ELEC3506/9506 Communication Networks – Lab 03

Instruction

This lab instruction comprises two phases, where each contains multiple questions that students need to answer them during the lab session to comprehend the main goal of the phase. Although, it is not mandatory to bring the answers into your Final Lab Report. Indeed, the Final Lab Report you need to submit to the Canvas should be within maximum of 7 pages, stating your overall learning experience from the lab experiments and the main results you have obtained. In the first phase, students investigate the behavior of the transmission control protocol (TCP). To this end, students need to analyze a trace of TCP segments by transferring a 150KB file to a remote server. Then, students study the role of sequence number as well as acknowledge number in providing reliable data transfer; moreover, students get familiar with the congestion control algorithm, namely slow start and congestion avoidance; finally, students investigate the performance of the TCP connection between their systems and the server by analyzing the throughput and round-trip time diagram. In the second phase, students explore several aspects of the hypertext transfer protocol (HTTP), which are the basic GET/response interaction, HTTP message formats, retrieving large HTML files, retrieving HTML files with embedded objects, and HTTP authentication and security.

A template has been provided for each Lab Report that you should follow it to prepare the Final Lab Report. You can find the template in the Canvas. Before starting the lab, students should follow the instructions provided for the lab session. The instructions are summarized as follows:

- You need to follow up the lab instruction, provided in the Canvas, regarding forming groups and submitting the Final Lab Reports.
- Complete the lab according to the instructions in the following pages.
- You should observe lab rules and behave properly in the lab.
- Relevant materials: TCP - pages 715 – 735; Three-way shaking - pages 723 – 724;
  Push data (PSH) - page 725 – 726;
  Congestion Control - pages 769 – 773; HTTP - pages 861 - 868.
- Moreover, there are supplementary instructions for TCP as follows:
  - An easier way to view TCP flow in Wireshark: *Statistics -> Flow Graph -> Choose flow type: TCP flow -> OK*
  - Filter the specific TCP stream you want: Right click one message -> Follow TCP stream
  - Section 1 TCP Q4-15: Using the zip file is strongly recommended.
  - Section 1 TCP Q3 & Q14 are optional.
  - A Helpful formula for Q7:
    $EstimatedRTT = 0.875 \times Former\ EstimatedRTT + 0.125 \times SampleRTT$, where the *SampleRTT* is the real RTT you can find in Wireshark.
  - A good article to understand TCP sequence and acknowledgment numbers: http://packetlife.net/blog/2010/jun/7/understanding-tcp-sequence-

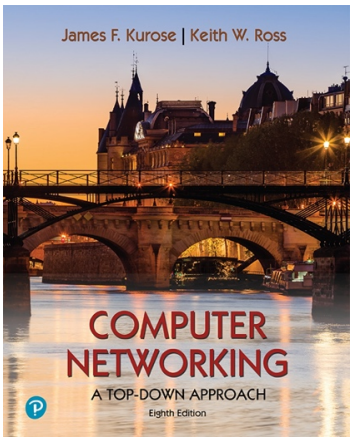
acknowledgment-numbers/?pdf

Notation: Students can find the corresponding concepts in the main reference provided for this course:

Phase 1: TCP

Supplement to *Computer Networking: A Top-Down Approach, 8th 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



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 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].

