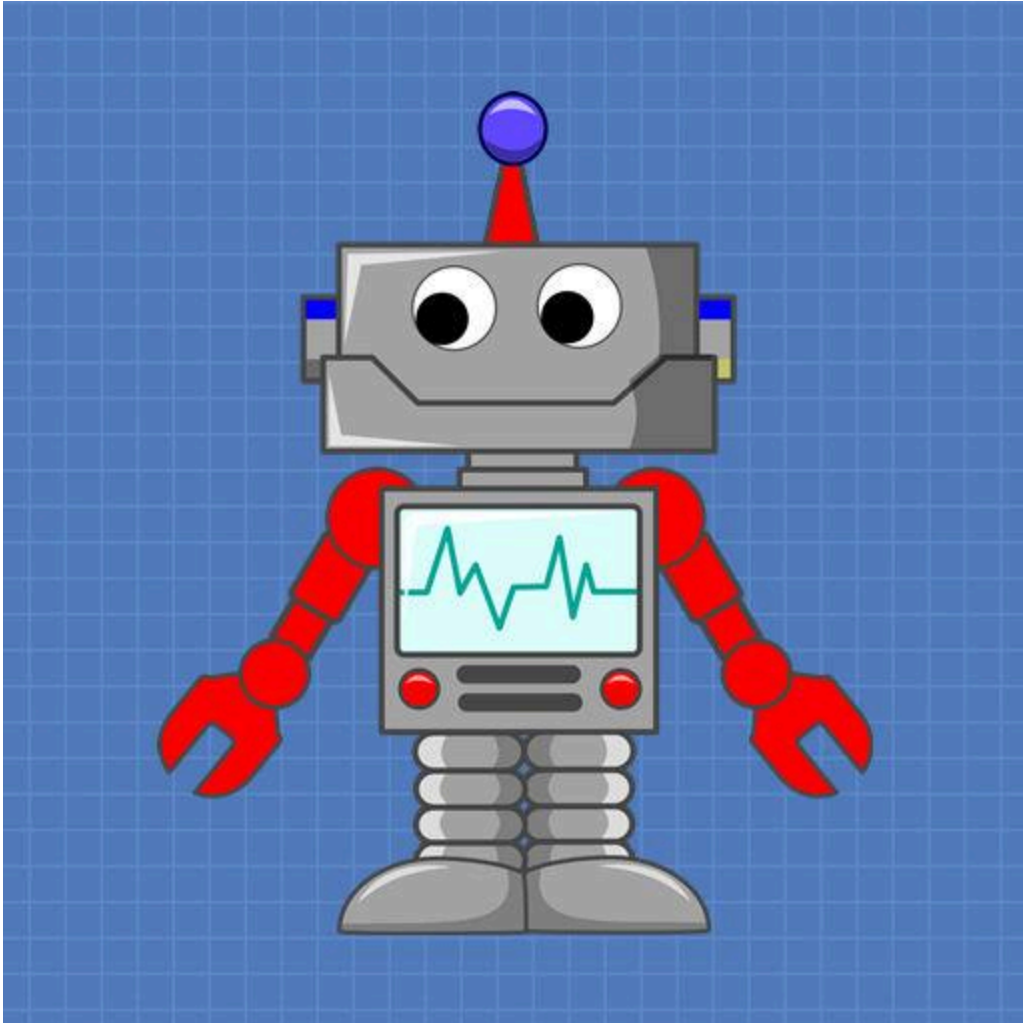


Using Ethernet with ESP32

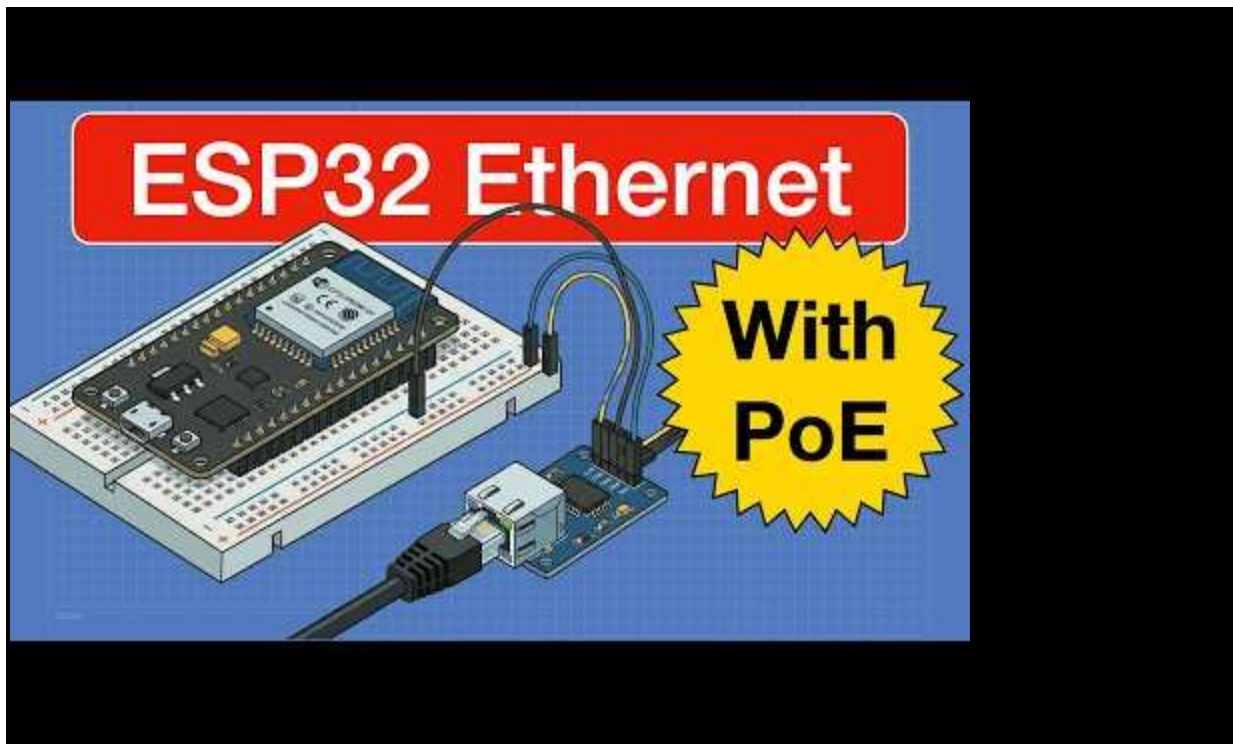


DroneBot Workshop Tutorial

<https://dronebotworkshop.com>

Wired networking, PoE, and a self-powered web camera

Today, we are going to connect an ESP32 to our network via Ethernet rather than Wi-Fi. We'll learn how Ethernet works, then do a few simple experiments to test it. Along the way, we'll put together a simple web server and a web camera powered by the Ethernet connection via PoE.

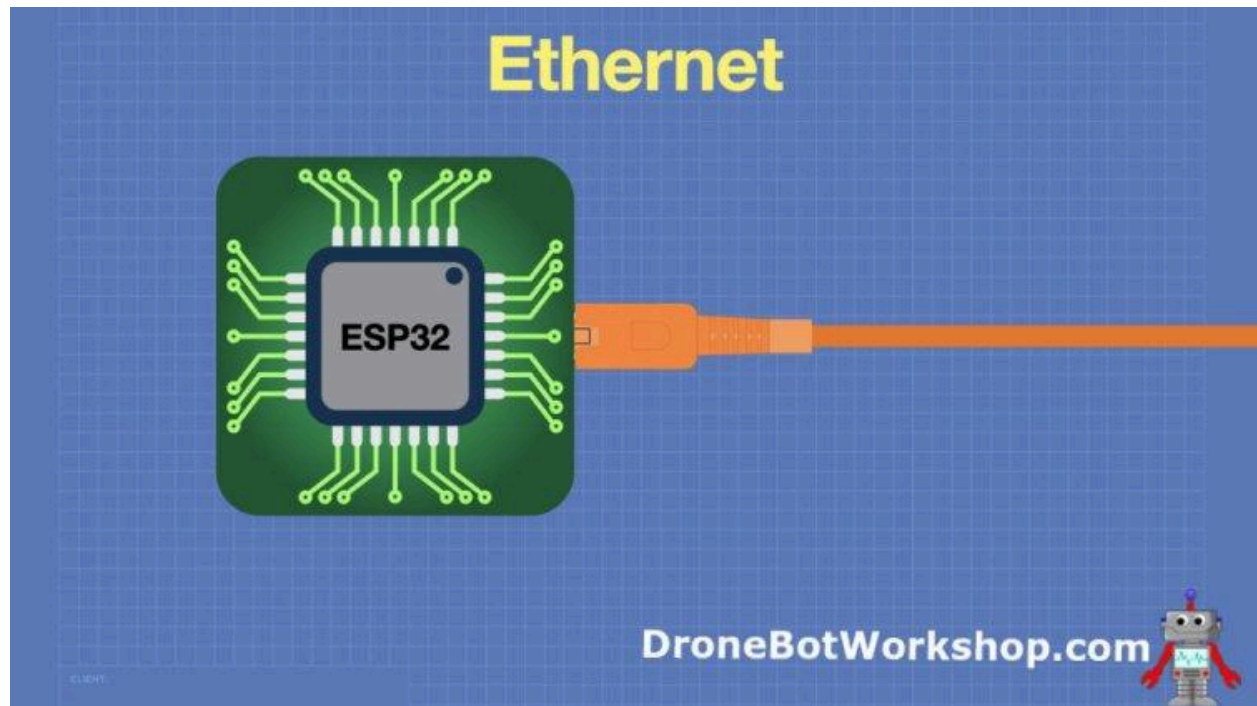


Introduction

Most ESP32 boards already include built-in Wi-Fi, which makes adding a wired Ethernet connection, at first glance, unnecessary. So why drag a cable across the workbench when wireless is right there for the taking?

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

It turns out there are several good reasons. Ethernet is faster, more secure, and far more reliable than Wi-Fi. It is not subject to the same kind of interference that plagues 2.4 GHz radio, and it does not slow down when a neighbor fires up a microwave oven. On top of all that, Ethernet offers a trick that Wi-Fi cannot match: **Power over Ethernet (PoE)**, which delivers both network data and electrical power to a device over a single cable.



This article covers all of it. It begins with a quick tour of Ethernet – where it came from, how it works, and how to choose the right cable. Then it walks through connecting an ESP32 to Ethernet two different ways: first using the inexpensive and very common W5500 adapter, and then using a purpose-built Ethernet-and-PoE ESP32 board from Waveshare. We'll finish up by building a Power-over-Ethernet web camera that needs only a single Ethernet cable for both power and communications.

<https://dronebotworkshop.com>

What You'll Need

Here is a quick rundown of the hardware featured in this article:

- **An ESP32-family microcontroller** – a Seeed Studio XIAO ESP32-S3 is used for the W5500 demos.
- **A WIZnet W5500 Ethernet adapter** – the compact “Lite” version is used here, but the full-size variant with the built-in voltage regulator works identically.
- **A Waveshare ESP32-S3-POE-ETH** – A development board, you'll also need the optional PoE module and OV5640 camera.
- **A PoE power injector** (or a PoE-capable Ethernet switch).
- A couple of ordinary **Cat 5e or Cat 6 Ethernet patch cables**.

