Department of Electrical and Computer Engineering

ECE 363 Communications Networks
Lab 1

Name Student no. Date Lab Title Lab Section	Arfaz Hossain V00984826 February 8, 2024 Introduction to WireShark and Layered Protocol B03

Discussion

In this lab, students are tasked with familiarizing themselves with Wireshark, a widely used network protocol analyzer. Wireshark passively observes packets transmitted to or from a designated network interface, providing insights into network traffic without actively participating. It captures copies of packets exchanged by applications and protocols running on end systems, enabling analysis of various network activities. Through this exploration, students gained practical experience in examining network protocols and understanding their layered structure. The lab exercises helped students gain more knowledge into capturing traces, identifying protocols, calculating overhead, examining packet headers, and utilizing networking tools such as "ifconfig", "ping", and "netstat" to solidifying their understanding of network operations.

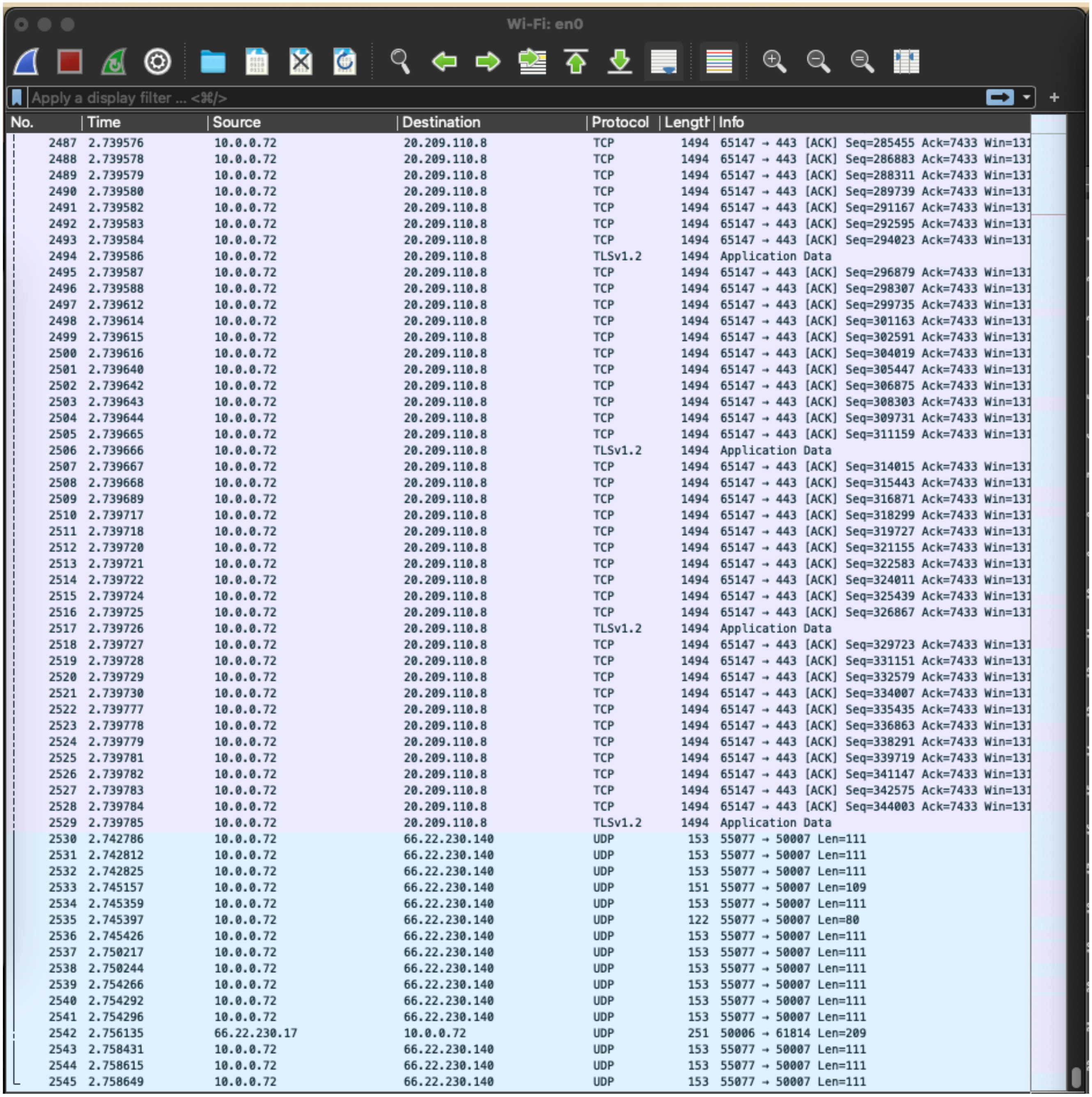
Procedure

This lab introduced students to Wireshark, covering the installation process, getting acquainted with the tool, and analyzing layered protocols, especially focusing on **HTTP GET** packets. It provided practical insights into network packet capture and analysis. Students learned to draw the structure of an **HTTP GET** packet and performed calculations related to packet overhead through looking at all the byte positions in the captured packet data and calculating the sizes of each packet in the capture. Students used Wireshark's capture filters to refine packet capturing through employing the filter **tcp port 80** to capture relevant packets for their analysis using **wget** command while excluding unrelated network traffic. On the Layered Protocol analysis, students examine detailed information regarding different layers of the protocol in HTTP GET packets, using both provided and captured packets for analysis. The lab also encouraged exploration of networking tools like **ifconfig** **ping** and **netstat** to enhance understanding of computer networking concepts.

Please Turn Over

1.3.1 Running WireShark

1.3.1.1 Capture a trace without any filters.



No.	Time	Source	Destination	Protocol	Length	Info
2487	2.739576	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=285455 Ack=7433 Win=131
2488	2.739578	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=286883 Ack=7433 Win=131
2489	2.739579	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=288311 Ack=7433 Win=131
2490	2.739580	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=289739 Ack=7433 Win=131
2491	2.739582	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=291167 Ack=7433 Win=131
2492	2.739583	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=292595 Ack=7433 Win=131
2493	2.739584	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=294023 Ack=7433 Win=131
2494	2.739586	10.0.0.72	20.209.110.8	TLSv1.2	1494	Application Data
2495	2.739587	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=296879 Ack=7433 Win=131
2496	2.739588	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=298307 Ack=7433 Win=131
2497	2.739612	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=299735 Ack=7433 Win=131
2498	2.739614	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=301163 Ack=7433 Win=131
2499	2.739615	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=302591 Ack=7433 Win=131
2500	2.739616	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=304019 Ack=7433 Win=131
