CS 7349
Data and Network Security
Homework #1 Wireshark

Name: Bingying Liang
ID: 48999397

Due: Feb 5 2024

Name: Bingying Liang 

What to Submit

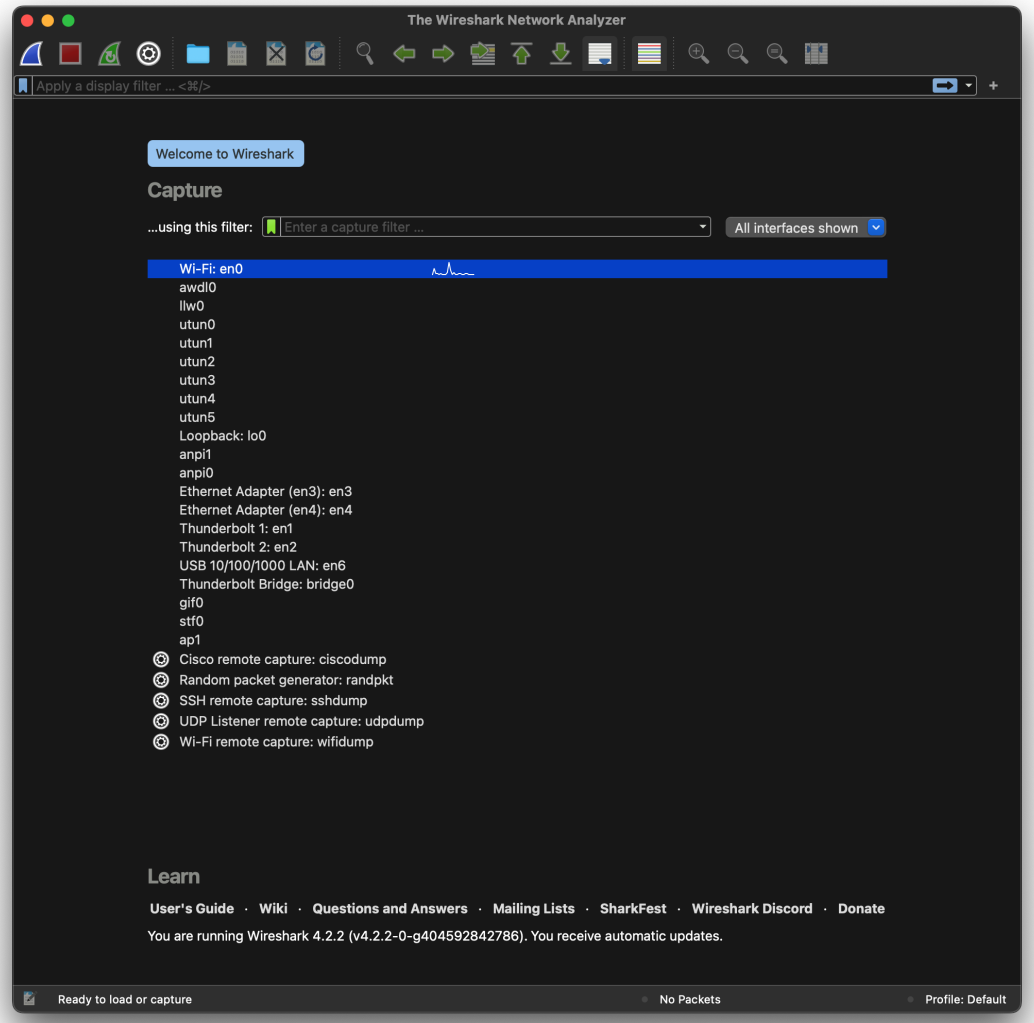
Your final submission shall be a single pdf document that includes this document, screen captures of your exercises plus your answers to each of the written questions (if any). Note that you are expected to clearly label each section so as to make it clear to the instructor what files and data belong to which exercise/question.

Collaboration is expected and encouraged; however, each student must hand in their own homework assignment. To the greatest extent possible, answers should not be copied but, instead, should be written in your own words. Copying answers from anywhere is plagiarism, this includes copying text directly from the textbook. Do not copy answers. Always use your own words. For each question list all persons with whom you collaborated and list all resources used in arriving at your answer. Resources include but are not limited to the textbook used for this course, papers read on the topic, and Google search results. Note that 'Google' is not a resource. Don't forget to place your name on the document.

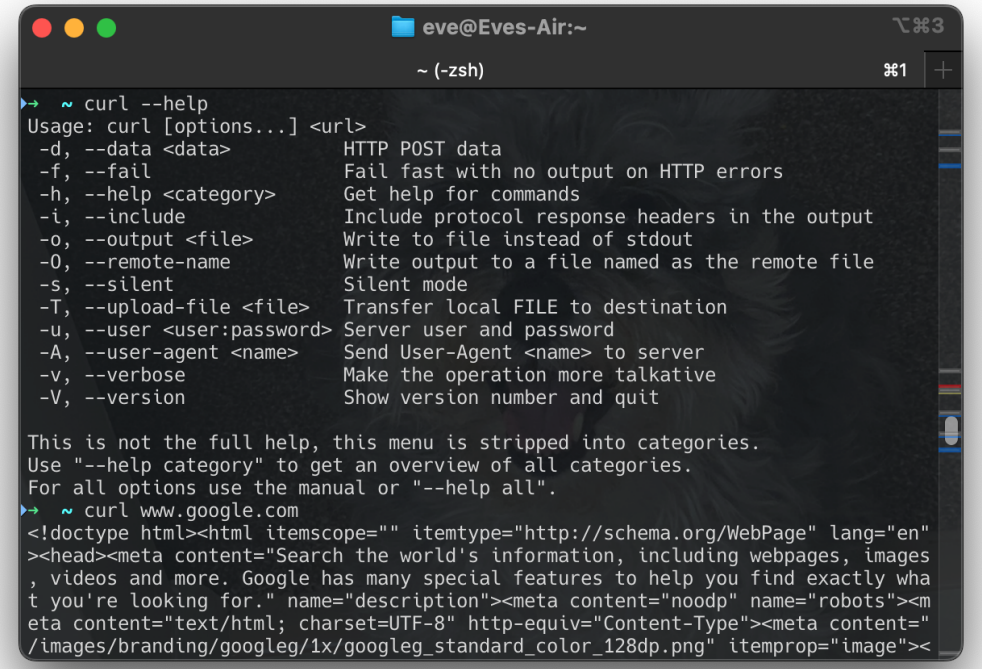
Exercise 1 : Wireshark Protocol Layers

The goal of this exercise is to become familiar with Wireshark, a network packet sniffer and analysis tool, and to observe how protocols and layering are represented in packets. This exercise is adapted from the Protocol Wireshark Lab by David Wetherall.

This exercise uses the Wireshark software tool to capture and examine a packet trace. A packet trace is a record of traffic at a location on the network, as if a snapshot was taken of all the bits that passed across a particular wire. The packet trace records a timestamp for each packet, along with the bits that make up the packet, from the lower-layer headers to the higher-layer contents. Wireshark runs on most operating systems, including Windows, Mac and Linux. It provides a graphical user interface that shows the sequence of packets and the meaning of the bits when interpreted as protocol headers and data. It color-codes packets by their type, and has various ways to filter and analyze packets to let you investigate the behavior of network protocols. Wireshark is widely used to troubleshoot networks. You can download Wireshark from www.wireshark.org if it is not already installed on your computer. We highly recommend that you watch the short, 5 minute video ?Introduction to Wireshark? that is on the site.



This exercise also introduces `wget` (Linux and Windows) and `curl` (Mac) to fetch web resources. `wget` and `curl` are command-line programs that let you fetch a URL. Unlike a web browser, which fetches and executes entire pages, `wget` and curl give you control over exactly which URLs you fetch and when you fetch them. Under Linux, wget can be installed via your package manager. Under Windows, `wget` is available as a binary; look for download information on http://www.gnu.org/software/wget/. Under Mac, `curl` comes installed with the OS. Both have many options (try `wget --help` or `curl --help` to see) but a URL can be fetched simply with `wget URL` or `curl URL`.



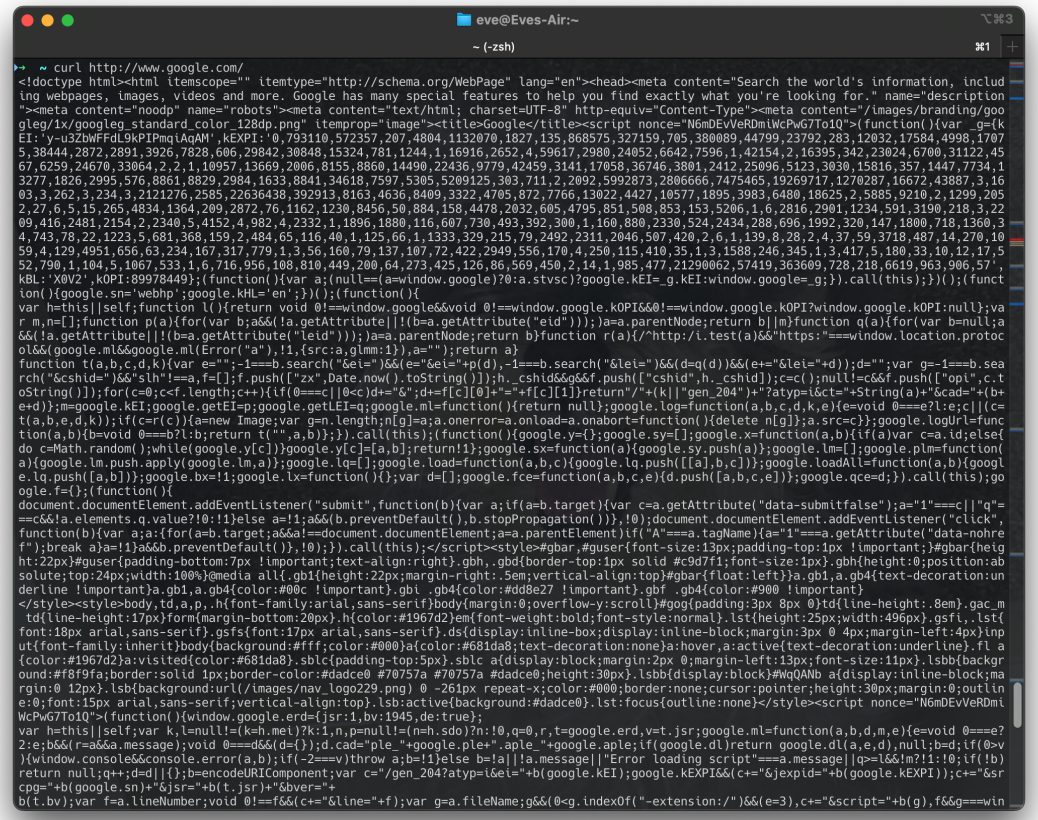
After installing Wireshark, perform each of the steps below.