Wi-Fi vs. Ethernet at a Glance

Before getting into the hardware, here is how wired Ethernet stacks up against Wi-Fi for the kind of projects a reader of the DroneBot Workshop builds.

Property	Ethernet	Wi-Fi
Maximum speed	10 Gbps (Cat 6A) and beyond	Depends on standard and conditions
Latency	Sub-millisecond, very consistent	Variable; jitter under load
Reliability	Excellent – physical connection	Affected by distance, walls, interference

Security	Must be physically tapped	Radio waves travel through walls
Power delivery	Yes – via PoE	No
Duplex	Full duplex on switched networks	Half duplex (shared channel)
Added users	No bandwidth loss per port	Shared airtime; degrades with users
Mobility	Tethered by a cable	Fully mobile

Ethernet is the right choice when a device sits in a fixed location, needs a rock-solid connection, and would benefit from PoE. Wi-Fi wins when the device has to move, or when pulling a cable to it is impractical. For many ESP32 projects – smart-home sensors in a wiring closet, security cameras on the side of a building, industrial controllers – Ethernet is not only a viable option but the better one.

Ethernet

Ethernet is by far the most common way to connect computers together on a local area network, and it has been around for more than half a century. A little bit of its history, and an understanding of how it does what it does, will make everything that follows much easier.

A Brief History of Ethernet

The Ethernet networking system was invented at Xerox's famous Palo Alto Research Center in May of 1973. It was created by Robert Metcalfe and David Boggs, originally as a way to connect the computers scattered throughout Xerox PARC. The name "Ethernet" comes from "ether," a hypothetical substance that physicists of an earlier era imagined filled the void between the stars – an appropriately poetic name for a medium that would one day carry signals all over the world.

That very first implementation of Ethernet ran at 2.94 Mbps over a thick, rigid coaxial cable that was physically tapped at the points where workstations attached to the network. This cable – nicknamed "Thicknet" – was later standardised as 10BASE5. A thinner coaxial variant called "Thinnet" (10BASE2) soon followed, and both ran at 10 Mbps.

In 1980, Xerox teamed up with Intel and Digital Equipment Corporation (DEC) to publish the first open Ethernet specification, and in 1982 the IEEE formally adopted the standard as IEEE 802.3. That adoption turned Ethernet from a single-company project into an industry-wide specification, and set it up to dominate local networking for the next forty years.

The real turning point for Ethernet came in 1990, when the coaxial cables of earlier decades were replaced by unshielded twisted-pair copper wire – the familiar eight-wire cable terminated in the RJ45-style connector still in use today. This is the cable used for the rest of this article. (Ethernet also runs over fibre optic cable, but fibre is used mostly in data centres and over long distances and is outside the scope of this article.)

From 1990 onward, speeds climbed steadily. Fast Ethernet (100BASE-TX) arrived in 1995, Gigabit Ethernet (1000BASE-T) in 1999, and 10 Gigabit Ethernet followed in the

mid-2000s. Today's fastest copper Ethernet standards reach 40 Gbps in data centres, and fibre-optic implementations run all the way up to 400 Gbps. For ESP32 projects, plain 10/100 Mbps Ethernet is more than fast enough for sensors, controllers, and even

Year	Milestone	Notes
1973	Ethernet invented at Xerox PARC	Metcalfe & Boggs; 2.94 Mbps on coax
1980	Xerox / Intel / DEC publish "DIX" standard	10 Mbps baseline
1982	IEEE 802.3 formally adopts Ethernet	Thicknet (10BASE5) and Thinnet (10BASE2)
1990	10BASE-T – Ethernet over twisted pair	IEEE 802.3i – RJ45 as we know it
1995	100BASE-TX – Fast Ethernet	IEEE 802.3u
1999	1000BASE-T – Gigabit Ethernet	IEEE 802.3ab; uses all 4 pairs
2006	10GBASE-T – 10 Gigabit over copper	IEEE 802.3an
Today	25G, 40G, 100G, 400G in the data centre	Primarily fibre-optic

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

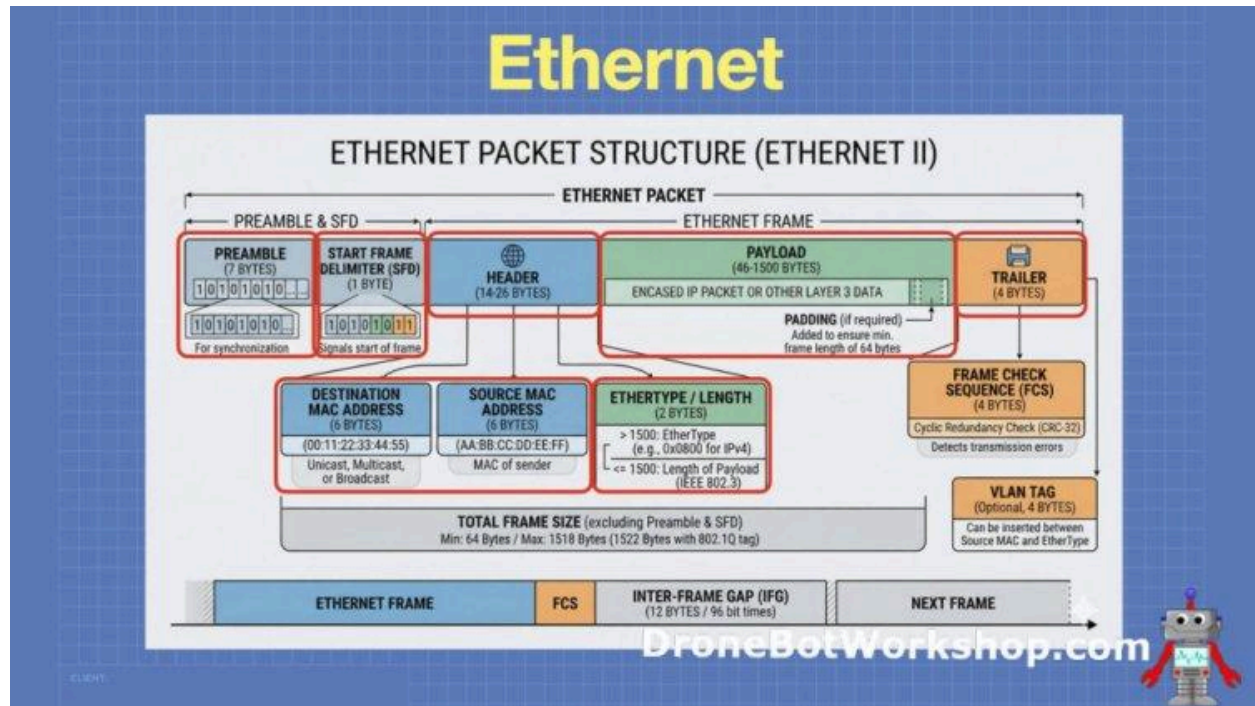
streaming video.

How Ethernet Works

At its heart, Ethernet is a packet-switched network. Data is broken up into small chunks called packets (or, more precisely, frames), and each frame is tagged with the addresses of the sender and the receiver. The network equipment along the way uses those addresses to forward each frame to where it needs to go.

The Ethernet Frame

Every chunk of Ethernet traffic is wrapped up in a frame with a well-defined structure. A frame begins with a preamble and a start-of-frame delimiter that give the receiver time to synchronise. That is followed by a header of 14 to 26 bytes that contains the destination and source MAC addresses and a field identifying the type of data that follows. Then comes the payload – the actual data being transmitted, anywhere from 46 to 1500 bytes – and finally a four-byte trailer known as the Frame Check Sequence, a CRC value the receiver uses to confirm the frame arrived intact.



MAC Addresses

Every Ethernet-capable device has a 48-bit (6-byte) Media Access Control or MAC address burned into it at the factory. MAC addresses are usually written as six pairs of hexadecimal digits, like AA:BB:CC:DD:EE:FF. The first three bytes identify the manufacturer and the last three identify the specific device. An address of all ones (FF:FF:FF:FF:FF:FF) is the broadcast address – a frame sent to that address is delivered to every device on the local network.

TIP: For projects using the W5500, you supply your own MAC address in code. Using a locally-administered address (the second-least-significant bit of the first byte set to 1, as in 02:AB:CD:EF:01:23) guarantees it will not collide with any real manufacturer's address on your LAN.

CSMA/CD – The Old Way

The original Ethernet was a shared-medium network: every device connected to the same cable, so only one could transmit at a time. To manage who got to talk and when, Ethernet used a protocol called Carrier Sense Multiple Access with Collision Detection, or CSMA/CD. A device that wanted to transmit would first listen to see if the cable was idle, then begin transmitting, and keep listening for a collision with another simultaneous transmission. If it heard one, both senders would back off for a random interval and try again.

Modern switched, full-duplex Ethernet has essentially rendered CSMA/CD obsolete – each device has a dedicated link to a switch, so collisions cannot happen – but the algorithm is a big part of Ethernet's history and is still occasionally useful to know about.

Hubs, Switches, and Routers

Three pieces of equipment dominate a modern Ethernet network, and the differences between them are worth understanding:

- **Hubs** were the earliest multi-port Ethernet devices. A hub repeats every incoming signal out of every other port – meaning all devices share the same bandwidth, and collisions can occur. Hubs are essentially obsolete today.
- **Switches** are the modern replacement. A switch learns which MAC address is reachable through which port (it builds a table internally as it observes traffic) and forwards each frame only to the port that needs it. That gives every device a dedicated, collision-free, full-duplex connection. Switches operate at Layer 2 – they route by MAC address.
- **Routers** operate one layer up. A router connects separate networks together, and routes traffic by IP address rather than MAC address. In a home or small office, the router is usually the device that connects the internal network to the internet and acts as the default gateway for everything on the LAN.

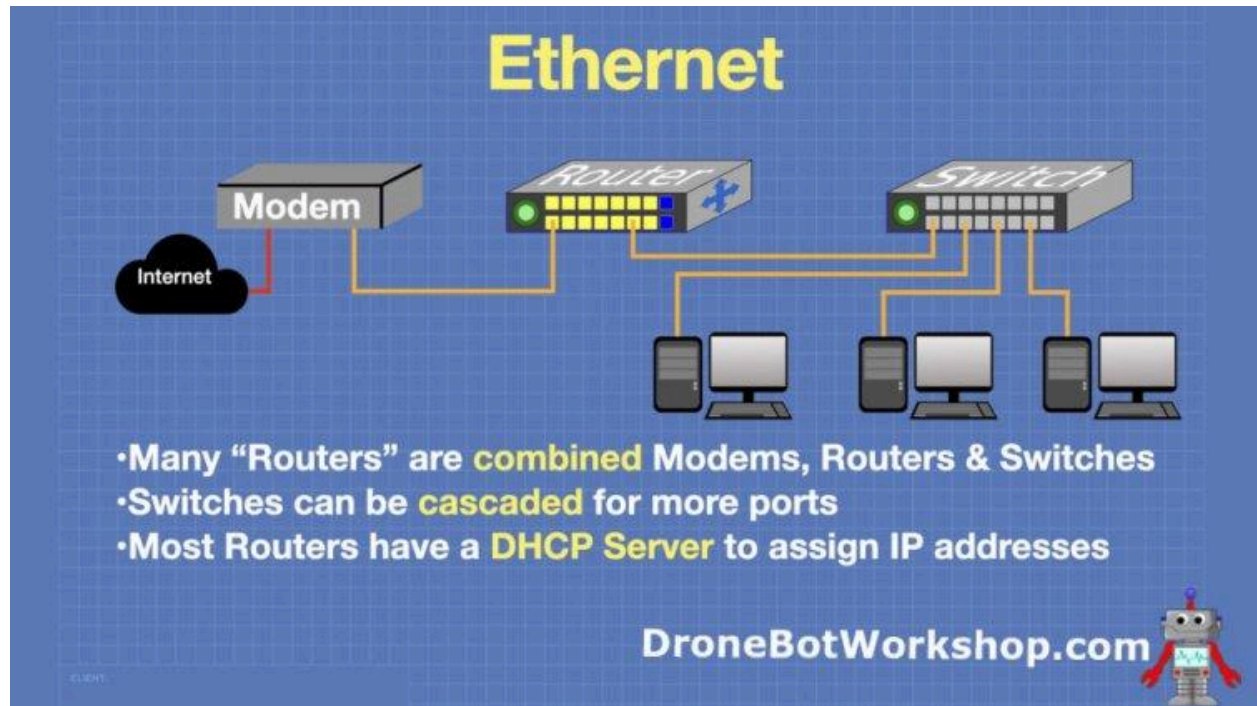
For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

A useful way to remember the distinction: a switch connects devices within the same network, while a router connects separate networks together.

