Department of Computer Science and Information Systems
COMPUTER NETWORKS (CS F303)
LAB-SHEET – 7
TOPIC: Transport Layer Protocol using Wireshark

Learning Objectives:

To understand the functioning of Transport Layer protocols (i.e. TCP and UDP)

In this lab, we'll investigate the behavior of the TCP protocol in detail. We will:

- Use of Filter Expressions in Wireshark to select the packets of our interest
- Analyze a trace of the TCP segments sent and received in transferring a 150KB text file from your computer to a remote server.
- Study TCP's use of sequence and acknowledgment numbers for providing reliable data transfer;
- Study TCP's congestion control algorithm – slow start and congestion avoidance – in action
- Analyze TCP's receiver-advertised flow control mechanism.
- Investigate the performance (throughput and round-trip time) of the TCP connection between your computer and the server.

Exercise 1: Filter Expression in Wireshark

Every field in the packet details pane can be used as a filter string. This will result in showing only the packets where this field exists. For example, the filter string: *tcp* will show all packets containing the TCP protocol. Wireshark provides a simple but powerful display filter language that allows you to build quite complex filter expressions. You can compare values in packets and combine expressions into more specific expressions.

Apply the following display filters to your capture and observe the effect of each on the display window:

```
tcp.port eq 80
tcp.port == 80 || tcp.port == 443
ip.addr == 172.22.24.65
ip.src==172.43.54.65 or ip.dst==172.43.54.65 (use IP address values from your capture)
ip.len == 1500
```

Display Filters Comparison Operators

English	C-like	Description and example
eq	==	Equal. ip.src==10.0.0.5
ne	!=	Not equal. ip.src!=10.0.0.5
gt	>	Greater than. frame.len > 10
lt	<	Less than. frame.len < 128
ge	>=	Greater than or equal to. frame.len ge 0x100
le	<=	Less than or equal to. frame.len <= 0x20
contains		Protocol, field or slice contains a value. sip.To contains "a1762"
matches	~	Protocol or text field matches Perl regular expression. http.host matches "acme\.(org|com|net)"
bitwise_and	&	Compare bit field value. tcp.flags & 0x02

Table: 1

Display Filters Logical Operations

English	C-like	Description and example
and	&&	Logical AND. `ip.src==10.0.0.5 and tcp.flags.fin`
or	||	Logical OR. `ip.scr==10.0.0.5 or ip.src==192.1.1.1`
xor	^^	Logical XOR. `tr.dst[0:3] == 0.6.29 xor tr.src[0:3] == 0.6.29`
not	!	Logical NOT. `not llc`

Table: 2

Note: The filter toolbar (Click on Expression button to view this) lets you quickly edit and apply display filters.

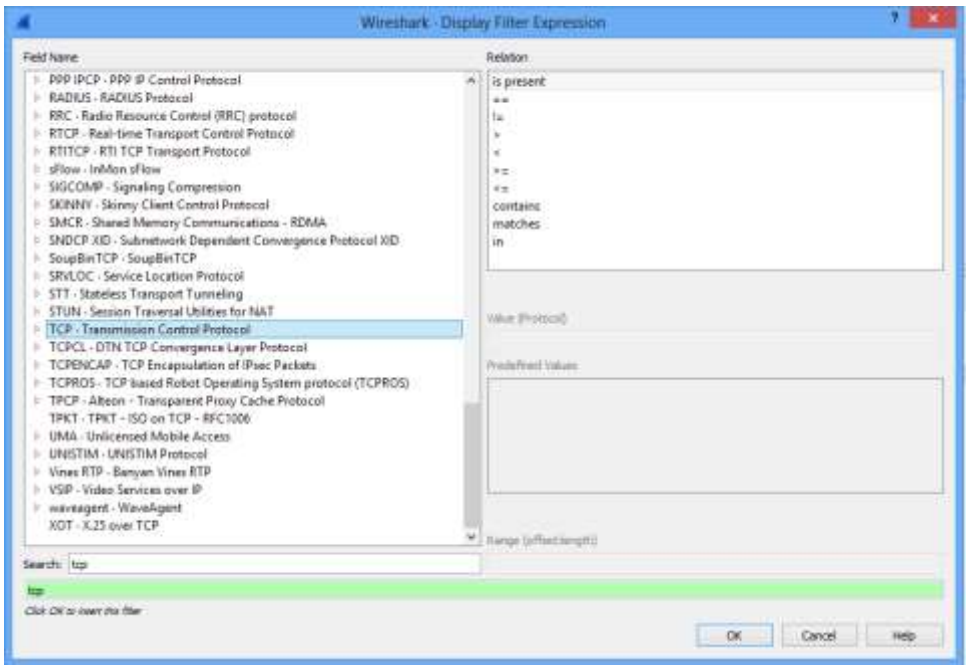

Fig. 1

Do as directed:

