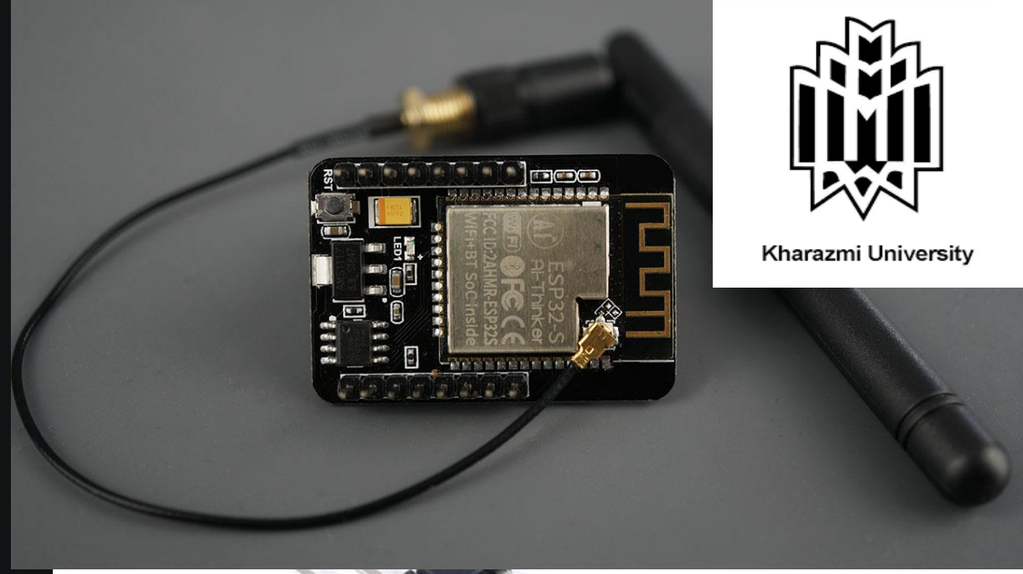
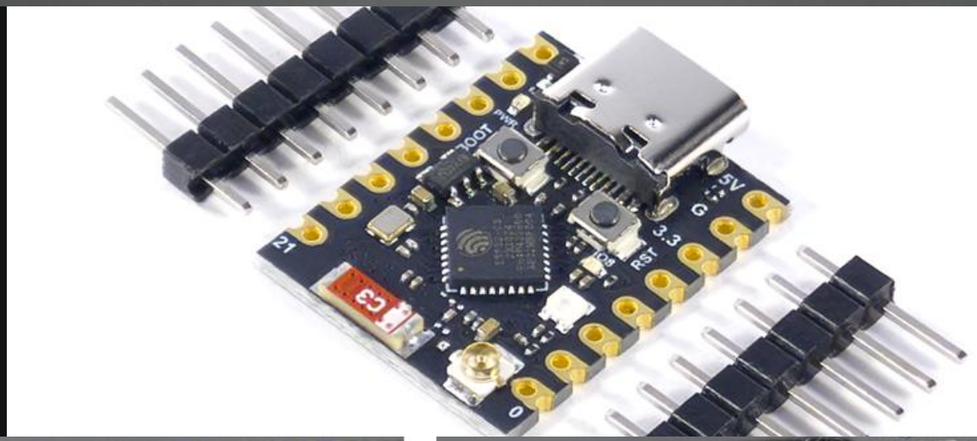
ESP32 WiFi

ESP32 modules provide built-in Wi-Fi connectivity, enabling devices to communicate over networks, act as access points, or form mesh networks for IoT applications.

