CSC3002F Networks Assignment 2

Sniffing with Wireshark

Lecturer: Dr. Josiah Chavula
Department of Computer Science
University of Cape Town, South Africa

Contents

1 Introduction **2**

2 Sniffing with Wireshark **3**

2.1 TCP Wireshark lab 3

2.1.1 Capturing a bulk TCP transfer from your computer to a remote server 3

2.1.2 A first look at the captured trace 4

2.1.3 TCP Basics 5

2.1.4 TCP congestion control in action (optional!) 7

2.2 IP Wireshark lab 7

2.2.1 Capturing packets from an execution of `traceroute` 8

2.2.2 A look at the captured trace 9

A Appendix: Automarker format **12**

1 Introduction

This assignment focuses on the network's 'packets' being sent around on a computer network. You will take a look at what is actually sent around on the transmission medium and what those packets 'look like'. You may have heard of IP packets, but what exactly is 'inside' it? Or sending 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.

What you have to submit *in one compressed file* where *no file is in a directory* (as you should recall from prior exposure to the automated marker in previous courses):

1. A trace file called 'tcptrace.pcap' containing your trace for the first half of the aassignment (questions 1-9) ;
2. A trace file called 'iptrace.pcap' containing your trace for the second half of the aassignment (questions 10-18) ;
3. The answers to the automarker-marked questions in a structured text file called 'automark.txt'; these questions are indicated with "`[AM]`"; details of the file structure and a sample file are on the Vula page for the assignment;
4. The answers (optionally including annotated screenshots) to the remaining questions of the Wireshark section, as indicated.

NOTE
You can submit only FIVE times, so be careful and think and check twice, or even thrice, before submitting
Your traces must be saved as 'pcap' files, not 'pcapng' files.

NOTE
The Automarker is uniquely configured for the Wireshark section, in that it processes your trace to pull out the answers, and compares it to what you have written in the txt file. As we have a list who gets assigned which file to upload, we immediately know who's been copying in case you submit someone else's trace and answer file. So don't even try, unless you want a 0 for the assignment and be added onto the departmental plagiarism list, or if you are already on there, go through the full UCT disciplinary procedures for plagiarism (see departmental plagiarism policy for details). *The format of the answer file and an example are available on Vula and shown in Appendix A of this file; check these before submitting it.*

2 Sniffing with Wireshark

⇒ There's a lengthy explanation of steps, but the trace capturing tasks are fairly short. You may wish to read through the TCP and IP section first, then capture the required traces before proceeding to answer the questions. ⇐

Note: when the description says you're better off closing all other networked apps before starting the capture, do yourself a favour, and do so. Then answer the questions with that 'clean' trace.(It is most fascinating to see how busy your networked apps are even when you think nothing is happening, but that's a different topic.)

2.1 TCP Wireshark lab

In this part of the practical assignment, you'll investigate the behaviour of the TCP protocol. You'll do so by analysing a trace of the TCP segments sent and received in transferring a file from your computer to a remote server. That file is allocated to you and is old enough to have had their copyright expired, such as Lewis Carrol's Alice's Adventures in Wonderland (many of those books are digitised and available from Project Gutenberg[1]). You'll study TCP's use of sequence and acknowledgement numbers for providing reliable data transfer. TCP's congestion control algorithm is nice, but your data will look 'hectic' and therefore is an optional exercise, and likewise on TCP's receiver-advertised flow control mechanism. You will also briefly consider TCP connection setup and 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 textbook. There is a separate Wireshark introduction pdf file on vula that you may want to read through, which gives a little overview of the tool, and there's the manual online. As they keep on changing things in the interface and there are slight differences across OSs, it may be that the screenshots below in this document aren't exactly the same as what you'll see, but worry not, the questions can be answered.

2.1.1 Capturing a bulk TCP transfer from your computer to a remote server

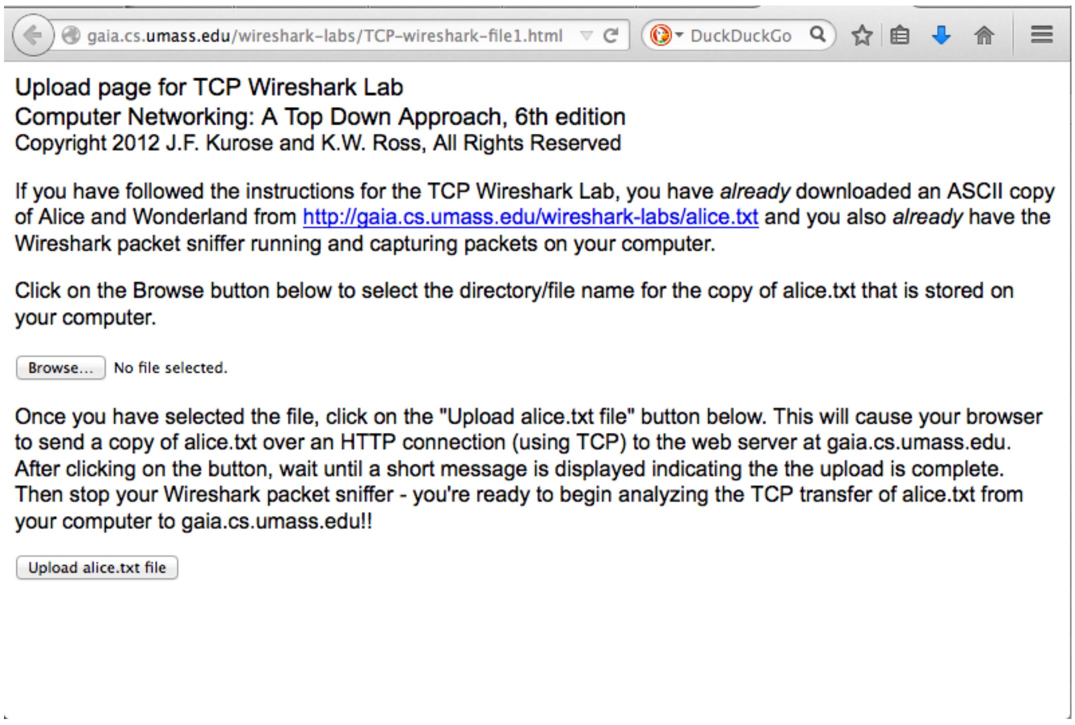
You'll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You'll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of your assigned Gutenberg ebook), and then transfer the file to a Web server using the HTTP POST method (see section 2.2.3 in the text). The POST method is used rather than the GET method, because you're going to transfer a large amount of data from your computer to another computer. You have to run 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. On Vula, go to CSC3002F,2017 Resources>Networks>Text File Allocations, and retrieve an ASCII copy of the Gutenberg ebook assigned to you, your gutenbergfile.txt. 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 the screenshot in Fig. 1. Use the Browse button in this form to enter the name of the file (full path name) on your computer containing yourgutenbergfile (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. Note: if it doesn't work, go to Capture –> Interfaces and select the NIC and then click Start