- Open the Wireshark program, then open a web browser and start capturing packets on your active network card.
- Type www.google.com in the address bar (URL bar) of your web browser and search for anything you want. Browse the top two-2-3 results given by google search. Further, click on URLs and images on the web pages.
- Stop Wireshark capture and close the browser completely.
- Analyze the captured packets and protocols and answer the following questions.
- Locate TCP handshake segments and find the sequence number of SYN, SYN+ACK, and ACK messages of all the TCP connections made by your computer.
- Find out the list of all TCP connections which have been reset.
- List all TCP segments which are sent and received by your machine having header lengths of more than 20 bytes.
- List all the duplicate ACK TCP segments.
- What filter will you use if you want to find the sequence number of any one out-of-order TCP segment captured in your trace file?

Exercise 2: Capturing a bulk TCP transfer from your computer to a remote server

We need to obtain a packet trace of the TCP transfer of a file from your computer to a remote web server using the HTTP POST method. We're using the POST method rather than the GET method as we'd like to transfer a large amount of data from your computer to another computer. Of course, we'll be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer.

Do as directed below:

- Start up your web browser. Go to the http://gaia.cs.umass.edu/wireshark-labs/alice.txt
- Retrieve an ASCII copy of *Alice in Wonderland*. Store this file somewhere on your computer.
- Next, go to http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html
- You should see a screen that looks like this:

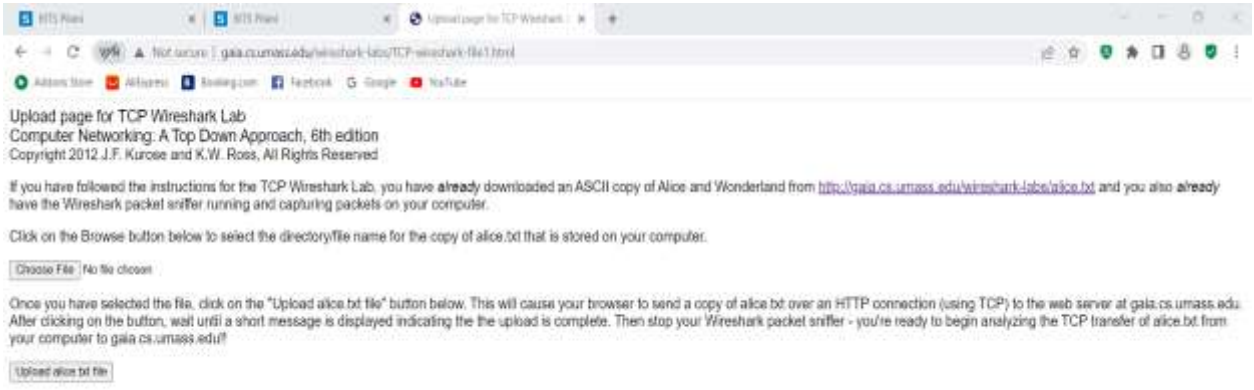

Fig. 2

- Use the *Browse* button in this form to enter the name of the file (full path name) on your computer containing *Alice in Wonderland* (or do so manually). Don't yet press the "*Upload alice.txt file*" button.
- Now start up Wireshark and begin packet capture *(Capture->Start)* and then press *OK* on the Wireshark Packet Capture Options screen (we'll not need to select any options here).
- Returning to your browser, press the "*Upload alice.txt file*" button to upload the file to the gaia.cs.umass.edu server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
- Stop Wireshark packet capture. Your Wireshark window should look similar to the window shown below in figure 3.

