Implementing a Packet Sniffer
CPSC 3600 Networked Systems

This assignment will introduce the concept of packet sniffing, header processing, and packet manipulation. Packets can be sniffed using Python's built in raw sockets, which allow you to read all packets coming in on a particular network interface as well as directly manipulate the different layer headers for packets sent over the raw socket. However, Windows does not natively support raw sockets, so we will be using the library ***scapy*** to help us sniff packets.

1 Assignment Instructions

You'll find seven Python files in the project template. You will need to complete code in six of these.

1. **sniffer.py**: This class is the actual packet sniffer that detects packets and routes them to your remaining code for processing and unpacking. You will need to complete a few lines of code in this file that handle what header type should be unpacked nexted, based on a value stored in the current header.
2. **layer_header.py**: This is an abstract class defining properties and functions that will be used in the header classes defined below. You should not edit this file.
3. **link_layer_headers/ethernet_header.py**: This class represents an Ethernet header. The properties of the Ethernet header are already defined. You will need to complete the constructor to unpack the header bytes and assigned them to the defined properties of the class.
4. **network_layer_headers/ipv4_header.py**: This class represents an IPv4 header. The properties of the IPv4 header are already defined. You will need to complete the constructor to unpack the header bytes and assigned them to the defined properties of the class.
5. **network_layer_headers/arp_header.py**: This class represents an ARP header. The properties of the ARP header are already defined. You will need to complete the constructor to unpack the header bytes and assigned them to the defined properties of the class.
6. **transport_layer_headers/tcp_header.py**: This class represents an TCP header. The properties of the TCP header are already defined. You will need to complete the constructor to unpack the header bytes and assigned them to the defined properties of the class.
7. **transport_layer_headers/udp_header.py**: This class represents an UDP header. The properties of the UDP header are already defined. You will need to complete the constructor to unpack the header bytes and assigned them to the defined properties of the class.

1.1 Installing Scapy

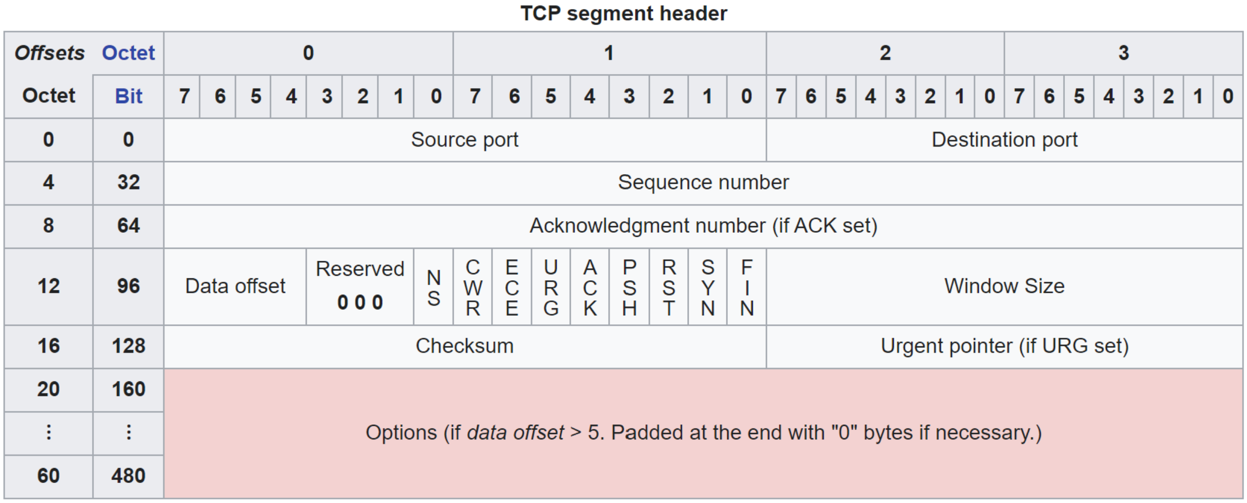
Scapy can be installed using pip, python's package management system. It should already be set up if you have installed python. You should be able to install scapy by opening a terminal and running the command **pip install scapy**. If you are using a Windows machine, you will also need to install Npcap. This is likely already installed if you have Wireshark on your computer.

