VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI-590018, KARNATAKA

PROJECT REPORT

on

"Multi-Hop Communication using ESP8266 NodeMCU units"

Project Report submitted in partial fulfillment of the requirement for the award of the degree of
Bachelor of Engineering
in
Electronics and Communication Engineering
For the academic year 2023-24

Submitted by

1CR21EC076 G K D S S P SAIRAM
1CR21EC190 SATHWIK R
1CR21EC225 TANUF AHAMED KASHIMJI

Under the guidance of

Internal
Prof P. Susheel Kumar
Asst. Professor
Department of ECE
CMRIT, Bengaluru

Department Of Electronics and Communication Engineering

CMR INSTITUTE OF TECHNOLOGY

#132, AECS LAYOUT, IT PARK ROAD, KUNDALAHALLI, BENGALURU-560037

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to Certify that the dissertation work **"Multi-Hop communication using ESP8266 NodeMCU units"** carried out by, G K D S S P SAIRAM, SATHWIK R , TANUF KASHIMJI USN: 1CR21EC076, 1CR21EC190, 1CR21EC225, bonafide students of **CMRIT** in partial fulfillment for the award of **Bachelor of Engineering** in **Electronics and Communication Engineering** of the **Visvesvaraya Technological University, Belagavi,** during the academic year **2023-24.**It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said degree.

Signature of Guide

Prof P. Susheel Kumar
Asst. Professor
Dept. of ECE, CMRIT
Bengaluru

Signature of HOD

Dr. R. Elumalai
Professor & HoD
Dept. of ECE, CMRIT
Bengaluru

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose consistent guidance and encouragement crowned our efforts with success.

We consider it as our privilege to express the gratitude to all those who guided in the completion of the project.

We express my gratitude to Principal, **Dr. Sanjay Jain,** for having provided me the golden opportunity to undertake this project work in their esteemed organization.

We sincerely thank DR**. R Elumalai**, HOD**,** Department of Electronics and Communication Engineering, CMR Institute of Technology for the immense support given to me.

We express my gratitude to our project guide **Prof P. Susheel Kumar,** Assistant Professor, for their support, guidance and suggestions throughout the project work.

Above all, we thank the Lord Almighty for His grace on us to succeed in this endeavor.

1CR21EC076 G K D S S P SAIRAM
1CR21EC190 SATHWIK R
1CR21EC225 TANUF AHAMED KASHIMJI

CONTENT

INTRODUCTION

This report presents the findings and outcomes of a project aimed at developing a multi-hop network using ESP8266 nodes. The project focused on creating a robust network infrastructure capable of transmitting data over multiple hops, facilitating communication between distant nodes within the network. Through a combination of hardware setup, software implementation, and networking protocols, our objective was to establish a reliable and efficient communication framework.

The report is structured to provide a comprehensive understanding of the project, beginning with an introduction to the background and motivation behind the project, followed by a review of relevant literature and existing solutions in the field of multi-hop networking. Subsequently, the report delves into the hardware setup, detailing the components used and the configuration process.

INTRODUCTION

The objectives of the project are clearly stated. In this case, the objective was to develop a multi-hub network using ESP8266 NodeMCU devices. The term "multi-hub network" implies a network architecture where multiple central nodes (hubs) are interconnected, facilitating communication between several devices.

The introduction also briefly discusses the scope of the project, indicating the boundaries and limitations within which the project was conducted. This could include factors such as time constraints, available resources, and technical challenges.

Overall, the introduction section serves to provide a concise overview of the project, highlighting its significance, objectives, and scope. It sets the context for the subsequent sections of the report, guiding the reader through the project's journey and findings.