Step 1: Capture a Trace

Proceed as follows to capture a trace of network traffic. We want this trace to look at the protocol structure of packets. A simple Web fetch of a URL from a server of your choice to your computer, which is the client, will serve as traffic.

1) Pick a URL and fetch it with wget or curl. For example, wget `http://www.google.com` or `curl http://www.google.com`. This will fetch the resource and either write it to a file (wget) or to the screen (curl). You are checking to see that the fetch works and retrieves some content. If the fetch does not work then try a different URL; if no URLs seem to work then debug your use of `wget/curl` or your Internet connectivity.

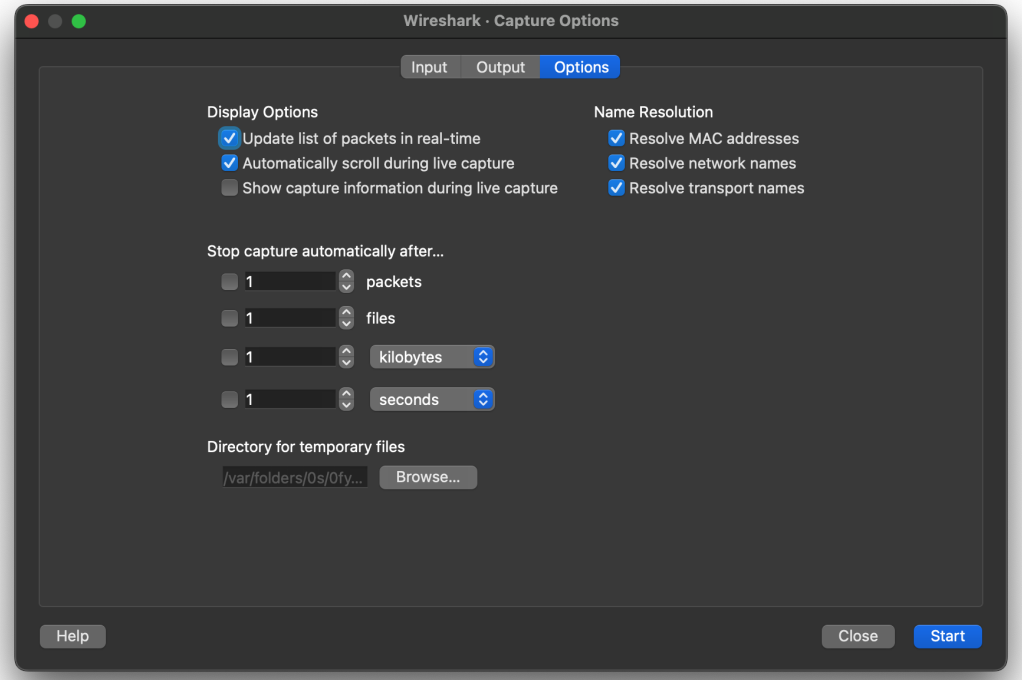
   Solution:

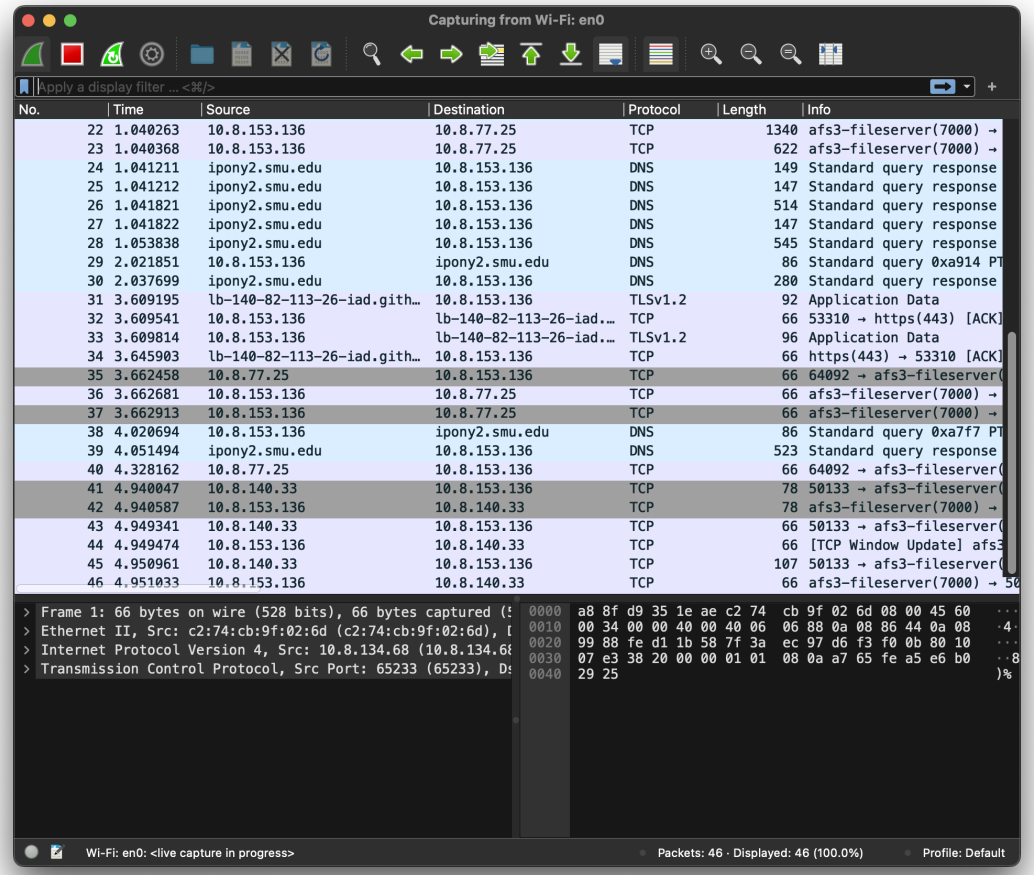


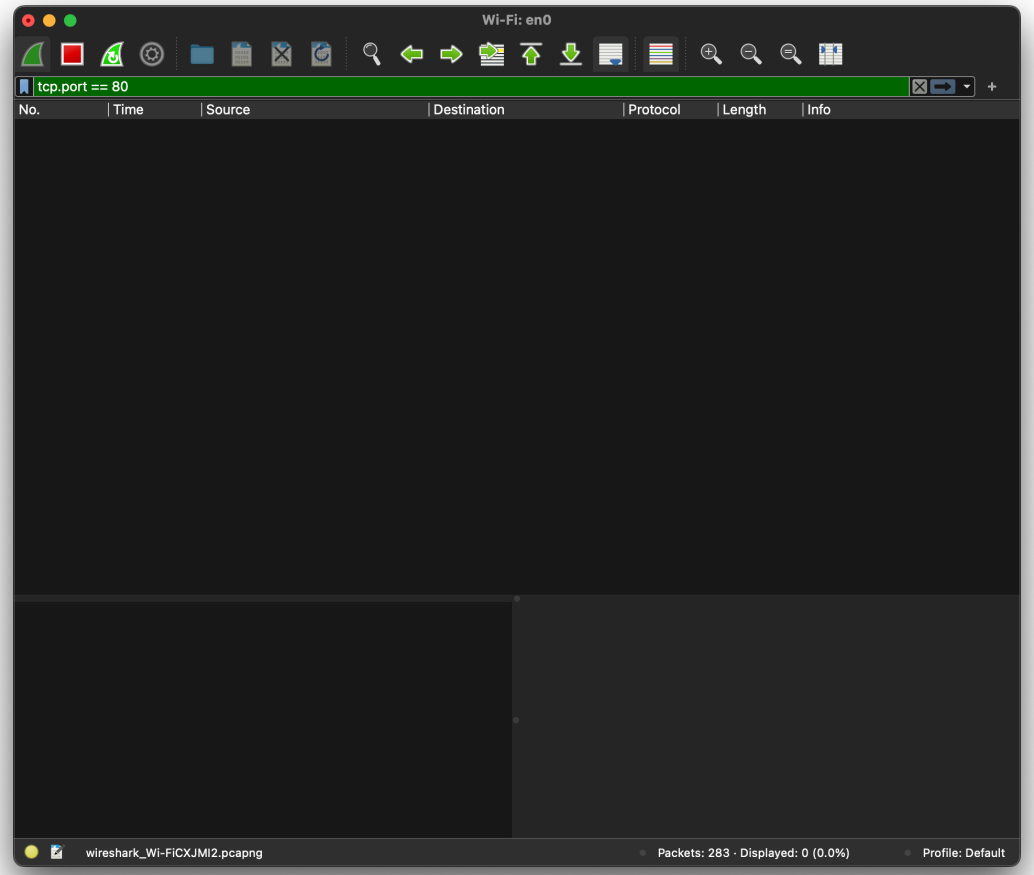
2) Close unnecessary browser tabs and windows. By minimizing browser activity you will stop your computer from fetching unnecessary web content and avoid incidental traffic in the trace.

