

Lab-07-A

Aim: WireShark Lab: Getting Started

Objective: Getting acquainted with Wireshark, and make some simple packet captures and observations.

❖ Getting Wireshark

In order to run Wireshark, you'll need to have access to a computer that supports both Wireshark and the *libpcap* or *WinPCap* packet capture library. The *libpcap* software will be installed when you install Wireshark.

See <http://www.wireshark.org/download.html> for a list of supported operating systems and download sites.

Download and install the Wireshark software:

- Go to <http://www.wireshark.org/download.html> and download and install the Wireshark binary for your computer.

❖ Running Wireshark

When you run the Wireshark program, you'll get a startup screen that looks something like the screen below. Different versions of Wireshark will have different startup screens – so don't panic if yours doesn't look exactly like the screen below!

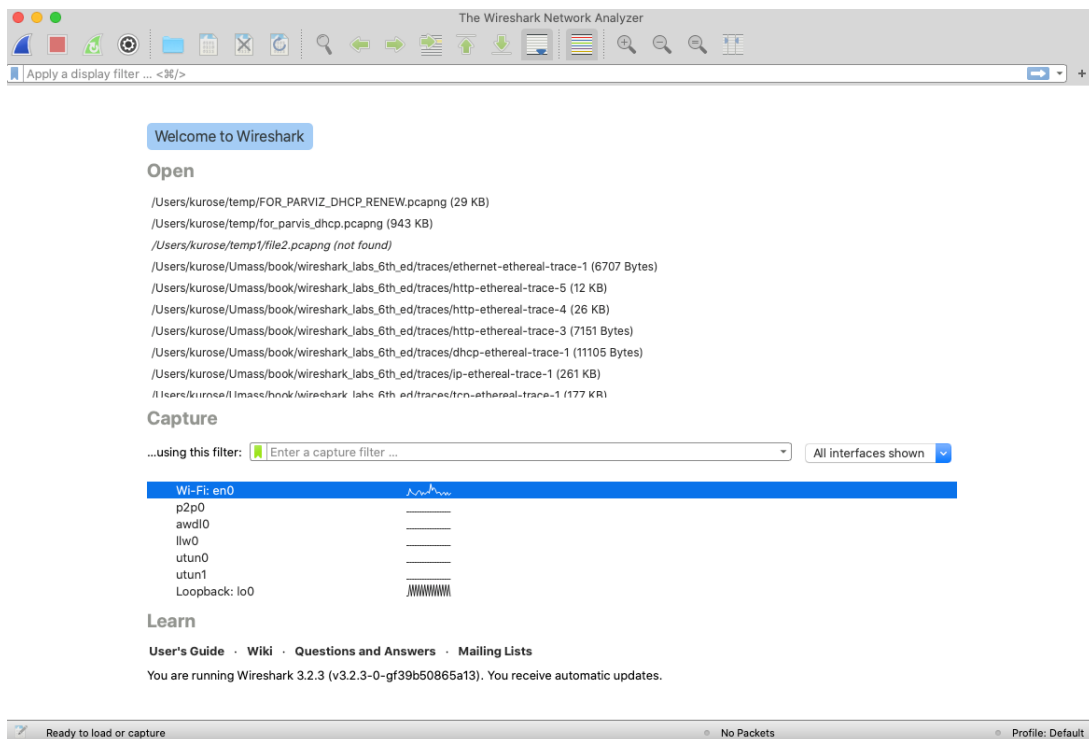


Figure 1: Initial Wireshark Screen

To do in the Lab:

We'll assume that your computer is connected to the Internet via a wired Ethernet interface or a WiFi interface. Do the following:

- Start up your favorite web browser, which will display your selected homepage.
- Start up the Wireshark software. You will initially see a window similar to that shown in Figure 1. Wireshark has not yet begun capturing packets.
- To begin packet capture, select the Capture pull down menu and select *Options*. This will cause the “Wireshark: Capture Interfaces” window to be displayed (on a PC). You should see a list of interfaces, as shown in Figure-2.

