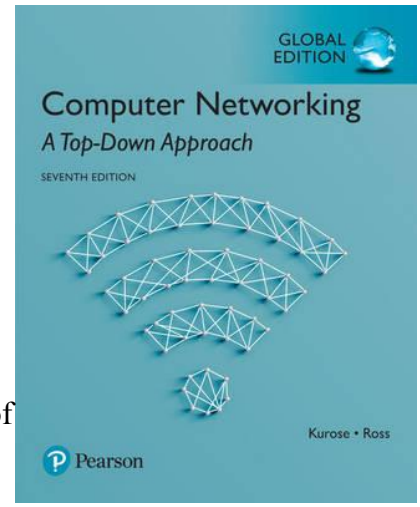
Wireshark TCP lab quiz results for Ngoc Dao

Correct answers will be available 28 Apr at 17:30 - 22 Jun at 17:00.

Score for this attempt: **12** out of 12
Submitted 27 Apr at 15:05
This attempt took 191 minutes.

Page 1 of 8



Adapted from Version: 2.0

In this lab, we'll investigate the behavior of TCP in detail. We'll do so 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 your computer to a remote server. We'll study TCP's use of sequence and acknowledgement numbers for providing reliable data transfer; we'll see TCP's congestion control algorithm - slow start and congestion avoidance - in action; and we'll look at TCP's receiver-advertised flow control mechanism. We'll also briefly consider TCP connection setup and we'll investigate the performance (throughput and round-trip time) of the TCP connection between your computer and the server.

Before beginning this lab, you'll probably want to review sections 3.5 and 3.7 in the text.

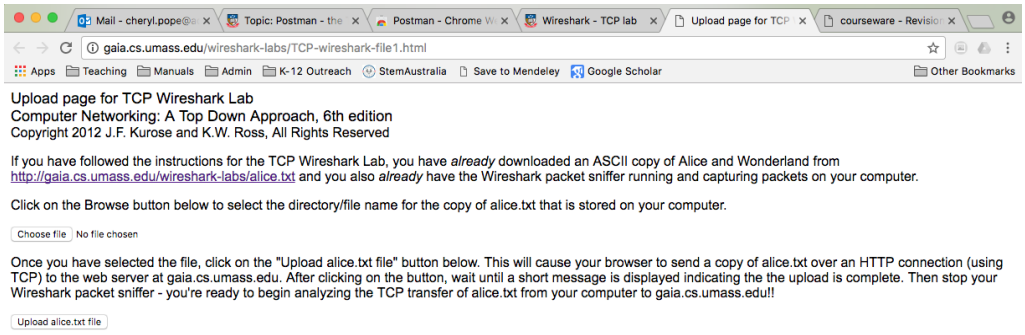
1. Capturing a bulk TCP transfer from your computer to a remote server

Before beginning our exploration of TCP, we'll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You'll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of Alice in Wonderland), and then transfer the file to a Web server using the HTTP POST method (see section 2.2.3 in the text). 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.