2501	2.739640	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=305447 Ack=7433 Win=131
2502	2.739642	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=306875 Ack=7433 Win=131
2503	2.739643	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=308303 Ack=7433 Win=131
2504	2.739644	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=309731 Ack=7433 Win=131
2505	2.739665	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=311159 Ack=7433 Win=131
2506	2.739666	10.0.0.72	20.209.110.8	TLSv1.2	1494	Application Data
2507	2.739667	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=314015 Ack=7433 Win=131
2508	2.739668	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=315443 Ack=7433 Win=131
2509	2.739689	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=316871 Ack=7433 Win=131
2510	2.739717	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=318299 Ack=7433 Win=131
2511	2.739718	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=319727 Ack=7433 Win=131
2512	2.739720	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=321155 Ack=7433 Win=131
2513	2.739721	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=322583 Ack=7433 Win=131
2514	2.739722	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=324011 Ack=7433 Win=131
2515	2.739724	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=325439 Ack=7433 Win=131
2516	2.739725	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=326867 Ack=7433 Win=131
2517	2.739726	10.0.0.72	20.209.110.8	TLSv1.2	1494	Application Data
2518	2.739727	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=329723 Ack=7433 Win=131
2519	2.739728	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=331151 Ack=7433 Win=131
2520	2.739729	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=332579 Ack=7433 Win=131
2521	2.739730	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=334007 Ack=7433 Win=131
2522	2.739777	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=335435 Ack=7433 Win=131
2523	2.739778	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=336863 Ack=7433 Win=131
2524	2.739779	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=338291 Ack=7433 Win=131
2525	2.739781	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=339719 Ack=7433 Win=131
2526	2.739782	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=341147 Ack=7433 Win=131
2527	2.739783	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=342575 Ack=7433 Win=131
2528	2.739784	10.0.0.72	20.209.110.8	TCP	1494	65147 → 443 [ACK] Seq=344003 Ack=7433 Win=131
2529	2.739785	10.0.0.72	20.209.110.8	TLSv1.2	1494	Application Data
2530	2.742786	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2531	2.742812	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2532	2.742825	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2533	2.745157	10.0.0.72	66.22.230.140	UDP	151	55077 → 50007 Len=109
2534	2.745359	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2535	2.745397	10.0.0.72	66.22.230.140	UDP	122	55077 → 50007 Len=80
2536	2.745426	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2537	2.750217	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2538	2.750244	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2539	2.754266	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2540	2.754292	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2541	2.754296	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2542	2.756135	66.22.230.17	10.0.0.72	UDP	251	50006 → 61814 Len=209
2543	2.758431	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2544	2.758615	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111
2545	2.758649	10.0.0.72	66.22.230.140	UDP	153	55077 → 50007 Len=111