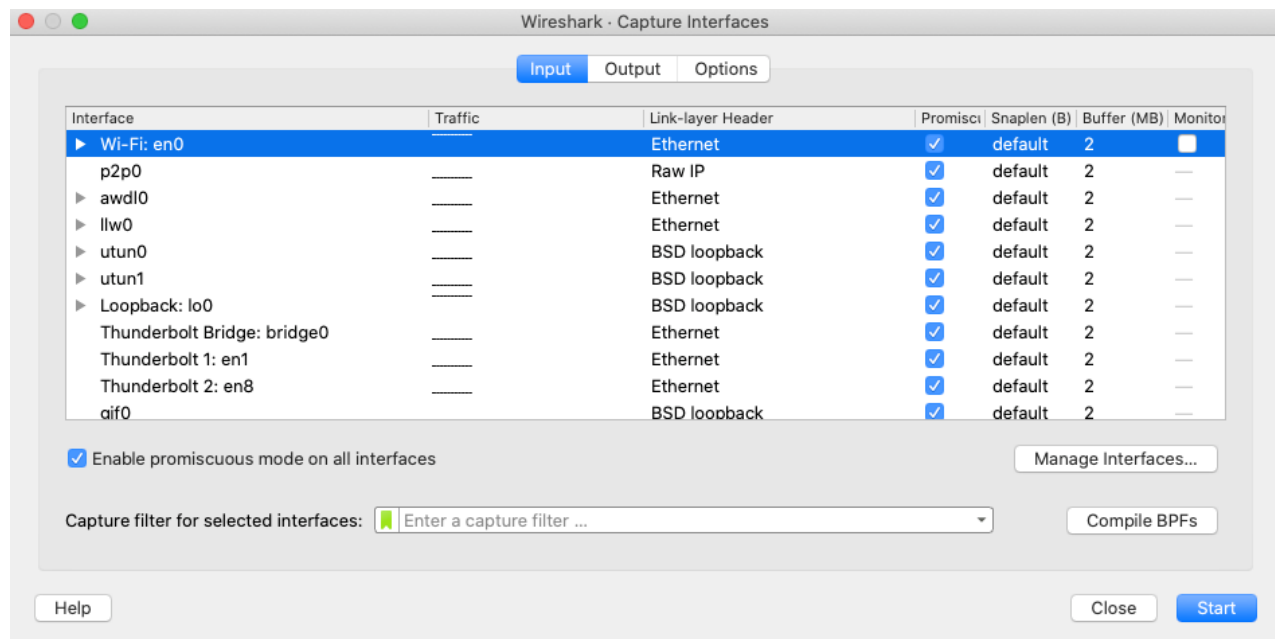


Figure 2: Wireshark Capture interface window

- You'll see a list of the interfaces on your computer as well as a count of the packets that have been observed on that interface so far. On a Windows machine, click on *Start* for the interface on which you want to begin packet capture (in the case in Figure-2, the WiFi or the Ethernet Connection). On a Windows machine, select the interface and click Start on the bottom of the window). Packet capture will now begin - Wireshark is now capturing all packets being sent/received from/by your computer!

