CSC3002F Networking Assignment 2

Packet Sniffing with Wireshark

1 Introduction

This assignment focuses on the 'packets' being sent around on a computer network. You will discover what is sent around on the transmission medium; what those packets 'look like'.

You may have heard of IP packets, but what exactly is 'inside' one? Or sending an HTTP request to open a web page, how is that put across the transmission medium? The Wireshark tool, a packet 'sniffer', will be used to obtain answers to such questions.

The assignment is based on two 'Wireshark labs' developed by Kim Jurose and Ken Ross as an adjunct to their book "Computer Networking: A Top Down Approach". Part one is the TCP lab and part two the IP lab. (An introduction to Wireshark available on the Vula page for this assignment.)

The assignment is partly automatically marked and partly manually marked. Automatically marked questions/parts are indicated with the suffix '[AM]' while manually marked parts are indicated with the suffix [MM]

You will submit (in a ZIP file):

1. A trace file called '`tcptrace.pcap`' containing your trace for the first half of the assignment (questions 1-9) ;
2. A trace file called '`iptrace.pcap`' containing your trace for the second half of the assignment (questions 10-18) ;
3. A structured text file called '`automark.txt`' containing answers to automatically marked questions. (File format below; sample on Vula assignment page.)
4. A document (Word/Open Office/LaTeX/…) containing answers (optionally including annotated screenshots) to manually marked questions.
   You should name it '`manualmark.`*`xxx`*' where the filename extension depends on the document format.

NOTE

- You can submit **only TEN times** and the automatic marker will **only indicate that an answer is right or wrong**, so be careful and think and check twice, or even thrice, before submitting!
- Your traces must be saved as '`pcap`' files, NOT '`pcapng`' files.

Continued

The format for `automark.txt` is as follows:

```
# q1
<Trace frame number where you found the answer.>
<Source IP>
<Source port>

# q2
<Trace frame number where you found the answer.>
<Destination IP>
<Destination port>

#q3
<Trace frame number where you found the answer.>
<Sequence number>

#q4
<Trace frame number.>
<Sequence number>
<Acknowledgement number>

#q5
<Trace frame number>
<Sequence number>

#q6
<Trace frame number for 1st segment>
<Time first segment sent>
<Time ACK received>
<RTT>
<Second segment RTT>
<Estimated RTT>

#q7
<Length of first>
<Length of second>
<Length of third>
<length of fourth>

#q8
<Minimum advertised buffer space>

#q10
<Trace frame number>
<IP Address>

# q11
<TTL value>

# q12
<Protocol value>

# q13
<header length>
<body length>

# q14
<Fragmented>
```

Continued

Sample file:

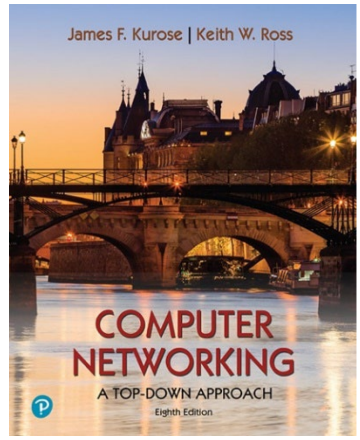
```
# Question One
271
197.239.129.141
59536

# Question two
271
128.119.245.12
80

# Question three
213
2841915405

# Question four
220
2223153991
2841915406

# Question five
222
2841915406

# Question six
222
7.879
8.098
0.249
0.249
0.249

# Question seven
634
1400
1400
2780

# Question eight
15872

# Question 10
9
10.0.2.15

# Question 11
1

# Question 12
17
# Question 13
20
36

# Question 14
no
```

Continued

The file may contain blank line and single line comments – a comment is preceded by a hash, '#', symbol. Real numbers should be recorded to three decimal places.

Continued

Wireshark Lab:
TCP v8.1[1]



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 text file 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[2].There is also a separate Wireshark introduction pdf on Vula that you may want to read through. It gives a little overview of the tool. (There's also the manual online.)

Since the Wireshark authors keep developing the software, and there may be slight differences across OSs, and it may be that the screenshots below aren't exactly the same as what you'll see.

---

[1] With modifications for UCT Networks assignment 2.

[2] 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.* Our website for this book is http://gaia.cs.umass.edu/kurose_ross You'll find lots of interesting open material there.

Continued

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 a project Gutenburg ASCII text), 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 Vula and retrieve your assigned Gutenburg text. Store this as a .txt 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 Figure 1.