[1] `https://www.gutenberg.org/`



Upload page for TCP Wireshark Lab
Computer Networking: A Top Down Approach, 6th edition
Copyright 2012 J.F. Kurose and K.W. Ross, All Rights Reserved

If you have followed the instructions for the TCP Wireshark Lab, you have *already* downloaded an ASCII copy of Alice and Wonderland from http://gaia.cs.umass.edu/wireshark-labs/alice.txt and you also *already* have the Wireshark packet sniffer running and capturing packets on your computer.

Click on the Browse button below to select the directory/file name for the copy of alice.txt that is stored on your computer.

Once you have selected the file, click on the "Upload alice.txt file" button below. This will cause your browser to send a copy of alice.txt over an HTTP connection (using TCP) to the web server at gaia.cs.umass.edu. After clicking on the button, wait until a short message is displayed indicating the the upload is complete. Then stop your Wireshark packet sniffer - you're ready to begin analyzing the TCP transfer of alice.txt from your computer to gaia.cs.umass.edu!!

Figure 1: Upload page for yourgutenbergfile.txt

– Return to your browser and 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.

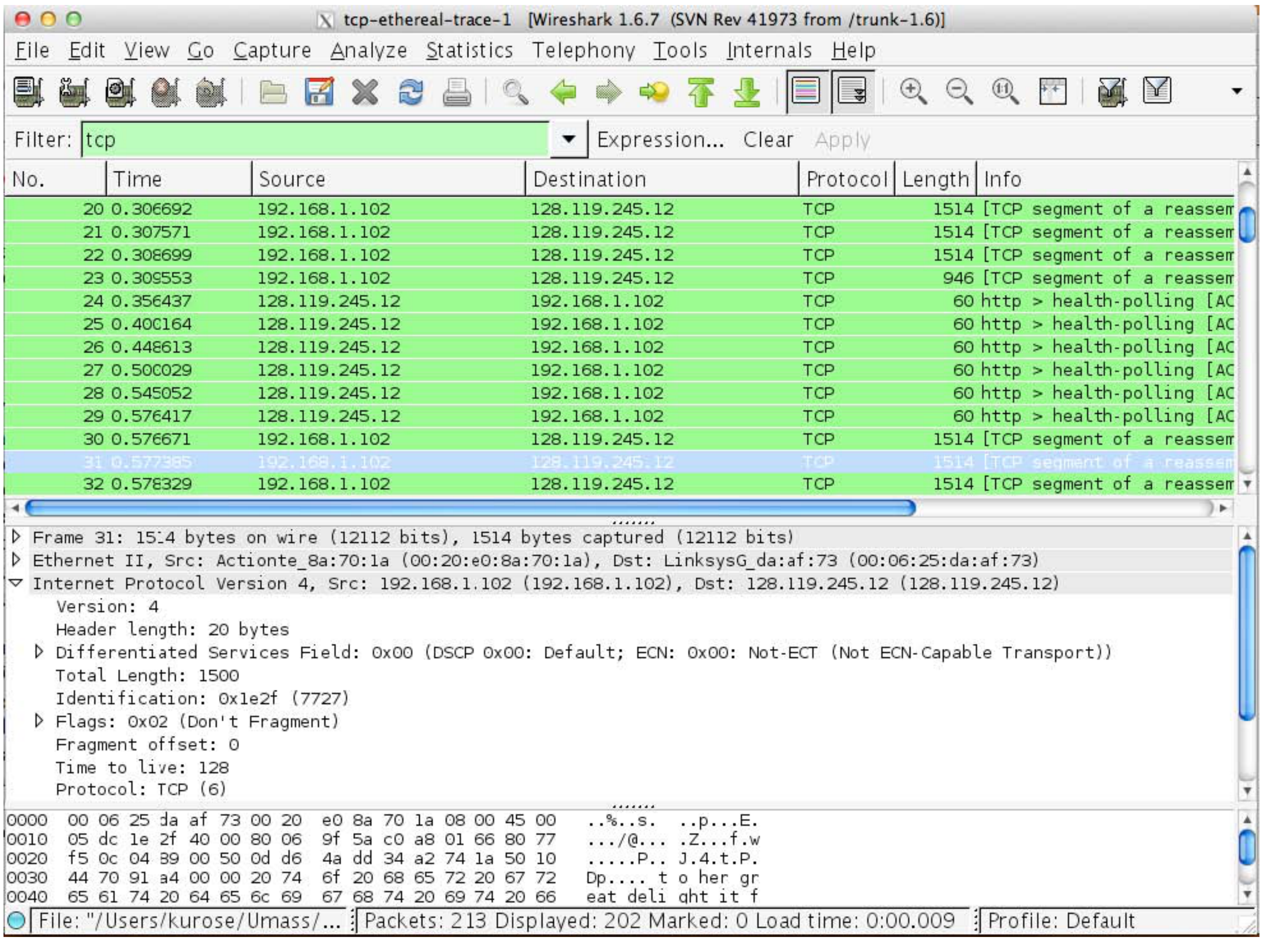
– Stop the packet capture and save as 'tcptrace.pcap'.