NODE MCU ESP8266 :

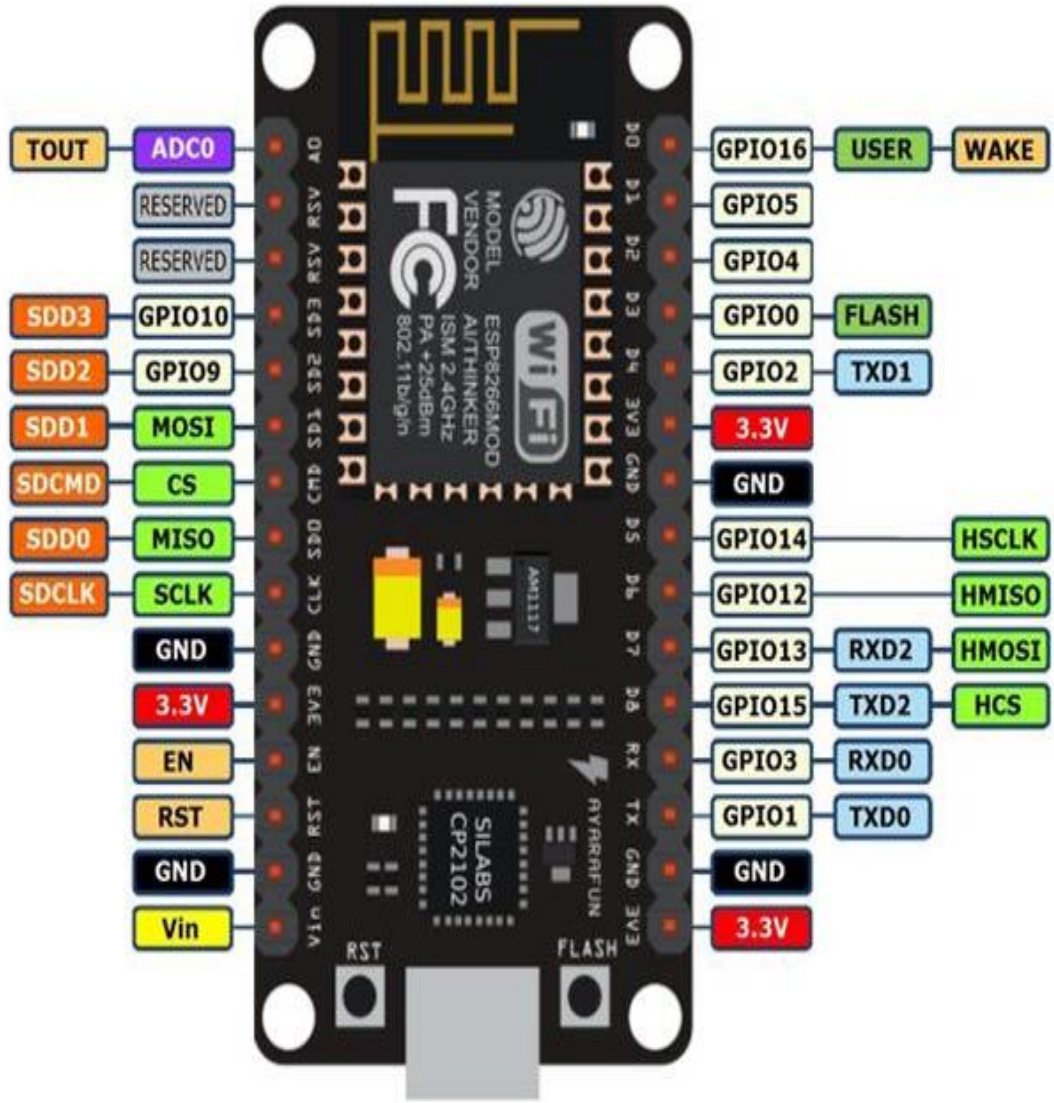
The ESP8266 NodeMCU is a development board based on the ESP8266 Wi-Fi module. It integrates the ESP8266 chipset with additional features such as USB-to-serial converter, voltage regulator, and GPIO pins, making it easy to prototype IoT projects and connect to Wi-Fi networks.