A Typical Home or Office Network

Here is how these pieces fit together in a very common arrangement. The internet connection arrives at a device called a modem – short for modulator/demodulator – which translates between the signalling used by an internet service provider (cable, fibre, DSL) and Ethernet. The modem feeds into a router, the router’s switch ports connect to the workstations and other devices on the network, and additional Ethernet switches can be cascaded if more ports are needed.

Most consumer-grade routers also run a DHCP (Dynamic Host Configuration Protocol) server, which automatically hands out IP addresses to devices that join the network. This is why a new device typically just “works” the moment it is plugged in – and it is the mechanism the W5500 sketches in this article rely on to get the ESP32 onto the network.




TIP: Most consumer “routers” today are really all-in-one devices that combine a modem, a router, and a small Ethernet switch in a single box, often with Wi-Fi bolted on as well. With one of those, everything described above happens inside a single piece of plastic.

Ethernet Cables – The Wiring Inside

An Ethernet cable contains eight individual conductors, arranged into four twisted pairs. The twisting is not decorative. Ethernet uses balanced, differential signalling: each pair carries a signal and its exact inverse, and the receiver looks at the difference between them. Any noise picked up along the cable gets induced equally into both conductors of the pair and cancels out at the receiving end. Twisting the conductors together tightens that cancellation, and twisting each of the four pairs at a slightly different rate further reduces crosstalk between them.


Each end of the cable terminates in an 8-position, 8-contact (8P8C) modular plug – almost universally, and slightly incorrectly, called an RJ45 connector.



Ethernet


ETHERNET CABLE PINOUT DIAGRAMS: T568A AND T568B

T568A STANDARD




T568A

THE DIFFERENCE



T568B STANDARD




T568B

WIRING TABLE

Pair	Wire Color	T568A Color	T568B Color
1	Blue	Blue	Blue
2	White/Blue	White/Blue	White/Orange
3	White/Green	White/Green	White/Green
4	White/Brown	White/Brown	White/Brown
5	Green	Green	Green
6	White/Orange	White/Orange	White/Blue
7	Orange	Orange	Orange
8	White/Brown	White/Brown	White/Brown

- Ethernet cables have **8 conductors** (Four Twisted-Pairs)
- **RJ45** Connector at each end
- Two standard **wiring configurations** (T568B Most Common)

DroneBotWorkshop.com



T568A vs. T568B – The Two Pinouts

There are two accepted standards for wiring the eight conductors into an RJ45 plug: T568A and T568B. They are defined by TIA/EIA-568, and they differ only in the ordering of two pairs of wires. Both are electrically identical, and both work equally well – what matters is that both ends of a given cable use the same standard.

Pin	T568A Color	T568B Color	10/100 Mbps	Gigabit
1	White/Green	White/Orange	TX+	BI_DA+
2	Green	Orange	TX–	BI_DA–
3	White/Orange	White/Green	RX+	BI_DB+
4	Blue	Blue	(unused)	BI_DC+
5	White/Blue	White/Blue	(unused)	BI_DC–
6	Orange	Green	RX–	BI_DB–
7	White/Brown	White/Brown	(unused)	BI_DD+
8	Brown	Brown	(unused)	BI_DD–

T568B is by far the most common wiring standard in North American commercial installations. T568A is mandated in some government projects and preferred by several international standards bodies. Either is fine – just be consistent.

Straight-Through, Crossover, and Rollover Cables

There are three traditional cable types worth knowing about:

- **Straight-through cables** are wired with the same standard on both ends – T568B-to-T568B or T568A-to-T568A. These are used to connect unlike devices, such as a computer to a switch or an ESP32 to a router. This is the kind of cable used for almost everything in this article.
- **Crossover cables** use T568A on one end and T568B on the other. This swaps the transmit and receive pairs, and was historically required to connect two like devices directly – two computers, or two switches. Almost every modern Ethernet port supports Auto-MDIX, which detects and corrects cable type electronically, so crossover cables are effectively obsolete.
- **Rollover cables** (sometimes called console cables) completely reverse the pin order, pin 1 at one end to pin 8 at the other. These were used for serial console connections to network equipment – not for regular Ethernet traffic.

WARNING: A note for W5500 users: the W5500 datasheet explicitly states that the chip does not support Auto-MDIX. In practice this almost never matters, because the W5500 will be plugged into a switch or router, both of which do support it. If a W5500 is ever connected directly to a computer's Ethernet port, however, make sure that at least one end of the link supports Auto-MDIX.

Ethernet Cable Categories

Ethernet cables are sold in categories – abbreviated “Cat” – that define their electrical characteristics, the speeds they support, and their maximum bandwidth. Higher

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

categories use tighter twists, better insulation, and sometimes additional shielding to achieve higher performance. The categories likely to be encountered today are:

Categ ory	Max Speed	Bandwi dth	Max Length	Notes
Cat 5e	1 Gbps	100 MHz	100 m	Most common; fine for home and office Gigabit
Cat 6	10 Gbps	250 MHz	55 m @ 10G / 100 m @ 1G	Recommended minimum for new installs
Cat 6A	10 Gbps	500 MHz	100 m	Best choice for PoE++ and long 10G runs
Cat 7	10 Gbps	600 MHz	100 m	Not a TIA/EIA standard – best avoided
Cat 8	25–40 Gbps	2000 MHz	30 m	Data-center only (switch-to-switch)

Older Cat 3 (up to 10 Mbps) and Cat 5 (up to 100 Mbps) cabling can still occasionally be found in older buildings – both are effectively obsolete and should be replaced.

Categories 7 and 7A are technically in use but were never recognized by the TIA/EIA standards body; much of what is sold as “Cat 7” online is actually Cat 6A. For a new installation, Cat 6 or Cat 6A is almost always the right choice.

TIP: For any ESP32 project using the W5500 or the Waveshare board – both of which top out at 100 Mbps – even old Cat 5 cable works perfectly. Practically speaking, Cat 5e is the cheapest, most widely available option and covers everything in this article.

<https://dronebotworkshop.com>

Shielded vs. Unshielded

Cables are also labeled UTP (Unshielded Twisted Pair), FTP (Foil Twisted Pair, with a foil wrap around the whole bundle), or STP/SFTP (Shielded Twisted Pair, with shielding around individual pairs and/or the full bundle). UTP is the ordinary choice for home and office use. Shielded cable is worth the extra cost for runs through industrial environments with big motors, drives, or other heavy electrical noise nearby.

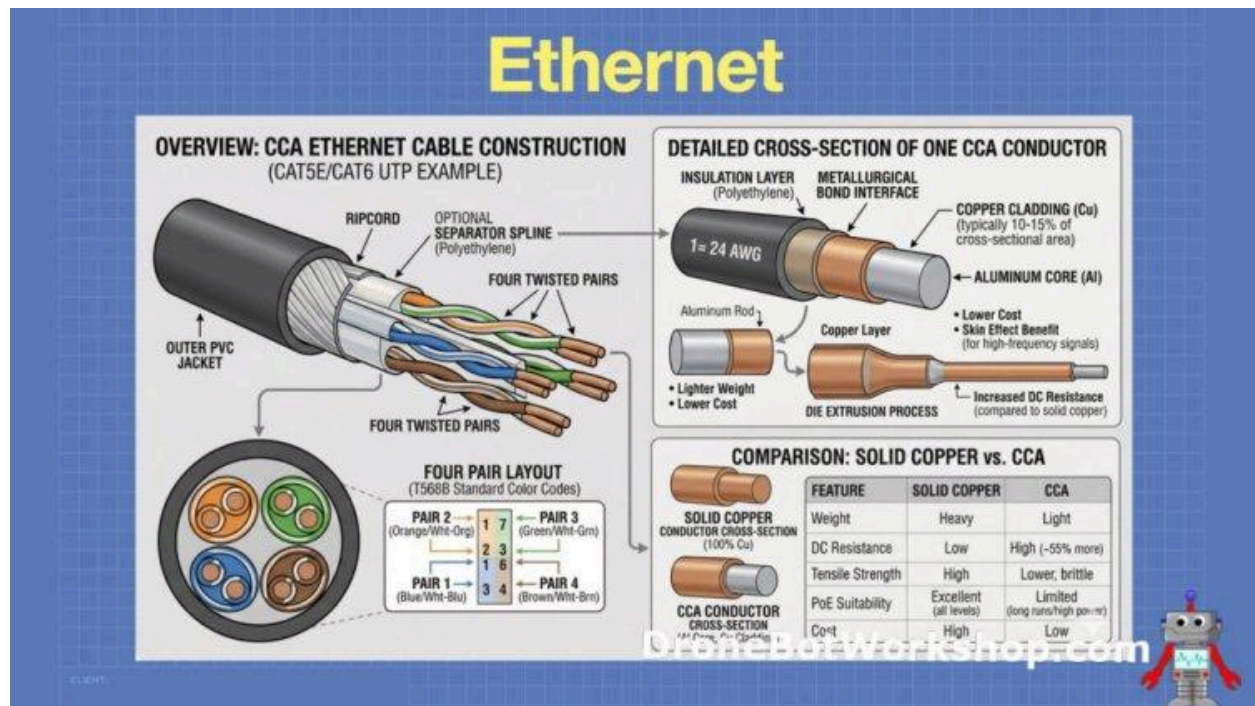
Watch Out for CCA – Copper-Clad Aluminum Cable

WARNING: One very important type of Ethernet cable to be aware of – and to avoid – is CCA, which stands for Copper-Clad Aluminum.

CCA cable replaces the solid copper conductors of a proper Ethernet cable with aluminum wire coated with a thin layer of copper. It technically works, thanks to a phenomenon called the skin effect: at high frequencies, electrical signals travel mostly on the outer surface of a conductor, so a copper-plated aluminum wire can carry Ethernet's high-frequency signals reasonably well. And because aluminum is much cheaper and lighter than copper, CCA cable is dramatically less expensive to manufacture and ship.

