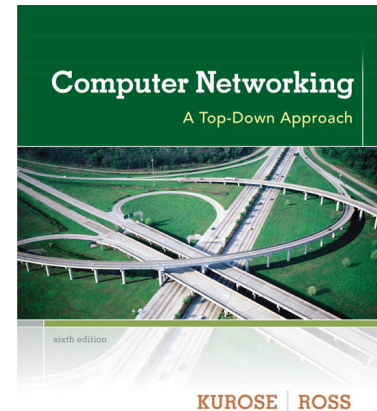


# Wireshark Lab: TCP v6.0

Supplement to *Computer Networking: A Top-Down Approach*, 6<sup>th</sup> ed., J.F. Kurose and K.W. Ross

“Tell me and I forget. Show me and I remember. Involve me and I understand.” Chinese proverb

© 2005-21012, J.F Kurose and K.W. Ross, All Rights Reserved



In this lab, we'll investigate the behavior of the celebrated TCP protocol 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 Carroll'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<sup>1</sup>.

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

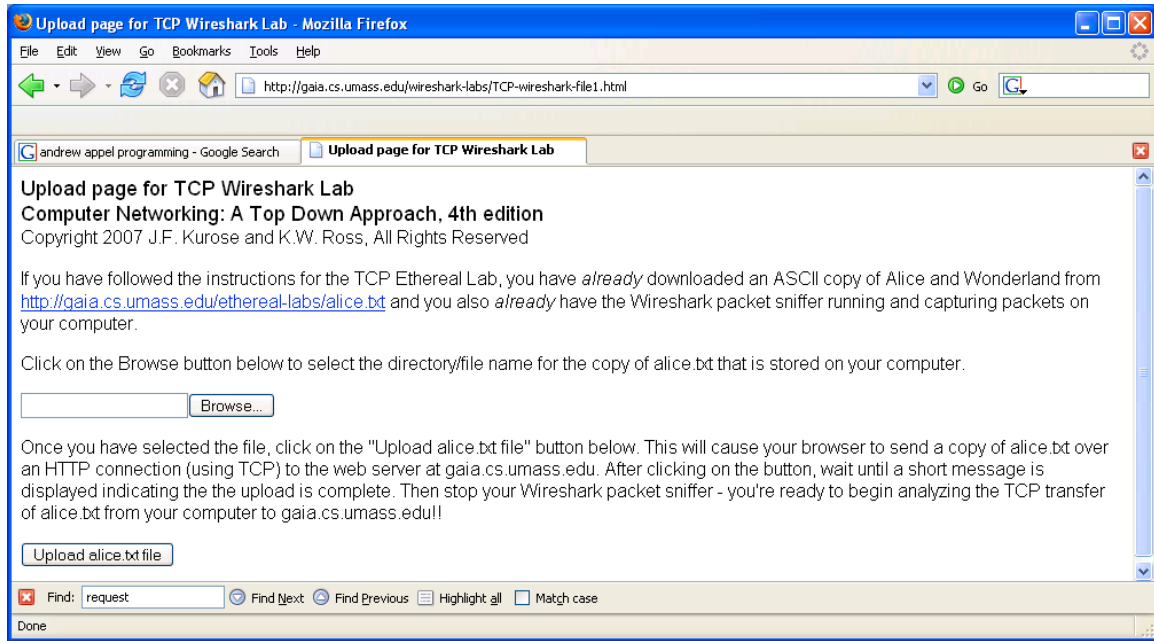
Before beginning our exploration of TCP, we'll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You'll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text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

