Wi-Fi

Using the ESP8266 as a simple microcontroller is great, but the reason why most people use it, is its Wi-Fi capabilities. In this chapter, we'll dive into the wonderful world of network protocols, like Wi-Fi, TCP, UDP, HTTP, DNS ... All these acronyms might intimidate you, but I'll try my best to explain them step-by-step and in an easy way.

Some paragraphs are in *italic*. These provide some extra information, but are not critical to understanding the ESP's Wi-Fi functions, so don't get frustrated if there are things you don't understand.

It's really hard to give a clear explanation, without over-complicating things and while keeping it short enough as well. If you've got any feedback or remarks, be sure to leave a comment to help improve this article. Thanks!

The TCP/IP stack

The system most people refer to as 'The Internet' isn't just one protocol: it's an entire stack of layers of protocols, often referred to as the **TCP/IP stack**. We'll go over these different layers, because we need to understand how our ESP8266 communicates with other devices on the network.

Layer	Protocols
Application	HTTP, FTP, mDNS, WebSocket, OSC ...
Transport	TCP, UDP
Internet	IP
Link	Ethernet, Wi-Fi ...

The Link layer

The link layer contains the physical link between two devices, an Ethernet cable, for example, or a Wi-Fi connection. This is the layer that is closest to the hardware.
To connect an ESP8266 to the network, you have to create a Wi-Fi link. This can happen in two different ways:

1. The ESP8266 connects to a wireless access point (WAP or simply AP). The AP can be built-in to your modem or router, for example.
   In this configuration, the ESP acts like a wireless **station.**
2. The ESP8266 acts as an **access point** and wireless stations can connect to it. These stations could be your laptop, a smartphone, or even another ESP in station mode.

Once the Wi-Fi link is established, the ESP8266 is part of a **local area network** (LAN). All devices on a LAN can communicate with each other.
Most of the time, the AP is connected to a physical Ethernet network as well, this means that the ESP8266 can also communicate with devices that are connected to the AP (modem/router) via a wired Ethernet connection (desktop computers, gaming consoles and set-top boxes, for instance).
If the ESP8266 is in station mode, it can communicate with any station that is connected to it, and two stations (e.g. a laptop and a smartphone) can also communicate with each other.

The ESP can be used in AP-only, station-only, or AP+station mode.

TL;DR

The link layer is the physical link between devices: in the case of the ESP8266, this is a WiFi connection. The ESP can act as a station and connect to an access point, or act as an access point and let other devices connect to it.

The Internet or Network layer

Although the devices are now physically connected (either through actual wires (Ethernet) or through radio waves (Wi-Fi)), they can't actually talk to each other yet, because they have no way of knowing where to send the message to.
That's where the **Internet Protocol** (IP) comes in. Every device on the network has a personal IP address. The DHCP server (Dynamic Host Configuration Protocol Server) makes sure that these addresses are unique.

This means that you can now send a message to a specific address.

There are two versions of the Internet Protocol: IPv4 and IPv6. IPv6 is an improved version of IPv4 and has much more addresses than IPv4 (because there are much more devices than available IPv4 addresses). In this article, we'll only talk about IPv4 addresses, since most LANs still use them.

The IP address consists of 4 numbers, for example *192.168.1.5* is a valid IPv4 address. It actually consists of two parts: the first part is *192.168.1*, this is the address of the local network. The last digit, 5 in this case, is specific to the device.

By using IP addresses, we can find the ESP8266 on the network, and send messages to it. The ESP can also find our computer or our phone, if it knows their respective IP addresses.

***Sub-net mask** (optional)*

This subdivision of the IP address is determined by the sub-net mask, often written as 255.255.255.0. You can see that it consists of four numbers, just like the IP address. If a part of the sub-net mask is 255, it means that the corresponding part of the IP address is part of the network address, if it's 0, the corresponding IP part is part of the address of the specific address. A different notation to "IP: 192.168.1.5, sub-net mask: 255.255.255.0" would be "192.168.1.5/24", because the binary representation of the sub-net mask is 11111111.11111111.11111111.00000000, and it has 24 ones.
If you want to know more about sub-nets, I'd recommend you to read the Wikipedia article.
(A quick tip to help you remember: it's called the sub-net mask, because if you perform a bitwise AND operation on the IP address and the sub-net mask (i.e. use the sub-net mask as a mask for the IP address), you get the address of the sub-net.)

***MAC addresses and ARP** (optional)*

It is actually impossible to send packets directly to another machine using only the IP address. To send a packet to a specific device on the LAN (Wi-Fi or Ethernet), you have to know its MAC-address. The MAC address is a unique number that is unique for every network device, and it never changes, it's hardwired in the network chip. This means that every ESP8266, every network card, every smartphone ... ever made, has a different MAC address.
So before the ESP can send a packet to your smartphone for example, it has to know its MAC address. It doesn't know this yet, the ESP only knows the IP address of the smartphone, say 192.168.1.6. To do this, the ESP sends a broadcast message (i.e. a message addressed to all devices on the LAN) saying "I'm looking for the MAC address of the device with the IP address 192.168.1.6". The ESP also includes its own IP and MAC address with the message. When the smartphone receives this broadcast message, it recognizes its own IP address, and responds to the ESP by sending its own MAC address. Now the ESP and the phone both know each other's IP and MAC addresses, and they can communicate using IP addresses. This method is called the Addres Resolution Protocol, or ARP.

