Lecture 16

LEDC and WiFi on ESP32

Agenda

01. LEDC on ESP32
02. Wireless protocols and communcations issues
03. TCP/IP on ESP32
04. UDP on ESP32
05. Wifi modes (Access Point / Station / DHCP / ESP-NOW)

Stuff

- **Lab 3.2:** It's okay to use USB power (teensy VCC) with the SG90 servos. But if you use a bigger servo, use a separate power supply (or battery).
- **Lab 3.2.2** hint: Look at chapter 29 of Atmega docs starting at page 383
- For grading questions, please use the "grading_questions" channel on Piazza and include the section it involves: e.g., *Lab 1.1 and 1.2 regrade request.*
- COVID 2-week testing on-going. 7231 red-passes are going out. 4467 grads, 2764 undergrads. You must be tested before end of this week or lose access to GMLab.

Waldo Dance Choreography

Types:

- Human-body
- Head
- Helicopter/plane
- Arms/wings

- Try to mimic

https://youtu.be/932n5qEyOsk?t=290 for 15 sec

Fixed-width Integer C and C++ Types

Original name	C++11 name	# bits	Lower bound	Upper ebound
char signed char	**int8_t**	8	-127	127
unsigned char	**uint8_t**	8	0	255
short (default int ATmega32U4) short int signed short signed short int	**int16_t**	16	-32,767	32,767
unsigned short unsigned short int	**uint16_t**	16	0	65,535
long (default int ESP32) long int signed long signed long int	**int32_t**	32	-2,147,483,647	2,147,483,647
unsigned long unsigned long int	**uint32_t**	32	0	4,294,967,295

01

ESP32 LEDC

For ESP32 instead of `analogWrite()` use LEDC routines

- LEDC has 16 channels, not assigned to pins
  - Need to attach them to a pin that drives an LED or PWM output.
  - Prefer ledc channels 0-7 which use ESP32 high speed mode.
- Huge range of PWM frequency and resolution, within limits
  - Frequency * Resolution must be **< 80 MHz**
    - Example: for PWM resolution of 1000, max PWM freq is 80Khz
  - Frequency * Resolution must be **> 1 KHz**
    - Example: too fast for 1 Hz and resolution of 256
- For reference `analogWrite` on Atmega32 (i.e., Arduino Uno) has freq 490Hz and 8-bit resolution (freq*res = 3920)

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/ledc.html

ESP32 LEDC routines

Example: channel 0 and pin 21. 20 bits -> 2^20 = 1,048,576

```
double      ledcSetup(uint8_t channel, double freq, uint8_t resolution_bits);
```

- Setup a `channel` to output a square wave at frequency `freq` (in Hz) that has a PWM resolution with `resolution_bits.` Returns the value of the frequency actually set.

```
void        ledcAttachPin(uint8_t pin, uint8_t channel);
```

- Setup a `channel` to output a square wave at a specific GPIO `pin`.

```
void        ledcWrite(uint8_t channel, uint32_t duty);
```

- Set the PWM duty cycle of `channel` as a proportion of resolution_bits. For example, `duty` in percentage is `((1<<resolution_bits-1) * duty / 100;`

Extra routines not usually used in MEAM510

```
uint32_t    ledcRead(uint8_t channel);          // returns duty cycle
double      ledcReadFreq(uint8_t channel);      // returns set frequency
void        ledcDetachPin(uint8_t pin);         // turns off pin
```

Ledc code found in Arduino/hardware/espressif/esp32/cores/esp32/esp32-hal-ledc.c

Q1a We need a 60Hz PWM frequency with a resolution of about 1 million. Can we get that with LEDC?

Frequency * Resolution must be < 80 MHz

Example: for PWM resolution of 1000, max PWM freq is 80Khz

Frequency * Resolution must be > 1 KHz

Example: too fast for 1 Hz and resolution of 256

Q1b What Arduino setup code will generate this ?

[use channel 0 and pin 21. Use 20 bits for one million]

LEDC Example

```
#define LEDC_CHANNEL       0 // use first of 16
#define LEDC_RESOLUTION_BITS  13
#define LEDC_RESOLUTION  ((1<<LEDC_RESOLUTION_BITS)-1)
#define LEDC_FREQ_HZ     5000
#define LED_PIN           2

// can set syntax to be like analogWrite() with input[ 0 : valueMax ]
void ledcAnalogWrite(uint8_t channel, uint32_t value, uint32_t valueMax = 255) {
  uint32_t duty =  LEDC_RESOLUTION * min(value, valueMax) / valueMax;
  ledcWrite(channel, duty);  // write duty to LEDC
}

void setup() {  // Setup timer and attach timer to a led pin
  ledcSetup(LEDC_CHANNEL, LEDC_FREQ_HZ, LEDC_RESOLUTION_BITS);
  ledcAttachPin(LED_PIN, LEDC_CHANNEL);
}

void loop() {
  static int brightness = 0;  // how bright the LED is
  static int fadeAmount = 5;  // how much to fade by

  ledcAnalogWrite(LEDC_CHANNEL, brightness);
  brightness = brightness + fadeAmount;
  if (brightness <= 0 || brightness >= 255)
    fadeAmount = -fadeAmount;  // reverse the direction
  delay(30);
}
```

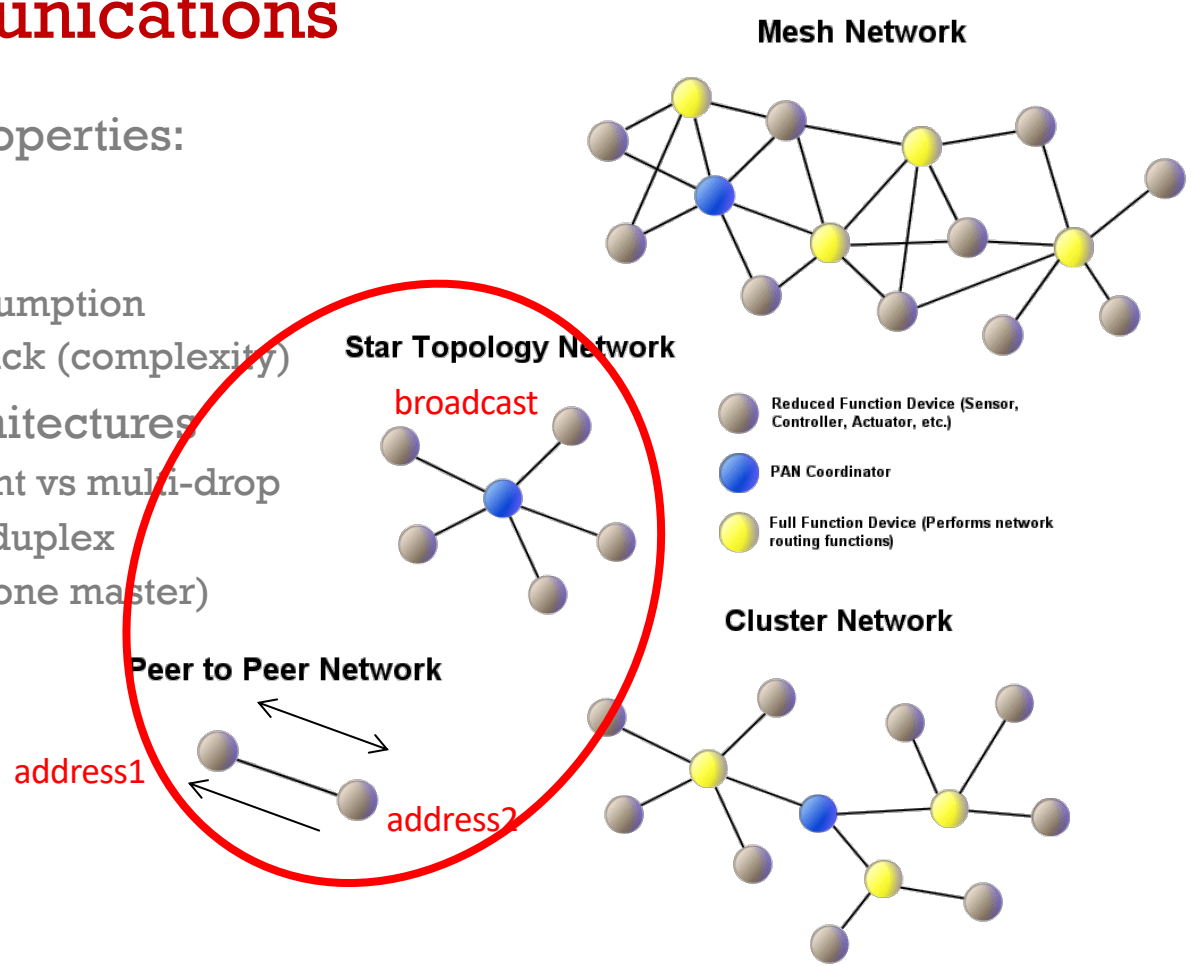
File on canvas -> resources -> ESP32 Arduino Lecture Examples