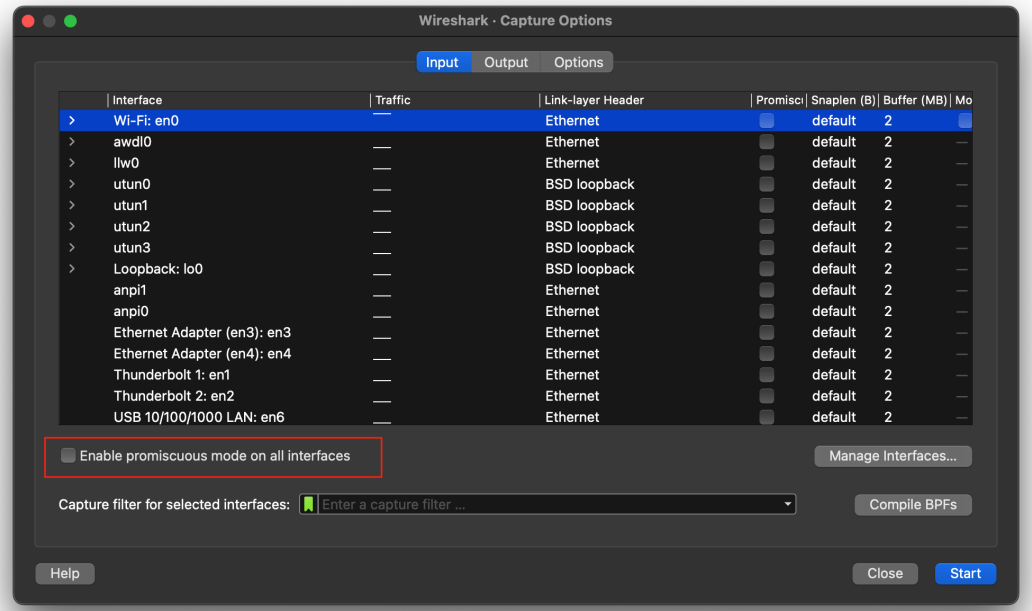
3) Launch Wireshark and start a capture with a filter of `tcp port 80` and *check enable network name resolution*. This filer will record only standard web traffic and not other kinds of packets that your computer may send. The checking will translate the addresses of the computers sending and receiving packets into names, which should help you to recognize whether the packets are going to or from your computer. Select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful. Uncheck capture packets in promiscuous mode. This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets sent to/from your computer. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive. On Wireshark 1.8, the capture filter box is present directly on the options screen, but on Wireshark 1.9, you set a capture filter by double-clicking on the interface.

Solution:


Wireshark · Capture Options
Input
Output
Options
Display Options
Update list of packets in real-time
Automatically scroll during live capture
Show capture information during live capture
Name Resolution
Resolve MAC addresses
Resolve network names
Resolve transport names
Stop capture automatically after...
1
packets
1
files
1
kilobytes
1
seconds
Directory for temporary files
/var/folders/0s/0fy...
Browse...
Help
Close
Start

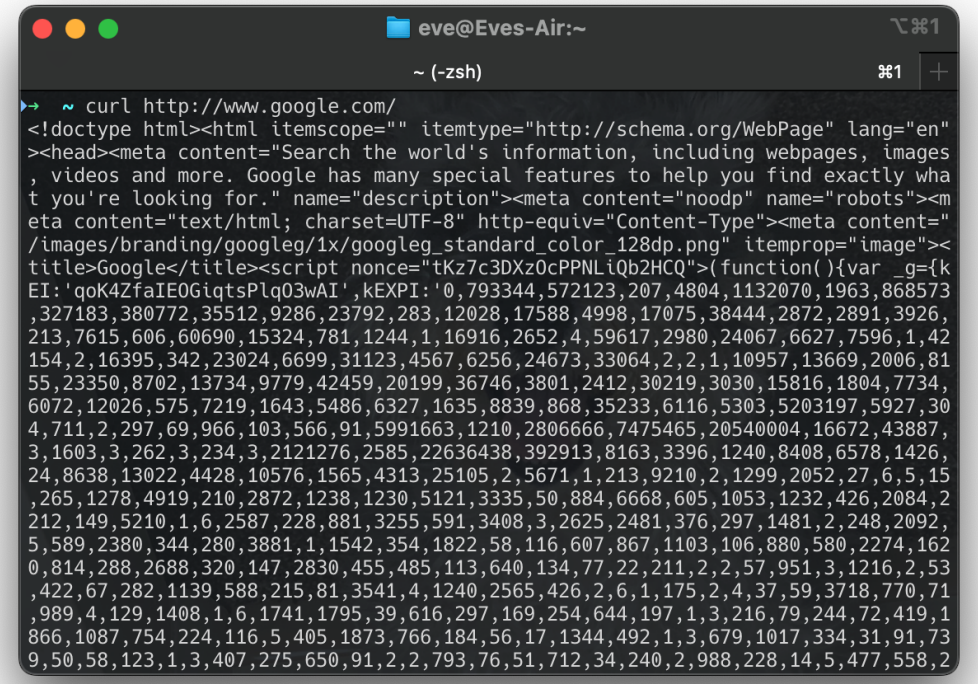

Capturing from Wi-Fi: en0
Apply a display filter ... <⌘/>
No. Time Source Destination Protocol Length Info
22 1.040263 10.8.153.136 10.8.77.25 TCP 1340 afs3-fileserver(7000) →
23 1.040368 10.8.153.136 10.8.77.25 TCP 622 afs3-fileserver(7000) →
24 1.041211 ipony2.smu.edu 10.8.153.136 DNS 149 Standard query response
25 1.041212 ipony2.smu.edu 10.8.153.136 DNS 147 Standard query response
26 1.041821 ipony2.smu.edu 10.8.153.136 DNS 514 Standard query response
27 1.041822 ipony2.smu.edu 10.8.153.136 DNS 147 Standard query response
28 1.053838 ipony2.smu.edu 10.8.153.136 DNS 545 Standard query response
29 2.021851 10.8.153.136 ipony2.smu.edu DNS 86 Standard query 0xa914 PT
30 2.037699 ipony2.smu.edu 10.8.153.136 DNS 280 Standard query response
31 3.609195 lb-140-82-113-26-iad.gith… 10.8.153.136 TLSv1.2 92 Application Data
32 3.609541 10.8.153.136 lb-140-82-113-26-iad.… TCP 66 53310 → https(443) [ACK]
33 3.609814 10.8.153.136 lb-140-82-113-26-iad.… TLSv1.2 96 Application Data
34 3.645903 lb-140-82-113-26-iad.gith… 10.8.153.136 TCP 66 https(443) → 53310 [ACK]
35 3.662458 10.8.77.25 10.8.153.136 TCP 66 64092 → afs3-fileserver(
36 3.662681 10.8.153.136 10.8.77.25 TCP 66 afs3-fileserver(7000) →
37 3.662913 10.8.153.136 10.8.77.25 TCP 66 afs3-fileserver(7000) →
38 4.020694 10.8.153.136 ipony2.smu.edu DNS 86 Standard query 0xa7f7 PT
39 4.051494 ipony2.smu.edu 10.8.153.136 DNS 523 Standard query response
40 4.328162 10.8.77.25 10.8.153.136 TCP 66 64092 → afs3-fileserver(
41 4.940047 10.8.140.33 10.8.153.136 TCP 78 50133 → afs3-fileserver(
42 4.940587 10.8.153.136 10.8.140.33 TCP 78 afs3-fileserver(7000) →
43 4.949341 10.8.140.33 10.8.153.136 TCP 66 50133 → afs3-fileserver(
44 4.949474 10.8.153.136 10.8.140.33 TCP 66 [TCP Window Update] afs3
45 4.950961 10.8.140.33 10.8.153.136 TCP 107 50133 → afs3-fileserver(
46 4.951033 10.8.153.136 10.8.140.33 TCP 66 afs3-fileserver(7000) → 50
> Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (5
> Ethernet II, Src: c2:74:cb:9f:02:6d (c2:74:cb:9f:02:6d), D
> Internet Protocol Version 4, Src: 10.8.134.68 (10.8.134.68
> Transmission Control Protocol, Src Port: 65233 (65233), Ds
0000 a8 8f d9 35 1e ae c2 74 cb 9f 02 6d 08 00 45 60 ···
0010 00 34 00 00 40 00 40 06 06 88 0a 08 86 44 0a 08 ·4··
0020 99 88 fe d1 1b 58 7f 3a ec 97 d6 f3 f0 0b 80 10 ···
0030 07 e3 38 20 00 00 01 01 08 0a a7 65 fe a5 e6 b0 ··8
0040 29 25)%
Wi-Fi: en0: <live capture in progress>
Packets: 46 · Displayed: 46 (100.0%)
Profile: Default

