Chapter 4
Connect to Networks

Dr. Zhang Jianwen

elezhan@nus.edu.sg

Office: E4-05-21

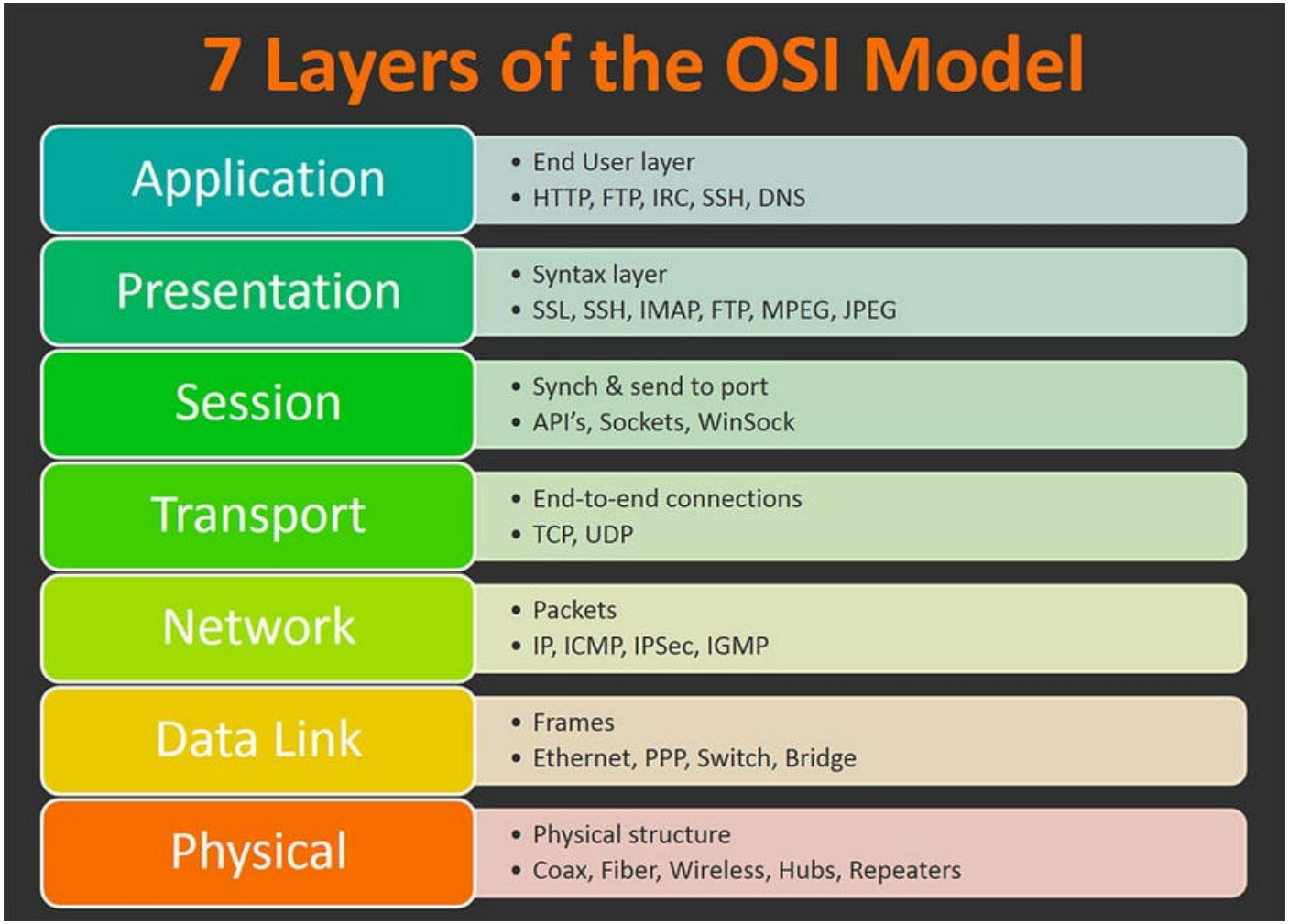
Hardware for the connections

- Dual core CPU, Wi-Fi and Bluetooth Low Energy (BLE) support on ESP32-S3 board.
  - Wi-Fi: IEEE 802.11 b/g/n standard, supporting 2.4 GHz band.
  - Bluetooth: Bluetooth v5.0 with Bluetooth Low Energy (BLE) support.
  - Integrated antenna for Wi-Fi and BLE.