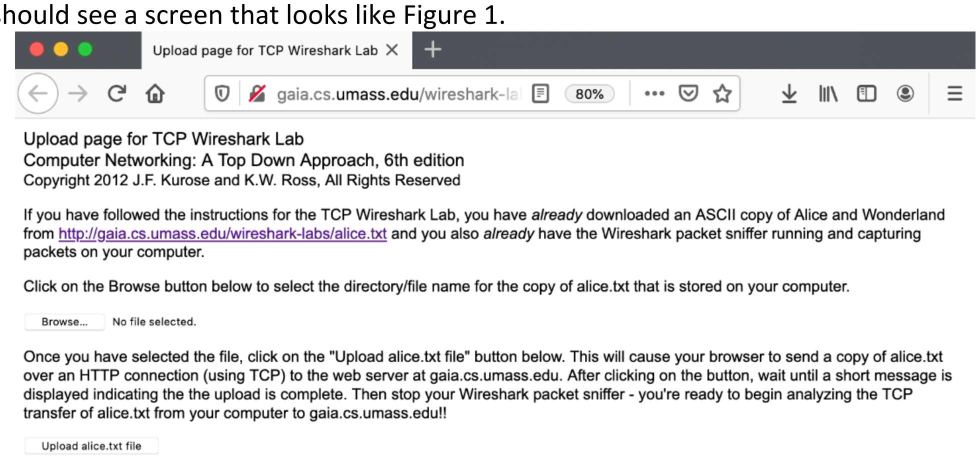


Figure 1: Page to upload the alice.txt file from your computer to gaia.cs.umass.edu

- Use the *Browse* button in this form to the file on your computer that you just created. Don't press the "*Upload alice.txt file*" button yet.[3]
- Now start up Wireshark and begin packet capture (see the earlier Wireshark labs if you need a refresher on how to do this).
- 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 in Figure 2.

---

[3] The button presumes that you're uploading the Gutenburg text for Alice in Wonderland, however, we have modified the lab requiring you to upload a personally assigned text.

Continued

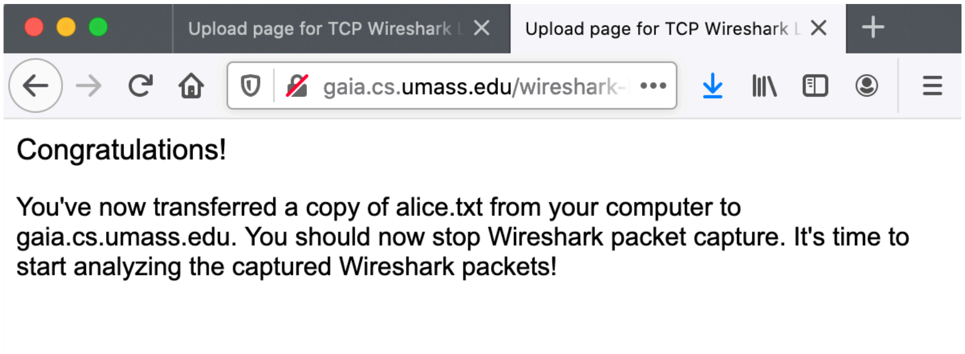


Figure 2: Success! You've uploaded a file to gaia.cs.umass.edu and have hopefully captured a Wireshark packet trace while doing so.

Note: You can download a packet trace that was captured while following the steps above on one of the author's computers [4]. In addition, to your own trace, you may find it helpful when you explore the questions below.

[4] You can download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces-8.1.zip and extract the trace file *tcp-wireshark-trace1-1*. This trace file can be used to answer this Wireshark lab without actually capturing packets on your own. This trace was made using Wireshark running on one of the author's computers, while performing the steps indicated in this Wireshark lab. Once you've downloaded a trace file, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the trace file name.

Continued

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.

Let's start by looking at the HTTP POST message that uploaded the alice.txt file to gaia.cs.umass.edu from your computer. Find that file in your Wireshark trace, and expand the HTTP message so we can take a look at the HTTP POST message more carefully. Your Wireshark screen should look something like Figure 3.