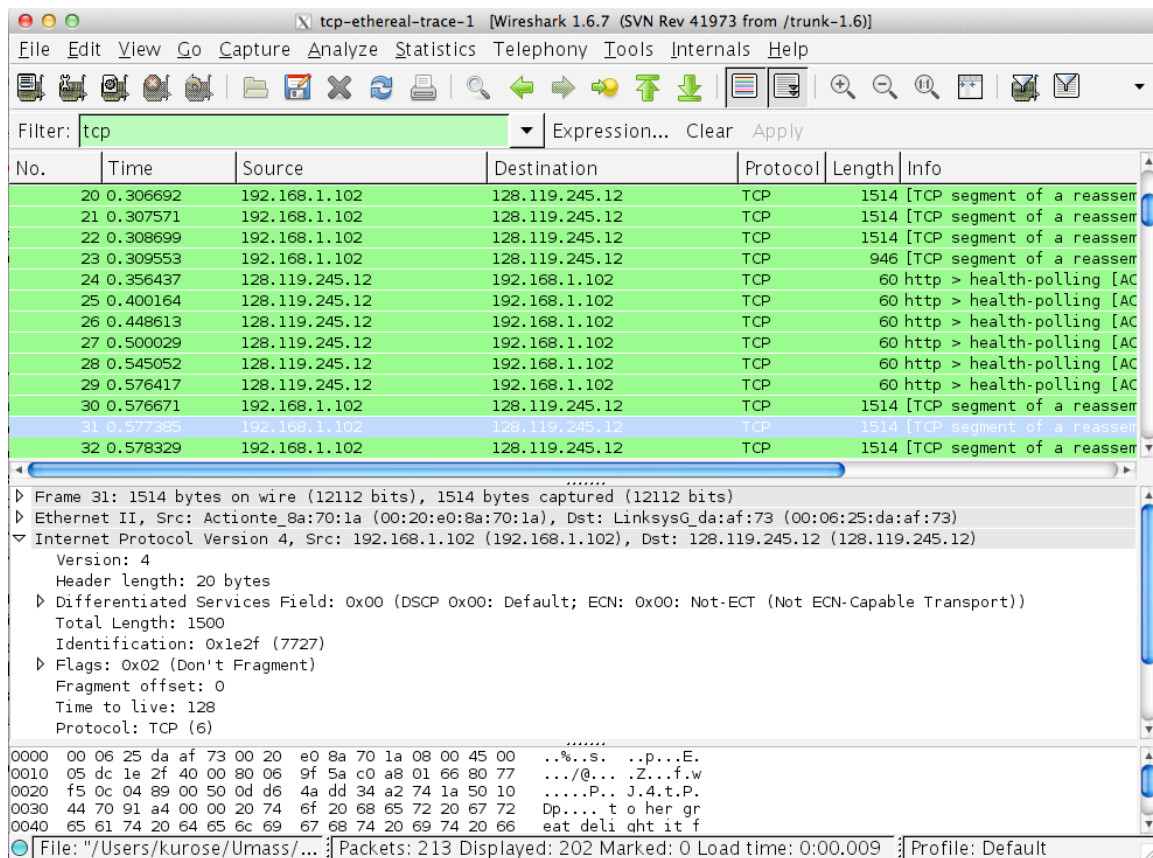
---

<sup>1</sup> References to figures and sections are for the 6<sup>th</sup> edition of our text, *Computer Networks, A Top-down Approach*, 6<sup>th</sup> ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2012.

- Start up your web browser. Go the <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>.
- You should see a screen that looks like:



- 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](http://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.



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<sup>2</sup>. 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" (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. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message. Depending on the version of

<sup>2</sup> Download the zip file <http://gaia.cs.umass.edu/wireshark-labs/wireshark-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.

Wireshark you are using, you 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. In more recent versions of Wireshark, you’ll see “[TCP segment of a reassembled PDU]” in the Info column of the Wireshark display to indicate that this TCP segment contained data that belonged to an upper layer protocol message (in our case here, HTTP). You should also see TCP ACK segments being returned from `gaia.cs.umass.edu` to your computer.

Answer the following questions, by opening the Wireshark captured packet file *tcp-ethereal-trace-1* in <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip> (that is download the trace and open that trace in Wireshark; see footnote 2). 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<sup>3</sup> 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.

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).
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?