Key Features:

1. **ESP8266 Wi-Fi Module:** The heart of the NodeMCU, providing wireless connectivity to Wi-Fi networks.
2. **USB-to-Serial Converter:** Allows for easy programming and communication with the NodeMCU via USB.
3. **GPIO Pins:** General-purpose input/output pins for interfacing with sensors, actuators, and other electronic components.
4. **Voltage Regulator:** Regulates the input voltage to provide a stable power supply for the board.
5. **Flash Memory:** Built-in flash memory for storing firmware, programs, and data.
6. **Integrated Development Environment (IDE) Support:** Compatible with Arduino IDE, NodeMCU Lua, and other programming environments.

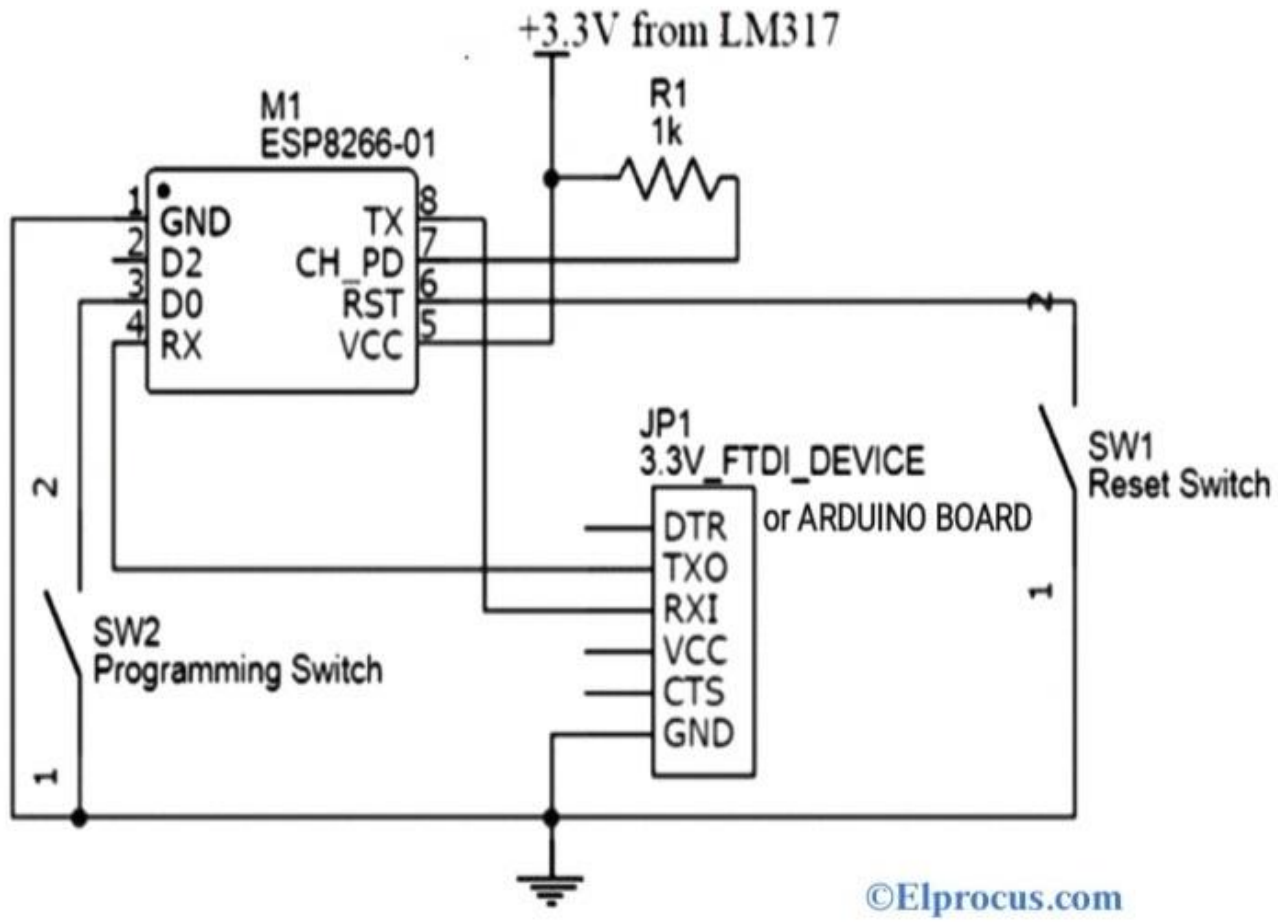
Pin Details:

- **VIN:** Input voltage pin. Typically connected to a power source between 5V and 12V.
- **GND:** Ground pin. Connected to the ground of the power supply.
- **3V3:** Output pin providing a regulated 3.3V supply.
- **EN:** Enable pin. Connected to VCC for normal operation.
- **D0-D8:** GPIO pins for digital input/output.
- **A0:** Analog input pin.
- **TX/RX:** Transmit and receive pins for serial communication.
- **SDA/SCL:** I2C data and clock pins.
- **MISO/MOSI/SCK:** SPI communication pins.
- **RST:** Reset pin for resetting the NodeMCU.



1. **VIN (Voltage Input):** This pin is used to supply voltage to the NodeMCU board. It typically accepts voltages between 5V and 12V.
2. **GND (Ground):** The ground pin is connected to the ground of the power supply or circuit to complete the electrical circuit.
3. **3V3 (3.3V Output):** This pin provides a regulated 3.3V output, which can be used to power external components that require a lower voltage.
4. **EN (Enable):** The enable pin is used to enable or disable the NodeMCU module. It is typically connected to VCC (3.3V) for normal operation.
5. **D0-D8 (Digital Pins):** These pins are general-purpose digital input/output pins. They can be used for interfacing with sensors, driving LEDs, or controlling other digital devices.
6. **A0 (Analog Input):** A0 is an analog input pin that can be used to read analog voltages from sensors or potentiometers.
7. **TX/RX (Serial Communication):** TX (Transmit) and RX (Receive) pins are used for serial communication with other devices such as computers or microcontrollers.
8. **SDA/SCL (I2C Pins):** SDA (Serial Data) and SCL (Serial Clock) pins are used for I2C communication with other devices such as sensors or displays.
9. **MISO/MOSI/SCK (SPI Pins):** These pins are used for SPI (Serial Peripheral Interface) communication with other devices such as SPI sensors or displays. MISO (Master In Slave Out), MOSI (Master Out Slave In), and SCK (Serial Clock) pins are used for data transfer.
10. **RST (Reset):** The reset pin is used to reset the NodeMCU module. It is typically pulled high by a pull-up resistor and connected to a push-button for manual reset.

Pin	Description	Typical Functionality
VIN	Voltage Input	Supply voltage (5V - 12V)
GND	Ground	Ground connection
3V3	3.3V Output	Regulated 3.3V output
EN	Enable	Enable/disable NodeMCU module
D0-D8	Digital Pins	General-purpose digital I/O
A0	Analog Input	Analog voltage input
TX/RX	Serial Communication	Transmit/receive serial data
SDA/SCL	I2C Pins	I2C serial data and clock
MISO/MOSI	SPI Pins	SPI data and clock
SCK	SPI Pins	SPI clock
RST	Reset	Reset NodeMCU module



1. Hardware Setup:

The hardware setup for the multi-hub network using ESP8266 NodeMCU involved careful selection and integration of components to ensure seamless operation and robust connectivity. The primary component, the ESP8266 NodeMCU board, was chosen for its versatility, compact size, and built-in Wi-Fi capabilities. These boards provided a cost-effective solution for wireless communication, making them ideal for IoT applications.

In addition to the ESP8266 NodeMCU boards, other hardware components such as sensors, actuators, and power supplies were integrated into the system as per the requirements of the specific application. Careful consideration was given to the compatibility and interoperability of these components to ensure smooth integration within the network.

The hardware setup was designed to be modular and scalable, allowing for easy expansion and customization as needed. This modular approach facilitated the addition of new devices to the network without significant modifications to the existing infrastructure.

