ECE 458

Communication Networks

Laboratory Experiment #1 Report

Introduction

In this lab, we completed simple tasks to get familiar with the basic operations of WireShark, visualized how protocols and layering are represented in packets via sniffed packet traces and received a brief introduction to handy networking tools, such as; ping, ifconfig, netstat, and wget.

Procedure

The first part of the lab involved downloading WireShark and becoming familiar with its GUI and basic features. The four main WireShark fields: Filter field, Captured packets, Details of the selected packet, and Content of the packet in hex/ASCII were identified and used to capture a trace. With our acquired knowledge in layered protocols and WireShark we can analyze a captured trace. A figure of the HTTP GET packet was created to display the location and size of the TCP, IP, and Ethernet protocols.

Network tools such as ping, ifconfig, netstat, and wget were identified and used to determine the number of ethernet interfaces in our personal computers. In addition, the commands were used to ping 10 packets to two websites to observe their statistical results.

The following are tools used throughout the experiment:

- WireShark
- Ubuntu terminal

The following sections from the lab manual were also utilized through the experiment:

- 1.2.1
    - To properly download and install the WireShark application.
- 1.2.2
    - Introduction to the four main fields of the WireShark graphical user interface.

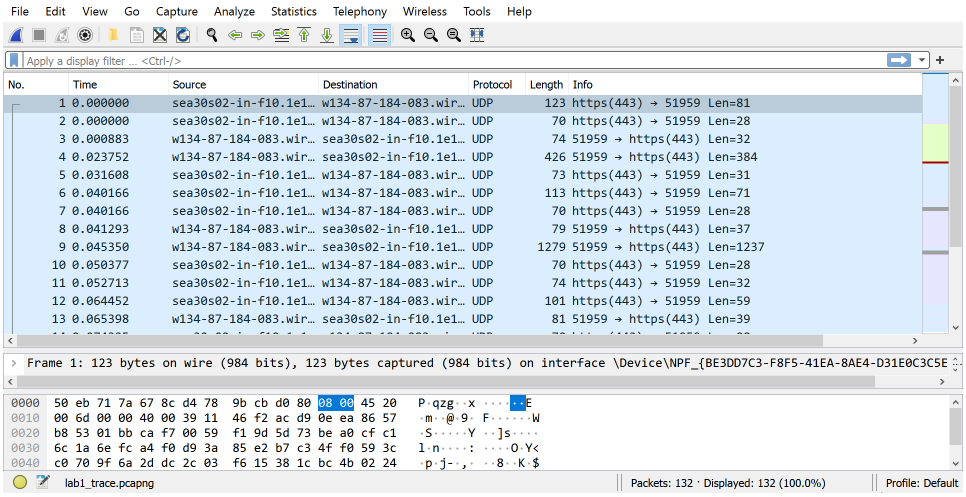
Pre-discussion Questions

1. The “demultiplexing key” field is used to determine the higher layer data unit.
2. See section 1.3.2 question 1, under discussion for the answer to this question

Discussion

1.3.1 Running WireShark

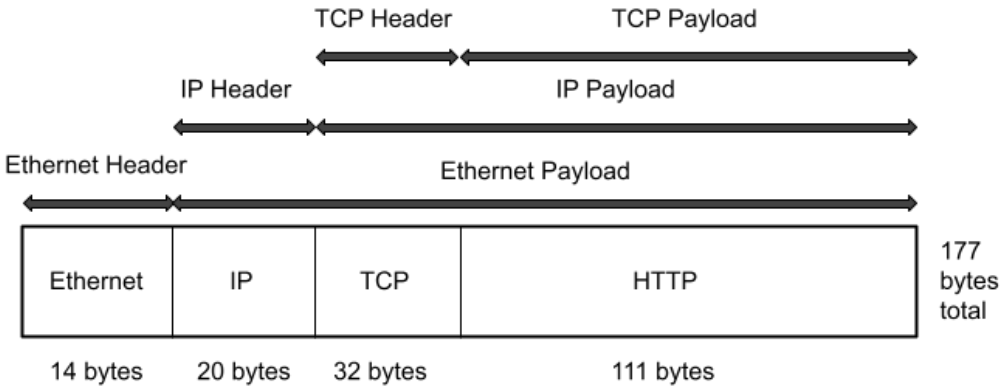
1. Below is a screenshot of a capture without any filters.



2. Three different protocols that appear in the protocol column of the unfiltered packet-listing window are the following:
    a. UDP
    b. TCP
    c. DNS
3. The HTTP GET message was sent at a timestamp of 3.007068s and the HTTP OK message was received at 3.080556s. Subtracting the HTTP GET timestamp from the HTTP OK message yields a value of **0.073488s**.

1.3.2 Layered Protocol

1. The following is a structure of an HTTP GET packet. The structure shows the location and size (in bytes) of the TCP, IP, and Ethernet protocols' headers. The structure also shows the range of the header and payload for each layer.



2. To calculate the average overhead of all the packets from the server to the client we can can do the following:

$$avg.\ overhead = \frac{sum\ of\ header\ sizes}{sum\ of\ packet\ sizes} = \frac{(total\ packet\ length) - (payload\ length)}{sum\ of\ packet\ sizes}$$

$$avg.\ overhead = \frac{(74*2)+(66*13)+((1484-1418)*7)+(1326-1260)+(642-576)+(177-111)+(216-150)}{(2*74)+(13*66)+(177)+(7*1484)+(642)+(1326)+(216)} \times 100$$

$$avg.\ overhead = 12.59\%$$

3. In the Ethernet header field, bytes 13 and 14 (type field) tell us that the next higher layer protocol is IP. The hexadecimal value is 0x0800.