If you have been able to create your own trace, answer the following question:

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

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 a Wireshark window that looks like:

---

<sup>3</sup> What do we mean by “annotate”? If you hand in a paper copy, please highlight where in the printout you’ve found the answer and add some text (preferably with a colored pen) noting what you found in what you’ve highlight. If you hand in an electronic copy, it would be great if you could also highlight and annotate.

tcp-ethereal-trace-1 [Wireshark 1.6.7 (SVN Rev 41973 from /trunk-1.6)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: tcp Expression... Clear Apply

| No. | Time     | Source         | Destination    | Protocol | Length | Info                         |
|-----|----------|----------------|----------------|----------|--------|------------------------------|
| 1   | 0.000000 | 192.168.1.102  | 128.119.245.12 | TCP      | 62     | health-polling > http [SYN]  |
| 2   | 0.023172 | 128.119.245.12 | 192.168.1.102  | TCP      | 62     | http > health-polling [SYN]  |
| 3   | 0.023265 | 192.168.1.102  | 128.119.245.12 | TCP      | 54     | health-polling > http [ACK]  |
| 4   | 0.026477 | 192.168.1.102  | 128.119.245.12 | TCP      | 619    | health-polling > http [POST] |
| 5   | 0.041737 | 192.168.1.102  | 128.119.245.12 | TCP      | 1514   | health-polling > http [POST] |
| 6   | 0.053937 | 128.119.245.12 | 192.168.1.102  | TCP      | 60     | http > health-polling [ACK]  |
| 7   | 0.054026 | 192.168.1.102  | 128.119.245.12 | TCP      | 1514   | health-polling > http [ACK]  |
| 8   | 0.054690 | 192.168.1.102  | 128.119.245.12 | TCP      | 1514   | health-polling > http [ACK]  |
| 9   | 0.077294 | 128.119.245.12 | 192.168.1.102  | TCP      | 60     | http > health-polling [ACK]  |
| 10  | 0.077405 | 192.168.1.102  | 128.119.245.12 | TCP      | 1514   | health-polling > http [ACK]  |
| 11  | 0.078157 | 192.168.1.102  | 128.119.245.12 | TCP      | 1514   | health-polling > http [ACK]  |
| 12  | 0.124085 | 128.119.245.12 | 192.168.1.102  | TCP      | 60     | http > health-polling [ACK]  |
| 13  | 0.124185 | 192.168.1.102  | 128.119.245.12 | TCP      | 1201   | health-polling > http [POST] |

Frame 1: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)

Ethernet II, Src: Actionte\_8a:70:1a (00:20:e0:8a:70:1a), Dst: LinksysG\_da:af:73 (00:06:25:da:af:73)

Destination: LinksysG\_da:af:73 (00:06:25:da:af:73)

Address: LinksysG\_da:af:73 (00:06:25:da:af:73)

... .. = IG bit: Individual address (unicast)

... .. = LG bit: Globally unique address (factory default)

Source: Actionte\_8a:70:1a (00:20:e0:8a:70:1a)

Address: Actionte\_8a:70:1a (00:20:e0:8a:70:1a)

... .. = IG bit: Individual address (unicast)

... .. = LG bit: Globally unique address (factory default)

```

0000  00 06 25 da af 73 00 20 e0 8a 70 1a 08 00 45 00  ..%.s.  ..p...E.
0010  00 30 1e 1d 40 00 80 06 a5 18 c0 a8 01 66 80 77  .0..@...  ....f.w
0020  f5 0c 04 89 00 50 0d d6 01 f4 00 00 00 00 70 02  ....P..  ....p.
0030  40 00 f6 e9 00 00 02 04 05 b4 01 01 04 02      @.....

```

File: "/Users/kurose/Umass/..." Packets: 213 Displayed: 202 Marked: 0 Load time: 0:00.011 Profile: Default

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/wireshark-labs/wireshark-traces.zip>; see earlier footnote) to study TCP behavior in the rest of this lab.

### 3. TCP Basics

Answer the following questions for the TCP segments:

- 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?
- 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?
- 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.
- 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.