02

Wireless Communications

Serial Communications

- Important Properties:
  - Data rate
  - Distance
  - Power consumption
  - Protocol Stack (complexity)
- Network architectures
  - Point to point vs multi-drop
  - Half vs full duplex
  - Broadcast (one master)
  - Mesh
  - Star
  - Cluster



Communication comparison (sorted by speed)

Method	net	Speed [kbs]	Range [m]	Power [mW]	Wirelss Freq	notes
Ethernet	bus	1000 to 1000000	100's	~500	wired	Complex stack (TCP/IP etc.)
WiFi (802.11) (b,g,n,ac)	wlan	1000 to 150000	70 - 200	~500	2.4Ghz, 5GHz	Complex stack (TCP/IP etc.)
ESP-NOW	Star pp	100-1800	~200	~500	2.4Ghz	Low latency, max 250 byte packet, upto 20 peers
Bluetooth		)00 to 00	10	~1000	2.4Ghz	Complex stack, wake 3s (latency 0.1s)
BLE (BT 4.0)	scatternet	125-2000	10	10-500	2.4Ghz	Low power std, fast startup (latency 0.006)
SPI or I2C	multidrop	1 to 1000	~10	5	wired	Simple synchronous
UART	p-p	110	~20	5	wired	Simple async 3-wire
Zigbee (xbee)	Star pp	20-250	10-100	~30	2.4Ghz, 915MHz	Low power, low speed, cost~$2, wake 15ms
Toy RC	p-p hd	0.07	10		27MHz 49MHz	Low cost

on most phones and laptops

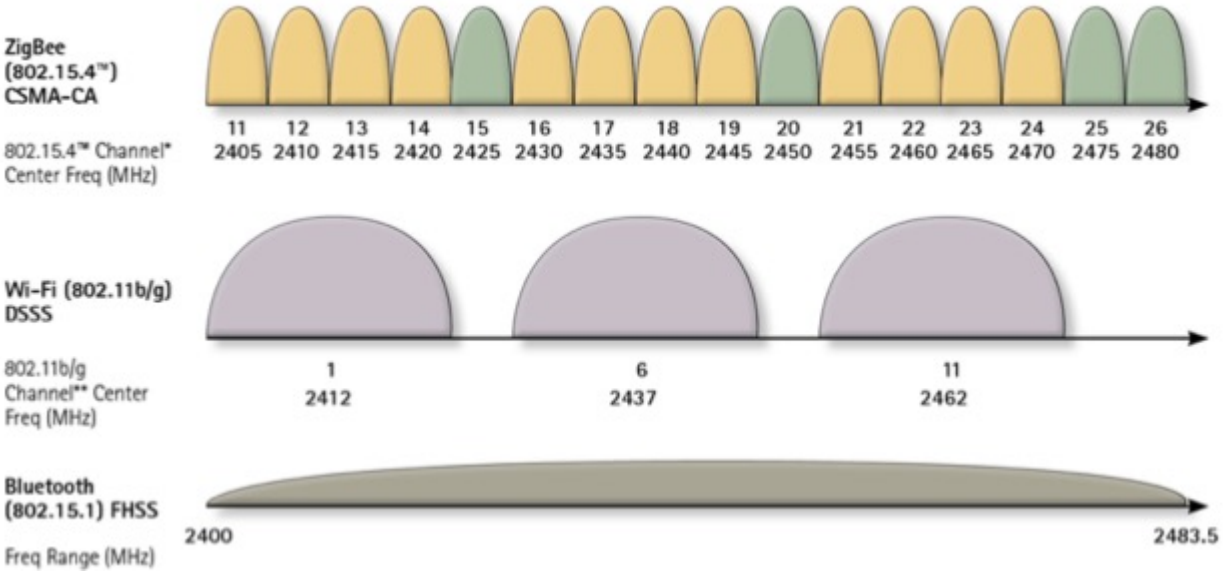
We'll cover later

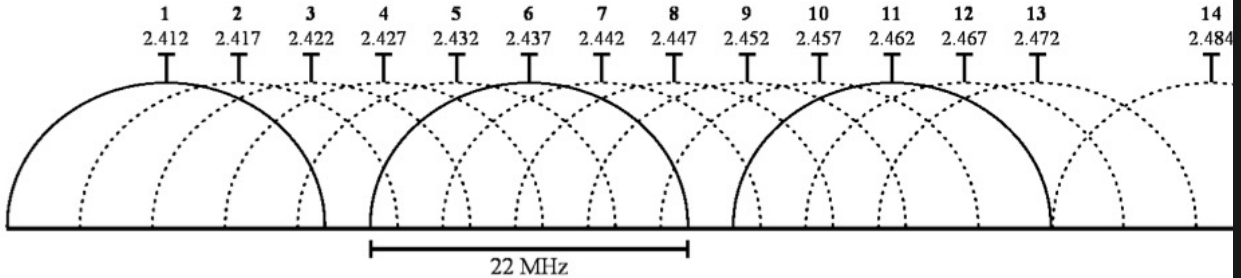
Monitor or usbprint

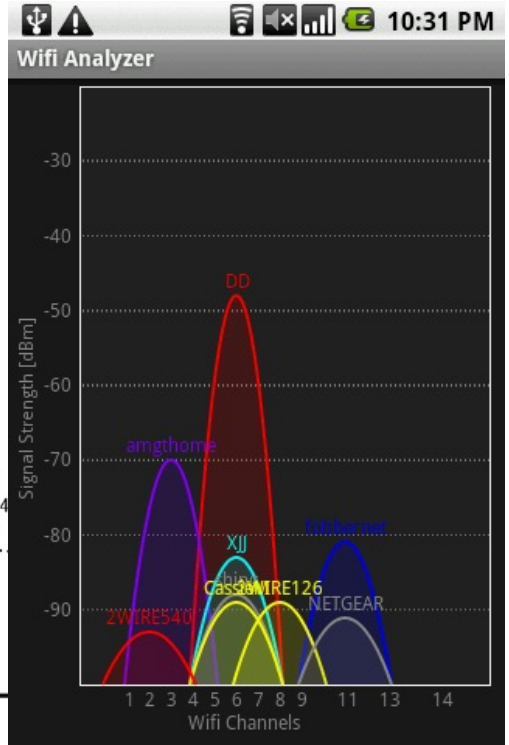
2.4GHz Freq Interference

- WiFi 802.11 2.4GHz and 5GHz
- Microwave Ovens run at 2.45GHz (but on 50% duty)