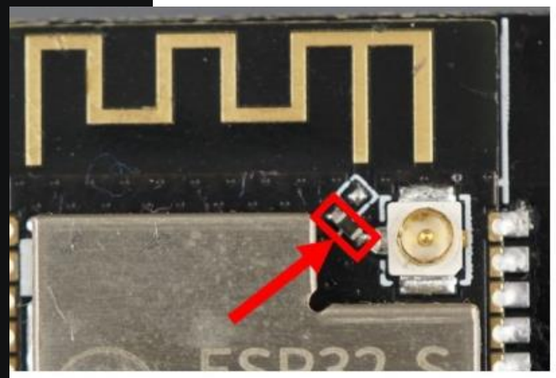


Kharazmi University

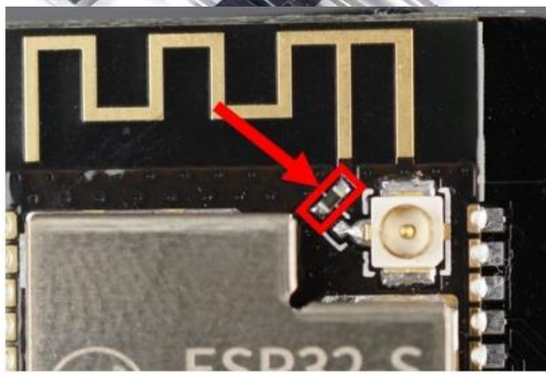


Instructor: Dr. A. Pormand

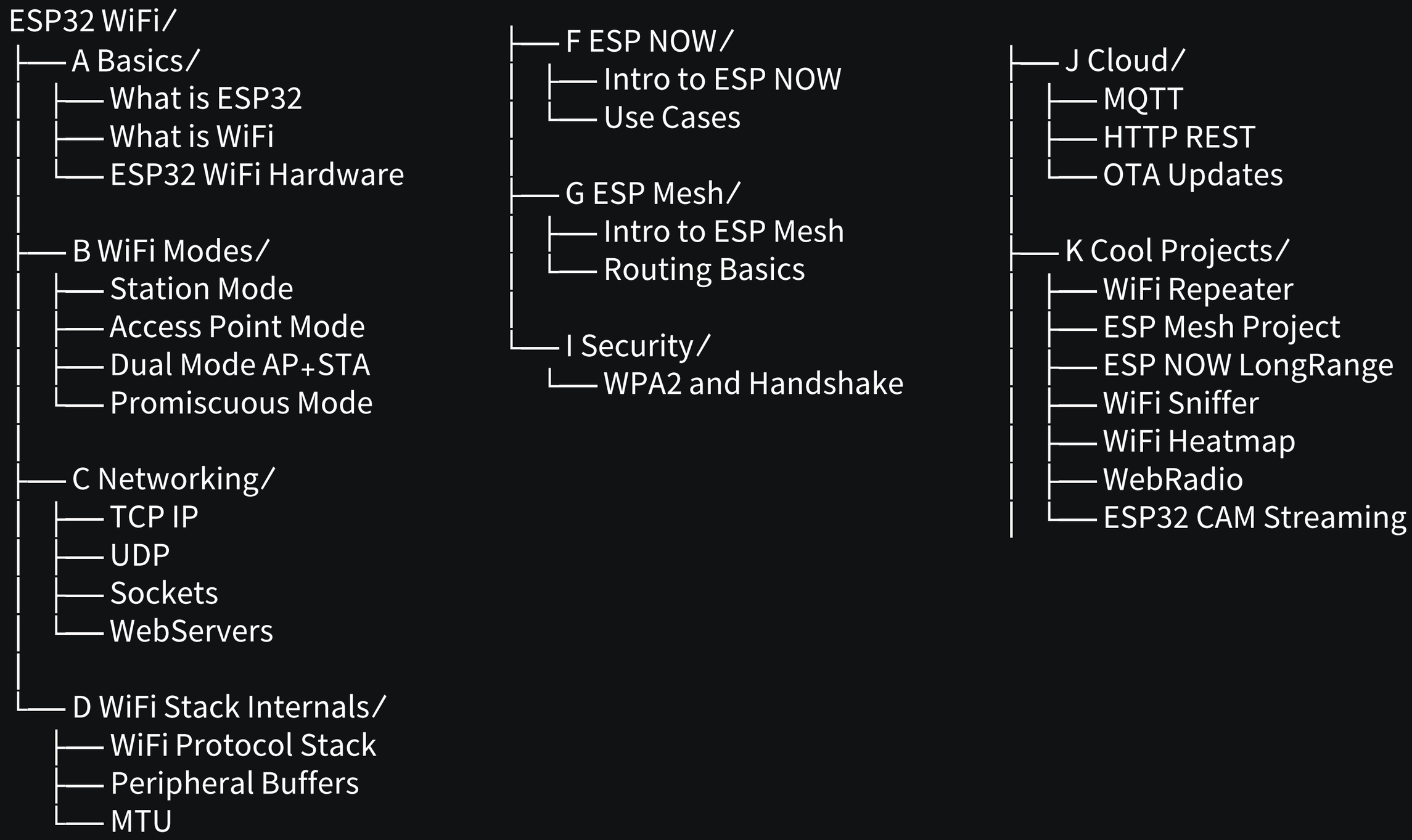
by Mohamad mahdi Latifi



External Antenna



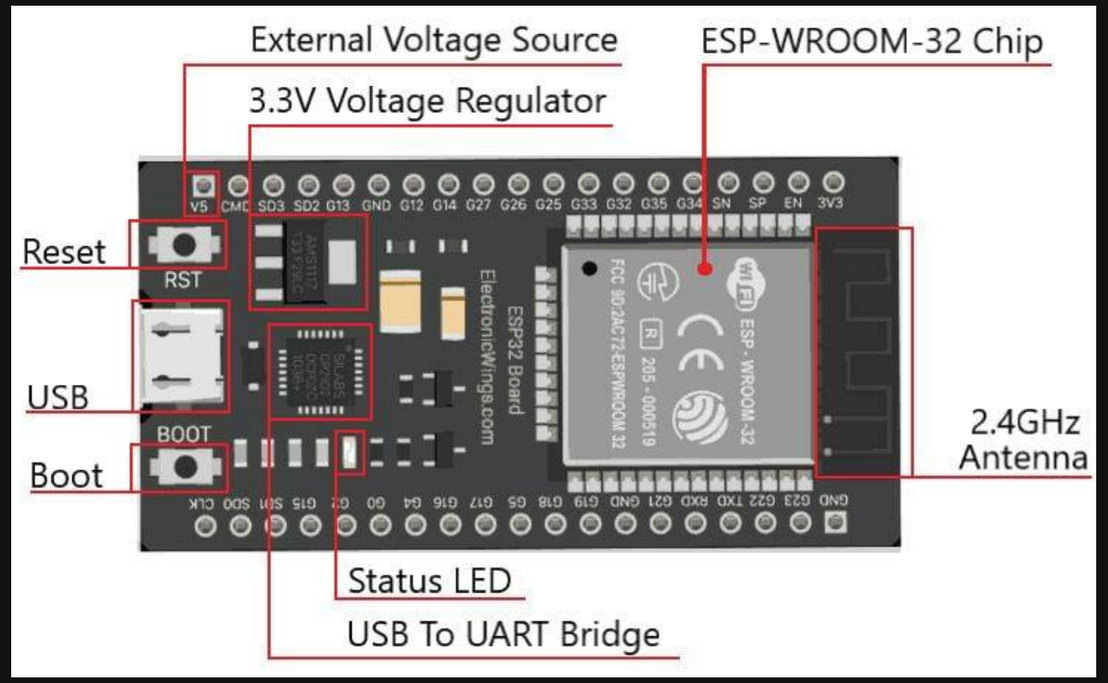
On-board Antenna

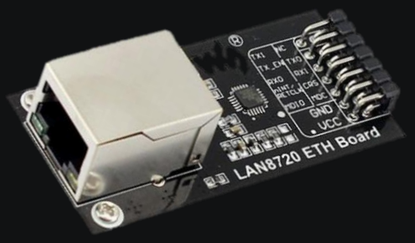

ESP32 WiFi/
A Basics/
What is ESP32
What is WiFi
ESP32 WiFi Hardware
B WiFi Modes/
Station Mode
Access Point Mode
Dual Mode AP+STA
Promiscuous Mode
C Networking/
TCP IP
UDP
Sockets
WebServers
D WiFi Stack Internals/
WiFi Protocol Stack
Peripheral Buffers
MTU
F ESP NOW/
Intro to ESP NOW
Use Cases
G ESP Mesh/
Intro to ESP Mesh
Routing Basics
I Security/
WPA2 and Handshake
J Cloud/
MQTT
HTTP REST
OTA Updates
K Cool Projects/
WiFi Repeater
ESP Mesh Project
ESP NOW LongRange
WiFi Sniffer
WiFi Heatmap
WebRadio
ESP32 CAM Streaming