*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?<sup>4</sup>
9. What is the minimum amount of available buffer space advertised at the receiver 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 (see Table 3.2 on page 247 in the text).
12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

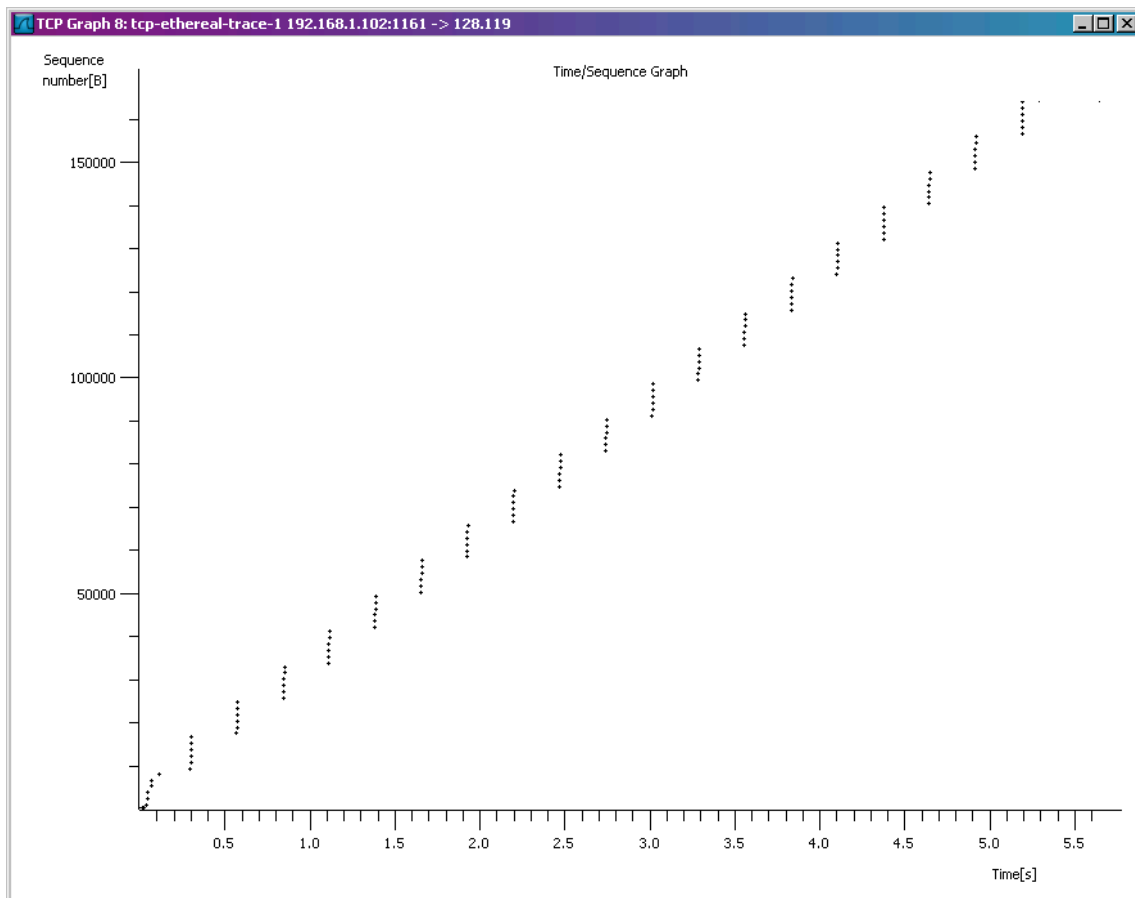
---

<sup>4</sup> The TCP segments in the tcp-ethereal-trace-1 trace file are all less than 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 reported segment 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.

## 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.

- 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 looks similar to the following plot, which was created from the captured packets in the packet trace *tcp-ethereal-trace-1* in <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip> (see earlier footnote ):



Here, 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.

Answer the following questions for the TCP segments the packet trace *tcp-ethereal-trace-1* in <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip>

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 slowstart 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