4. In the IP header field, bytes 10 (protocol field) tell us that the next higher layer protocol is TCP. The hexadecimal value is 6.

1.3.3 Networking Tools

1. Below is the execution of "ifconfig."

```
bhavanvir@bhavanvir:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::bbb5:588e:c805:3226  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:c4:2a:1c  txqueuelen 1000  (Ethernet)
        RX packets 819  bytes 1152342 (1.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 434  bytes 32170 (32.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 156  bytes 13090 (13.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 156  bytes 13090 (13.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

   There is 1 ethernet interface on my computer. You can determine this by counting the number of "(Ethernet)" tags when running "ifconfig."

2. An ethernet interface can be turned down/up by running the following command: "ifconfig [NIC_NAME] Down/Up" where "[NIC_NAME]" refers to the network interface name.

3. Below are the outputs for the "ping [WEBSITE] -c -10" command, where "[WEBSITE]" refers to the target website url.

```
bhavanvir@bhavanvir:~$ ping www.yahoo.ca -c 10
PING src.g03.yahoodns.net (98.136.103.23) 56(84) bytes of data.
64 bytes from w2.src.vip.gq1.yahoo.com (98.136.103.23): icmp_seq=1 ttl=54 time=19.5 ms
64 bytes from w2.src.vip.gq1.yahoo.com (98.136.103.23): icmp_seq=2 ttl=54 time=19.6 ms
64 bytes from w2.src.vip.gq1.yahoo.com (98.136.103.23): icmp_seq=3 ttl=54 time=17.8 ms
64 bytes from w2.src.vip.gq1.yahoo.com (98.136.103.23): icmp_seq=4 ttl=54 time=14.7 ms
64 bytes from w2.src.vip.gq1.yahoo.com (98.136.103.23): icmp_seq=5 ttl=54 time=16.9 ms
64 bytes from w2.src.vip.gq1.yahoo.com (98.136.103.23): icmp_seq=6 ttl=54 time=15.0 ms
64 bytes from w2.src.vip.gq1.yahoo.com (98.136.103.23): icmp_seq=7 ttl=54 time=16.7 ms
64 bytes from w2.src.vip.gq1.yahoo.com (98.136.103.23): icmp_seq=8 ttl=54 time=17.3 ms
64 bytes from w2.src.vip.gq1.yahoo.com (98.136.103.23): icmp_seq=9 ttl=54 time=18.0 ms
64 bytes from w2.src.vip.gq1.yahoo.com (98.136.103.23): icmp_seq=10 ttl=54 time=20.5 ms

--- src.g03.yahoodns.net ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9036ms
rtt min/avg/max/mdev = 14.651/17.587/20.478/1.811 ms
bhavanvir@bhavanvir:~$
```

```
bhavanvir@bhavanvir:~$ ping www.google.ca -c 10
PING www.google.ca (142.250.217.99) 56(84) bytes of data.
64 bytes from sea09s30-in-f3.1e100.net (142.250.217.99): icmp_seq=1 ttl=118 time=12.6 ms
64 bytes from sea09s30-in-f3.1e100.net (142.250.217.99): icmp_seq=2 ttl=118 time=11.1 ms
64 bytes from sea09s30-in-f3.1e100.net (142.250.217.99): icmp_seq=3 ttl=118 time=14.7 ms
64 bytes from sea09s30-in-f3.1e100.net (142.250.217.99): icmp_seq=4 ttl=118 time=17.2 ms
64 bytes from sea09s30-in-f3.1e100.net (142.250.217.99): icmp_seq=5 ttl=118 time=12.6 ms
64 bytes from sea09s30-in-f3.1e100.net (142.250.217.99): icmp_seq=6 ttl=118 time=13.2 ms
64 bytes from sea09s30-in-f3.1e100.net (142.250.217.99): icmp_seq=7 ttl=118 time=13.4 ms
64 bytes from sea09s30-in-f3.1e100.net (142.250.217.99): icmp_seq=8 ttl=118 time=12.8 ms
64 bytes from sea09s30-in-f3.1e100.net (142.250.217.99): icmp_seq=9 ttl=118 time=14.6 ms
64 bytes from sea09s30-in-f3.1e100.net (142.250.217.99): icmp_seq=10 ttl=118 time=16.9 ms

--- www.google.ca ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9149ms
rtt min/avg/max/mdev = 11.104/13.900/17.202/1.847 ms
bhavanvir@bhavanvir:~$
```

   Both website pings feature 0% packet loss, but the ping for "www.google.ca" has a round-trip time that is 0.036ms longer than the round-trip time for "www.yahoo.ca."

Conclusion

WireShark's basic operations were covered in this lab. WireShark was used to collect, examine, and calculate the structure of an HTTP GET packet. We were familiarized with how packets are sent and received between source and destination. Using information gathered from WireShark, we calculated the average overhead of all the packages from the server to the client. We also learned to utilize a header field to determine the higher layer protocol. We also learnt and explored how to use commands from networking tools, such as:

- ping: send a packet from a source host to a target IP address.
- ifconfig: configure a network interface.
- netstat: display routing tables, interference statistics, and network connections.
- wget: fetch a URL.

The end result of completing this lab is a stronger understanding of WireShark, protocols, and networking tools.

References

[1] M. Maruthamuthu. “How to Enable (UP)/Disable (DOWN) Network Interface Port (NIC) in Linux?” 2daygeek.com. [Online]. Available: https://www.2daygeek.com/enable-disable-up-down-nic-network-interface-port-linux/

[2] J. Gerend, et al. “netstat” docs.microsoft.com. [Online]. Available: https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/netstat

Feedback

The questions in the lab manual are often difficult to interpret.