Overall, the hardware setup laid the foundation for the multi-hub network, providing the necessary components for wireless communication and device integration.

2. Software Implementation:

The software implementation for the multi-hub network involved developing firmware for the ESP8266 NodeMCU devices using Arduino IDE or similar platforms. The programming was done in C/C++ to leverage the capabilities of the ESP8266 chipset and the Arduino framework.

The firmware implemented custom communication protocols and algorithms to enable data exchange between nodes within the network. These protocols included mechanisms for addressing, routing, and error detection to ensure reliable transmission over the wireless medium.

Special attention was given to optimizing the firmware for resource-constrained environments, such as the limited memory and processing power of the ESP8266 NodeMCU boards. Efficient data structures and algorithms were employed to minimize memory usage and maximize performance.

The software implementation also included provisions for over-the-air (OTA) updates, allowing for remote firmware updates and maintenance of the network. OTA updates ensured that the network remained up-to-date with the latest features and security patches without the need for physical intervention. Overall, the software implementation played a crucial role in the functionality and performance of the multi-hub network, providing the intelligence and control necessary for effective communication between devices.

3. Networking Protocol:

The networking protocol for the multi-hub network was designed to facilitate multi-hop communication between nodes, enabling data exchange across a distributed system. The protocol incorporated several key features to ensure reliable and efficient transmission over the wireless medium.

One of the primary features of the networking protocol was routing, which determined the path that data packets would take through the network to reach their destination. Various routing strategies were explored, including hop count, signal strength, and network topology, to optimize performance and scalability.

Addressing was another important aspect of the protocol, allowing nodes to identify and communicate with each other within the network. Unique identifiers were assigned to each node, ensuring that data packets were delivered to the correct destination.

Error detection and correction mechanisms were also incorporated into the protocol to mitigate the effects of transmission errors and packet loss. Techniques such as checksums and retransmission were used to ensure data integrity and reliability.

Overall, the networking protocol provided the framework for communication within the multi-hub network, enabling seamless data exchange between devices across multiple hops

Programming and Development:

- Programming Language: C/C++ (using Arduino IDE), Lua (using NodeMCU firmware).
- Development Environment: Arduino IDE, NodeMCU firmware, Espressif SDK.
- Libraries: Various libraries available for interfacing with sensors, actuators, and communication protocols.

OBJECTIVE OF THE PROJECT

- Our project aims to create a system using ESP8266 modules to connect multiple devices wirelessly and efficiently.
- Design a reliable communication method between devices using ESP8266 capabilities.
- Develop a protocol for easy configuration of each device in the network.
- Set up the hardware with ESP8266 modules for seamless wireless connectivity.
- Ensure efficient handling of different types of data, such as sensor readings and commands.
- Manage the network effectively, including node discovery and error handling.
- Create a user-friendly interface for monitoring and controlling the network.

WORKING / IMPLEMENTATION

The ESP8266 NodeMCU is a versatile development board that integrates the ESP8266 Wi-Fi module with additional features, making it an ideal platform for IoT projects. At its core, the ESP8266 Wi-Fi module provides wireless connectivity, allowing the NodeMCU to communicate with other devices and networks over Wi-Fi. The NodeMCU board includes a USB-to-serial converter, voltage regulator, and GPIO pins, enabling easy programming and interfacing with external components. Using programming languages such as C/C++ (via Arduino IDE) or Lua (using NodeMCU firmware), developers can write code to control the NodeMCU's behavior, such as reading sensor data, sending/receiving data over Wi-Fi, and interacting with other devices. The NodeMCU can operate as a standalone device, acting as a server or client in a network, or it can be used as a repeater to extend the range of a Wi-Fi network. With its compact size, low cost, and rich feature set, the ESP8266 NodeMCU is a powerful tool for building connected IoT applications.

