**Lab 3: Wireshark – TCP**

CSC/CPE138 – Computer Network and Internet

Section 2: 9am – 10:15am

Professor Jun Dai

Vaukee Lee

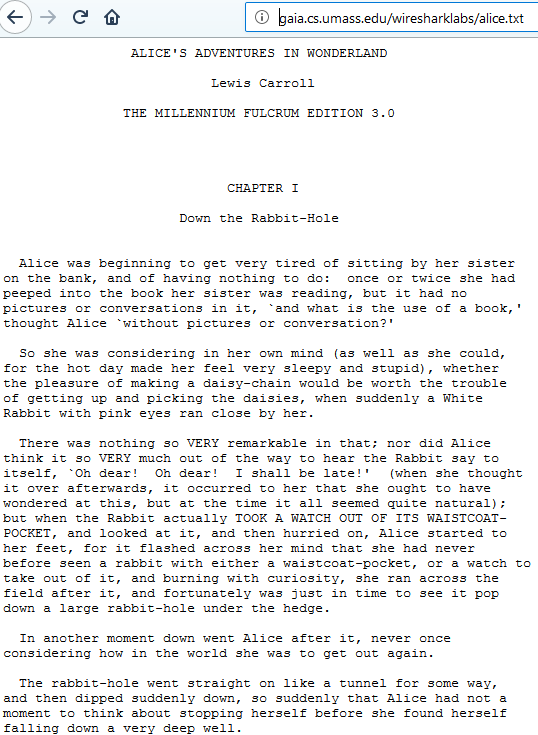
**Introduction:**

In this lab, we’ll investigate the behavior of the TCP protocol in closer detail by analyzing a trace of the TCP segments sent and received in transferring a 150KB file (containing the text of Lewis Carrol’s Alice’s Adventures in Wonderland) from a computer to a remote server. We’ll be looking at TCP’s use of sequence and acknowledgement numbers for providing reliable data transfer, congestion control algorithm – slow start and congestion avoidance – in action, and TCP’s receiver-advertised flow control mechanism. Finally, we will investigate the performance (throughput and round-trip time) of the TCP connection between the computer and the server.

**Part 1: Capturing a bulk TCP transfer from your computer to a remote server**

For this part, we will fetch the text file from the link (<http://gaia.cs.umass.edu/wiresharklabs/alice.txt>) and save the file to any spot in our computer. The POST method will be used rather than the GET method to transfer a large amount of data from a computer to another computer.

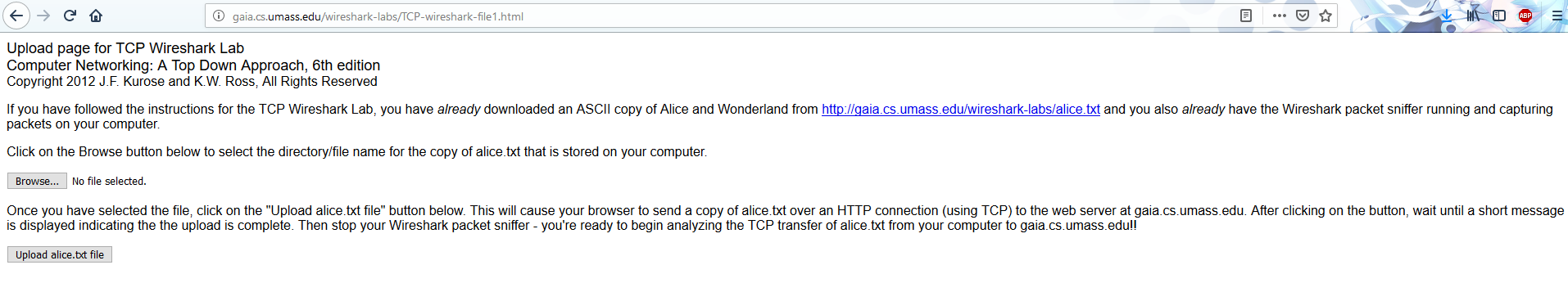
Step 1: Retrieve the text file Figure 3.1.1

The text file should look like this after you have copied and pasted the link.

Step 2: Go to provided link

(<http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html>) and you should see a similar page below. Note that the image below has been cropped for better fitment.

Figure 3.1.2



Step 3: Start up Wireshark

Once we have Wireshark up and running, we must begin the process of uploading the alice.txt file to the gaia.cs.umass.edu server. After the upload is done, a congratulation message will appear as seen in Figure 3.1.3, and we can then stop the packet capturing process in Wireshark.