One trace without any filters:

Time	**Source**	**Destination**	**Protocol**
2.791961	2604:3d08:2481:4f00:18a1:72ab:c591:ca53	2603:1063:2200:30::42	TCP

Length	**Info**
74	63756 → 443 [ACK] Seq=1 Ack=34 Win=4095 Len=0

1.3.1.2 List at least 3 different protocols that appear in the protocol column of the unfiltered packet-listing window.

I got 3 different protocols which appeared in the protocol column of the unfiltered packet-listing window. These protocols represent different layers and functionalities within network communications. These are: **TCP, UDP and**

TCP (Transmission Control Protocol): TCP is a connection-oriented protocol used for reliable, ordered, and error-checked delivery of data packets over a network. It is widely used for applications such as web browsing, email, file transfer, and more.[1]

Time	Source	Destination	Protocol
2.791961	2604:3d08:2481:4f00:18a1:72ab:c591:ca53	2603:1063:2200:30::42	TCP
Length	**Info**		
74	63756 → 443 [ACK] Seq=1 Ack=34 Win=4095 Len=0		

UDP (User Datagram Protocol): UDP is a connectionless protocol that offers a minimal amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP is commonly used in applications where speed is more critical than reliability, such as real-time communication protocols, online gaming, and streaming media.[2]

Time	Source	Destination	Protocol	Length	Info
2.584709	10.0.0.122	10.0.0.255	UDP	305	58739 → 54915 Len=263

TLSv1.2 (Transport Layer Security version 1.2): TLS is a cryptographic protocol used to establish a secure communications channel between two systems. TLSv1.2 is a version of TLS and represents one of the iterations of the protocol. It provides privacy and data integrity between communicating applications and is commonly used in securing web communications, email transmission, and other forms of data exchange over networks.[3]

Time	Source	Destination	Protocol
1.693690	2620:1ec:8f8::10	2604:3d08:2481:4f00:18a1:72ab:c591:ca53	TLSv1.2
Length	**Info**		
125	Change Cipher Spec, Encrypted Handshake Message		

