Lab #4: TCP

Name(s): Aidan Sharpe

Date: April 17th, 2023

Based on Version: July 2005

© 2005 J.F. Kurose, K.W. Ross. All Rights Reserved

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 uploading 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.

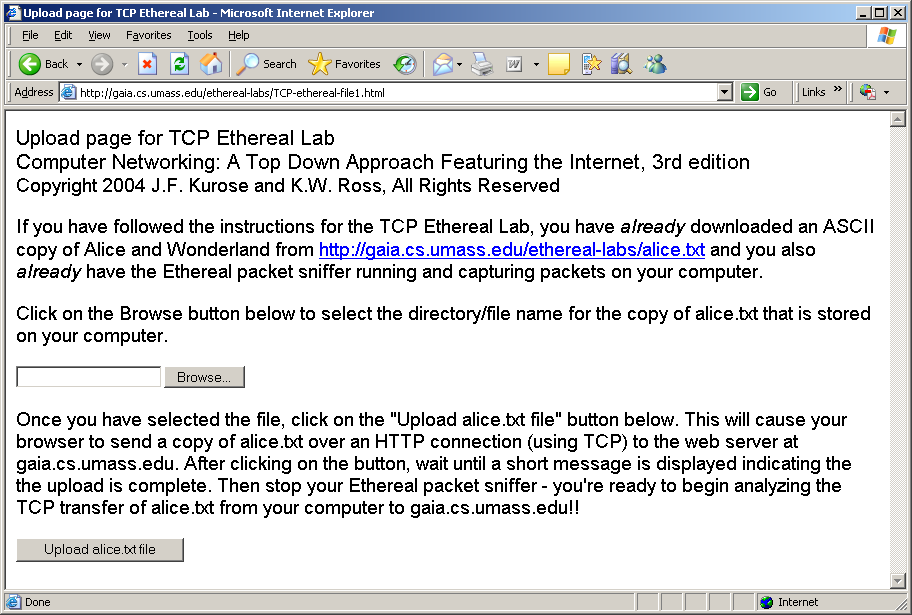
Before beginning this lab, you’ll probably want to review sections 3.5 and 3.7 in the text.[[1]](#footnote-2)

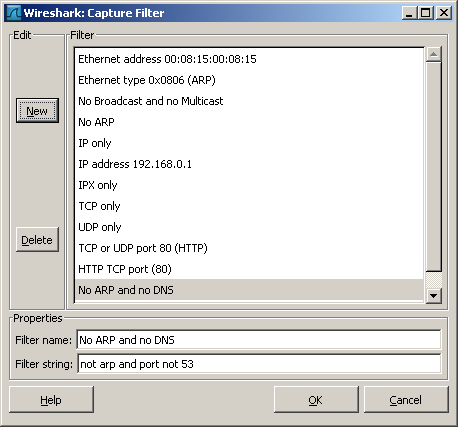
1. Capturing a bulk TCP transfer from your computer to a remote server

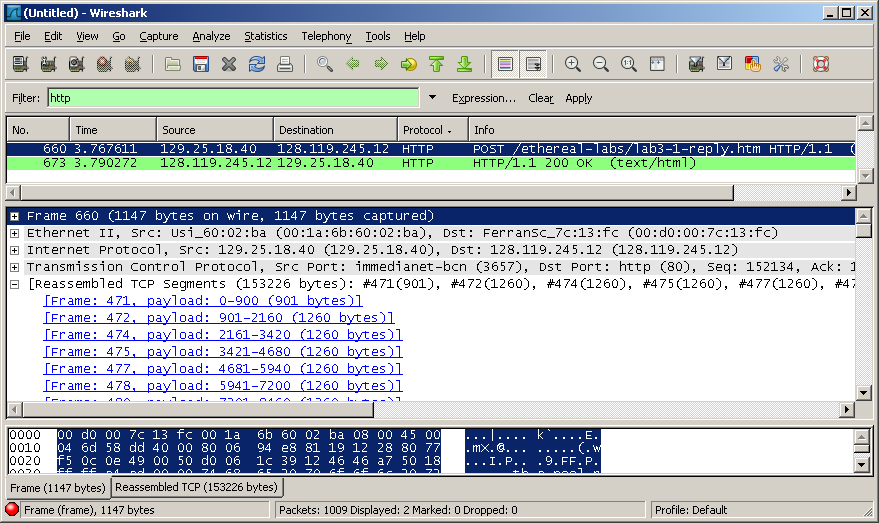
Before beginning our exploration of TCP, we’ll need to use WireShark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You’ll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text 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:

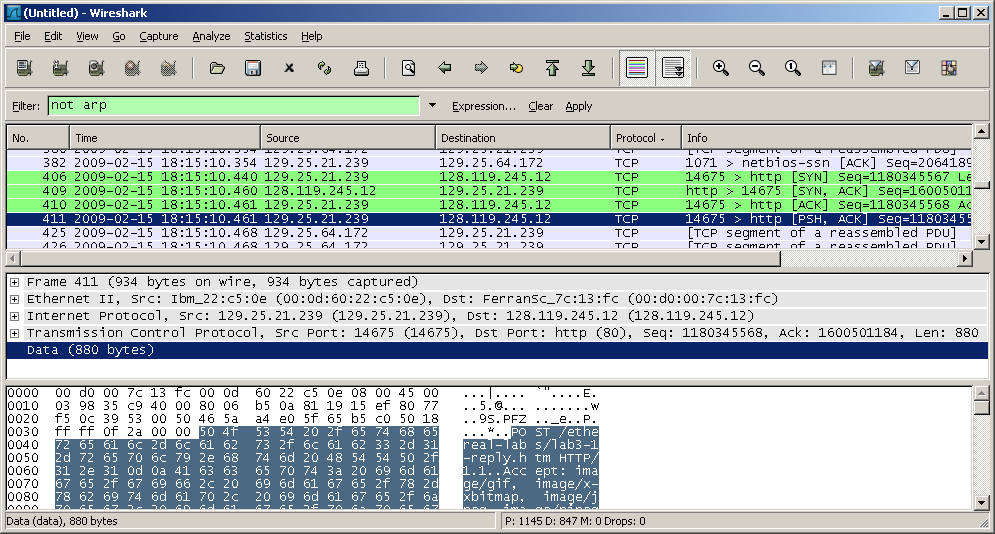
* Retrieve an ASCII copy of *Alice in Wonderland* at <http://gaia.cs.umass.edu/ethereal-labs/alice.txt> . Store this file somewhere on your computer; your Desktop is fine.
* Next go to <http://gaia.cs.umass.edu/ethereal-labs/TCP-ethereal-file1.html> using the web browser of your choice.
* You should see a screen that looks like this:



