# Chapter 5: Connecting to Access Points (AP)

## Objective

At the end of chapter 5 you will understand the fundamentals of being a WiFi Station (STA) and connecting to a WiFi Access Point (AP). You will have an introduction to the TCP/IP Networking stack and you will have basic understanding of the first three layers of the stack i.e. physical, datalink and network layers of WiFi which enable connections and encryption. You will understand some of the basics of IP networking (addresses, netmasks). Lastly, you will understand the role of the WICED Device Configuration Table (DCT).

**Most importantly, you will be able to use WICED to connect your IOT device to a WiFi Network.**

## Time: 1 Hr

## Fundamentals

### TCP/IP Networking Stack

Almost all complicated systems manage the overall complexity by dividing the system into layers. The “Network Stack” or more accurately, the “TCP/IP Network Stack” is exactly that, a hierarchical system for reliably communicating over multiple networking mediums (WiFi, Ethernet, etc.) Each layer isolates the user of that layer from the complexity of the layer below it, and simplifies the communication for the layer above it. You might hear about the [OSI Network Model](https://en.wikipedia.org/wiki/OSI_model) which is another, similar way to describe networking layers, however, it is easier to envision IP networks with TCP/IP model.

Each layer takes the input of the layer above it and then embeds that information into one or more of the Protocol Data Units (PDUs) of that layer. A PDU is the atomic unit of data for a particular layer. e.g. the Datalink Layer takes an IP packet and divides it up into 1 or more WiFi Frames. The physical layer takes Datalink Layer Frames and divides them up into bits.

|  |  |  |  |
| --- | --- | --- | --- |
| [**Layer**](https://en.wikipedia.org/wiki/Internet_protocol_suite) | [**Protocol**](https://en.wikipedia.org/wiki/Internet_protocol_suite) | [**Protocol Data Unit**](https://en.wikipedia.org/wiki/Protocol_data_unit) | **Comment** |
| [Application](https://en.wikipedia.org/wiki/Application_layer) | [DNS](https://en.wikipedia.org/wiki/Domain_Name_System), [DHCP](https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol), [MQTT](https://en.wikipedia.org/wiki/MQTT), [HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol), etc. | Data | The layers below the application provide the mechanism to trade useful data. The application layer is the actual protocol to do something useful in the device e.g. HTTP (get or put data), DNS (find a IP address from a name), MQTT (publish or subscribe) etc. |
| [Transport](https://en.wikipedia.org/wiki/Transport_layer) | [TCP](https://en.wikipedia.org/wiki/Transmission_Control_Protocol)  [UDP](https://en.wikipedia.org/wiki/User_Datagram_Protocol) | (TCP) [Segments](https://en.wikipedia.org/wiki/Transmission_Control_Protocol#TCP_segment_structure)  (UDP) [Datagram](https://en.wikipedia.org/wiki/Datagram) | Reliable, ordered, error checked stream of bytes – think of it as a pipe between computers  A connectionless datagrams (blobs of data) |
| [Network](https://en.wikipedia.org/wiki/Network_layer) | [IP](https://en.wikipedia.org/wiki/Internet_Protocol) | [Packets](https://en.wikipedia.org/wiki/Network_packet) | An IP network can send and receive IP packets with source and destination IP addresses to anywhere on the Internet. |
| [Data-Link](https://en.wikipedia.org/wiki/Data_link_layer) | [802.11 MAC](https://en.wikipedia.org/wiki/IEEE_802.11) | [Frame](https://en.wikipedia.org/wiki/Frame_(networking)) | A frame is the atomic unit of transmission in the network. All of the data from the layers above are broken into frames by the data link layer.  Converts bits into unencrypted frames. This layer only communicates on the Local Area Network |
| [Physical](https://en.wikipedia.org/wiki/Physical_layer) | 802.11([a](https://en.wikipedia.org/wiki/IEEE_802.11#802.11a_.28OFDM_waveform.29),[b](https://en.wikipedia.org/wiki/IEEE_802.11#802.11b),[g](https://en.wikipedia.org/wiki/IEEE_802.11#802.11g),[n](https://en.wikipedia.org/wiki/IEEE_802.11#802.11n),[ac](https://en.wikipedia.org/wiki/IEEE_802.11#802.11ac)) | Bits | Sends and receives streams of bits over the WiFi Radio |

### (Physical/Datalink) WiFi Basics

There are two ends of a WiFi network, the Station (i.e. the IOT device) and the Access Point (i.e the wireless router). In order for a Station to connect to a WiFi Network, it must have the following information **SSID**, **Band**, **Channel** and **Encryption**. In addition, in order to send data, all WiFi Datalink Frames are all labeled with the source and destination **MAC Addresses**.