The problems show up quickly, though. CCA has higher DC resistance than solid copper, which causes voltage drop and signal loss over long runs. The connectors do not crimp as reliably onto aluminum as they do onto copper, so terminations fail over time. And – this one matters especially here – the extra resistance generates heat when PoE current flows through it. Under heavy PoE load, a CCA cable can get hot enough to pose a fire hazard.

Although some CCA cable is marked with a category number on the jacket (“Cat 6 CCA”), it does not actually meet the relevant TIA/EIA specifications and should not be treated as equivalent to solid-copper cable. CCA cable is specifically prohibited for use in permanent installations by many building codes, and is, in fact, illegal to install in certain jurisdictions.



When in doubt, check the cable jacket. Quality copper cable is typically labeled with the category, UL and ETL certifications, and often explicitly says “100% copper” or “bare copper conductors.” If a bulk cable price looks too good to be true, it probably has CCA conductors. Avoid it.

Ethernet vs. Wi-Fi – The Short Version

To wrap up the Ethernet primer, here is a quick recap of why this medium is worth the extra cable:

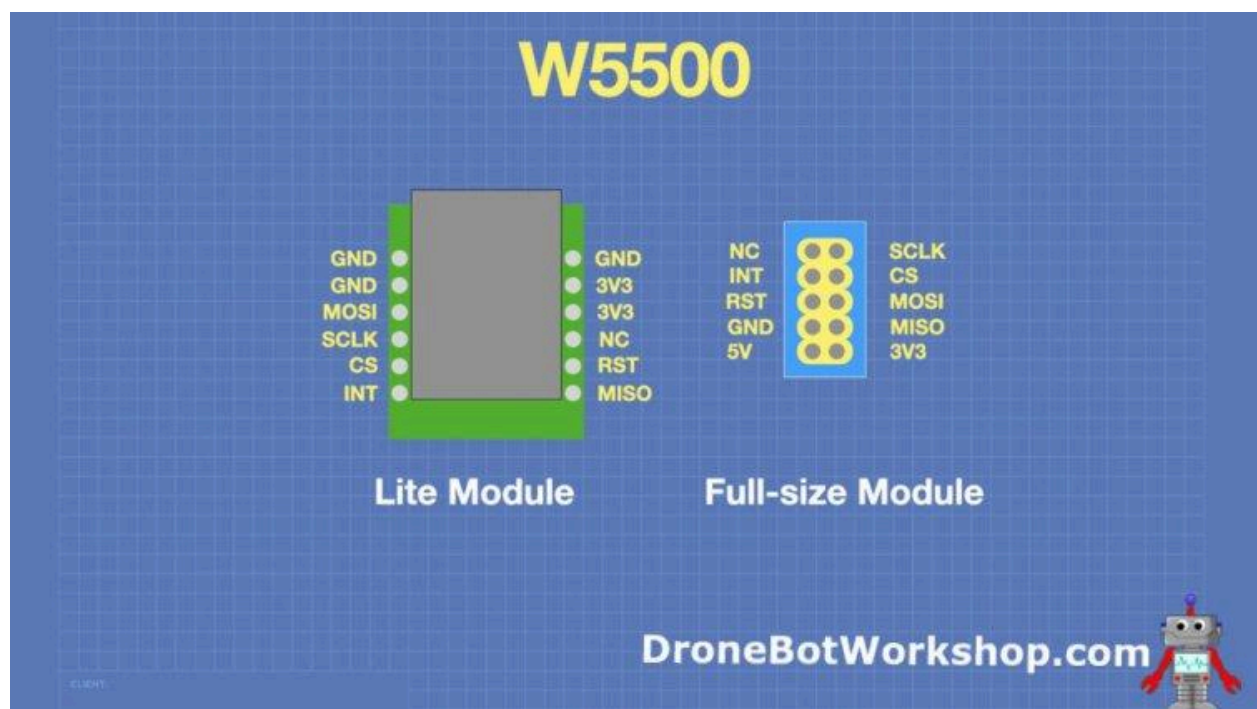
- Higher maximum speed than Wi-Fi, and consistently so.
- Very low latency – sub-millisecond on a local switch.
- Immune to the radio-frequency interference that Wi-Fi struggles with.
- Higher physical security – an attacker needs access to the cable itself.
- Full duplex on switched networks – transmit and receive simultaneously.
- Bandwidth does not degrade when other users join the network.
- And – the star of the second half of this article – Power over Ethernet.

PoE is covered in detail in its own section later. For now, on to the hardware.

Using the W5500 Adapter

The first hardware option is a very common, inexpensive board based on the WIZnet W5500 chip. It pops up everywhere – on Amazon, AliExpress, and in almost every embedded Ethernet tutorial online – because it is cheap, well supported, and makes adding Ethernet to a microcontroller surprisingly painless.

The W5500 comes in two common module formats. The full-size board has an on-board 3.3 V regulator and can be powered from 5 V. The smaller “Lite” module omits the regulator and expects 3.3 V directly. The Lite is used in this article, but everything here applies equally to the full-size version – just feed the larger board 5 V instead of 3.3 V. Both modules have the same functional pinout and the same 5 V-tolerant I/O.



What's Inside the W5500

The key thing to understand about the W5500 is that it is not merely an Ethernet PHY – it is a complete, hardwired TCP/IP stack baked into silicon. Most Ethernet adapters available for microcontrollers only handle the physical-layer signaling; the host microcontroller still has to run a software TCP/IP stack in its own RAM and CPU time. The W5500 is different. It does all of that in hardware. The ESP32 just opens a “socket” on the chip via SPI, and the W5500 takes care of ARP, IP, TCP, UDP, fragmentation, retransmissions – the whole lot.

The W5500 supports the following protocols natively in hardware: TCP, UDP, ICMP (the protocol used by ping), IPv4, ARP, IGMP, and PPPoE. It also offers eight independent hardware sockets, allowing the ESP32 to support eight simultaneous TCP or UDP connections with no additional code complexity. A 32 KB internal buffer memory is shared among those sockets, and it can be divvied up however the application needs.

Feature	Specification
Interface	SPI (Mode 0 or Mode 3), up to 80 MHz
TCP/IP Stack	Hardwired – zero MCU overhead
Independent Sockets	8 simultaneous hardware sockets
Internal Buffer	32 KB, configurable per socket
PHY	10BASE-T / 100BASE-TX embedded

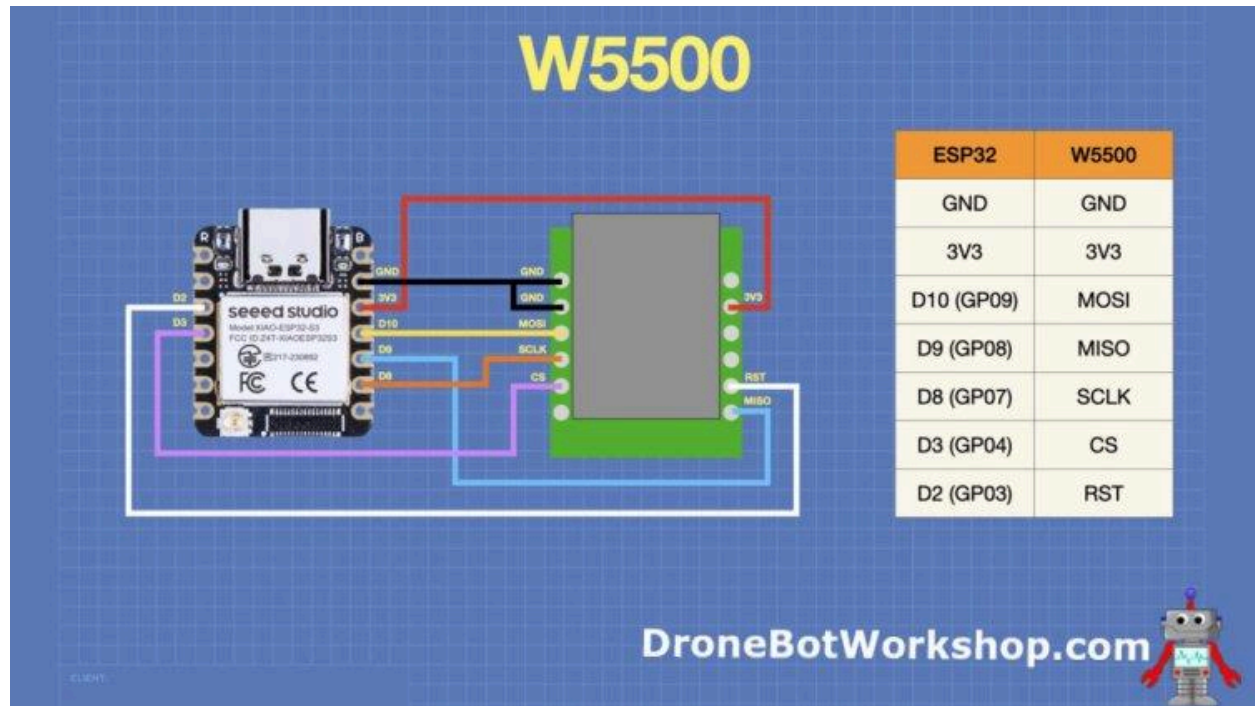
Auto-Negotiation	10/100 Mbps, full or half duplex
Supply	3.3 V core; 5 V-tolerant I/O
Current	~132 mA operating, 13 mA in power-down
Wake-on-LAN	Magic-packet over UDP
Auto-MDIX	Not supported

Wiring the W5500 to an ESP32

The W5500 talks to the host microcontroller over SPI, so any ESP32 board with a free SPI bus will work. The Seeed Studio XIAO ESP32-S3 is used here because it is tiny, breadboard-friendly, and has plenty of GPIO. The code and pinout below are specific to that board, but every ESP32 has the same basic SPI peripheral – any ESP32 will work; the pin numbers may just need to be adjusted.

The wiring between the W5500 Lite and the XIAO ESP32-S3:

W5500 Lite Pin	XIAO ESP32-S3 Pin	GPIO	Function
G	GND	–	Ground (tie at least two)
V	3V3	–	3.3 V power supply
MO	D10	9	SPI MOSI – master-out, slave-in
MI	D9	8	SPI MISO – master-in, slave-out
SCK	D8	7	SPI clock
CS	D3	4	SPI chip select
RST	D2	3	Reset (active low)
INT	–	–	Interrupt out (leave unconnected)



TIP: Connect 3.3 V – not 5 V – to the V pin of the W5500 Lite. The Lite version does not have an on-board regulator. The full-size W5500 module can be powered from either 3.3 V or 5 V via its on-board regulator.

It is worth noting that while the W5500 is a 3.3 V device, its I/O pins are 5 V-tolerant. That means the SPI lines can be driven from a 5 V microcontroller without a level shifter. The ESP32 is itself a 3.3 V device, so this does not come up here, but it is a handy feature when using the W5500 with an Arduino Uno.

Arduino IDE Setup

Both of the sketches that follow require the ESP32 Arduino core (version 3.x), installed through the Boards Manager. Under Tools, select the board that matches your hardware – for the XIAO ESP32-S3, choose “XIAO_ESP32S3” under the Seeed ESP32 family.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

The sketches rely on two libraries, both of which come bundled with the ESP32 core – so there is nothing extra to install:

- **ETH.h** – the ESP32’s built-in Ethernet driver, with first-class W5500 support starting in core 3.x.
- **WiFi.h** – included not because Wi-Fi is being used, but because several of the network event types and helper classes (`WiFiEvent_t`, `WiFiServer`, `WiFiClient`) are defined there and work transparently over Ethernet.
- **SPI.h** – also built-in; just required as an `#include`.