Please Turn Over

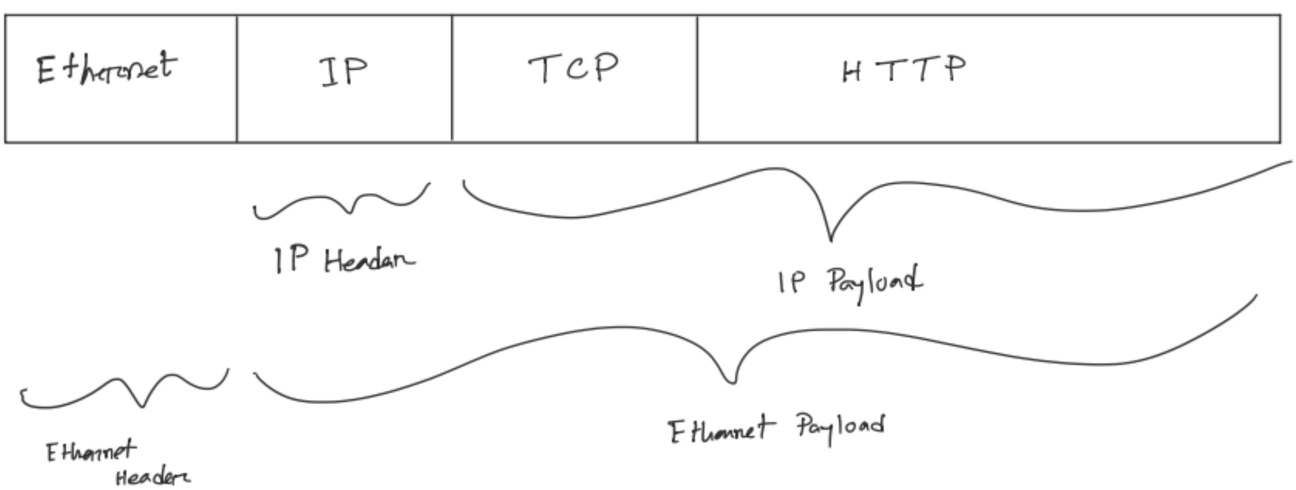
1.3.1.3 How long did it take from the HTTP GET message being sent to the HTTP OK reply being received?

4	0.021669	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	HTTP	214	GET / HTTP/1.1
5	0.046933	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	86	80 → 49905 [ACK] Seq=1 Ack=129 Win=66816 Len=
6	0.104078	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [ACK] Seq=1 Ack=129 Win=66816 Len=
7	0.104305	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	49905 → 80 [ACK] Seq=129 Ack=1209 Win=130112
8	0.106071	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [PSH, ACK] Seq=1209 Ack=129 Win=66
9	0.106073	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [ACK] Seq=2417 Ack=129 Win=66816 L
10	0.106074	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [PSH, ACK] Seq=3625 Ack=129 Win=66
11	0.106075	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [ACK] Seq=4833 Ack=129 Win=66816 L
12	0.106076	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [PSH, ACK] Seq=6041 Ack=129 Win=66
13	0.106112	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [ACK] Seq=7249 Ack=129 Win=66816 L
14	0.106215	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	49905 → 80 [ACK] Seq=129 Ack=7249 Win=124928
15	0.106282	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	49905 → 80 [ACK] Seq=129 Ack=8457 Win=123712
16	0.108209	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	[TCP Window Update] 49905 → 80 [ACK] Seq=129
17	0.110477	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [PSH, ACK] Seq=8457 Ack=129 Win=66
18	0.110479	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [ACK] Seq=9665 Ack=129 Win=66816 L
19	0.110647	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	49905 → 80 [ACK] Seq=129 Ack=10873 Win=12864
20	0.113627	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [PSH, ACK] Seq=10873 Ack=129 Win=6
21	0.113746	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	49905 → 80 [ACK] Seq=129 Ack=12081 Win=12985
22	0.120337	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [ACK] Seq=12081 Ack=129 Win=66816
23	0.120339	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [PSH, ACK] Seq=13289 Ack=129 Win=6
24	0.120340	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [ACK] Seq=14497 Ack=129 Win=66816
25	0.120482	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	49905 → 80 [ACK] Seq=129 Ack=15705 Win=12742
26	0.120653	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	[TCP Window Update] 49905 → 80 [ACK] Seq=129
27	0.121679	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [PSH, ACK] Seq=15705 Ack=129 Win=6
28	0.121680	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [ACK] Seq=16913 Ack=129 Win=66816
29	0.121681	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [PSH, ACK] Seq=18121 Ack=129 Win=6
30	0.121726	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	49905 → 80 [ACK] Seq=129 Ack=19329 Win=12742
31	0.121759	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	[TCP Window Update] 49905 → 80 [ACK] Seq=129
32	0.123469	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	1294	80 → 49905 [ACK] Seq=19329 Ack=129 Win=66816
33	0.123470	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	HTTP	327	HTTP/1.1 200 OK (text/html)
34	0.123583	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	49905 → 80 [ACK] Seq=129 Ack=20778 Win=12960
35	0.125584	2604:3d08:2481:4f00:18a1…	2607:f8b0:400a:804::2003	TCP	86	49905 → 80 [FIN, ACK] Seq=129 Ack=20778 Win=1
36	0.140397	2607:f8b0:400a:804::2003	2604:3d08:2481:4f00:18a1…	TCP	86	80 → 49905 [FIN, ACK] Seq=20778 Ack=130 Win=6

