Wireless Client Server Model with Two ESP8266 Boards on Wi-Fi

Akshaj Rai
Dept. of Electronics and Communication Engineering
Nirma University, Ahmedabad, India
20bec099@nirmauni.ac.in

Sanchit Sharma
Dept. of Electronics and Communication Engineering
Nirma University, Ahmedabad, India
20bec108@nirmauni.ac.in

***Abstract*—The Internet of Things (IoT) has revolutionized the way devices communicate and share data wirelessly. This research paper explores the practical implementation of wireless connectivity using ESP8266 microcontrollers. Two ESP8266 boards are configured, one as an access point (AP) and the other in station mode, to establish a Wi-Fi connection. The AP collects data from a BMP280 sensor, while the station displays this data on an OLED screen. The project highlights the seamless exchange of real-time data facilitated by HTTP GET requests and responses, emphasizing the accessibility of ESP8266 technology. The research showcases the transformative potential of ESP8266 in the context of IoT and computer networks, making it a fundamental component in reshaping wireless communication in the digital age.**

***Index Terms*—Introduction, System Architecture Diagram, Methodology, Working, Results, Conclusion, Future Scope, Acknowledgement**

I. Introduction

The Internet of Things (IoT) has revolutionized the way devices communicate and share data wirelessly, forging the path for a smarter, more interconnected world. At the heart of this transformative landscape lies the ESP8266 microcontroller, a compact yet potent hardware platform. Developed by Espressif Systems, the ESP8266 has gained recognition for its versatility, affordability, and robust Wi-Fi capabilities, positioning it as a fundamental enabler in the realm of IoT applications.

The significance of the ESP8266 transcends its physical size and cost-effectiveness. It boasts a comprehensive TCP/IP stack, facilitating effortless integration with wireless networks and the broader internet. Operating across various frequencies, supporting multiple Wi-Fi standards, and offering compatibility with diverse network infrastructures, the ESP8266 has become an essential component in shaping wireless communication.

The rapid ascent of ESP8266 to prominence can be attributed to its role as an IoT workhorse, a linchpin in the ever-evolving world of interconnected devices and smart technologies. This microcontroller has unlocked new possibilities for IoT applications, from environmental monitoring and home automation to industrial control systems and beyond.

This research paper embarks on a journey to explore the practical implementation of ESP8266's capabilities, with a particular focus on the establishment of a wireless connection between two ESP8266 boards. One board serves as an access point (AP), equipped with a BMP280 sensor for environmental data collection. The other board operates in station mode, featuring an OLED display for real-time data visualization. The primary objective is to enable seamless data exchange between these boards through HTTP GET requests and responses, underscoring ESP8266's potential within the dynamic sphere of IoT, wireless communication, and computer networks. As we delve deeper into the intricacies of this research, we illuminate not only the hardware and software components, libraries, and frameworks that enable wireless data exchange but also the broader context of computer networks. This paper serves as a comprehensive exploration of ESP8266's capabilities and the boundless possibilities inherent in this versatile microcontroller, both within the confines of this project and the wider landscape of IoT, wireless communication, and computer networks.

II. System Architecture Diagram

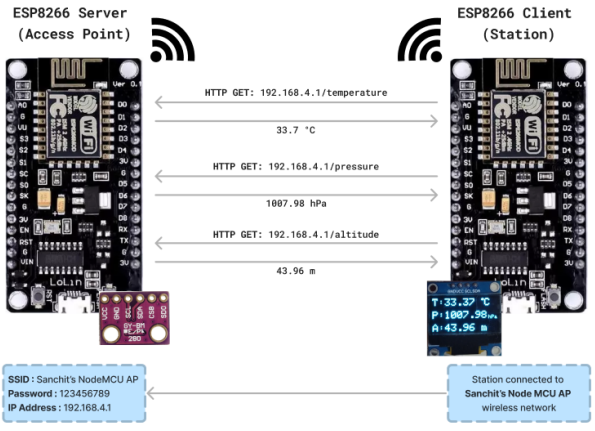

Fig. 1. Visualisation of HTTP GET Request and Response

III. Methodology

A. Hardware and Components

The core of this project is built upon the utilization of ESP8266 microcontrollers, specifically the ESP8266 modules. These modules are chosen for their combination of a compact form factor and integrated Wi-Fi capabilities, making them an ideal choice for our intended applications. One ESP8266 board is assigned the pivotal role of an access point (AP), establishing a Wi-Fi network and hosting the necessary infrastructure for data exchange. This AP is also equipped with a BMP280 sensor, which excels in environmental data collection, particularly in measuring temperature and atmospheric pressure.