CODE :

IP ADRESS CODE :

```
#iinclude<ESP8266WiFi.h>
Const char*ssid = “ YOUR SSID”;//your wifi SSID
Const char*password = “Your password”;//Your wifi password
        void setup( ){
         serial.begin(115200);
         delay(10);
 // connect to WiFi
Serial.print( );
Serial print(“ Connecting to”);
Serial println(ssid);
WiFi.begin(ssid,password);
while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
  }
   Serial.println("");
   Serial.println("WiFi connected");
  // Print the IP address of the server NodeMCU
   Serial.println("Server IP address: ");
   Serial.println(WiFi.localIP());
}
void loop() {

}
```

SERVER CODE :

```
#include <ESP8266WiFi.h>
        const char *ssid = "YOUR_SSID";      // Your Wi-Fi SSID
        const char *password = "YOUR_PASSWORD";  // Your Wi-Fi password
        const int serverPort = 80;          // Port to listen on
        WiFiServer server(serverPort);
        void setup() {
        Serial.begin(115200);
         delay(10);
        // Connect to WiFi
        Serial.println();
        Serial.print("Connecting to ");
        Serial.println(ssid);
         WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
 }
Serial.println("");
        Serial.println("WiFi connected");
         // Start the server
        server.begin();
        Serial.println("Server started");  }
        void loop() {
// Check if a client has connected
        WiFiClient client = server.available();
        if (!client) {
         return;  }
   // Wait until the client sends some data
         Serial.println("New client connected");
        while (!client.available()) {
        delay(1);  }
 // Read the first line of the request
        String request = client.readStringUntil('\r');
        Serial.println(request);
         client.flush();
        // Send a response to the client
        client.println("Hey Sathwik");
        client.println("Content-Type: text/html");
        client.println(""); // Blank line
        client.println("<h1>Hello from NodeMCU 1!</h1>");
 // The client will be disconnected after the response is sent
        delay(1);
         Serial.println("Client disconnected");
}
```

REPEATER CODE :

```
#include <ESP8266WiFi.h>
        const char *ssid = "YOUR_SSID";      // Your Wi-Fi SSID
        const char *password = "YOUR_PASSWORD";  // Your Wi-Fi password
        const int serverPort = 80;          // Port to listen on
        const char *serverIP = "192.168.209.210"; // Server IP address
  WiFiServer server(serverPort);
  void setup() {
        serial.begin(115200);
        delay(10);
        // Connect to Wi-Fi
        Serial.println();
         Serial.print("Connecting to ");
        Serial.println(ssid);  WiFi.begin(ssid, password);

        while (WiFi.status() != WL_CONNECTED)
        {
                delay(500);
                Serial.print(".");
        }
        Serial.println("");
        Serial.println("WiFi connected");
        // Start the server
        server.begin();
         Serial.println("Server started");
}
void loop(){
// Check if a client has connected
        WiFiClient client = server.available();
         if (!client) {
        return;
 }
// Wait until the client sends some data
         Serial.println("New client connected");
        while (!client.available()) {
        delay(1);
 }
// Read the first line of the request
        String request = client.readStringUntil('\r');
        Serial.println(request);
        client.flush();
        // Connect to the server
        WiFiClient serverClient;
         Serial.print("Connecting to server: ");
        Serial.println(serverIP);
        if (!serverClient.connect(serverIP, serverPort))
 {
        Serial.println("Connection to server failed");
```

```
        return;
}
        Serial.println("Connected to server");
  // Forward the request to the server
         serverClient.print(request);

        // Wait for the response from the server
        while (serverClient.connected() && !serverClient.available())
{
        delay(1);
 }
 // Read and forward the response to the client
         const int bufferSize = 1024; // Adjust the buffer size as needed  uint8_t
        buffer[bufferSize];
        size_t bytesRead;
        while ((bytesRead = serverClient.readBytes(buffer, bufferSize)) > 0) {
        client.write(buffer, bytesRead); // Forward response to client
 }
        Serial.println();
         Serial.println("Request forwarded");
// Disconnect from the server
         serverClient.stop();  client.stop();
// Close the connection with the client
        Serial.println("Client disconnected");
}
         Serial.println("Disconnected from server");
```