(0.123478 seconds – 0.021669 seconds) = 0.101809 seconds

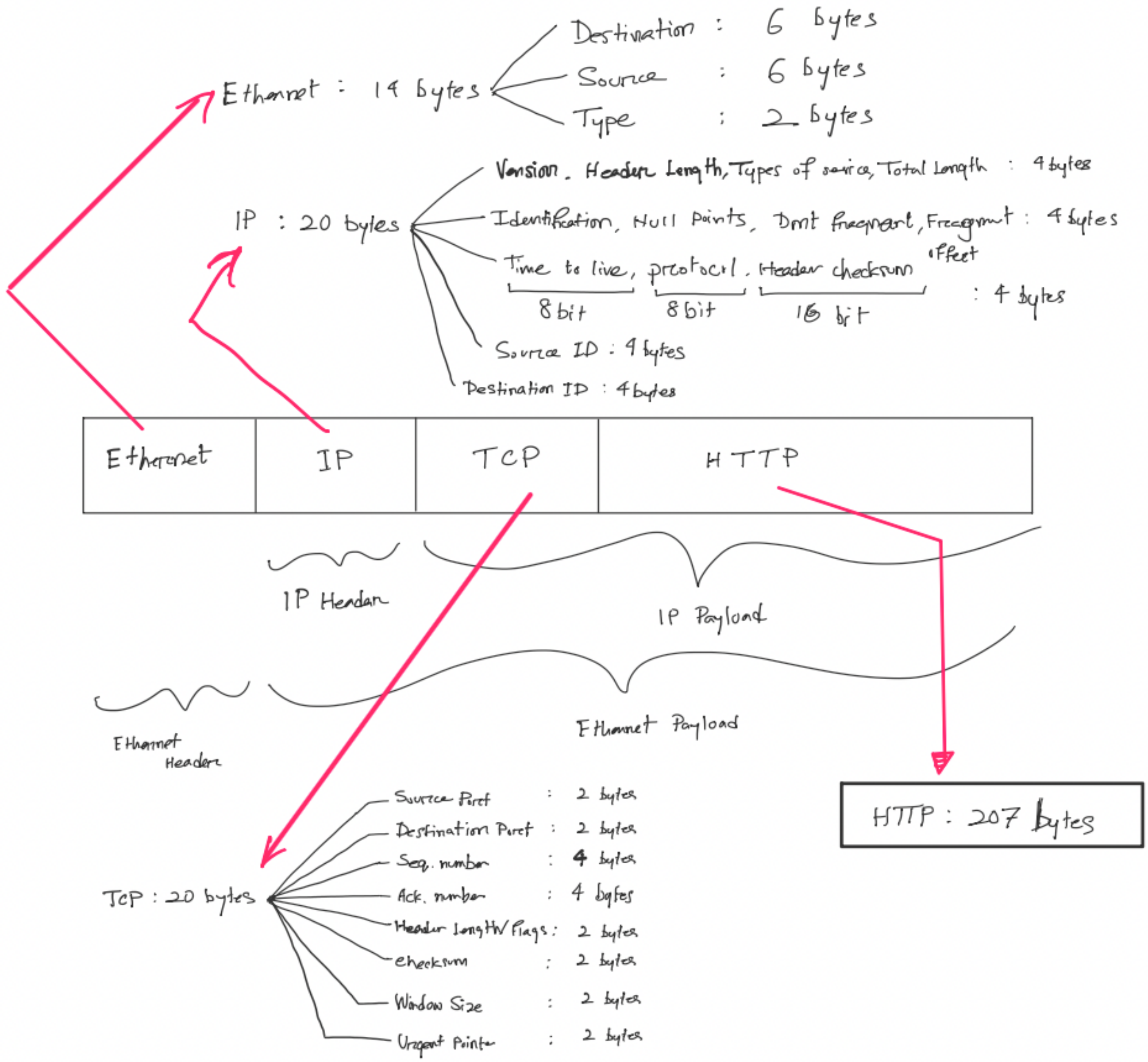
1.3.2 Layered Protocol

1.3.2.1. Draw the structure of an HTTP GET packet.



In a packet structure of an **HTTP GET** request, we first encase the data within a TCP header. This TCP packet is then wrapped within an IP packet by adding an IP header. This IP packet is finally inserted into an Ethernet frame, which adds the Ethernet header and footer.

The packet structure of an **HTTP GET** request contains three key components that are layered or nested within each other: Ethernet, IP, and **TCP**. Considering the overall packet as a long, thin rectangle, and going from left to right (since leftmost elements being the first sent on the wire), we start with the Ethernet Header. The Ethernet header typically has 14 bytes, so I marked it on the first section of my rectangle. The remaining part of this layer is considered the *Ethernet payload*, which is the data that IP passed to Ethernet over the network.



Within the Ethernet payload, we can find both the *IP header*, and the *IP payload*. The *IP header*, depending on whether it's IPv4 or IPv6, will either have 20 bytes (IPv4) or 40 bytes (IPv6). The remaining part of the *IP payload* is the *TCP Header*, which can vary in size but typically is 20 bytes unless options are being used. The rest of the TCP segment encompasses the actual **HTTP GET** request.

1.3.2.2. In the provided trace (lab1-wget-trace.pacp), calculate the average overhead of all of the packets from the server to the client (in percentage). (Hint: For a packet, the overhead is the size of all headers over the packet's total size. The average overhead is the ratio of the sum of the headers' size over the sum of the packets' size).

Header Length: $2\times40+ 26\times20 + 24\times32 = 1368$
Packet Length: $(2\times74) + (13\times66) + (7\times1484) + 177 + 1362 + 164 + 216 = 13313$
Overhead: $\left({}^{1368}/_{13313}\right) = 0.1028$

1.3.2.3. Which bytes in the Ethernet header field tell that the next higher layer protocol is IP? What is its hexadecimal value?