Complementing this, the second ESP8266 board is configured in station mode, serving as the client in the wireless communication system. Its key feature is the integration of an OLED display, which becomes the canvas for real-time data visualization. This OLED display enhances the project's user-friendliness, providing a tangible and intuitive interface for users to access and interpret the collected data. The carefully orchestrated synergy between these hardware components ensures that the project is poised for success in diverse applications, from environmental monitoring to industrial control systems.

B. Software and Programming

In the realm of programming and controlling the ESP8266 boards, the project relies on the extensively embraced Arduino Integrated Development Environment (IDE) as its programming platform of choice. This decision is supported by the fact that the Arduino IDE seamlessly interfaces with the ESP8266 micro-controllers, thanks to the inclusion of the ESP8266 Arduino core. This core extends the IDE's capabilities, making it fully compatible with the ESP8266 modules, and thus an ideal choice for our development needs.

One of the significant advantages of employing the Arduino IDE is the simplification of the setup and coding processes for the ESP8266 boards. With a user-friendly interface and a rich ecosystem of libraries and resources, Arduino enables streamlined development and a shorter learning curve, ensuring accessibility and ease of use. The project utilizes the standard Arduino programming language, which is well-documented and understood by a vast community of developers, making it an inclusive platform for researchers and enthusiasts from diverse backgrounds. This strategic choice empowers a broader audience to engage with the project and leverage its potential for various applications in the field of wireless communication and IoT.

C. Wi-Fi Configuration

At the very core of this project is the intricate configuration of Wi-Fi connectivity between the access point (AP) and the station ESP8266 board. The project hinges on both ESP8266 modules adeptly establishing a stable and secure Wi-Fi connection, with the AP ESP8266 assuming the pivotal role of a network host.

This intricate Wi-Fi configuration is made possible by the utilization of the "ESP8266WiFi.h" library, a key component that underpins the communication infrastructure. The library's capabilities in configuring, managing, and securing the Wi-Fi connection are indispensable, ensuring that the project operates seamlessly and reliably in the context of IoT and wireless communication. With this firm foundation in place, the project is well-prepared to explore diverse applications, guaranteeing the integrity and security of data exchanged between the two ESP8266 boards.

D. Sensor Data Acquisition

The core of our data acquisition process revolves around the BMP280 sensor, which has been seamlessly integrated into the Access Point (AP) ESP8266 board. The BMP280 sensor plays a pivotal role in our project by serving as the primary data collection component, allowing us to obtain precise measurements of two fundamental environmental parameters: temperature and atmospheric pressure. These parameters have wide-ranging applications across various fields, making the BMP280 sensor an incredibly versatile asset for our project.

To facilitate communication between the BMP280 sensor and the AP ESP8266 board, we employ the I2C communication protocol. This protocol is not only renowned for its reliability but is also a widely accepted method for data retrieval. It simplifies the exchange of data between the sensor and the ESP8266 and, critically, guarantees the accuracy of our measurements.

An indispensable tool in our project's toolkit is the "Adafruit_BMP280.h" library. This library plays a pivotal role in establishing seamless communication with the BMP280 sensor, allowing us to acquire precise data concerning temperature and atmospheric pressure. By thoughtfully combining these components and leveraging the capabilities of the library, we ensure the accuracy and reliability of the environmental data we collect. This, in turn, enhances the applicability of our project across a diverse range of domains, from weather monitoring to industrial control systems, underscoring the BMP280 sensor's significance.

The successful operation of this component is underpinned by the "Adafruit_BMP280.h" library and the fundamental data communication provided by the "Wire.h" library. These libraries play a crucial role in ensuring the precise acquisition of temperature and atmospheric pressure data.

E. Data Visualization

At the heart of our station ESP8266, configured for seamless communication, lies the SSD1306 OLED display. This high-resolution OLED display serves as our platform for visualizing data in real-time, offering users an intuitive and dynamic interface to interpret the collected sensor data. What sets this display apart is its ability to render both textual and graphical information with precision and clarity.

The magic behind the dynamic rendering on the OLED screen is the ”Adafruit_SSD1306.h” library. This library serves as the engine that powers our data visualization process, ensuring that information is not only readily accessible but also comprehensible to users. It streamlines the process of presenting data, making it more engaging and user-friendly.

