Kate Moreland

kem0149

COMP 4320: Lab 2

Computer Network Lab (2)

This lab will go over two 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 serveral UDP packets will be produced. You will then answer an additional 7 questions based on your trace.

What to turn in for this lab:

1. 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 answer to the questions given in the lab.
2. In addition to your responses, you must upload an unlisted video to YouTube (duration up to 15 minutes) for each part of your demo. This video should showcase your lab skills and demonstrate your understanding of the material by explaining your answers to the lab questions. Please include the YouTube link in your report.

# 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

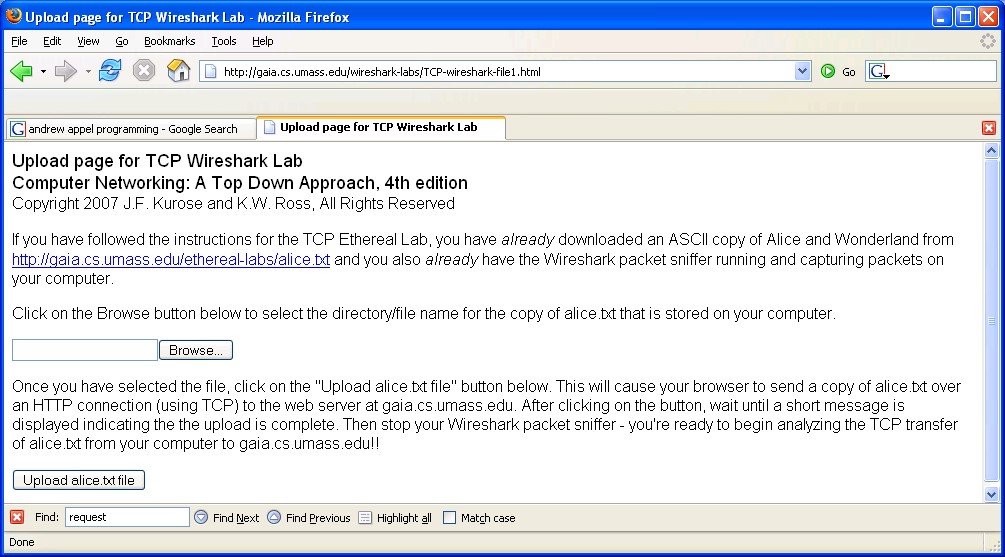
Before beginning our exploration of TCP, we’ll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You’ll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text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

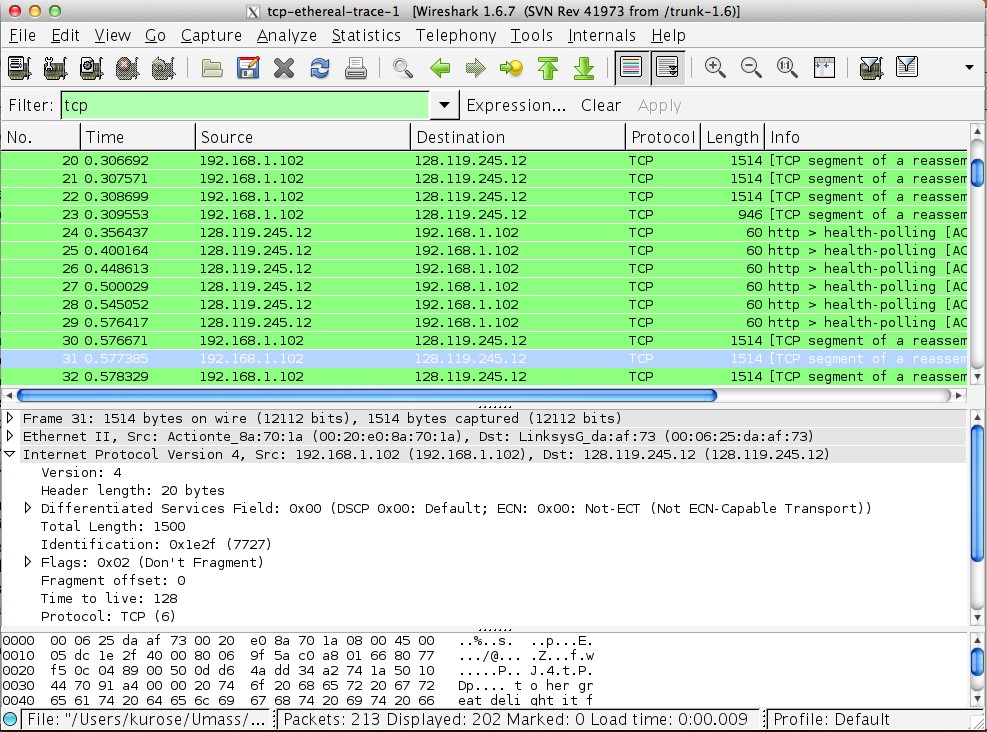
POST method. We’re using the POST method rather than the GET method as we’d like to transfer a large amount of data from your computer to another computer. Of course, we’ll be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer.

Do the following:

* Start up your web browser. Go to [this link](http://gaia.cs.umass.edu/wireshark-labs/alice.txt) and retrieve an ASCII copy of *Alice in Wonderland*. Store this file somewhere on your computer and name it “alice.txt”.
* Next go to [this link.](http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html) You should see the following screen:



* 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.