Figure 3.1.3

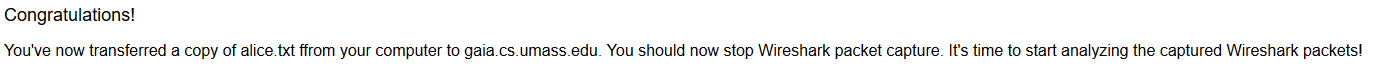
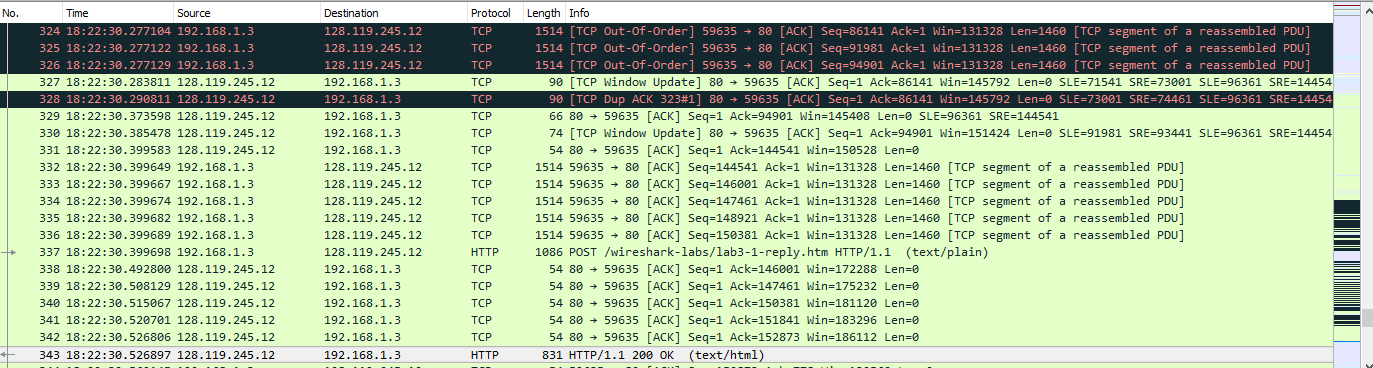


Figure 3.1.4

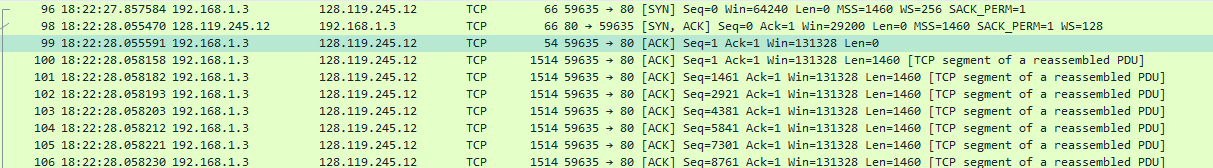


In this image from the Wireshark packet capturing, we can see that the packets were sent into segments and reassembled. There were many cases where the packet was duplicated and retransmitted.

**Part 2: First look into the captured trace**

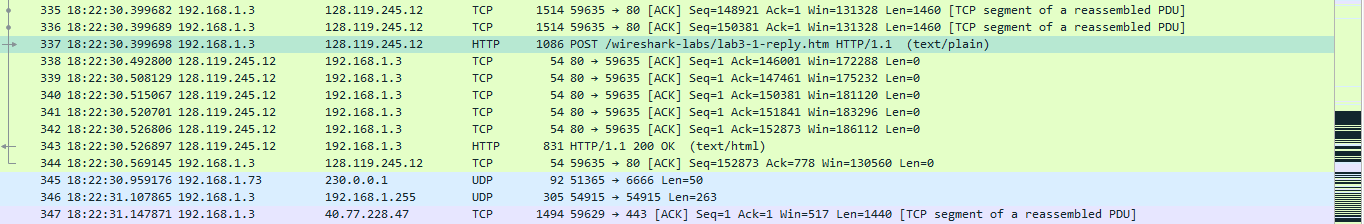
In this part, we will take a high-level view of the trace by filtering out the trace by typing in “tcp” all in lowercase.

Figure 3.2.1



This image shows the 3-way hand shake with the SYN message with many ACK messages.

Figure 3.2.2



This image shows the HTTP POST message and the TCP segment of a reassembled PDU.

**Questions and Answers**

Figure 3.2.3



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? To answer this question, it’s probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the “details of the selected packet header window” (refer to Figure 2 in the “Getting Started with Wireshark” Lab if you’re uncertain about the Wireshark windows.*

It looks like my port number is 59635 and IP address is 192.168.1.3.

2. *What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?*

gaia.cs.umass.edu

IP address = 128.119.245.12

Port # = 80

Question 3 – Same answer as Q1.