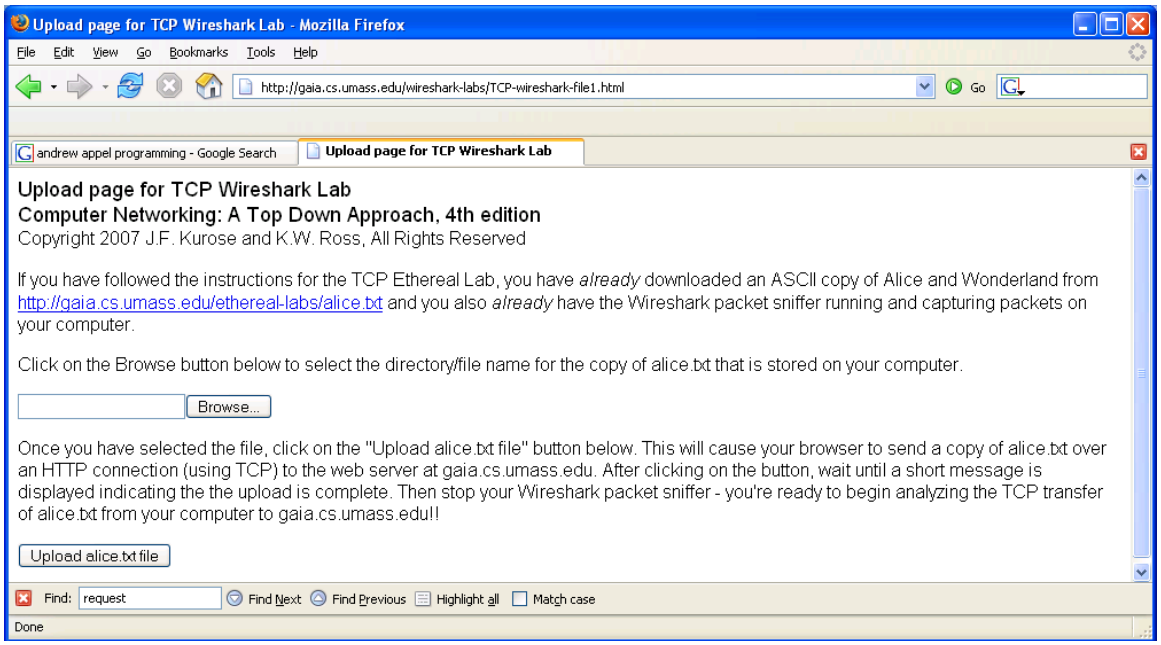
1. Capturing a bulk TCP transfer from your computer to a remote server

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.

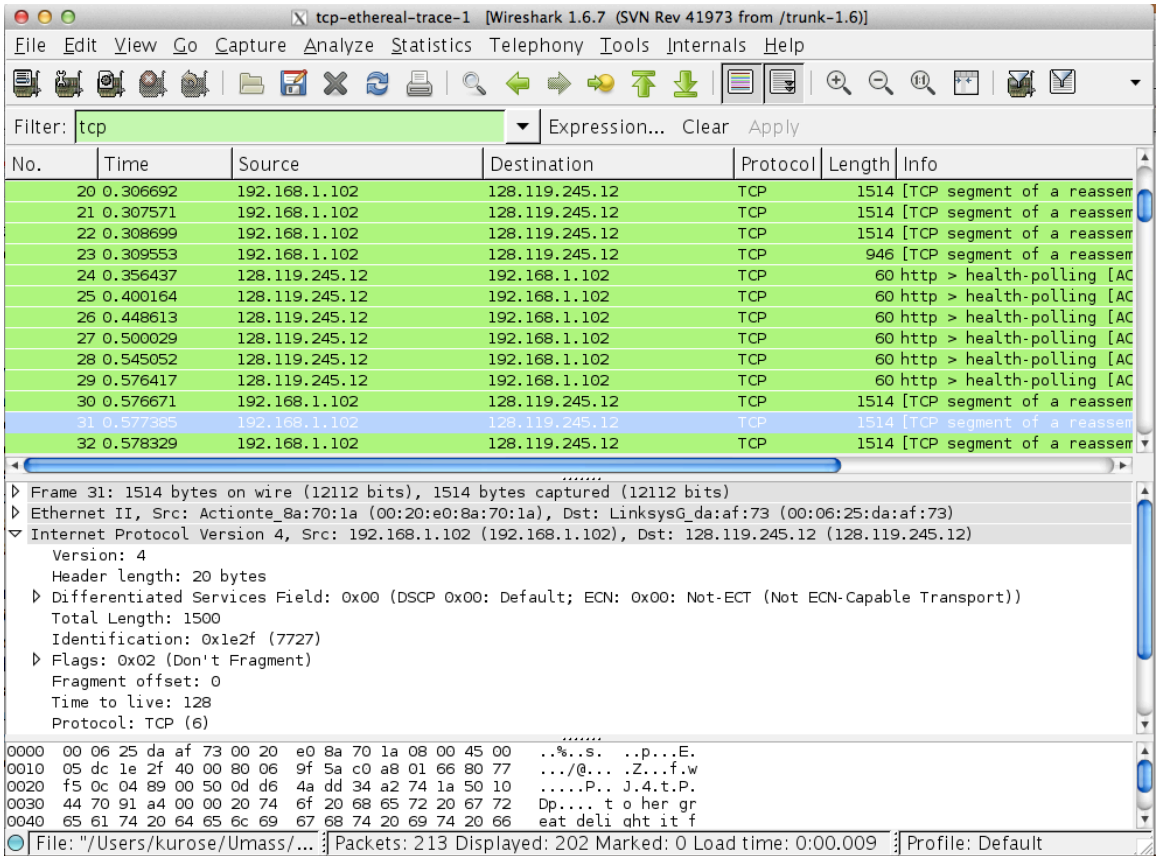
Do the following:

[1] References to figures and sections are for the 8th edition of our text, *Computer Networks, A Top-down Approach, 8th ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2020.*

- 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 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[2]. 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

[2] 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[3] 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.

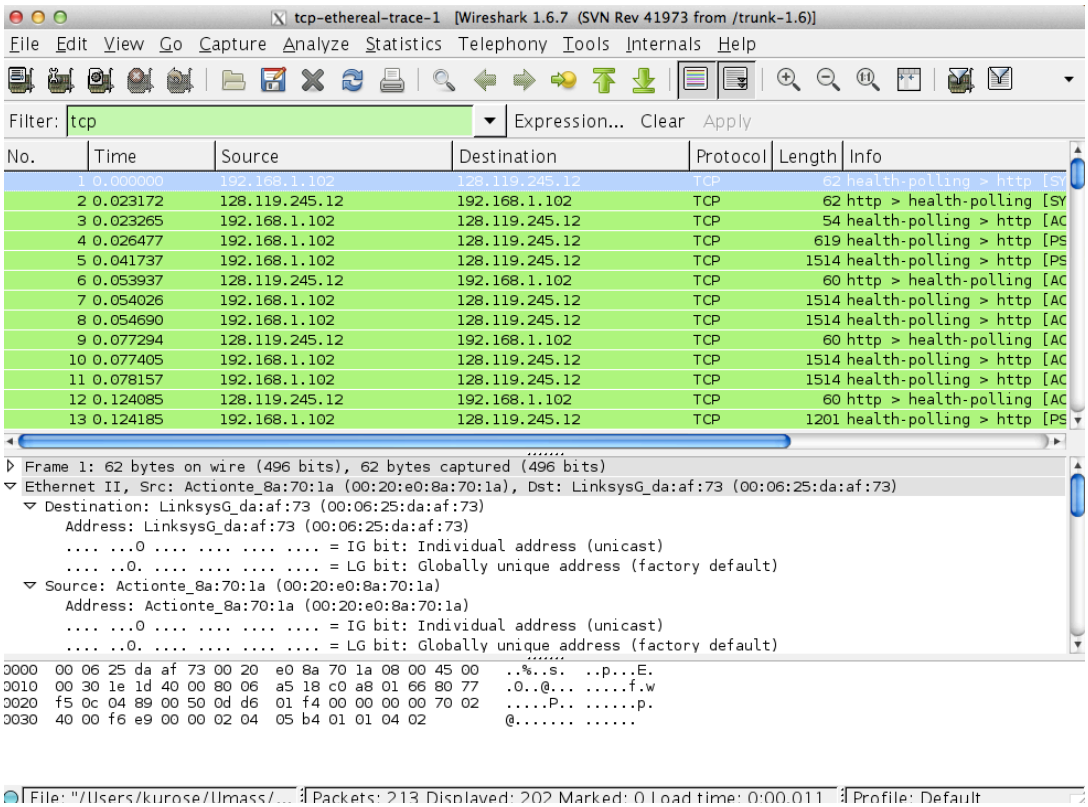
1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it's probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the "details 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:

[3] 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.