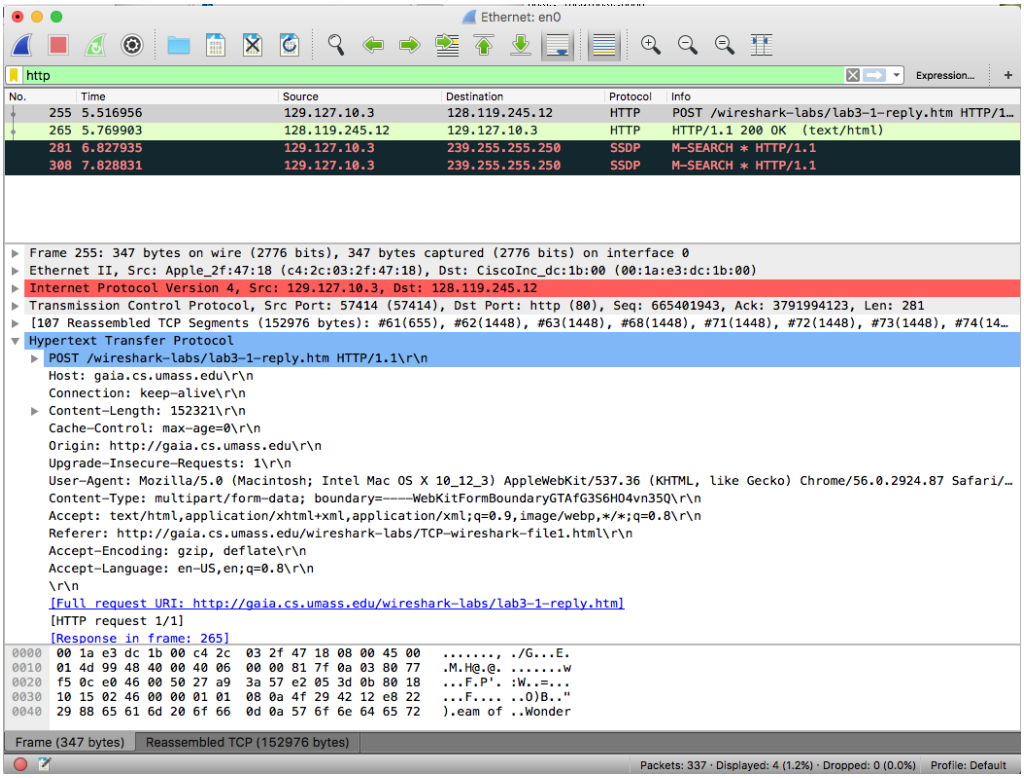
Do the following:

- Start up your web browser. Go the **http://gaia.cs.umass.edu/wireshark-labs/alice.txt (http://gaia.cs.umass.edu/wireshark-labs/alice.txt)** and 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 (http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html)** .
- You should see a screen that looks like:



(https://myuni.adelaide.edu.au/courses/85255/files/12505218/download?wrap=1)

- 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.
- 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.



Question 1

1 / 1 pts

Page 2 of 9

2. A first look at the captured trace

Before analyzing the behavior of the TCP connection in detail, let's take a high level view of the trace.

- First, filter the packets displayed in the Wireshark window by entering "tcp” (lowercase, no quotes, and don't forget to press return after entering!) into the display filter specification window towards the top of the Wireshark window.

What you should see is series of TCP and HTTP messages between your computer and gaia.cs.umass.edu or your web proxy, if you are going through a proxy. There are three main sequence of events you should see:

1. You should see the *initial three-way handshake containing a SYN message*.
2. You should see an HTTP POST message and (possibly depending on your host) a series of "HTTP Continuation” messages being sent

from your computer to gaia.cs.umass.edu (or the proxy). Recall from our discussion in the earlier HTTP Wireshark lab, that is no such thing as an HTTP Continuation message - this is Wireshark's way of indicating that there are multiple TCP segments being used to carry a single HTTP message.

3. You should also see TCP ACK segments being returned from gaia.cs.umass.edu (or the proxy) to your computer acknowledging the TCP segments your computer sent that have been received.
4. You should see the HTTP 200 OK being returned from the server in response to the post.

Make sure you can identify these three types of events before proceeding. If your capture is large, it may be helpful to filter the packets by Destination (i.e. tcp && ip.addr == a.b.c.d where a.b.c.d is the destination address for the POST) so that you can focus on the packets sent to/from gaia.

Note the IP address and port numbers of both the source and destination of the POST request and the response.