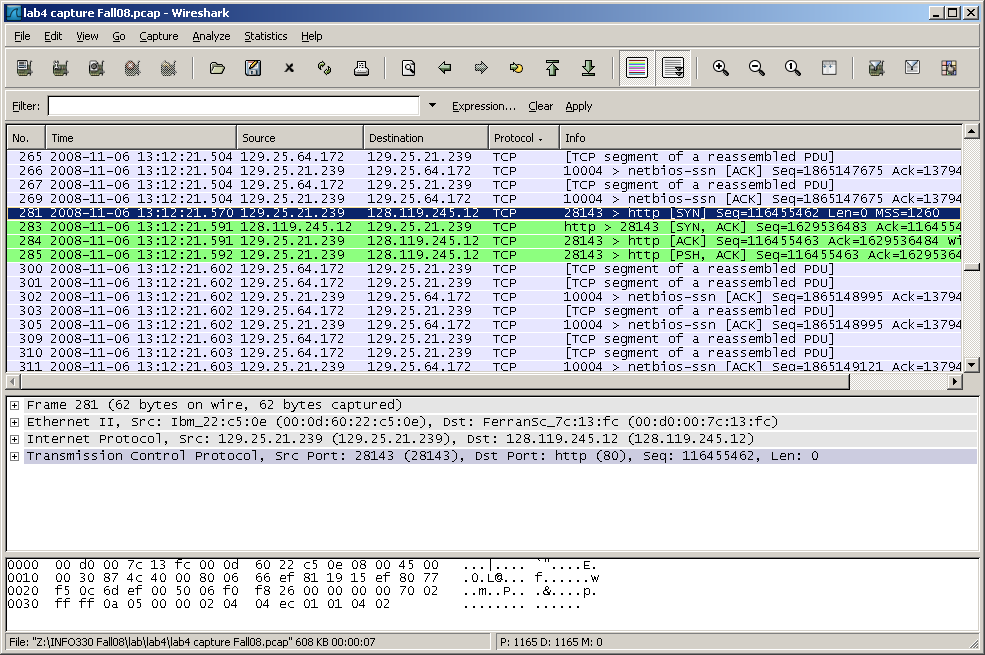
* Use the *Browse* button in this form to enter the name of the file (full path name) on your computer containing *Alice in Wonderland*.**Do NOT yet press the “*Upload alice.txt file*” button.**
* Now prepare to start packet capture with WireShark.
  + As usual, **shut off promiscuous mode** in the Capture > Options.
  + Select the adapter you’re using. It’s the Intel choice in the Rush 205 lab.
  + You can select a Capture Filter of “No ARP and no DNS”, which displays as “not arp and port not 53”. This step is optional, it just cuts down the extraneous data in your file.  
    
  + **Start the 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**. You should have an HTTP message which POSTs the file. Your WireShark window should look similar to the window shown below.  
    Notice that the HTTP POST command in this example summarizes the separate TCP segments it assembled to upload the file – packet #471 (901 bytes), #472 (1260 B), #474 (1260 B), etc.



**You might see no HTTP protocol messages**. In that case, look for TCP messages which contain the [SYN], [SYN,ACK] and [ACK] messages. The TCP packets which follow are posting the file, like the next example which has the HTTP POST command.



If you are unable to run WireShark on a live network connection, you can download a packet trace file that was captured while following the steps above on one of the author’s computers[[2]](#footnote-3). You may well find it valuable to download this trace even if you’ve captured your own trace and use it, as well as your own trace, when you explore the questions below.