–Your Wireshark window should look similar to the screenshot in Fig. 2. (Note: it may look slightly different, which is due to either difference in operating system or yet another version of the software.). Scroll down to the IP section and do that part of the trace capture.

NOTE: you certainly will be able to run Wireshark in the labs. It may well be the case that when you (want to) start doing this exercise only some 5 hours before the assignment deadline, some server is down or the network happens to be not working. *that's your problem* and not an excuse or a valid reason for extension. There's plenty of time to do these exercises, and most of the time, the network and servers are working.

2.1.2 A first look at the captured trace

Before analysing the behaviour 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 Wireshark you are using, you might see a series of "HTTP Continuation" messages being sent from your computer to gaia.cs.umass.edu. There 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.


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

Filter: tcp

```
No.  Time      Source          Destination     Protocol Length Info
  20 0.306692  192.168.1.102   128.119.245.12  TCP      1514 [TCP segment of a reassem
  21 0.307571  192.168.1.102   128.119.245.12  TCP      1514 [TCP segment of a reassem
  22 0.308699  192.168.1.102   128.119.245.12  TCP      1514 [TCP segment of a reassem
  23 0.309553  192.168.1.102   128.119.245.12  TCP       946 [TCP segment of a reassem
  24 0.356437  128.119.245.12  192.168.1.102   TCP        60 http > health-polling [AC
  25 0.400164  128.119.245.12  192.168.1.102   TCP        60 http > health-polling [AC
  26 0.448613  128.119.245.12  192.168.1.102   TCP        60 http > health-polling [AC
  27 0.500029  128.119.245.12  192.168.1.102   TCP        60 http > health-polling [AC
  28 0.545052  128.119.245.12  192.168.1.102   TCP        60 http > health-polling [AC
  29 0.576417  128.119.245.12  192.168.1.102   TCP        60 http > health-polling [AC
  30 0.576671  192.168.1.102   128.119.245.12  TCP      1514 [TCP segment of a reassem
  31 0.577385  192.168.1.102   128.119.245.12  TCP      1514 [TCP segment of a reassem
  32 0.578329  192.168.1.102   128.119.245.12  TCP      1514 [TCP segment of a reassem
```

```
▷ Frame 31: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 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)
▽ Internet Protocol Version 4, Src: 192.168.1.102 (192.168.1.102), Dst: 128.119.245.12 (128.119.245.12)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 1500
    Identification: 0x1e2f (7727)
  ▷ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (6)
```

```
0000  00 06 25 da af 73 00 20  e0 8a 70 1a 08 00 45 00   ..%..s.  ..p...E.
0010  05 dc 1e 2f 40 00 80 06  9f 5a c0 a8 01 66 80 77   .../@... .Z...f.w
0020  f5 0c 04 89 00 50 0d d6  4a dd 34 a2 74 1a 50 10   .....P.. J.4.t.P.
0030  44 70 91 a4 00 00 20 74  6f 20 68 65 72 20 67 72   Dp.... t o her gr
0040  65 61 74 20 64 65 6c 69  67 68 74 20 69 74 20 66   eat deli ght it f
```

File: "/Users/kurose/Umass/... | Packets: 213 Displayed: 202 Marked: 0 Load time: 0:00.009 | Profile: Default

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply

Figure 2: A screenshot of Wireshark after packet capture.

Answer the following questions, by using your own trace. See Appendix A for the format of the txt file for the questions that will be marked by the automatic marker (indicated with "`[AM]`"). For the other manually marked questions, write the answer in a separate file (e.g., MS Word. Open Office, LaTeX) and, where applicable, include a screenshot of the packet(s) within the trace that you used to answer the question asked, or annotate the packet to explain your answers. (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.)