Match the items below that have the same value

source port of POST request	destination port of ac
source port of acknowledgments (ACKs)	destination port of P
source IP of POST request	destination IP of ack
destination IP of POST request	source IP of acknow

The IP address in a TCP segment identifies the host the segment is from (source IP) or to (destination IP). The port number identifies which application process on the host the segment is from (source port) or to (destination port)

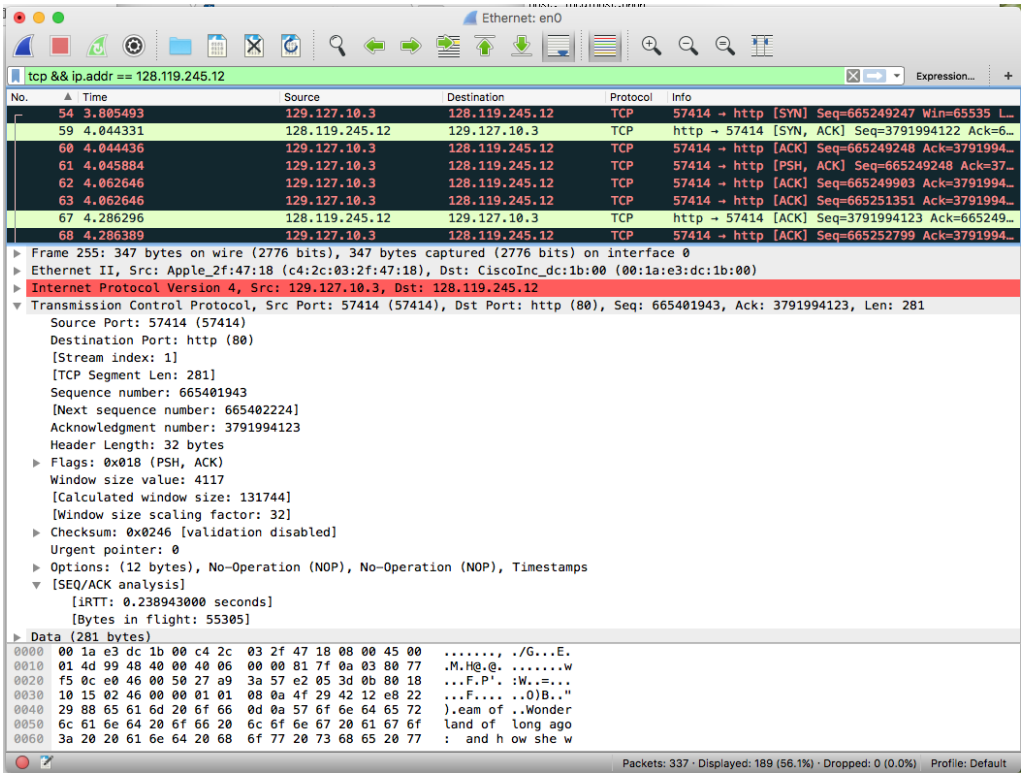
As an analogy, consider you are sending a letter. You address it to from <my name> at <my family house>; to John Smith at Smith House. When John replies he addresses his reply as from John Smith at Smith House; to <my name> at <my family house>

Question 2

1 / 1 pts

Page 3 of 9

Since this lab is about TCP rather than HTTP, let's change Wireshark's "listing of captured packets" window so that it shows information about the TCP segments containing the HTTP messages, rather than about the HTTP messages. To have Wireshark do this, select Analyze->Enabled Protocols. Then uncheck the HTTP box and select OK. You should now see an Wireshark window that looks like:



This is what we're looking for - a series of TCP segments sent between your computer and gaia.cs.umass.edu. We will use the packet trace that you have captured to study TCP behaviour in the rest of this lab.

3. TCP connection establishment

Examine what is carried in the first 3 TCP segments sent. Compare the header fields (sequence number, port numbers, window size, checksum, etc). Be sure to note what Flags are set. Look at the data (if any) the segments contain.

By default, wireshark will show you relative sequence numbers (ie a connection will appear to start with sequence number 0). To get the real sequence numbers contained in the TCP segments, you need to choose preferences from the wireshark menu and choose TCP under protocols. Make sure "Relative sequence numbers" is unchecked. Click OK.