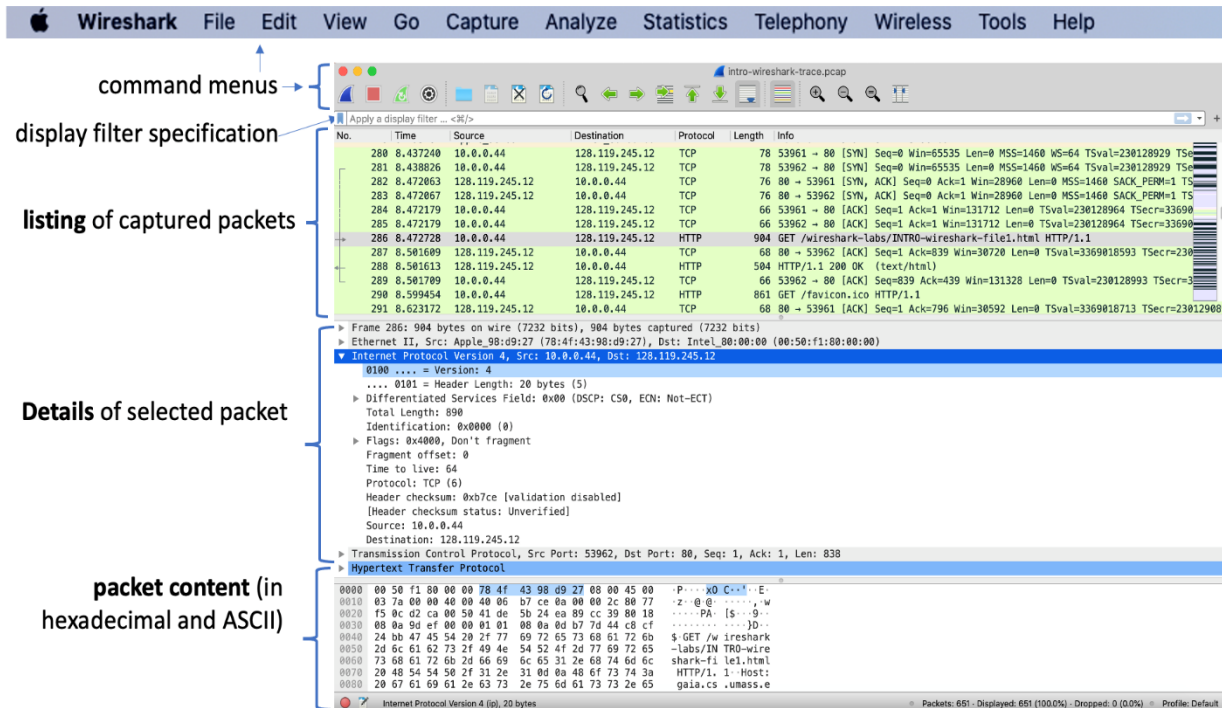


Figure 3: Wireshark window, during and after capture

- Once you begin packet capture, a window similar to that shown in Figure 3 will appear. This window shows the packets being captured. By selecting the *Capture* pulldown menu and selecting *Stop*, or by click on the red Stop square, you can stop packet capture. But don't stop packet capture yet. Let's capture some interesting packets first. To do so, we'll need to generate some network traffic. Let's do so using a web browser, which will use the HTTP protocol.
- While Wireshark is running, enter the URL: <http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html> and have that page displayed in your browser. In order to display this page, your browser will contact the HTTP server at gaia.cs.umass.edu and exchange HTTP messages with the server in order to download this page. The Ethernet or WiFi frames containing these HTTP messages (as well as all other frames passing through your Ethernet or WiFi adapter) will be captured by Wireshark.
- After your browser has displayed the INTRO-wireshark-file1.html page (it is a simple one line of congratulations), stop Wireshark packet capture by selecting stop in the Wireshark capture window. The main Wireshark window should now look similar to Figure 3. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP

message exchanges with the `gaia.cs.umass.edu` web server should appear somewhere in the listing of packets captured.

- Type in “http” (without the quotes, and *in lower case* – all protocol names are in lower case in Wireshark, and make sure to press your enter/return key) into the display filter specification window at the top of the main Wireshark window. Then select *Apply* (to the right of where you entered “http”) or just hit return. This will cause only HTTP message to be displayed in the packet-listing window. Figure 4 below shows a screenshot after the http filter has been applied to the packet capture window shown earlier in Figure 3. Note also that in the Selected packet details window, we’ve chosen to show detailed content for the Hypertext Transfer Protocol application message that was found within the TCP segment, that was inside the IPv4 datagram that was inside the Ethernet II (WiFi) frame.

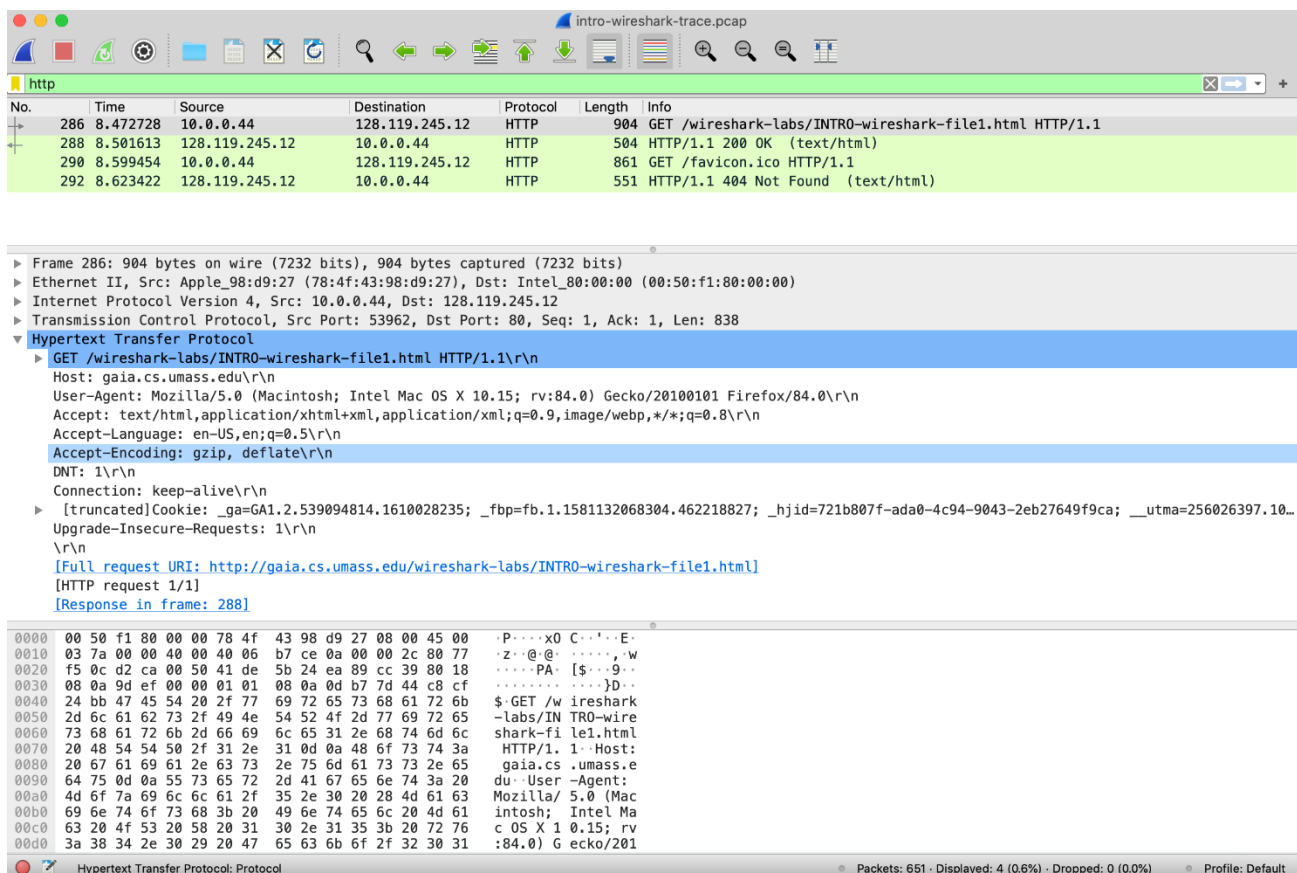


Figure 4: Looking at the details of the HTTP message that contained a GET of `http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html`