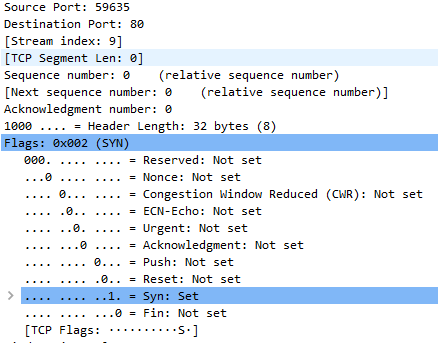
**Part 3 TCP Basics**

*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? What is it in the segment that*

*identifies the segment as a SYN segment?*

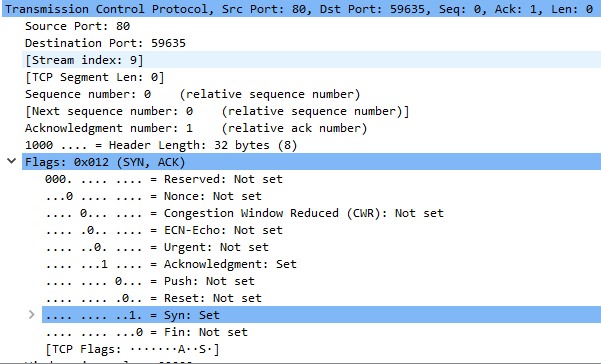
Figure 3.3.1



Sequence #: 0

The flags identify that the segment is 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?*



Sequence #: 0

ACK Field Value: 1

Value determined whether the “file” was received.

TCP flags identifies the SYNACK segment.

*6. What is the sequence number of the TCP segment containing the HTTP POST command?*

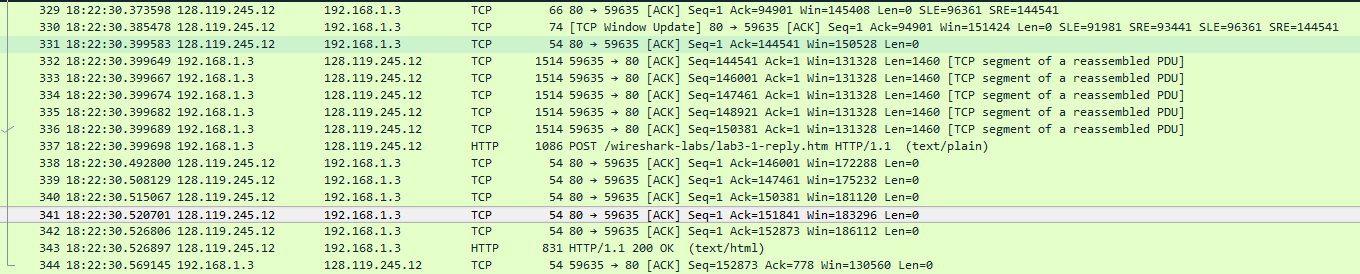
*Note that in order to find the POST command, you’ll need to dig into the packet content field*

*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 (see Section 3.5.3, page 239 in text) 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 on page 239 for all subsequent segments.*



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Sequence # | 1 | 144541 | 146001 | 147461 | 148921 | 150381 | 151841 |
| Time 6:22:30: | 399583 | 399649 | 399667 | 399674 | 399682 | 399689 | 399698 |
| Time Received | 0.014105 | 0.000066 | 0.000018 | 0.000007 | 0.000008 | 0.000007 | 0.000009 |
| iRTT Value | 0.198007 | 0.198007 | “ | “ | “ | “ | “ |
| RTT to ACK | 0.31251 |  |  |  |  |  |  |

Estimated Round Trip Time

EstimatedRTT = (1 – **α**) • EstimatedRTT + **α** • SampleRTT

Set **α** = 0.125 and set initial EstimatedRTT = 0

Solution:

ERTT = 0.875 \* 0 + 0.125 \* .31251 == 0.0406375

ERTT = 0.875 \* 0.0406375 + 0.125 \* 0.31251 == 0.074621

iterate to .198

*8. What is the length of each of the first six TCP segments?*

In bytes

w/ headers 54 -> 1514 -> 1514 -> 1514 -> 1514 -> 1514 -> 1086

Actual 0 -> 1460 -> 1460 -> 1460 -> 1460 -> 1460 -> 1032

*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?*

The minimum amount of buffer space (receiver window) advertised for the entire trace is 1460 bytes. This receiver window grows steadily until a maximum receiver buffer size of 64240 bytes. The sender is never throttled due to lacking receiver buffer space in this particular case.



