6. Best Practices

- **Stable Wi-Fi Connection**: Ensure the ESP32 is connected to a stable Wi-Fi network during OTA updates
- **Avoid Delays in Loop**: Minimize or avoid using `delay()` in the `loop()` function to prevent OTA timeouts.
- **Error Handling**: Implement robust error handling for OTA and sensor operations.
- **Security**: Use OTA passwords to secure updates and prevent unauthorized access.
- **Backup Plan**: Always have a backup plan (e.g., serial upload) in case OTA fails.

Conclusion

OTA programming is a powerful tool for updating ESP32 devices remotely. By following the steps outlined in this report, you can successfully implement OTA functionality in your ESP32-based air quality monitoring system. This approach not only simplifies the update process but also enhances the scalability and maintainability of your IoT deployments.