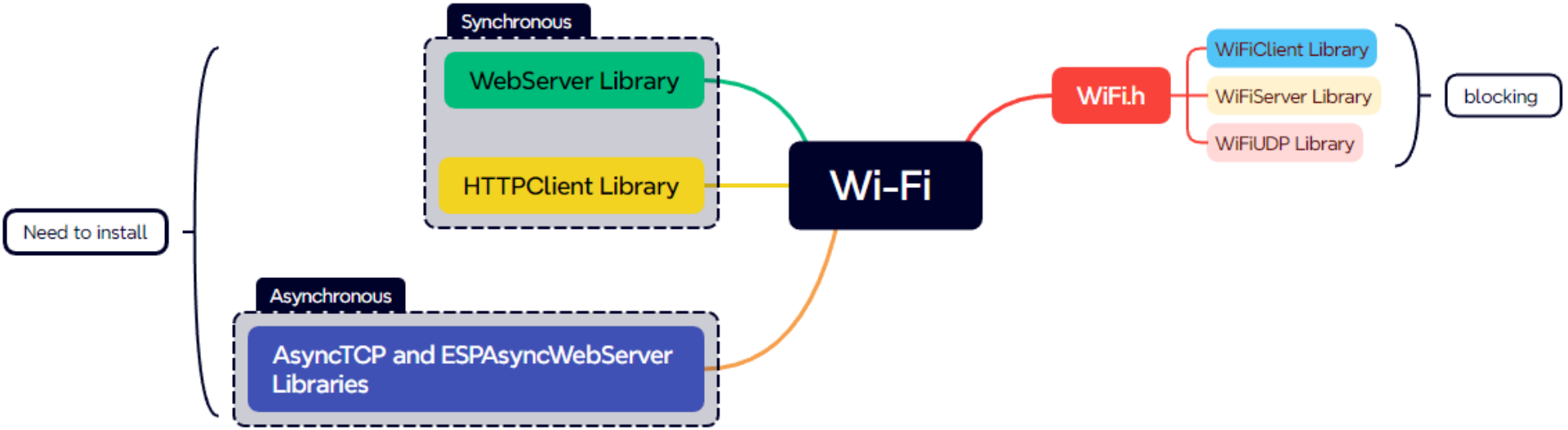
Wi-Fi Libraries

- Wi-Fi and Bluetooth are the basis for other communication protocols. They provide the low-level connections for any high level application.
- Libraries
  - WiFi library (WiFiClient, WiFiServer and WiFiUDP): primary library used for handling Wi-Fi connections.
  - Synchronous webservice: HTTPClient and WebServer libraries
  - Asynchronous webservice: ESPAsyncWebServer and AsyncTCP libraries

Wi-Fi Libraries



Main libraries for Wi-Fi and applications



Wi-Fi connection APIs (STA mode)

- `WiFi.begin(ssid, password);`

Connects to a specified Wi-Fi network using the SSID and password as STA.

- **ssid**: the name of the Wi-Fi network (as a const char*).
- **Password**: the password for the Wi-Fi network (as a const char*).

- `WiFi.status();`

Returns the current connection status. Key status codes include:

- WL_CONNECTED: Connected to a network.
- WL_NO_SSID_AVAIL: SSID not available.
- WL_CONNECT_FAILED: Connection failed.
- WL_IDLE_STATUS: Idle status (not connected).
- WL_DISCONNECTED: Disconnected from a network.

- `WiFi.macAddress();`

Returns the MAC address of the ESP32-S3 in WIFI_STA mode.

- `WiFi.localIP();`

Returns the IP address assigned to the ESP32-S3 by the DHCP server.

```
#include <WiFi.h>  // Include the Wi-Fi library

const char* ssid = "YourNetworkName";     // Replace with your network SSID (name)
const char* password = "YourPassword";    // Replace with your network password

void setup() {
  Serial.begin(115200);  // Initialize serial communication for debugging
  WiFi.begin(ssid, password);  // Start Wi-Fi connection

  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {  // Wait for the Wi-Fi to connect
    delay(1000);
    Serial.print(".");
  }

  Serial.println("\nConnected to Wi-Fi network");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());  // Print the IP address
}

void loop() {
  // Your main code here
}
```

Wi-Fi connection APIs (STA mode)

- `WiFi.disconnect();`

Disconnects from the current Wi-Fi network. It is used to switch between different Wi-Fi networks, reset the Wi-Fi connection and save power by turning off the Wi-Fi connection when it is not needed.