The elegant integration of OLED technology and the ”Adafruit_SSD1306.h” library is a cornerstone of our project's appeal. It enhances the user experience, making our solution compelling and versatile for a wide array of applications. Whether it's environmental monitoring, home automation, or industrial control systems, the combination of OLED technology and specialized libraries amplifies the project's user-friendliness and functionality.

The successful operation of the OLED display and its synergy with our project are driven by the ”Adafruit_SSD1306.h” library, alongside the fundamental data communication enabled by the ”Wire.h” library. These libraries ensure that data is not just collected but also presented with precision and style, making our project both informative and aesthetically pleasing.

F. HTTP Data Exchange

To establish a robust data exchange mechanism between the access point (AP) and the station ESP8266 boards, an HTTP-based approach has been meticulously adopted. In this setup, the AP ESP8266 operates as a dedicated server, proficiently hosting a web server designed to respond promptly to incoming HTTP GET requests initiated by the station ESP8266, acting in the capacity of a client. This streamlined architecture facilitates the seamless transmission of real-time data from the AP to the station, where it is promptly displayed on the OLED screen.

The implementation of this HTTP-based framework is significantly empowered by the incorporation of essential libraries. On the server side, the ”ESPAsyncWebServer.h” library plays a pivotal role in configuring and managing the HTTP server on the AP ESP8266, ensuring that it efficiently handles incoming data requests. Simultaneously, on the client side, the ”ESP8266HTTPClient.h” library serves as the linchpin for initiating and meticulously managing HTTP GET requests directed towards the server, ensuring reliable data retrieval.

In addition, the ”ESP8266WiFi.h” and ”WiFiClient.h” libraries are instrumental in fortifying the client's capabilities, ensuring a steadfast and dependable connection to Wi-Fi networks. This comprehensive integration of HTTP-based data exchange, coupled with the judicious use of these libraries, exemplifies the project's commitment to efficient, secure, and seamless wireless communication between ESP8266 boards.

IV. Working

The project harnesses the synergy of two ESP8266 boards, collaborating seamlessly to establish wireless communication. One ESP8266 board assumes the Access Point (AP) configuration, functioning as the server within the system. This AP, fortified with a BMP280 sensor, serves as an environmental monitoring instrument with the capability to measure critical parameters, including temperature, pressure, and altitude.

In contrast, the second ESP8266 board operates in Station mode, embodying the client's role. This client ESP8266 is thoughtfully integrated with an OLED display, which acts as the project's dynamic data visualization platform. The client periodically initiates HTTP GET requests to dedicated endpoints hosted by the server, requesting distinct sensor readings. These endpoints are conveniently named as /temperature, /pressure, and /altitude.

The process of sensor data retrieval is initiated upon receiving a GET request. The BMP280 sensor, deployed on the server side, is entrusted with the task of delivering precise and up-to-date environmental measurements. The sensor provides these measurements through dedicated methods, such as bmp.readTemperature(), bmp.readPressure(), and bmp.readAltitude(), ensuring data accuracy.

Subsequent to successful data acquisition, the server diligently assembles a comprehensive HTTP response. It may include standard HTTP status codes, with 200 (OK Code) signifying a successful response and -1 (Error Code) indicating an error scenario. This data-enriched response is promptly transmitted back to the client over the established Wi-Fi connection.

At the client's end, the response is subjected to careful interpretation. The primary focus is on extracting the sensor data component, which is subsequently directed to the OLED display for real-time visualization. The OLED display, serving as the project's user interface, plays a pivotal role in presenting users with the most current environmental readings, ensuring accessibility and clarity.

This cyclical process of HTTP GET request and response takes place at predefined intervals, typically set at 5-second

intervals. This rhythmic data exchange strategy ensures that the client consistently acquires and visually portrays the latest sensor data. The project's adaptability transcends its utility across a diverse range of applications, spanning weather monitoring, home automation, and environmental sensing. This dynamic synergy exemplifies the immense potential of ESP8266 boards and Wi-Fi communication in creating effective and interactive solutions within the realm of the Internet of Things (IoT).