TIP: There is also an older Arduino library called “Ethernet” (by Arduino). That library works, but it has some rough edges on the ESP32-S3 – most noticeably it can pick the wrong SPI bus. The built-in `ETH.h` library is the best choice on modern ESP32 cores, and it is what the sketches below use.

Demo 1 – DHCP Client Sketch

The first sketch is simple. It brings up the Ethernet interface, requests an IP address via DHCP, prints all network configuration to the Serial Monitor, and then reports the link status every 5 seconds. It is the networking equivalent of “Blink” – the smallest thing that proves everything works.

The sketch uses three libraries – SPI, WiFi, and ETH – even though Wi-Fi is not used. The WiFi library just defines the event types and some basic networking helpers that ETH uses under the hood.

All the pin assignments are defined up top so the sketch can easily be adapted to a different ESP32 board. A boolean flag tracks whether there is a live connection. The Ethernet event handler is the heart of the sketch – it is called automatically by the Ethernet library every time something interesting happens (the hardware comes up, the cable is plugged in, a DHCP lease is granted, and so on). When the IP address is granted, the handler prints the IP address, subnet mask, gateway, MAC address, link speed, and duplex mode.

```
/*  
  W5500 Ethernet DHCP Demo  
  w5500_dhcp_demo.ino  
  Shows basic operation of W5500 Ethernet module  
  Uses W5500 Lite Ethernet module  
  Uses Seeeduino XIAO ESP32-S3  
  Uses Eth and WiFi Libraries  
  DroneBot Workshop 2026  
  https://dronebotworkshop.com  
*/  
  
#include <SPI.h>  
#include <WiFi.h> // ← required for WiFiEvent_t and WiFi.onEvent()  
#include <ETH.h>  
  
// — Pin definitions —————  
#define W5500_CS_PIN 4    // D3  = GPIO4  
#define W5500_RST_PIN 3   // D2  = GPIO3  
#define W5500_INT_PIN -1  // Not wired  
#define W5500_SCK_PIN 7   // D8  = GPIO7  
#define W5500_MISO_PIN 8  // D9  = GPIO8  
#define W5500_MOSI_PIN 9  // D10 = GPIO9  
  
static bool ethConnected = false;  
  
// — Ethernet event handler —————  
void onEthEvent(WiFiEvent_t event) {  
  switch (event) {
```

```
case ARDUINO_EVENT_ETH_START:

    Serial.println("ETH: hardware started");

    ETH.setHostname("xiao-w5500");

    break;


case ARDUINO_EVENT_ETH_CONNECTED:

    Serial.println("ETH: cable connected ✓");

    break;


case ARDUINO_EVENT_ETH_GOT_IP:

    Serial.println("\n—— Network Configuration ——");

    Serial.print("  IP Address  : ");

    Serial.println(ETH.localIP());

    Serial.print("  Subnet Mask : ");

    Serial.println(ETH.subnetMask());

    Serial.print("  Gateway    : ");

    Serial.println(ETH.gatewayIP());

    Serial.print("  MAC Address : ");

    Serial.println(ETH.macAddress());

    Serial.print("  Link Speed  : ");

    Serial.print(ETH.linkSpeed());

    Serial.println(" Mbps");

    Serial.print("  Full Duplex : ");

    Serial.println(ETH.fullDuplex() ? "yes" : "no");

    Serial.println("——");

    ethConnected = true;

    break;
```

```
    case ARDUINO_EVENT_ETH_DISCONNECTED:
        Serial.println("ETH: disconnected");
        ethConnected = false;
        break;

    case ARDUINO_EVENT_ETH_STOP:
        Serial.println("ETH: stopped");
        ethConnected = false;
        break;

    default:
        break;
}

// -----

void setup() {
    Serial.begin(115200);
    delay(1500);
    Serial.println("\n=== W5500 Lite - ETH.h Test ===\n");

    // Register event handler BEFORE calling ETH.begin()
    WiFi.onEvent(onEthEvent);

    Serial.println("Starting W5500 via ETH.begin()...");
    bool ok = ETH.begin(ETH_PHY_W5500,
                        1,
```

```
        W5500_CS_PIN,  
        W5500_INT_PIN,  
        W5500_RST_PIN,  
        SPI2_HOST,  
        W5500_SCK_PIN,  
        W5500_MISO_PIN,  
        W5500_MOSI_PIN);  
  
if (!ok) {  
    Serial.println("ETH.begin() returned false - W5500 not detected.");  
    Serial.println("Check MO, MI, SCK, CS, RST, V, G wiring.");  
    while (true) delay(1000);  
}  
  
Serial.println("ETH.begin() OK - waiting for DHCP...");  
  
unsigned long timeout = millis();  
while (!ethConnected) {  
    if (millis() - timeout > 10000) {  
        Serial.println("Timeout - no IP received. Check cable and router.");  
        while (true) delay(1000);  
    }  
    delay(100);  
}  
  
// -----  
  
void loop() {
```

```
static unsigned long lastPrint = 0;

if (millis() - lastPrint >= 5000) {

    lastPrint = millis();

    Serial.print("[");

    Serial.print(millis() / 1000);

    Serial.print("s]  Link: ");

    Serial.print(ethConnected ? "UP" : "DOWN");

    Serial.print("    IP: ");

    Serial.println(ETH.localIP());

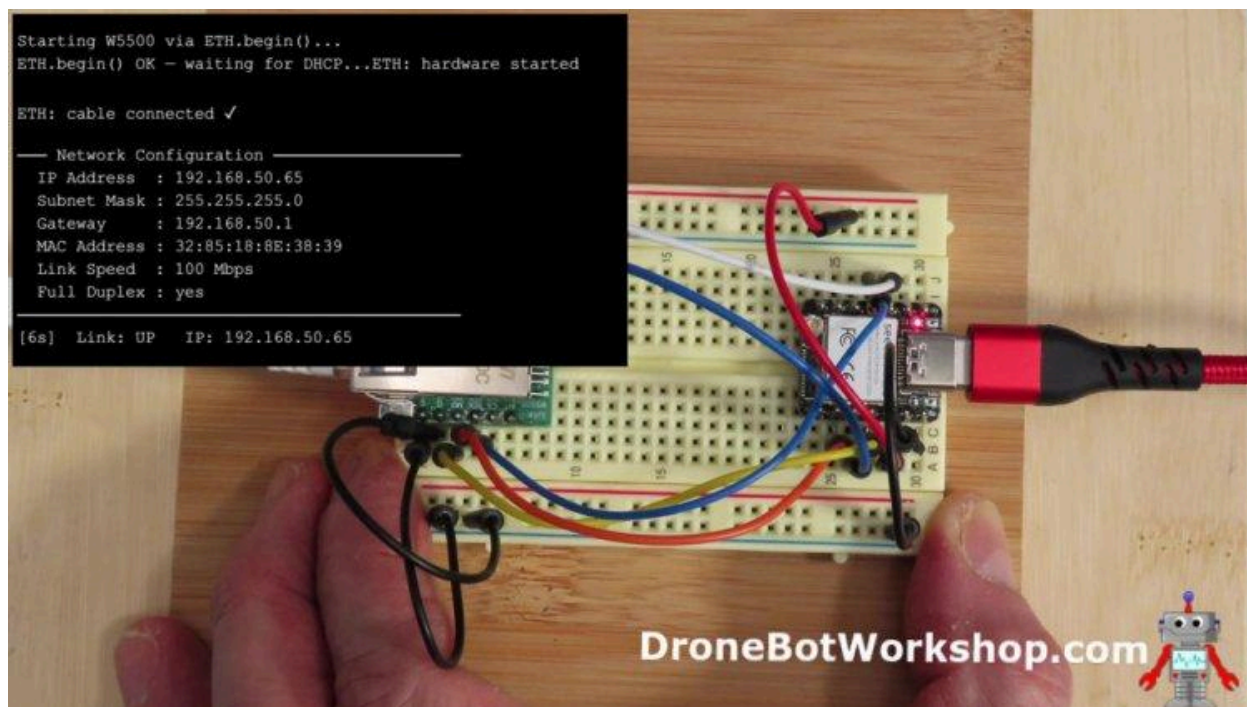
}

}
```

Upload the sketch, open the Serial Monitor at 115200 baud, and the W5500 will start up, the cable will connect, and the router will assign an IP address. From there, the loop reports the link status and address every five seconds – useful for verifying that everything keeps working after the cable is unplugged and reconnected.

TIP: The “1” passed as the second argument to `ETH.begin()` is the PHY address of the W5500 on its internal bus – not a DHCP setting. It is always 1 for the W5500. DHCP is the default behavior; for a static IP, call `ETH.config(ip, gateway, subnet, dns)` before `ETH.begin()`.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>



<https://dronebotworkshop.com>

Demo 2 – A Simple Ethernet Web Server

Once the board can get on the network, the obvious next step is to serve something.

This second sketch turns the ESP32 into a small HTTP web server that responds to any request with a static HTML page – a self-contained “Welcome to the Workshop” status card showing the board, network type, controller chip, and online status. It uses the same hardware setup as the DHCP sketch – nothing needs to change on the breadboard.

The server logic is straightforward: the sketch listens for incoming TCP connections on port 80, reads the HTTP request headers until it sees the blank line that marks the end of them, then sends back a standard HTTP/1.1 response followed by the HTML body. The HTML itself lives in a raw string literal up at the top of the sketch, which means it can be edited without fighting with escape characters.

A few things worth pointing out about this sketch:

- **WiFiServer, not EthernetServer.** Because ETH.h is in use, the WiFiServer / WiFiClient classes work transparently over the Ethernet connection. Any library that normally works with WiFiClient – MQTT clients, HTTPS libraries, and so on – will work over Ethernet with no code changes.
- **Raw string literal for the HTML.** Wrapping the HTML in `R"html(...)html"` allows multi-line HTML to be pasted in with all its quotes and angle brackets without escaping a single character. Much cleaner than a hundred `client.println()` calls.
- **Static page.** This example serves the same page on every request. Once it is working, extending it to respond dynamically – read a sensor, flip a GPIO, report uptime – is straightforward.

```
/*  
  ESP32 Ethernet Web Server  
  w5500_enet_web_server.ino  
  Simple ESP32 static web server using Ethernet  
  Uses W5500 Ethernet Module  
  Uses Seeeduino XIAO ESP32  
  DroneBot Workshop 2026  
  https://dronebotworkshop.com  
*/
```

```
#include <SPI.h>  
#include <WiFi.h>  
#include <ETH.h>
```

```
// — Pin definitions —  
  
#define W5500_CS_PIN 4    // D3  = GPIO4  
#define W5500_RST_PIN 3  // D2  = GPIO3  
#define W5500_INT_PIN -1 // Not wired  
#define W5500_SCK_PIN 7  // D8  = GPIO7  
#define W5500_MISO_PIN 8 // D9  = GPIO8  
#define W5500_MOSI_PIN 9 // D10 = GPIO9
```

```
// — HTTP server on port 80 —
```

```
WiFiServer server(80);
```

```
static bool ethConnected = false;
```

```
// — The HTML page - edit freely between the R"html( ... )html" —
```

```
const char PAGE[] = R"html(
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DroneBot Workshop</title>
  <style>
    * { margin: 0; padding: 0; box-sizing: border-box; }

    body {
      font-family: Arial, sans-serif;
      background: #1a1a2e;
      display: flex;
      justify-content: center;
      align-items: center;
      min-height: 100vh;
    }

    .card {
      background: #16213e;
      border: 2px solid #e94560;
      border-radius: 16px;
      padding: 48px 56px;
      text-align: center;
      box-shadow: 0 0 40px rgba(233,69,96,0.3);
      max-width: 480px;
      width: 90%;
    }
  </style>
</html>
)
```

```
}

.logo {

  font-size: 48px;

  margin-bottom: 12px;

}


h1 {

  color: #e94560;

  font-size: 2rem;

  margin-bottom: 8px;

  letter-spacing: 1px;

}


.tagline {

  color: #a8dadc;

  font-size: 1.1rem;

  margin-bottom: 32px;

}


.divider {

  border: none;

  border-top: 1px solid #e9456044;

  margin: 24px 0;

}


.info-grid {

  display: grid;
```

```
    grid-template-columns: 1fr 1fr;

    gap: 12px;

    text-align: left;

}

.info-item label {

    color: #a8dadac;

    font-size: 0.75rem;

    text-transform: uppercase;

    letter-spacing: 1px;

    display: block;

}

.info-item span {

    color: #f1faee;

    font-size: 0.95rem;

    font-weight: bold;

}

.footer {

    margin-top: 28px;

    color: #555;

    font-size: 0.75rem;

}

</style>

</head>

<body>

    <div class="card">
```