- `WiFi.reconnect();`

Try reconnecting to the last known Wi-Fi network.

- `WiFi.isConnected();`

Returns true if the ESP32 is currently connected to a Wi-Fi network.

- `WiFi.setHostname(hostname);`

Sets a custom hostname for the ESP32 in STA mode.

- `WiFi.begin(ssid, password, channel, bssid);`

Start Wi-Fi STA with specified Wi-Fi channel and BSSID (MAC address of the AP), which can speed up the connection process.

Reconnecting to Wi-Fi Automatically

```
#include <WiFi.h>

const char* ssid = "your-SSID";
const char* password = "your-PASSWORD";

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

  // Begin Wi-Fi connection in Station Mode
  WiFi.begin(ssid, password);

  Serial.print("Connecting to Wi-Fi...");

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }

  Serial.println("\nConnected to Wi-Fi");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}
```

```
void loop() {
  // Check if the ESP32 is still connected to Wi-Fi
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("Wi-Fi connection lost.
                                  Reconnecting...");
    WiFi.disconnect();
    WiFi.reconnect();

    while (WiFi.status() != WL_CONNECTED) {
      delay(1000);
      Serial.print(".");
    }

    Serial.println("\nReconnected to Wi-Fi");
  }

  delay(5000);  // Check Wi-Fi status every 5 seconds
}
```

Wi-Fi connection APIs (STA mode)

- `WiFi.config(local_ip, gateway, subnet, primaryDNS, secondaryDNS);`

Configures the static IP address, gateway, subnet mask, primary, and secondary DNS servers. All the parameters are IPAddress type data. The local_ip is the only compulsory parameter. The rest, if not provided, will be obtained via DHCP.

- ❖ local_ip: The static IP address you want to assign to the ESP32.
- ❖ gateway: (Optional) The IP address of the network gateway (usually your router).
- ❖ subnet: (Optional)The subnet mask (usually 255.255.255.0 for most home networks).
- ❖ primaryDNS:(Optional)The primary DNS server (the IP address used to resolve domain names).
- ❖ secondaryDNS:(Optional)The secondary DNS server (used if the primary fails).

Setting a Static IP Address

```
#include <WiFi.h>

const char* ssid = "your-SSID";
const char* password = "your-PASSWORD";

// Define a static IP address, gateway, and subnet
IPAddress local_IP(192, 168, 1, 184);  // Change to your preferred static IP
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);

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

  // Configure the ESP32 to use a static IP
  if (!WiFi.config(local_IP, gateway, subnet)) {
    Serial.println("Failed to configure static IP");
  }

  // Begin Wi-Fi connection in Station Mode
  WiFi.begin(ssid, password);

  Serial.print("Connecting to Wi-Fi...");

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }

  Serial.println("\nConnected to Wi-Fi");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());  // Print the ESP32's static IP address
}

void loop() {
  // Put your main code here, to run repeatedly:
}
```

Wi-Fi connection APIs

- `WiFi.RSSI();`

Returns the Received Signal Strength Indicator (RSSI) of the connected network.

- `WiFi.mode(WIFI_STA);`

Sets the mode of the Wi-Fi (e.g., station mode, access point mode, or both) and getting the current Wi-Fi mode. The ESP32 supports different modes below. Return a WiFiMode_t type.