2.4GHz Freq Interference







Industrial, Scientific and Medical (ISM) Radio Band

- ISM unlicensed freq bands

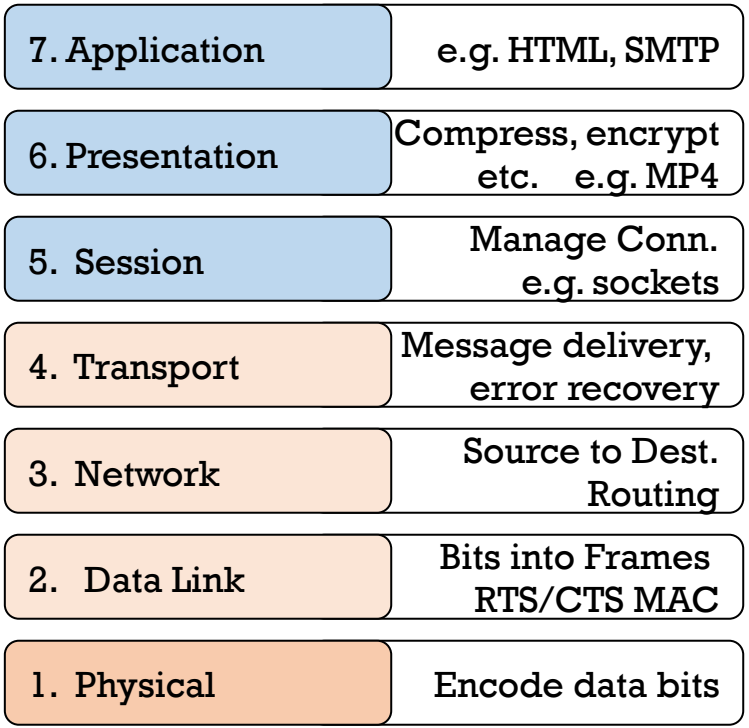
Frequency range		Bandwidth	Center frequency	Availability	
6.765 MHz	6.795 MHz	30 kHz	6.780 MHz	Subject to local acceptance	
13.553 MHz	13.567 MHz	14 kHz	13.560 MHz	Worldwide	-RFID
26.957 MHz	27.283 MHz	326 kHz	27.120 MHz	Worldwide	-RC Toys
40.660 MHz	40.700 MHz	40 kHz	40.680 MHz	Worldwide	-RC Toys
433.050 MHz	434.790 MHz	1.74 MHz	433.920 MHz	Region 1 only and subject to local acceptance	
902.000 MHz	928.000 MHz	26 MHz	915.000 MHz	Region 2 only (with some exceptions)	-Cordless phones
2.400 GHz	2.500 GHz	100 MHz	2.450 GHz	Worldwide	-Microwave ovens
5.725 GHz	5.875 GHz	150 MHz	5.800 GHz	Worldwide	
24.000 GHz	24.250 GHz	250 MHz	24.125 GHz	Worldwide	-Amateur satellite
61.000 GHz	61.500 GHz	500 MHz	61.250 GHz	Subject to local acceptance	
122.000 GHz	123.000 GHz	1 GHz	122.500 GHz	Subject to local acceptance	
244.000 GHz	246.000 GHz	2 GHz	245.000 GHz	Subject to local acceptance	

http://en.wikipedia.org/wiki/ISM_band

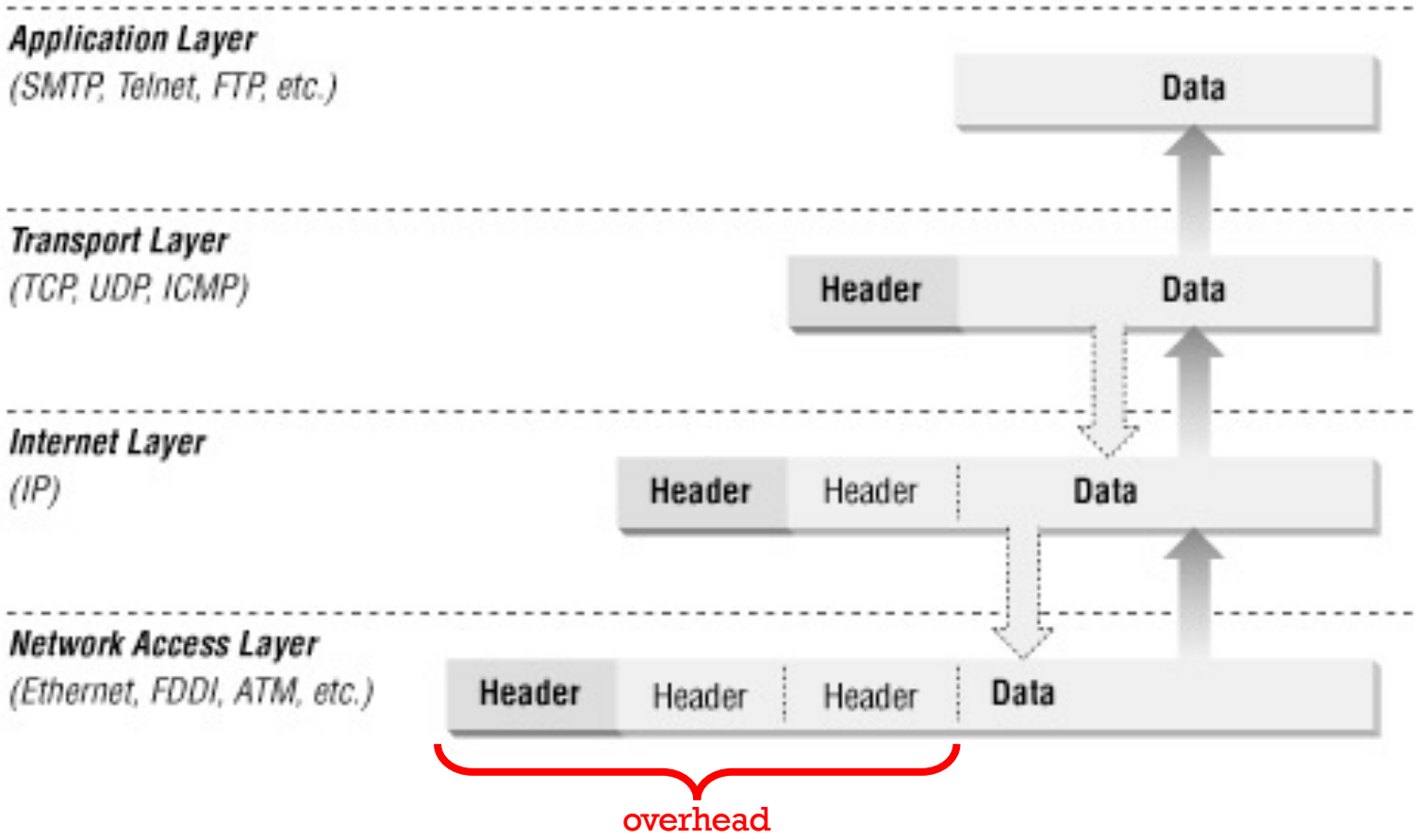
OSI – ISO 7 layer model

Open Systems Interconnection at the International Standards Organization

- Layer 1 and some 2 :
  - RS232 w/UART (async serial – old std)
  - SPI, I2C, I2S (sync serial)
  - RS485 w/UART (multi-drop)
- Layer 3
  - Internet Protocol (IP)
- Layer 4
  - Transmission Control protocol (TCP)
  - User Datagram Protocol (UDP)



