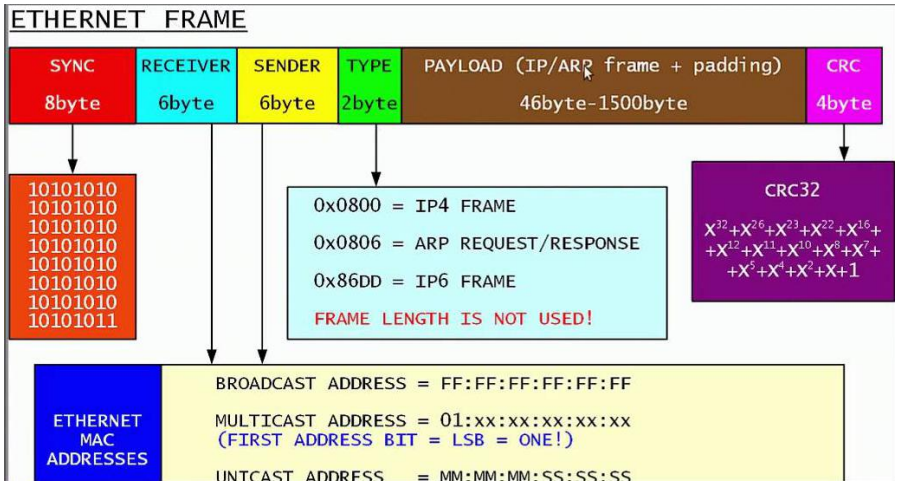
Python Packet Sniffer

Section 1: Introduction
Overview:
1. Notice when we connect physical connection to router eg, & measure I or V using multimeter, there will be pwm signal (up and down OR 1 and 0)
2. These is what our computer knows as binary (data)
3. One of the reason to implement this is to monitor data across network

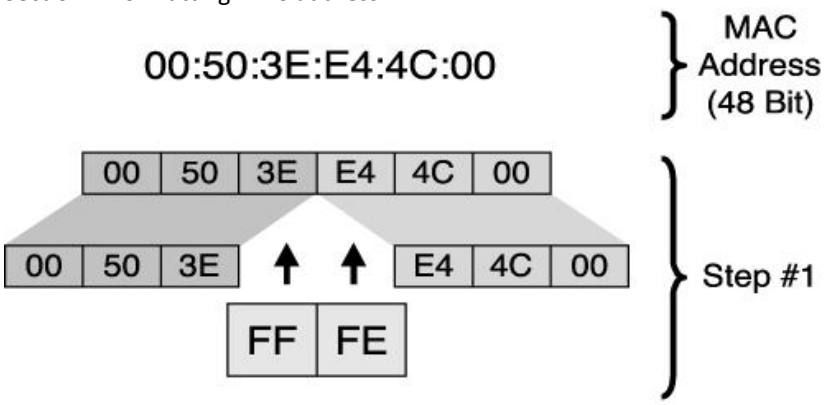
A lil bit of breakdown of this program:
1. get data from computer to router
2. HTTP request will be wrapped up in IP packet
3. All of these is wrapped up in ethernet frame



4. The sync is important to ensure our computer & router are in sync so they know when receiving the packet(all data received without error)--but don't really have infos that is useful to human
5. Receiver & Sender (one will be our computer & the other will be the router)--sending & receiving data
6. The type is the ethernet type or protocol. Here,we just want to check whether we are working with regular traffic IPv4 (within standard)
7. Payload is the main data.
8. Check the PacketSniffer1 codes

```
#unpack ethernet frame
def ethernet_frame(data): #when see data, pass to this fn
	#dest_mac == receiver, src_mac is sender, proto == type
	dest_mac,src_mac,proto =    struct.unpack('! 6s 6s H',data[:14]) #grab 1st 14 byte
	#CONVERT DATA TO & FROM BYTE,we will just look to 1st 14 bytes
	return get_mac_addr(dest_mac),get_mac_addr(src_mac),socket.htons(proto),data[14:] # these are fn to convert addr to human readable
	#note: data[14:]== data 14 to the en, grab data 14 to the end (payload is here)
```