Exercises

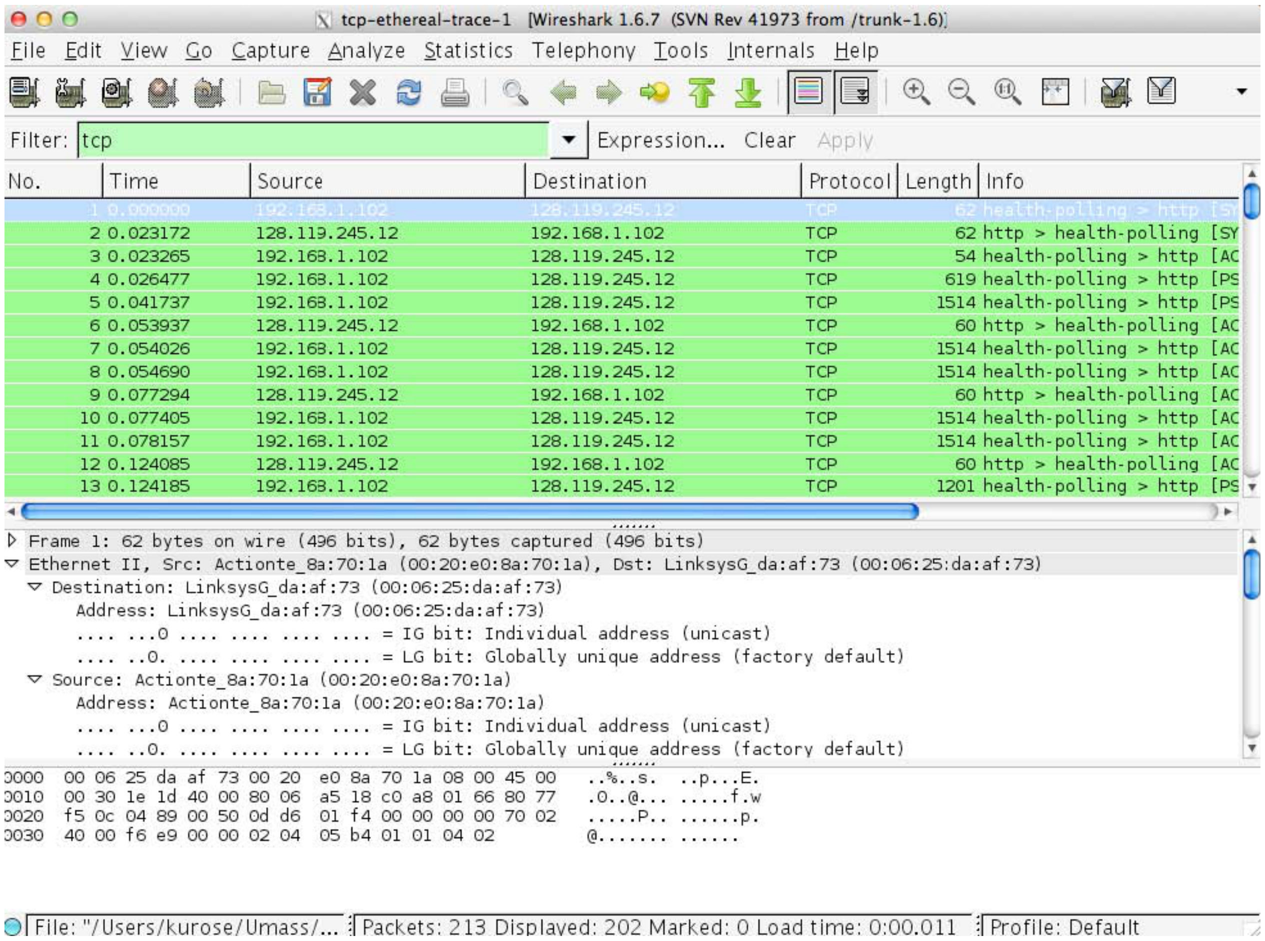
1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it's probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the "details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

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]`

2.1.3 TCP Basics

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 the one in Fig. 3.


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 [SY
    2 0.023172   128.119.245.12   192.168.1.102    TCP          62 http > health-polling [SY
    3 0.023265   192.168.1.102    128.119.245.12   TCP          54 health-polling > http [AC
    4 0.026477   192.168.1.102    128.119.245.12   TCP         619 health-polling > http [PS
    5 0.041737   192.168.1.102    128.119.245.12   TCP        1514 health-polling > http [PS
    6 0.053937   128.119.245.12   192.168.1.102    TCP          60 http > health-polling [AC
    7 0.054026   192.168.1.102    128.119.245.12   TCP        1514 health-polling > http [AC
    8 0.054690   192.168.1.102    128.119.245.12   TCP        1514 health-polling > http [AC
    9 0.077294   128.119.245.12   192.168.1.102    TCP          60 http > health-polling [AC
   10 0.077405   192.168.1.102    128.119.245.12   TCP        1514 health-polling > http [AC
   11 0.078157   192.168.1.102    128.119.245.12   TCP        1514 health-polling > http [AC
   12 0.124085   128.119.245.12   192.168.1.102    TCP          60 http > health-polling [AC
   13 0.124185   192.168.1.102    128.119.245.12   TCP        1201 health-polling > http [PS