## 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.

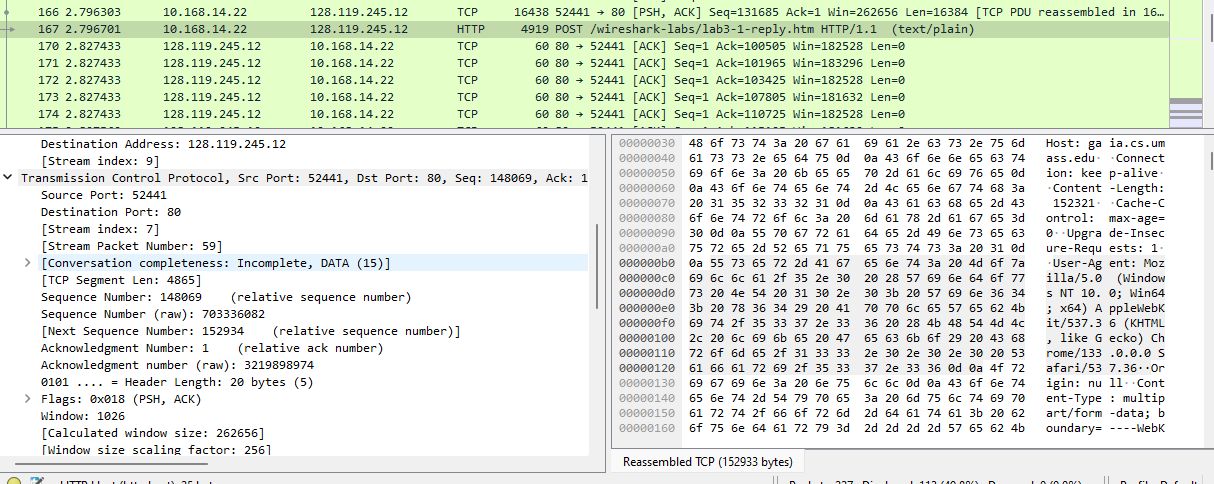
• First, filter the packets displayed in the Wireshark window by entering “tcp” (lowercase, no quotes, and don’t forget to press return after entering!) into the display filter specification window towards the top of the Wireshark window.

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:

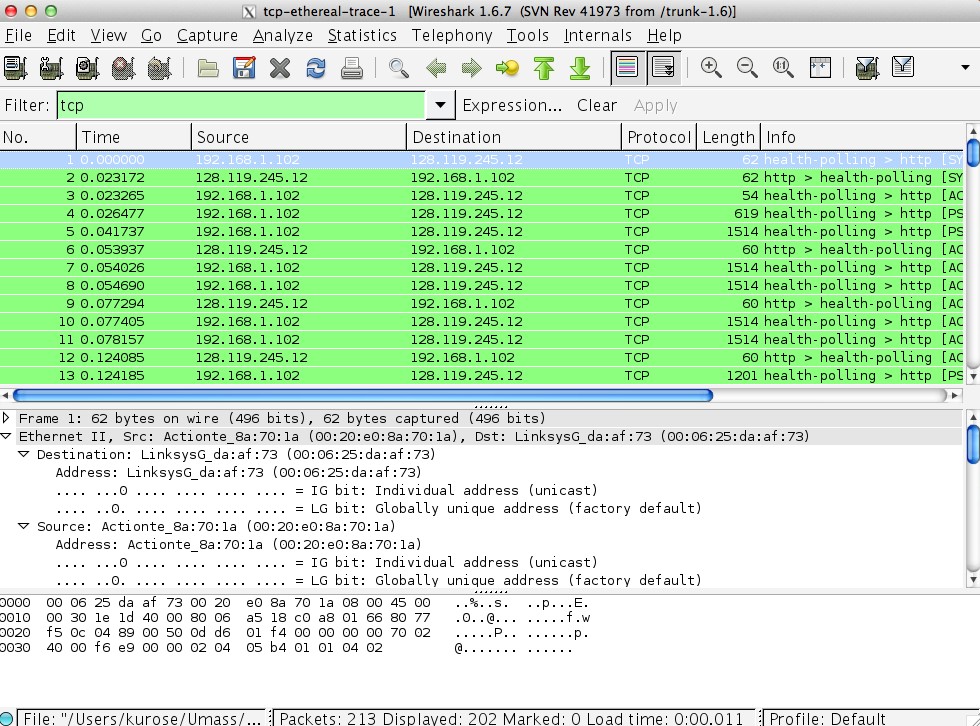
1. What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu? To answer this question, it’s probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the “details of the selected packet header window”

The IP address used by the source to transport the file is 10.168.14.22. The TCP port number is 52441.

1. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?

The IP address of gaia.cs.umass.edu is 128.119.245.12. The port number for gaia.cs.umass.edu is 80.

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:



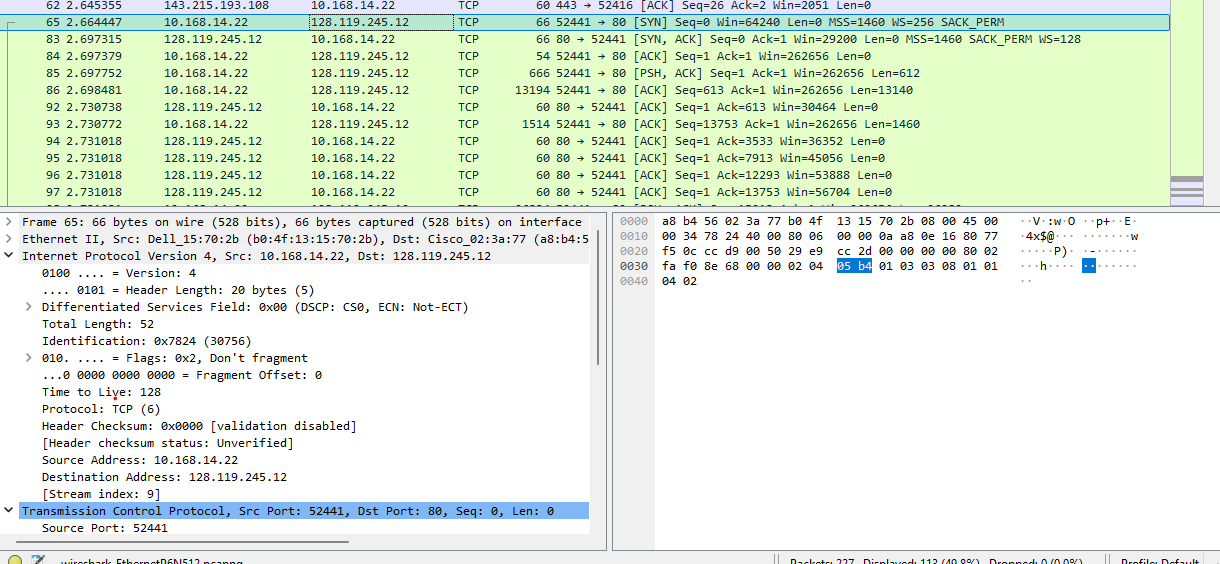
This is what we’re looking for - a series of TCP segments sent between your computer and gaia.cs.umass.edu. We will use the packet trace that you have captured to study TCP behavior in the rest of this lab.