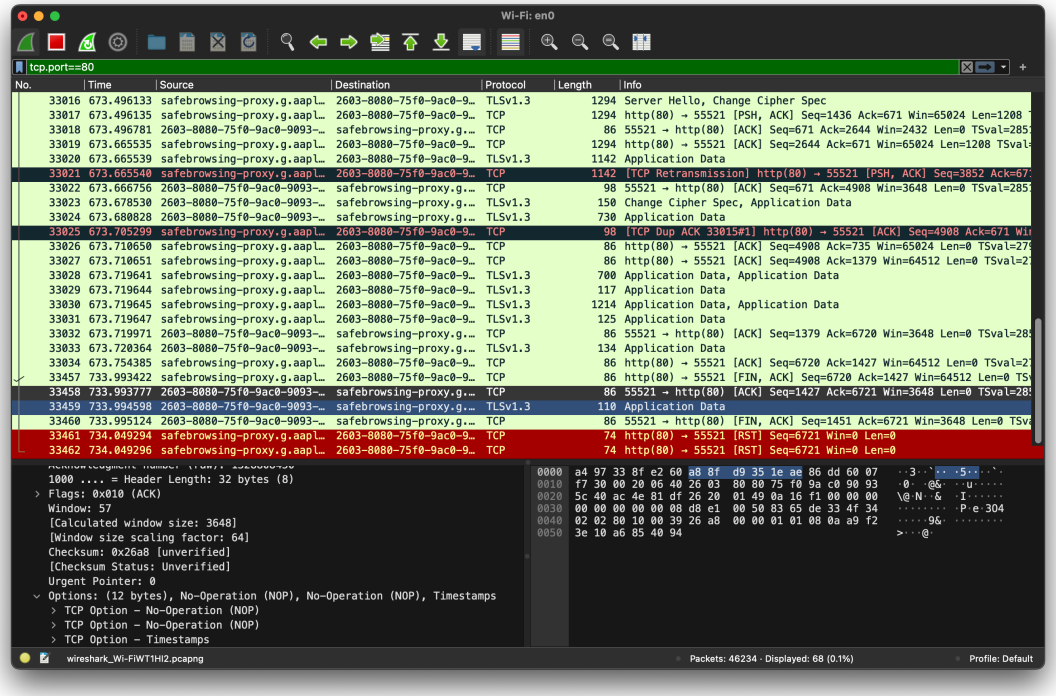

Wi-Fi: en0
tcp.port == 80
No.
Time
Source
Destination
Protocol
Length
Info
wireshark_Wi-FiCXJMI2.pcapng
Packets: 283 · Displayed: 0 (0.0%)
Profile: Default


Wireshark · Capture Options
Input
Output
Options
Interface
Traffic
Link-layer Header
Promisc
Snaplen (B)
Buffer (MB)
Wi-Fi: en0 Ethernet default 2
awdl0 Ethernet default 2
llw0 Ethernet default 2
utun0 BSD loopback default 2
utun1 BSD loopback default 2
utun2 BSD loopback default 2
utun3 BSD loopback default 2
Loopback: lo0 BSD loopback default 2
anpi1 Ethernet default 2
anpi0 Ethernet default 2
Ethernet Adapter (en3): en3 Ethernet default 2
Ethernet Adapter (en4): en4 Ethernet default 2
Thunderbolt 1: en1 Ethernet default 2
Thunderbolt 2: en2 Ethernet default 2
USB 10/100/1000 LAN: en6 Ethernet default 2
Enable promiscuous mode on all interfaces
Manage Interfaces...
Capture filter for selected interfaces:
Enter a capture filter ...
Compile BPFs
Help
Close
Start

4) When the capture is started, repeat the web fetch using wget/curl above. This time, the packets will be recorded by Wireshark as the content is transferred.

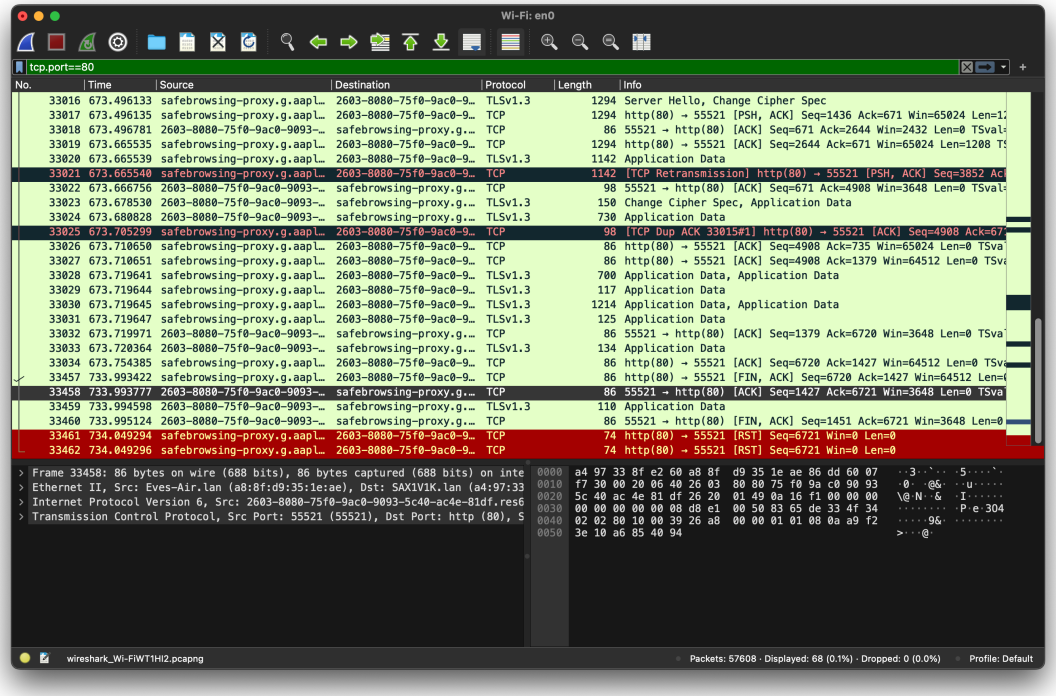
Solution:





5) After the fetch is successful, return to Wireshark and use the menus or buttons to stop the trace. If you have succeeded, the upper Wireshark window will show multiple packets, and most likely it will be full. How many packets are captured will depend on the size of the web page, but there should be at least 8 packets in the trace, and typically 20-100, and many of these packets will be colored green. Congratulations! You have captured your first trace.

Solution:

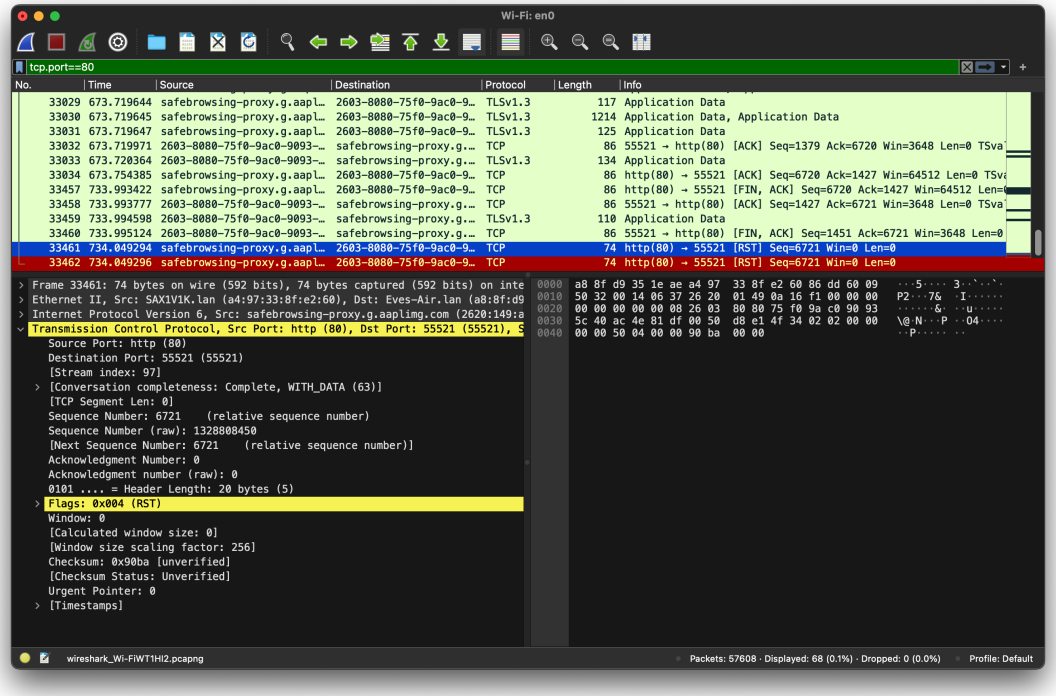


Turn In: Screen capture your Wireshark trace and turn it in.