Section2: Formatting MAC address



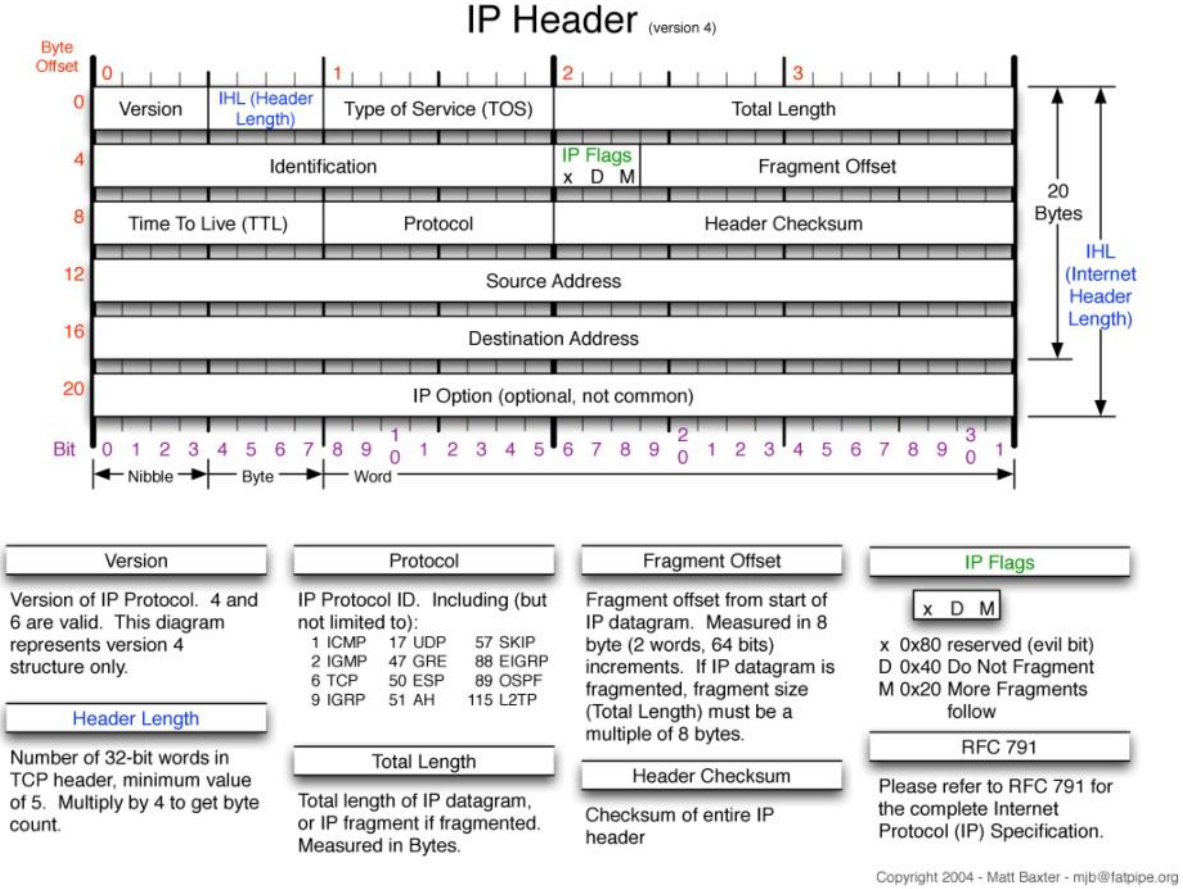
Refer PacketSniffing2 code

```
#return properly formatted MAC address AA:BB:CC:DD:EE:FF
def get_mac_addr(bytes_addr):
	bytes_str = map('{:02x}'.format,bytes_addr)
	  #format addr to 2 decimal places,loop through bytes_addr(iterate)
	return ':'.join(bytes_str).upper()
	  #colon in between all of them,everything uppercase
```

Section3: Capturing Traffic

```
#loop,wait for packet,whenever they do, extract the packet
def main():
	#create socket for connection
	conn = socket.socket(socket.AF_PACKET,socket.SOCK_RAW, socket_ntohs(3))
	#last one ensure byte order is proper so we could read it
	while True:#keep loopng & wait for any data to come across
#receive it all,store them in a var named raw_data(real data that gonne be sent to ethernet
frame),addr=src addr
		raw_data, addr = conn.recvfrom(65536)
		dest_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)
		print('\nEthernet Frame:')
		print('Destination: {}, Source: {}, Protocol: {}'.format(dest_mac,src_mac,eth_proto))
```