V. Results

A. Access Point/Server Setup

In this subsection, we highlight the successful configuration of the Access Point (AP) ESP8266. We delve into the steps involved in setting up the AP, which serves as the server in our system. This includes details about establishing a Wi-Fi network, configuring network parameters, and ensuring a stable connection. We also discuss the specific library or code used to achieve this successful setup.

```
Setting AP (Access Point)…AP IP address: 192.168.4.1
```

Fig. 2. Successful Access Point/Server Setup

B. Station/Client Setup

Here, we outline the successful setup of the Station ESP8266, configured to operate in station mode. We explain how it was configured to connect to the AP ESP8266, detailing the Wi-Fi credentials used and any necessary settings. This section highlights the crucial role of the client ESP8266 in the project.

```
Connecting to Sanchit's NodeMCU AP
..............
Connected to WiFi
```

Fig. 3. Successful Station/Client Setup

C. Data Reception

This subsection delves into the meticulous process of receiving data from the BMP280 sensor, showcasing its remarkable accuracy and reliability in measuring temperature, atmospheric pressure and approximate altitude. To illustrate the sensor's performance in providing essential environmental data, we present specific measurements. Furthermore, we confirm the success of data transmission by displaying the HTTP response code 200 (OK), affirming the precision and effectiveness of our system.

```
HTTP Response code: 200
HTTP Response code: 200
HTTP Response code: 200
Temperature: 33.42 *C - Pressure: 1007.96 hPa - Approx. Altitude: 44.30 m
```

Fig. 4. Successful Data Reception

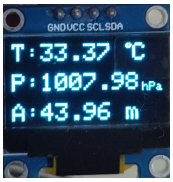

Fig. 5. Data Visualisation on SSD1306 OLED

D. Connection Failed

In this section, we tackle scenarios where the connection between the AP and the Station ESP8266 encountered challenges. We detail the issues that emerged, including the emergence of error codes and disruptions in communication. Notably, when the AP network experiences disturbances, it results in the appearance of the error code -1 at the Station/Client side.

```
Error code: -1
Error code: -1
Error code: -1
Temperature: -- *C - Pressure: -- hPa - Approx. Altitude: -- m
```

Fig. 6. Connection Failed

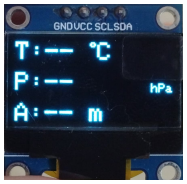

Fig. 7. Connection Failed Visualisation on SSD1306 OLED

VI. Conclusion

This project showcases the remarkable capabilities of ESP8266 microcontrollers in enabling seamless wireless communication. The implementation of an access point (AP) and a station (client) mode, supported by a robust set of libraries and modules, exemplifies the potential of these versatile devices in the realm of Internet of Things (IoT) and computer networks. The collaboration between two ESP8266 boards, one serving as a server equipped with a BMP280 sensor and the other as a client with an OLED display, exemplifies how IoT can be leveraged for real-time data exchange. The server's role in acquiring sensor data and the client's function in displaying this data on an OLED screen offer practical applications in areas such as weather monitoring, home automation, and environmental sensing. This project underscores the significance of ESP8266 boards as powerful tools for creating responsive

and interactive IoT systems that can truly redefine the way we connect and communicate in the digital age.

VII. Future Scope

In the realm of future work, several exciting avenues for system enhancement emerge. Diversifying the range of integrated sensors opens the door to a more comprehensive environmental monitoring solution, with sensors for humidity, air quality, light, or sound providing a broader spectrum of data. Device integration could equip the client ESP8266 board with actuators or relays, enabling it to not only display data but also take actions in response to sensor information, extending its utility to control smart home devices or appliances. User interface enhancements, such as interactive graphical interfaces on the OLED display, can elevate the user experience, potentially incorporating touchscreen capabilities and advanced data visualization. Data logging and remote storage could support long-term data analysis and trend identification, while automation and alerts could create early warning systems. Network security and encryption may play a pivotal role in safeguarding data, ensuring privacy and integrity. Finally, energy efficiency optimizations could enhance battery-operated device longevity and reduce operational costs. These avenues collectively represent the future potential for advancing the system's capabilities.

VIII. Acknowledgement

We would like to express our sincere gratitude to Prof. Akash Mecwan, whose guidance and expertise were instrumental in the successful completion of this project. Prof. Mecwan's valuable insights, unwavering support, and commitment to fostering a learning environment were indispensable throughout the project's development. His mentorship not only enriched our understanding of the subject matter but also inspired us to explore new horizons in the field of computer networks and the Internet of Things. I am deeply appreciative of his contributions to this endeavour.