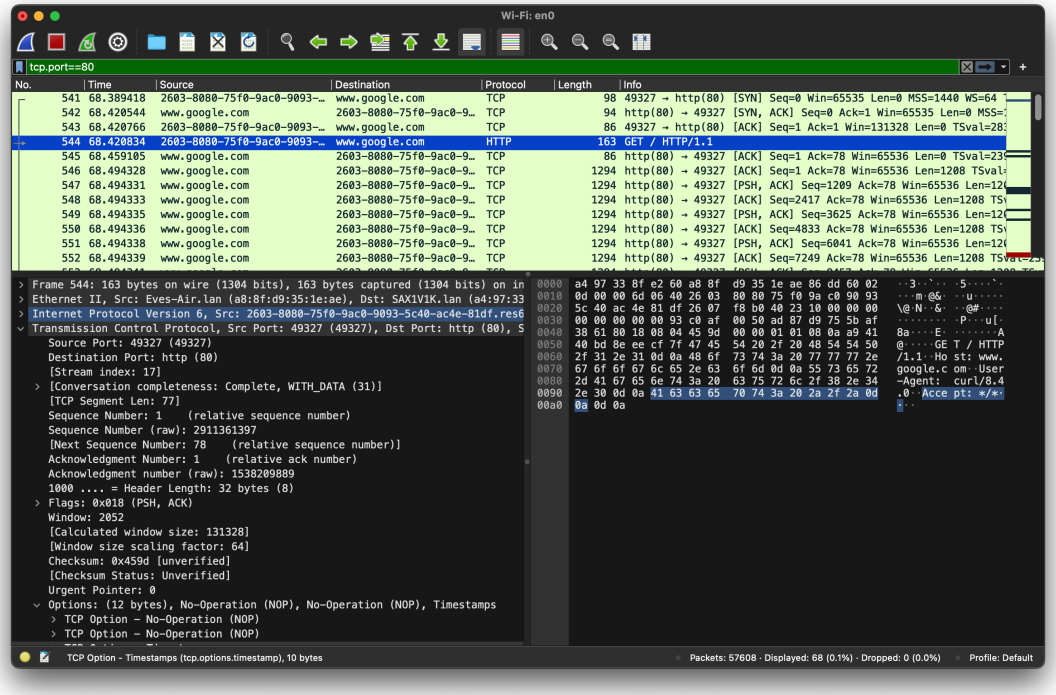
Solution: I post the Screen capture above.

Step 2: Inspect the Trace

Wireshark will let us select a packet (from the top panel) and view its protocol layers, in terms of both header fields (in the middle panel) and the bytes that make up the packet (in the bottom panel). In the figure above, the first packet is selected (shown in blue). Note that we are using *packet* as a general term here. Strictly speaking, a unit of information at the Link Layer is called a frame. At the Network Layer it is called a packet, at the Transport Layer a segment, and at the Application Layer a message. Wireshark is gathering frames and presenting us with the higher-layer packet, segment, and message structures it can recognize that are carried within the frames. We will often use *packet* for convenience, as each frame contains one packet and it is often the packet or higher-layer details that are of interest.



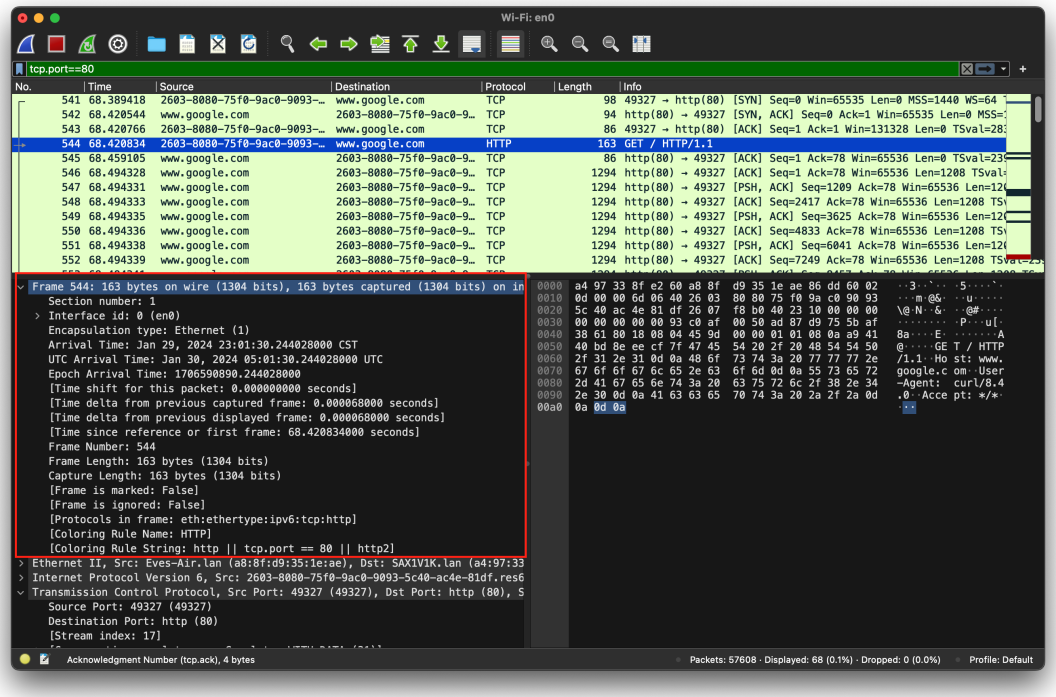
Select a packet for which the Protocol column is ‘HTTP’ and the Info column says it is a GET. It is the packet that carries the web (HTTP) request sent from your computer to the server. (You can click the column headings to sort by that value, though it should not be difficult to find an HTTP packet by inspection.) Let’s have a closer look to see how the packet structure reflects the protocols that are in use.



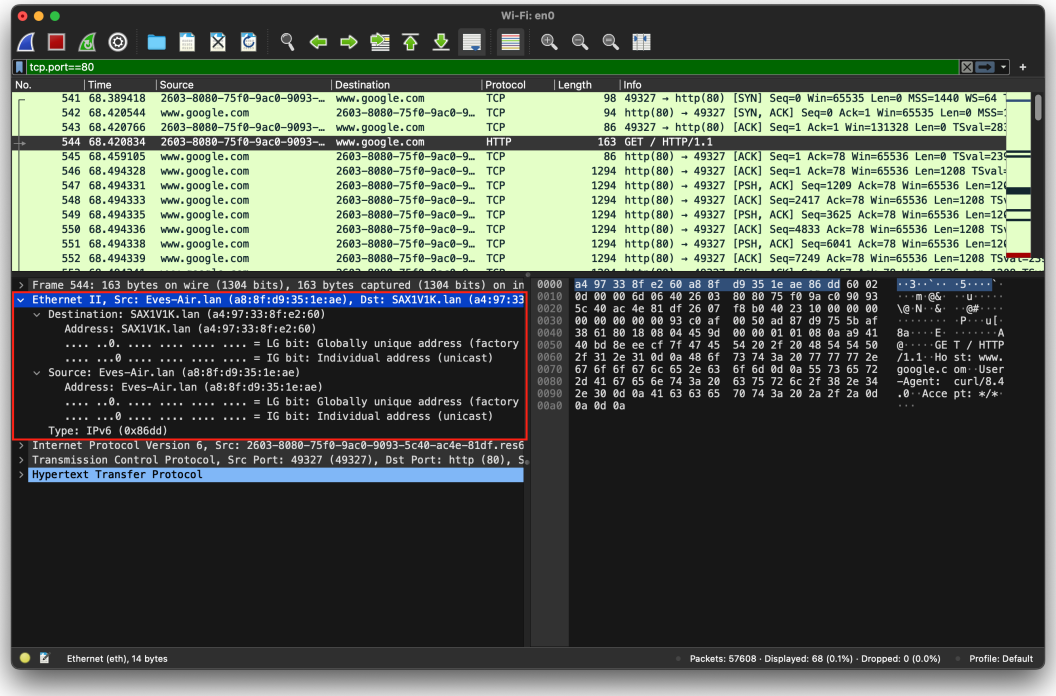
Since we are fetching a web page, we know that the protocol layers being used are from the HTTP stack (namely, HTTP at the Application Layer, TCP at the Transport Layer, IP at the Network Layer, and either Ethernet or WiFi at the Link and Physical layers). That is, HTTP is the application layer web protocol used to fetch URLs. Like many Internet applications, it runs on top of the TCP/IP Transport and Network layer protocols. The Link and Physical layer protocols depend on your network, but are typically combined in the form of Ethernet if your computer is wired, or WiFi if your computer is wireless.

With the HTTP GET packet selected, look closely to see the similarities and differences between it and our protocol stack as described next. The protocol blocks are listed in the middle panel. You can expand each block (by clicking on the '+' expander or icon) to see its details.

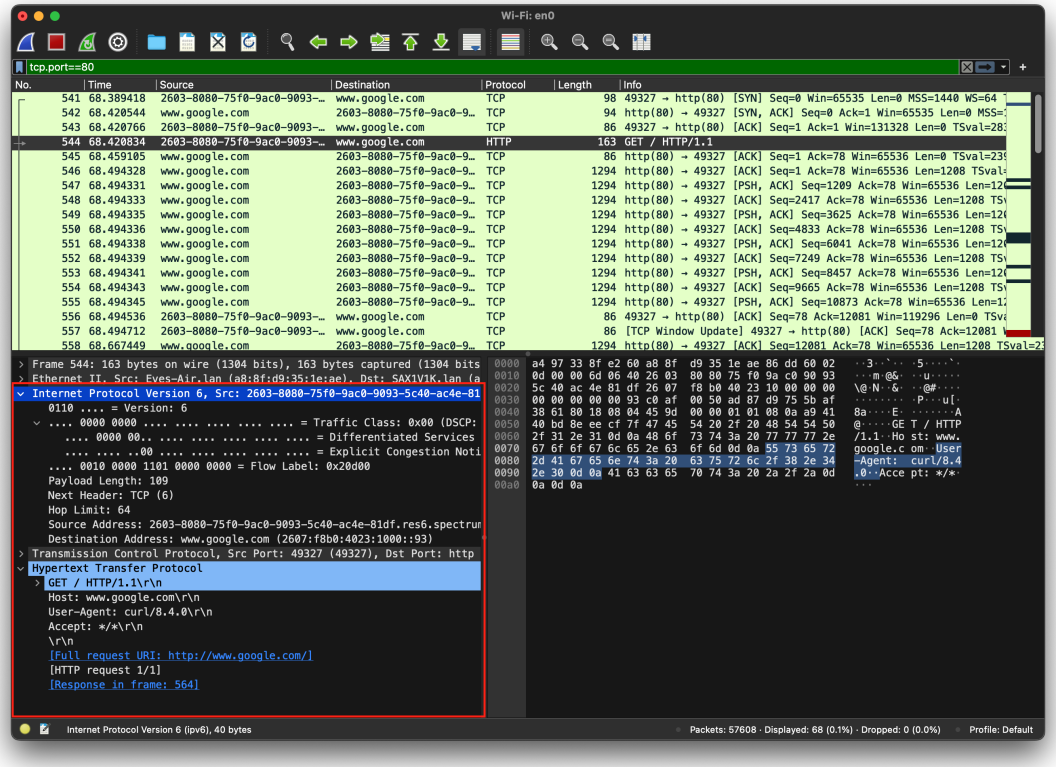
- The first Wireshark block is 'Frame'. This is not a protocol, it is a record that describes overall information about the packet, including when it was captured and how many bits long it is.