▷ 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)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = 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)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
      .... ..0. .... .... .... .... = 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
```

Figure 3: Another screenshot of Wireshark after packet capture.

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

Exercises

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? What is it in the segment that identifies the segment as a SYN segment? `[AM]`

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? 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? `[AM]`

5. 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. `[AM]`

6. 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.) `[AM]`

7. What is the length of each of the first six TCP segments? `[AM]`
8. a) What is the minimum amount of available buffer space advertised at the received for the entire trace? `[AM]`
   b) Does the lack of receiver buffer space ever throttle the sender? note: Answer this in the document that will not be marked automatically
9. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question? NOTE: Answer this in the document that will not be marked automatically.

2.1.4 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 TCP segment in the Wireshark's "listing of captured-packets" window. Then select the menu : Statistics –> TCP Stream Graph –> Time-Sequence-Graph(Stevens). In theory, you should see a plot that looks similar to the plot as shown in Figure 4, 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. In practice, you're unlikely to see such neat results with your own trace.

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.

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? If not (or extremely hard to find out form the graph), why not? Comment on ways in which the measured data differs from the idealised behaviour of TCP that we've studied in the text.

Note: the reason why this exercise is optional is because your own trace will not nearly be as neat as this one in Fig. 4. That's not your bad. Can you think of a reason why yours will be so different?

2.2 IP Wireshark lab

In this part of the practical assignment, you'll investigate the IP protocol, focusing on the IP datagram. You'll do so by analysing a trace of IP datagrams sent and received by an execution of the `traceroute` program[2]. We'll investigate the various fields in the IP datagram, and study IP fragmentation in detail.

[2]The `traceroute` program itself is explored in more detail elsewhere in the Wireshark ICMP lab that we don't cover in the pracs, but is available on Vula in case you're interested

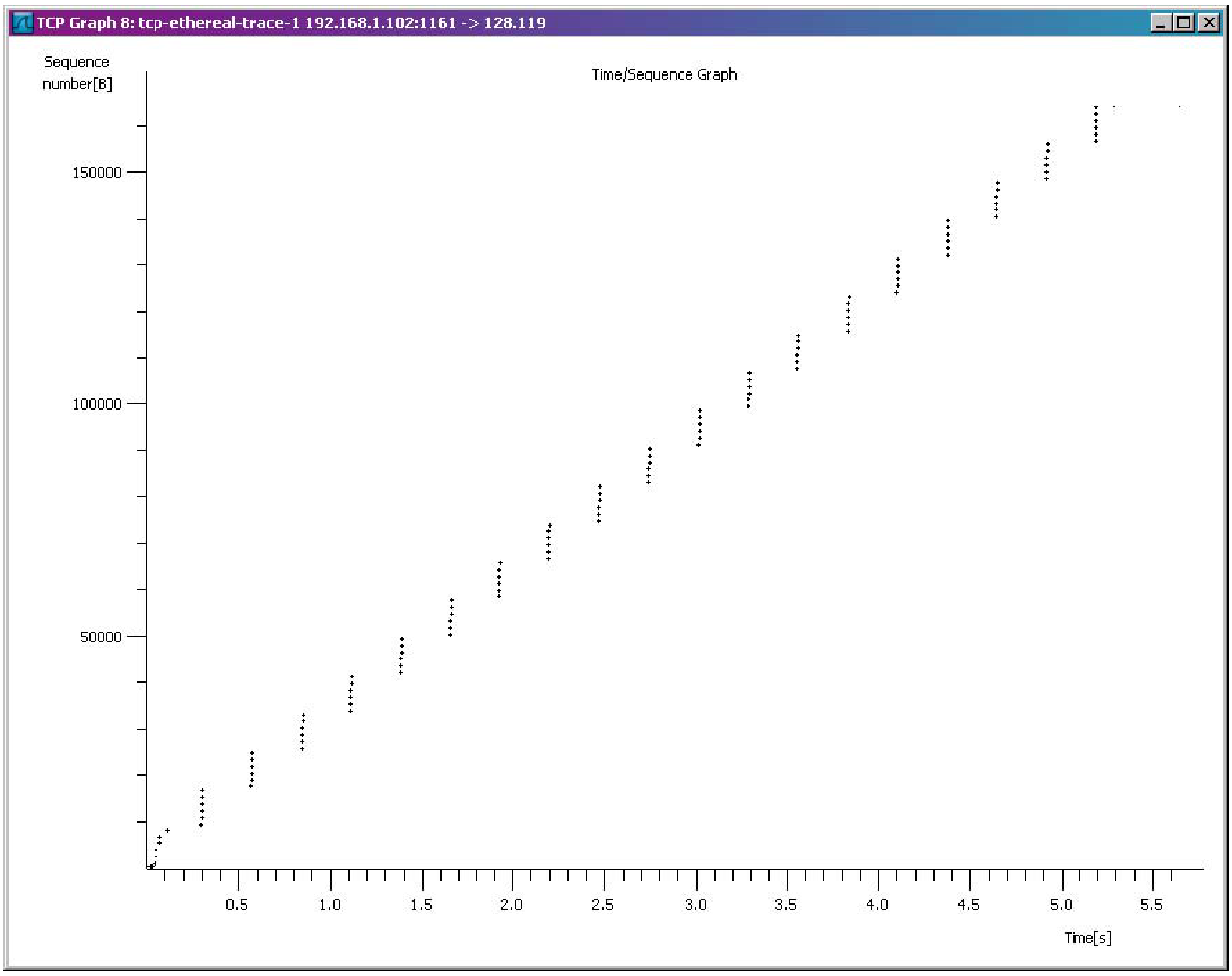


Figure 4: Time sequence graph with the book's clean data.

Before beginning this lab, you'll probably want to review sections 1.4.3 in the text and section 3.4 of RFC 2151[3] to update yourself on the operation of the `traceroute` program. You'll also want to read Section 4.4 in the text, and probably also have RFC 791[4] on hand as well, for a discussion of the IP protocol.

2.2.1 Capturing packets from an execution of `traceroute`

In order to generate a trace of IP datagrams for this lab, we'll use the `traceroute` program to send datagrams of different sizes towards some destination, X. 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 behaviour, 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

[3]`ftp://ftp.rfc-editor.org/in-notes/rfc2151.txt`
[4]`ftp://ftp.rfc-editor.org/in-notes/rfc791.txt`

message back to the sender; and so on. In this manner, the host executing `traceroute` can learn the identities of the routers between itself and destination X by looking at the source IP addresses in the datagrams containing the ICMP TTL-exceeded messages.

We'll want to run `traceroute` and have it send datagrams of various lengths.

- Windows. The `tracert` program (used for our ICMP Wireshark lab) provided with Windows does not allow one to change the size of the ICMP echo request (ping) message sent by the `tracert` program[5].
- Linux/Unix/MacOS. With the Unix/MacOS `traceroute` command, the size of the UDP datagram sent towards the 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
  ```

Do the following:

- Start up Wireshark and begin packet capture (Capture −> Start) and then press OK on the Wireshark Packet Capture Options screen (we won't need to select any options here), if you did not continue with the TCP capture after all.
- If you are using a Windows platform, use the command prompt[6]. If you are using a Unix or Mac platform, enter three `traceroute` commands, one with a length of 56 bytes, one with a length of 2000 bytes, and one with a length of 3500 bytes. You can do the same with `pingplotter`, but not with `tracert`.
- Next, send a set of datagrams with a longer length, by selecting Edit −> Advanced Options −> Packet Options and enter a value of 2000 in the Packet Size field and then press OK. Then press the Resume button.
- Stop Wireshark tracing when it has completed the trace. Save the trace as 'iptrace.pcap'.

