Computer Network Lab (2)

This lab will go over three parts:

Part1. Transmission Control Protocol (TCP)

- In this part of the lab, you will be shown the basics of examining TCP packets in Wireshark, and what information you can glean from these packets. You will be asked to answer 12 questions as you work through this section

Part2. User Datagram Protocol (UDP)

- In this part of the lab, you will perform a packet capture of some traffic generated by you, in which several UDP packets will be produced. You will then answer an additional 7 questions based on your trace.

Part3. IPv4 vs IPv6 Traceroute

- In this part of the lab, you will compare paths to the same destination using IPv4 and IPv6 traceroute, and analyze differences in hop count, latency, and ICMP/transport behavior as seen in Wireshark.

What to turn in for this lab:

Each part of the lab has a list of questions regarding your work in that part of the lab. You are required to give a report in **PDF** format including your answers to the questions given in the lab. Include clearly labeled screenshots (or Wireshark export graphs) where requested.

Part 1: TCP

In this lab, we'll investigate the behavior of the 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.

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

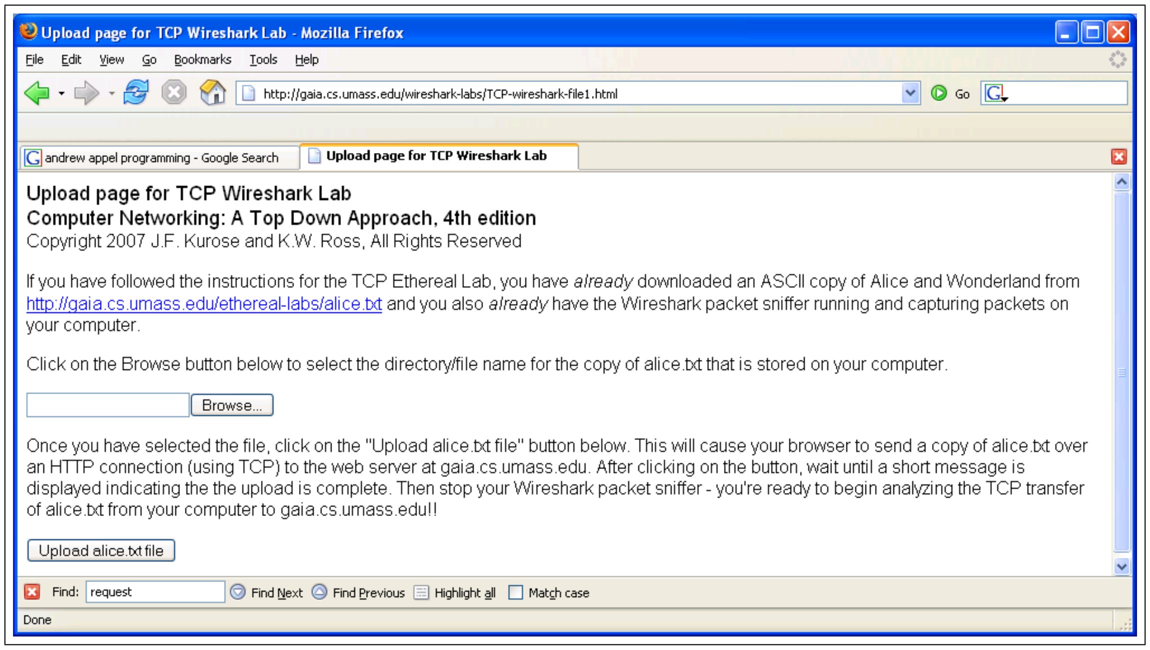
Before beginning our exploration of TCP, we'll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You'll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text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

Computer Networking: a Top-Down Approach, 8th Edition, Kurose and Ross.