What is ESP32

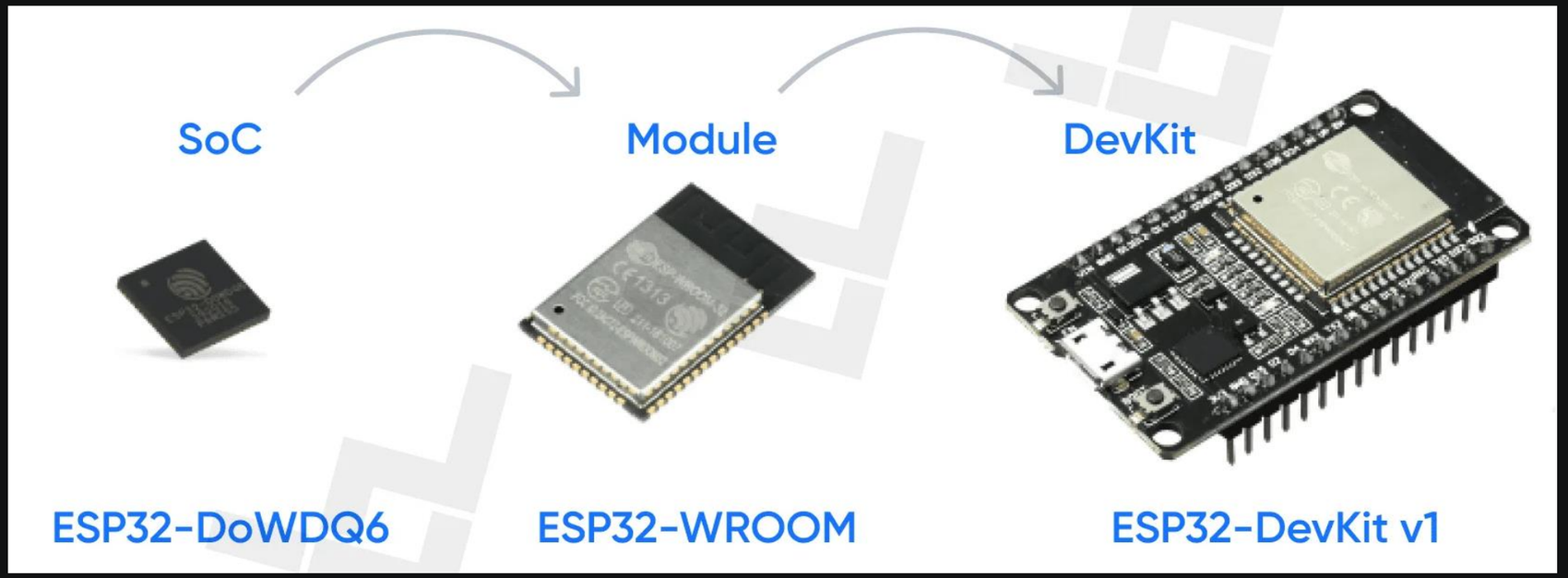
What is ESP32

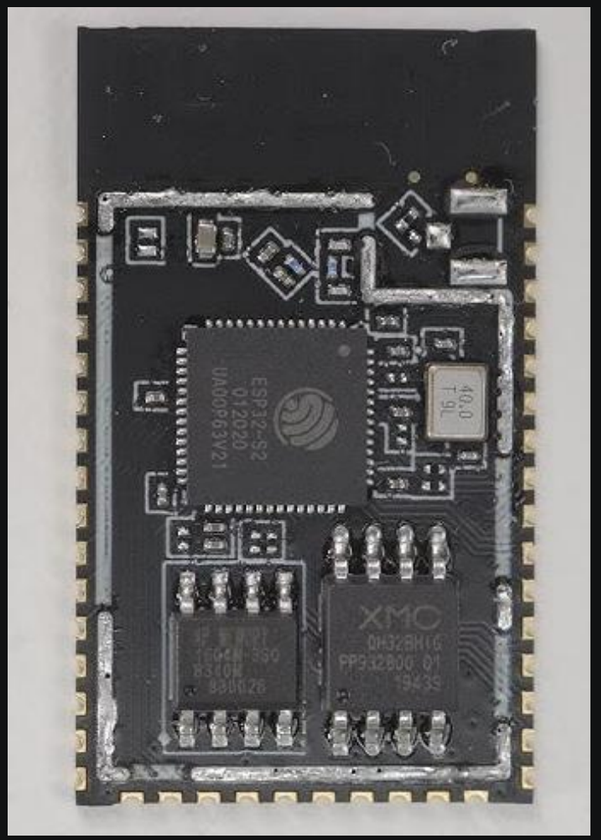
The ESP32 is a powerful, low-cost microcontroller with built-in Wi-Fi and Bluetooth.





External PHY
LAN8720 ETH Board





ESP32-S2

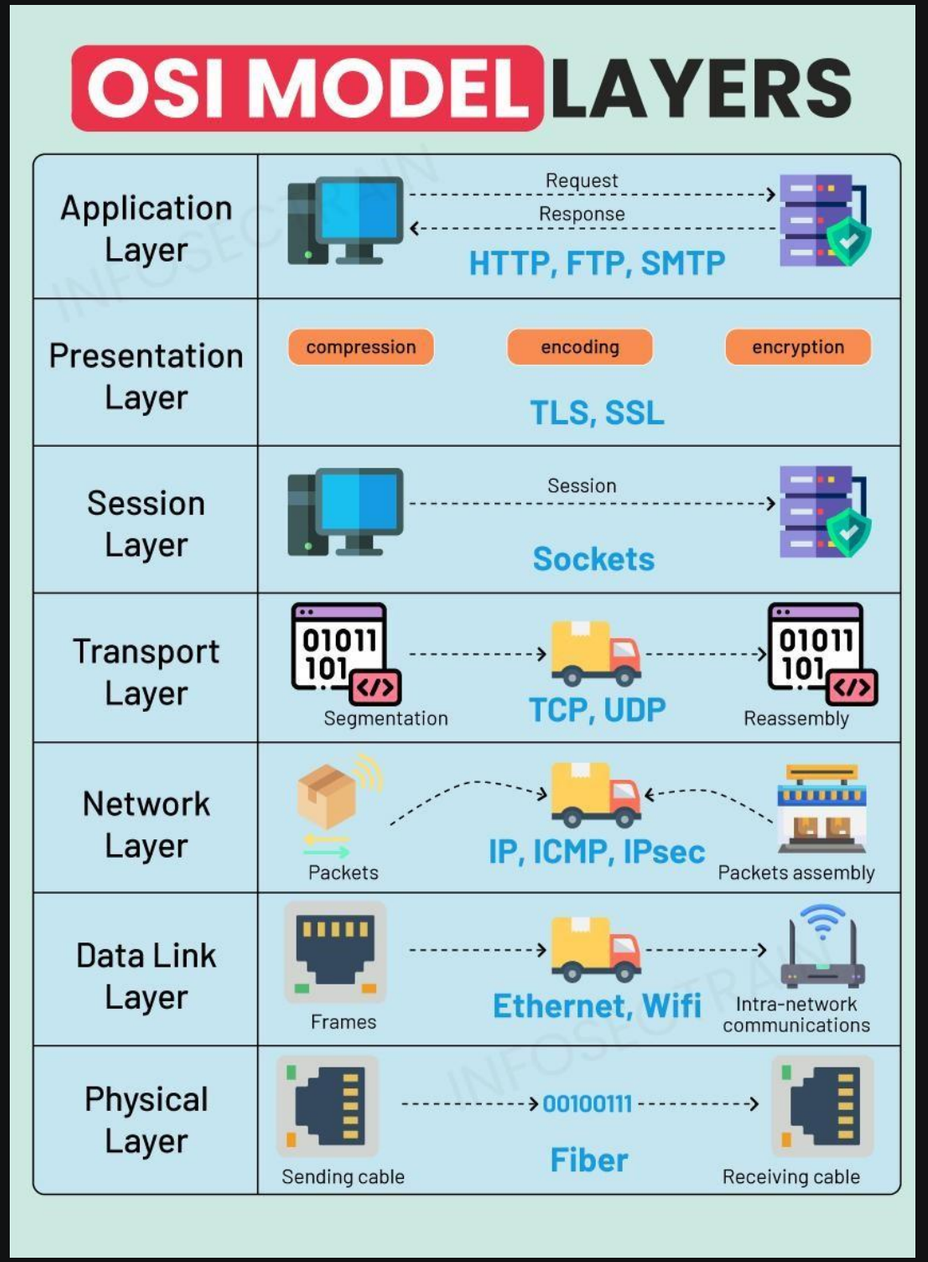
Network Layers

Network Layers

Network layers are a way to organize communication functions in a network. Each layer has a specific role, like sending data, routing, or error checking, to make devices communicate efficiently.

The OSI (Open Systems Interconnection) Model (The 7-Layer Network Stack)

1. **Physical Layer** – The actual signals (radio waves, cables, voltage).
2. **Data Link Layer** – Local network communication (MAC addresses, frames).
3. **Network Layer** – Routing between networks (IP addresses).
4. **Transport Layer** – Reliable connection & data delivery (TCP/UDP).
5. **Session Layer** – Manages sessions between devices.
6. **Presentation Layer** – Data formatting, encoding, encryption.
7. **Application Layer** – What applications use (HTTP, MQTT, browsers, apps).