What about the Internet?

As you might have noticed, I only talked about the *local* area network, these are the computers in your own house. So how can the ESP8266 communicate with the Internet, you may ask? Well, there's a lot of network infrastructure involved in 'The Internet', and they all obey the IP rules, to make sure most of your packets arrive at there destination. It's not that simple of course, there's a lot of things going on, like routing and Network Address Translation (NAT), but that falls outside the scope of this article, and it's not really something most people have to worry about.

TL;DR

The Internet layer uses IP addresses in order to know where it should send the data. This means that two devices can now send packets of data to each other, even over the Internet.

The Transport layer

The different devices in the network do their best to deliver these IP packets to the addressee, however, it's not uncommon for a packet to get lost, so it will never arrive. Or the packet might get corrupted on the way: the data is no longer correct. IP also can't guarantee that the packets arrive in the same order they were sent in. This means that we can't *reliably* send messages yet by only using the link and the Internet layer, since we can never know when and whether a packet will arrive, or know for certain that a received packet is correct.
We need a third layer on top of the Internet layer: the Transport layer.

There are mainly two protocols that make up this third layer: the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

- **TCP** makes sure that all packets are received, that the packets are **in order**, and that corrupted packets are re-sent. This means that it can be used for communication between multiple applications, without having to worry about **data integrity** or packet loss. This is why it's used for things like downloading webpages, sending email, uploading files etc.
- **UDP** on the other hand, doesn't guarantee that every packet reaches its destination, it does **check for errors** however, but when it finds one, it just destroys the packet, without re-sending it. This means that it's **not as reliable as TCP**, but it's faster, and has a much **lower latency**, because it doesn't require an open connection to send messages, like TCP does. That's why it's used in voice and video chats, and for example in online games.

If you want to know more about the differences between TCP and UDP, check out this video.

TL;DR

The IP protocol is not reliable, and has no error checking. TCP solves this by re-sending lost or corrupt packages, and orders packets that are received in the wrong order. UDP also checks for corrupt packages, but doesn't re-send them, so it has less latency than TCP.

The Application layer

We now have reliable communication using TCP, but there's still one problem. Think of it this way: you are sending a letter, and TCP guarantees that it will arrive at its destination, but if the receiver doesn't understand the language it's written in, he won't know what to do with it.
In other words, we need a fourth layer of protocols, for two programs to be able to communicate with each other.
There's lots of different protocols out there, but we'll mostly focus on the protocols for web servers and browsers.

HyperText Transfer Protocol

The HyperText Transfer Protocol, or HTTP, is the protocol (cfr. language) that is used by both web servers and web clients in order to communicate. It uses text to perform send requests and responses from the client to the server and back again.
For example, when you type http://www.google.com into the address bar of a web browser (client), it will send an HTTP GET request to the Google web server. The server understands this HTTP request, and will send the Google webpage as a response. Or when you upload an image to Instagram, your browser sends an HTTP POST request with your selfie attached to the Instagram server. The server understands the request, saves the image and adds it into the database, sends the URL of the new image back to your browser, and the browser will add the image on the webpage.
As you can see, neither the client nor the server has to worry about the integrity of the messages they send, and they know that the recipient understands their language, and that it will know what to do with a certain HTTP request.
Most modern sites use a secure version of HTTP, called HTTPS. This secure connection encrypts the data, for security reasons. (You don't want anyone reading the packets from your mail server, or the packets you sent to your bank, for instance.)

WebSocket

HTTP is great for things like downloading webpages, uploading photos etc. but it's quite slow: every time you send an HTTP request, you have to start a new TCP connection to the server, then send your request, wait for the server to respond, and download the response. Wouldn't it be great if we didn't have to open a new connection every time we want to send some data, and if we could send and receive data at the same time at any moment we'd like? That's where WebSocket comes to the rescue: you can keep the TCP connection with the server open at all times, you get perfect TCP reliability, and it's pretty fast.

Open Sound Control

HTTP and WebSocket both use TCP connections. What if I want lower latency? Well, Open Sound Control, or OSC, uses UDP to send small pieces of data, like ints, floats, short text etc ... with very low latency. It was originally designed for controlling low latency audio applications, but it's a very flexible protocol, so it's often used for low-latency tasks other than audio control.

Domain Name System

As mentioned before, you can only send a message to another computer if you know its IP address. But when you browse the Internet, you only know a website's domain name (e.g. www.google.com). Your computer uses the Domain Name System to translate this domain name to the right IP address. More on this later.

Sources

- https://en.wikipedia.org/wiki/Internet_protocol_suite
- https://en.wikipedia.org/wiki/Port_(computer_networking)
- https://en.wikipedia.org/wiki/Transmission_Control_Protocol
- https://en.wikipedia.org/wiki/Internet_Protocol
- https://en.wikipedia.org/wiki/User_Datagram_Protocol