POST method. 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 to this link and retrieve an ASCII copy of *Alice in Wonderland*. Store this file somewhere on your computer and name it "alice.txt".
- Next go to this link. You should see the following screen:


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

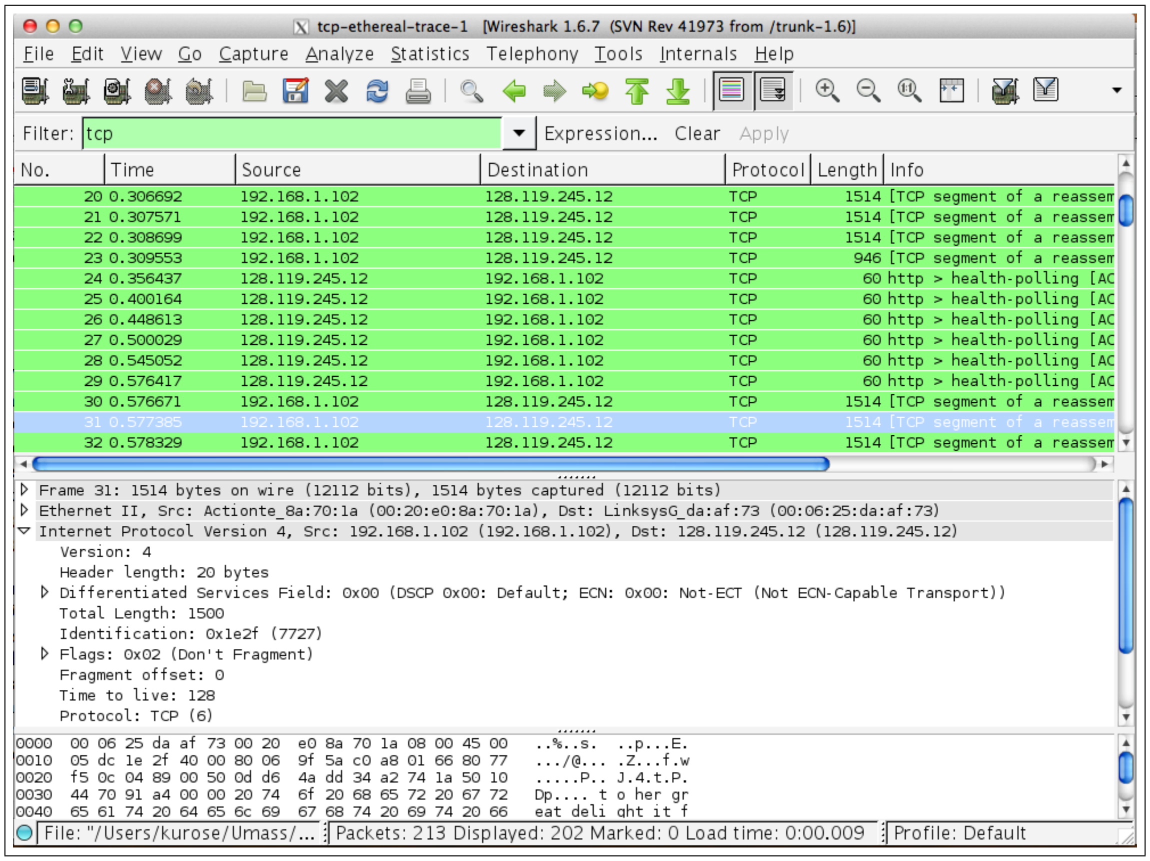
Upload page for TCP Wireshark Lab
Computer Networking: A Top Down Approach, 4th edition
Copyright 2007 J.F. Kurose and K.W. Ross, All Rights Reserved

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!!

- Use the Browse button in this form to enter the name of the file (full path name) on your computer containing Alice in Wonderland (or do so manually). Don't yet press the "Upload alice.txt file" button.
- Now start up Wireshark and begin packet capture (Capture → Start) and then press OK on the Wireshark Packet Capture Options screen (we'll not need to select any options here).
- Returning to your browser, press the "Upload alice.txt file" button to upload the file to the gaia.cs.umass.edu server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
- Stop Wireshark packet capture. Your Wireshark window should look similar to the window shown below.


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

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.

