ELEN4017: Network Fundamentals
Lab 4

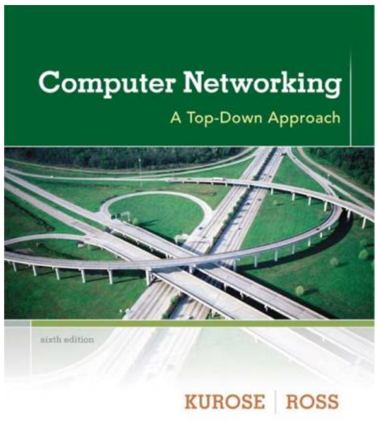
Objective

The objective of this lab is to use the Wireshark tool to analyse features of TCP in the transport layer and IP in the network layer.

Instructions:

- This lab must be completed individually by each student.
- There are two parts of this lab: ‘Lab4a’ about TCP and ‘Lab4b’ about IP.
- The handout of ‘Lab4a’ contains instructions for carrying out the lab and has also got 12 questions. Similarly, handout for ‘Lab4b’ contains lab instructions as well as 15 associated questions.
- An individual lab report, consisting of answers to the questions in the handouts, must be submitted to the demonstrators before the end of the session.
- In addition to the report, demonstrators may ask questions to test your understanding of the material and marks for the lab will be allocated based on your ability to answer the questions.

Lab 4a: TCP in Wireshark



Supplement to *Computer Networking: A Top-Down Approach, 6th 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].

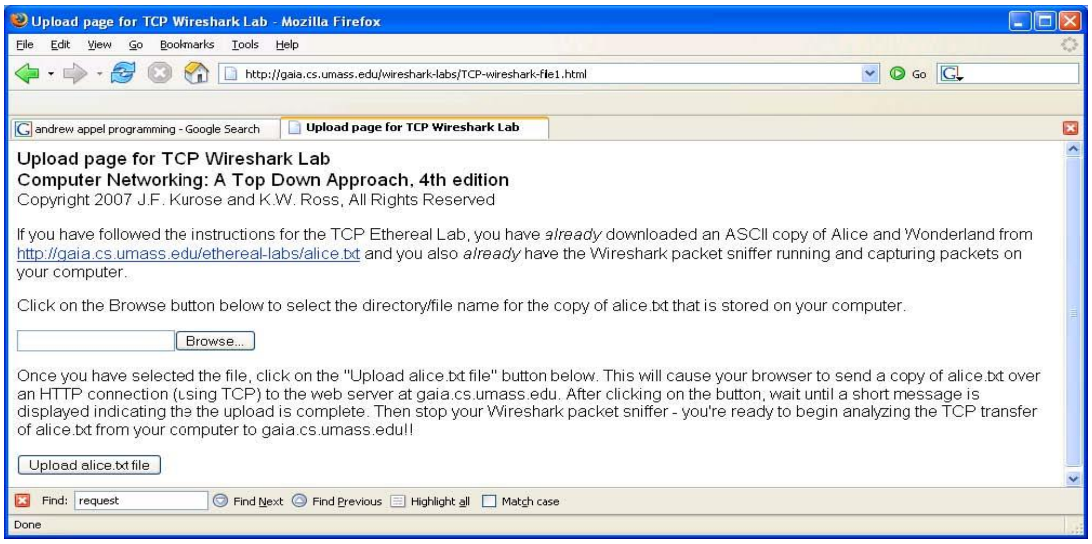
1. Capturing a bulk TCP transfer from your computer to a remote server

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.

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

Do the following:

- Start up your web browser. Go the http://gaia.cs.umass.edu/wiresharklabs/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 the screenshot shown in figure 1:



Upload page for TCP Wireshark Lab - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html
Go
andrew appel programming - Google Search
Upload page for TCP Wireshark Lab
Browse...
Upload alice.txt file
Find: request
Find Next
Find Previous
Highlight all
Match case
Done

Upload page for TCP Wireshark Lab
Computer Networking: A Top Down Approach, 4th edition
Copyright 2007 J.F. Kurose and K.W. Ross, All Rights Reserved