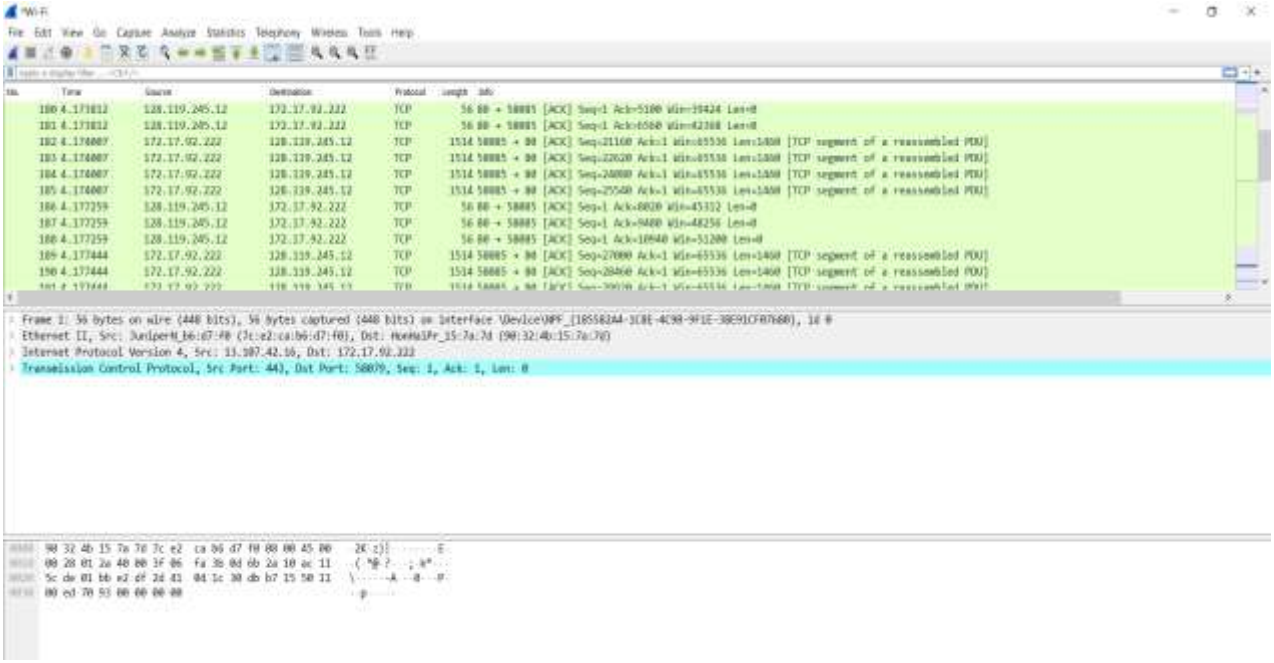

Fig. 3

Exercise 2: A first look at the captured trace

Filter the TCP packets and then try to analyze behavior of the TCP connection. Look for series of TCP and HTTP messages between your computer and gaia.cs.umass.edu. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message. You should also see TCP ACK segments being returned from gaia.cs.umass.edu to your computer.

Answer the following questions.

1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu?
2. Explore the details of TCP Packet used to carry the HTTP messages, using the "details of the selected packet header window".
3. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?

Note: To view information about the TCP segments containing the HTTP messages, rather than about the HTTP messages, select ***Analyze->Enabled Protocols***. Then uncheck the HTTP box and select *OK*. You should now see a Wireshark window that looks like:

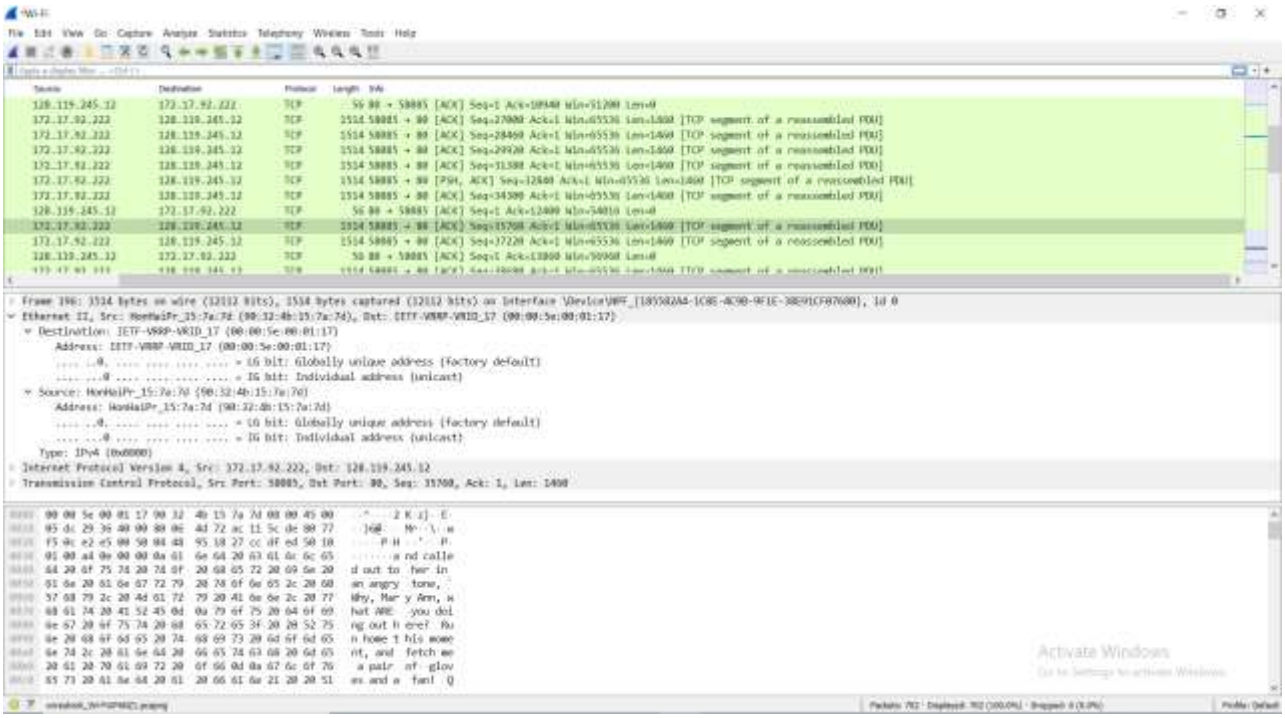

Fig. 4

Exercise 3: Understanding TCP Basics

Answer the following questions for the TCP segments:

4. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? Explore the segment that identifies itself as a SYN segment?
5. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?
6. What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field.
7. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the `EstimatedRTT` value after the receipt of each ACK? Assume that the value of the `EstimatedRTT` is equal to the measured RTT for the first segment, and then is computed using the `EstimatedRTT` equation for all subsequent segments.

Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the gaia.cs.umass.edu server. Then select: Statistics->TCP Stream Graph- >Round Trip Time Graph.

8. What is the length of each of the first six TCP segments?
9. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?
10. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?
11. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment?
12. What is the throughput (bytes transferred per unit time) for the TCP connection?

Exercise 4: TCP congestion control in action

Let's now examine the amount of data sent per unit time from the client to the server. We'll use one of Wireshark's TCP graphing utilities - *Time-Sequence-Graph(Stevens*) - to plot out data.
Select a TCP segment in the Wireshark's "listing of captured-packets" window. Then select the menu: *Statistics->TCP Stream Graph-> Time-Sequence Graph(Stevens*). You should see a plot that was created from the captured packets which can be described as follows:

- You should see a set of dots stacked above each other.
- Each dot in the plot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent.
- The stacked set of dots represents a series of packets that were sent back-to-back by the sender.

- Use the Time-Sequence-Graph (Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server.
- Identify where TCP's slow-start phase begins and ends, and where congestion avoidance takes over?

Exercise 5: UDP Packets Capturing and Analysis

Start capturing packets in Wireshark and then generate UDP packets (by clicking on www.youtube.com or you can simply generate DNS traffic). It's also likely that without explicitly doing anything, some UDP packets (sent by others) will appear in your trace. Note: Simple Network Management Protocol sends SNMP messages inside of UDP, so it's likely that you'll find some SNMP messages (and therefore UDP packets) in your trace.

Stop the packet capture and then filter the UDP packets sent and received at your host. Pick one of these UDP packets and expand the UDP fields in the details window.

Answer the following questions:

1. Select *one* UDP packet from your trace. From this packet, determine how many fields there are in the UDP header. Name these fields.
2. By consulting the displayed information in Wireshark's packet content field for this packet, determine the length (in bytes) of each of the UDP header fields.
3. The value in the Length field is the length of what?
4. What is the maximum number of bytes that can be included in a UDP payload? (Hint: the answer to this question can be determined by your answer to 2. above)
5. What is the largest possible source port number? (Hint: see the hint in 4.)
6. What is the protocol number for UDP? Give your answer in both hexadecimal and decimal notation. To answer this question, you'll need to look into the Protocol field of the IP datagram containing this UDP segment.
7. Examine a pair of UDP packets in which your host sends the first UDP packet and the second UDP packet is a reply to this first UDP packet. (Hint: for a second packet to be sent in response to a first packet, the sender of the first packet should be the destination of the second packet). What is the relationship between the port numbers in the two packets?

xx—oo—xx