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:

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?
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 (see Section 3.5.3, page 242 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 242 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.*

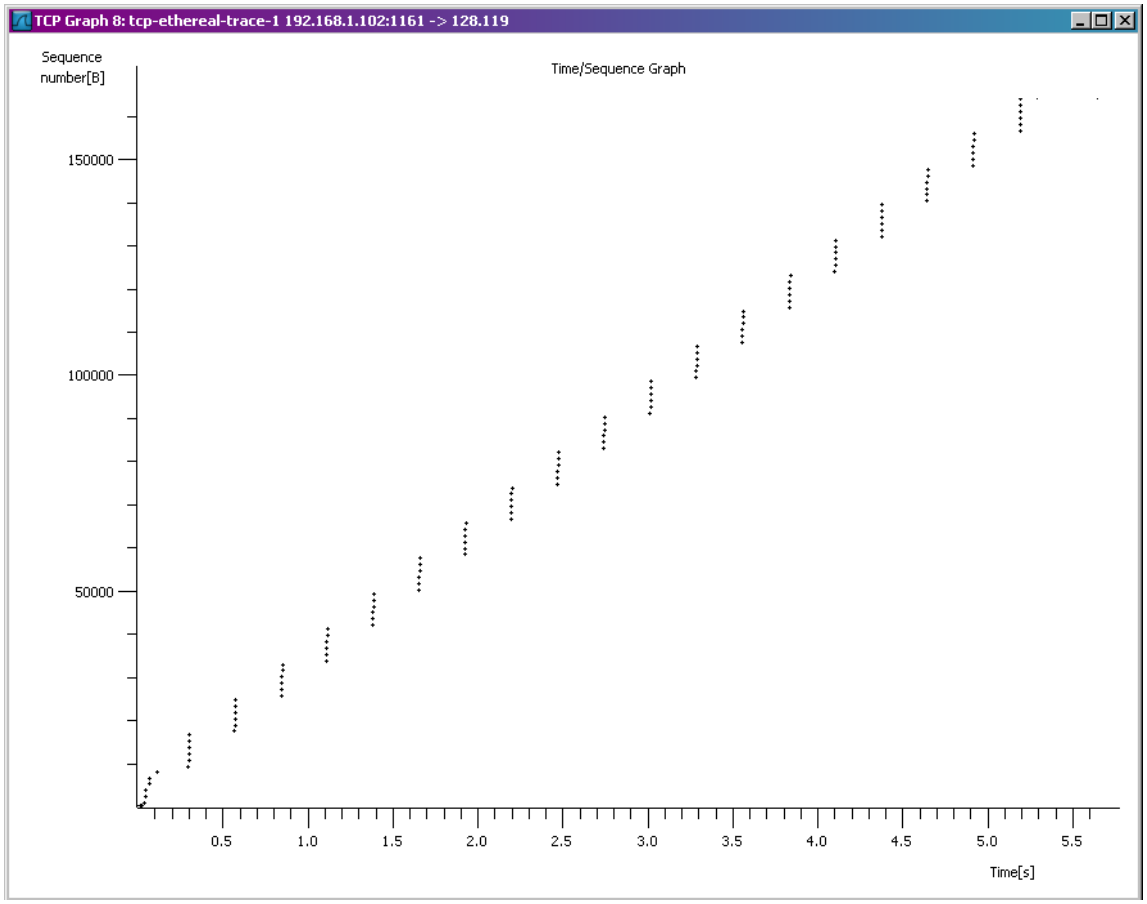
8. What is the length of each of the first six TCP segments?[4]
9. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?
10. Are there any retransmitted segments in the trace file? What did 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 (see Table 3.2 on page 250 in the text).
12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

[4] 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 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.