- The second block is 'Ethernet'. Note that you may have taken a trace on a computer using 802.11 (WiFi) yet still see an Ethernet block instead of an 802.11 block. Why? It happens because we asked Wireshark to capture traffic in Ethernet format on the capture options, so it converted the real 802.11 header into a pseudo-Ethernet header.

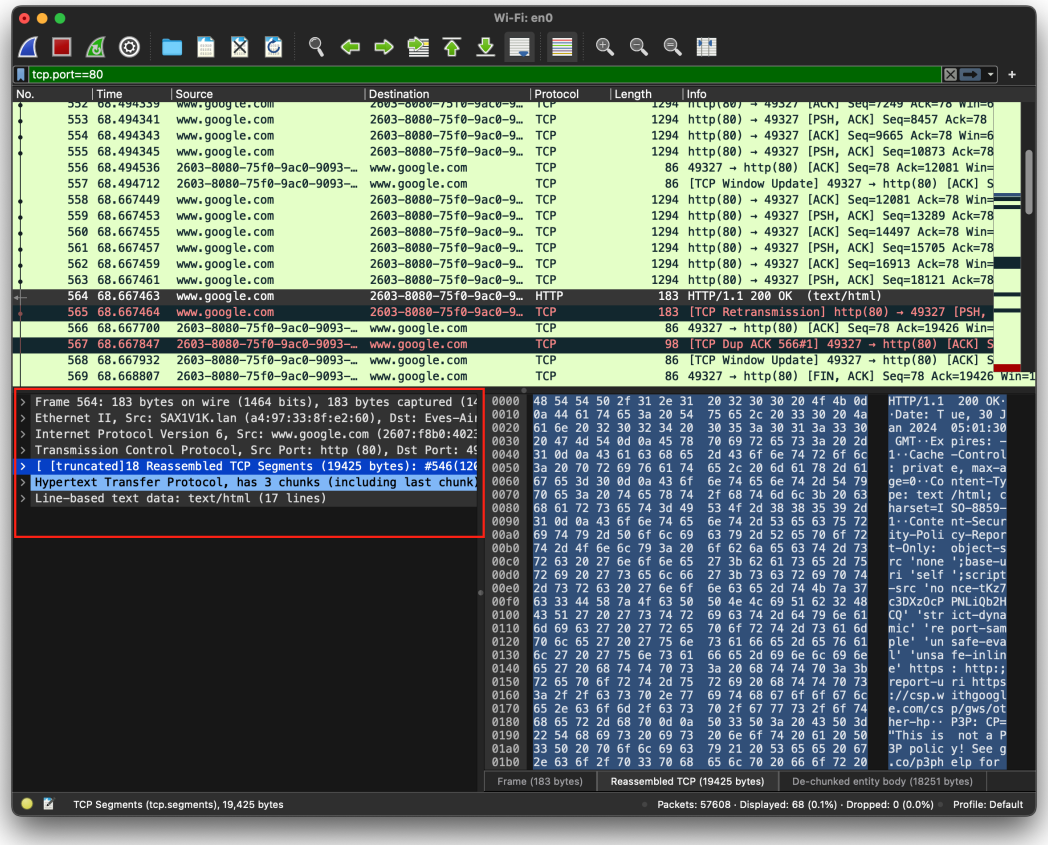


- Then come IP, TCP, and HTTP, which are just as we wanted. Note that the order is from the bottom of the protocol stack upwards. This is because as packets are passed down the stack, the header information of the lower layer protocol is added to the front of the information from the higher layer protocol. That is, the lower layer protocols come first in the packet 'on the wire'.

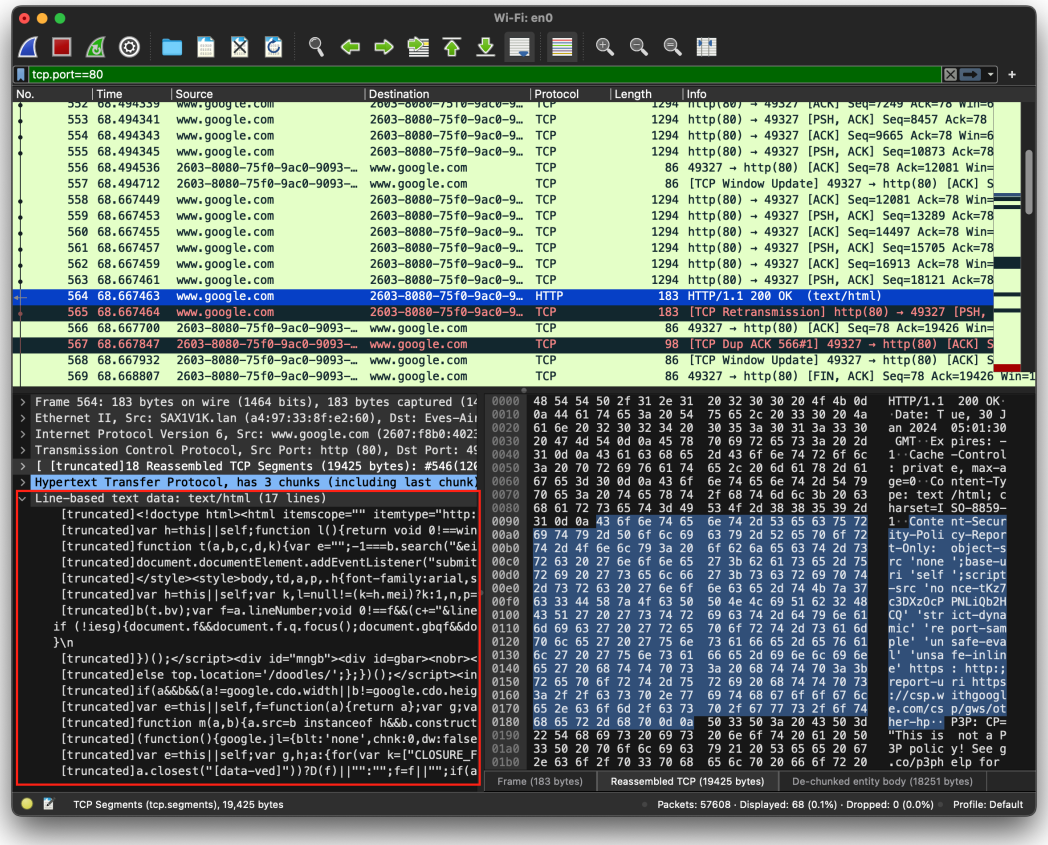


Now find another HTTP packet, the response from the server to your computer, and look at the structure of this packet for the differences compared to the HTTP GET packet. This packet should have ?200 OK? in the Info field, denoting a successful fetch. In your trace, there should be two extra blocks in the detail panel.

- The first extra block says '[11 reassembled TCP segments ...]'. Details in your capture will vary, but this block is describing more than the packet itself. Most likely, the web response was sent across the network as a series of packets that were put together after they arrived at the computer. The packet labeled HTTP is the last packet in the web response, and the block lists packets that are joined together to obtain the complete web response. Each of these packets is shown as having protocol TCP even though the packets carry part of an HTTP response. Only the final packet is shown as having protocol HTTP when the complete HTTP message may be understood, and it lists the packets that are joined together to make the HTTP response.



- The second extra block says ‘Line-based text data ...’. Details in your capture will vary, but this block is describing the contents of the web page that was fetched. In our case it is of type text/html, though it could easily have been text/xml, image/jpeg, or many other types. As with the Frame record, this is not a true protocol. Instead, it is a description of packet contents that Wireshark is producing to help us understand the network traffic.



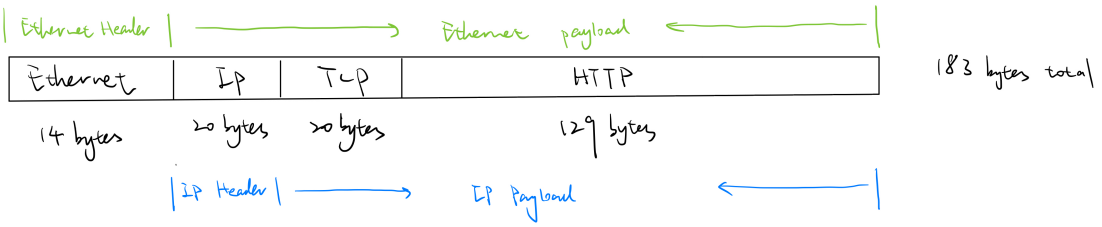
Turn In: Screen capture your Wireshark packet structure screen.