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

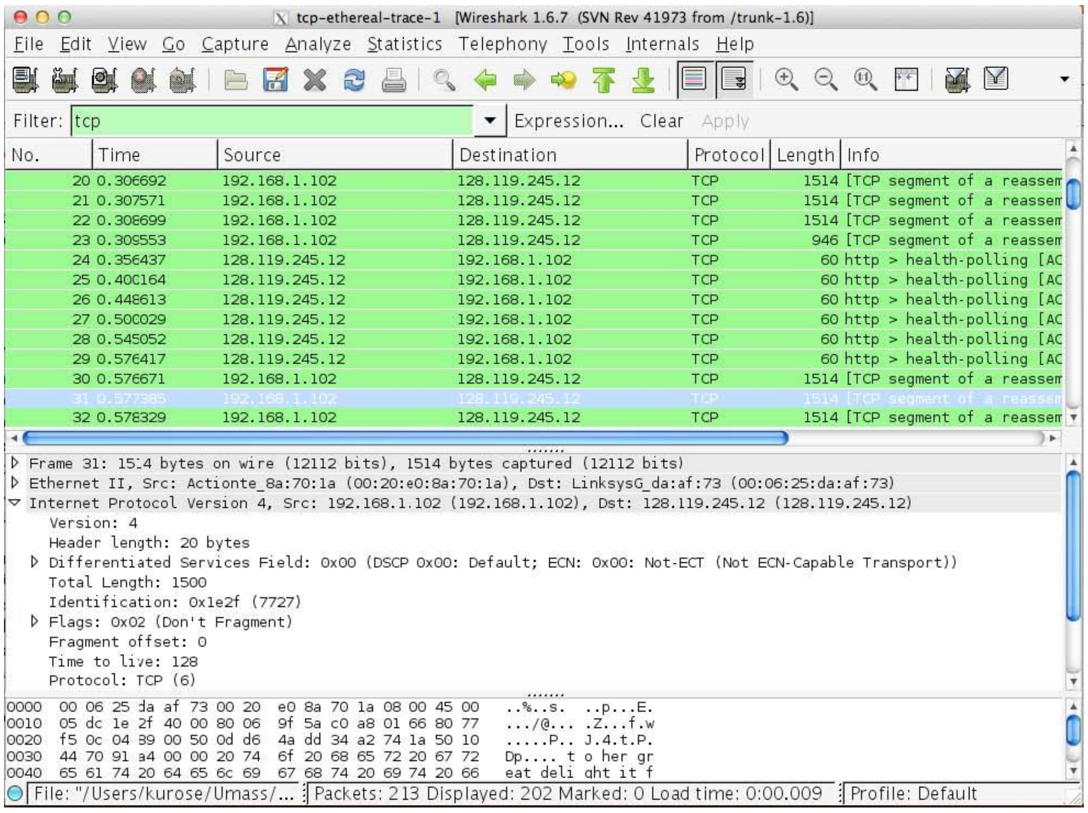
Click on the Browse button below to select the directory/file name for the copy of alice.txt that is stored on your computer.

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 TCP Wireshark Lab

- 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 in figure 2:

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



```
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
   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
```

Figure 2: Wireshark window showing packet capture

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 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 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.

[2] Download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the file tcpethereal-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.

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 contains data that belongs 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.