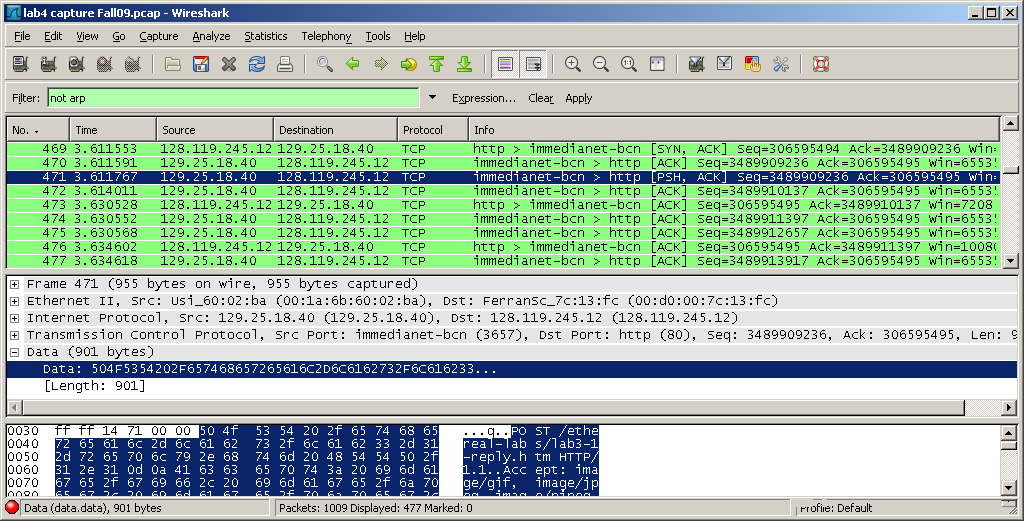
2. A first look at the captured trace

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” into the display filter specification window towards the top of the WireShark window. (If this doesn’t work well, sort by Protocol, and focus on the TCP messages.)

What you should see is 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 and *might* see a series of “HTTP Continuation” messages being sent from your computer to gaia.cs.umass.edu. 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. You should also see TCP ACK segments being returned from gaia.cs.umass.edu to your computer.

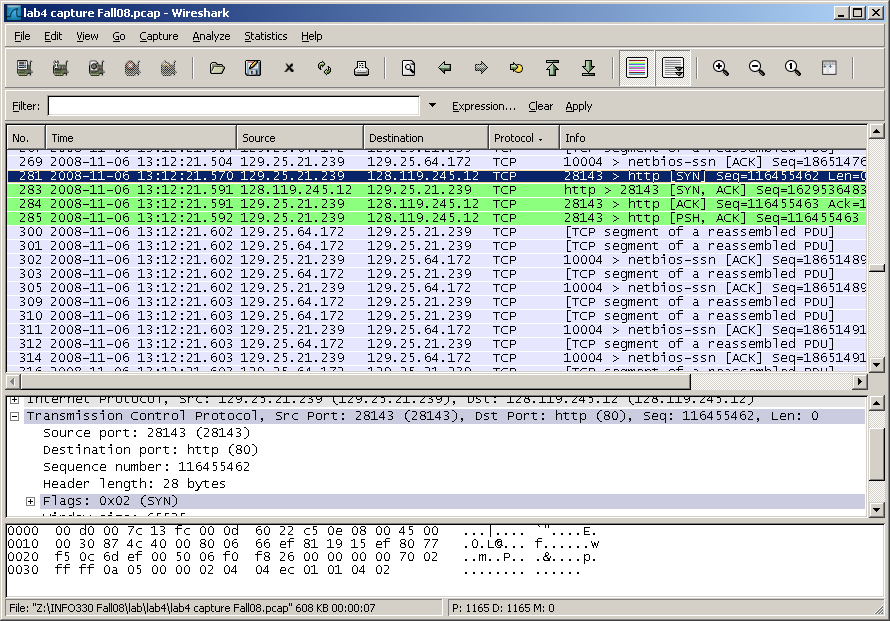
Whenever possible, when answering a question you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. Annotate the printout to explain your answer. To print a packet, use *File->Print*, choose *Selected packet only*, choose *Packet summary line,* and select the minimum amount of packet detail that you need to answer the question. (Or use a screen capture, Alt-Print Scr.  )

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*. Also, in this lab we would like to see TCP’s sequence numbers (and not the relative sequence numbers that WireShark may instead display). To see the sequence numbers, go to Edit > Preferences > Protocols > TCP and uncheck “Relative sequence numbers and window scaling” and uncheck “Analyze TCP sequence numbers”. Click Ok.   
Click on the Protocol header to sort the other headers from the TCP headers. 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 (and/or the packet trace [*tcp-ethereal-trace-1* in http://gaia.cs.umass.edu/ethereal-labs/ethereal-traces.zip](http://gaia.cs.umass.edu/ethereal-labs/traces/lab3-1-trace); see footnote 2) to study TCP behavior in the rest of this lab.