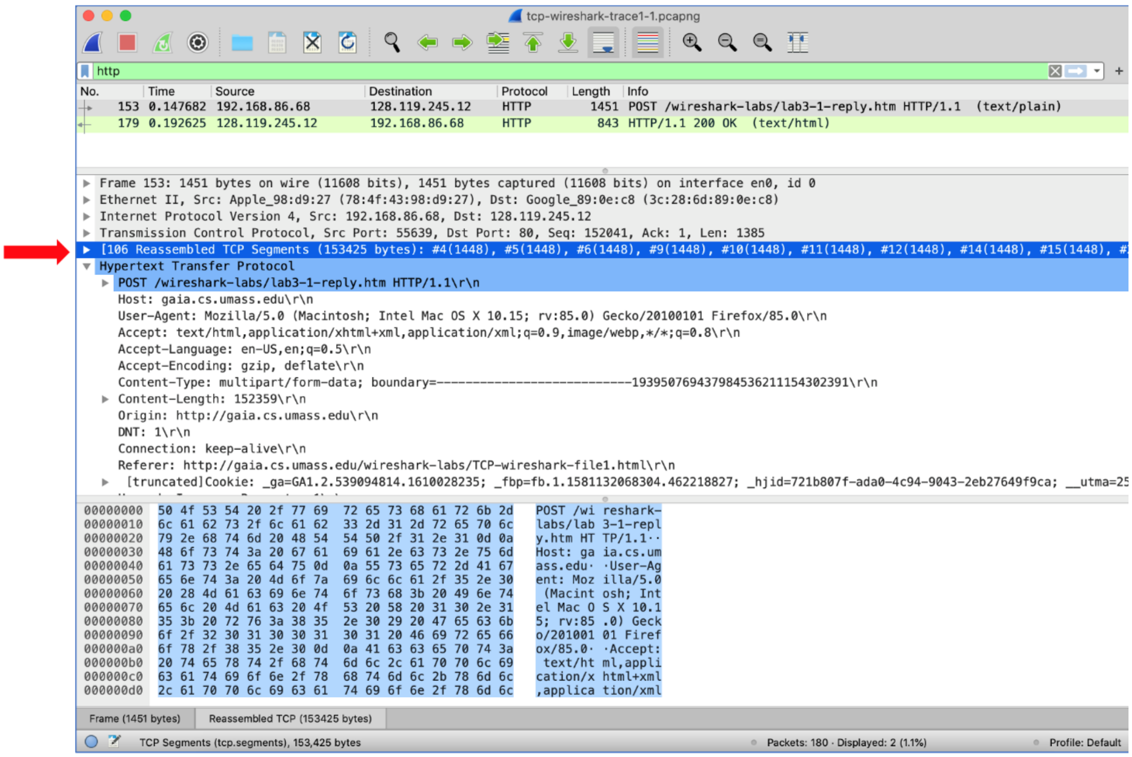


Figure 3: expanding the HTTP POST message that uploaded alice.txt from your computer to gaia.cs.umass.edu

There are a few things to note here:

- The body of your application-layer HTTP POST message contains the contents of the file alice.txt, which is a large file of more than 152K bytes. OK – it's not *that* large, but it's going to be too large for this one HTTP POST message to be contained in just one TCP segment!
- In fact, as shown in the Wireshark window in Figure 3 we see that the HTTP POST message was spread across 106 TCP segments. This is shown where the red arrow is placed in Figure 3 [Aside: Wireshark doesn't have a red arrow like that; we added it to the figure to be helpful ☺]. If you look even more carefully there, you can see that Wireshark is being really helpful to you as well, telling you that the first TCP segment containing the beginning of the POST message is packet #4 in the particular trace for the example in Figure 3, which is the trace *tcp-wireshark-trace1-1* noted in footnote 2. The second TCP segment containing the POST message in packet #5 in the trace, and so on.

Continued

Let's now "get our hands dirty" by looking at some TCP segments.

- 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. Your Wireshark display should look something like Figure 4. In Figure 4, we've noted the TCP segment that has its SYN bit set – this is the first TCP message in the three-way handshake that sets up the TCP connection to gaia.cs.umass.edu that will eventually carry the HTTP POST message and the text file. We've also noted the SYNACK segment (the second step in TCP three-way handshake), as well as the TCP segment (packet #4, as discussed above) that carries the POST message and the beginning of the alice.txt file. Of course, the packet numbers will be different in your own trace file, but you should see similar behavior to that shown in Figures 3 and 4.

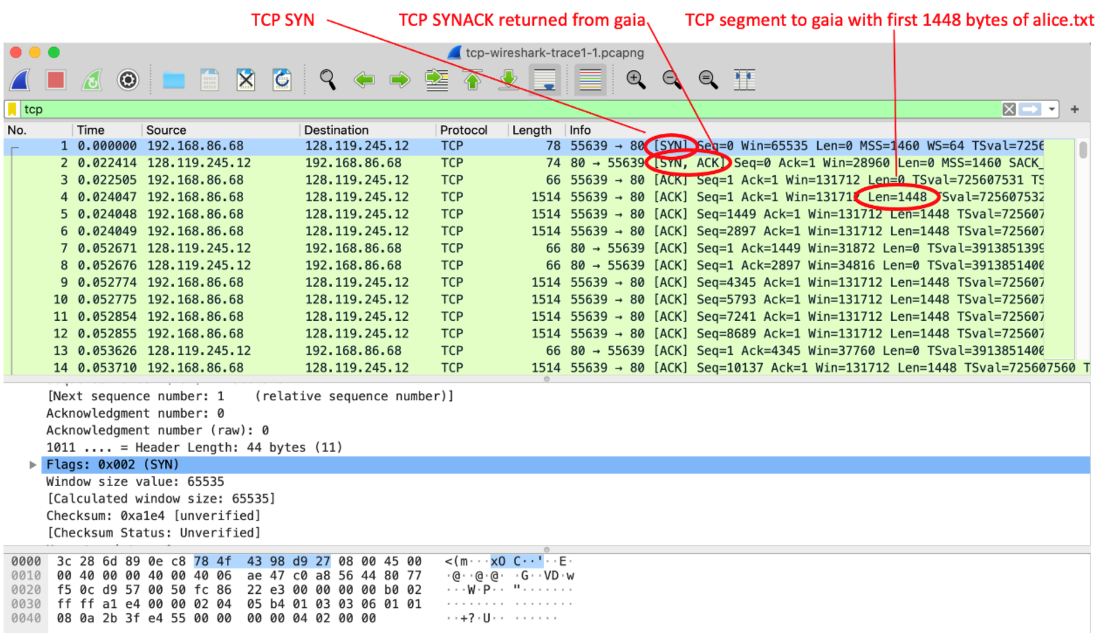


Figure 4: TCP segments involved in sending the HTTP POST message (including the file alice.txt) to gaia.cs.umass.edu

Answer the following questions. Parts that are labelled '[AM]' will be automatically marked and parts that are labelled '[MM]' will be manually marked, while parts with no suffix will not be marked. Where applicable, for manually marked questions, include a screenshot of the packet(s) within the trace that you used to answer a question, or annotate the packet text to explain your answers[5].