- Find the HTTP GET message that was sent from your computer to the gaia.cs.umass.edu HTTP server. (Look for an HTTP GET message in the “listing of captured packets” portion of the Wireshark window (see Figures 3 and 4) that shows “GET” followed by the gaia.cs.umass.edu URL that you entered. When you select the HTTP GET message, the Ethernet frame, IP datagram, TCP segment, and HTTP message header information will be displayed in the packet-header window.
- Exit Wireshark

Exercise-1:

1. Which of the following protocols are shown as appearing (i.e., are listed in the Wireshark “protocol” column) in your trace file: TCP, QUIC, HTTP, DNS, UDP, TLSv1.2?
2. How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since Wireshark tracing began. (If you want to display the Time field in time-of-day format, select the Wireshark *View* pull down menu, then select *Time Display Format*, then select *Time-of-day*.)
3. What is the Internet address of the gaia.cs.umass.edu (also known as www-net.cs.umass.edu)? What is the Internet address of your computer or (if you are using the trace file) the computer that sent the HTTP GET message?
4. See the fields in the TCP segment carrying the HTTP message. What is the destination port number (the number following “Dest Port:” for the TCP segment containing the HTTP request) and to which this HTTP request is being sent?

Lab-07-B

Aim: WireShark UDP Packets

Objective: Trace the UDP Packets which are sent and received by your host using WireShark.

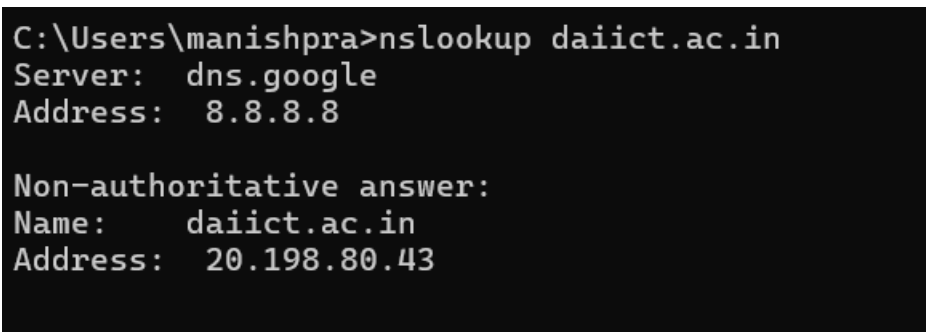
❖ *Pathway:*

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 **Domain Name System (DNS) protocol typically sends DNS query and response messages inside of UDP**, so it's likely that you'll find some DNS messages (and therefore UDP packets) in your trace.

❖ nslookup

Specifically, you can try out the *nslookup* command, which invokes the underlying DNS protocol, which in turn will send UDP segments from/to the host issuing the *nslookup*.

To run *nslookup* you just type the *nslookup* command on the command line in a DOS window, Mac IOS terminal window, or Linux shell. Figure 1 is a screenshot of running *nslookup* on the Windows CLI on our college's website 'daiict.ac.in' and in return we get the IP of the same.



```
C:\Users\manishpra>nslookup daiict.ac.in
Server:  dns.google
Address:  8.8.8.8

Non-authoritative answer:
Name:     daiict.ac.in
Address:  20.198.80.43
```

Figure 1: the basic nslookup command

To do in the Lab:

- After starting packet capture on Wireshark, run nslookup for a hostname that you haven't visited for a while.
- Then stop packet capture, set your Wireshark packet filter to '*udp*'(type it) so that Wireshark only displays the UDP segments sent and received at your host.
- Pick the first UDP segment and expand the UDP fields in the details window. If you are unable to find UDP segments in your trace or are unable to run Wireshark on a live network connection, you can download a packet trace containing some UDP segments.
- **NOTE:** Wireshark also assigns the packet numbers on the basis of packets getting traced in the order they are sent or received. The actual packet numbers are different and can be seen while expanding the details of the packet selected.

Exercise-2:

1. Select the first UDP segment in your trace. What is the packet number of this segment in the trace report? What type of application-layer payload or protocol message is being carried in this UDP segment? Look at the details of this packet in Wireshark. How many fields there are in the UDP header? (You shouldn't look for any book to answer them! Answer these questions directly from what you observe in the packet trace.) What are the names of these fields?
2. By consulting the displayed information in Wireshark's packet content field for this packet (or by consulting the textbook), what is the length (in bytes) of each of the UDP header fields?
3. The value in the Length field is the length of what? 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 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 the 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). What is the packet number of the first of these two UDP segments in the trace file? What is the value in the source port field in this UDP segment? What is the value in the destination port field in this UDP segment? What is the packet number of the second of these two UDP segments in the trace file? What is the value in the source port field in this second UDP segment? What is the value in the destination port field in this second UDP segment? Describe the relationship between the port numbers in the two packets.