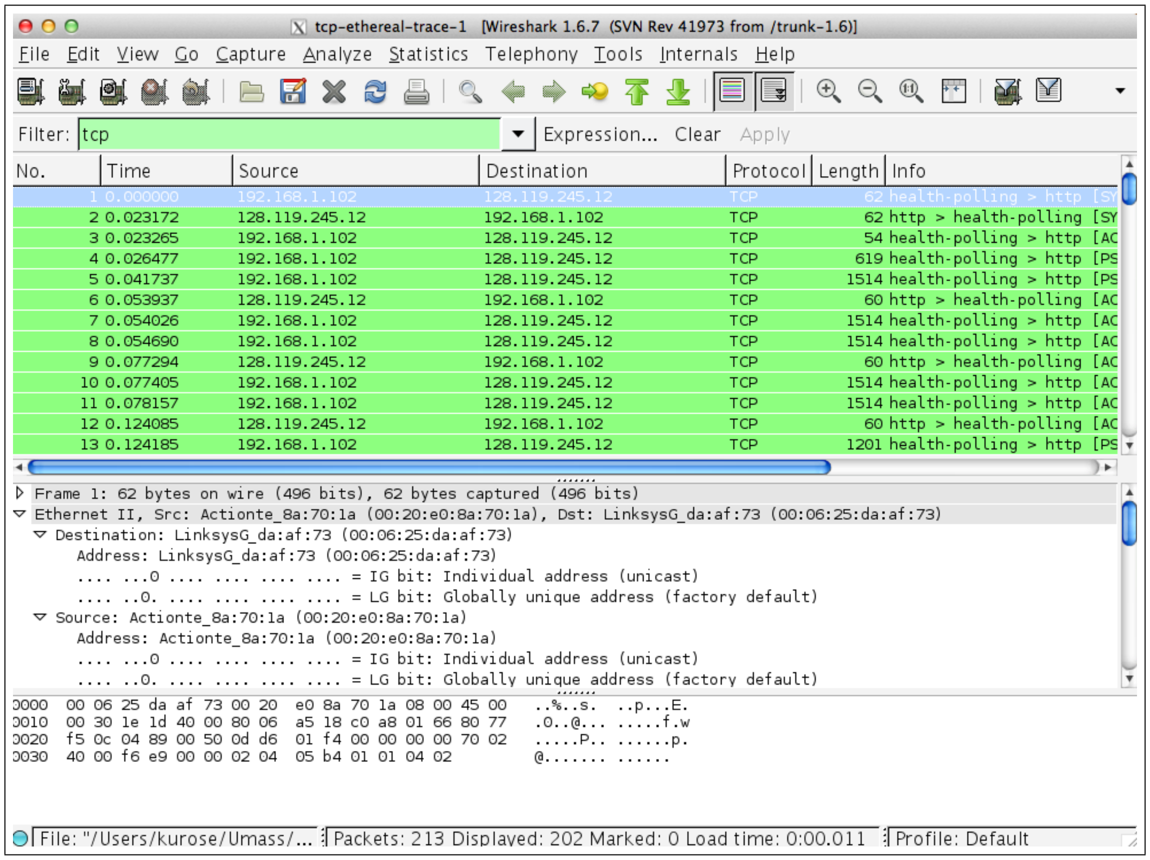
What you should see is series of TCP and HTTP messages between your computer and gaia.cs.umass.edu. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message. Depending on the version of Wireshark you are using, you might see a series of "HTTP Continuation" messages being sent from your computer to gaia.cs.umass.edu. Recall from our discussion in the earlier HTTP Wireshark lab, that is no such thing as an HTTP Continuation message – this is Wireshark's way of indicating that there are multiple TCP segments being used to carry a single HTTP message. In more recent versions of Wireshark, you'll see "[TCP segment of a reassembled PDU]" in the Info column of the Wireshark display to indicate that this TCP segment contained data that belonged to an upper layer protocol message (in our case here, HTTP). You should also see TCP ACK segments being returned from gaia.cs.umass.edu to your computer.

Answer the following questions:

1. What is the IP address and TCP port number used by your client computer (source) to transfer 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"

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?