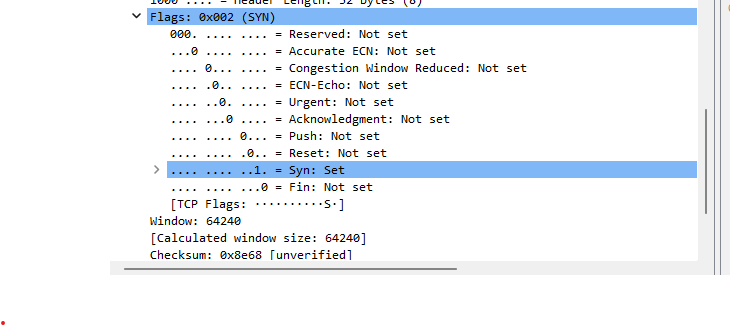
## TCP Basics

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?

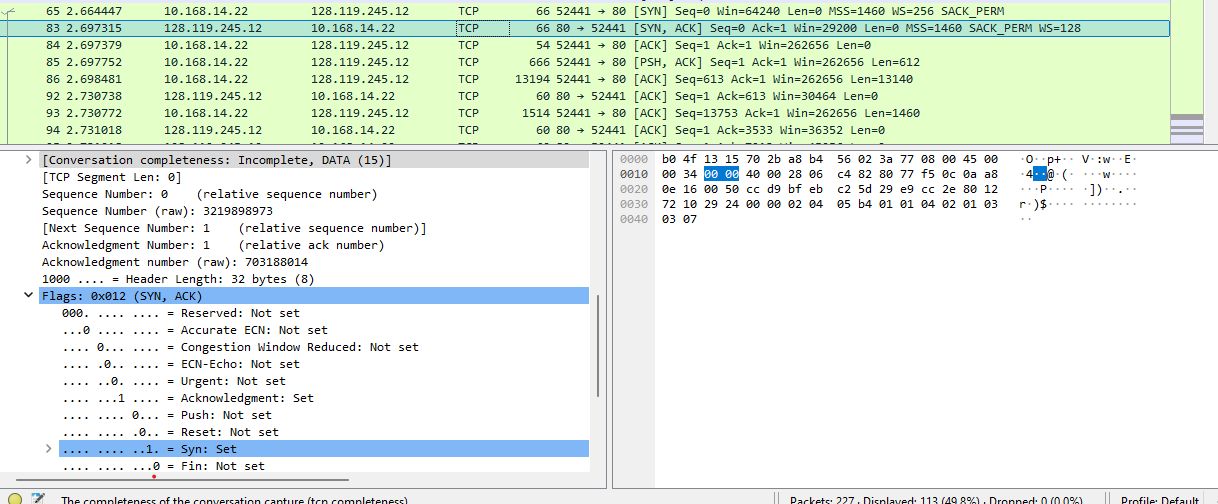
The TCP SYN segment number is 0 in the relative sequence but the raw is 703188013. The thing that establishes it as a SYN segment is the flag that is raised by setting it to 1.





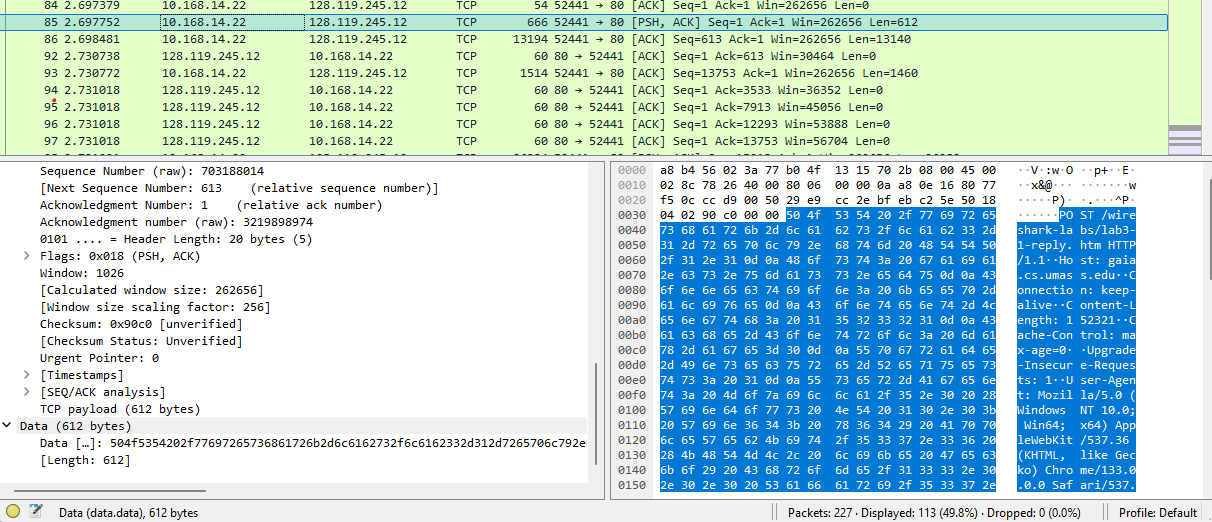
1. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the Acknowledgment field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?