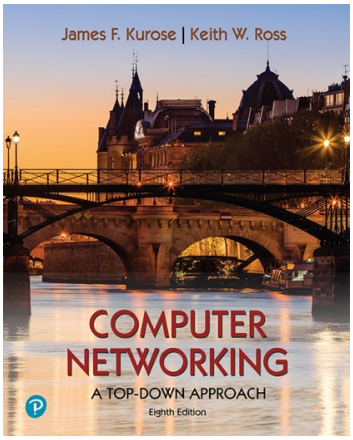
Answer the following questions for the TCP segments the packet trace *tcp-ethereal-trace-1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip

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

Phase 2: HTTP

Supplement to *Computer Networking: A Top-Down Approach, 8th 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



Having gotten our feet wet with the Wireshark packet sniffer in the introductory lab, we're now ready to use Wireshark to investigate protocols in operation. In this lab, we'll explore several aspects of the HTTP protocol: the basic GET/response interaction, HTTP message formats, retrieving large HTML files, retrieving HTML files with embedded objects, and HTTP authentication and security. Before beginning these labs, you might want to review Section 2.2 of the text.[1]

1. The Basic HTTP GET/response interaction

Let's begin our exploration of HTTP by downloading a very simple HTML file - one that is very short, and contains no embedded objects. Do the following:

1. Start up your web browser.
2. Start up the Wireshark packet sniffer, as described in the Introductory lab (but don't yet begin packet capture). Enter "http" (just the letters, not the quotation marks) in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window. (We're only interested in the HTTP protocol here, and don't want to see the clutter of all captured packets).
3. Wait a bit more than one minute (we'll see why shortly), and then begin Wireshark packet capture.
4. Enter the following to your browser
   http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html
   Your browser should display the very simple, one-line HTML file.
5. Stop Wireshark packet capture.

[1] References to figures and sections are for the 8th edition of our text, *Computer Networks, A Top-down Approach, 8h ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2020.*

Your Wireshark window should look similar to the window shown in Figure 1. If you are unable to run Wireshark on a live network connection, you can download a packet trace that was created when the steps above were followed.[2]

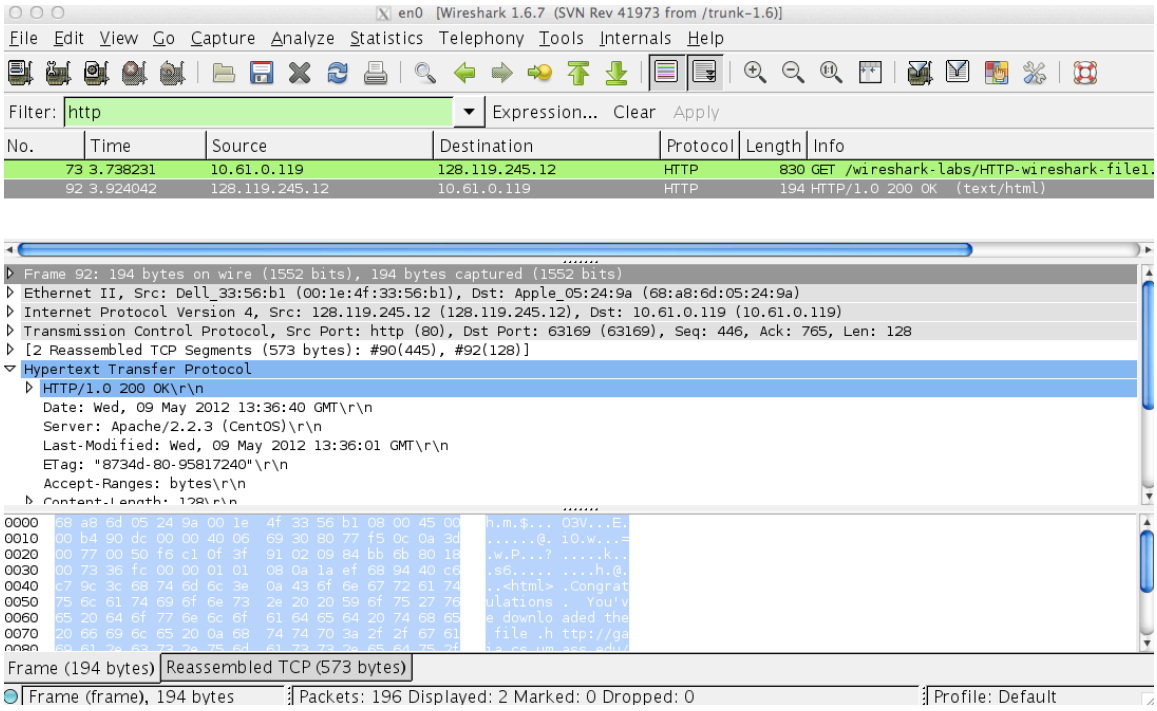


Figure 1: Wireshark Display after http://gaia.cs.umass.edu/wireshark-labs/ HTTP-wireshark-file1.html has been retrieved by your browser