Each of the Ethernet Header field has 14 bytes of data, and the last 2 bytes of this field has 2 bytes allocated for specifying the "type" for the next higher layer. Bytes 13 and 14 points the next layer (Internet Protocol, IP), and their hexadecimal value is 0x0800.

1.3.2.4. Which bytes in the IP header field tell that the next higher layer protocol is TCP? What is its hexadecimal value?

Each of the Internet Protocol/ IP Header field has 20 bytes of data, and the last 10^{th} byte of the field has one byte allocated for specifying the "protocol" for the next higher layer. The 10^{th} byte points the next layer (TCP layer), and their hexadecimal value is 6.

1.3.3 Networking Tools

Explore the usage of \ifcon g", \ping", \netstat", and answer the following questions. (Hint: If you are not sure about how to use these commands, please refer to \Sec. 1.1.3 Networking Tools".)

1. How many Ethernet interfaces are in your computer, and how to determine it?

To find the number of Ethernet interfaces on the computer, the 'ifconfig' command proves instrumental, offering a comprehensive overview of network interfaces. Specifically, the command **ifconfig -a** provides detailed information about all available interfaces. By utilizing a refined command like **ifconfig | grep -E '^en\d*:' | grep -v 'lo0**' one can filter out irrelevant details and focus solely on Ethernet interfaces.

```
~/Downloads/SecondYearEngineering git:(main)±2 (0.045s)
ifconfig | grep -E '^en\d*:' | grep -v 'lo0'
en3: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
en4: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
en1: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
en2: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
~/Downloads/SecondYearEngineering git:(main)±2
```

The provided code snippet, generated by the refined command, reveals the presence of multiple Ethernet interfaces:

- en3: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
- en4: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
- en1: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
- en2: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
- en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500

There are five distinct Ethernet interfaces on the computer, namely en0, en1, en2, en3 and en4.