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:


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
```

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 behavior in the rest of this lab.

TCP Basics

Answer the following questions for the TCP segments:

1. 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?

2. 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 Acknowledgment 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?

3. 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.

4. 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 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 given in the lecture for all subsequent segments.

5. What is the length of each of the first six TCP segments?

6. 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?

7. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

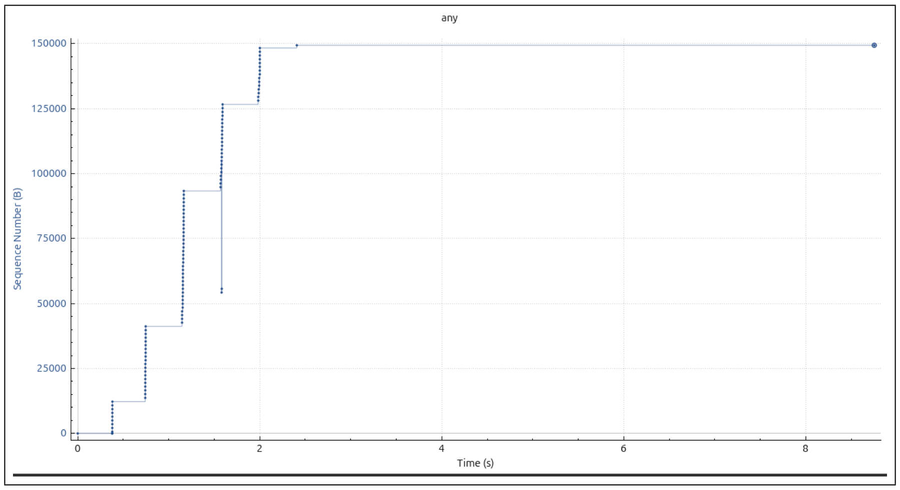
8. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment.

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

TCP congestion control in action

Let's now examine the amount of data sent per unit time from the client to the server. Rather than (tediously!) calculating this from the raw data in the Wireshark window, we'll use one of Wireshark's TCP graphing utilities - Time-Sequence-Graph - to plot out data.

- Select a TCP segment in the Wireshark's "listing of captured-packets" window. Then select the menu : Statistics $\rightarrow$ TCP Stream Graph $\rightarrow$ Time-Sequence-Graph(Stevens). You should see a plot that looks similar to the following plot:



- Here, each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stacked above each other represents a series of packets that were sent back-to-back by the sender.

Answer the following questions for the TCP segments from your packet trace:

Use the Time-Sequence-Graph(Stevens) plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Comment on ways in which the measured data differs from the idealized behavior of TCP that we've studied in the text.

Part 2: UDP

In this lab, we'll take a quick look at the UDP transport protocol. As we saw in Chapter 3 of the text, UDP is a streamlined, no-frills protocol. Because UDP is simple and sweet, we'll be able to cover it pretty quickly in this lab.

At this stage, you should be a Wireshark expert. Thus, we are not going to spell out the steps as explicitly as in earlier part. In particular, we are not going to provide example screenshots for all the steps.

The Assignment

Do the following:

Start capturing packets in Wireshark and then do something that will cause your host to send and receive several UDP packets. It's also likely that just by doing nothing (except capturing packets via Wireshark) that some UDP packets sent by others will appear in your trace. In particular, the Simple Network Management Protocol (SNMP – see section 5.7 in the text) sends SNMP messages inside of UDP, so it's likely that you'll find some SNMP messages (and therefore UDP packets) in your trace.

After stopping packet capture, set your packet filter so that Wireshark only displays the UDP packets sent and received at your host. Pick one of these UDP packets and expand the UDP fields in the details window

Answer the following questions based on your trace:

1. Select one UDP packet from your trace. From this packet, determine how many fields there are in the UDP header. (You shouldn't look in the textbook! Answer these questions directly from what you observe in the packet trace.) Name these fields.
2. By consulting the displayed information in Wireshark's packet content field for this packet, determine the length (in bytes) of each of the UDP header fields.
3. The value in the Length field is the length of what? (You can consult the text for this answer). Verify your claim with your captured UDP packet.
4. What is the maximum number of bytes that can be included in a UDP payload? (Hint: the answer to this question can be determined by your answer to 2. above)
5. What is the largest possible source port number? (Hint: see the hint in 4.)