If you are seeing several Bad Checksums, read **http://wiki.wireshark.org/TCP_Checksum_Verification (http://wiki.wireshark.org/TCP_Checksum_Verification)**

Recall that when writing TCP socket code, we actually create 2 sockets. The first that accepts new connections, and the second that accepts data from the client and is created when the new connection is established.

Before the POST is sent, this connection establishment must take place. Looking at the connection establishment segments. What control bit in the header allows the server to know whether to deliver this segment to the ServerSocket (the one to which connection requests are sent) or the Socket (the socket created for communication once the connection is established)

SYN

A common misconception is that the destination port is different. But inspecting the connection request and other segments from the client, you can see that the only header field that is the same in all segments from the client that are not connection requests and different in connection requests is the SYN Flag being set to 1 during the connection request.

A second common misconception is thinking the sequence number indicates it is a connection request. But as you can see from inspecting the segments, the client's choice of initial sequence number does not give the server any clue to this being the first segment.

Question 3

1 / 1 pts

Page 4 of 9

Sequence numbers are critical in reliable protocols that can resend segments to allow the receiver to identify whether a received segment is a duplicate that has been resent or whether it is a new segment. They are also needed for re-ordering segments if the network does not guarantee in order delivery (the Internet network layer does not guarantee in order delivery).

Examine the sequence numbers used by the client and server in the connection set up and the sending of the alice.txt file.

Which of the following statements are true?

Select all that apply.

- [ ] The initial sequence number of the client and server are the same
- [x] neither of the other statements are true

TCP connections do not start at zero as this would increase the chance that an old duplicate request (that also started at 0) might be mistaken for a new request.

The starting sequence numbers of the client and server are chosen independent of each other. Although it is *possible* for the sender and receiver to by chance choose the same initial sequence, it is highly unlikely.

The initial sequence number of the connection is 0

Question 4

1 / 1 pts

Page 5 of 9

TCP uses acknowledgements to make sure segments are received. If an acknowledgement is not received within a "reasonable" amount of time (we'll talk about what is "reasonable" shortly), then the sender will send the segment again.

Look at the acknowledgement (ACK) segments in your Wireshark window. Under the "info" column they will show as [ACK] which indicates the ACK flag has been set to 1 in this segment. Check the Flags of an ACK packet to see that this is the case.

Choose an ACK segment sent from gaia.cs.umass.edu (or the proxy) in response to part of your POST message. Find the corresponding TCP segment that it acknowledges. Your intuition may lead you astray here, so you may want to choose the SEQ/ACK analysis in the packet detail window to see what segment is being acknowledged to make sure you have found a matching SEQ/ACK pair.

Examine the sequence number and acknowledgement number in the original segment and it's corresponding acknowledgement.

Which of the following statements is correct about the segment and its acknowledgment?

The ACK number of the acknowledgment is equal to the Sequence number of the segment *after* the one being acknowledged.

TCP's ACK number equals the next byte it is expecting, not what it received.

The ACK number of the acknowledgment is equal to the Sequence number of the segment being acknowledged.

- The ACK number of the acknowledgment is equal to the Sequence number of the segment *before* the one being acknowledged.

Question 5

1 / 1 pts

Look through capture. Does the client ever resend any segments due to not receiving an ACK?

Which of the following parts of the segment would indicate that the segment is definitely a retransmission?

Select all that apply.

- [ ] IP and port number seen before
- [ ] retransmit flag
- [ ] data/payload
- [x] sequence number

Question 6

1 / 1 pts

Look at the ACKs being returned by the server in response to the segments carrying parts of the alice.txt file. Note that these segments from the server do not contain any data (the server is not sending anything to the browser, the browser is sending alice.txt to the server).

Look carefully at the sequence numbers of these segments containing the ACKnowledgements.

Do these ACKs from the server change which byte of data the *client* is expecting (should the server suddenly decide to send data)?