- **WIFI_STA**: Station mode, which is used to connect the ESP32 to an existing Wi-Fi network as a client.
- **WIFI_AP**: Access point mode, where the ESP32 creates its own network and allows other devices to connect to it.
- **WIFI_AP_STA**: Both station and access point mode. The ESP32 can connect to an existing network and create its own network simultaneously.
- **WIFI_OFF**: Turns off the Wi-Fi interface.

```
#include <WiFi.h>
// Wi-Fi credentials for Station mode (client)
const char* ssid_STA = "Your_SSID";
const char* password_STA = "Your_PASSWORD";

// Wi-Fi credentials for Access Point (AP) mode
const char* ssid_AP = "ESP32_AP";           // SSID of the AP you create
const char* password_AP = "12345678";       // Password of the AP

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

  // Set the ESP32-S3 to dual-mode: Access Point (AP) and Station (STA)
  WiFi.mode(WIFI_AP_STA);

  // Start Station (STA) and attempt to connect to the Wi-Fi network
  WiFi.begin(ssid_STA, password_STA);
  Serial.println("Connecting to WiFi (Station mode)...");

  // Wait until connected to Wi-Fi
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }

  Serial.println("");
  Serial.print("Connected to Wi-Fi network with IP Address: ");
  Serial.println(WiFi.localIP());

  // Set up the Access Point (AP)
  WiFi.softAP(ssid_AP, password_AP);
  Serial.println("Access Point (AP) started");

  // Get the IP address of the AP
  IPAddress IP_AP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP_AP);
}

void loop() {
  // In the loop, you can add any functionality or checks for clients
  //connected to the AP
}
```

Wi-Fi connection APIs (AP mode)

- `WiFi.softAP(ssid, password);`

Creates a Wi-Fi network with a given SSID (name) and password. If no password is provided, it creates an open network.

- `WiFi.softAPmacAddress();`

Returns the MAC address of the ESP32-S3 in WIFI_AP mode.

- `WiFi.softAPIP();`

Returns the IP address of the ESP32 as the access point (such as 192.168.4.1)

- `WiFi.softAPConfig(local_ip, gateway, subnet);`

Configures the IP address, gateway, and subnet for the access point. Similar as the STA version with variations.

Wi-Fi connection APIs (AP mode)

- `WiFi.softAPgetStationNum();`

Returns the int number of devices (stations) currently connected to the access point.

- `WiFi.softAPdisconnect();`

Disconnects all connected stations from the access point..

- `WiFi.softAP(ssid, password, channel, ssid_hidden, max_connection);`

This extended version of WiFi.softAP() allows to set the channel, hide the SSID, and specify the maximum number of connections.

- **ssid**: AP network ID.
- **password**: password of the AP.
- **channel**: Wi-Fi channel used (int type).
- **ssid_hidden**: Boolean type
- **max_connection**: integer type.

Custom IP and Multiple Connections

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

// Set the SSID and password for the access point
const char* ssid = "Custom_AP";
const char* password = "mypassword";

// Custom IP settings
IPAddress local_ip(192, 168, 10, 1);
IPAddress gateway(192, 168, 10, 1);
IPAddress subnet(255, 255, 255, 0);

// Create a web server object
WebServer server(80);

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

  // Configure the access point with a custom IP
  WiFi.softAPConfig(local_ip, gateway, subnet);

  // Start the Wi-Fi access point
  WiFi.softAP(ssid, password, 6, 0, 8);  // 8 clients max, channel 6, visible SSID

  // Print the custom IP address of the access point
  IPAddress IP = WiFi.softAPIP();
  Serial.print("Access Point IP: ");
  Serial.println(IP);

  // Set up a route for the root URL
  server.on("/", []() {
    server.send(200, "text/html", "<h1>ESP32 Access Point with Custom IP</h1>");
  });

  // Start the web server
  server.begin();
  Serial.println("Web server started");
}
```

```
void loop() {
  // Handle incoming client requests
  server.handleClient();
}
```

Monitoring Connected Devices

```
#include <WiFi.h>

const char* ssid = "Monitor_AP";
const char* password = "password123";

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

  // Start the Wi-Fi access point
  WiFi.softAP(ssid, password);

  // Print the IP address of the access point
  IPAddress IP = WiFi.softAPIP();
  Serial.print("Access Point IP: ");
  Serial.println(IP);
}

void loop() {
  // Print the number of connected clients
  int numStations = WiFi.softAPgetStationNum();
  Serial.print("Connected devices: ");
  Serial.println(numStations);