2.2.2 A look at the captured trace

In your trace, you should be able to see the series of ICMP Echo Request (in the case of Windows machine) or the UDP segment (in the case of Unix) sent by your computer and the ICMP TTL-exceeded messages returned to your computer by the intermediate routers. In the questions below, we'll assume you are using a Windows machine; the corresponding questions for the case of a Unix machine should be clear. When answering a question below you should hand in a softcopy of the packet(s) within the trace that you used to answer the question asked. 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 to the manually marked questions.

Exercises

10. Select the first ICMP Echo Request message sent by your computer, and expand the Internet Protocol part of the packet in the packet details window (see also Fig. 7). What is the IP address of your computer? `[AM]`

11. Within the IP packet header, what is the value in the upper layer protocol field? `[AM]`

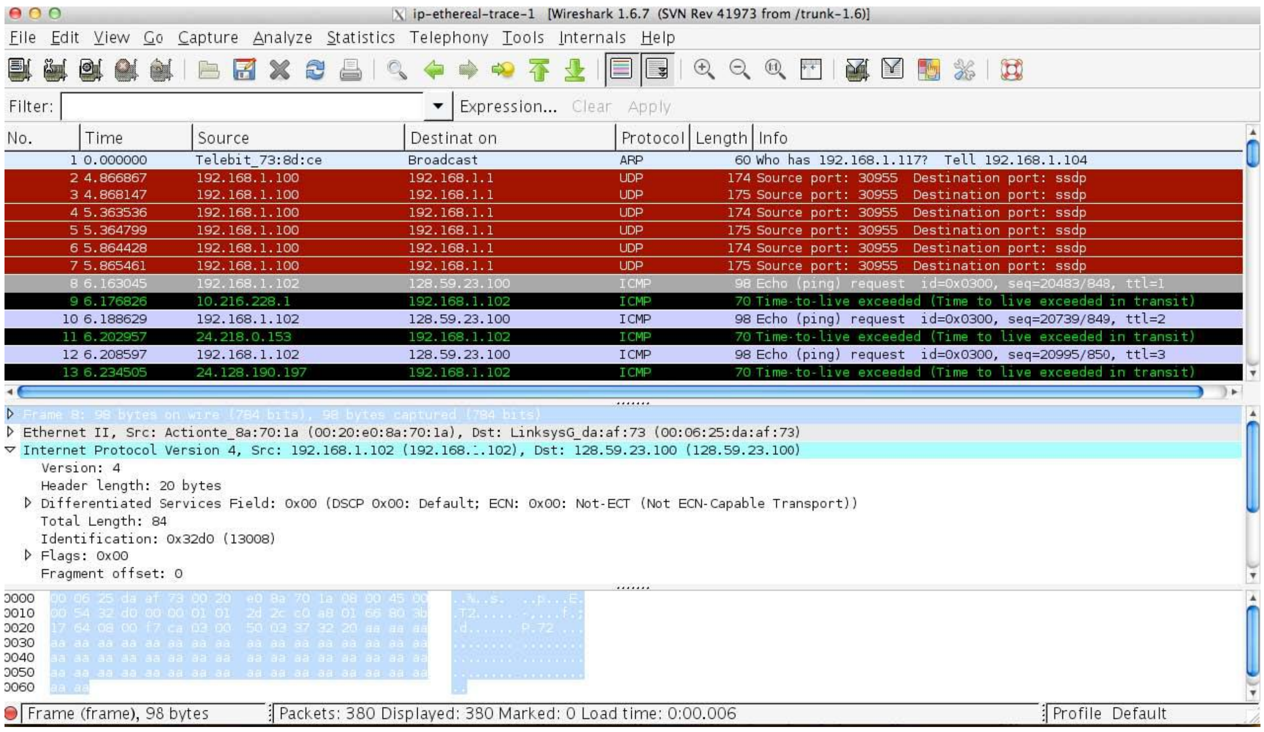
[5]A nicer Windows `traceroute` program is pingplotter, available both in free version and shareware versions at http://www.pingplotter.com. Download and install `pingplotter`, and test it out by performing a few `traceroutes` to your favorite sites. The size of the ICMP echo request message can be explicitly set in `pingplotter` by selecting the menu item Edit − > Options − > Packet Options and then filling in the Packet Size field. The default packet size is 56 bytes. Once `pingplotter` has sent a series of packets with the increasing TTL values, it restarts the sending process again with a TTL of 1, after waiting Trace Interval amount of time. The value of Trace Interval and the number of intervals can be explicitly set in `pingplotter`.

[6]If you use pingplotter, then: start up pingplotter and enter the name of a target destination in the "Address to Trace Window." Enter 3 in the "# of times to Trace" field, so you don't gather too much data. Select the menu item Edit −> Advanced Options −> Packet Options and enter a value of 56 in the Packet Size field and then press OK. Then press the Trace button.