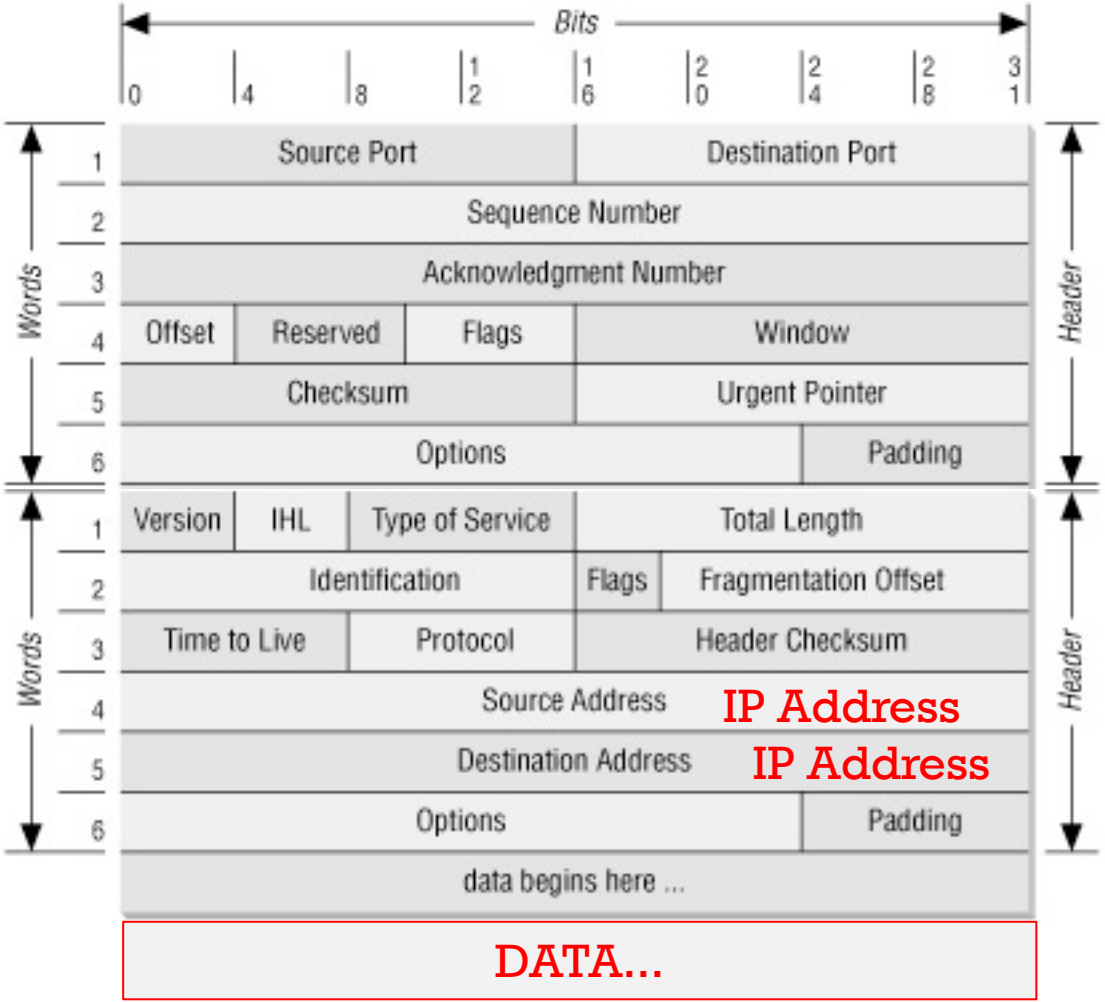
Data Encapsulation



Packets

TCP header
(20bytes + option bits)

IP header
(20bytes + option bits)



Latency vs bandwidth vs data rate

Case 1:

- Data is two bytes long
- We send 100 packets per sec.
- Assume TCP no options

Case 2:

- Data is 200 bytes long
- We send one packet per sec.
- Assume TCP no options

In chat for each case

Q2a: What is the data rate (amount of *data* we send per second)?

Q2b: What is the bandwidth required for each case?

Q2c: What is the avg latency (avg time before we receive data)?

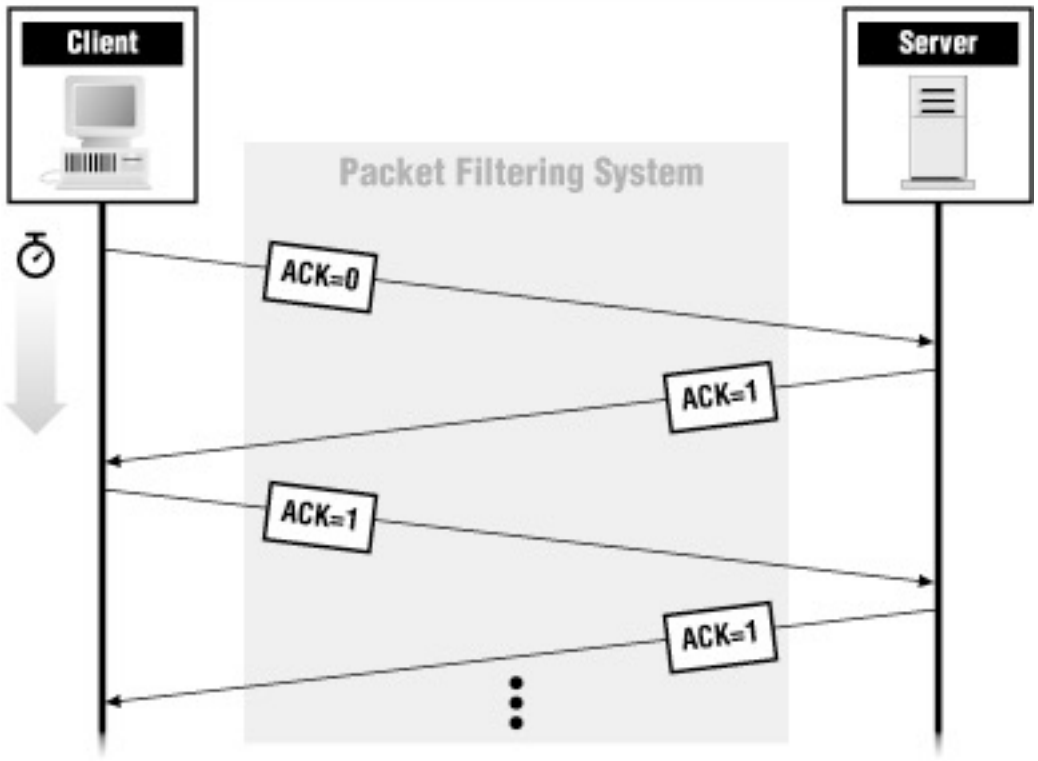
Q2d: What is the data efficiency (databits / total bits transmitted)

TCP (Transport Control Protocol)

- Receiver acknowledges packets – if not, sender resends after timout.

Reliable Transport:

- Receive in order
- Receive all data
- Receive no duplicates



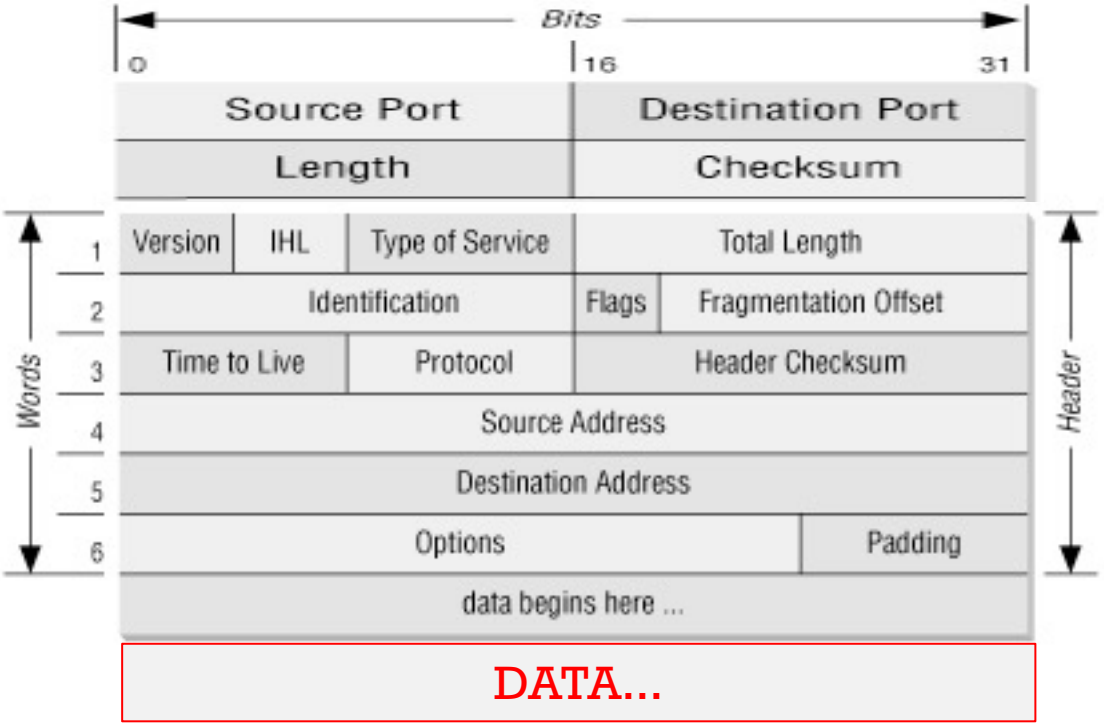
UDP Packets (~~User~~ Datagram Protocol)