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

```
<div class="logo">&#x1F916;</div>

<h1>Welcome to the Workshop</h1>

<p class="tagline">DroneBot Workshop</p>

<hr class="divider">

<div class="info-grid">

  <div class="info-item">

    <label>Board</label>

    <span>XIAO ESP32-S3</span>

  </div>

  <div class="info-item">

    <label>Network</label>

    <span>Wired Ethernet</span>

  </div>

  <div class="info-item">

    <label>Controller</label>

    <span>W5500 Lite</span>

  </div>

  <div class="info-item">

    <label>Status</label>

    <span style="color:#4caf50;">&#x25CF; Online</span>

  </div>

</div>

<p class="footer">Serving over hardwired TCP/IP &mdash; W5500 hardwired stack</p>

</div>

</body>

</html>

)html";
```

<https://dronebotworkshop.com>

```
// — Ethernet event handler —————  
  
void onEthEvent(WiFiEvent_t event) {  
  
    switch (event) {  
  
        case ARDUINO_EVENT_ETH_START:  
  
            Serial.println("ETH: started");  
  
            ETH.setHostname("dronebot-workshop");  
  
            break;  
  
        case ARDUINO_EVENT_ETH_CONNECTED:  
  
            Serial.println("ETH: cable connected");  
  
            break;  
  
        case ARDUINO_EVENT_ETH_GOT_IP:  
  
            Serial.println("\n—— Network Configuration ——");  
  
            Serial.print("  IP Address  : ");  
  
            Serial.println(ETH.localIP());  
  
            Serial.print("  Gateway    : ");  
  
            Serial.println(ETH.gatewayIP());  
  
            Serial.print("  MAC        : ");  
  
            Serial.println(ETH.macAddress());  
  
            Serial.print("  Speed      : ");  
  
            Serial.print(ETH.linkSpeed());  
  
            Serial.println(" Mbps");  
  
            Serial.println("—————");  
  
            Serial.print("\n  Open http://");  
  
            Serial.print(ETH.localIP());  
  
            Serial.println(" in your browser.\n");
```



```
        ethConnected = true;

        break;

    case ARDUINO_EVENT_ETH_DISCONNECTED:

        Serial.println("ETH: disconnected");

        ethConnected = false;

        break;

    default:

        break;

}

}

// -----

void setup() {

    Serial.begin(115200);

    delay(1500);

    Serial.println("\n=== DroneBot Workshop - Web Server ===\n");

    WiFi.onEvent(onEthEvent);

    if (!ETH.begin(ETH_PHY_W5500,

        1,

        W5500_CS_PIN,

        W5500_INT_PIN,

        W5500_RST_PIN,

        SPI2_HOST,

        W5500_SCK_PIN,
```

```
        W5500_MISO_PIN,
        W5500_MOSI_PIN)) {
    Serial.println("ETH.begin() failed - check wiring.");
    while (true) delay(1000);
}

// Wait for DHCP
while (!ethConnected) delay(100);

// Start the server now that we have an IP
server.begin();
Serial.println("Web server started - waiting for connections...\n");
}

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

    if (!client) return;

    Serial.print("Client connected from ");
    Serial.println(client.remoteIP());

    // Wait up to 3 seconds for the browser to send its request
    unsigned long timeout = millis();
    while (client.connected() && !client.available()) {
        if (millis() - timeout > 3000) {
            Serial.println(" Timeout - no request received.");
            client.stop();
        }
    }
}
```

```
        return;
    }

    delay(1);
}

// Read and discard the HTTP request headers
// (we serve the same page regardless of what was requested)
while (client.available()) {
    String line = client.readStringUntil('\n');
    if (line == "\r") break; // Blank line = end of request headers
}

// Send HTTP response
client.println("HTTP/1.1 200 OK");
client.println("Content-Type: text/html; charset=utf-8");
client.println("Connection: close");
client.println(); // Blank line = end of response headers
client.print(PAGE); // Send the HTML body

delay(1);
client.stop();
Serial.println(" Response sent, client disconnected.");
}
```

Flash this sketch, check the Serial Monitor for the assigned IP address, and type that address into a browser on the same network. The browser will display a dark-themed status card with the Workshop branding and the four-cell info grid showing Board, Network, Controller, and Status. From here, the page can be taken in any direction – swap the static HTML for a dashboard that reads sensors, control GPIO pins from

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

buttons, or add JavaScript to poll for live data. A wired Ethernet web server is also much more reliable than a Wi-Fi one – there are no mysterious disconnects to chase down.



<https://dronebotworkshop.com>

Power Over Ethernet

So far, the ESP32 has been talking to the network over Ethernet, but it has still needed a separate power supply – in this case, a USB-C cable. The next step is to get rid of that cable too, and power the board over the same Ethernet run that already carries the data. That is what Power over Ethernet, or PoE, does.

PoE is genuinely clever engineering. A single Cat 5e or Cat 6 cable delivers both the data signal and the DC electrical power needed to run the device on the far end, eliminating the need for a separate power supply. Safety features are built into the specification to protect ordinary non-PoE devices (like a laptop's network card) from accidentally being injected with 48 V, and the source cuts power the moment it sees the device unplugged – so a live cable is never left dangling from a wall socket.



The Two Devices Involved

Every PoE setup has exactly two roles in play:

- **PSE – Power Sourcing Equipment.** The device that provides the power. This is typically a PoE-capable Ethernet switch, or a standalone PoE injector – a small in-line device that plugs into an ordinary switch and an AC outlet, and feeds power onto the outgoing cable. A PoE injector is used in this project.
- **PD – Powered Device.** The device that receives the power. Common PDs include IP cameras, Voice-over-IP phones, wireless access points, and – in this article – the Waveshare ESP32-S3-POE-ETH with its PoE module installed.

The PoE Negotiation – Four Steps

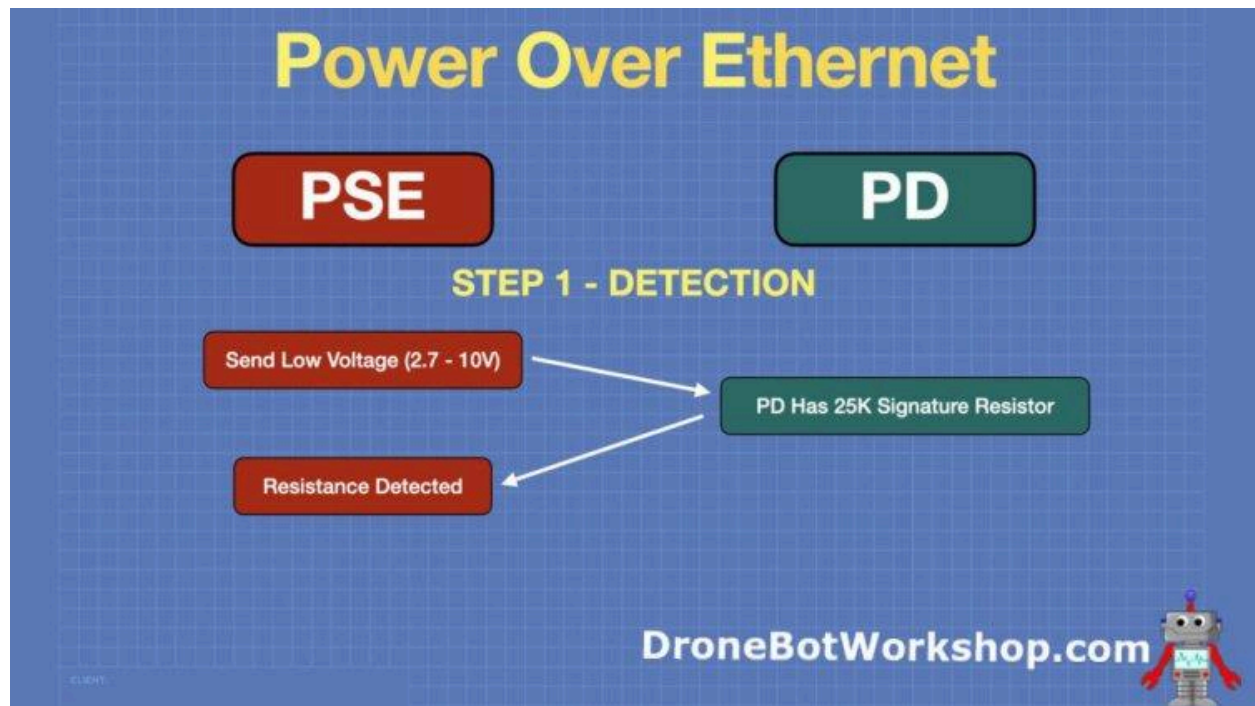
The PSE and the PD do not just start shoving 48 V down the cable the moment they are plugged together. They go through a structured negotiation with four distinct phases. Getting this right is what keeps PoE safe to use alongside non-PoE equipment on the same network.

Step 1 – Detection

Before any real power is delivered, the PSE needs to confirm that the device on the other end of the cable is a legitimate PoE-capable PD, and not, say, a laptop. If it gets this wrong, it could damage an innocent device by suddenly applying 48 V.

To do the check, the PSE sends a small, low-voltage probe signal down the cable – somewhere between 2.7 V and 10 V. A genuine PoE-powered device has a 25 k Ω signature resistor wired into it; a non-PoE device does not. The PSE measures the resistance it sees on the other end, and if it finds that 25 k Ω signature, it knows it is

talking to a real PoE device, and it is safe to proceed. If the signature is not present, no power is delivered. This is also why a regular Ethernet switch cannot be turned into a PoE switch by feeding power into one of its ports – without the signature resistor, a PoE PD will never see a signal it trusts, and the handshake will stall.