[5] For the author's class, when answering the following questions with hand-in assignments, students sometimes need to print out specific packets (see the introductory Wireshark lab for an explanation of how to do this) and indicate where in the packet they've found the information that answers a question. They do this by marking paper copies with a pen or annotating electronic copies with text in a colored font. There

Continued

1. What is the IP address and TCP port number used by the client computer (source) that is transferring the alice.txt file to gaia.cs.umass.edu? [AM]
   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? [AM]

Since this lab is about TCP rather than HTTP, now 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, as in Figure 4 above. (Select Analyze > Enabled Protocols, uncheck the HTTP box and select OK.) This is what we're looking for–a series of TCP segments sent between your computer and gaia.cs.umass.edu!

3. TCP Basics

Answer the following questions for the TCP segments:

3. 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? [AM]
   Note: this is the "raw" sequence number carried in the TCP segment itself; it is NOT the packet # in the "No." column in the Wireshark window. Remember there is no such thing as a "packet number" in TCP or UDP; as you know, there are sequence numbers in TCP and that's what we're after here. Also note that this is not the relative sequence number with respect to the starting sequence number of this TCP session.).
   What is it in this TCP segment that identifies the segment as a SYN segment? [MM]
4. What is the *sequence number* of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? [AM]
   What is it in the segment that identifies the segment as a SYNACK segment? [MM]
   What is the value of the Acknowledgement field in the SYNACK segment? [AM]
   How did gaia.cs.umass.edu determine that value? [MM]
5. What is the sequence number of the TCP segment containing the header of the HTTP POST command? [AM]
   Note that in order to find the POST message header, you'll need to dig into the packet content field at the bottom of the Wireshark window, *looking for a segment with the ASCII text "POST" within its DATA field*[6,7].
6. Consider the TCP segment containing the HTTP "POST" as the first segment in the data transfer part of the TCP connection.

are also learning management system (LMS) modules for teachers that allow students to answer these questions online and have answers auto-graded for these Wireshark labs at http://gaia.cs.umass.edu/kurose_ross/lms.htm

[6] *Hint:* this TCP segment is sent by the client soon (but not always immediately) after the SYNACK segment is received from the server.

[7] Note that if you filter to only show "http" messages, you'll see that the TCP segment that Wireshark associates with the HTTP POST message is the *last* TCP segment in the connection (which contains the text at the *end* of alice.txt: "THE END") and *not* the first data-carrying segment in the connection. Students (and teachers!) often find this unexpected and/or confusing.

Continued

- At what time was the first segment (the one containing the HTTP POST) in the data transfer part of the TCP connection sent? [AM]
- At what time was the ACK for this first data-containing segment received? [AM]
- What is the RTT for this first data-containing segment? [AM]
- What is the RTT value for the second data-carrying TCP segment and its ACK. [AM]
- What is the `EstimatedRTT` value (see Section 3.5.3, page 242 in text) after the ACK for the second data-carrying segment is received? [AM]
  Assume that in making this calculation after receiving the ACK for the second segment that the initial 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, and a value of $\alpha = 0.125$.

  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.*

7. What is the length (header plus payload) of each of the first four data-carrying TCP segments?[8] [AM]
8. What is the minimum amount of available buffer space advertised to the client by gaia.cs.umass.edu among these first four data-carrying TCP segments[9]? [AM]
   Does the lack of receiver buffer space ever throttle the sender for these first four data-carrying segments? [MM]
9. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question? [MM]

[8] The TCP segments in the tcp-wireshark-trace1-1 trace file are all less than 1480 bytes. This is because the computer on which the trace was gathered has an interface card that limits the length of the maximum IP datagram to 1500 bytes, and there is a *minimum* of 40 bytes of TCP/IP header data. This 1500-byte value is a fairly typical maximum length for an Internet IP datagram.

[9] Give the Wireshark-reported value for "Window Size Value" which must then be multiplied by the Window Scaling Factor to give the actual number of buffer bytes available at gaia.cs.umass.edu for this connection.

Continued

4. TCP congestion control in action [Optional]

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 client-sent TCP segment in the Wireshark's "listing of captured-packets" window corresponding to the transfer of alice.txt from the client to gaia.cs.umass.edu. Then select the menu: *Statistics->TCP Stream Graph-> Time-Sequence-Graph(Stevens*[10]). You should see a plot that looks similar to the plot in Figure 5, which was created from the captured packets in the packet trace *tcp-wireshark-trace1-1* (see footnote[2]). You may have to expand, shrink, and fiddle around with the intervals shown in the axes in order to get your graph to look like Figure 5. (In practice, you're unlikely to see such neat results.)