Be aware that python maintains different package repositories for each version of python installed on a machine, as well as for each virtual machine you've configured. If you install scapy and it is not recognized in your development environment, you likely have your IDE set to use a different installation of python than you command window defaults to.

2 Protocol Headers

Your sniffer neesd to be able to handle five different header types. I've included the format for these headers in this section. We haven't discussed two of these headers yet, however both are straightforward to work with.

2.1 TCP Header

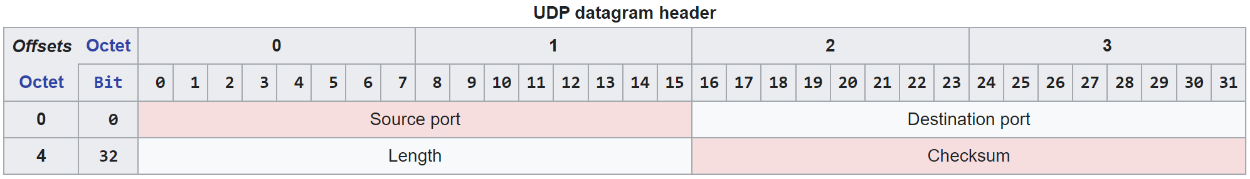


TCP segment header

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.)																															
⋮	⋮																																
60	480																																

TCP's header is composed of a fixed-length header of 20 bytes and a variable-length set of options. The typical TCP header does not include any options, however, since it is always a possibility you will always need to check to see if any are present. This can be done using the *data offset* field.

2.2 UDP Header

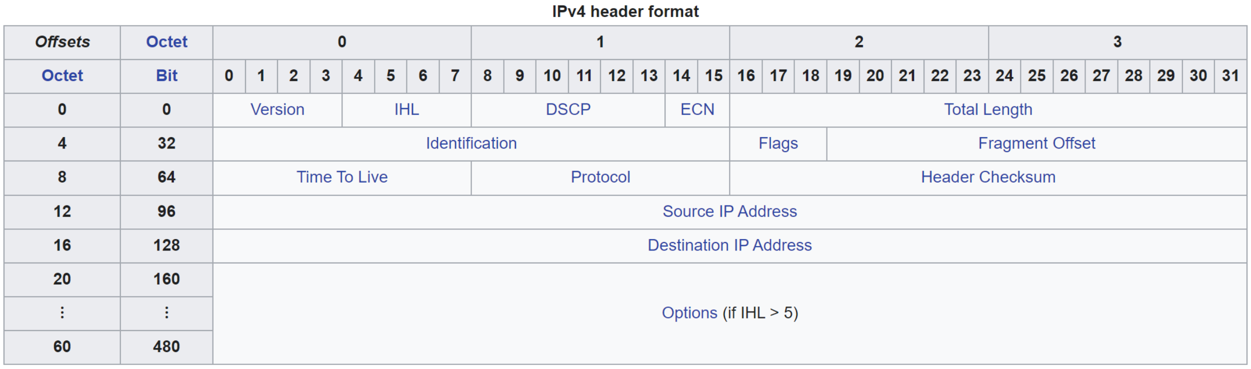


UDP datagram header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

UDP's header is always 8 bytes in length.