NOTE: Since there are no YES and no options. Select True for YES and False for NO

- True
- False

As they contain no data, they do not change the byte the client is expecting and therefore, the sequence number does not change (note that all the ACKs have the same sequence number).

Note however, that the SYN ACK from the server does increase the initial sequence number by 1. This helps the client and server to ensure reliability of the initial handshake (no old duplicate SYN ACKs or SYNs, if either side sees something unexpected the connection is reset)

Question 7

1 / 1 pts

Page 6 of 9

If a reliable sender does not receive an ACKnowledgement, the segment may have been lost and the sender will resend it. But how long should the sender wait before resending?

If the sender chooses too short a time, it will resend segments unnecessarily (the ACK will arrive shortly after resending) wasting network resources.

If the sender chooses too long a time, it will slow the transmission unnecessarily.

So what would be reasonable? Clearly we want to wait for at least the amount of time it is taking typical segments to be acknowledged. If it takes about 2 days to get a letter from our friend in Sydney, we aren't going to be phoning him up after one day asking where the letter is. So as a minimum, the sender should wait whatever the current typical round trip time (RTT).

To estimate the typical RTT we need to keep track of our past RTTs and

update it with new RTTs. The equation:

EstimatedRTT = (1-a) * EstimatedRTT + a * SampleRTT

captures this, where 'a' is a weighting that determines the influence the past estimate or the latest sample have on the current value. For TCP the recommended value is a = 0.125, giving

EstimatedRTT = 0.875 * EstimatedRTT + 0.125 * SampleRTT

In our letter example above, it is unlikely that we will call our friend exactly after the second day, even if that is the typical amount of time a letter from Sydney takes. After all, we know that there is some variation in postal delivery times. Likewise, we know that there is some variation in the delays on the Internet. So it makes sense to leave a little extra time, based on the expected amount of variation.

We estimate the variability in RTT with a similar equation

DevRTT = (1-b) * DevRTT + b * | SampleRTT - EstimatedRTT |

the recommended value for b is 0.25, giving

DevRTT = .75 * DevRTT + .25 * | SampleRTT - EstimatedRTT |

So the total amount of time the sender will wait (called the timeout interval) is:

TimeoutInterval = EstimatedRTT + n * DevRTT

in TCP, 'n' is set to 4, giving

TimeoutInterval = EstimatedRTT + 4 * DevRTT

We will look at the calculation of the timeout interval in the next question.

Download the file **tcp-post-capture (https://myuni.adelaide.edu.au/courses/85255/files/12505008/download?wrap=1)** ↓ **(https://myuni.adelaide.edu.au/courses/85255/files/12505008/download?download_frd=1)** and open it in Wireshark. In it you will find a capture of a POST of the alice.txt

Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. You may have to look in the packet contents at the bottom of the Wireshark window to find the POST.

Look at what time the first 3 segments (including the HTTP POST as the first) were sent and when the ACK for each of these segments was received. Be careful here, the receiver may send multiple ACKs for a segment with each ACK acknowledging *part* of the segment (remember an ACK in TCP says what byte is expected next, not what segment has been received). This behaviour is known as "ACK division". The segment is not considered acknowledged until *all* of the data sent in that segment is acknowledged.

Given the difference between when each TCP segment was sent, and when its acknowledgement was received, calculate the RTT value for each of the three segment/ack pairs.

Calculate 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 the DevRTT is equal to 1/2 the measured RTT, and then is computed using the EstimatedRTT equation for all subsequent segments.

What is the Timeout Interval after the first 3 HTTP segments (to 4 decimal places ie x.xxxx)?

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. However, note that it shows the RTT of each ACK not each segment. So if the server returns multiple ACKs acknowledging partial receipt of segments, called "ACK division" as described above, you will see a point for each of these partial ACKs.