*10. Are there any retransmitted segments in the trace file? What did you check for (in the trace)*

*in order to answer this question?*

Yes I looked for TCP Retransmission.

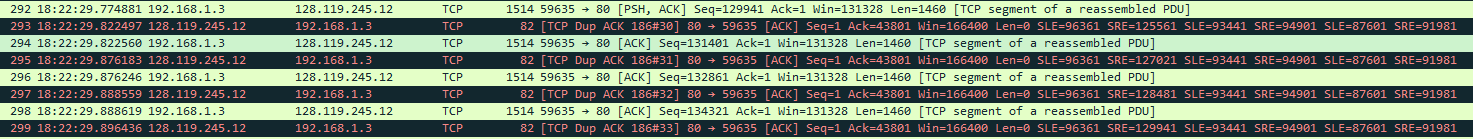


*11. How much data does the receiver typically acknowledge in an ACK? Can you identify cases*

*where the receiver is ACKing every other received segment (see Table 3.2 on page 247 in the*

*text).*

The length of data is around 1460 bytes.



This image is the case for a duplicate ACK.

*12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how*

*you calculated this value*.

Length of each packet is 1460 bytes

I will use the total length of the text file which is 152872 bytes

RTT is 0.198007

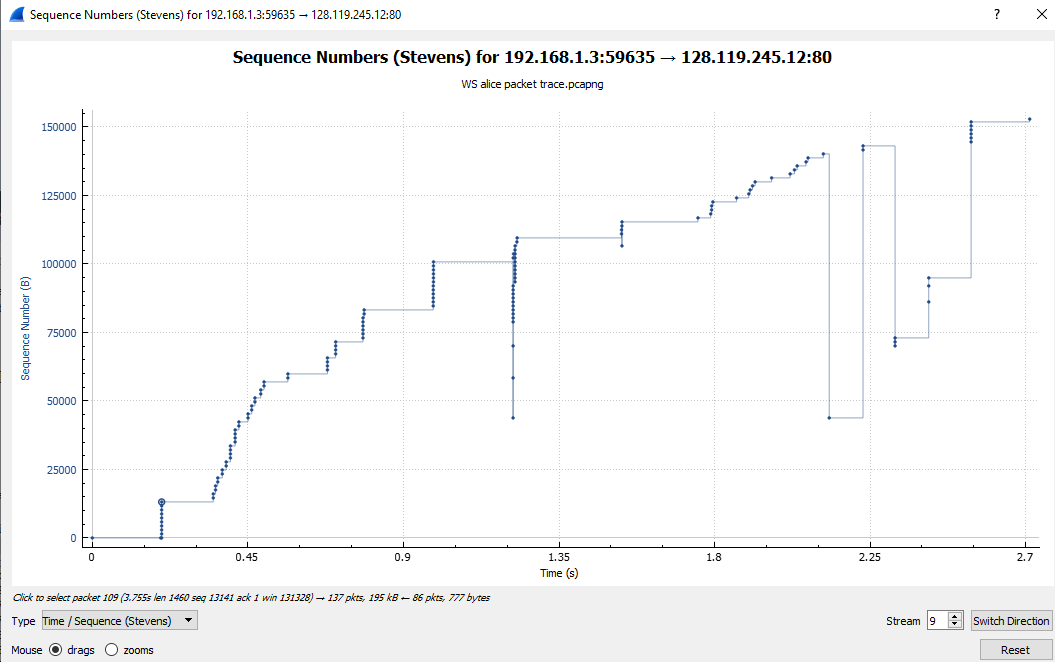
Throughput = Length / RTT

Throughput = 0.772 MB/Sec

**4. TCP Congestion Control in Action**

13. 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. Can you identify where TCP’s slow start phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we’ve studied in the text.

14. Answer each of two questions above for the trace that you have gathered when you transferred a file from your computer to gaia.cs.umass.edu.



Note: That this graph includes measurement from other sources along with the alice.txt file.

It looks like slow start starts from .7 seconds to the 1.8 second mark where there is a pause and then things start moving. The congestion avoidance would those linear bits in the graph around time of .3s to .5s and 1.8s and 2s then the fast recovery happens along the dips when there is congestion.

In the slow start phase, the graph should have look like an exponential graph, but we see pauses along the way making the average to look linear. The congestion avoidance seems to replicate what we learned along with the congestion points.

**Conclusion**

In conclusion, this lab was a great experience in seeing how TCP works in a real-life scenario. I was expecting to see the TCP behavior graph to be similar to the one we learned but there are extra things that packet sniffing picks up, so they too are recorded. It was pretty cool to see the crazy amounts of duplicate packet ACK occurring along with the retransmission ones.