Unreliable

- Much simpler (no verification, no resend)

UPD header
(8 bytes)

IP header
(20 bytes)



WiFi Quiz/summary

Q3: Which protocol would be better for streaming live video:

A) TCP or B) UDP ?

Q4: Which protocol would be better for bank transactions?

A) TCP or B) UDP ?

Q5: Which protocol do movies usually use?

A) TCP or B) UDP ?

There are other wireless protocols, we won't use in this class

03

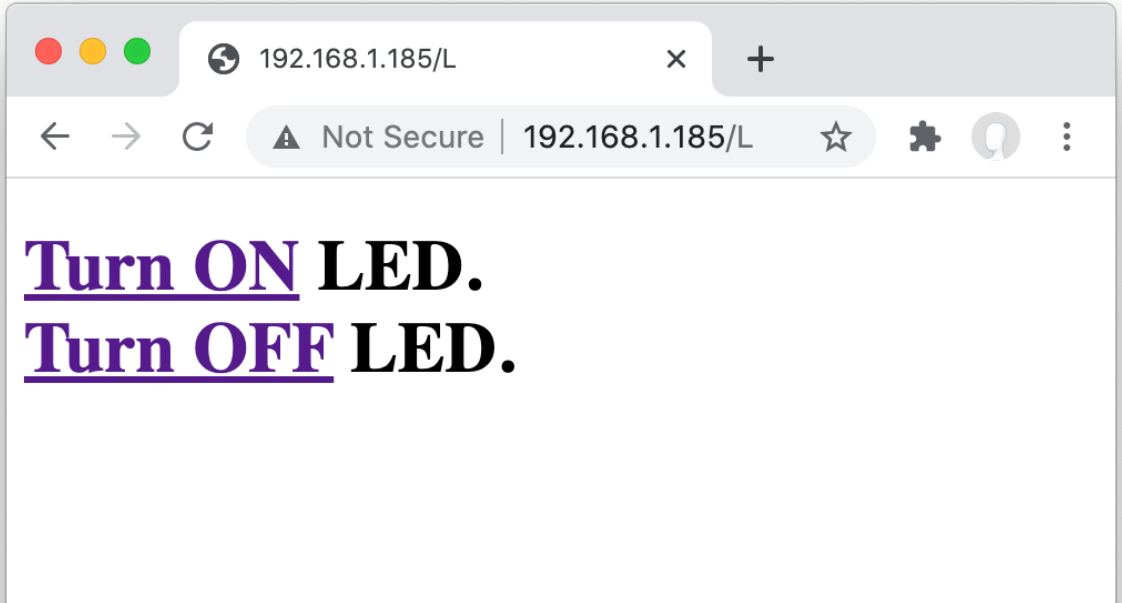
TCP-IP on ESP32

ESP32 Internet of Things (IOT)

- Control from your smart phone or laptop.
- 2.4GHz 802.11 (transmit b/g) (receive b/g/n)
- Bluetooth 4.2 (BLE) (we won't go into BLE for this class)
- Full ISO stack (HTML)
- Caveats:
  - Limited space and HTML code
  - ESP32 WiFi sometimes has lag (0.5 – 4 sec)
  - Only from local net (port-forwarding)
  - http://www.wikihow.com/Set-Up-Port-Forwarding-on-a-Router

Web server on a chip

- You can create a webpage that runs on the ESP32
  - The oscillosorta is one example.
- We'll go through a simple example with two arduino parts
  - `setup()`
  - `loop()`



Sample Web Code [part 1 of 2]: setup()

```
#include <WiFi.h>

const char* ssid = "yourSSID";
const char* password = "yourpassword";

WiFiServer server(80);   // port 80 is standard for websites

void setup() {
  Serial.begin(115200);
  WiFi.softAP(ssid,password);
  Serial.print("AP IP address: HTML//");
  Serial.print(WiFi.softAPIP());
  server.begin();
}
```

New include file

SSID + passwd

Std setup

Wifi Access Point setup
Uses DHCP

Sample Web Site Structure [part 2 of 2]: loop()

```
void loop(){
  WiFiClient client = server.available(); // loop until we have a client
  if (client) {
    while (client.connected()) {          // loop while connected
     if (client.available()) {            // if client has a request

      // Load the request into a String buffer one byte at a time
      // if the request is finished
      //   post the HTML code to be displayed
      //   interpret the string request and process

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

Sample Web Code: loop()

```
void loop(){
  WiFiClient client = server.available();
  if (client) {
    String currentLine = "";       // incoming data
    while (client.connected()) {
      if (client.available()) {    // if bytes to read
        char c = client.read();    // read a byte,
        if (c == '\n') {
          if (currentLine.length() == 0) {// if blank line
            client.print(body);
            break;     // exit while loop
          } else {
            if (currentLine.startsWith("GET /H "))
              digitalWrite(LEDPIN, HIGH);  // LED ON
            if (currentLine.startsWith("GET /L "))
              digitalWrite(LEDPIN, LOW);   // LED OFF
            currentLine = "";   // if new line but not blank, clear data
          }
        } else if (c != '\r') {  // if anything but a CR
          currentLine += c;      // add it to the end
        }
      }
    }
    client.stop();  // close the connection
  }
}
```

```
const char body[] PROGMEM = R"===(
 <!DOCTYPE html>
 <html><body>
 <h1>
 <a href="/H">Turn ON</a> LED.<br>
 <a href="/L">Turn OFF</a> LED.<br>
 </h1>
 </body></html>
)===";
```

Sample HTTP request messages

GET /L HTTP/1.1
Host: 777f4a18c8ef.ngrok.io
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Referer: http://777f4a18c8ef.ngrok.io/H
Upgrade-Insecure-Requests: 1
X-Forwarded-For: 172.58.207.29
X-Forwarded-Proto: http

921 bytes sent and only 2 bytes "/L" are actual data in this case.

GET /favicon.ico HTTP/1.1
Host: 777f4a18c8ef.ngrok.io
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Referer: http://777f4a18c8ef.ngrok.io/L
X-Forwarded-For: 172.58.207.29
X-Forwarded-Proto: http

Some HTML tags to play with

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>

<ul>
  <li>Bulleted list item </li>
  <li>Bulleted list item </li>
</ul>
```

```
// CSS to style the on/off buttons
<head><style>
html { font-family: Helvetica; display: inline-block;
        margin: 0px auto; text-align: center;
}
.button { background-color: #4CAF50; border: none;
        color: white; padding: 16px 40px;
        text-decoration: none; font-size: 30px;
        margin: 2px; cursor: pointer;
}
.button2 {background-color: #555555;
}
</style></head>
```

https://www.w3schools.com/tags/

ESP32 TCP Summary

- TCP can be used to make websites on ESP32
- You have c++ Arduino code on ESP32, you have web code (HTML, CSS, Javascript) that can run on websites, but are sourced on the ESP32.
- For small data transfers (like control info), HTML is not very efficient, but wifi is very fast so maybe efficiency is not important.
- We'll go into more web coding interfaces later