[**SSID**](https://en.wikipedia.org/wiki/Service_set_(802.11_network)) **(the name of the wireless network)**

SSID stands for Service Set Identifier. The SSID is the network name and is composed of 0-32 bytes a.k.a octects. The name does not have to be human readable (e.g. ASCII) but because it is uncoded bytes it is effectively case sensitive (be careful).

**Band (either 2.4GHz or 5GHz)**

WiFi radios encode 1’s and 0’s with one of a number of different schemes depending on the type of WiFi (a,b,g,n, ac,ax). The type of encoding doesn’t matter to your IOT application as the chip takes care of it. The data is then transmitted into the 2.4GHz or 5GHz band (which band is important).

**[Channel number](https://en.wikipedia.org/wiki/List_of_WLAN_channels)**

The available channels are band (2.4GHz) and geographically (where in the world) specific. 2.4GHz is pretty simple, there are channels 1-14 with 1-11 available all over the world. 5GHz is way crazier and you need to look at the table depending on the region.

**However, from the station point of view (and therefore this class) none of this matters as when you try to join an SSID the WICED SDK will scan all the channels looking for the correct SSID.**

[**Encryption (open, wep, wpa, wpa2)**](https://en.wikipedia.org/wiki/Wireless_security)

In order to provide security for WiFi networks it is common to use data link layer encryption. The types of network encryption are Open (i.e. no security), [Wireless Equivalent Privacy (WEP)](https://en.wikipedia.org/wiki/Wired_Equivalent_Privacy) which is not very secure, [WiFi Protected Access (WPA)](https://en.wikipedia.org/wiki/Wi-Fi_Protected_Access) and WPA2 which has largely displaced WPA (you must support WPA2 to use the WiFi logo on your product). There are two version of WPA1/2 one called “Personal” or “PreShared Key” and one called “Enterprise”.

WEP and WPA PSK both use a password—called a key—to encrypt the data. The WEP encryption scheme is very hackable. The PSK key scheme of WPA is very secure as it uses [AES](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard) however, sharing keys is a painful unsecure process because it means that everyone has the same key. To solve the key distribution problem, most enterprise networking solutions, use WPA2 Enterprise which requires use of a [RADIUS](https://en.wikipedia.org/wiki/RADIUS) server to handle authentication of each station individually.

**Enterprise security is an oncoming crisis for the IOT market and is a differentiated feature of WICED.**

[**Media Access Control (MAC) Address**](https://en.wikipedia.org/wiki/MAC_address)

The WiFi MAC address is a 48-bit unique number that is assigned by the manufacturer (Cypress). In order for the Datalink layer to send a frame it must address the frame with a source and destination MAC address. Other devices on the network will only pass frames into the higher levels of the stack that are addressed to them. Remember that the Datalink Layer does not know anything about the higher layers (e.g. IP)

The datalink layer needs to be able to figure out the MAC address of a particular IP Address in order to send it out on the WiFi network. In order to figure out this mapping there is a protocol called Address Resolution Protocol (ARP).

[**ARP**](https://en.wikipedia.org/wiki/Address_Resolution_Protocol)

Inside of every device there is an ARP table that has a map of MAC Address to IP address. In order to discover the MAC address of a IP address an “ARP request” is broadcast to the network. All devices attached to a network listen for ARP requests. If you hear an ARP request with your IP address in it, you respond with your MAC address. From that point forward both sides add that information to their ARP table (and in fact if you hear others ARPing you can update your table as well). The brilliant part of this scheme is that if you ARP for an IP address that is not on your network, the router will respond with its MAC address (the subject of the next section).

### IP Networking



**The Internet** is a mesh of interconnected **IP networks**. **The Cloud** is all of the internet that is accessible by your network, but may also mean servers that are attached to a network somewhere on the internet.

All **devices** on the Internet have a legal [**IP address**](https://en.wikipedia.org/wiki/IP_address) and belong to an (IP) **Network** that is defined by a **Netmask**. **Routers** are devices that connect IP networks by taking IP packets from one network and forwarding them along to the correct next network. This is a complicated task and is outside of the scope of this class but is the reason which Cisco is worth $151B. For the purposes of this class you should just think that once you have connected to the network that your packets are magically transported to the other end.