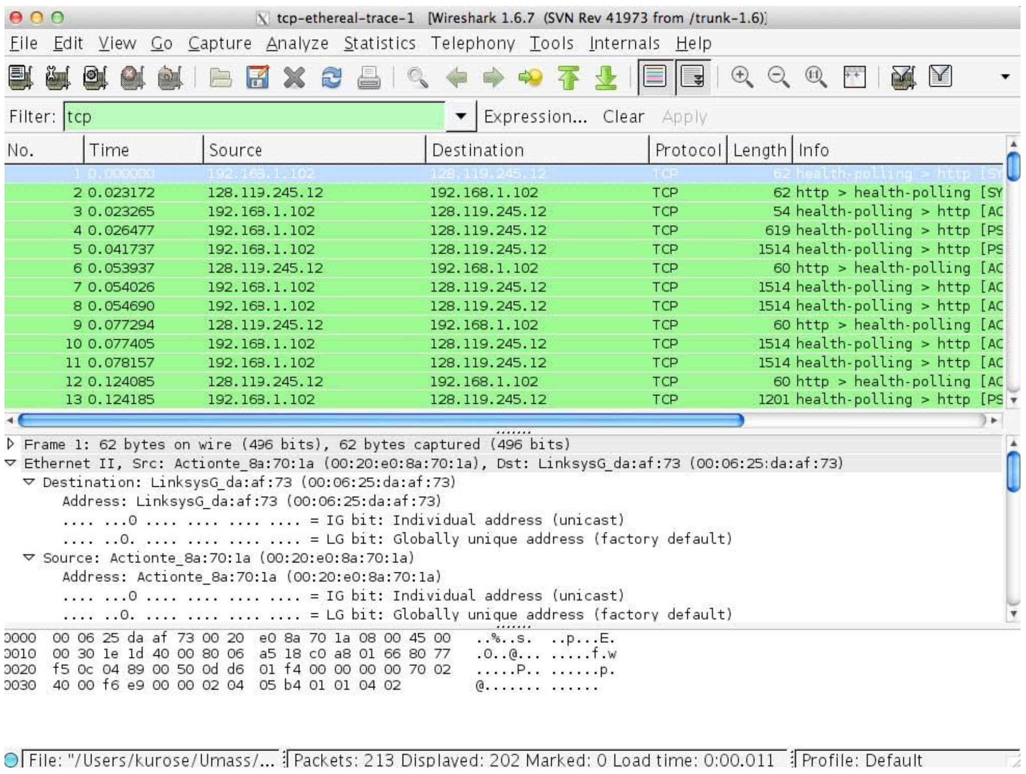
Answer the following questions, by opening the Wireshark captured packet file *tcpethereal-trace-1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip (that is download the trace and open that trace in Wireshark; see footnote 2).

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 the screenshot given in figure 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: Wireshark window with TCP packets only

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/wiresharktraces.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 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 in section 3.5.3 for all subsequent segments.

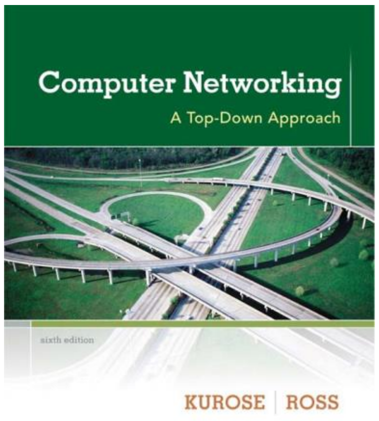
Note: Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the gaia.cs.umass.edu server. Then select: Statistics->TCP Stream Graph->Round Trip Time Graph.

8. What is the length of each of the first six TCP segments? (See the note below)

Note: 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.

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 in the text).
12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

Lab 4b: IP in Wireshark



Supplement to *Computer Networking: A Top-Down Approach, 6th 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 IP protocol, focusing on the IP datagram. We'll do so by analyzing a trace of IP datagrams sent and received by an execution of the `traceroute` program. We'll investigate the various fields in the IP datagram, and study IP fragmentation in detail.

Before beginning this lab, you'll probably want to review sections 1.4.3 in the text[1] and section 3.4 of RFC 2151 [https://www.rfc-editor.org/rfc/rfc2151.txt] 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 [https://www.rfc-editor.org/rfc/rfc791.txt] on hand as well, for a discussion of the IP protocol.

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 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

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

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 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.

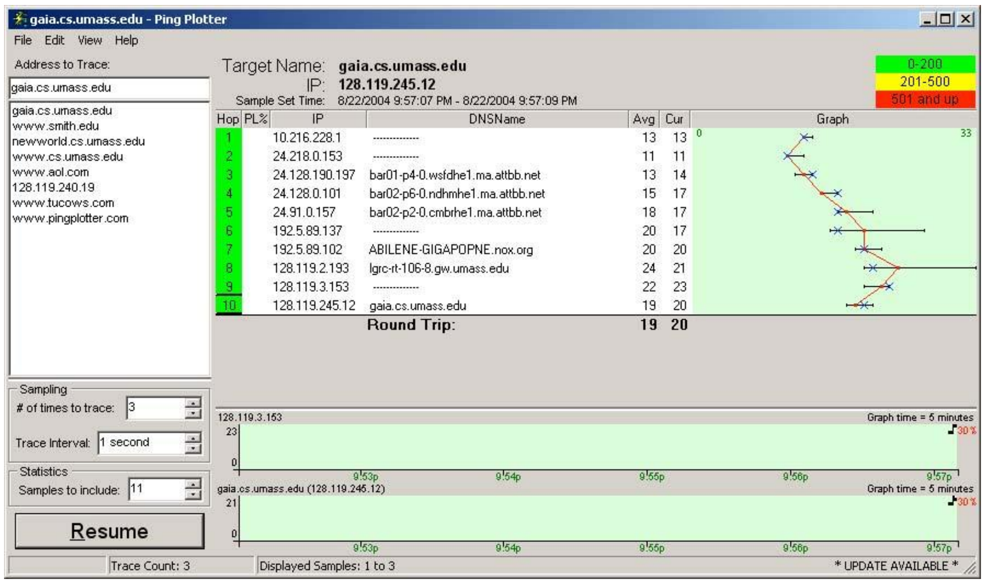
We'll want to run `traceroute` and have it send datagrams of various lengths.