  // Wait for 5 seconds before checking again
  delay(5000);
}
```

WiFiClient Library

- It is a part of the WiFi library.
- Create a TCP/IP client over Wi-Fi.
- The devkit can programmed as a client that can connect to a remote server over the internet or a local network, and send or receive data.
- Especially useful in IoT projects where the ESP32 needs to communicate with web services, APIs, or other devices.

WiFiClient use cases

- **HTTP Client**: Making HTTP requests to web servers, APIs, or cloud services.
- **MQTT Client**: Communicating with MQTT brokers for real-time data transfer in IoT applications.
- **Socket Communication**: Connecting to other devices using TCP sockets.
- **Local Network Communication**: Communicating with local servers like databases, web servers, or file servers.
- **Data Logging**: Sending sensor data to a remote server for storage or analysis.

WiFiClient Library API

- `WiFiClient client; : Creates an instance of a Wi-Fi client.`

- `client.connect(host, port): Connects to a specified host and port.`
  - **`host`**`: a domain name (e.g., "example.com") or an IP address in string format (e.g., "192.168.1.10").`
  - **`port`**`: port on the remote server, which could be 80 for HTTP.`
  - **`Return`**`: true if successful, otherwise false.`

- `client.write(data): Sends data to the connected server.`

- `client.read(): Reads data from the server.`

- `client.available(): Checks if data is available to read.`

- `client.stop(): Disconnects from the server.`

Simple HTTP GET Request

```
#include <WiFi.h>

const char* ssid     = "your_SSID";
const char* password = "your_PASSWORD";

const char* host = "example.com";
const int port = 80;  // Standard HTTP port

WiFiClient client;

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

  // Connect to the Wi-Fi network
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected.");

  // Connect to the remote server
  Serial.print("Connecting to ");
  Serial.print(host);
  Serial.print(":");
  Serial.println(port);

  if (client.connect(host, port)) {
    Serial.println("Connected to server!");
    // Once connected, you can send or receive data

    // Example of sending an HTTP GET request
    client.println("GET / HTTP/1.1");
    client.println("Host: example.com");
    client.println("Connection: close");
    client.println();
  } else {
    Serial.println("Connection failed.");
  }
}

void loop() {
  // Handle incoming data if connected
  while (client.connected() || client.available()) {
    if (client.available()) {
      String line = client.readStringUntil('\n');
      Serial.println(line);
    }
  }

  // Close the connection when the server is done responding
  if (!client.connected()) {
    Serial.println("Disconnecting.");
    client.stop();
  }
}
```

Sending Sensor Data to a Remote Server

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <DHT.h>

#define DHTPIN 4
#define DHTTYPE DHT11

const char* ssid = "your-SSID";
const char* password = "your-PASSWORD";

const char* host = "your-server.com";
const int httpPort = 80;  // HTTP port

DHT dht(DHTPIN, DHTTYPE);
WiFiClient client;

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

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nConnected to Wi-Fi");
}

void loop() {
  // Read temperature and humidity from the sensor
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor");
    return;
  }

  // Connect to the server
  if (client.connect(host, httpPort)) {
    String data = "temperature=" + String(temperature) + "&humidity="
+ String(humidity);

    // Send HTTP POST request
    client.println("POST /upload HTTP/1.1");
    client.println("Host: your-server.com");
    client.println("Content-Type: application/x-www-form-urlencoded");
    client.print("Content-Length: ");
    client.println(data.length());
    client.println();
    client.println(data);

    // Read the server's response
    while (client.connected() || client.available()) {
      if (client.available()) {
        String response = client.readStringUntil('\n');
        Serial.println("Server response: " + response);
      }
    }

    // Close the connection
    client.stop();
  } else {
    Serial.println("Failed to connect to server");
  }

  // Wait before sending the next reading
  delay(60000);  // 60 seconds delay
}
```

WiFiServer Library

- It is a part of the WiFi library.
- Create a TCP server over a Wi-Fi network.
- Listen for incoming client connections (from devices like smartphones, computers, or other IoT devices) and communicate with them.
- Useful when a server receives data from clients or serves responses (e.g., sending sensor data, controlling devices, etc.)

WiFiServer use cases

- **Web Server**: The ESP32 can serve HTML pages to web clients over the network, making it a great choice for web-based control and monitoring applications.
- **Device Control**: You can control devices (e.g., LEDs, motors) by sending commands to the ESP32 over a web or TCP server.
- **Data Monitoring**: The ESP32 can serve real-time data from sensors to clients via web browsers or TCP sockets.
- **Local Network Communication**: The ESP32 can facilitate communication between multiple devices on a local network using TCP.