An IP Address uniquely identifies an individual device with a 32-bit number that is general expressed as four hex-bytes separated with colons. E.g. 192.168.15.7. IP addresses is divided into two parts the network address (which is the first x number of bits) and the client address which are the last 32-x bits. The netmask defines the split of network/client. E.g. the netmask for 192.168.15.\* is 255.255.255.0

An [IP Network](https://en.wikipedia.org/wiki/Subnetwork) (sometimes called an IP Subnetwork) is the collection of devices that are all share the same network address e.g. all of the devices on 192.168.15.\* (netmask 255.255.255.0) are all part of the same IP Network.

Most commonly IP addresses for IOT type devices are assigned dynamically by a Dynamic Host Control Protocol (DHCP) server. To dynamically assign a DHCP address you first send a UDP broadcast datagram asking for an IP address. When a DHCP server hears the request it responds with the required information. DHCP is integrated into WICED and handles this exchange of information for you automatically when enabled.

### Device Configuration Table (DCT)

The device configuration table is a section of the WICED flash with a predefined format that is used to store fundamental information about the system (i.e. client ap ssid, client ap passphrase etc.). This information is then used by the WICED firmware to do the right thing e.g. wiced\_network\_up reads the network information from the DCT and connects to that network.

The table is built during the make process and written into the flash along with the programming of the application. It can also be modified on the fly by your app (and written).

When building a WICED App you can either use the default DCT or you can make a customer one. To preconfigure the DCT table you need to create a .h file (generally called wifi\_config\_dct.h with the correct #defines. You then need to add “WIFI\_CONFIG\_DCT\_H := wifi\_config\_dct.h” to the makefile.

**You can get a template for the file in the directory “include/default\_wifi\_config\_dct.h”**



The device can go into three modes which you can see in the table above: Configuration AP (lines 4-7), Soft AP (10-13), and Client Mode (lines 17-23). It is also possible to have multiple network interfaces as well as support WiFI and Ethernet (line 26). For the purposes of this chapter we will only be a CLIENT so you will only need to touch 20-23.

To find the definition (or possible definitions) of the #defines you can highlight and right click, select “open declaration”. For example, if you select “WICED\_SECURITY\_OPEN” you will see



### Documentation

The relevant documentation for the networking management functions are in the WICED-SDK documentation under Components🡪Management🡪Network Management



Functions that allows you to interface with the IP networking are available in the documentation under Components🡪IP Communication🡪Raw IP and Components🡪IP Communication🡪DNS Lookup



### Introducers

One of the main difficulties of getting IOT devices connected to the network is configuring the network information. There are a number of possible strategies for solving this problem including:

* Connecting to the IOT device with Bluetooth and then using a phone based App to configure the device
* Connecting the IOT device to a computer with USB or Serial connection and then configuring the device with a computer based application
* Starting a WiFi Access Point with a web server on the IOT device, then connecting to the IOT device with a computer or a cellphone.
* Preprogramming the device with the required information

WICED supports all of these methods. In this class we use the pre-programmed method in the interest of simplicity and time. However, each of the other methods are demonstrated in the sample applications.

## Exercise(s)

### 01 Create an App that attaches to an open network, have the LED blink red on failure and green on success

1. Make a new folder to hold the app
2. Copy the template default\_wifi\_config\_dct.h and name it wifi\_config\_dct.h
3. Modify the wifi\_config\_dct.h
4. Create and modify the makefile (don’t forget to add the WIFI\_CONFIG\_DCT\_H)
5. Create the project name.c (use the function wiced\_network\_up to read the DCT and start the network)
6. Check the error codes and do the appropriate blinking

### 02 Modify (02) to attach to a WPA2 PSK network

### 03 Modify (02) to print out networking information

* IP address
* Netmask
* Router
* The IP address of [www.cypress.com](http://www.cypress.com)
* MAC Address of your device

### 04 Create an application that can switch between two different SSIDs

1. Start the Application and connect to the SSID that is currently in the DCT
2. If the user presses a button, switch SSIDs, write the DCT, print diagnostics
3. Demonstrate that the SSID is saved by switching then resetting the device

### 05 Create an application that the user can enter the SSID and passphrase and write to the DCT

1. Start the Application and connect to the SSID that is currently in the DCT
2. Wait for the user input, take and validate the user input, write to the DCT
3. Reattach to the network
4. Demonstrate that it is saved by rebooting

## Related Example “Apps”

## Known Errata + Enhancements + Comments

I don’t know what you can add to the DCT other than “$(NAME)\_RESOURCES”.