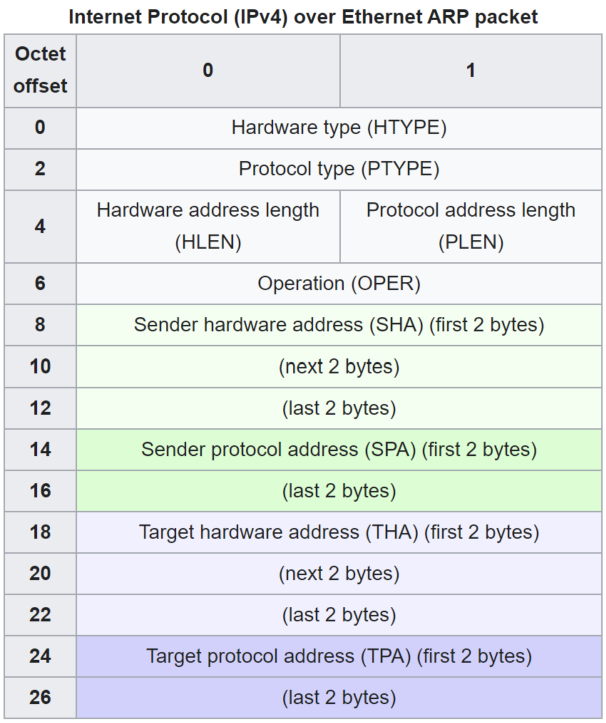
2.3 IPv4 Header



IPv4 header format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset												
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
⋮	⋮																																
60	480																																

IPv4's header is composed of a fixed-length header of 20 bytes and a variable-length set of options. The typical IPv4 header does not include any options, however, since it is always a possibility you will always need to check to see if any are present. This can be done using the *IHL* field.



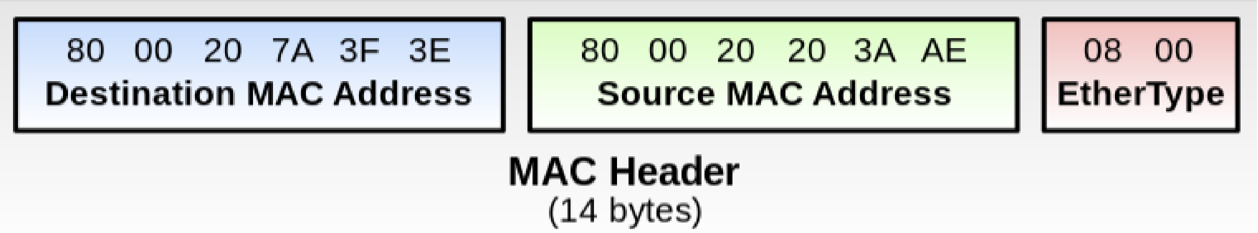
Internet Protocol (IPv4) over Ethernet ARP packet

Octet offset	0	1
0	Hardware type (HTYPE)	
2	Protocol type (PTYPE)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	
8	Sender hardware address (SHA) (first 2 bytes)	
10	(next 2 bytes)	
12	(last 2 bytes)	
14	Sender protocol address (SPA) (first 2 bytes)	
16	(last 2 bytes)	
18	Target hardware address (THA) (first 2 bytes)	
20	(next 2 bytes)	
22	(last 2 bytes)	
24	Target protocol address (TPA) (first 2 bytes)	
26	(last 2 bytes)	

2.4 ARP Header

ARP's header is technically variable length, however for our purposes we'll assume that it is always 28 bytes in length. The HLEN and PLEN values store the length of the hardware and protocol addresses. ARP messages typically communicate MAC addresses (associated with the link layer) and IP addresses (associated with the network layer). When this is true, the hardware address fields are composed of six bytes and the protocol address fields are composed of four bytes, meaning that HLEN is set to 6 and PLEN is set to 4. If different addresses are communicated then the header may change length depending on how many bytes are used to represent those addresses.

2.5 Ethernet



Ethernet's header is always 14 bytes in length.

3 Test cases

The autograder will test each of the parameters of each packet header separately. It does this by creating a packet with a known value for a given header parameter (the value is chosen randomly) and then sending that packet across the network while your sniffer is running. It sends the packet twice in case one were to be missed by your packet sniffer. It then examines the packets returned by your sniffer to see if one of them contains the expected value for the target header parameter.

It should be noted that it is technically possible for a test to fail due to packet loss or corruption. If a test randomly fails, try re-submitting your project. If the test fails again, it is likely a problem with your code.

4 Submitting your code

You can submit your project through Gradescope (which can be accessed via Canvas). You'll see the Implementing a Packet Sniffer assignment listed on your dashboard. **Zip up your files, preserving the folder structure, and submit the zip file. You must submit this as a zip.** A batch of tests will begin running once you submit your code. These should complete fairly quickly, after which you'll see what tests you passed and failed.