References

[1] https://www.aranacorp.com/en/configure-an-esp8266-as-a-wi-fi-access-point/
[2] https://arduino-esp8266.readthedocs.io/en/stable/esp8266wifi/station-class.html
[3] https://www.bosch-sensortec.com/products/environmental-sensors/pressuresensors/bmp280/
[4] https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf
[5] https://www.javatpoint.com/computer-network-http

Appendix

Access-Point/Server Code

```
#include <ESP8266WiFi.h>
#include "ESPAsyncWebServer.h"
#include <Wire.h>
#include <Adafruit_BMP280.h>

// Set access point network credentials
const char* ssid = "Sanchit's NodeMCU AP";
const char* password = "123456789";
#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BMP280 bmp; // I2C

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

String readTemp() {
  return String(bmp.readTemperature());
  //return String(1.8 * bmp.readTemperature()
      + 32);
}

String readPres() {
  return String(bmp.readPressure() / 100.0F);
}

String readAlti() {
  return
      String(bmp.readAltitude(SEALEVELPRESSURE_HPA));
}



void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);
  Serial.println();

  // Setting the ESP as an access point
  Serial.print("Setting AP (Access Point)");
  // Remove the password parameter, if you
      want the AP (Access Point) to be open
  WiFi.softAP(ssid, password);

  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);

  server.on("/temperature", HTTP_GET,
      [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain",
        readTemp().c_str());
  });

  server.on("/pressure", HTTP_GET,
      [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain",
        readPres().c_str());
  });

  server.on("/altitude", HTTP_GET,
      [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain",
        readAlti().c_str());
  });

  if (!bmp.begin(0x76)) {
    Serial.println("Could not find a
    valid BMP280 sensor, check wiring!");
    while (1);
  }

  // Start server
  server.begin();
}
```

```
void loop(){

}
```

Station/Client Code

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>

//Required credentials for connecting to the
    AP
const char* ssid = "Sanchit's NodeMCU AP";
const char* password = "123456789";

//Different domain name with URL path for
    respective requests
const char* serverNameTemp =
    "http://192.168.4.1/temperature";
const char* serverNamePres =
    "http://192.168.4.1/pressure";
const char* serverNameAlti =
    "http://192.168.4.1/altitude";

#include <Wire.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display
    width, in pixels
#define SCREEN_HEIGHT 64 // OLED display
    height, in pixels

// Declaration for an SSD1306 display
    connected to I2C (SDA, SCL pins)
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH,
    SCREEN_HEIGHT, &Wire, OLED_RESET);

String temperature;
String pressure;
String altitude;


unsigned long previousMillis = 0;
const long interval = 5000;

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

  if(!display.begin(SSD1306_SWITCHCAPVCC,
      0x3C)) {
    Serial.println(F("SSD1306 allocation
        failed"));
    for(;;); // Don't proceed, loop forever
  }
  display.clearDisplay();
  display.setTextColor(WHITE);

  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

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

void loop() {
  unsigned long currentMillis = millis();

  if(currentMillis - previousMillis >=
      interval) {
    // Check WiFi connection status
    if ((WiFi.status() == WL_CONNECTED)) {
      temperature =
          httpGETRequest(serverNameTemp);
      pressure = httpGETRequest(serverNamePres);
      altitude = httpGETRequest(serverNameAlti);

      Serial.println("Temperature: " +
          temperature + " *C - Pressure: " +
          pressure + " hPa - Approx. Altitude:
          " + altitude + " m");

      display.clearDisplay();

      // display temperature
      display.setTextSize(2);
      display.setCursor(0,0);
      display.print("T:");
      display.print(temperature);
      display.print(" ");
      display.setTextSize(1);
      display.cp437(true);
      display.write(248);
      display.setTextSize(2);
      display.print("C");

      // display pressure
      display.setTextSize(2);
      display.setCursor(0, 25);
      display.print("P:");
      display.print(pressure);
      display.setTextSize(1);
      display.setCursor(110, 33);
      display.print("hPa");

      // display altitude
      display.setTextSize(2);
      display.setCursor(0, 50);
      display.print("A:");
      display.print(altitude);
      display.print(" m");

      display.display();

      // save the last HTTP GET Request
      previousMillis = currentMillis;
    }
    else {
      Serial.println("WiFi Disconnected");
    }
  }
}

String httpGETRequest(const char* serverName)
    {
  WiFiClient client;
  HTTPClient http;
```

```
  // Your IP address with path or Domain name
      with URL path
  http.begin(client, serverName);

  // Send HTTP POST request
  int httpResponseCode = http.GET();

  String payload = "--";

  if (httpResponseCode>0) {
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    payload = http.getString();
  }
  else {
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  // Free resources
  http.end();

  return payload;
}
```