**Caution**: the TCP messages may not appear to be in chronological order (!). The TCP connection messages (SYN, SYN ACK, etc.) might appear after some of the data transfer messages, like the next image.  **The TCP PSH message has the POST command in it, so analyze that message, its ACK, and the next five pairs of “[TCP segment…” and ACK messages.**



3. TCP Analysis

Answer the following questions for the TCP segments:

1. **What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu? What is the IP address and port number used by gaia.cs.umass.edu to receive the file?**

150.250.221.33:60184

128.119.245.12:80

1. **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?**

Sequence Number:2540495029 (raw); 0 (relative)

Flags is a 12 bit number. The value 0x002 (bit1 is set) indicates that SYN is set.

1. **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 Number: 3383784835 (raw); 0 (relative)  
   ACK Number: 254045030 (raw); 1(relative). Determined from the next sequence number.

SYNACK is indicated by setting bits 1 and 4 of Flags. Flags will be 0x012.

1. **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 shut off the TCP filter from earlier, and dig into the packet content field at the bottom of the WireShark window, looking for a segment with a “POST” within its DATA field.**

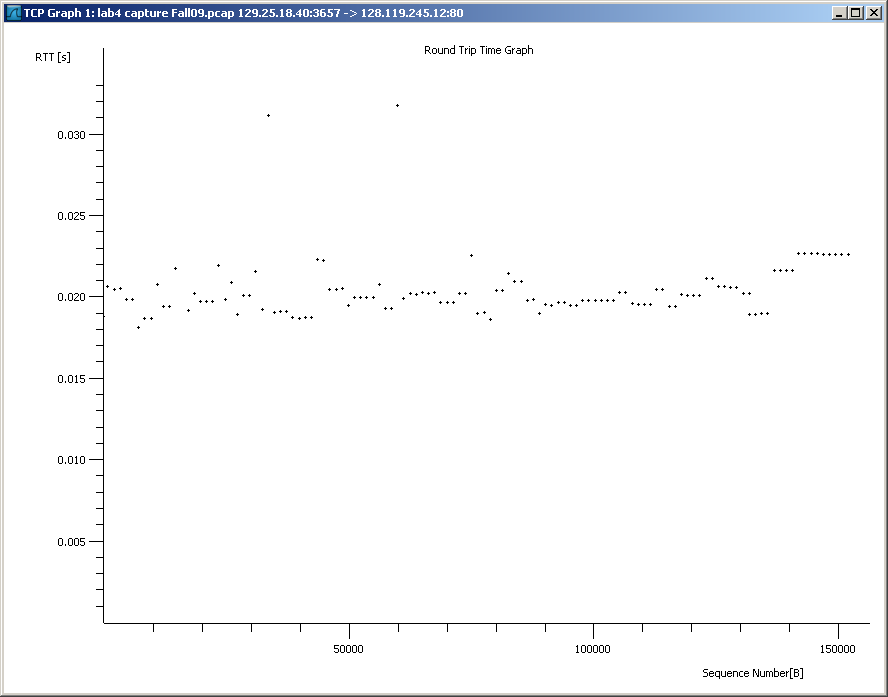
Sequence Number: 2540623479 (raw); 128450 (relative)