LAB – 07 – C

Aim: Analyze TCP Packets in WireShark

Objective: Trace the TCP packets sent by the host and receiver, analyze them, calculate RTTs and understand the congestion control playing the role in their transmission.

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 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.

Pathway:

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.
- 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.
- 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.

Steps:

- a) Start up your web browser. Go the <http://gaia.cs.umass.edu/wireshark-labs/alice.txt> and retrieve an ASCII copy of *Alice in Wonderland*. Store this as a .txt file somewhere on your computer.
- b) Next go to <http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html>.
- c) You should see a screen that looks like Figure 1.

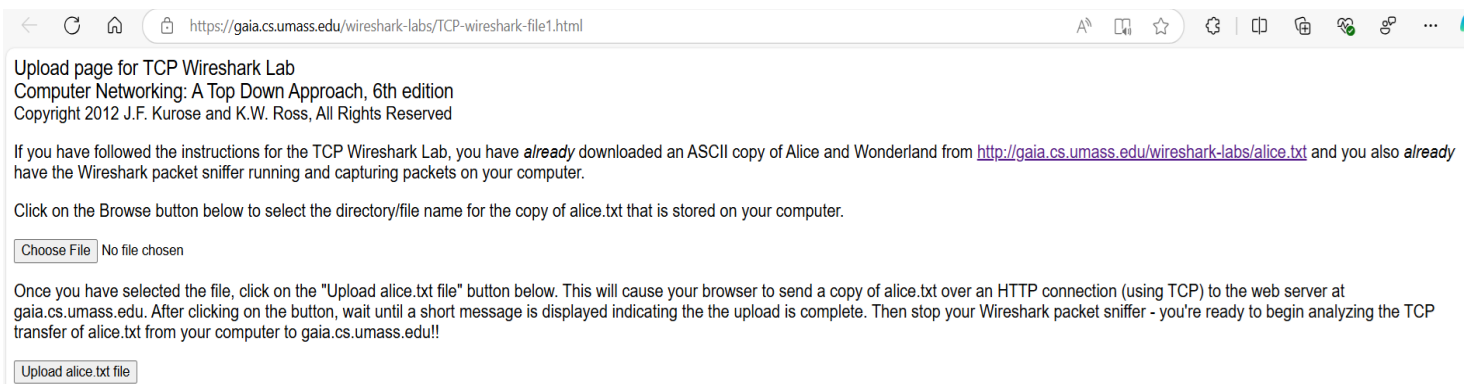


Figure 1: Page to upload the alice.txt file from your computer to gaia.cs.umass.edu

- d) Use the *Browse* button in this form to the file on your computer that you just created containing *Alice in Wonderland*. **Don't press the "Upload alice.txt file" button yet.**
- e) Now start up Wireshark and begin packet capture (see the earlier Wireshark labs if you need a refresher on how to do this).
- f) 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.
- g) Stop Wireshark packet capture. Your Wireshark window should look similar to the window shown in Figure 2.

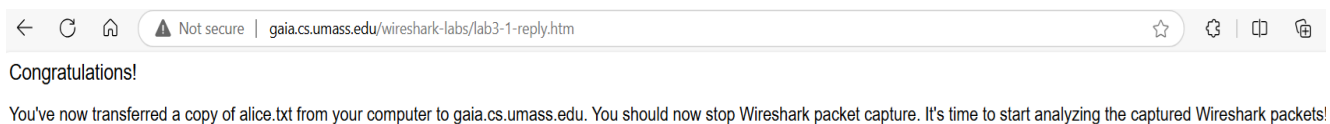


Figure 2: Success! You've uploaded a file to gaia.cs.umass.edu and have hopefully captured a Wireshark packet trace while doing so.

2. A first look at the captured trace

Before analyzing the behavior of the TCP connection in detail, let's take a high-level view of the trace.