04

UDP on ESP32

UDP packets

- Send from one IPaddress to another IPaddress (can broadcast)
- No error recovery, no packet retransmit, no guarantee
- Typically used for streaming applications
- Much faster than TCP (less latency, similar bandwidth)

- Use "PacketSender" to debug.
  https://en.wikipedia.org/wiki/Packet_Sender

IP Address for you to use on your ESP32 (also on piazza)

Name	IP Address
Aizenberg, Noam	192.168.1.100
Ajmera, Yug	192.168.1.101
Alijaj, Martin	192.168.1.102
An, Zijian	192.168.1.103
Anbuchelvan, Ashwath	192.168.1.104
Aribidesi, Abudurazaq A	192.168.1.105
Armendariz Gonzalez, M	192.168.1.106
Avanesov, Ivan	192.168.1.107
Bhikule, Rohit Rajkumar	192.168.1.108
Bunn, David C	192.168.1.109
Candia, David V	192.168.1.110
Cao, Yinan	192.168.1.111
Chen, Guanyu	192.168.1.112
Chevireddi, Jayadev	192.168.1.113
Chokshi, Anokhee Pallav	192.168.1.114
Clark, Joshua Allan	192.168.1.115
Cody, Ethan P	192.168.1.116
Davis, Lenning Alston	192.168.1.117
Del Rio Grageda, Martin	192.168.1.118
Dilamani, Natasha	192.168.1.119
Donlon, Ethan	192.168.1.120
Duan, Ye	192.168.1.121
Duhamel, Justin P	192.168.1.122
Feng, Anna Q	192.168.1.123
Friedman, Jason Robert	192.168.1.124
Gaikwad, Chaitanya	192.168.1.125
Gao, Haihui	192.168.1.126
Ge, Alex H	192.168.1.127
Genovese, Nicholas C	192.168.1.128
Glickman, Miriam F	192.168.1.129
Gong, Yiang	192.168.1.130
Grant, Evan Douglas	192.168.1.131
Guo, Jinyuan	192.168.1.132
Gupte, Ruchi Rajesh	192.168.1.133
Gurrala, Venkata	192.168.1.134
Hahn, Zachary	192.168.1.135
Han, Jiangxue	192.168.1.136
Hardikar, Archit Nitin	192.168.1.137
Ho, Sarah M	192.168.1.138
Huang, Weihao	192.168.1.139
Hurvitz, Aviva	192.168.1.140
Jayakumar, Aadith Kumar	192.168.1.141
Jiang, Wenyang	192.168.1.142
Jin, Zhao	192.168.1.143
Kyimpopkin, Ariana	192.168.1.144
Lanter, Alec	192.168.1.145
Lara, Orlando	192.168.1.146
Le, Shangzhi	192.168.1.147
Lee, Christopher S	192.168.1.148
Li, Minglang	192.168.1.149
Li, Weiyu	192.168.1.150
Libsch, Xerxes	192.168.1.151
Lin, Gary	192.168.1.152
Lu, Yilan	192.168.1.153
Ma, Kaidi	192.168.1.154
Mairena, Jonathan	192.168.1.155
Manikandan, Akshay	192.168.1.156
May, Robert	192.168.1.157
Mehta, Anamil	192.168.1.158
Mehta, Bharg Jigesh	192.168.1.159
Musser, Ethan Joel	192.168.1.160
Nguyen, Tuan Minh	192.168.1.161
Nunez, Alan P	192.168.1.162
Owens, Paul	192.168.1.163
Oyang, George	192.168.1.164
Panagopoulou, Sofia	192.168.1.165
Parashar, Raktimjyoti	192.168.1.166
Parekh, Harshil	192.168.1.167
Park, Jeonginn Ho	192.168.1.168
Peralta, Sebastian J	192.168.1.169
Perez, Sophie I	192.168.1.170
Prescott, Richard K	192.168.1.171
Qiu, Quan	192.168.1.172
Rogers, Jared E	192.168.1.173
Sadhu, Saptarshi	192.168.1.174
Sahu, Divyanshu Rajendra	192.168.1.175
Santhanam, Harsha N	192.168.1.176
Schlatterer, William H	192.168.1.177
Sen, Raima Tapaskumar	192.168.1.178
Shah, Pranav Prashant	192.168.1.179
Shah, Vanshil Atul	192.168.1.180
Shen, Jia	192.168.1.181
Shen, Li	192.168.1.182
Sieg, Philip Gregory	192.168.1.183
Simpkins, Jacoby Niko	192.168.1.184
Song, Jiacheng	192.168.1.185
Stein, Abigail S	192.168.1.186
Sun, Bowen	192.168.1.187
Sun, Sunan	192.168.1.188
Tao, Sicheng	192.168.1.189
Teng, Fangyi	192.168.1.190
Tian, Jiadi	192.168.1.191
Tinney, Oscar S	192.168.1.192
Tong, Dayong	192.168.1.193
Voyer, Andres	192.168.1.194
Wang, Shufan	192.168.1.195
Wang, Xiaoyu	192.168.1.196
Williams, Hunter R	192.168.1.197
Xie, Shijie	192.168.1.198
Xing, Rubo	192.168.1.199
Ye, Ling	192.168.1.200
Yedla, Mallika R	192.168.1.201
Yee, Wesley Adam	192.168.1.202
Yeh, Yu-Hao Richard	192.168.1.203
You, Haoxiang	192.168.1.204
Zhang, Brian	192.168.1.205
Zhang, Congyuan	192.168.1.206
Zhang, Huiran	192.168.1.207
Zhao, Tianyun	192.168.1.208
Zhou, Mingyan	192.168.1.209
Zhu, Haitao	192.168.1.210

UDP receiver – sample code: setup() [part 1 of 2]

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

const char* ssid     = "yourhomessid";
const char* password = "yourhomepass";

WiFiUDP UDPTestServer;
IPAddress myIPaddress(192, 168, 1, 58);
```

New include files

USE YOUR ASSIGNED IP

```
void setup() {
  Serial.begin(115200);
  Serial.println();
  Serial.print("Connecting to ");  Serial.println(ssid);

  WiFi.config(myIPaddress, IPAddress(192, 168, 1, 1),
              IPAddress(255, 255, 255, 0));

  WiFi.begin(ssid, password);
  UDPTestServer.begin(2808); // any UDP port# up to 65535
                             // but higher is safer > 1023
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected");
}
```

Std setup

Optional. If omitted will use DHCP

Wait until connected

UDP receiver – sample code: loop() [part 2 of 2]