1. 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 and the first five Continuation messages after it)? **These are HTTP protocol messages, and you’re looking in the TCP portion of them. You can omit the first four digits of each ACK, since that probably won’t change during the transmission.**   
   *If you don’t have Continuation messages*, use the HTTP POST and the first five reassembled packets it cites – in the example in section 1, the POST command is packet #749, and the packets are 254, 334, 335, 336, and 350. The Info field for the packets might read “**[TCP segment of a reassembled PDU]**”.  
   At what time was each segment sent?   
   When was the ACK for each segment received? **The ACKs are TCP Protocol messages, with Info starting ‘http > *sourceportnumber* [ACK]*’,* where *sourceportnumber* is the port number of the client HTTP connection. Your Info field might differ in format, but it should say [ACK] somewhere! Match the Next Sequence Number from the HTTP message to the Info section of the TCP ACK, or if you’re using the TCP messages, look for mirror pairs of sequence and acknowledgement numbers (seq=A and ack=B in one message, seq=B and ack=A in the ack message).**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? Enter it in the table below.  
   What is the EstimatedRTT value (see page 237 in text or below after the table) after the receipt of each ACK? *Assume that the value of the EstimatedRTT is equal to the measured SampleRTT for the first segment*, and then is computed using the EstimatedRTT equation (see after summary table below) for all subsequent segments.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Packet | Send Time  (s) | Seq | ACK time (s) | SampleRTT  (s) | EstimatedRTT (s) |
| http POST  or TCP PSH | 27.945166024 | 2540623479 (raw)  128450 (relative) |  |  |  |
| TCP reassembled1 | 27.881997476 | 2540495030 (raw)  1  (relative) |  | 0.020331362 |  |
| TCP reassembled2 | 27.882124915 | 2540495697  (raw)  668  (relative) |  | 0.020331362 |  |
| TCP reassembled3 | 27.883098569 | 2540502567 (raw)  7538  (relative) |  | 0.020331362 |  |
| TCP reassembled4 | 27.900633410 | 2540508063 (raw)  13034  (relative) |  | 0.020331362 |  |
| TCP reassembled5 | 27.903077918 | 2540510811 (raw)  15782  (relative) |  | 0.020331362 |  |

EstimatedRTT = 0.875\*EstimatedRTT + 0.125\*SampleRTT

WireShark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select an HTTP Continuation packet in the “listing of captured packets” window that is being sent from the client to the gaia.cs.umass.edu server, OR **select one of the TCP packets that was reassembled by the HTTP POST command**. Then select: *Statistics->TCP Stream Graph->Round Trip Time Graph.* **Show this graph** for your upload. The RTT graph should look like this.



1. What is the data length of each of the first six TCP frames (from question 5)?[[3]](#footnote-4) Give the total frame lengths and the data contents length of each segment.
2. What is the minimum amount of available buffer space advertised at the received for the entire trace? (That’s the TCP Window size.) Does the lack of receiver buffer space ever throttle the sender? How could you tell if it did?
3. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?
4. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (not every one)?
5. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

**What to Turn In**

You should submit:

* Nothing from sections 1 and 2.
* From section 3, answers to questions 1 to 10. For question 5 you are filling out the table, **and** showing the graph.

***Be careful, there are multiple parts to most questions, so be sure you answer all of them!***

1. All references to the text in this lab are to *Computer Networking: A Top-down Approach Featuring the Internet,* 3rd edition. But the 4th and 5th editions have the same chapters. [↑](#footnote-ref-2)
2. Download the zip file <http://gaia.cs.umass.edu/ethereal-labs/ethereal-traces.zip> and extract the file tcp-ethereal-trace-1. The traces in this zip file were collected by WireShark running on one of the author’s computers, while performing the steps indicated in the WireShark lab. Once you have downloaded the trace, you can load it into WireShark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the tcp-ethereal-trace-1 trace file. [↑](#footnote-ref-3)
3. The TCP segments in the tcp-ethereal-trace-1 trace file are all less that 1460 bytes. This is because the computer on which the trace was gathered has an Ethernet card that limits the length of the maximum IP packet to 1500 bytes (40 bytes of TCP/IP header data and 1460 bytes of TCP payload). This 1500 byte value is the standard maximum length allowed by Ethernet. If your trace indicates a TCP length greater than 1500 bytes, and your computer is using an Ethernet connection, then WireShark is reporting the wrong TCP segment length; it will likely also show only one large TCP segment rather than multiple smaller segments. Your computer is indeed probably sending multiple smaller segments, as indicated by the ACKs it receives. This inconsistency in report edsegment lengths is due to the interaction between the Ethernet driver and the WireShark software. We recommend that if you have this inconsistency, that you perform this lab using the provided trace file. [↑](#footnote-ref-4)