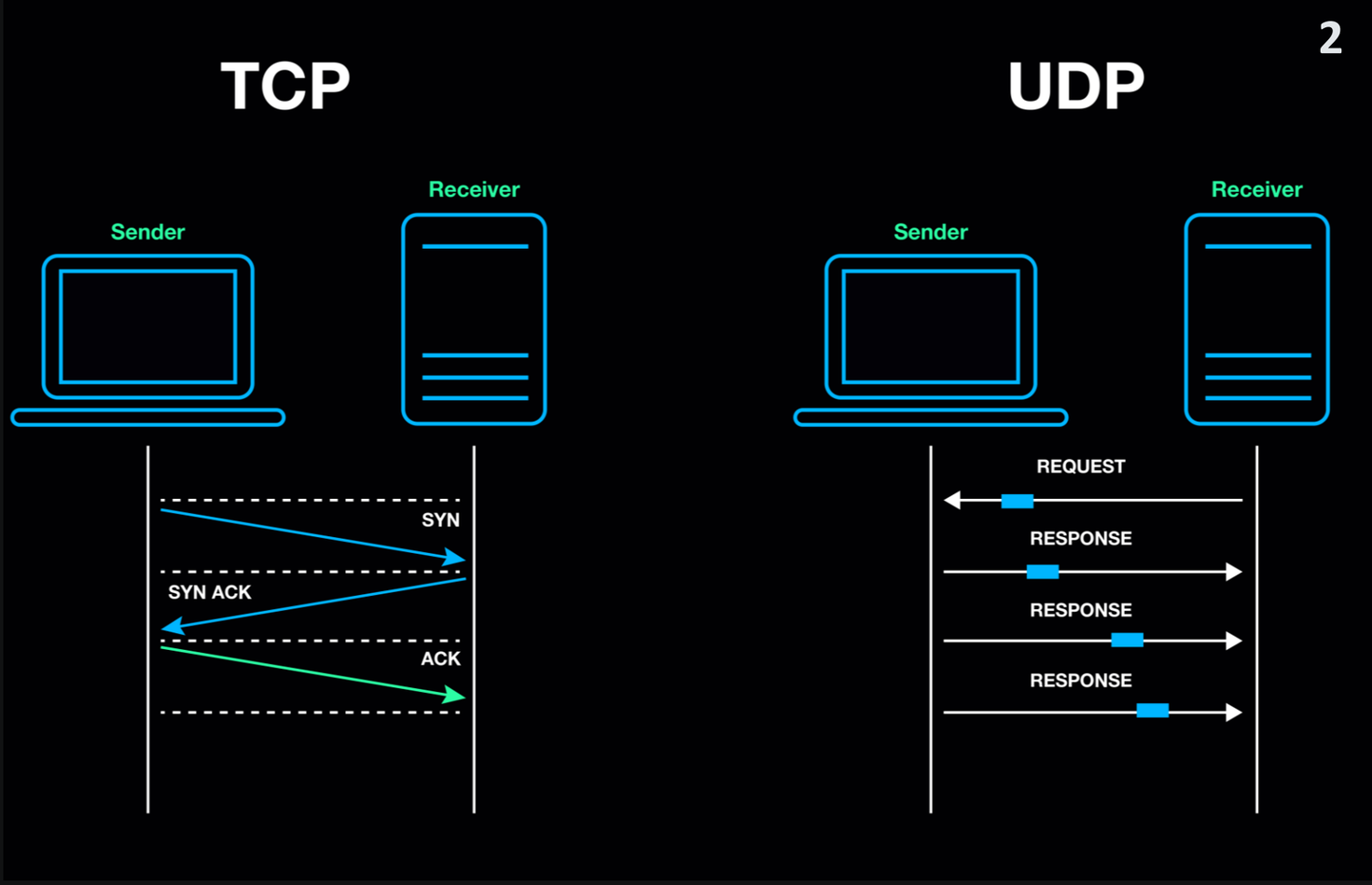
4 - **Transport Layer**

TCP - UDP

TCP (Transmission Control Protocol)
Reliable, ordered, and error-checked communication.

Uses handshakes, acknowledgments, and retransmission.

Used for HTTP, MQTT, HTTPS, and most web traffic.



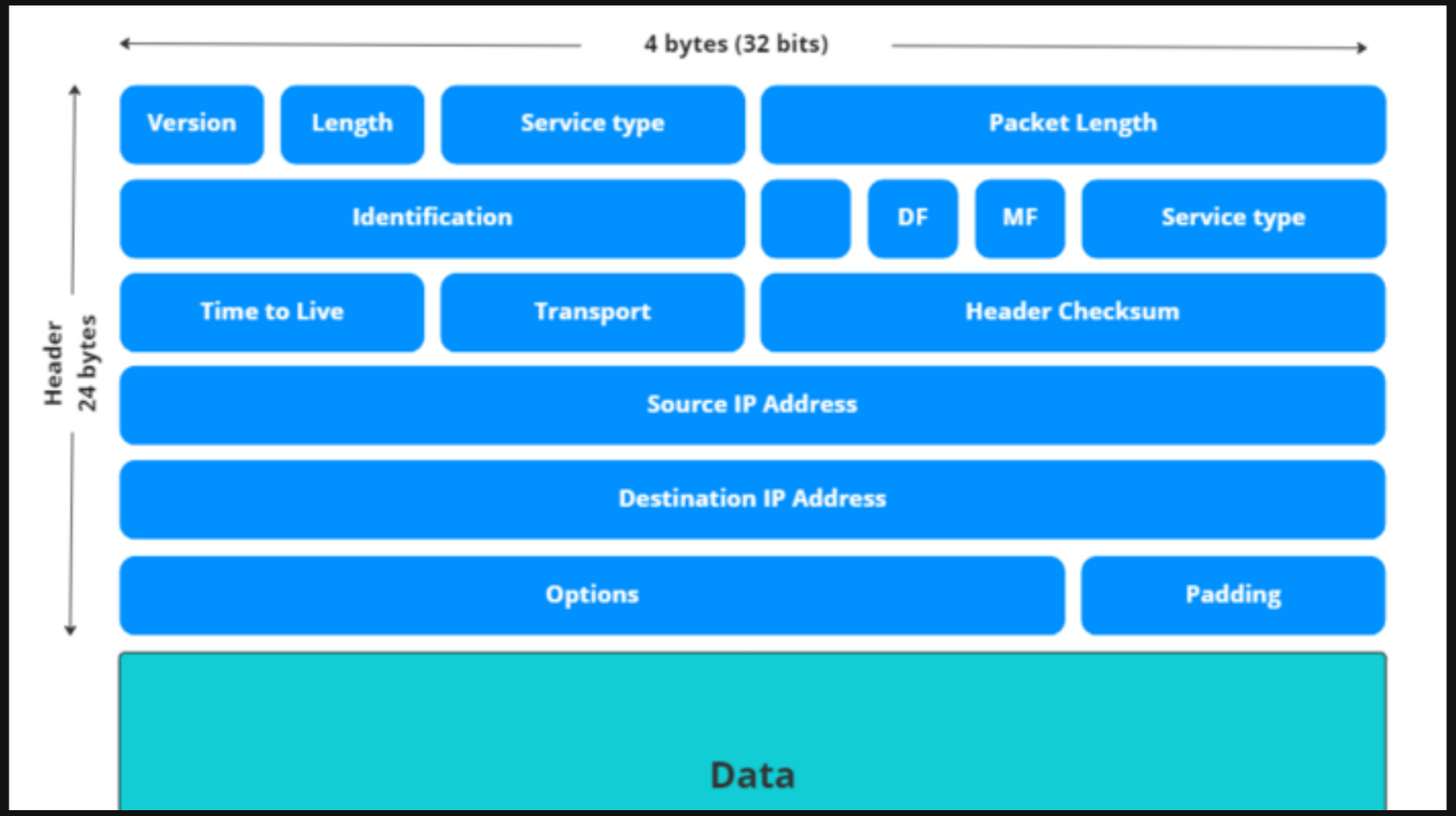
UDP (User Datagram Protocol)

Fast, connectionless, and lightweight.

No handshakes or retransmission — “send and forget.”
Perfect for real-time data, broadcasting, and low-latency IoT.

TCP/IP: The Foundation of Internet Communication

TCP/IP is the core communication model used by WiFi networks. TCP provides reliable, ordered data delivery with retransmission and acknowledgment, while IP handles addressing and routing between devices.