The example in Figure 1 shows in the packet-listing window that two HTTP messages were captured: the GET message (from your browser to the gaia.cs.umass.edu web server) and the response message from the server to your browser. The packet-contents window shows details of the selected message (in this case the HTTP OK message, which is highlighted in the packet-listing window). Recall that since the HTTP message was carried inside a TCP segment, which was carried inside an IP datagram, which was carried within an Ethernet frame, Wireshark displays the Frame, Ethernet, IP, and TCP packet information as well. We want to minimize the amount of non-HTTP data displayed (we're interested in HTTP here, and will be investigating these other protocols is later labs), so make sure the boxes at the far left of the Frame, Ethernet, IP and TCP information have a plus sign or a right-pointing triangle (which means there is hidden, undisplayed information), and the HTTP line has a minus sign or a down-pointing triangle (which means that all information about the HTTP message is displayed).

---

[2] Download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the file http-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 http-ethereal-trace-1 trace file. The resulting display should look similar to Figure 1. (The Wireshark user interface displays just a bit differently on different operating systems, and in different versions of Wireshark).

(*Note:* You should ignore any HTTP GET and response for favicon.ico. If you see a reference to this file, it is your browser automatically asking the server if it (the server) has a small icon file that should be displayed next to the displayed URL in your browser. We'll ignore references to this pesky file in this lab.).

By looking at the information in the HTTP GET and response messages, answer the following questions. When answering the following questions, you should print out the GET and response messages (see the introductory Wireshark lab for an explanation of how to do this) and indicate where in the message you've found the information that answers the following questions. When you hand in your assignment, annotate the output so that it's clear where in the output you're getting the information for your answer (e.g., for our classes, we ask that students markup paper copies with a pen, or annotate electronic copies with text in a colored font).

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?
2. What languages (if any) does your browser indicate that it can accept to the server?
3. What is the IP address of your computer? Of the gaia.cs.umass.edu server?
4. What is the status code returned from the server to your browser?
5. When was the HTML file that you are retrieving last modified at the server?
6. How many bytes of content are being returned to your browser?
7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one.

In your answer to question 5 above, you might have been surprised to find that the document you just retrieved was last modified within a minute before you downloaded the document. That's because (for this particular file), the gaia.cs.umass.edu server is setting the file's last-modified time to be the current time, and is doing so once per minute. Thus, if you wait a minute between accesses, the file will appear to have been recently modified, and hence your browser will download a "new" copy of the document.

2. The HTTP CONDITIONAL GET/response interaction

Recall from Section 2.2.5 of the text, that most web browsers perform object caching and thus perform a conditional GET when retrieving an HTTP object. Before performing the steps below, make sure your browser's cache is empty. (To do this under Firefox, select *Tools->Clear Recent History* and check the Cache box, or for Internet Explorer, select *Tools->Internet Options->Delete File;* these actions will remove cached files from your browser's cache.) Now do the following:

- Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser
  http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file2.html
  Your browser should display a very simple five-line HTML file.

- Quickly enter the same URL into your browser again (or simply select the refresh button on your browser)
- Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.
- (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-2 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author's computers.)

Answer the following questions:

8. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE" line in the HTTP GET?
9. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?
10. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE:" line in the HTTP GET? If so, what information follows the "IF-MODIFIED-SINCE:" header?
11. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

3. Retrieving Long Documents

In our examples thus far, the documents retrieved have been simple and short HTML files. Let's next see what happens when we download a long HTML file. Do the following:

- Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser
  http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html
  Your browser should display the rather lengthy US Bill of Rights.
- Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed.
- (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-3 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author's computers.)