12. Has this IP datagram been fragmented? [AM]

13. Explain how you determined whether or not the datagram has been fragmented.

14. How many bytes are in the IP header? How many bytes are in the payload of the IP datagram?



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

No.  Time       Source            Destinat on      Protocol Length Info
   1 0.000000   Telebit_73:8d:ce  Broadcast        ARP         60 Who has 192.168.1.117?  Tell 192.168.1.104
   2 4.866867   192.168.1.100     192.168.1.1      UDP        174 Source port: 30955  Destination port: ssdp
   3 4.868147   192.168.1.100     192.168.1.1      UDP        175 Source port: 30955  Destination port: ssdp
   4 5.363536   192.168.1.100     192.168.1.1      UDP        174 Source port: 30955  Destination port: ssdp
   5 5.364799   192.168.1.100     192.168.1.1      UDP        175 Source port: 30955  Destination port: ssdp
   6 5.864428   192.168.1.100     192.168.1.1      UDP        174 Source port: 30955  Destination port: ssdp
   7 5.865461   192.168.1.100     192.168.1.1      UDP        175 Source port: 30955  Destination port: ssdp
   8 6.163045   192.168.1.102     128.59.23.100    ICMP        98 Echo (ping) request  id=0x0300, seq=20483/848, ttl=1
   9 6.176826   10.216.228.1      192.168.1.102    ICMP        70 Time-to-live exceeded (Time to live exceeded in transit)
  10 6.188629   192.168.1.102     128.59.23.100    ICMP        98 Echo (ping) request  id=0x0300, seq=20739/849, ttl=2
  11 6.202957   24.218.0.153      192.168.1.102    ICMP        70 Time-to-live exceeded (Time to live exceeded in transit)
  12 6.208597   192.168.1.102     128.59.23.100    ICMP        98 Echo (ping) request  id=0x0300, seq=20995/850, ttl=3
  13 6.234505   24.128.190.197    192.168.1.102    ICMP        70 Time-to-live exceeded (Time to live exceeded in transit)

▷ Frame 8: 98 bytes on wire (784 bits), 98 bytes captured (784 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)
▽ Internet Protocol Version 4, Src: 192.168.1.102 (192.168.1.102), Dst: 128.59.23.100 (128.59.23.100)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x32d0 (13008)
  ▷ Flags: 0x00
    Fragment offset: 0

0000  00 06 25 da af 73 00 20  e0 8a 70 1a 08 00 45 00   ..%..s. ..p...E.
0010  00 54 32 d0 00 00 01 01  2d 2c c0 a8 01 66 80 3b   .T2..... -,...f.;
0020  17 64 08 00 f7 ca 03 00  50 03 37 32 20 aa aa aa   .d...... P.72 ...
0030  aa aa aa aa aa aa aa aa  aa aa aa aa aa aa aa aa   ........ ........
0040  aa aa aa aa aa aa aa aa  aa aa aa aa aa aa aa aa   ........ ........
0050  aa aa aa aa aa aa aa aa  aa aa aa aa aa aa aa aa   ........ ........
0060  aa aa                                              ..