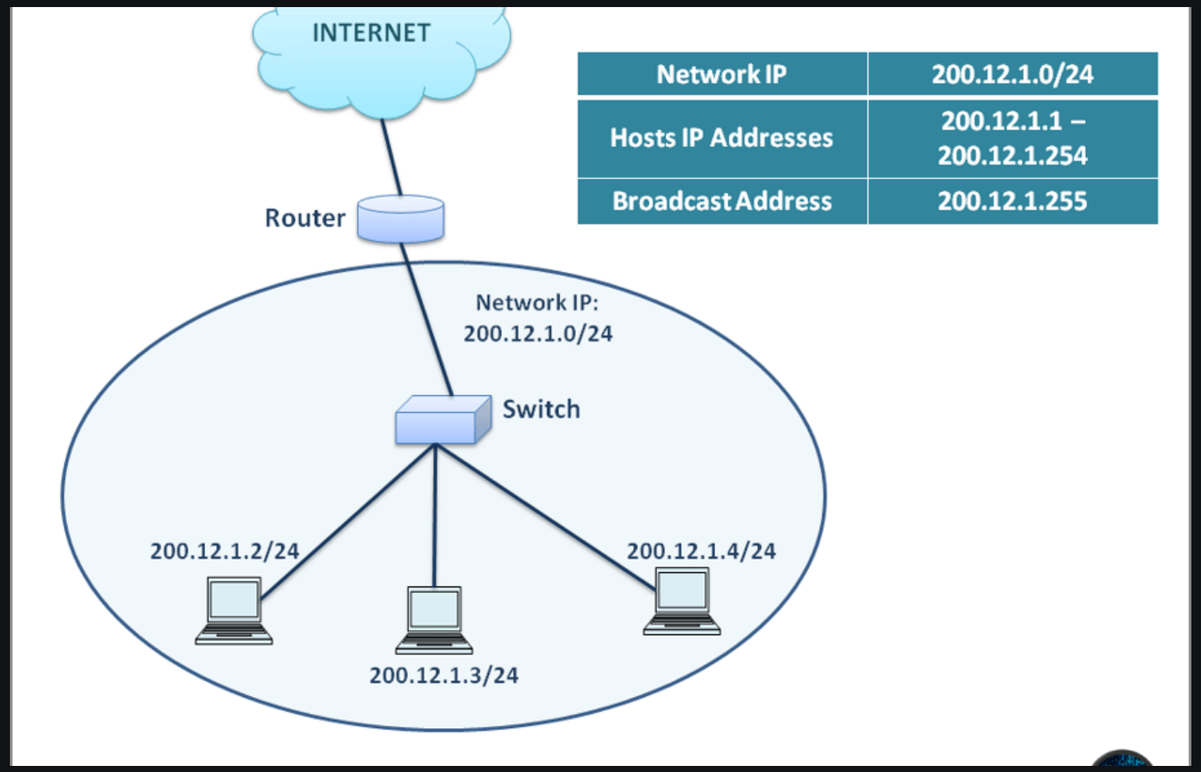


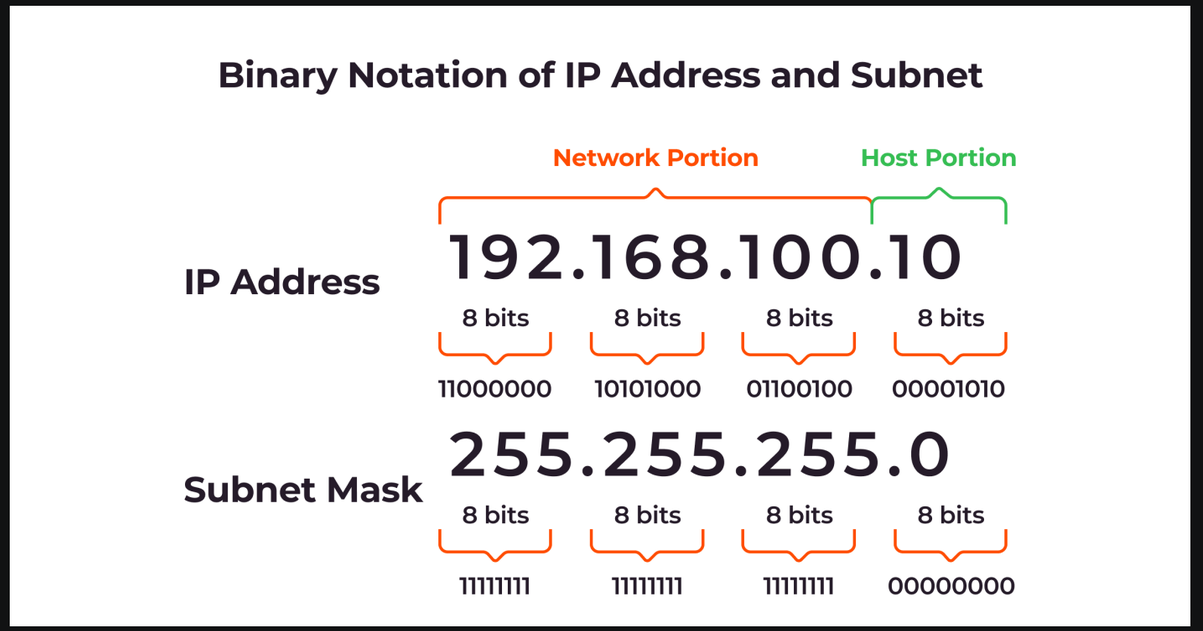
Ethernet Header	IP Header	TCP Header	Data/Payload
Src MAC, Dst MAC	Src IP, Dst IP, TTL, DSCP	Src Port, Dst Port, Seq, ACK	HTTP/MQTT/...

3 - IP Layer (Network Layer)

Responsible for addressing and routing packets across networks.
Adds source and destination IP addresses to each packet.
Determines the best path for data to reach its destination.

IP settings include **IP address**, **subnet mask**, and **default gateway** to define the device's identity and network path.