Step 3: Packet Structure

To show your understanding of packet structure, draw a figure of an HTTP GET packet that shows the position and size in bytes of the TCP, IP and Ethernet protocol headers. Your figure can simply show the overall packet as a long, thin rectangle. Leftmost elements are the first sent on the wire. On this drawing, show the range of the Ethernet header and the Ethernet payload that IP passed to Ethernet to send over the network. To show the nesting structure of protocol layers, note the range of the IP header and the IP payload. You may have questions about the fields in each protocol as you look at them. We will explore these protocols and fields in detail in future labs. To work out sizes, observe that when you click on a protocol block in the middle panel (the block itself, not the '+' expander) then Wireshark will highlight the bytes it corresponds to in the packet in the low- er panel and display the length at the bottom of the window. For instance, clicking on the IP version 4 header of a packet in our trace shows us that the length is 20 bytes. (Your trace will be different if it is IPv6, and may be different even with IPv4 depending on various options.) You may also use the overall packet size shown in the Length column or Frame detail block.

Turn In: Hand in your packet drawing.

Solution: [1]

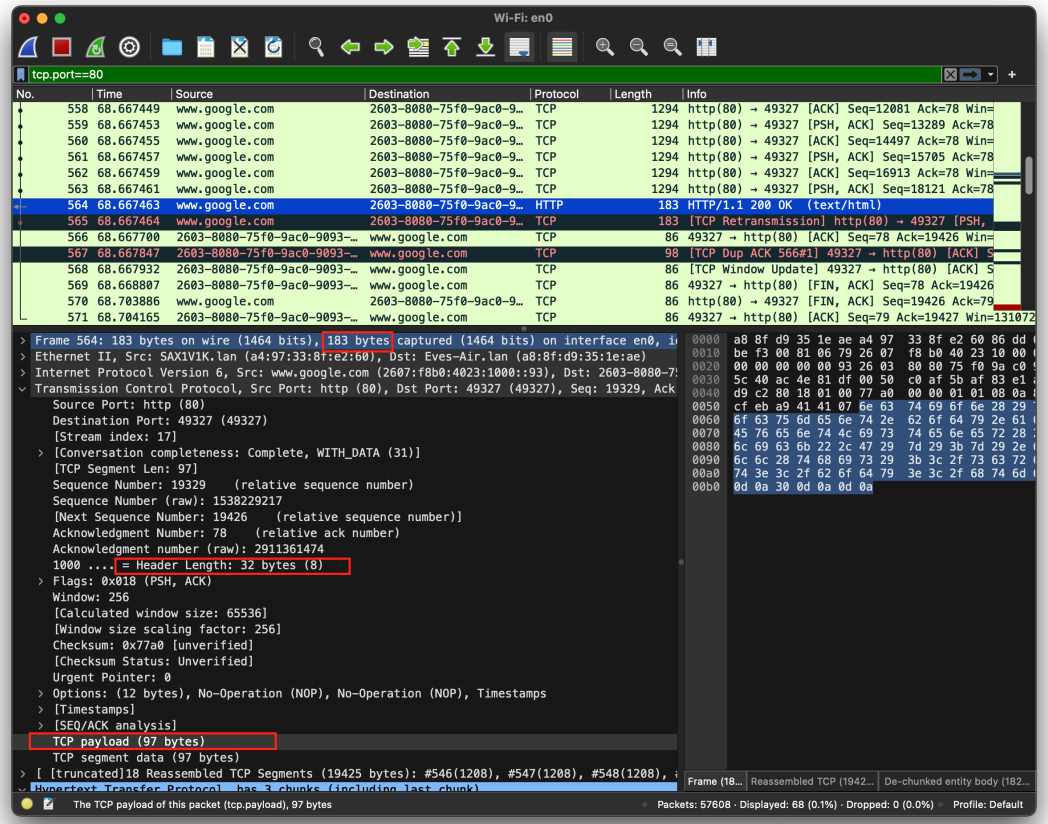


Step 4: Protocol Overhead

Estimate the download protocol overhead, or percentage of the download bytes taken up by protocol overhead. To do this, consider HTTP data (headers and message) to be useful data for the network to carry, and lower layer headers (TCP, IP, and Ethernet) to be the overhead. We would like this overhead to be small, so that most bits are used to carry content that applications care about. To work this out, first look at only the packets in the download direction for a single web fetch. You might sort on the Destination column to find them. The packets should start with a short TCP packet described as a SYN ACK, which is the beginning of a connection. They will be followed by mostly longer packets in the middle (of roughly 1 to 1.5KB), of which the last one is an HTTP packet. This is the main portion of the download. And they will likely end with a short TCP packet that is part of ending the connection. For each packet, you can inspect how much overhead it has in the form of Ethernet / IP / TCP headers, and how much useful HTTP data it carries in the TCP payload. You may also look at the HTTP packet in Wireshark to learn how much data is in the TCP payloads over all download packets.

Turn In: Your estimate of download protocol overhead as defined above. Tell us whether you find this overhead to be significant.

Solution:



From the wireshark I can get:

$$\begin{aligned}
&\text{Total packet size} = 183 \text{ bytes} \\
&\text{Ethernet Header} = 14 \text{ bytes (Typically for Ethernet II)} \\
&\text{Ipv6 Header} = 40 \text{ bytes (as per the standard size)} \\
&\text{TCP header length} = 32 \text{ bytes} \\
&\text{TCP payload} = 97 \text{ bytes}
\end{aligned}$$

$$\begin{aligned}
\text{Total Header Size} &= \text{Ethernet Header} + \text{IPv6 Header} + \text{TCP Header} \\
&= 14 + 40 + 32 = 86 \text{ bytes} \\
\text{Total Useful Data} &= 97 \text{ bytes} \\
\therefore \text{Overhead Percentage} &= \frac{\text{Total Header Size}}{\text{Total Packet Size}} \times 100 \\
&= \frac{86}{183} \times 100 = 47\%
\end{aligned}$$

Step 5: Demultiplexing

When an Ethernet frame arrives at a computer, the Ethernet layer must hand the packet that it contains to the next higher layer to be processed. The act of finding the right higher layer to process received packets is called demultiplexing. We know that in our case the higher layer is IP. But how does the Ethernet protocol

know this? After all, the higher-layer could have been another protocol entirely (such as ARP). We have the same issue at the IP layer ? IP must be able to determine that the contents of IP message is a TCP packet so that it can hand it to the TCP protocol to process. The answer is that protocols use information in their header known as a ?demultiplexing key? to determine the higher layer. Look at the Ethernet and IP headers of a download packet in detail to answer the following questions:

1) Which Ethernet header field is the demultiplexing key that tells it the next higher layer is `IP`? What value is used in this field to indicate `IP`?

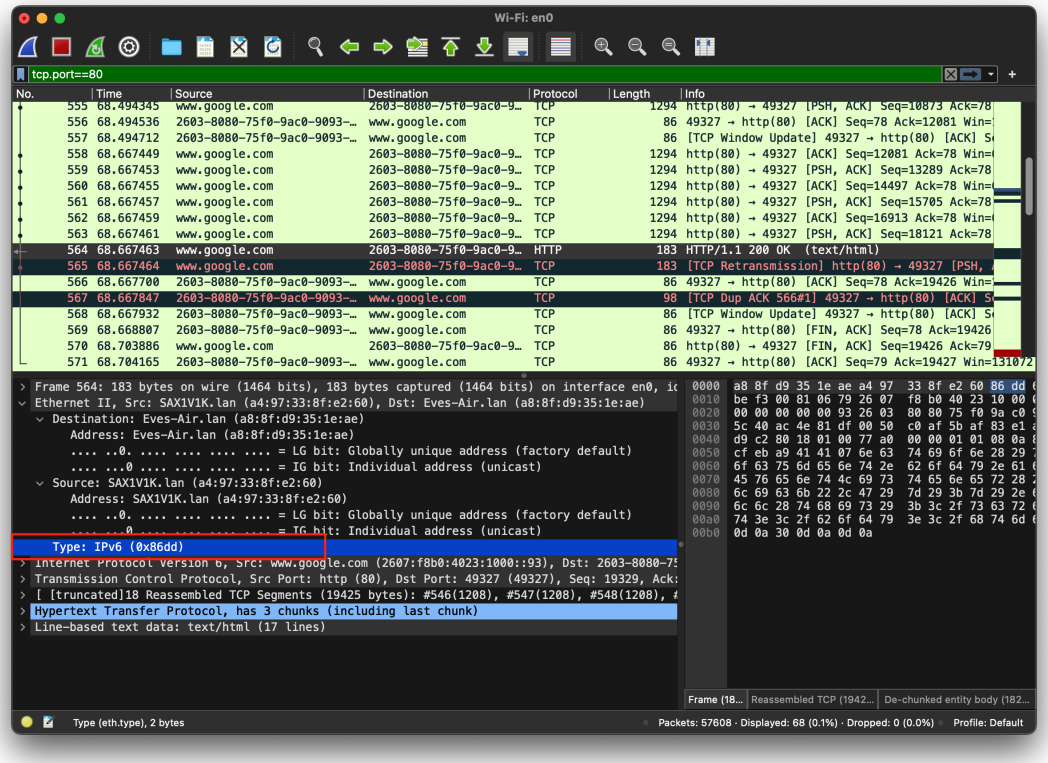
2) Which IP header field is the demultiplexing key that tells it the next higher layer is `TCP`? What value is used in this field to indicate `TCP`?

Turn In: Hand in your answers to the above questions.