The sequence number is 0 relative but the raw is 3219898973. The acknowledgment number is 1 but the raw number is 703188014. It received the packet that was sent so it is sending back an ACK of the next packet it should receive. The segment identifies the segment as a SYNACK segment by raising both the SYN and ACK field.



1. 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.

The sequence number is 1 relative but the raw is 703188014.



1. 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.

1st – The sequence number is 1. It was sent at 2.697752. It was received at 2.697752. The RTT is 0s. The EstimatedRTT is 2.697752s.

2nd – The sequence number is 613. It was sent at 2.698481. It was received at 2.730738. The RTT is 0.032257s. The EstimatedRTT is 2.362989s.

3rd – The sequence number is 13753. It was sent at 2.730772. It was received at 2.731018. The RTT is 0.000246s. The EstimatedRTT is 2.066646s

4th – The sequence number is 15213. It was sent at 2.731031. It was received at 2.762895. The RTT is 0.031864s. The EstimatedRTT is 1.815804.

5th – The sequence number is 41493. It was sent at 2.763010. It was received at 2.763102. The RTT is 0.000092s. The EstimatedRTT is 1.591769s.

6th – The sequence number is 53173. It was sent at 2.763131. It was received at 2.763201. The RTT is 0.00007s. The EstimatedRTT is 1.392928s.

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

1st - 612

2nd - 13140

3rd - 1460

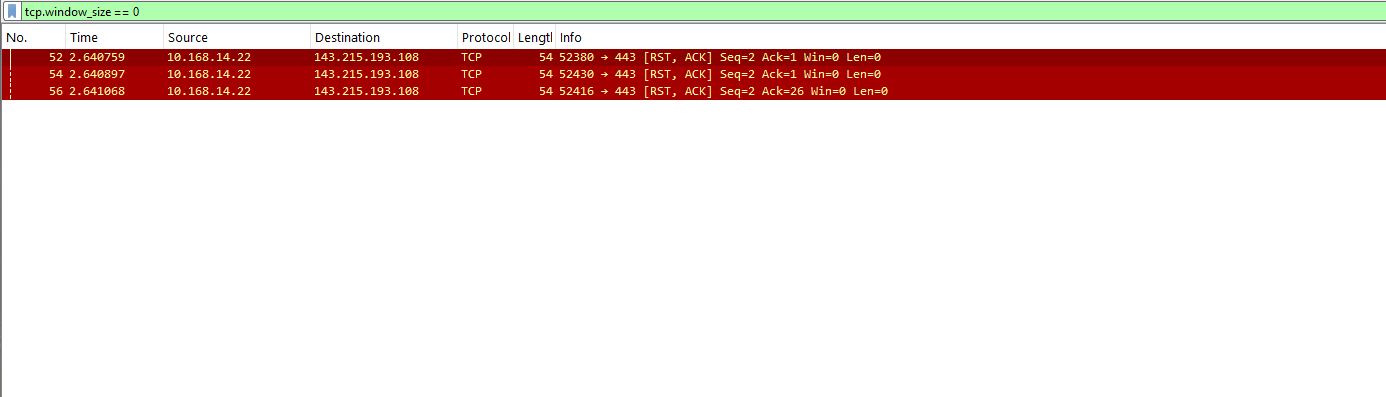
4th - 26280

5th - 11680

6th - 5840

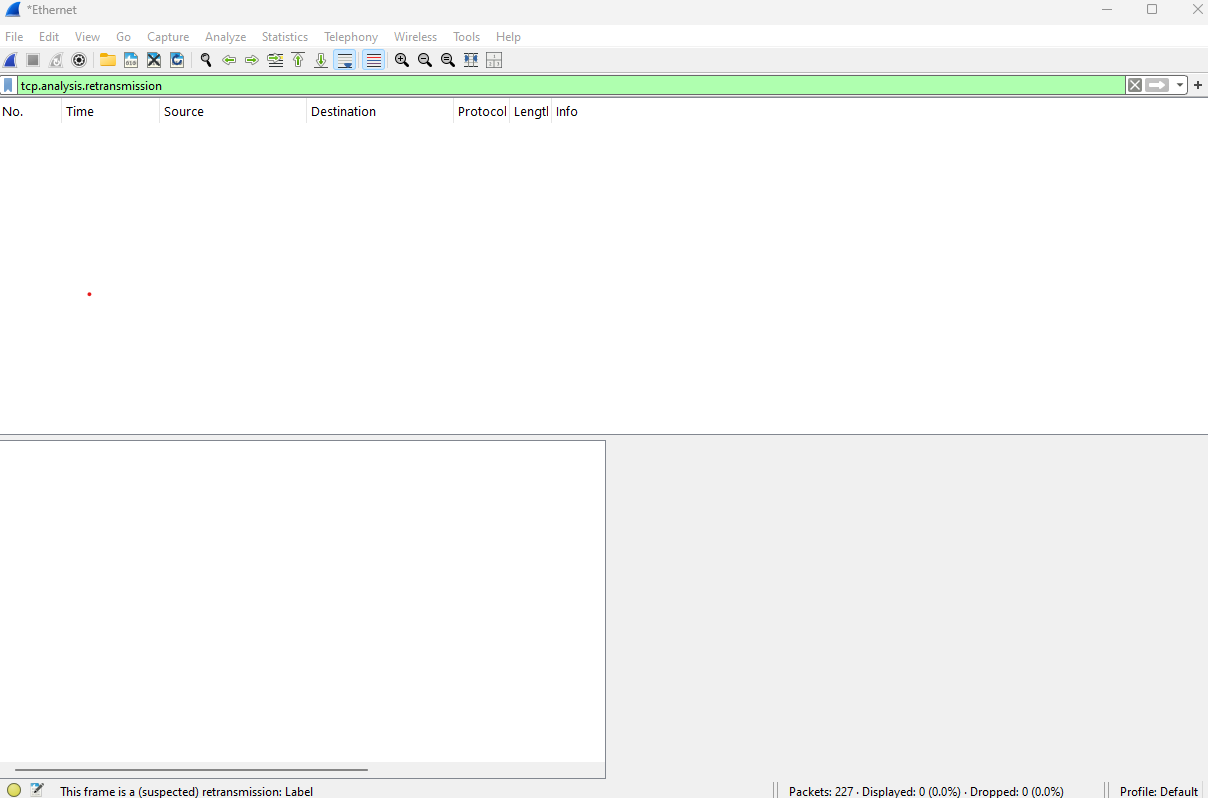
1. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?