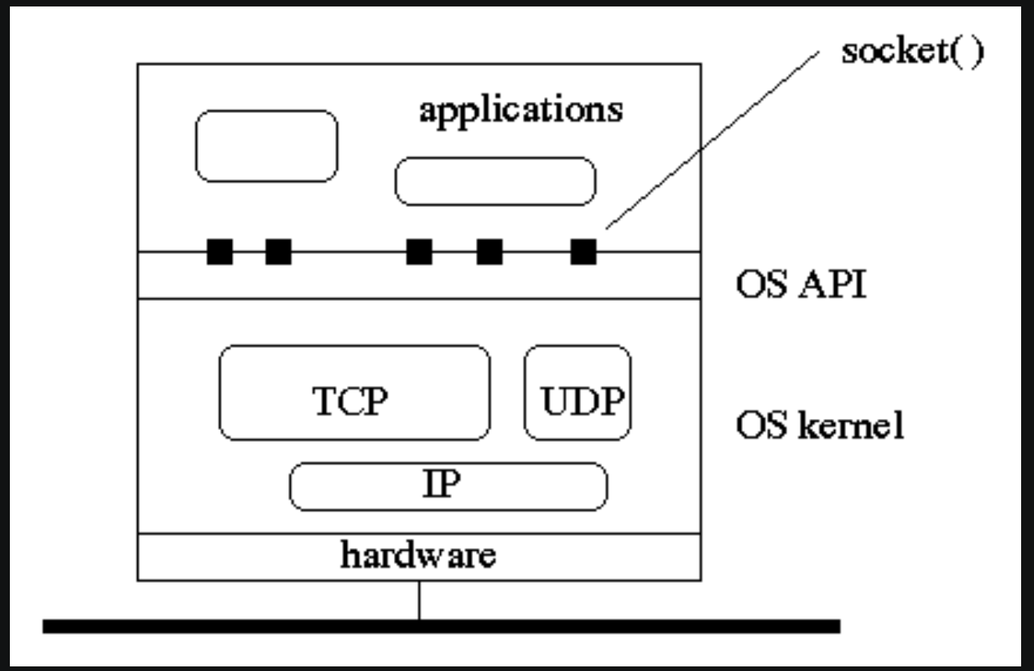


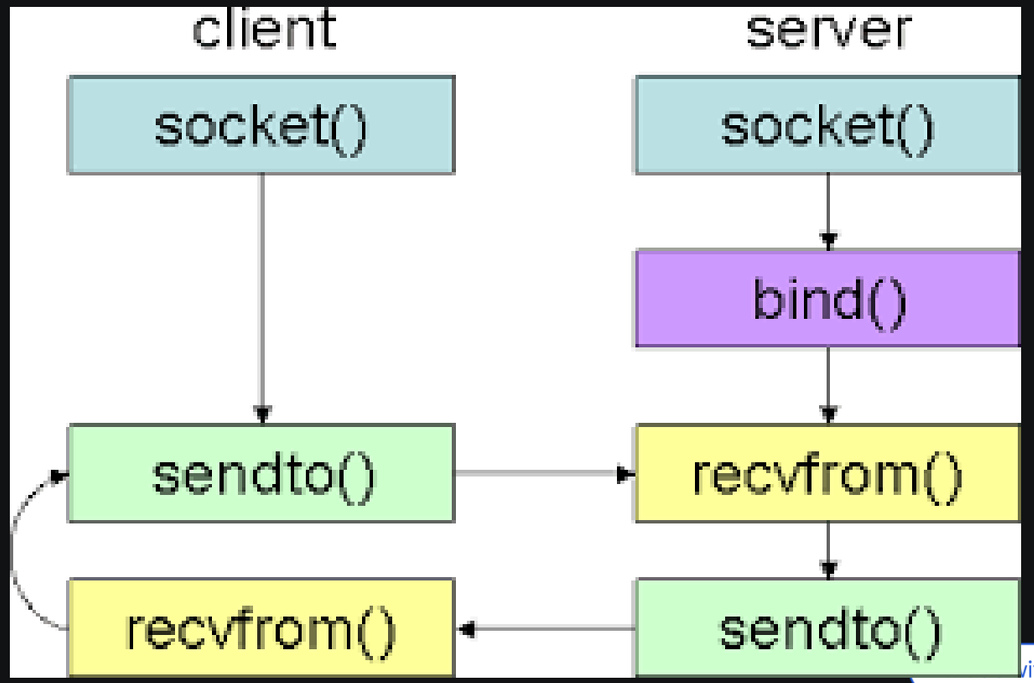


5 - Session Layer

Socket

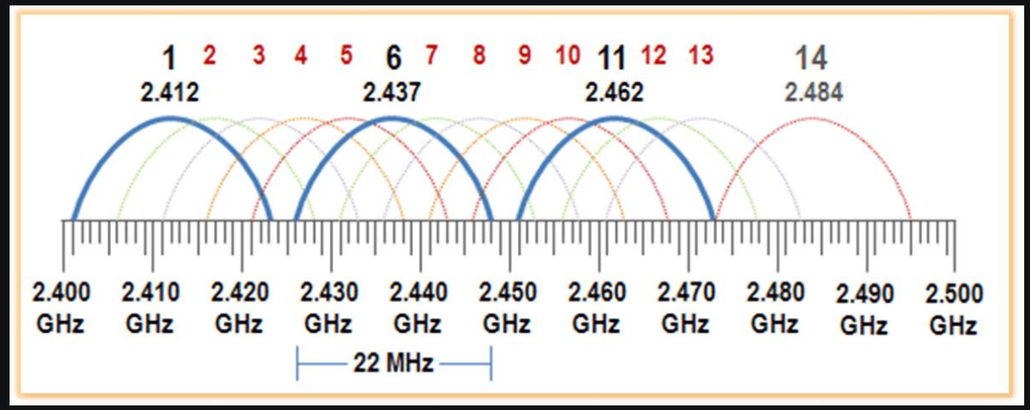
A socket connects your application to the network.
It maps your app’s port to the transport/IP layers.
The OS handles headers, routing, and packet delivery.
Your program just sends and receives data through the socket.





What is WiFi

Wi-Fi stands for Wireless Fidelity. It's a marketing term (not a strict technical acronym) used to describe **wireless local area network (WLAN)** technology based on IEEE 802.11 standards.



Wi-Fi operates in the **bottom two layers**:

◆ **Layer 1—Physical Layer**

- Wi-Fi uses **radio waves** to transmit bits (0s and 1s).
- Uses frequencies like **2.4 GHz** and **5 GHz**.
- Handles modulation, signal strength, channels, antennas.

◆ **Layer 2—Data Link Layer**

- Wi-Fi follows the **IEEE 802.11** standard.
- Responsible for:
  - MAC addresses
  - Wi-Fi frames
  - Authentication & association
  - Managing retransmissions
  - Handling collisions & encryption (WPA2/WPA3)

Wi-Fi = 802.11 protocol at Layer 1 + Layer 2.
IP, TCP, HTTP, etc. sit above Wi-Fi.

Standard	Frequency	Maximum Speed
802.11	2.4 GHz	2 Mbps
802.11a	5 GHz	54 Mbps
802.11b	2.4 GHz	11 Mbps
802.11g	2.4 GHz	54 Mbps
802.11n	2.4 and 5 GHz	600 Mbps
802.11ac	5 GHz	1300 Mbps
802.11ad	2.4 GHz, 5 GHz and 60 GHz	7 Gbps

Wi-Fi Modes

Station Mode (STA)

Connects to an existing Wi-Fi network (like a phone or laptop joining a router).

Access Point Mode (AP)

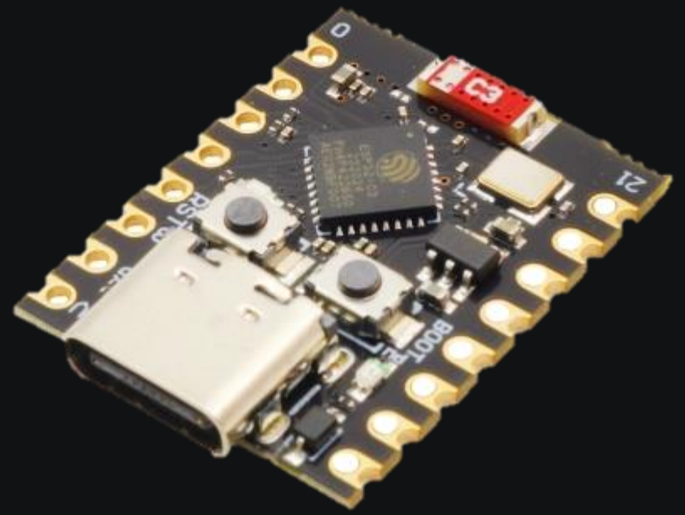
Creates its own Wi-Fi network so other devices can connect to it.

Promiscuous Mode

- Listens to all nearby Wi-Fi packets, not just ones addressed to itself (used for sniffing/monitoring).

Dual Mode (AP + STA)

Works as client and hotspot at the same time – connects to a router and lets others connect to it.



ESP32-C3 Super Mini
eca 247 T

Wi-Fi Modes

Station Mode (STA)

STA_static.ino

```ino
#include <WiFi.h>
const char* STA_SSID = "YourSSID";
const char* STA_PASSWORD = "YourPassword";
// Static IP configuration
IPAddress local_IP(192, 168, 1, 200);    // ESP32 static IP
IPAddress gateway(192, 168, 1, 1);       // Router IP
IPAddress subnet(255, 255, 255, 0);      // Subnet mask
IPAddress primaryDNS(8, 8, 8, 8);        // Primary DNS
IPAddress secondaryDNS(8, 8, 4, 4);      // Secondary DNS (optional)
void setup() {
  Serial.begin(115200);
  delay(1000);

  Serial.println("Connecting to Wi-Fi (Static IP)...");

  WiFi.mode(WIFI_STA);

  // Configure static IP
  if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
    Serial.println("STA Static IP Configuration Failed!");
  }

  // Connect to Wi-Fi
  WiFi.begin(STA_SSID, STA_PASSWORD);

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

STA_dynamic.ino

```ino
#include <WiFi.h>
const char* STA_SSID = "YourSSID";
const char* STA_PASSWORD = "YourPassword";
void setup() {
  Serial.begin(115200);
  delay(1000);
  // --------- Connect to Wi-Fi ---------
  Serial.print("Connecting to Wi-Fi...");

  WiFi.mode(WIFI_STA);
  WiFi.begin(STA_SSID, STA_PASSWORD);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected!");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
  Serial.println("Network Info:");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
  Serial.print("Subnet Mask: ");
  Serial.println(WiFi.subnetMask());
  Serial.print("Gateway: ");
  Serial.println(WiFi.gatewayIP());
  Serial.print("Primary DNS: ");
  Serial.println(WiFi.dnsIP());
}
void loop() {
  // nothing here
```

Wi-Fi Modes

- Access Point Mode (AP)

AP.ino

```ino
#include <WiFi.h>

// AP credentials
const char* AP_SSID = "ESP32_AP_Test";
const char* AP_PASSWORD = "12345678";

// AP configuration
IPAddress apIP(192, 168, 10, 1);          // ESP32 IP
IPAddress subnet(255, 255, 255, 0);       // Subnet mask
int channel = 6;                          // Wi-Fi channel (1-13)

// DHCP range (custom)
IPAddress dhcpStart(192, 168, 10, 10);

void setup() {
  Serial.begin(115200);
  delay(1000);
  Serial.println("Starting ESP32 in AP mode...");