CLIENT CODE :

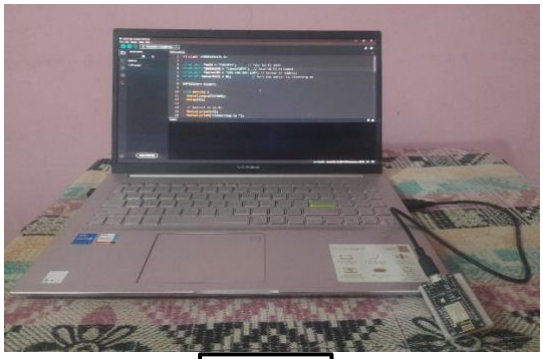
```
#include <ESP8266WiFi.h>
        const char *ssid = "YOUR_SSID";      // Your Wi-Fi SSID
        const char *password = "YOUR_PASSWORD";  // Your Wi-Fi password
        const char *serverIP = "192.168.209.234"; // Server IP address
        const int serverPort = 80;          // Port the server is listening on
        WiFiClient client;
void setup()
{
        Serial.begin(115200);
        delay(10);
        // Connect to Wi-Fi
        Serial.println();
         Serial.print("Connecting to ");
         Serial.println(ssid);
          WiFi.begin(ssid, password);
        while (WiFi.status() != WL_CONNECTED)
 {
        delay(500);
         Serial.print(".");
 }
        Serial.println("");
         Serial.println("WiFi connected");
// Connect to server
        Serial.print("Connecting to server: ");
         Serial.println(serverIP);
         if (!client.connect(serverIP, serverPort))
 {
         Serial.println("Connection failed");
        return;
 }
        Serial.println("Connected to server");
}
        void loop() {
// Send a request to the server
                client.println("GET / HTTP/1.1");
        client.println("Host: 192.168.209.234"); // Server IP
        client.println("Connection: close");
        client.println();
 // Wait for the response
        while (client.connected() && !client.available())
{
        delay(1);
 }
        // Print the response
         while (client.available())
        String line = client.readStringUntil('\r');
        Serial.print(line);
```

```
 }
        Serial.println();
         Serial.println("Request sent");
// Disconnect from the server
        client.stop();

        Serial.println("Disconnected from server");
// Wait before sending the next request

        delay(5000);
}
```

SETUP



Server

Repeater

Client

OUTPUT

```
arduino                                                    Copy code

Connecting to YourWiFiNetwork
........
WiFi connected
Hey Sathwik from Server
Hey Sathwik from Server
Hey Sathwik from Server
...
```

CONCLUSION / FUTURE WORK

The completion of the initial phase of our project marks a significant milestone, as we have successfully established Wi-Fi connections with ESP8266 modules. This accomplishment not only validates our chosen approach but also lays a solid foundation for the subsequent phases of development. Moving forward, our focus will be on implementing additional features to fully realize our project objectives. This includes refining communication protocols to ensure efficient data exchange between nodes and developing a user-friendly interface to enhance accessibility and usability.

With the successful establishment of Wi-Fi connections, we are now well-positioned to embark on the next phase of our project journey. Our primary goal remains to create a robust multi-node hub system leveraging ESP8266 technology. This system will serve as a cornerstone for connecting and managing devices in offline locations, where traditional internet connectivity may be limited or unavailable.

Our progress thus far demonstrates that we are on track to achieve our project goals. The successful establishment of Wi-Fi connections signifies the technical feasibility of our approach, while the upcoming focus on additional features underscores our commitment to creating a comprehensive solution. As we continue to advance, we are confident that our project will not only meet but exceed expectations, delivering a versatile and reliable multi-node hub system that can be deployed in diverse environments."

This elaboration provides a more detailed explanation of the completion of the initial phase, the focus areas for the next phase, and the overarching goals and aspirations of the project.