The calculated window size is 262656. Yes, it does. I can tell this by looking through the filter tcp.window\_size == 0.



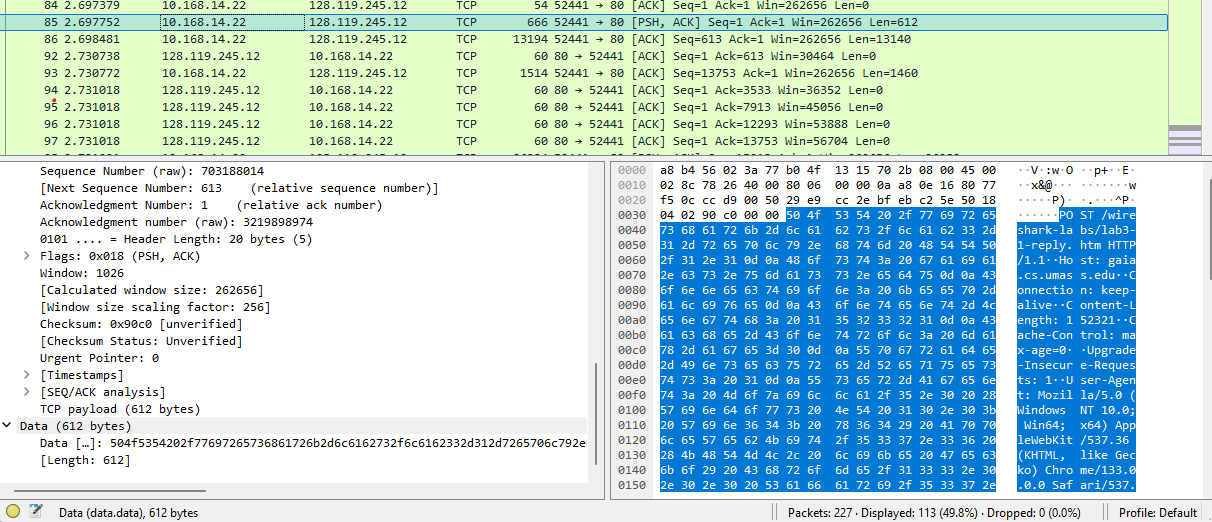
1. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

There are no retransmitted segments in the trace file. I typed in tcp.analysis.retransmission in the filter.



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

The receiver typically acknowledges around 365 bytes in each ACK. There are cases such as when 1460 is sent over multiple ACKs.



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

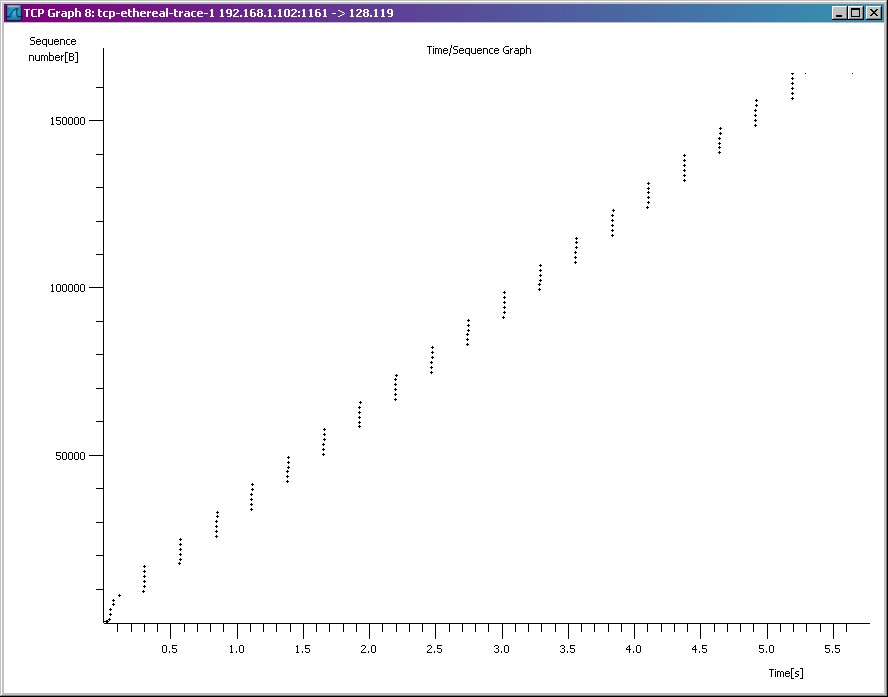
Throughput = total bytes transferred / seconds