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

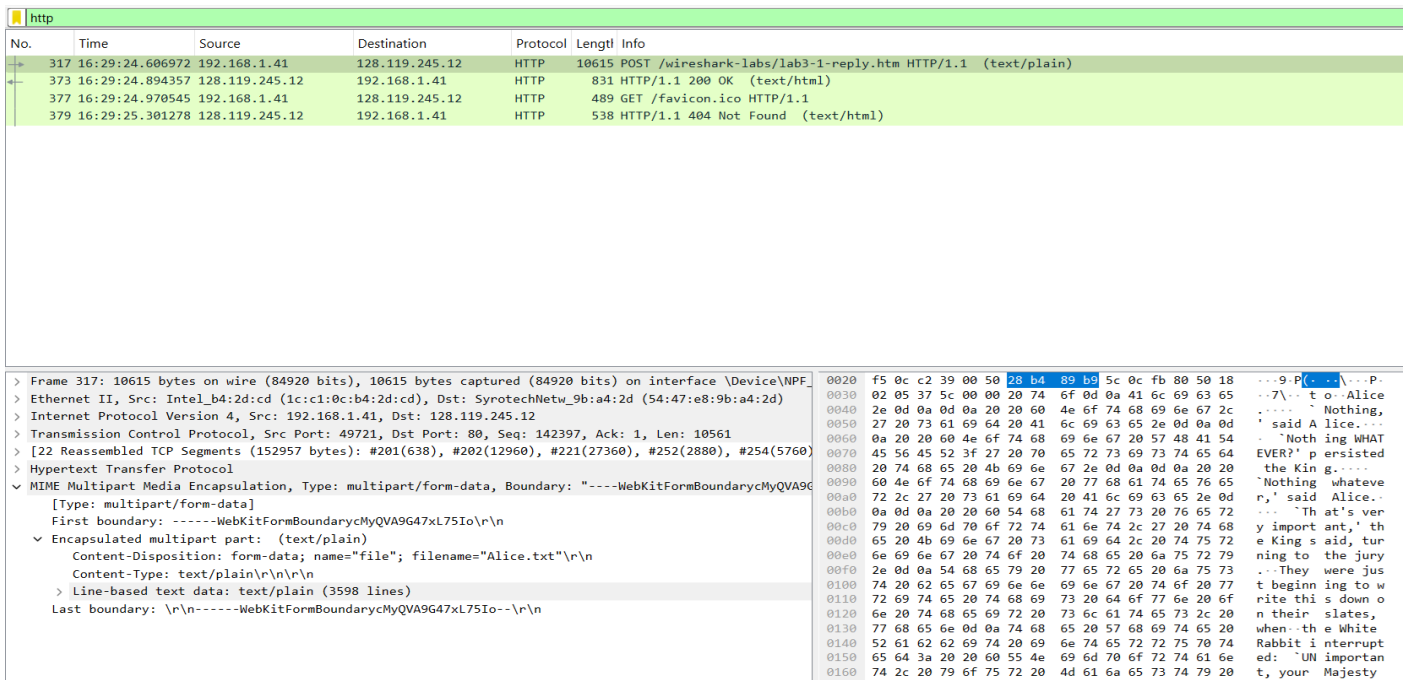


Figure 3: Expanding the HTTP POST message that uploaded alice.txt from your computer to gaia.cs.umass.edu

There are a few things to note here:

- The body of your application-layer HTTP POST message contains the contents of the file alice.txt, which is a large file of more than 152K bytes. OK – it’s not *that* large, but it’s going to be too large for this one HTTP POST message to be contained in just one TCP segment!
- In fact, as shown in the Wireshark window in Figure 3 we see **that the HTTP POST message was spread across 22 TCP segments**. This is shown where the red box is placed in Figure 3 [Aside: Wireshark doesn’t have a red box like that; we added it to the figure to be helpful ☺]. If you look even more carefully there, you can see that Wireshark is being really helpful to you as well, telling you that the first TCP segment containing the beginning of the POST message is packet #201 in the particular trace for the example in Figure 3. The second TCP segment containing the POST message in packet #202 in the trace, and so on.

Let’s now “get our hands dirty” by looking at some TCP segments.

- First, filter the packets displayed in the Wireshark window by entering “tcp” (lowercase, no quotes, and don’t forget to press return after entering!) into the display filter specification window towards the top of the Wireshark window.
- Your Wireshark display should look something like Figure 4.
- In Figure 4, we’ve noted the TCP segment that has its SYN bit set – this is the first TCP message in the three-way handshake that sets up the TCP connection to gaia.cs.umass.edu that will eventually carry the HTTP POST message and the alice.txt file.

- We've also noted the SYN-ACK segment (the second step in TCP three-way handshake), as well as the TCP segment (packet #201, as discussed above) that carries the POST message and the beginning of the alice.txt file. Of course, in your own trace report, the packet numbers will be different, but you should see similar behavior to that shown in Figures 3 and 4.