  WiFi.mode(WIFI_AP);

  // Set AP config (IP + subnet)
  if (!WiFi.softAPConfig(apIP, apIP, subnet)) {
    Serial.println("AP IP configuration failed!");
  }

  // Start AP     (ssid, password, channel, hidden, max_connection);
  if (!WiFi.softAP(AP_SSID, AP_PASSWORD, channel, false, 4)) {
    Serial.println("Failed to start AP!");
    return;
  }
```

Wi-Fi Modes

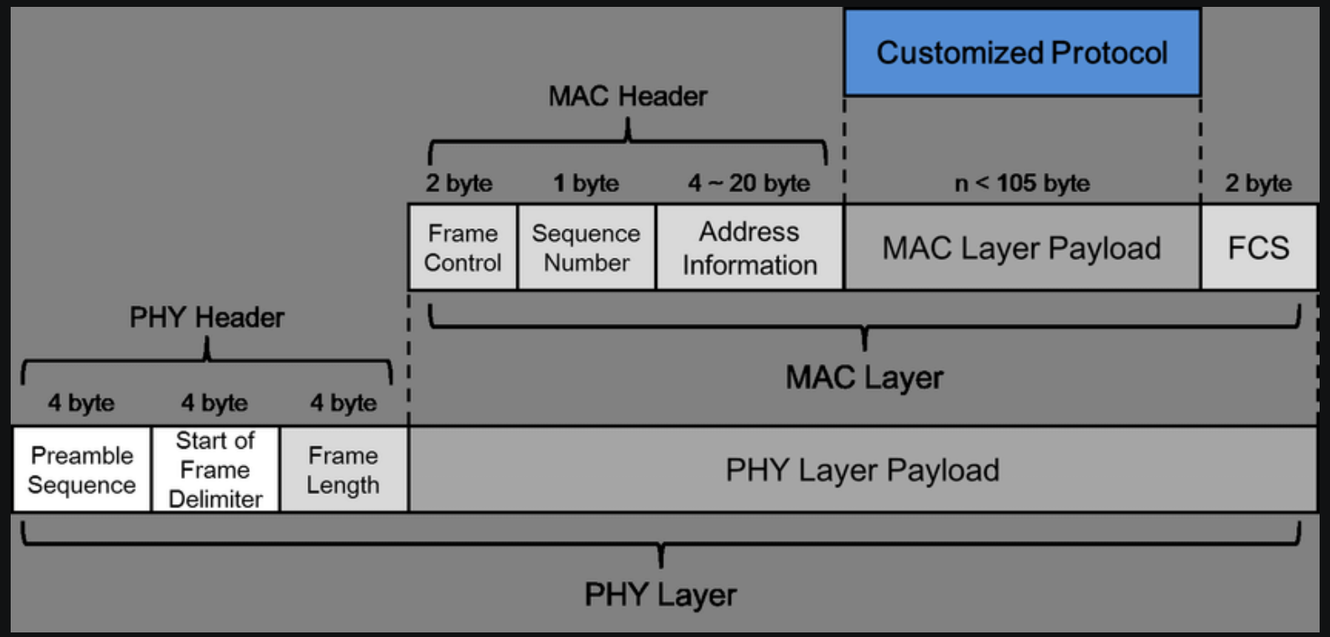
- Dual Mode (AP + STA)

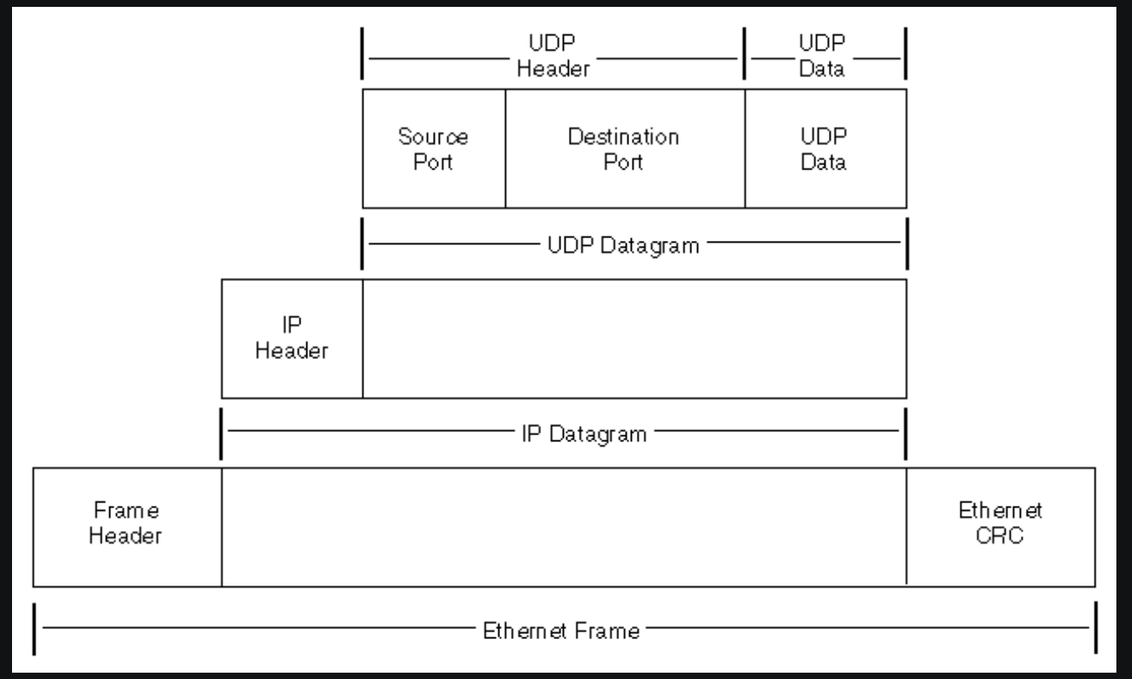
```ino
#include <WiFi.h>

// --- STA (connect to your router)
const char* sta_ssid = "HomeWiFi";
const char* sta_pass = "HomePass";

// --- AP (ESP32 creates hotspot)
const char* ap_ssid  = "ESP32_AP";
const char* ap_pass  = "12345678";

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

  // Dual mode
  WiFi.mode(WIFI_AP_STA);

  // Start Station
  WiFi.begin(sta_ssid, sta_pass);

  // Start Access Point
  WiFi.softAP(ap_ssid, ap_pass);

  Serial.println("Dual WiFi Mode Started");
  Serial.print("STA IP: "); Serial.println(WWiFi.localIP());
  Serial.print("AP  IP: "); Serial.println(WiFi.softAPIP());
}

void loop() {
}
```

Wi-Fi Modes

Promiscuous Mode

captures Wi-Fi **Layer 2 frames (MAC level)**, not higher-layer IP/TCP/HTTP data unless you manually decode payloads.





Wi-Fi Modes

Promiscuous Mode

treating captured HTTP data like a DNA-style byte sequence for pattern detection.

```
#include "WiFi.h"
#include "esp_wifi.h"
int channelToSniff = 6;   // <-- REAL WiFi channel to sniff
void snifferPacketHandler(void* buf, wifi_promiscuous_pkt_type_t type) {
  wifi_promiscuous_pkt_t *pkt = (wifi_promiscuous_pkt_t*)buf;
  uint8_t *data = pkt->payload;
  uint16_t len = pkt->rx_ctrl.sig_len;

  // Send start marker + length
  Serial.write(0xFE);
  Serial.write((uint16_t)len & 0xFF);
  Serial.write((uint16_t)len >> 8);
  // Send raw packet bytes
  Serial.write(data, len);
}
void setup() {
  Serial.begin(115200);
  delay(500);
  WiFi.mode(WIFI_MODE_NULL);            // Disable everything
  esp_wifi_set_promiscuous(true);
  esp_wifi_set_promiscuous_rx_cb(snifferPacketHandler);
  wifi_promiscuous_filter_t filt = {
    .filter_mask = WIFI_PROMIS_FILTER_MASK_ALL  // capture ALL types of frames
  };
  esp_wifi_set_promiscuous_filter(&filt);
  esp_wifi_set_channel(channelToSniff, WIFI_SECOND_CHAN_NONE);
  Serial.println("Sniffer started.");
}
void loop() {
  // nothing here – packets handled by callback
}
```

UDP from esp to python

Wireless Oscilloscope

ESP32 reads analog values and streams them over UDP Wi-Fi. Python receives the data and plots it live as a simple wireless oscilloscope.