(actual **iconfig -a** output)

```
~/Downloads/SecondYearEngineering git:(main)±2 (0.046s)
ifconfig -a
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
        nd6 options=201<PERFORMNUD,DAD>
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
anpi0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=400<CHANNEL_IO>
        ether 7e:9a:a6:3e:9b:17
        media: none
        status: inactive
anpi1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=400<CHANNEL_IO>
        ether 7e:9a:a6:3e:9b:18
        media: none
        status: inactive
en3: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=400<CHANNEL_IO>
        ether 7e:9a:a6:3e:9b:f7
        nd6 options=201<PERFORMNUD,DAD>
        media: none
        status: inactive
en4: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=400<CHANNEL_IO>
        ether 7e:9a:a6:3e:9b:f8
        nd6 options=201<PERFORMNUD,DAD>
        media: none
        status: inactive
en1: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        options=460<TSO4,TSO6,CHANNEL_IO>
        ether 36:ad:d9:0e:1f:c0
        media: autoselect <full-duplex>
        status: inactive
en2: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        options=460<TSO4,TSO6,CHANNEL_IO>
        ether 36:ad:d9:0e:1f:c4
        media: autoselect <full-duplex>
        status: inactive
ap1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=6460<TSO4,TSO6,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
        ether 72:ed:3c:0f:09:7f
        inet6 fe80::70ed:3cff:fe0f:97f%ap1 prefixlen 64 scopeid 0xa
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect (<unknown type>)
        status: inactive
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=6460<TSO4,TSO6,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
        ether 50:ed:3c:0f:09:7f
        inet 10.0.0.72 netmask 0xffffff00 broadcast 10.0.0.255
        inet6 fe80::8b0:87a9:657b:a601%en0 prefixlen 64 secured scopeid 0xb
        inet6 2604:3d08:2481:4f00:1801:790d:3af:1f31 prefixlen 64 autoconf secured
        inet6 2604:3d08:2481:4f00:51b2:f565:17de:3c26 prefixlen 64 deprecated autoconf temporary
        inet6 2604:3d08:2481:4f00::e68d prefixlen 64 dynamic
        inet6 2604:3d08:2481:4f00:18a1:72ab:c591:ca53 prefixlen 64 deprecated autoconf temporary
        inet6 2604:3d08:2481:4f00:1005:aa0d:9974:1994 prefixlen 64 autoconf temporary
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
        status: active
```

2. How to turn down/up an Ethernet interface?

To deactivate or "turn down" an Ethernet interface, we can use the command **sudo ifconfig enX down**, replacing **enX** with the specific interface name we want to deactivate, in our case en0, en1, en2, en3 and en4. The **sudo** command is necessary to gain administrative privileges required for network interface configuration. For example: **sudo ifconfig en0 down** effectively disables the Ethernet interface 0, terminating its network communication capabilities.

Conversely, to activate or "turn up" an Ethernet interface and restore its functionality, you use the command **sudo ifconfig enX up** replacing **enX** with the specific interface name we want to deactivate, in our case en0, en1, en2, en3 and en4. For example: **sudo ifconfig en3 up** effectively enables the Ethernet interface 3, reactivating its network communication capabilities.

3. Ping 10 packets to two websites. Compare the statistic results (i.e., the packet loss rate and average round-trip time).