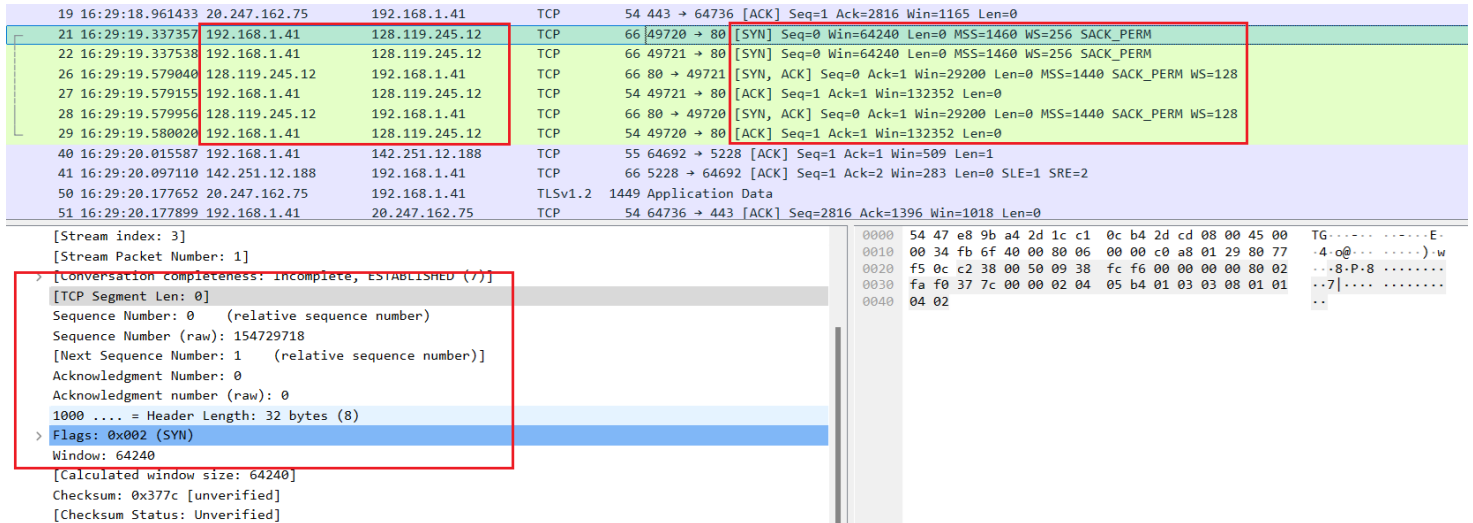


Figure 4: TCP segments involved in sending the HTTP POST message (including the file alice.txt) to gaia.cs.umass.edu

Questions: [answer them from your own trace report]

- What is the IP address and TCP port number used by the client computer (source) that is transferring the alice.txt file to gaia.cs.umass.edu? (Hint: Explore the details of a TCP Packet).
- What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?

Keep the filter to TCP. We are going to solve some more questions on it.

3. TCP Basics

NOTE: It is the “raw” sequence number carried in the TCP segment itself; it is *NOT* the packet # in the “No.” column in the Wireshark window. Remember there is no such thing as a “packet number” in TCP or UDP; as you know, there *are* sequence numbers in TCP and that’s what we’re after here. Also note that this is not the relative sequence number with respect to the starting sequence number of this TCP session.

Questions:

- 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 this TCP segment that identifies the segment as a SYN segment? (Hint: Refer note)

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 it in the segment that identifies the segment as a SYNACK segment? What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? (Hint: To know the client, check your PC's IP address)
5. Count the no. of packets required for transferring the whole file('Alice.txt'). Don't include the packets used for connection re/establishment. (Hint: Check for 'ACK' packets which went from client to server, and the 'Len' is not '0')
6. Consider the TCP segment containing the HTTP "POST" as the first segment in the data transfer part of the TCP connection.
 - At what time was the first segment (the one containing the HTTP POST) in the data-transfer part of the TCP connection sent?
 - At what time was the ACK for this first data-containing segment received?
 - What is the RTT for this first data-containing segment?
 - What is the RTT value the second data-carrying TCP segment and its ACK?
7. What is the length (header plus payload) of each of the first four data-carrying TCP segments?
8. What is the minimum amount of available buffer space advertised to the client by gaia.cs.umass.edu among these first four data-carrying TCP segments¹? Does the lack of receiver buffer space ever throttle the sender for these first four data-carrying segments?
9. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?
10. How much data does the receiver typically acknowledge in an ACK among the first ten data-carrying segments sent from the client to gaia.cs.umass.edu? Can you identify cases where the receiver is ACKing every other received segment among these first ten data-carrying segments?
11. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

4. 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(Stevens)*—to plot out data.

- Select a client-sent TCP segment in the Wireshark's "listing of captured-packets" window corresponding to the transfer of alice.txt from the client to gaia.cs.umass.edu.

Then select the menu: *Statistics->TCP Stream Graph-> Time-Sequence-Graph(Stevens)*².

You should see a plot that looks similar to the plot in Figure 5, which was created from the

captured packets in the packet trace *tcp-wireshark-trace1-1*. You may have to expand, shrink, and fiddle around with the intervals shown in the axes in order to get your graph to look like Figure 5.