Section4: Unpacking IP Packet Header



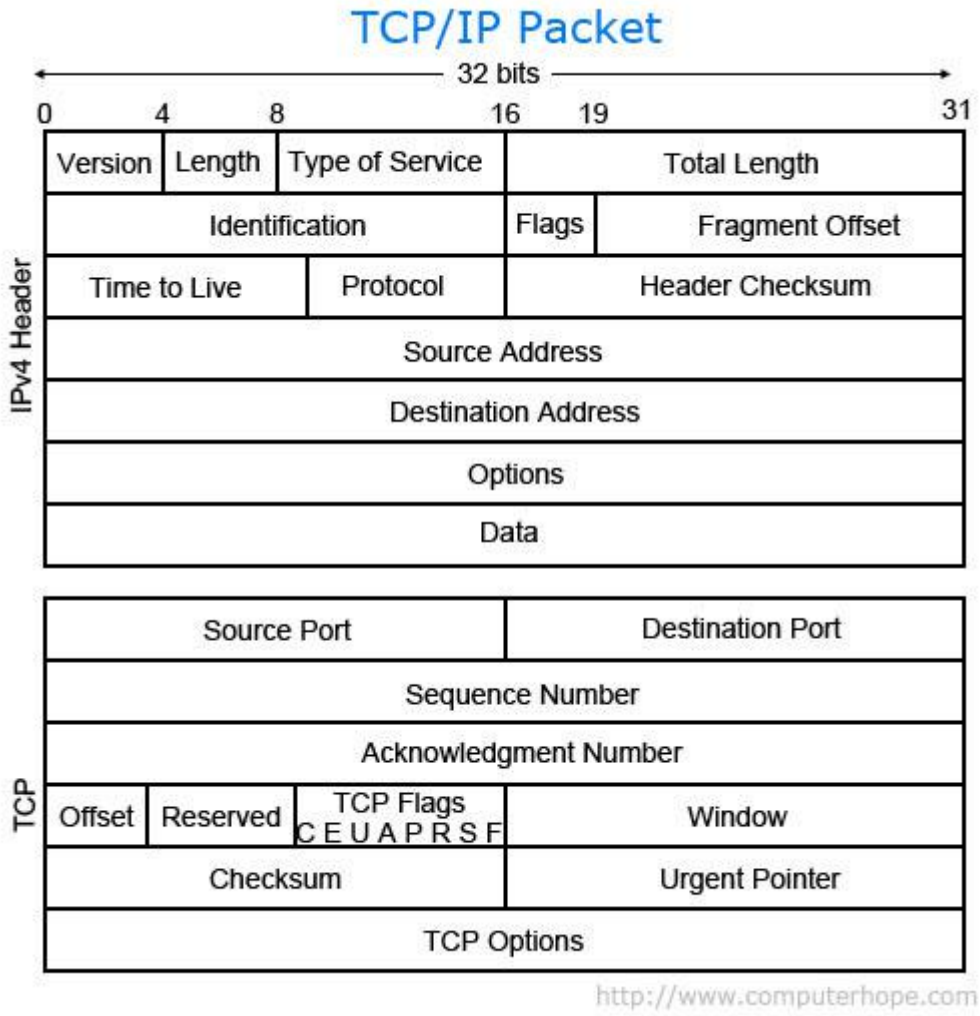
This section unpack the IP data

```
#unpack IPv4 packet
def ipv4_packet(data):
    version_header_length = data[0]
    #bitwise operation, compare 2 bytes,gonna get result when both bytes are 1
    version = version_header_length >> 4
    #header length determine where the data start,right after header ends,data begins
    header_length = (version_header_length & 15)*4
    #unpack everything
    ttl, proto, src, target = struct.unpack('! 8x B B 2x 4s 4s',data[:20]) #(format of data,data)
    return version, header_length,ttl,proto,src,target, ipv4(src), ipv4(target), data[header_length:]
#note:: IP packet is broken up into header & the actual data

#Return properly formatted IPv4 address
def ipv4(addr):
    return '.'.join(map(str,addr)) #strings joined together with dot.
```

Section5: Unpacking ICMP and TCP Data

This section, we need to find out what data is inside by looking at the protocol
A number will be returned. Refer to Ip packet diagram above to figure out which proto it is.
99% are consist of TCP,UDP and ICMP



So far we already unpack the IP section. Now, we gonna focus on TCP.
Basically this is what u will get when visiting sites like facebook, instagram etc.

The offset, Reserved & TCP Flags itself is 16 bit (These are all in 1 chunk)
If we want to get the offset, we need to bitwise it to the right, so the reserved & flags will be removed out.

Section 6: Displaying Packet Data

```
#8 is the ethernet protocol, verify whether we are working with ipv4 or not
        if eth_proto == 8:
            (version,header_length,ttl,proto,src,target,data) = ipv4_packet(data)
            print(TAB_1 + 'IPv4 Packet:')
            print(TAB_2 + 'version: {}, Header_Length:{},TTL:{}'.format(version,header_length,ttl))
            print(TAB_2 + 'Protocol: {}, Sources:{},Target:{}'.format(proto,src,target))

            if proto ==1:
                icmp_type,code,checksum,data = icmp_packet(data)
                print(TAB_1 + 'ICMP Packet:')
                print(TAB_2 + 'Type: {}, Code:{},Checksum:{}'.format(icmp_type,code,checksum))
                print(TAB_2 + 'Data:')
                print(format_multi_line(DATA_TAB_3,data))
```

Section7: Run the program!
Make sure to run as root