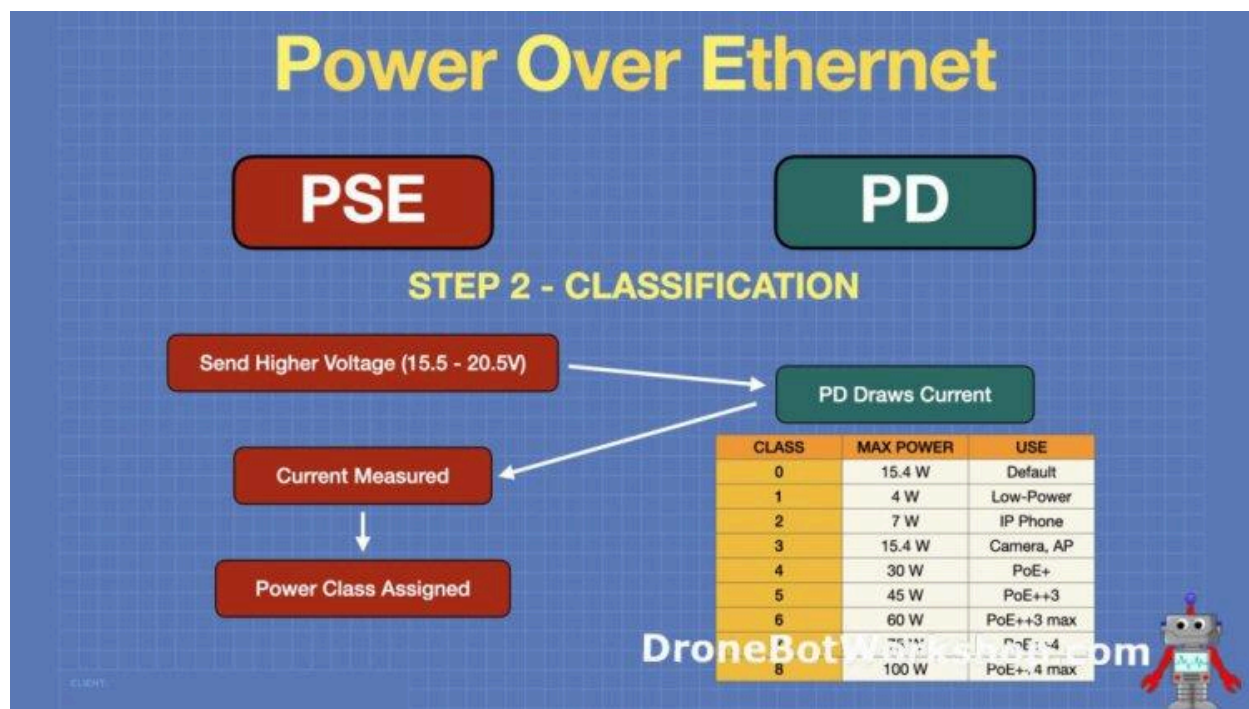
Step 2 – Classification

Once a valid PD has been detected, the PSE needs to know how much power it is expected to deliver. It figures this out by raising the probe voltage into the 15.5–20.5 V range and measuring the current the PD draws in response. The PD is designed to draw a specific amount of current in that voltage range, and that amount places it into one of eight power classes. The PSE then knows it should budget that much power for this port.

Class	Max Power at PSE	Typical Use

0	15.4 W	Default / unclassified – treated as Class 3
1	4.0 W	Very low-power sensors and endpoints
2	7.0 W	IP phones
3	15.4 W	Standard PoE – basic cameras, APs
4	30.0 W	PoE+ (IEEE 802.3at) – higher-power devices
5	45.0 W	PoE++ Type 3 (IEEE 802.3bt)
6	60.0 W	PoE++ Type 3
7	75.0 W	PoE++ Type 4
8	90–100 W	PoE++ Type 4 maximum

Note that if the PD does not fit cleanly into one of the numbered classes, it defaults to Class 0 – which the PSE treats as the equivalent of Class 3 (15.4 W). The ESP32-based board used here is a Class 0/3 device.



Step 3 – Power Delivery

With the PD detected and classified, the PSE now applies the full operating voltage – somewhere between 44 V and 57 V DC, with 48 V being the typical value. That voltage is delivered down the Ethernet cable, and there are three possible ways to do it:

- **Mode A – Endspan.** The DC power rides on the same two pairs that carry 10/100 Mbps data (pins 1 and 2 for positive, pins 3 and 6 for negative). This works because Ethernet data is differential AC, so DC can sit on top of the AC signal without interfering with it. This is the most common mode, and it is what most PoE-capable network switches use.
- **Mode B – Midspan.** The DC power uses the spare pairs (pins 4 and 5 for positive, pins 7 and 8 for negative) – the ones that are not used for data at 10/100 Mbps. This mode is common on midspan PoE injectors.
- **4-Pair (PoE++).** Required for the highest-wattage classes, this uses all four pairs simultaneously. It is defined by IEEE 802.3bt and enables the delivery of 90+ watts over a single cable.


The Waveshare ESP32-S3-POE-ETH is flexible enough to accept either Mode A or Mode B, whichever the PSE happens to deliver.

Power Over Ethernet

PSE **PD**

STEP 3 - POWER DELIVERY

- Once Classified, PSE applies full voltage (**44 - 57V**)
- Three methods of applying voltage:
 - **MODE A (Endspan)** - Uses same pairs used for data
 - **MODE B (Midspan)** - Uses spare pairs
 - **PoE++** - Uses all four pairs. Required for high-power

DroneBotWorkshop.com 

Step 4 – Ongoing Monitoring

The PSE does not just deliver power and walk away. It continuously monitors the DC load at regular intervals, watching for two things: the PD disappearing entirely, and the PD drawing more power than its assigned class allows.

If the PD is unplugged, the PSE sees the load disappear and cuts power within a few hundred milliseconds, so a freshly disconnected cable is immediately safe to touch. If the PD suddenly draws more power than its class permits, the PSE will also shut down that port to protect itself, the cable, and anything else on the line. This ongoing supervision is a big part of why PoE has proven to be a very safe power-delivery technology, even around untrained installers.

Power Over Ethernet

PSE

PD

STEP 4 - ONGOING MONITORING

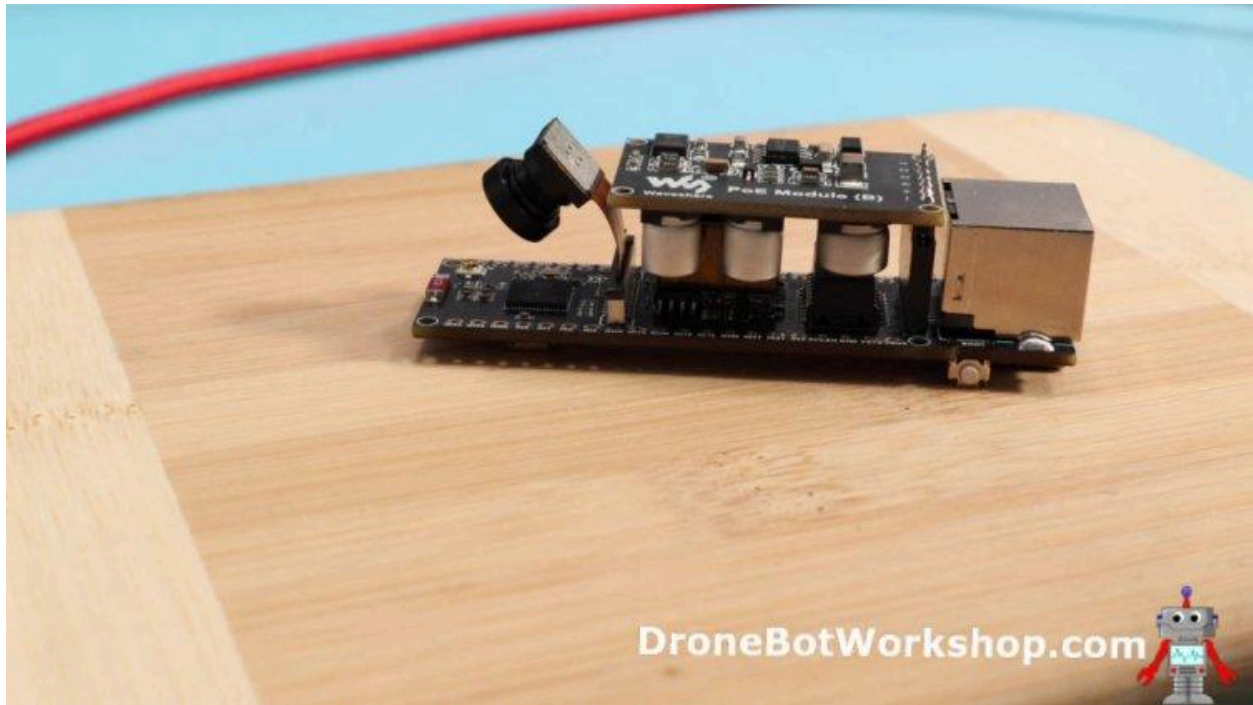
- PSE Monitors **DC Load** at regular intervals
- If PD is **unplugged** PSE cuts power
- If PD **exceeds power** for its class power is cut
- Ensures power is never applied to an **unconnected cable**

DroneBotWorkshop.com



The Waveshare ESP32-S3-POE-ETH

The hardware platform for the PoE demo is the Waveshare ESP32-S3-POE-ETH: a compact, feature-rich ESP32-S3 development board with a permanently-attached W5500 Ethernet interface and, importantly, a slot for an optional PoE module on top.



Waveshare sells the board in several configurations – just the Ethernet board on its own, the Ethernet board plus camera, the Ethernet board plus the PoE module, and the full kit with everything included. The unit used here is fitted with an OV5640 5-megapixel camera. An OV2640 will also work – both are supported in the example code.

The essentials of the board:

Feature	Specification

MCU	ESP32-S3, dual-core Xtensa LX7 @ 240 MHz
Flash	16 MB
PSRAM	8 MB
Wi-Fi	802.11 b/g/n 2.4 GHz
Bluetooth	Bluetooth 5.0 LE
Ethernet	W5500, 10/100 Mbps, via SPI (hard-wired on board)
PoE	IEEE 802.3af-compatible, via an optional module
Camera Interface	DVP 8-bit; supports OV2640 and OV5640
USB	USB-C (USB-OTG on ESP32-S3)
MicroSD	MicroSD card slot (SPI)
Buttons	Boot and Reset – side-mounted
GPIO	2.54 mm header pins for user I/O
Supply	5 V via USB-C, or PoE via Ethernet

Two things are especially worth calling out. First, the PoE module is completely hardware – there is no software involvement at all. Not a single line of code changes

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

when running the board off USB vs. PoE. The board does whatever it was going to do, and the PoE module quietly provides the 5V supply from the Ethernet cable. Second, the Boot and Reset buttons are on the side of the board rather than the top. That is unusual, but it turns out to be very convenient once the PoE module is mounted, because there is no longer any room on top to push a button anyway.



Full documentation, including pinouts, example code, schematics, and downloadable demo sketches, lives on the Waveshare Wiki:

<https://www.waveshare.com/wiki/ESP32-S3-THE>. The downloads are at the bottom of the page.