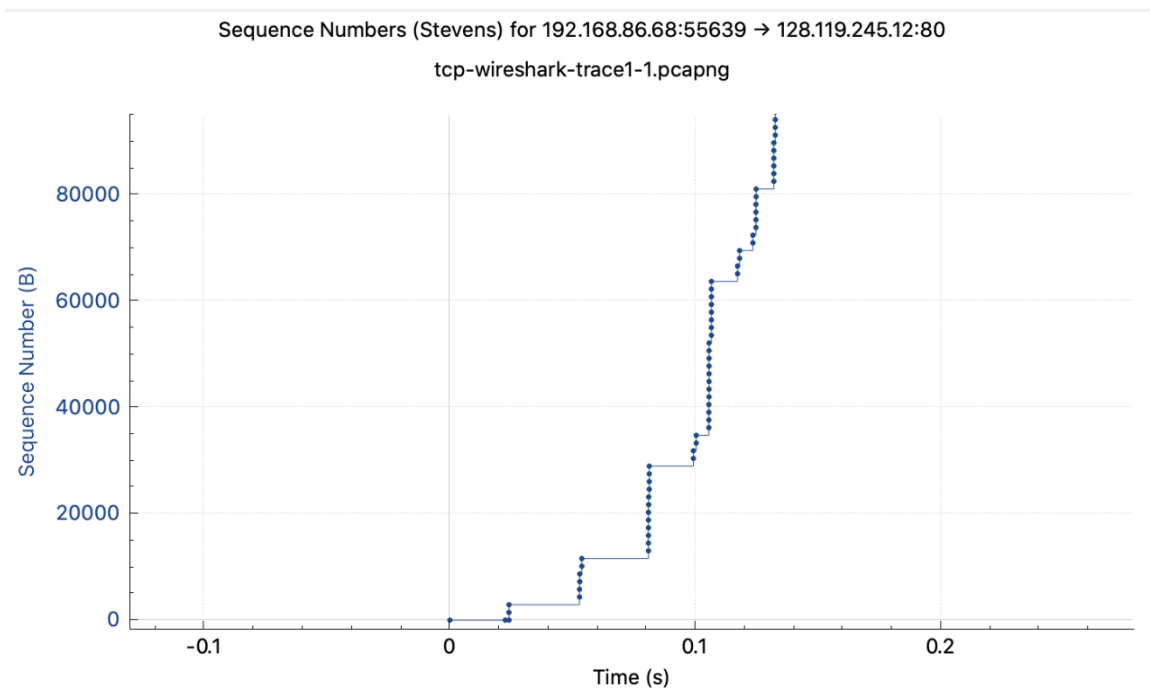


Figure 5: A sequence-number-versus-time plot (Stevens format) of TCP segments.

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

Questions:

12. 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. Consider the “fleets” of packets sent around $t = 0.025$, $t = 0.053$, $t = 0.082$ and $t = 0.1$. Comment on whether this looks as if TCP is in its slow start phase, congestion avoidance phase or some other phase. Figure 6 shows a slightly different view of this data.
13. These “fleets” of segments appear to have some periodicity. What can you say about the period?
14. Answer each of two questions above for the trace that you have gathered when you transferred a file from your computer to `gaia.cs.umass.edu`

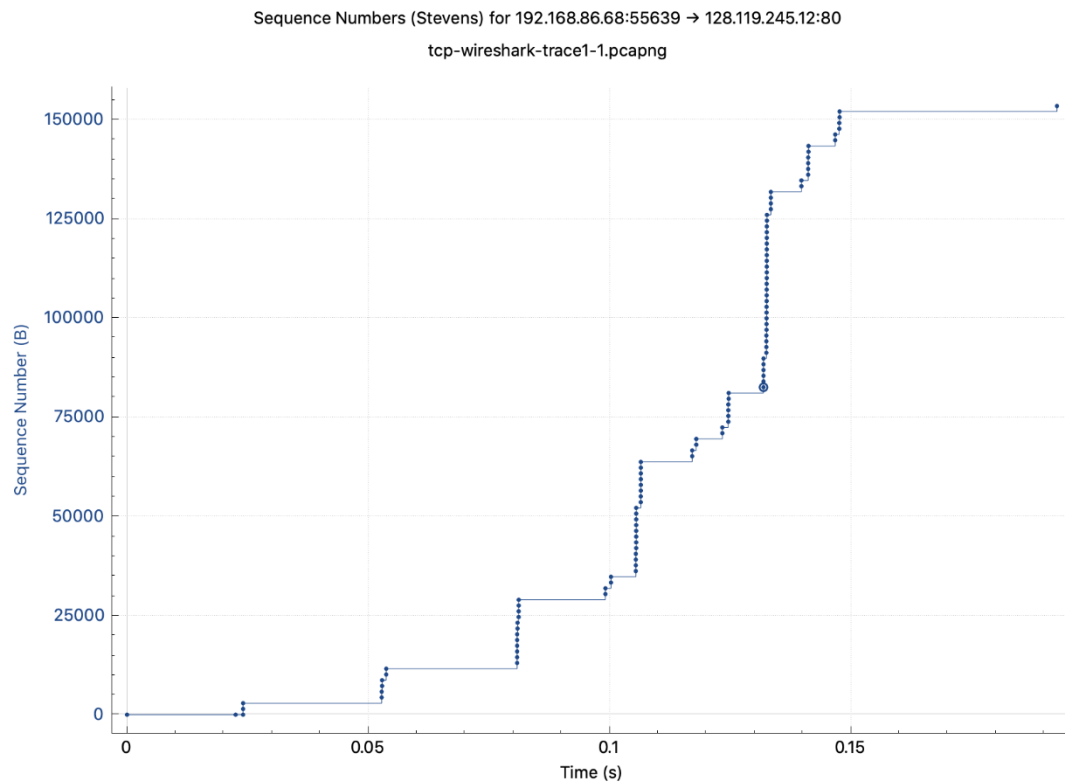


Figure 6: Another view of the same data as in Figure 5.

+

References:

- <https://www.wireshark.org/download.html>
- https://gaia.cs.umass.edu/kurose_ross/index.php