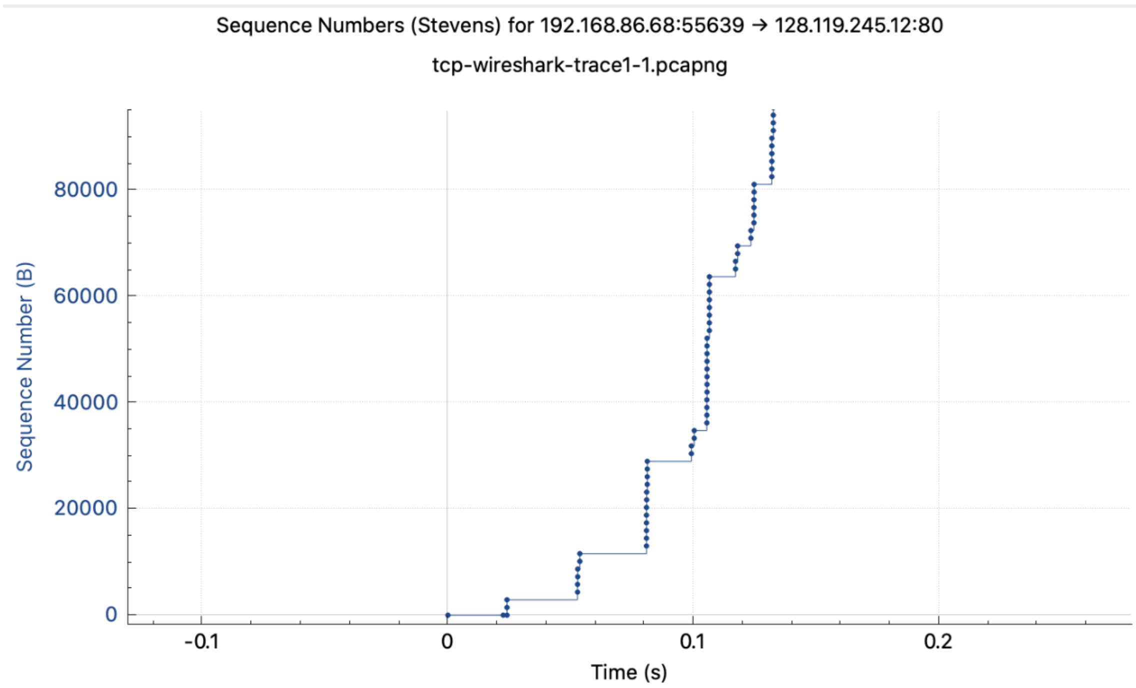


Figure 5: A sequence-number-versus-time plot (Stevens format) of TCP segments.

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 (sometimes called a "fleet" of packets) that were sent back-to-back by the sender.

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 behaviour of TCP that we've studied in the text.

[10] William Stevens wrote the "bible" book on TCP, known as *TCP Illustrated*.

Continued

Note: the reason why this exercise is optional is because your own trace will not nearly be as neat as the one in Fig. 5. That's not your bad. Can you think of a reason why yours is so different?

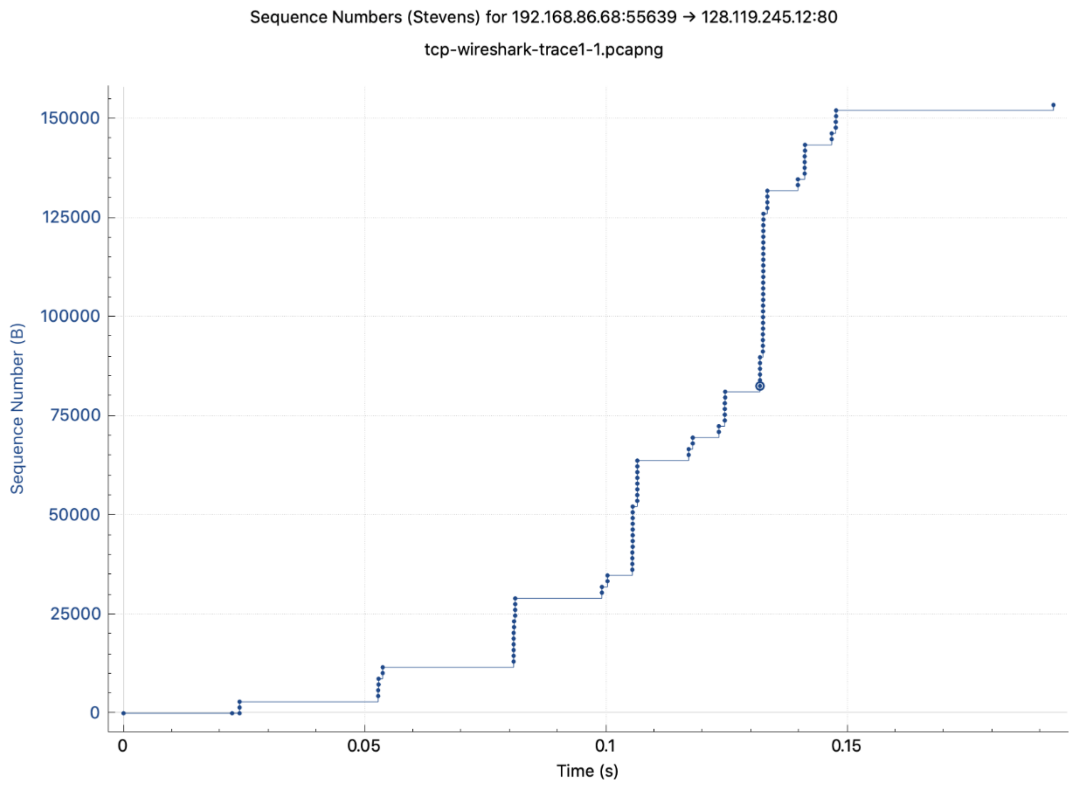
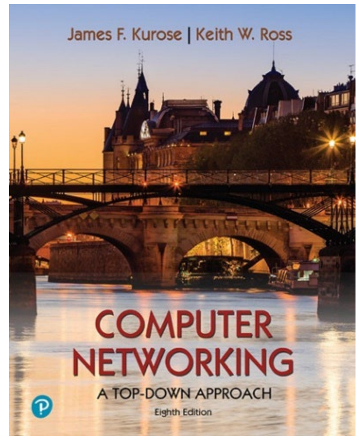