WiFiServer library API

- `WiFiServer server(port);`

Creates an instance of the WiFiServer class that listens for incoming connections on the specified port.

- `void server.begin();`

Starts the server and listen to the incoming connections.

- `server.available();`

Checks if a client is trying to connect. It returns a **WiFiClient** object for communication or an empty object if no client is connected.

- `client.connected();`

Checks whether the client is still connected.

- `void server.stop();`
- Stops the server from listening.

WiFiServer library API

- `client.write(data);`
- `client.print(data);`
- `client.println(data);`

`Sends data to the connected client.`

- `client.read();`
- `client.readStringUntil();`

`Receives data from the client.`

Control an LED via Wi-Fi Web Server

```
#include <WiFi.h>

// Replace with your network credentials
const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";

WiFiServer server(80);

// Define the pin for LED
const int ledPin = 2;

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

  // Set up the LED pin
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // Start the server
  server.begin();
  Serial.println("Server started");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}
```

```
void loop() {
  WiFiClient client = server.available();

  if (client) {
    String request = client.readStringUntil('\r');
    client.flush();

    // Check the request to turn the LED on or off
    if (request.indexOf("/LED=ON") != -1) {
      digitalWrite(ledPin, HIGH); // Turn LED on
    } else if (request.indexOf("/LED=OFF") != -1) {
      digitalWrite(ledPin, LOW); // Turn LED off
    }

    // Prepare the response
    String response = "HTTP/1.1 200 OK\r\n";
    response += "Content-Type: text/html\r\n\r\n";
    response += "<html><body>";
    response += "<h1>ESP32 LED Control</h1>";
    response += "<p><a href=\"/LED=ON\">Turn LED ON</a></p>";
    response += "<p><a href=\"/LED=OFF\">Turn LED OFF</a></p>";
    response += "</body></html>";

    // Send the response
    client.print(response);

    // Close the connection
    client.stop();
  }
}
```

```
#include <WiFi.h>

// Replace with your network credentials
const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";

// Create a WiFiServer object on port 80 (HTTP)
WiFiServer server(80);

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

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

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

```
void loop() {
  // Check if a client has connected
  WiFiClient client = server.available();
  if (client) {
    Serial.println("New Client connected");

    // Wait until the client sends some data
    while (client.connected() && !client.available()) {
      delay(1);
    }

    // Read the request from the client
    String request = client.readStringUntil('\r');
    Serial.println("Received request: " + request);

    // Prepare the response
    String response = "HTTP/1.1 200 OK\r\n";
    response += "Content-Type: text/html\r\n\r\n";
    response += "<html><body><h1>Hello from ESP32!</h1></body></html>";

    // Send the response to the client
    client.print(response);

    // Close the connection
    client.stop();
  }
}
```

Simple Wi-Fi Web Server

Real-Time Sensor Data Server

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiServer.h>
#include <DHT.h>

#define DHTPIN 4
#define DHTTYPE DHT11

const char* ssid = "your-SSID";
const char* password = "your-PASSWORD";

DHT dht(DHTPIN, DHTTYPE);

// Create a web server on port 80
WiFiServer server(80);

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

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);

  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nConnected to Wi-Fi");

  // Start the server
  server.begin();
  Serial.println("Server started");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}
```

```
void loop() {
  // Check for a client connection
  WiFiClient client = server.available();
  if (client) {
    Serial.println("New Client Connected");
    client.readStringUntil('\r');  // Read the request
    client.flush();

    // Get temperature and humidity
    float temperature = dht.readTemperature();
    float humidity = dht.readHumidity();

    // Send an HTML response with sensor data
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println();
    client.println("<html><body>");
    client.println("<h1>ESP32 Sensor Data</h1>");
    client.print("<p>Temperature: ");
    client.print(temperature);
    client.println(" &#8451;</p>");
    client.print("<p>Humidity: ");
    client.print(humidity);
    client.println(" %</p>");
    client.println("</body></html>");
    client.println();

    // Close the connection
    client.stop();
    Serial.println("Client disconnected");
  }
}
```

WiFiUDP library

- It is a part of the WiFi library.
- Send and receive data using the User Datagram Protocol (UDP) over Wi-Fi.
- UDP is a lightweight, connectionless communication protocol that is ideal for use cases where speed is important, and where occasional packet loss is acceptable.
- It is often used in real-time communication systems, like video streaming, IoT sensors, and multiplayer games.