Throughput = 160468 / (4.412013 - 0.077527) = 37021.2292761 bytes/sec

## TCP congestion control in action

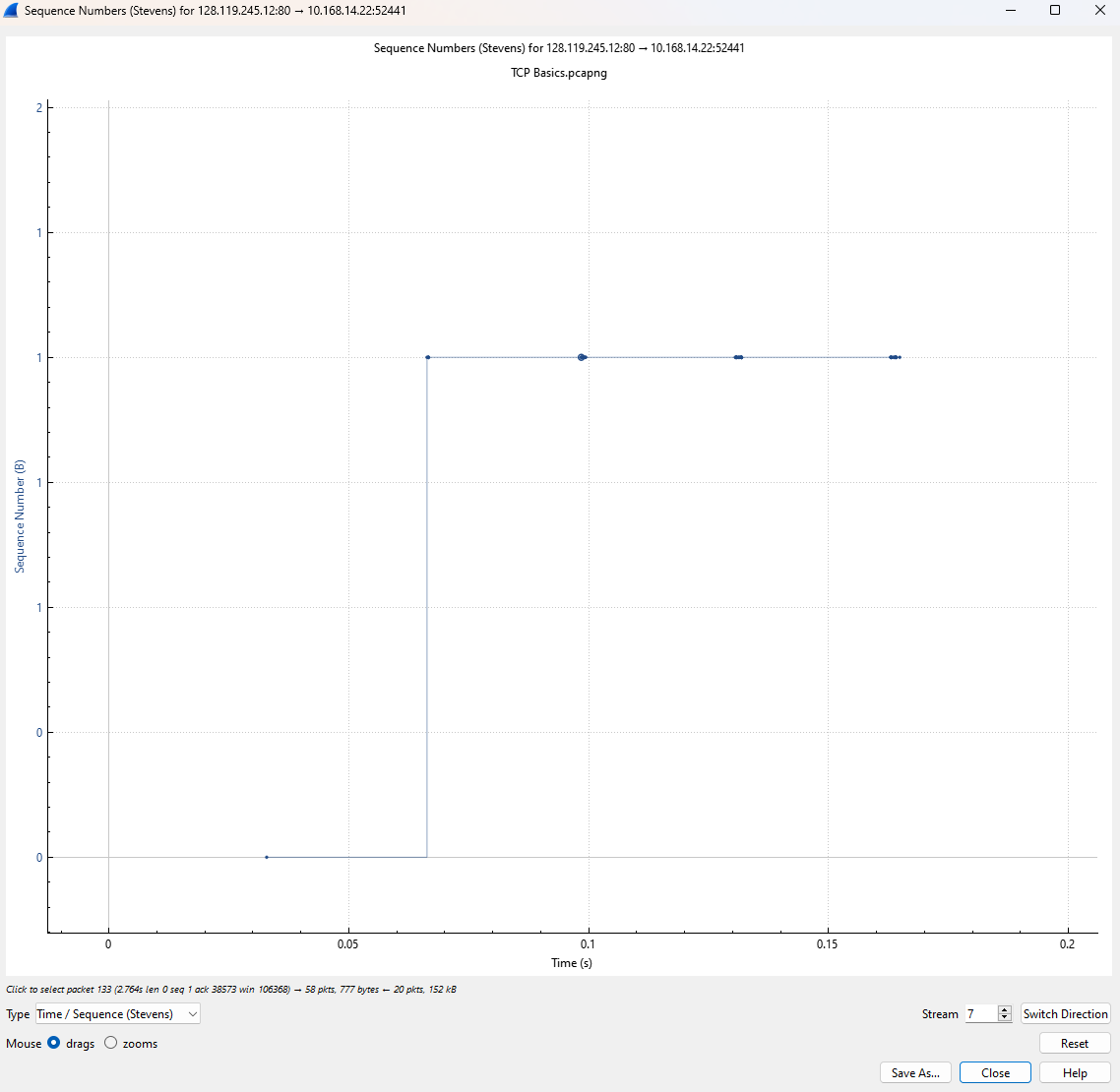
Let’s now examine the amount of data sent per unit time from the client to the server. Rather than (tediously!) calculating this from the raw data in the Wireshark window, we’ll use one of Wireshark’s TCP graphing utilities - Time-Sequence-Graph - to plot out data.

* Select a TCP segment in the Wireshark’s “listing of captured-packets” window. Then select the menu : Statistics → TCP Stream Graph → 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.

The slowstart begins around 0.65s and doesn’t go up more than that. This is different from the ideal model because it is not constant like the ones in class.