Figure 6: Another view of the same data as in Figure 5.

Continued

Wireshark Lab:
IP v8.1



Supplement to *Computer Networking: A Top-Down Approach, 8^{th} 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 celebrated IP protocol, focusing on the IPv4 and IPv6 datagram. This lab has three parts. In the first part, we'll analyze packets in a trace of IPv4 datagrams sent and received by the `traceroute` program (the `traceroute` program itself is explored in more detail in the Wireshark ICMP lab). We'll study IP fragmentation in Part 2 of this lab, and take a quick look at IPv6 in Part 3 of this lab.

Before getting started, you'll probably want to review sections 1.4.3 in the text[11] and section 3.4 of RFC 2151 to update yourself on the operation of the `traceroute` program. You'll also want to read Section 4.3 in the text, and probably also have RFC 791 on hand as well, for a discussion of the IP protocol. Although we've removed the topic of IP fragmentation from the 8^{th} edition of our textbook (to make room for new material), you can find material on IP fragmentation from the 7^{th} edition of our textbook (and earlier) at http://gaia.cs.umass.edu/kurose_ross/Kurose_Ross_7th_edition_section_4.3.2.pdf.

[11] References to figures and sections are for the 8^{th} edition of our text, *Computer Networks, A Top-down Approach, 8^{h} ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2020.* Our website for this book is http://gaia.cs.umass.edu/kurose_ross You'll find lots of interesting open material there.

Continued

Capturing packets from an execution of traceroute

In order to generate a trace of IPv4 datagrams for the first two parts of this lab, we'll use the `traceroute` program to send datagrams of two different sizes to gaia.cs.umass.edu. Recall that `traceroute` operates by first sending one or more datagrams with the time-to-live (TTL) field in the IP header set to 1; it then sends a series of one or more datagrams towards the same destination with a TTL value of 2; it then sends a series of datagrams towards the same destination with a TTL value of 3; and so on. Recall that a router must decrement the TTL in each received datagram by 1 (actually, RFC 791 says that the router must decrement the TTL by *at least* one). If the TTL reaches 0, the router returns an ICMP message (type 11 – TTL-exceeded) to the sending host. As a result of this behavior, a datagram with a TTL of 1 (sent by the host executing `traceroute`) will cause the router one hop away from the sender to send an ICMP TTL-exceeded message back to the sender; the datagram sent with a TTL of 2 will cause the router two hops away to send an ICMP message back to the sender; the datagram sent with a TTL of 3 will cause the router three hops away to send an ICMP message back to the sender; and so on. In this manner, the host executing `traceroute` can learn the IP addresses of the routers between itself and the destination by looking at the source IP addresses in the datagrams containing the ICMP TTL-exceeded messages.

Let's run `traceroute` and have it send datagrams of two different sizes. The larger of the two datagram lengths will require `traceroute` messages to be fragmented across multiple IPv4 datagrams.

- **Linux/MacOS.** With the Linux/MacOS `traceroute` command, the size of the UDP datagram sent towards the final destination can be explicitly set by indicating the number of bytes in the datagram; this value is entered in the `traceroute` command line immediately after the name or address of the destination. For example, to send `traceroute` datagrams of 2000 bytes towards gaia.cs.umass.edu, the command would be:
  ```
  %traceroute gaia.cs.umass.edu 2000
  ```
- **Windows.** The `tracert` program provided with Windows does not allow one to change the size of the ICMP message sent by `tracert`. So it won't be possible to use a Windows machine to generate ICMP messages that are large enough to force IP fragmentation. However, you can use `tracert` to generate small, fixed length packets to perform Part 1 of this lab. At the DOS command prompt enter:
  ```
  >tracert gaia.cs.umass.edu
  ```

Continued

If you want to do the second part of this lab, you can download a packet trace file that was captured on one of the author's computers[12].