Frame (frame), 98 bytes | Packets: 380 Displayed: 380 Marked: 0 Load time: 0:00.006 | Profile Default
```

Figure 5: Screenshot related to question 10.

Next, sort the traced packets according to IP source address by clicking on the Source column header; a small downward pointing arrow should appear next to the word Source. If the arrow points up, click on the Source column header again. Select the first ICMP Echo Request message sent by your computer, and expand the Internet Protocol portion in the “details of selected packet header” window. In the “listing of captured packets” window, you should see all of the subsequent ICMP messages (perhaps with additional interspersed packets sent by other protocols running on your computer) below this first ICMP. Use the down arrow to move through the ICMP messages sent by your computer.

15. Which fields in the IP datagram always change from one datagram to the next within this series of ICMP messages sent by your computer?

16. Which fields stay constant? Which of the fields *must* stay constant? Which fields must change? Why?

17. Describe the pattern you see in the values in the Identification field of the IP datagram

Next (with the packets still sorted by source address) find the series of ICMP TTL-exceeded replies sent to your computer by the nearest router.

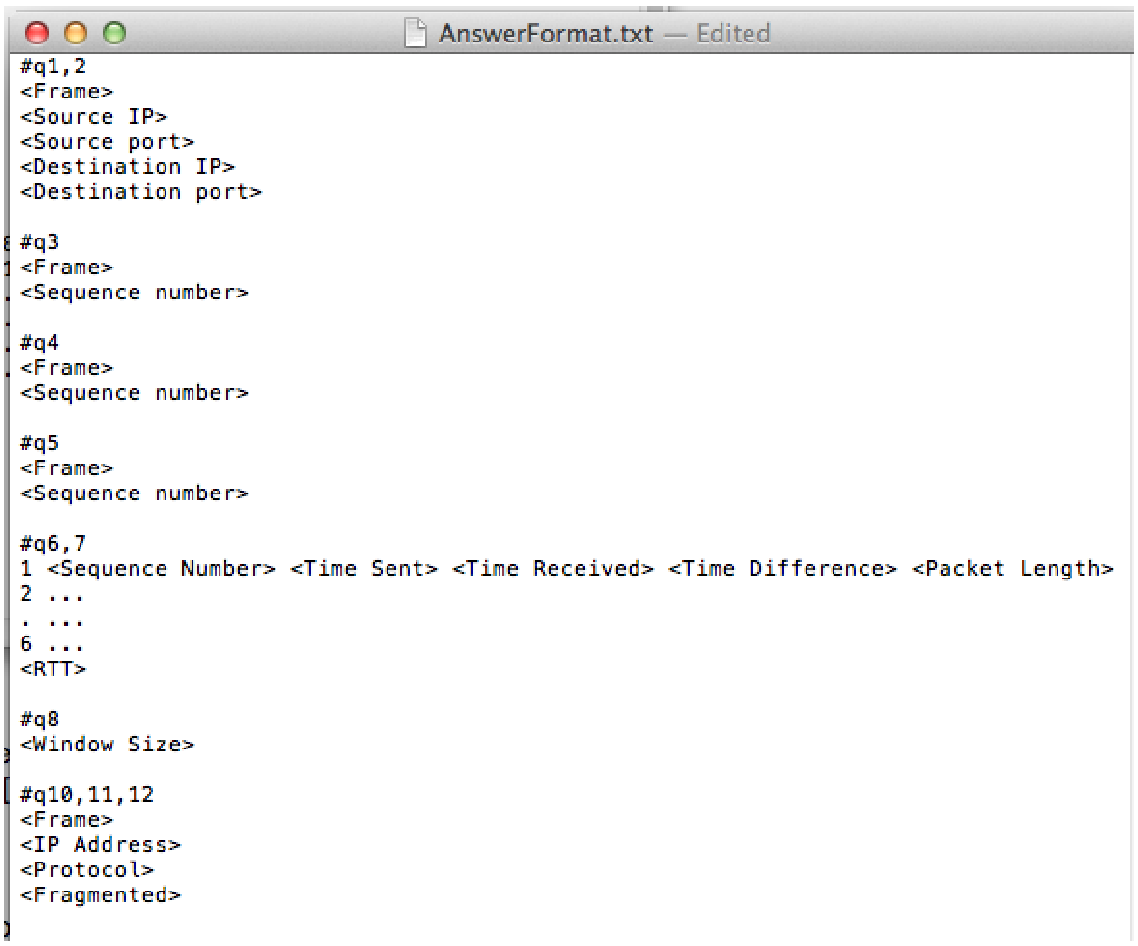
17. Describe how you found it. What is the value in the Identification field and the TTL field?

18. Do these values remain unchanged for all of the ICMP TTL-exceeded replies sent to your computer by the nearest (first hop) router? Why?

Credits: these Wireshark exercises are heavily based on Kurose and Ross' material for the Wireshark labs accompanying the book, but note that *some strategic modifications have been made* to prevent you from copying the answers from those online available (well, you could, but then the answers would be wrong). That said, if you're unsure what to do, you could find that lab assignment and answers and practice with that first.

A Appendix: Automarker format

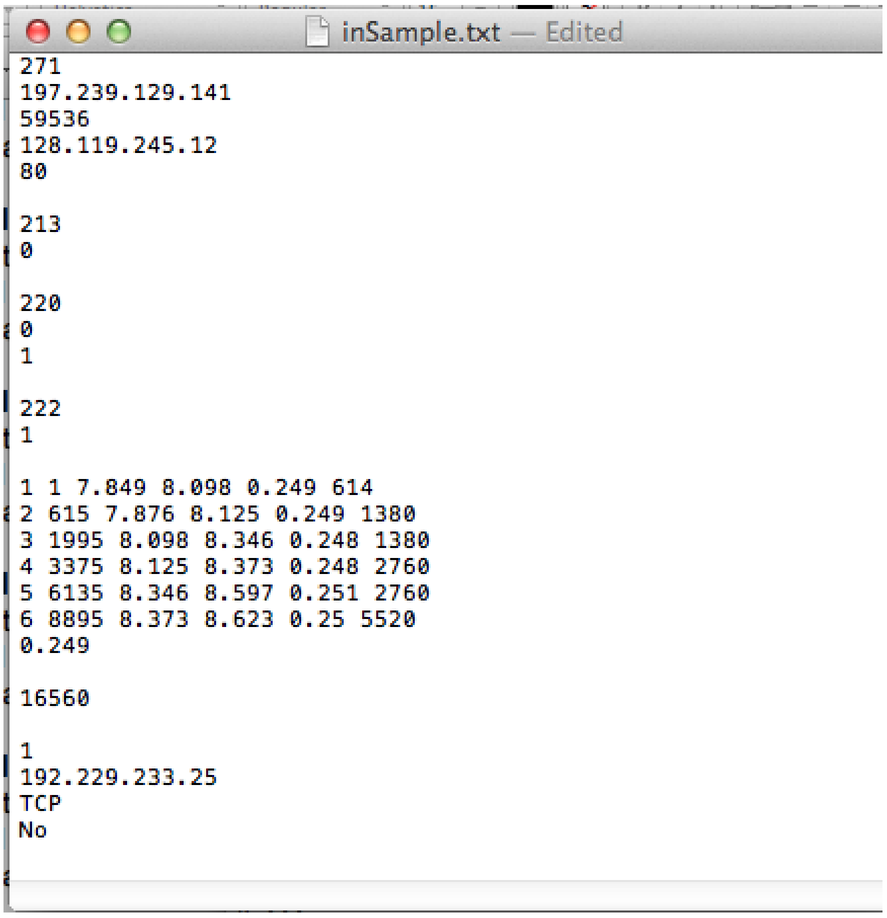
The questions that will be automatically marked have to be typed up in a `txt` file in the following format:



```
#q1,2
<Frame>
<Source IP>
<Source port>
<Destination IP>
<Destination port>

#q3
<Frame>
<Sequence number>

#q4
<Frame>
<Sequence number>

#q5
<Frame>
<Sequence number>

#q6,7
1 <Sequence Number> <Time Sent> <Time Received> <Time Difference> <Packet Length>
2 ...
. ...
6 ...
<RTT>

#q8
<Window Size>

#q10,11,12
<Frame>
<IP Address>
<Protocol>
<Fragmented>
```

Figure 6: Format of the answer file for the questions that will be marked by the automatic marker.

On the next page you can see an example of how your file may look like; this is sample data, so your answers will be different.



inSample.txt — Edited

```
271
197.239.129.141
59536
128.119.245.12
80

213
0

220
0
1

222
1

1 1 7.849 8.098 0.249 614
2 615 7.876 8.125 0.249 1380
3 1995 8.098 8.346 0.248 1380
4 3375 8.125 8.373 0.248 2760
5 6135 8.346 8.597 0.251 2760
6 8895 8.373 8.623 0.25 5520
0.249

16560

1
192.229.233.25
TCP
No
```

Figure 7: Sample file with hypothetical answers (that will be marked by the automatic marker).