WiFiUDP use cases

- **Real-Time Sensor Data Transmission**: Sending data with minimal delay, such as temperature or location updates.
- **Broadcast Messaging**: Sending messages to multiple devices on the same network via UDP broadcasting.
- **Multiplayer Games**: Sending quick game updates between clients and a server in real-time.
- **IoT Device Discovery**: Devices can use UDP to broadcast discovery packets to find each other on the same network.
- **Streaming Data**: Streaming audio or video where packet loss can be tolerated (UDP doesn't ensure packet delivery, but it's faster than TCP).
- **Remote Control Systems**: Sending quick control commands over a network to remote devices.

WiFiUDP library API

- `WiFiUDP udp;  :`

Creates an instance of the WiFiUDP class.

- `udp.begin(port);`

Initializes UDP and binds it to the specified port.

- `udp.beginPacket(host, port);`

Starts a new UDP packet destined for a specified remote host and port.

```
udp.beginPacket("192.168.1.100", 1234);
```

- `udp.write(buffer, length);`

Writes data to the UDP packet.

```
udp.write("Hello ESP32", 10);
```

- `udp.endPacket();`

Marks the end of the packet and sends it.

WiFiUDP Library

- udp.parsePacket();

Checks if a UDP packet is available for reading and returns the size of the packet.

```
int packetSize = udp.parsePacket();
```

- udp.read(buffer, length);

Reads incoming data from the UDP buffer.

```
char incomingPacket[255];
udp.read(incomingPacket, 255);
```

- udp.remoteIP();

Gets the IP address of the sender of the received packet.

```
IPAddress senderIP = udp.remoteIP();
```

- udp.remotePort();

Gets the port number of the sender of the received packet.

```
int senderPort = udp.remotePort();
```

- udp.stop();

Closes the UDP connection.

```ino
#include <WiFi.h>
#include <WiFiUdp.h>

const char* ssid = "yourSSID";
const char* password = "yourPASSWORD";
WiFiUDP udp;

// Target IP address
IPAddress remoteIP(192, 168, 1, 100);
unsigned int remotePort = 1234; // Target port

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  udp.begin(1234); // Listening port
}

void loop() {
  udp.beginPacket(remoteIP, remotePort);
  udp.write("Hello, ESP32 UDP here!");
  udp.endPacket();
  delay(2000); // Send every 2 seconds
}
```

Send in UDP

```ino
#include <WiFi.h>
#include <WiFiUdp.h>

const char* ssid = "yourSSID";
const char* password = "yourPASSWORD";
WiFiUDP udp;
unsigned int localPort = 1234; // Listening port

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  udp.begin(localPort);
}

void loop() {
  int packetSize = udp.parsePacket();
  if (packetSize) {
    char incomingPacket[255];
    int len = udp.read(incomingPacket, 255);
    if (len > 0) {
      incomingPacket[len] = '\0';  // Null-terminate
    }

    Serial.printf("Received packet: '%s'\n", incomingPacket);
    Serial.print("From IP: ");
    Serial.println(udp.remoteIP());
    Serial.print("From port: ");
    Serial.println(udp.remotePort());
  }
}
```

Receive in UDP

UDP Broadcast
(Sending Data to All Devices)

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

const char* ssid = "your-SSID";
const char* password = "your-PASSWORD";

WiFiUDP udp;
const int broadcastPort = 4210;  // Port to broadcast on

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

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nConnected to Wi-Fi");
}

void loop() {
  // Get the broadcast IP address
  IPAddress broadcastIP = ~WiFi.subnetMask() | WiFi.localIP();

  // Start a UDP packet to the broadcast address
  udp.beginPacket(broadcastIP, broadcastPort);

  // Send the broadcast message
  udp.write("Hello, all devices!");

  // Finish and send the packet
  udp.endPacket();

  Serial.println("Broadcast packet sent");

  // Wait before sending the next broadcast
  delay(5000);
}
```