- **Windows.** The `tracert` program provided with Windows does not allow one to change the size of the ICMP echo request (ping) message sent by the `tracert` program. 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*.

- **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'll not need to select any options here).
- If you are using a Windows platform, 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. You should see a *pingplotter* window that looks something like figure 4:



Address to Trace: gaia.cs.umass.edu

gaia.cs.umass.edu
www.smith.edu
newworld.cs.umass.edu
www.cs.umass.edu
www.aol.com
128.119.240.19
www.tucows.com
www.pingplotter.com

Target Name: **gaia.cs.umass.edu**
IP: **128.119.245.12**
Sample Set Time: 8/22/2004 9:57:07 PM - 8/22/2004 9:57:09 PM

Hop	PL%	IP	DNSName	Avg	Cur
1		10.216.228.1	-------------	13	13
2		24.218.0.153	-------------	11	11
3		24.128.190.197	bar01-p4-0.wsfdhe1.ma.attbb.net	13	14
4		24.128.0.101	bar02-p6-0.ndhmhe1.ma.attbb.net	15	17
5		24.91.0.157	bar02-p2-0.cmbrhe1.ma.attbb.net	18	17
6		192.5.89.137	-------------	20	17
7		192.5.89.102	ABILENE-GIGAPOPNE.nox.org	20	20
8		128.119.2.193	lgrc-rt-106-8.gw.umass.edu	24	21
9		128.119.3.153	-------------	22	23
10		128.119.245.12	gaia.cs.umass.edu	19	20
			Round Trip:	**19**	**20**

Sampling
# of times to trace: 3
Trace Interval: 1 second

Statistics
Samples to include: 11

128.119.3.153 Graph time = 5 minutes
gaia.cs.umass.edu (128.119.245.12) Graph time = 5 minutes

Trace Count: 3 Displayed Samples: 1 to 3 * UPDATE AVAILABLE *

Figure 4: Pingplotter window

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.

Finally, send a set of datagrams with a longer length, by selecting *Edit>Advanced Options->Packet Options* and enter a value of 3500 in the *Packet Size* field and then press OK. Then press the Resume button.

Stop Wireshark tracing.

- 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.

  Stop Wireshark tracing.

If you are unable to run Wireshark on a live network connection, you can download a packet trace file that was capture 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 Windows computers2. 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 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.