# 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:

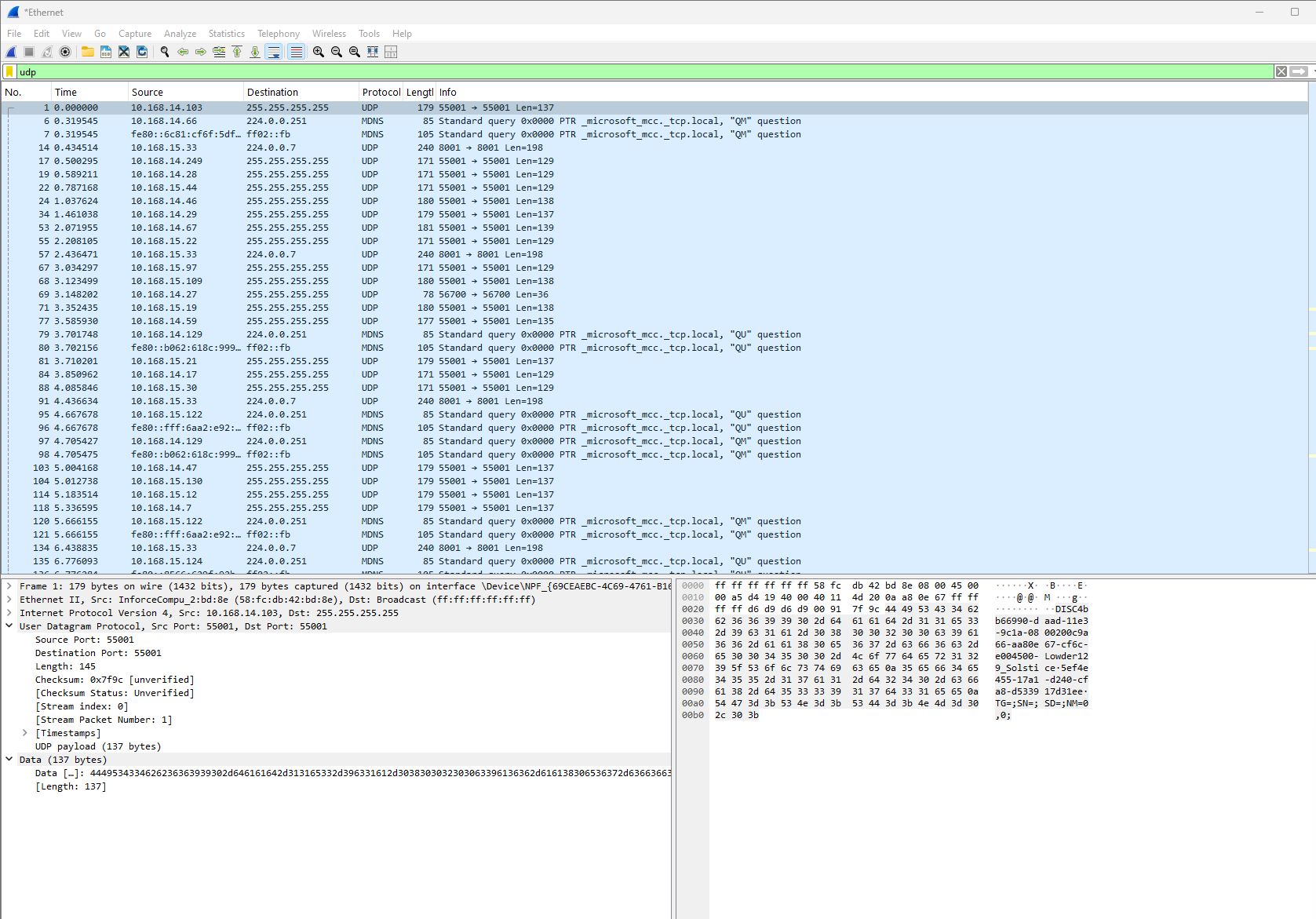
Start capturing packets in Wireshark and then do something that will cause your host to send and receive several UDP packets. It’s also likely that just by doing nothing (except capturing packets via Wireshark) that some UDP packets sent by others will appear in your trace. In particular, the Simple 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.

There are 4 fields. They are source port, destination port, length, and checksum.

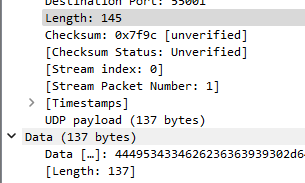


1. By consulting the displayed information in Wireshark’s packet content field for this packet, determine the length (in bytes) of each of the UDP header fields.

The length of packet for source port is 2 bytes. The length of packet destination port is 2 bytes. The length of length is 2 bytes. The length of checksum is 2 bytes.

1. 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.

The value of the length field is the length of the UDP and the data combined in bytes. I can tell this by adding the 8 bytes of the header and the 137 bytes of the data together to get 145.



1. 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)

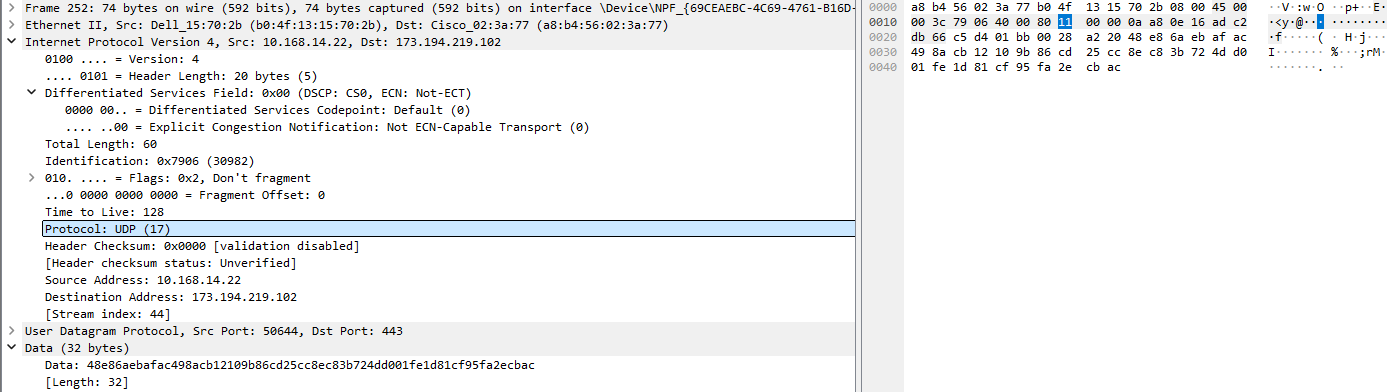
The payload is the length – the length of the header. Payload = length - 8

1. What is the largest possible source port number? (Hint: see the hint in 4.)

The largest possible source port number is 2 bytes so it is 65535 in decimal.