Do the following:

- Start up Wireshark and begin packet capture. *(Capture->Start* or click on the blue shark fin button in the top left of the Wireshark window).
- Enter two `traceroute` commands, using gaia.cs.umass.edu as the destination, the first with a length of 56 bytes. Once that command has finished executing, enter a second `traceroute` command for the same destination, but with a length of 3000 bytes.
- Stop Wireshark tracing.

If you're unable to run Wireshark on a live network connection, you can use the packet trace file, *ip-wireshark-trace1-1.pcapng*, referenced in footnote 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, as you explore the questions below.

Part 1: Basic IPv4

In your trace, you should be able to see the series of UDP segments (in the case of MacOS/Linux) or ICMP Echo Request messages (Windows) sent by `traceroute` on your computer, and the ICMP TTL-exceeded messages returned to your computer by the intermediate routers. In the questions below, we'll assume you're using a MacOS/Linux computer; the corresponding questions for the case of a Windows machine should be clear. Your screen should look similar to the screenshot in Figure 2, where we have used the display filter "udp||icmp" (see the light-green-filled display-filter field in Figure 2) so that only UDP and/or ICMP protocol packets are displayed.

[12] You can download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces-8.1.zip and extract the trace file *ip-wireshark-trace1-1.pcapng*. This trace file can be used to answer these Wireshark lab questions without actually capturing packets on your own. The trace was made using Wireshark running on one of the author's computers, while performing the steps in this Wireshark lab. Once you've downloaded a trace file, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the trace file name.

Continued

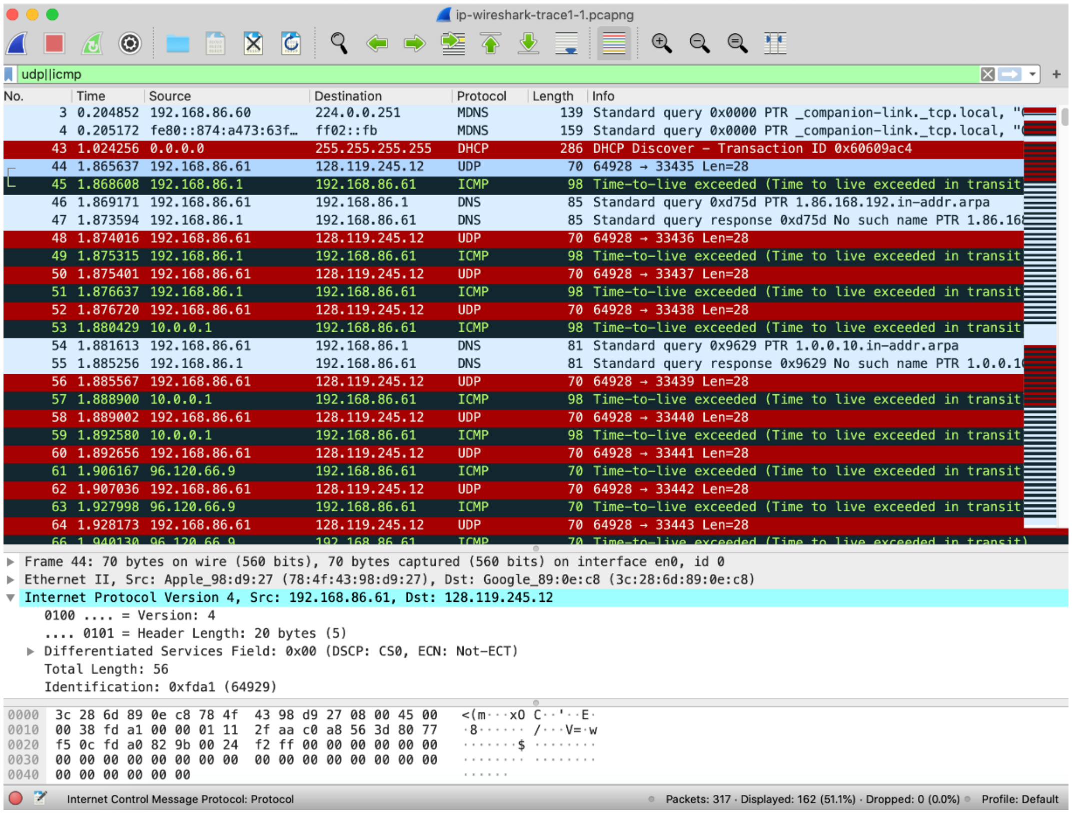


Figure 2: Wireshark screenshot, showing UDP and ICMP packets in the tracefile *ip-wireshark-trace1-1.pcapng*

Answer the following questions[13]. If you're doing this lab as part of class, your teacher will provide details about how to hand in assignments, whether written or in an LMS.