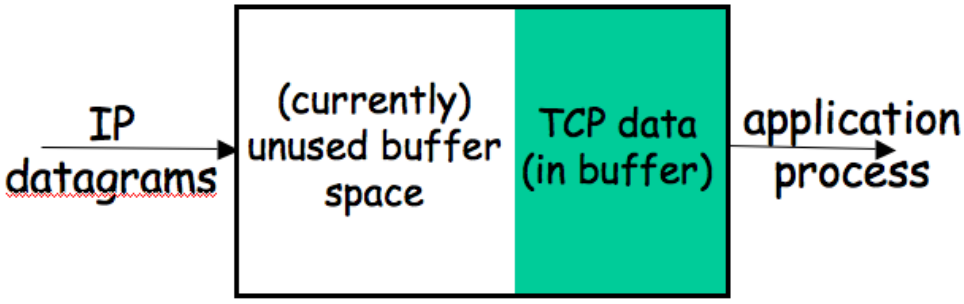
0.6381

Question 8

1 / 1 pts

Page 7 of 9

Flow control is used by the sender to limit their sending rate to what the receiver is capable of receiving. A receiver does not have unlimited buffer space to store segments waiting for the application to read them. As shown in the picture below, datagrams come in from the network and are stored in the buffer until delivered to the application. If the application reads at a slower rate than the datagrams arrive, this buffer will begin to fill. Once it is full, the datagrams will have to be dropped.



Clearly there is no point in the sender sending more data than the receiver has space to store. But how can the sender know how much buffer space the receiver has?

It's also important to note that both ends of a TCP connection can send data. For example a web server can return web pages over a connection, but the client can also send data through a POST over the same connection. So the client needs to know how much data the server is able to receive and the server needs to know how much data the client can receive.

Look carefully at the header fields in the TCP segments sent while transmitting the alice.txt file.

What header field is used by the sender of a segment to tell the other end how much buffer space it has left?

Window size

The Window Size header indicates how much space is left.

Question 9

1 / 1 pts

The host on each side of the TCP connection sends its window size during connection set up and this window size is used for the life of the connection.

- True
- False

As the amount of data stored in the buffer and rate at which the application reads the data can change over the lifetime of the connection, the window size is constantly being updated and its current value is sent with *every* segment.

Question 10

1 / 1 pts

Look at the first three segments and the acknowledgements of the POST again in the capture file *tcp-post-capture.*

Look at the maximum size of the segments (in Bytes) and the minimum receive window size in the acknowledgements(use the calculated window size value that wireshark has calculated for you as the size in Bytes of the window).

Is the sender forced to reduce the sending rate due to lack of buffer space in the receive window?

Before answering, check the SYN ACK packet from the server and look at the options fields. It is using a TCP option known as window scaling which allows the receiver to express larger window sizes than 65,535 (the largest number you can represent with the 16 bit window size field). Window scaling multiplies the window size in the header by a multiple.

- True
- False

The smallest advertised window is 2172 bytes and this needs to be multiplied by 4 from the scaling option. The sender always sends less than the advertised window, so the receiver window size is not constraining it.

Question 11

1 / 1 pts

Page 8 of 9

4. TCP congestion control in action

Let's now examine the amount of data sent per unit time from the client to the server. Rather than (tediously!) calculating this from the raw data in the Wireshark window, we'll use one of Wireshark's TCP graphing utilities - Time-Sequence-Graph(Stevens) - to plot out data.

- Open the *tcp-post-capture* file in Wireshark, if you don't already have it open.
- 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).

Each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stacked above each other represents a series of packets that were sent back-to-back by the sender. You may also want to look at the tcptrace-style graph.

Does the connection leave slow-start and enter congestion avoidance?

- (●) True

When the window size reaches 65,535 Bytes, we can see the changeover and more linear growth. Note that the graph will not be smooth like our idealized graphs as the RTT of the segments and their ACKs are not all equal.

- ( ) False

Question 12

1 / 1 pts

Does the TCP connection in the *tcp-post-capture* file experience significant congestion in the network?

- True
- False

No timeouts occur, as we do not see the congestion window drop back at any point.

Quiz score: **12** out of 12