First Run – The Pre-Loaded Demo

Before writing any code, it is worth checking that the board itself is alive. Every unit ships with a small demo program pre-flashed that brings the Ethernet interface up and starts the camera. To try it, connect the board to a computer via USB-C, open the

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

Arduino IDE's Serial Monitor at 115200 baud, and watch what comes out. The board will report that PSRAM is enabled and indicate it is waiting for an Ethernet connection. Plug in an Ethernet cable, and the board will report a connection, grab an IP address via DHCP, and print the gateway it is talking to.

If the camera is fitted, the demo also starts the camera up. (A "partition not found" message may appear if there is no microSD card in the slot – that is normal and harmless.) Once the board reports a successful start, the hardware is good, and the next step – running the user code – can begin.

Arduino IDE – Required Settings

Waveshare's example sketches require a specific set of board options in the Arduino IDE. Getting these wrong is the single most common reason that sketches fail to compile or behave unexpectedly. Before uploading anything, go into Tools and set:

- Board: ESP32S3 Dev Module
- USB CDC On Boot: Enabled (this defaults to Disabled – change it)
- Flash Size: 16 MB (128 Mb)
- Partition Scheme: 16M Flash (3MB APP/9.9MB FATFS) – or use the partitions.csv included with the example
- PSRAM: OPI PSRAM

TIP: With the exception of Partition Scheme, these same settings are required for every one of Waveshare's ESP32-S3-POE-ETH examples. Set them once and leave them alone.

Demo – The ETH Web Cam

The example used here is the one that immediately stands out from the Waveshare Wiki: ETH_Web_CAM, the Ethernet version of the familiar ESP32 webcam. It is the last

<https://dronebotworkshop.com>

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

of the 13 sketches in Waveshare's demo zip, and it is the one that ties everything together – an ESP32, an OV5640 camera, an Ethernet connection, and PoE all running off a single cable.

The complete demo zip can be downloaded from the “Demo” link near the bottom of the [Waveshare Wiki page](#). Extract it, and **the ETH_Web_CAM folder contains the multi-file Arduino sketch**, ready to use. The example is used here as-is – there is no need to modify it.

Flashing the Sketch – A Quirk to Be Aware Of

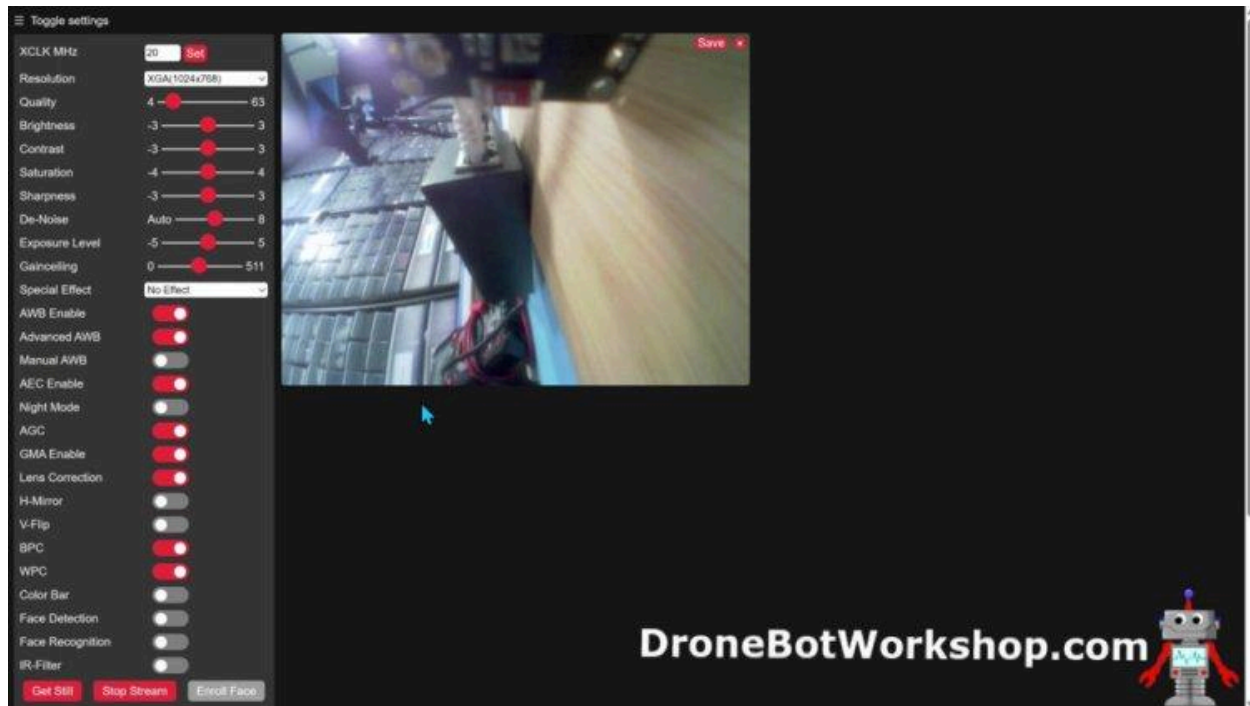
WARNING: Before flashing the sketch, detach the camera. One of the camera's pins is shared with a pin the ESP32-S3 uses during bootloader mode. If the camera is connected at flash time, the chip cannot enter the bootloader, and the upload will fail.

The procedure is:

1. Detach the camera ribbon cable from the board.
2. Connect USB-C. Open the Arduino IDE, open the ETH_Web_CAM sketch, and upload it.
3. After the upload succeeds, disconnect the USB-C and plug in the Ethernet cable.
4. Reconnect the USB-C cable, open the Serial Monitor, and note the IP address the board receives.
5. Unplug the power, reattach the camera, and plug the power back in.
6. Open a browser to *http://<ip-address>* to see the camera stream.

When you visit the page, you'll see the familiar ESP32-CAM control page, but accessible over Ethernet rather than Wi-Fi. The Waveshare Wiki recommends setting the resolution to XGA (1024×768), which is what I did in this image. You can experiment with the settings to get the best picture.

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

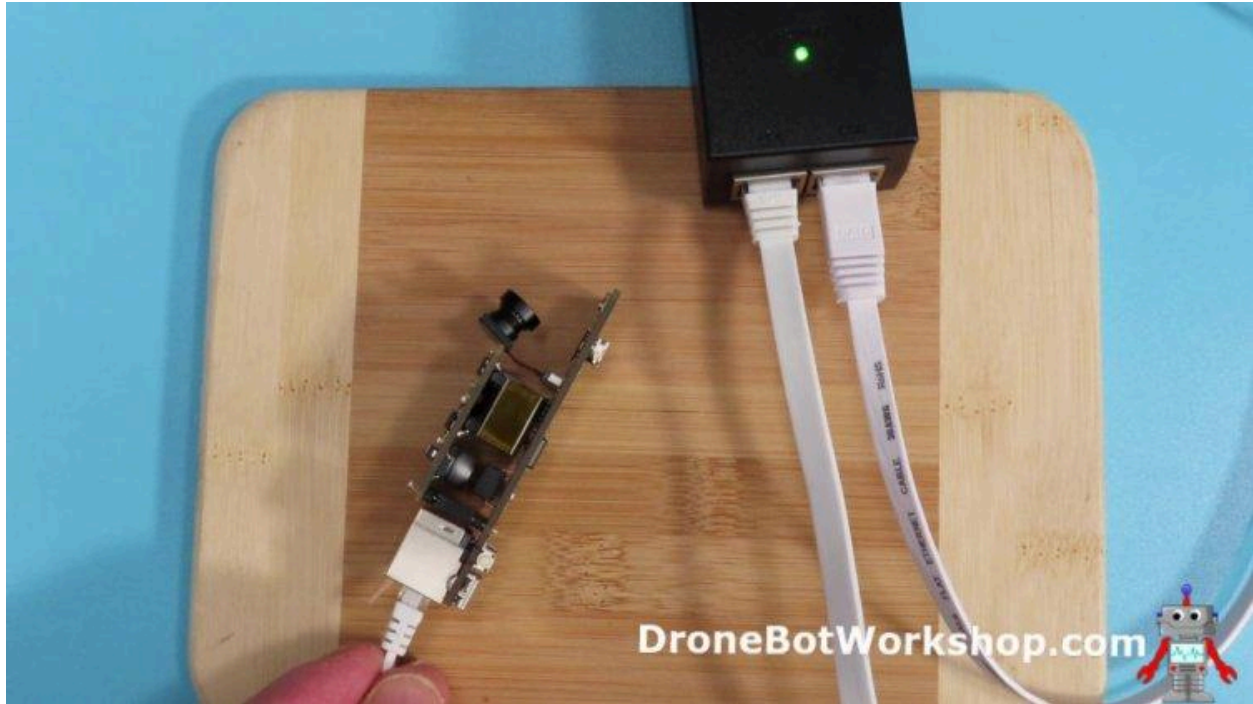


Running on PoE Power

The real payoff comes when the USB cable is swapped for a PoE feed. With the PoE module attached to the Waveshare board, plug the Ethernet cable that was going directly into the router into the PoE side of the injector, and run a second cable from the Ethernet side of the injector to the router. Plug the AC end of the injector into the wall, and remove the USB-C cable from the board entirely.

The board boots, runs through its Ethernet handshake and DHCP, starts the camera, and serves video – all while drawing power from the Ethernet cable. There is no software change whatsoever: the PoE module on the Waveshare board is purely hardware, and it converts whatever arrives on the Ethernet cable to the 5 V rail the board runs on. To the ESP32, it looks exactly like a USB power source.

<https://dronebotworkshop.com>



That is the whole point of PoE: a single Ethernet cable, carrying both data and power, running a live HTTP video stream with no separate power supply anywhere in the picture. Drop the board into an outdoor enclosure, run one cable to it, and you have a wired, powered, internet-connected camera.

TIP: For several devices that need to be powered this way, a PoE-capable Ethernet switch is a better option than a bank of individual injectors. Just remember that a regular switch cannot be turned into a PoE switch by injecting power into one of its ports – the switch has no 25 k Ω signature resistor, and no modern PD will draw power from it. A real PoE PSE is required.

Conclusion

We've covered quite a bit today – a half-century of Ethernet history, the wiring and categories of Ethernet cable, the dangers of CCA cabling, the W5500 adapter and a

For more projects and tutorials visit the DroneBot Workshop - <https://dronebotworkshop.com>

couple of ESP32 sketches, the four-step PoE negotiation, and finally a complete PoE-powered web camera running on nothing but a single cable.

Ethernet is not the right choice for every ESP32 project – for a device that has to move, Wi-Fi is still the obvious answer. But for fixed installations, especially when a cable will be run to the device anyway, Ethernet is an excellent choice. Better speed, better reliability, better security, and the option to deliver power over the same cable, all with only modest extra cost on the hardware side.

A few of the projects where Ethernet with the ESP32 really shines:

- PoE security cameras using the Waveshare ESP32-S3-POE-ETH, as built above.
- Industrial sensor nodes in wiring closets and control panels, where Wi-Fi would be unreliable.
- Home-automation hubs that need to stay connected 24/7, free of Wi-Fi reset cycles.
- Data-loggers mounted outdoors, powered over a single weatherproof Ethernet run.
- Wired MQTT sensor networks using the same MQTT libraries used over Wi-Fi – the ETH library swaps in transparently.
- Ethernet-based control of 3D printers, CNC machines, and other workshop tools where reliability matters.
- Outdoor wireless access points, fed both power and backhaul over a single PoE run.

So grab a few W5500 modules and start building your first wired Ethernet project!