10. Select the first UDP segment sent by your computer via the `traceroute` command to gaia.cs.umass.edu and expand the Internet Protocol part of the packet in the packet details window. What is the IP address of your computer? [AM]
11. What is the value in the time-to-live (TTL) field in this IPv4 datagram's header? [AM]

[13] For the author's class, when answering the following questions with hand-in assignments, students sometimes need to print out specific packets (see the introductory Wireshark lab for an explanation of how to do this) and indicate where in the packet they've found the information that answers a question. They do this by marking paper copies with a pen or annotating electronic copies with text in a colored font. There are also learning management system (LMS) modules for teachers that allow students to answer these questions online and have answers auto-graded for these Wireshark labs at http://gaia.cs.umass.edu/kurose_ross/lms.htm

Continued

12. What is the value in the upper layer protocol field in this IPv4 datagram's header? [Note: the answers for Linux/MacOS differ from Windows here]. [AM]
13. How many bytes are in the IP header? How many bytes are in the payload of the IP datagram? [AM]
14. Has this IP datagram been fragmented? [AM]
15. Explain how you determined whether or not the datagram has been fragmented. [MM]

Next, let's look at the *sequence* of UDP segments being sent from your computer via `traceroute`, destined to 128.119.245.12. The display filter that you can enter to do this is "ip.src==192.168.86.61 and ip.dst==128.119.245.12 and udp and !icmp". This will allow you to easily move sequentially through just the datagrams containing just these segments. Your screen should look similar to Figure 3.

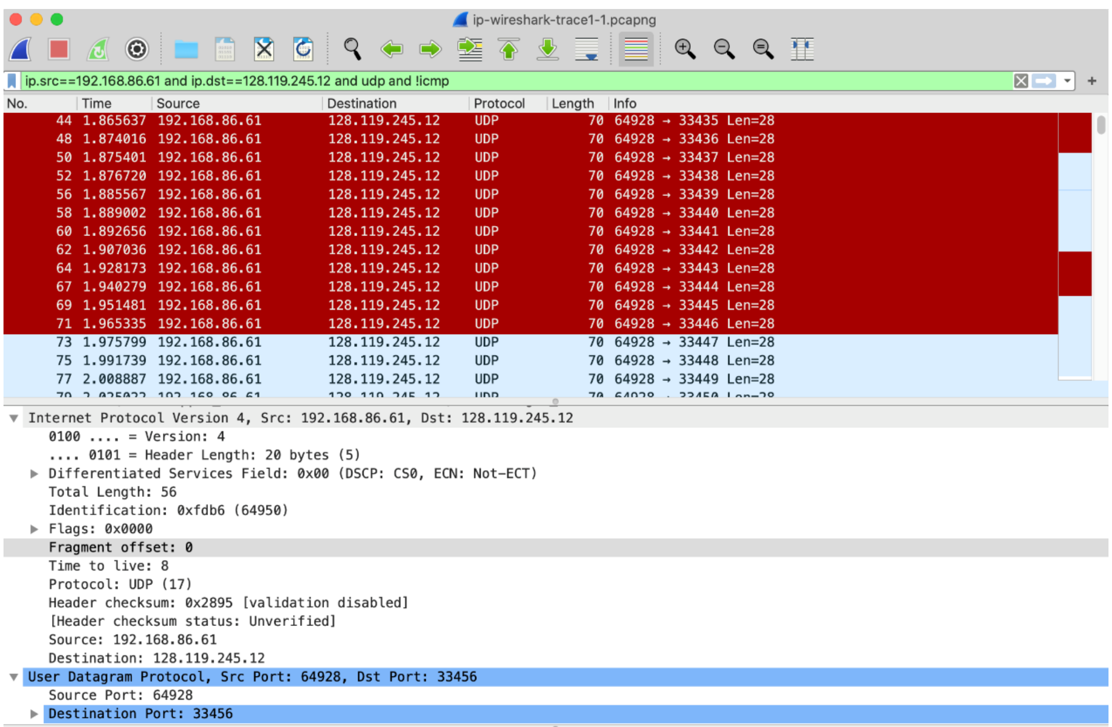


Figure 3: Wireshark screen shot, showing up segments in the tracefile *ip-wireshark-trace1-1.pcapng* using the display filter `ip.src==192.168.86.61 and ip.dst==128.119.245.12 and udp and !icmp`

---

16. Which fields in the IP datagram *always* change from one datagram to the next within this series of UDP segments sent by your computer destined to 128.119.245.12, via `traceroute`? [MM]
17. Which fields in this sequence of IP datagrams (containing UDP segments) stay constant? Why? [MM]

Continued

18. Describe the pattern you see in the values in the Identification field of the IP datagrams being sent by your computer. [MM]

Now let's take a look at the ICMP packets being returned to your computer by the intervening routers where the TTL value was decremented to zero (and hence caused the ICMP error message to be returned to your computer). The display filter that you can use to show just these packets is "ip.dst==192.168.86.61 and icmp".

19. What is the upper layer protocol specified in the IP datagrams returned from the routers? [Note: the answers for Linux/MacOS differ from Windows here]. [MM]
20. Are the values in the Identification fields (across the sequence of all of ICMP packets from all of the routers) similar in behavior to your answer to question 18 above? [MM]
21. Are the values of the TTL fields similar, across all of ICMP packets from all of the routers? [MM]

End