```python
import socket
import matplotlib.pyplot as plt
from collections import deque

UDP_IP = "0.0.0.0"
UDP_PORT = 5005

sock = socket.socket(socket.AF_INET, socket.SOCK_DG
sock.bind((UDP_IP, UDP_PORT))

# Plot buffer
buf = deque(maxlen=200)

plt.ion()
fig, ax = plt.subplots()
line, = ax.plot([], [])
ax.set_ylim(0, 3.3)     # ESP32 voltage range
ax.set_xlim(0, 200)

while True:
    data, _ = sock.recvfrom(1024)
    v = float(data.decode())
    buf.append(v)

    line.set_xdata(range(len(buf)))
    line.set_ydata(list(buf))

    ax.relim()
    ax.autoscale_view()

    plt.pause(0.001)

```

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

const char* ssid     = "2.";
const char* password = "12345678an";

IPAddress udpAddress(192,168,1,50);   // PC IP
const int udpPort = 5005;

WiFiUDP udp;
int adcPin = 34;   // Analog pin

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

  while (WiFi.status() != WL_CONNECTED) delay(200);

  udp.begin(udpPort);
}

void loop() {
  int raw = analogRead(adcPin);
  float voltage = raw * (3.3 / 4095.0);   // Convert

  char msg[32];
  sprintf(msg, "%.3f", voltage);

  udp.beginPacket(udpAddress, udpPort);
  udp.print(msg);
  udp.endPacket();

  delay(20);    // ~50 samples/sec
}

```

Receive UDP in ESP32

Reads all incoming bytes from Python,controls the built-in LED based on the first byte,and prints the first 10 bytes as ASCII for quick inspection.

MTU (Maximum Transmission Unit) sits at the **Data Link layer** and defines the largest frame payload.
If an IP packet is bigger, Layer 2 **fragments it** for transmission over the network.

```python
import socket
import random

UDP_IP = "192.168.1.100"  # ESP32 IP
UDP_PORT = 5005

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Example: send 1000 bytes in one go
data_bytes = [random.randint(0, 255) for _ in range(1000)]
sock.sendto(bytes(data_bytes), (UDP_IP, UDP_PORT))
```

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

const char* ssid     = "2.";
const char* password = "12345678an";

const int udpPort = 5005;
WiFiUDP udp;
int ledPin = 2;
const int BUF_SIZE = 2048; // enough for 1 big packet
uint8_t buf[BUF_SIZE];

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

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) { delay(200); Serial.print("."); }
  Serial.println("\nWi-Fi connected.");
  udp.begin(udpPort);
  Serial.println("UDP listening...");
}

void loop() {
  int packetSize = udp.parsePacket();
  if (packetSize > 0) {
    int len = udp.read(buf, BUF_SIZE);
    if (len <= 0) return;

    // LED based on first byte
    digitalWrite(ledPin, (buf[0] % 2 == 0) ? HIGH : LOW);

    // Print first 10 bytes as ASCII
    int toPrint = (len < 10) ? len : 10;
    Serial.print("First bytes ASCII: ");
    for (int i = 0; i < toPrint; i++) {
      Serial.print((char)buf[i]); Serial.print(' ');
    }
    Serial.println();
  }
}
```

ESP-NOW

ESP-NOW operates **at Layer 2 (Data Link / MAC layer)** of the OSI model:
- Uses the **Wi-Fi radio directly** without a full TCP/IP stack.
- Communicates via **MAC addresses**, not IP addresses.
- Supports **peer-to-peer or broadcast** messaging with very low latency.
- Ideal for **sensor networks, IoT, and fast device-to-device communication**.

ESP32 #1 reads analog values and broadcasts them via ESP-NOW.ESP32 #2 receives the data and prints it to Serial for real-time plotting.

Cool Projects

ESP32 NAT Router with WPA2 Enterprise support

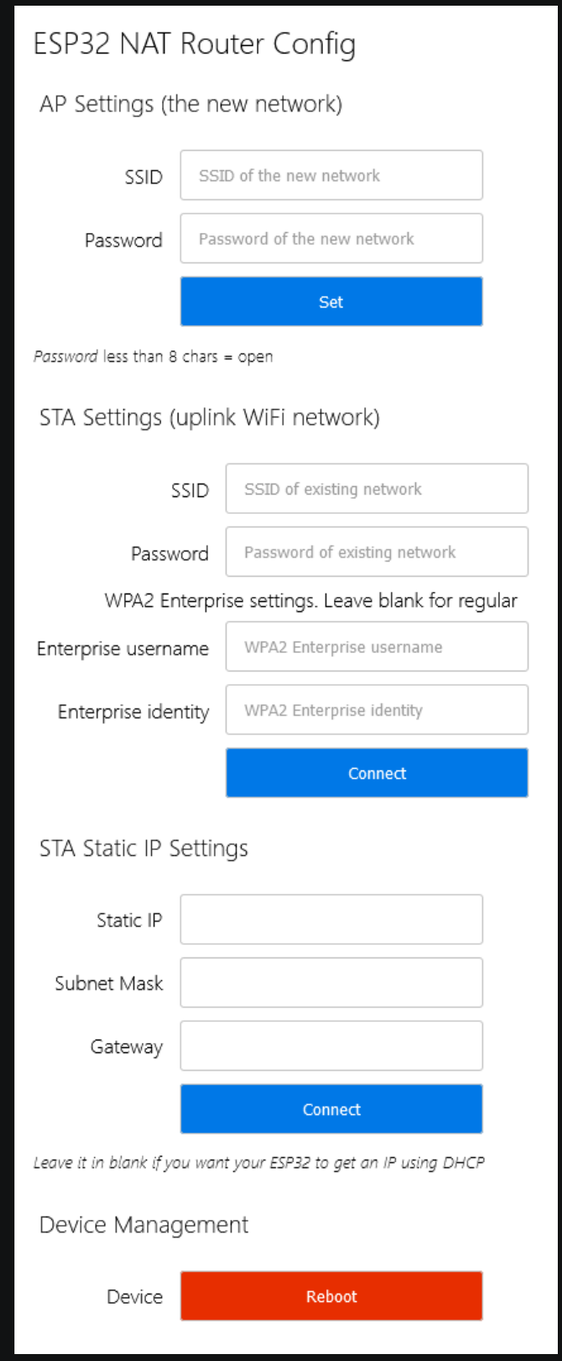
Simple range extender for an existing WiFi network

Mesh protocol built on Wi-Fi

nodes connect to each other, not all to the routerOne root node connects to the normal Wi-Fi router and bridges to the internetOther nodes form a tree (parent/child) and relay packets (multi-hop) to extend rangeWith IP enabled, all nodes can use normal TCP/UDP

painless Mesh

Runs on ESP8266 & ESP32 using Arduino
Self-forming, self-healing mesh (nodes auto-join and re-route)
No master – all nodes are equal and can reach each other via multi-hop
Supports broadcast and single-node messages
Messages sent as easy-to-use JSON strings
Built-in time synchronization across all nodes
Mesh topology auto-updates, so nodes know the network layout
Asynchronous callbacks for received messages (no blocking loops)



Sensors in Pulse Oximeters

Light Source

mploys red and infrared LEDs
r wavelength-specific
bsorption.

Light Detectors

- Photodiodes: High sensitivity, fast response
- Phototransistors: Alternative with good sensitivity
- LDRs: Rare, slower and less sensitive

Advantages

Fast, low noise output enables precise measurement.

ESP32 Technical Reference Manual

Chatgpt

TNX