6. What is the protocol number for UDP? Give your answer in both hexadecimal and decimal notation. To answer this question, you'll need to look into the Protocol field of the IP datagram containing this UDP segment.
7. Examine a pair of UDP packets in which your host sends the first UDP packet and the second UDP packet is a reply to this first UDP packet. (Hint: for a second packet to be sent in response to a first packet, the sender of the first packet should be the destination of the second packet). Describe the relationship between the port numbers in the two packets.

Part 3: IPv4 vs IPv6 Traceroute

In this part, you will compare the network path to the same destination over IPv4 and IPv6 and analyze the differences in behavior using Wireshark.

Setup and Capture

- Pick a well-connected destination that supports both IPv4 and IPv6 (e.g., `www.google.com`, `www.cloudflare.com`, or your university website). If unsure, you can also use a known dual-stack hostname such as `ipv6.google.com`.
- Start Wireshark capture. Optionally set a display filter to focus on traceroute-related traffic: `icmp or icmpv6 or udp.port == 33434`.

- Run traceroute twice to the same destination:
  - Linux/macOS: `traceroute -4 <hostname>` and `traceroute -6 <hostname>`
  - Windows: `tracert -4 <hostname>` and `tracert -6 <hostname>`
- Stop the capture after both runs complete.

Background Notes

When you run `traceroute`, the program does not directly ask routers along the way to identify themselves. Instead, it sends out a series of special probe packets and relies on the routers' automatic responses. In the classic version (on Linux or macOS), `traceroute` sends UDP packets with gradually increasing TTL (IPv4) or Hop Limit (IPv6) values, usually starting at destination port 33434. Each router that forwards the packet decreases the TTL/Hop Limit by one. When the value reaches zero, that router discards the packet and replies with an ICMP *Time Exceeded* message, revealing its own address to you.

Other variants of `traceroute` use different probe types. For example, some versions send ICMP Echo requests (similar to `ping`), or even TCP SYN packets to mimic normal connection attempts. On Windows, the built-in `tracert` command uses ICMP Echo probes by default. No matter which method is used, the idea is the same: collect the ICMP responses from each hop along the path to map out the route and measure the delays at each step.

Answer the following questions:

1. Write down the exact command(s) you used for both IPv4 and IPv6 traceroute. In your Wireshark capture, check what type of probe was actually sent (UDP or ICMP Echo) by looking at the protocol and port/type fields. Include a screenshot showing one probe packet and the matching ICMP *Time Exceeded* reply for both IPv4 and IPv6..

2. How many hops did it take to reach the destination using IPv4, and how many using IPv6? Are they the same length, or is one path longer? If different, by how many hops?

3. For each hop number h that appears in *both* IPv4 and IPv6 runs, record the average RTT reported by traceroute. Find two hops where the IPv4 and IPv6 RTTs differ noticeably (more than about 5 ms). Suggest a possible explanation for the difference (e.g., different routing paths, ICMP rate-limiting, or asymmetric return routes).

4. Open the IP headers in Wireshark for one probe in each family. Report the following values:
   - For IPv4: the initial TTL value used by the probe; any DSCP/ECN values if present.
   - For IPv6: the initial Hop Limit used; the Traffic Class and Flow Label values.

   Briefly explain why TTL (IPv4) and Hop Limit (IPv6) are conceptually the same field, just with different names.

5. If your traceroute output shows any $\star$ entries or missing hop addresses, explain why. Use your packet capture to support your answer (possible reasons include routers filtering ICMP, rate limiting responses, or lack of reverse DNS entries).

6. Compare the final destination's responses in IPv4 and IPv6. Does the reply come from the same site, or possibly a different anycast location? Is the RTT similar or different? Summarize your findings and include one screenshot for the last hop in each run.

End of Lab (2)