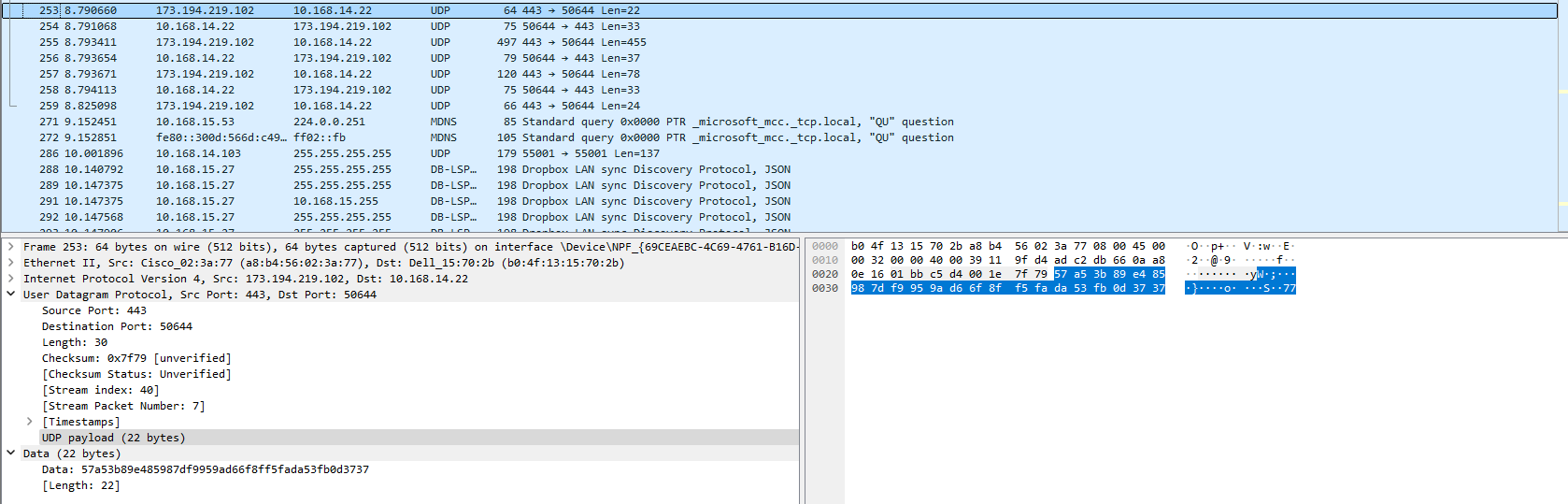
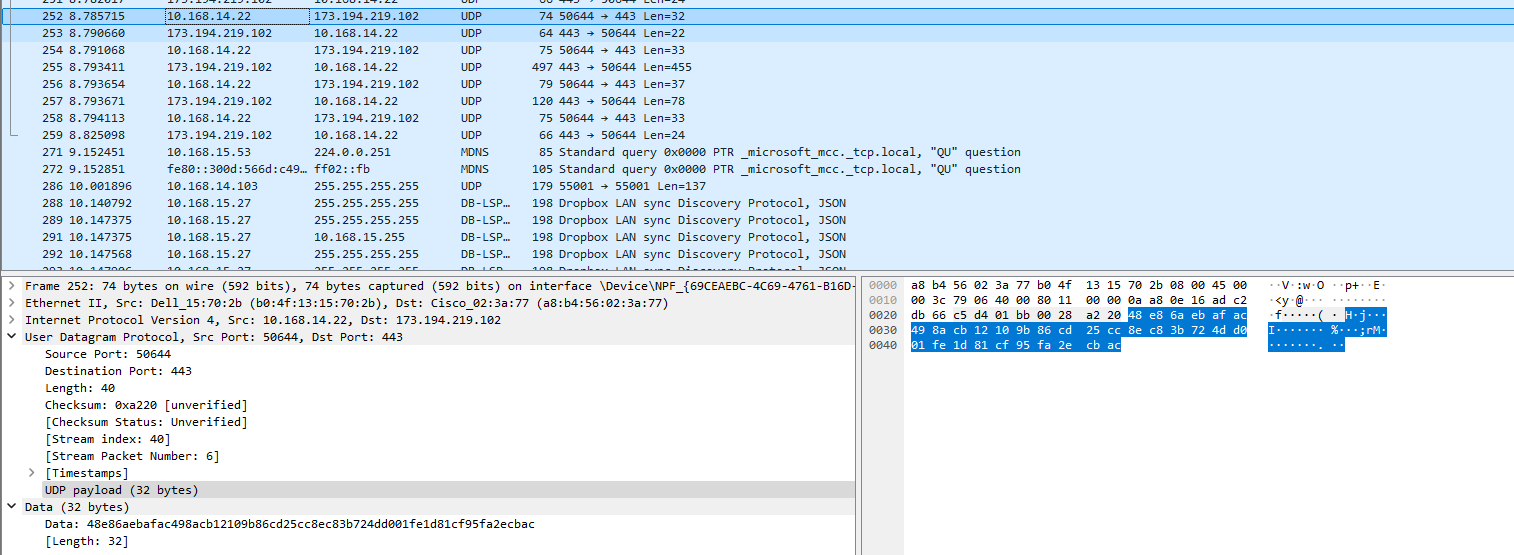
1. What is the protocol number for UDP? Give your answer in both hexadecimal and decimal notation. To answer this question, you’ll need to look into the Protocol field of the IP datagram containing this UDP segment.

The protocol number for UDP is 17 in decimal and 11 in hexadecimal.



1. 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.

In the first packet sent the source port is 50644 and the destination port is 443. In the second packet sent the sourse port is 443 and the destination port is 50644.



End of Lab (2)

YouTube Link:

<https://youtu.be/6v80IkcG84c>