In the packet-listing window, you should see your HTTP GET message, followed by a multiple-packet TCP response to your HTTP GET request. This multiple-packet response deserves a bit of explanation. Recall from Section 2.2 (see Figure 2.9 in the text) that the HTTP response message consists of a status line, followed by header lines, followed by a blank line, followed by the entity body. In the case of our HTTP GET, the

entity body in the response is the *entire* requested HTML file. In our case here, the HTML file is rather long, and at 4500 bytes is too large to fit in one TCP packet. The single HTTP response message is thus broken into several pieces by TCP, with each piece being contained within a separate TCP segment (see Figure 1.24 in the text). In recent versions of Wireshark, Wireshark indicates each TCP segment as a separate packet, and the fact that the single HTTP response was fragmented across multiple TCP packets is indicated by the "TCP segment of a reassembled PDU" in the Info column of the Wireshark display. Earlier versions of Wireshark used the "Continuation" phrase to indicated that the entire content of an HTTP message was broken across multiple TCP segments.. We stress here that there is no "Continuation" message in HTTP!

Answer the following questions:

12. How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill or Rights?
13. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?
14. What is the status code and phrase in the response?
15. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?

4. HTML Documents with Embedded Objects

Now that we've seen how Wireshark displays the captured packet traffic for large HTML files, we can look at what happens when your browser downloads a file with embedded objects, i.e., a file that includes other objects (in the example below, image files) that are stored on another server(s).

Do the following:

- Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser
  http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file4.html
  Your browser should display a short HTML file with two images. These two images are referenced in the base HTML file. That is, the images themselves are not contained in the HTML; instead the URLs for the images are contained in the downloaded HTML file. As discussed in the textbook, your browser will have to retrieve these logos from the indicated web sites. Our publisher's logo is retrieved from the gaia.cs.umass.edu web site. The image of the cover for our 5th edition (one of our favorite covers) is stored at the caite.cs.umass.edu server. (These are two different web servers inside cs.umass.edu).
- Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed.

- (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-4 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author's computers.)

Answer the following questions:

16. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?
17. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.

5 HTTP Authentication

Finally, let's try visiting a web site that is password-protected and examine the sequence of HTTP message exchanged for such a site. The URL http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html is password protected. The username is "wireshark-students" (without the quotes), and the password is "network" (again, without the quotes). So let's access this "secure" password-protected site. Do the following:

- Make sure your browser's cache is cleared, as discussed above, and close down your browser. Then, start up your browser
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser
  http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html
  Type the requested user name and password into the pop up box.
- Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.
- (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-5 packet trace to answer the questions below; see footnote 2. This trace file was gathered while performing the steps above on one of the author's computers.)

Now let's examine the Wireshark output. You might want to first read up on HTTP authentication by reviewing the easy-to-read material on "HTTP Access Authentication Framework" at http://frontier.userland.com/stories/storyReader$2159

Answer the following questions:

18. What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?
19. When your browser's sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?

The username (wireshark-students) and password (network) that you entered are encoded in the string of characters (d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcms=) following

the "`Authorization: Basic`" header in the client's HTTP GET message. While it may appear that your username and password are encrypted, they are simply encoded in a format known as Base64 format. The username and password are *not* encrypted! To see this, go to http://www.motobit.com/util/base64-decoder-encoder.asp and enter the base64-encoded string d2lyZXNoYXJrLXN0dWRlbnRz and decode. *Voila!* You have translated from Base64 encoding to ASCII encoding, and thus should see your username! To view the password, enter the remainder of the string Om5ldHdvcms= and press decode. Since anyone can download a tool like Wireshark and sniff packets (not just their own) passing by their network adaptor, and anyone can translate from Base64 to ASCII (you just did it!), it should be clear to you that simple passwords on WWW sites are not secure unless additional measures are taken.

Fear not! As we will see in Chapter 8, there are ways to make WWW access more secure. However, we'll clearly need something that goes beyond the basic HTTP authentication framework!