Let's ping two websites: **www.google.ca and www.canada.ca**.

```
~/Downloads/SecondYearEngineering git:(main)±2 (9.171s)
ping -c 10 www.google.ca
PING www.google.ca (142.250.217.99): 56 data bytes
64 bytes from 142.250.217.99: icmp_seq=0 ttl=60 time=32.066 ms
64 bytes from 142.250.217.99: icmp_seq=1 ttl=60 time=23.884 ms
64 bytes from 142.250.217.99: icmp_seq=2 ttl=60 time=25.978 ms
64 bytes from 142.250.217.99: icmp_seq=3 ttl=60 time=29.575 ms
64 bytes from 142.250.217.99: icmp_seq=4 ttl=60 time=35.686 ms
64 bytes from 142.250.217.99: icmp_seq=5 ttl=60 time=26.022 ms
64 bytes from 142.250.217.99: icmp_seq=6 ttl=60 time=27.556 ms
64 bytes from 142.250.217.99: icmp_seq=7 ttl=60 time=23.232 ms
64 bytes from 142.250.217.99: icmp_seq=8 ttl=60 time=23.759 ms
64 bytes from 142.250.217.99: icmp_seq=9 ttl=60 time=48.632 ms

--- www.google.ca ping statistics ---
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 23.232/29.639/48.632/7.367 ms
```

```
~/Downloads/SecondYearEngineering git:(main)±2 (9.147s)
ping -c 10 www.canada.ca
PING e4073.dscb.akamaiedge.net (104.112.189.71): 56 data bytes
64 bytes from 104.112.189.71: icmp_seq=0 ttl=59 time=34.506 ms
64 bytes from 104.112.189.71: icmp_seq=1 ttl=59 time=25.116 ms
64 bytes from 104.112.189.71: icmp_seq=2 ttl=59 time=29.343 ms
64 bytes from 104.112.189.71: icmp_seq=3 ttl=59 time=25.308 ms
64 bytes from 104.112.189.71: icmp_seq=4 ttl=59 time=32.800 ms
64 bytes from 104.112.189.71: icmp_seq=5 ttl=59 time=35.164 ms
64 bytes from 104.112.189.71: icmp_seq=6 ttl=59 time=15.277 ms
64 bytes from 104.112.189.71: icmp_seq=7 ttl=59 time=20.579 ms
64 bytes from 104.112.189.71: icmp_seq=8 ttl=59 time=27.134 ms
64 bytes from 104.112.189.71: icmp_seq=9 ttl=59 time=29.484 ms

--- e4073.dscb.akamaiedge.net ping statistics ---
10 packets transmitted, 10 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 15.277/27.471/35.164/5.926 ms
```

None of the websites had any packet losses, which means that all 10 packets were sent and 10 received packets were also received from each web-domains. When pinging **Google**, the round-trip times range from *23.232 to 48.632 milliseconds*, with an average of *29.639 milliseconds* and a standard deviation of *7.367 milliseconds*. Conversely, when pinging **Government of Canada**, the round-trip times range from *15.277 to 35.164 milliseconds*, with an average of *27.471 milliseconds* and a standard deviation of *5.926 milliseconds*.

Average Round-Trip Time (RTT): The average RTT for pinging **www.google.ca** is 29.639 milliseconds, while the average RTT for pinging **www.canada.ca** is 27.471 milliseconds. Therefore, on average, pinging **www.canada.ca** has a slightly lower RTT compared to **www.google.ca**.

In summary, both websites demonstrate stable network connections with no packet loss observed during the 10-packet ping tests. However, **www.canada.ca** generally exhibits a slightly lower average round-trip time compared to **www.google.ca**.

Conclusion

In conclusion, this lab provided students with valuable hands-on experience in capturing and analyzing HTTP GET packets using Wireshark. By engaging with the tool, students gained familiarity with packet transmission processes between sources and destinations in computer networks. Moreover, they explored essential commands such as **wget** and **ping**, enabling them to manage ethernet connections and send packets to websites using only **Command Line Interface (CLI)**. The practical exercises helped students acquired foundational knowledge of Computer Network Packets as well as the functionality of Wireshark and its applications in capturing diverse packets across networks. This lab serves as a fundamental introduction to network packet analysis, empowering students to understand and utilize Wireshark effectively in their future lab works in this course.

Reference

[1] Wikipedia Contributors, "Transport Layer Security," *Wikipedia*, Mar. 26, 2019. https://en.wikipedia.org/wiki/Transport_Layer_Security (accessed Feb. 08, 2024).

[2] Wikipedia Contributors, "Transmission Control Protocol," *Wikipedia*, Mar. 15, 2019. https://en.wikipedia.org/wiki/Transmission_Control_Protocol (accessed Feb. 08, 2024).

[3] Wikipedia Contributors, "User Datagram Protocol," *Wikipedia*, Oct. 02, 2019. https://en.wikipedia.org/wiki/User_Datagram_Protocol (accessed Feb. 08, 2024).