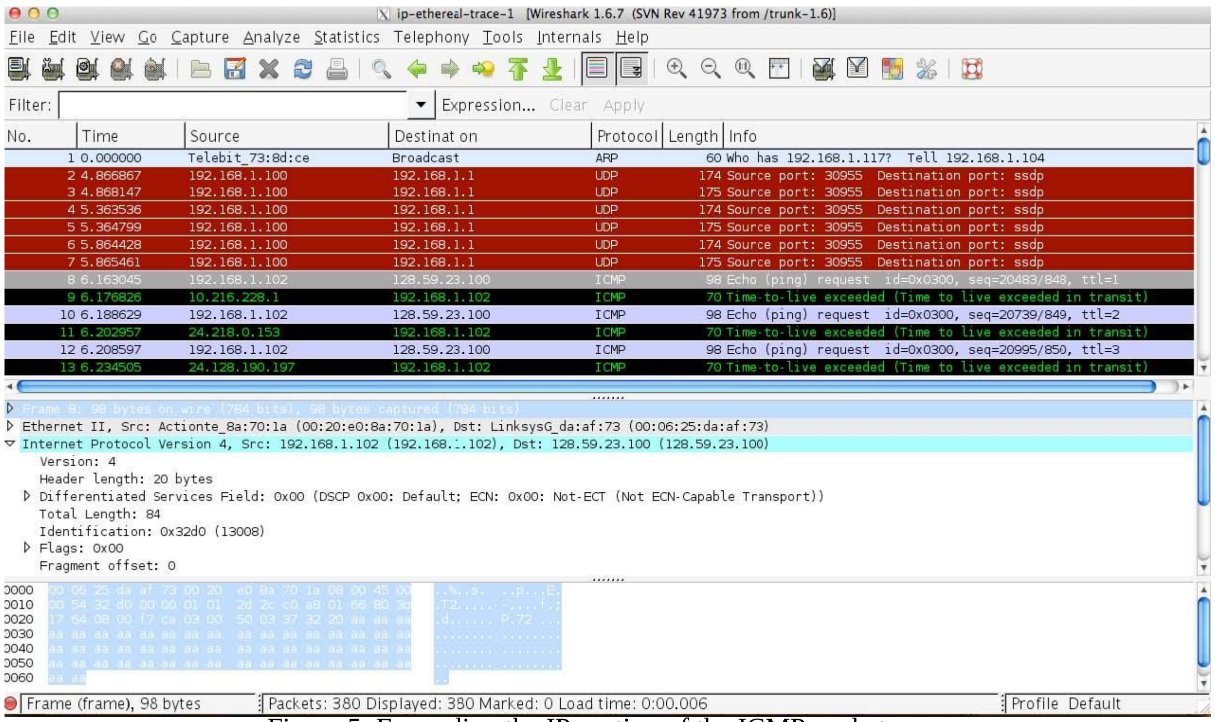
1. Select the first ICMP Echo Request message sent by your computer, and expand the Internet Protocol part of the packet in the packet details window, as shown in figure 5. What is the IP address of your computer?

2. Within the IP packet header, what is the value in the upper layer protocol field?

3. How many bytes are in the IP header? How many bytes are in the payload *of the IP datagram*? Explain how you determined the number of payload bytes.

4. Has this IP datagram been fragmented? Explain how you determined whether or not the datagram has been fragmented.

[2] Download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the file *ipethereal-trace-1*. The traces in this zip file were collected by Wireshark running on one of the book 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 ip-ethereal-trace-1 trace file.


ip-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:
Expression... Clear Apply

```
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: Expanding the IP portion of the ICMP packet

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.

5. Which fields in the IP datagram *always* change from one datagram to the next within this series of ICMP messages sent by your computer?
6. Which fields stay constant? Which of the fields *must* stay constant? Which fields must change? Why?
7. 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 (first hop) router.

8. What is the value in the Identification field and the TTL field?
9. Do these values remain unchanged for all of the ICMP TTL-exceeded replies sent to your computer by the nearest (first hop) router? Why?

Fragmentation

Sort the packet listing according to time again by clicking on the *Time* column.

10. Find the first ICMP Echo Request message that was sent by your computer after you changed the *Packet Size* in *pingplotter* to be 2000. Has that message been fragmented across more than one IP datagram? [Note: if you find your packet has not been fragmented, you should download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the *ipethereal-trace-1* packet trace. If your computer has an Ethernet interface, a packet size of 2000 *should* cause fragmentation.[3]]

11. Observe the first fragment of the fragmented IP datagram. What information in the IP header indicates that the datagram been fragmented? What information in the IP header indicates whether this is the first fragment versus a latter fragment? How long is this IP datagram?

12. Observe the second fragment of the fragmented IP datagram. What information in the IP header indicates that this is not the first datagram fragment? Are the more fragments? How can you tell?

13. What fields change in the IP header between the first and second fragment?

Now find the first ICMP Echo Request message that was sent by your computer after you changed the *Packet Size* in *pingplotter* to be 3500.

14. How many fragments were created from the original datagram?

15. What fields change in the IP header among the fragments?

[3] The packets in the *ip-ethereal-trace-1* trace file in http://gaia.cs.umass.edu/wireshark-labs/wiresharktraces.zip are all less that 1500 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 upper-layer protocol payload). This 1500 byte value is the standard maximum length allowed by Ethernet. If your trace indicates a datagram longer 1500 bytes, and your computer is using an Ethernet connection, then Wireshark is reporting the wrong IP datagram length; it will likely also show only one large IP datagram rather than multiple smaller datagrams.. This inconsistency in reported 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 *ip-ethereal-trace-1* trace file.