```
const int UDP_PACKET_SIZE = 100;
// can be up to 65535
byte packetBuffer[UDP_PACKET_SIZE+1];

void handleUDPServer() {
  int cb = UDPTestServer.parsePacket();
  if (cb) {
    UDPTestServer.read(packetBuffer, UDP_PACKET_SIZE);
    Serial.printf("%s\n",packetBuffer);
  }
}


void loop() {
  handleUDPServer();
  delay(1);
}
```

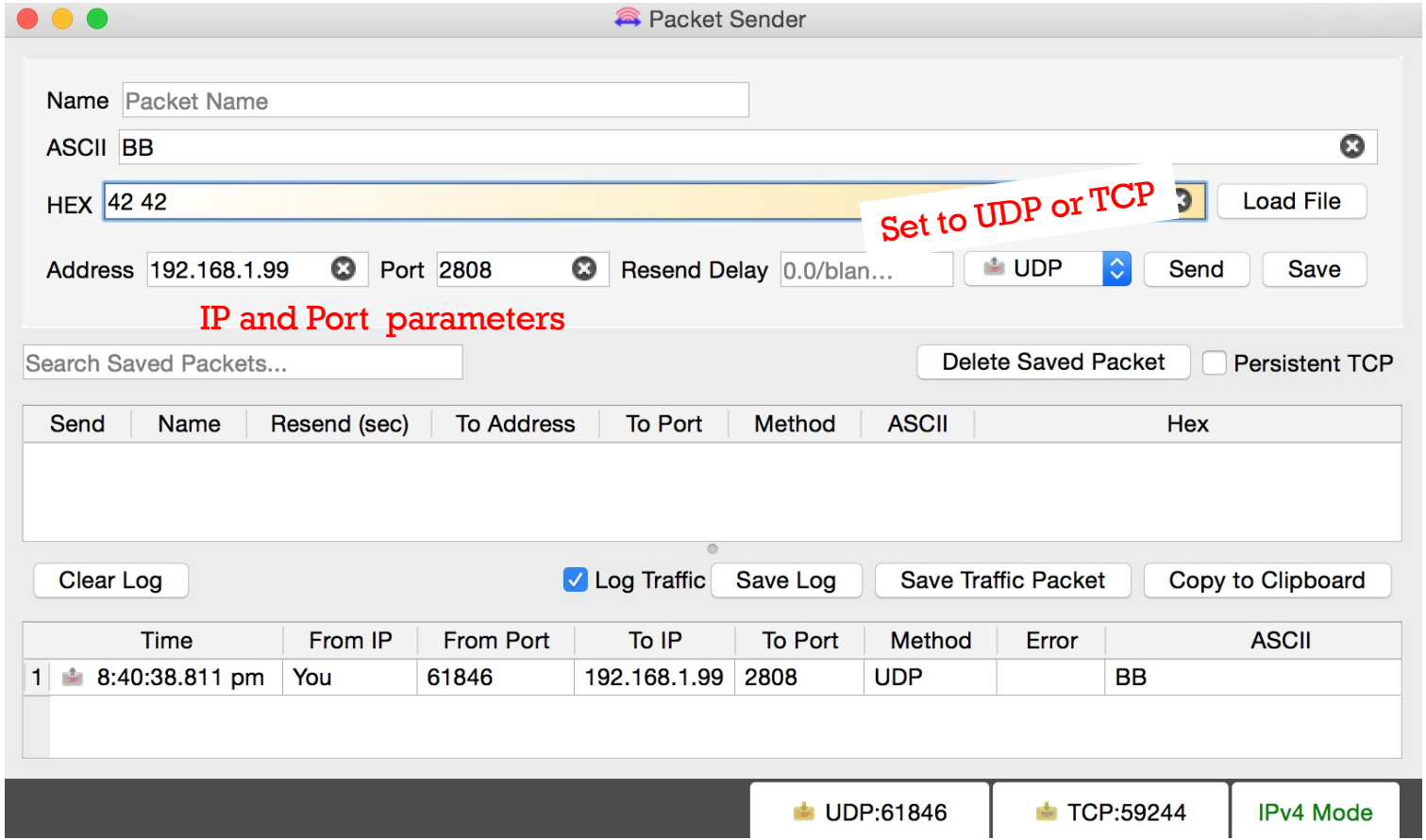
Check for packet

Read packets into mydata

Do something with info in packet

Loop checks for Packet events

PacketSender



UDP Sender: [part 1 of 2] setup()

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

const char* ssid     = "yourhomessid";
const char* password = "yourhomepass";

WiFiUDP udp;
IPAddress ipTarget(192, 168, 1, 58);
IPAddress myIP(192, 168, 1, 99);
```

```
void setup() {
  Serial.begin(115200);
  pinMode(LED_BUILTIN,OUTPUT);
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.config(myIP,
              IPAddress(192, 168, 1, 1),
              IPAddress(255, 255, 255, 0));
  WiFi.begin(ssid, password);
  udp.begin(2808);

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

https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

UDP Sender: [part 2 of 2] loop()

```
const int UDP_PACKET_SIZE = 100;
char udpBuffer[UDP_PACKET_SIZE];

void fncUdpSend()
{
  // send what ever you want upto buffer size
  udp.beginPacket(ipTarget, 2808);  // send to UDPport 2808
    udp.printf("%s",udpBuffer);
  udp.endPacket();
  Serial.println(udpBuffer);
}


// send udp packet every 4 seconds
void loop() {
  delay (4000);
  strcpy(udpBuffer, "hello testing message");
  fncUdpSend();
}
```

Data Packets: String manipulations

- String is an array of bytes
- What if we want to send/receive an `int` (2 bytes) in binary?

```
void fncUdpSend(int i) // send int i
{
  udpBuffer[0] = i & 0xff; // send 1st (LSB) byte of i
  udpBuffer[1] = i>>8;     // send 2nd (MSB) byte of i
  udpBuffer[2] = 0;        // null terminate string
  udp.beginPacket(ipTarget, 2808);  // send to UDPport 2808
    udp.printf("%s",udpBuffer);
  udp.endPacket();
}

void handleUDPServer() { // receive and print int i
  int i, cb = UDPTestServer.parsePacket();
  if (cb) {
    UDPTestServer.read(packetBuffer, packetSize);
```

Q6 What code will reassemble the int?

```
    Serial.println(i);
  }
}
```

Difference between UPD and TCP on ESP32

- TCP has some latency on ESP32 (~60mS each packet)
- UDP has low latency (avg ~2mS)
- UDP delivery not guaranteed (~3% packet loss in clean env)

- TCP can use HTML to send (e.g. make a website, use a phone or laptop)
- UDP needs UDP sender program (custom program or 2nd ESP32)

ESP32 UDP Quiz/ Summary

- Q7 Can we use 1234 for a UDP port number for this class?
- Q8 Can we send a bitmap image over UDP?

- *PacketSender* useful tool for debugging

05

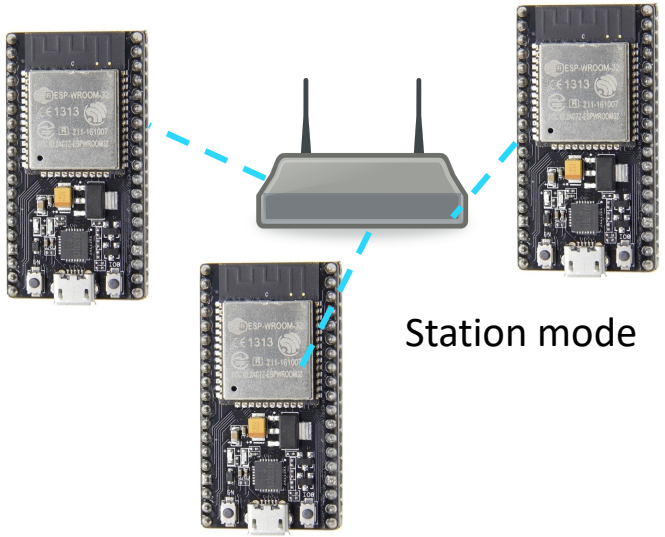
WiFi Modes

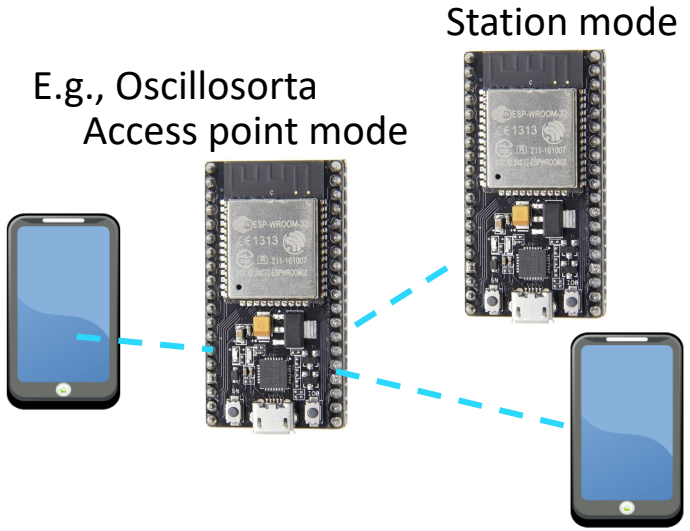
WiFi.modes()

- WiFi.mode(WIFI_STA) Station mode
  - ESP32 connects to router or other access point (input SSID etc.)

  [Need a router to connect to other things]
- WiFi.mode(WIFI_AP) Access Point Mode
  - ESP32 allows other devices to connect to it (supplies SSID etc.)

  [connect directly to ESP32 – no other router needed]
- WiFi.mode(WIFI_AP_STA) acts as both access point and station,
  - We haven't played with this – let us know if you learn about it.
- ESP-NOW – Proprietary ESP32 to ESP32 mode
  - More on this in future lectures





IPaddresses

Private IPV4 Address Spaces

IP Address Range	number of addresses	Classful
10.0.0.0 – 10.255.255.255	16,777,216	Class A
172.16.0.0 – 172.31.255.255	1,048,576	Class B
192.168.0.0 – 192.168.255.255	65,536	Class C

- DHCP
  - IP address automatically assigned by router
  - use `WiFi.localIP()` to find out what it is
- Static IP Address in station mode.
  - use:

```
WiFi.config(IPAddress(192, 168, 1, 99),    // local IP
            IPAddress(192, 168, 1, 1),     // Gateway
            IPAddress(255, 255, 255, 0));  // subnet mask
```

  - If fixed IP address is not specified, defaults to DHCP

Access Point Mode vs Station mode

```
void setup() {   // for Station mode - need a router
  WiFi.config(IPAddress(192, 168, 1, 2), IPAddress(192, 168, 1, 2),
              IPAddress(255, 255, 255, 0));

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  server.begin();
}
```

Omit if you want no password

Omit if you want to use DHCP, but remember to print `WiFi.localIP()`

```
void setup() {   // for AP mode - don't need a router
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);
  // may need to add delay(100); to make this work
  WiFi.softAPConfig(IPAddress(192, 168, 1, 2), IPAddress(192, 168, 1, 2),
                    IPAddress(255, 255, 255, 0));
  server.begin();
}
```

Web server on a chip without router (DHCP)

```
#include <WiFi.h>
const char* ssid = "myuniqueSSID"; // come up with your own personal SSID
const char* password = "something";

WiFiServer server(80);

void setup() {
  Serial.begin(115200);
  Serial.print("Access point "); Serial.print(ssid);
  WiFi.softAP( ssid, password) ; // omit password if you want it open.
  IPAddress IP = WiFi.softAPIP();
  Serial.print(" AP IP address"); Serial.println(IP);

  server.begin();
}
```

Web server on a chip without router (setting IP)

```
#include <WiFi.h>
const char* ssid = "myuniqueSSID"; // come up with your own personal SSID


WiFiServer server(80);

void setup() {
  Serial.begin(115200);
  Serial.print("Access point "); Serial.print(ssid);
  WiFi.softAP( ssid) ;  // example of open access point (no password)
  IPAddress Ip(192, 168, 1, 2);
  IPAddress NMask(255, 255, 255, 0);
  WiFi.softAPIP(Ip, Ip, NMask);
  Serial.print(" AP IP address"); Serial.println(IP);

  server.begin();
}
```

Web server on a chip with a router (using DHCP)

```
#include <WiFi.h>

const char* ssid = "yourhomerouterSSID";
const char* password = "yourhomepassword";

WiFiServer server(80);   // port 80 is standard for websites

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid,password);
  while(WiFi.status()!= WL_CONNECTED ) {
    delay(500);
    Serial.print(".");
  }
  Serial.print("Use this URL to connect: http://");
  Serial.print(WiFi.localIP());   Serial.println("/");
  server.begin();
}
```

SSID + passwd

Std setup

Wifi setup
Uses DHCP

Web server on a chip with a router (setting IP)

```
#include <WiFi.h>

const char* ssid = "yourhomerouterSSID";
const char* password = "yourhomepassword";

WiFiServer server(80);   // port 80 is standard for websites

void setup() {
  Serial.begin(115200);
  WiFi.config(IPAddress(192, 168, 1, 2), IPAddress(192, 168, 1, 2),
              IPAddress(255, 255, 255, 0));
  WiFi.begin(ssid,password);
  while(WiFi.status()!= WL_CONNECTED ) {
    delay(500);
    Serial.print(".");
  }
  server.begin();
}
```

SSID + passwd

Std setup

Debugging WiFi on ESP32

- Using AP mode on laptop: don't forget to set the proper SSID on your network. (e.g. many laptops will automatically go back to AirPennNet after a network goes down)
- Using STA mode on laptop is more convenient because you don't lose access to the internet.
- Debug HTML and websites with chrome and "*developer tools*"
- Debug a receiver with *Packetsender*
- Debug a sender with a debugged receiver… Maybe use *Wireshark* a comprehensive opensource free packet analyzer tool.

ESP-NOW (more on this mode later)

- Simple setup using MAC address
- Espressif Proprietary (only works with ESP)
- Very fast to connect and low latency. Great for sending small messages (low overhead).
- Not guaranteed like UDP, but can receive ACK if message fails.
- Can network up to 20 ESPs (10 if using encrypted modes).
- No router, no DHCP, no HTTP overhead
- Maximum packet size is 250 bytes
- Likely still slower than laptop or phone wifi.

WiFi Setup Summary

- ESP32 can be a station with router or its own access point.
- Set up static IP with wifi.config. If you omit this line, DHCP will be used.
- You will need to set up your own IP address for MEAM510
- The Arduino code examples are available on Canvas -> files -> resources -> ESP32 Arduino Lecture Examples

Using Libraries with Arduino

- Using libraries with Arduino
- Libraries (not .h files) are the way to modularize your source code or use code that others have made without having to read the source code.
- You don't need to do this. But if you want to, there's a nice tutorial:
  - https://www.arduino.cc/en/Hacking/LibraryTutorial
  - It also explains some rudimentary aspects of c++ vs c.
- Using the arduino libraries in general
  - https://www.arduino.cc/en/Guide/Libraries
  - https://www.arduino.cc/en/Reference/Libraries

Link References

- Espressif IoT Development Framework (esp-idf)
  https://docs.espressif.com/projects/esp-idf/en/latest/index.html
- ESP32 tutorials (including wifi)
  https://www.instructables.com/id/IOT-Made-Simple-Playing-With-the-ESP32-on-Arduino-/
- Accesspoint
  https://randomnerdtutorials.com/esp32-access-point-ap-web-server/
- Arduino on ESP32 examples
  https://github.com/espressif/arduino-esp32/tree/master/libraries/ESP32/examples
- Tons of ESP32 links: http://esp32.net/

Other references

- TCP/IP and UDP more in depth reference
  - https://hpbn.co/building-blocks-of-udp/
  - https://hpbn.co/building-blocks-of-tcp/
- HTML, CSS, javascript and other web programming
  - https://www.w3schools.com/html/default.asp
- LEDC
  - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/ledc.html C++ Espressif reference doc
  - https://randomnerdtutorials.com/esp32-pwm-arduino-ide/ Arduino tutorial
  - https://github.com/espressif/arduino-esp32/blob/master/cores/esp32/esp32-hal-ledc.c Arduino code

Answer in Chat

Answer how you feel about each topic below with:

1. I don't understand this topic at all
2. I don't know now, but know what to do to get by
3. I understand some, but expect to get the rest later
4. I understand completely already

A. LEDC

B. TCP vs UDP

C. Using ESP32 for web applications