Solution:

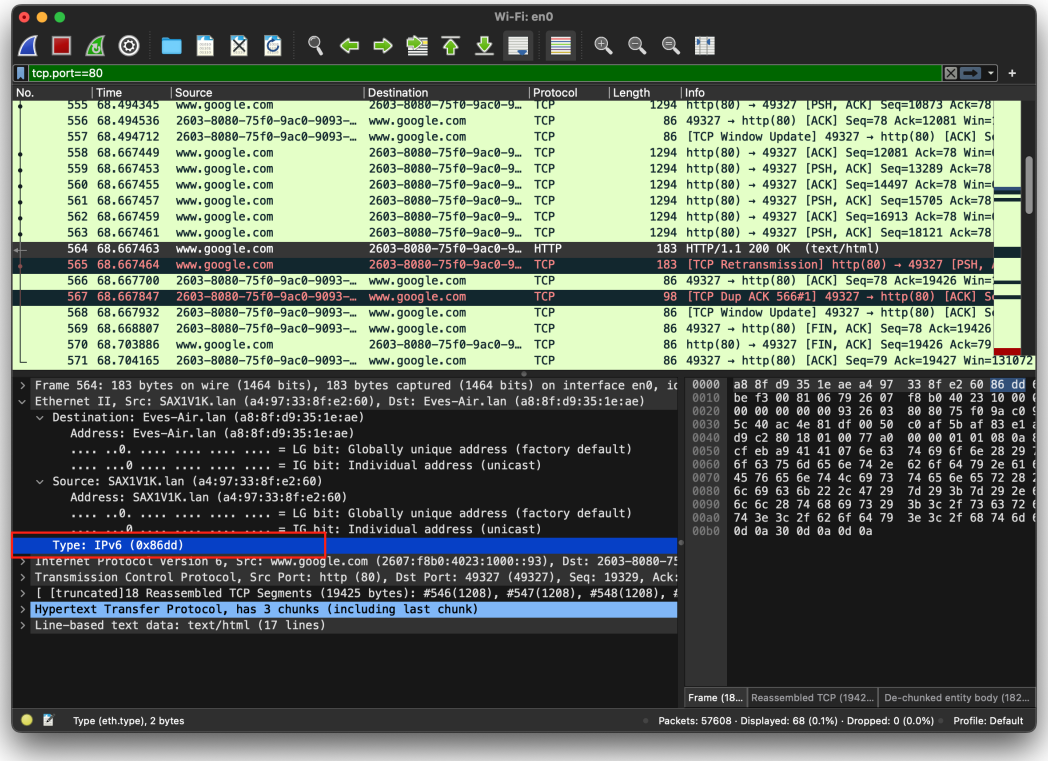
1) Field: The demultiplexing key in the Ethernet header that indicates the next higher layer protocol is the Type field.

   Value for IP: In this frame, the Type field value is 0x86dd, which indicates that the next higher layer is IPv6. This value is used to demultiplex the Ethernet frame to the correct higher layer protocol, in this case, IPv6.



2) Field: Within the IP header (specifically IPv6 in this case), the demultiplexing key that indicates the next higher layer protocol is the Next Header field.

Value for TCP: The Next Header field value in this packet is 6, which signifies that the next higher layer protocol is TCP. This value ensures that the IPv6 payload is correctly handed off to the TCP protocol for further processing.



Exercise 2 : Wireshark IPv4

The goal of this exercise is to become familiar with IPv4 (Internet Protocol version 4). This exercise is adapted from the IPv4 Wireshark Lab by David Wetherall.

This exercise uses the Wireshark software tool to capture and examine a packet trace.

This exercise uses `wget` (Linux and Windows) and `curl` (Mac) to fetch web resources.

This exercise uses `traceroute` to find the router level path from your computer to a remote Internet host. `traceroute` is a standard command-line utility for discovering the Internet paths that your computer uses. It is widely used for network troubleshooting. It comes pre-installed on Window and Mac, and can be installed using your package manager on Linux. On Windows, it is called `tracert`. It has various options, but simply issuing the command `traceroute www.uwa.edu.au` will cause your computer to find and print the path to the remote computer (here `www.uwa.edu.au`).

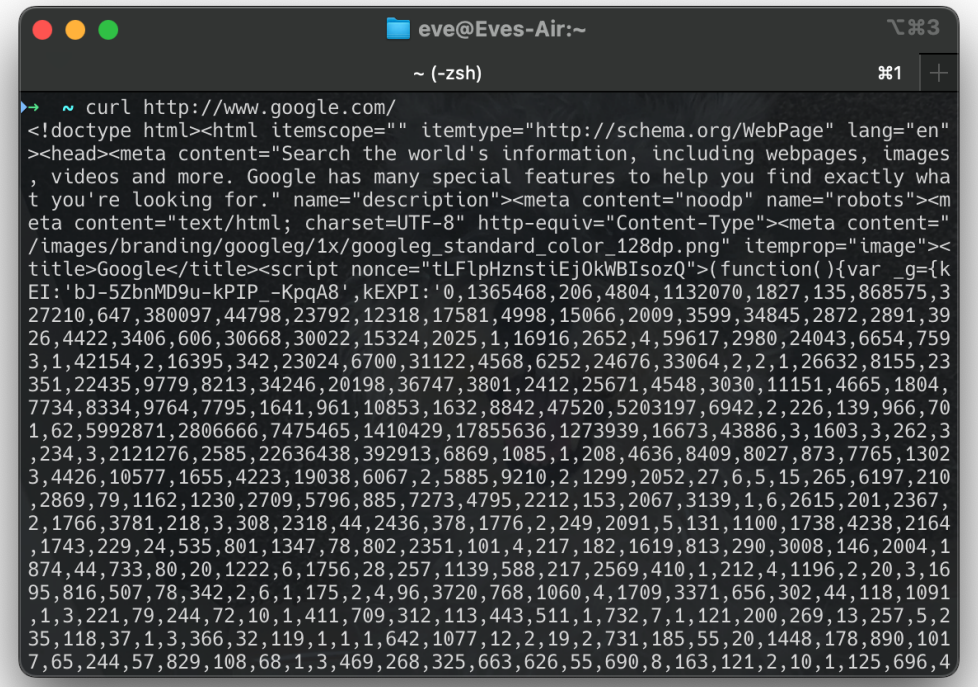
Perform each of the steps below.

Step 1: Capture a Trace

Proceed as follows to capture a trace of network traffic. We want this trace to look at the protocol structure of packets. A simple Web fetch of a URL from a server of your choice to your computer, which is the client, will serve as traffic.

1) Pick a URL at a remote server and fetch it with `wget` or `curl`. For example, `wget http://www.google.com` or `curl http://www.google.com`. This will fetch the resource and either write it to a file (wget) or to the screen (curl). You are checking to see that the fetch works and retrieves some content. With wget you want a single response with status code 200 OK. If the fetch does not work then try a different URL; if no URLs seem to work then debug your use of `wget/curl` or your Internet connectivity.

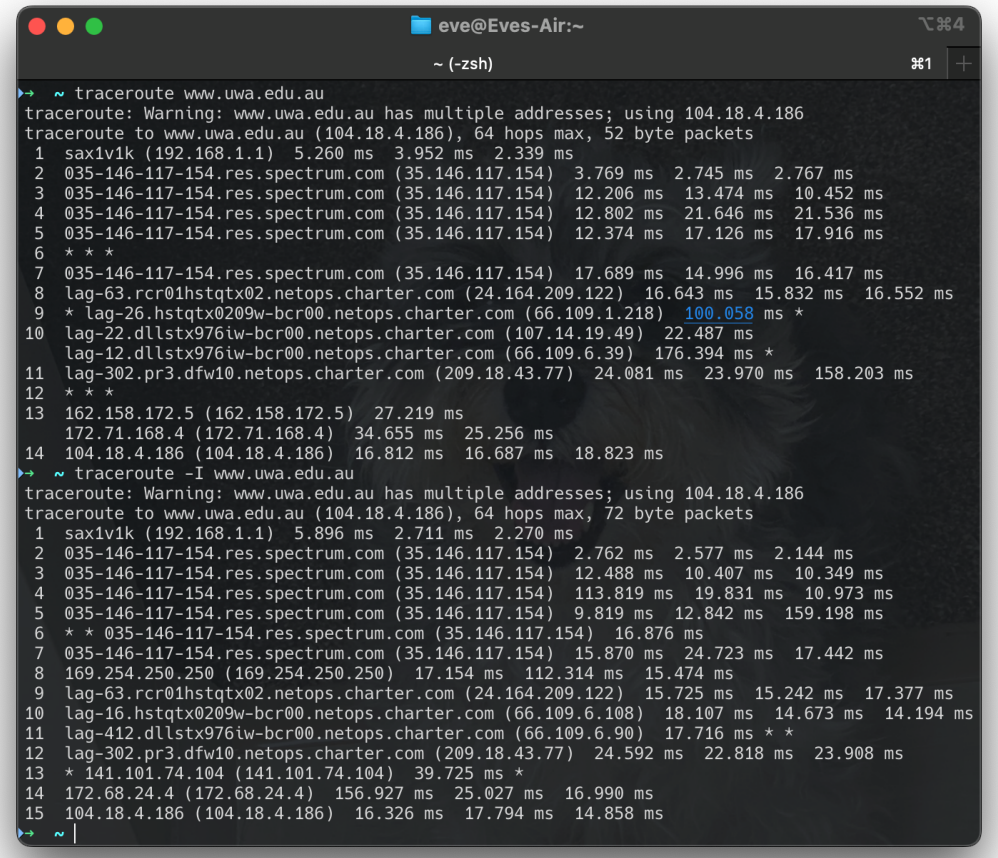
   Solution:



2) Close unnecessary browser tabs and windows. By minimizing browser activity you will stop your computer from fetching unnecessary web content and avoid incidental traffic in the trace.

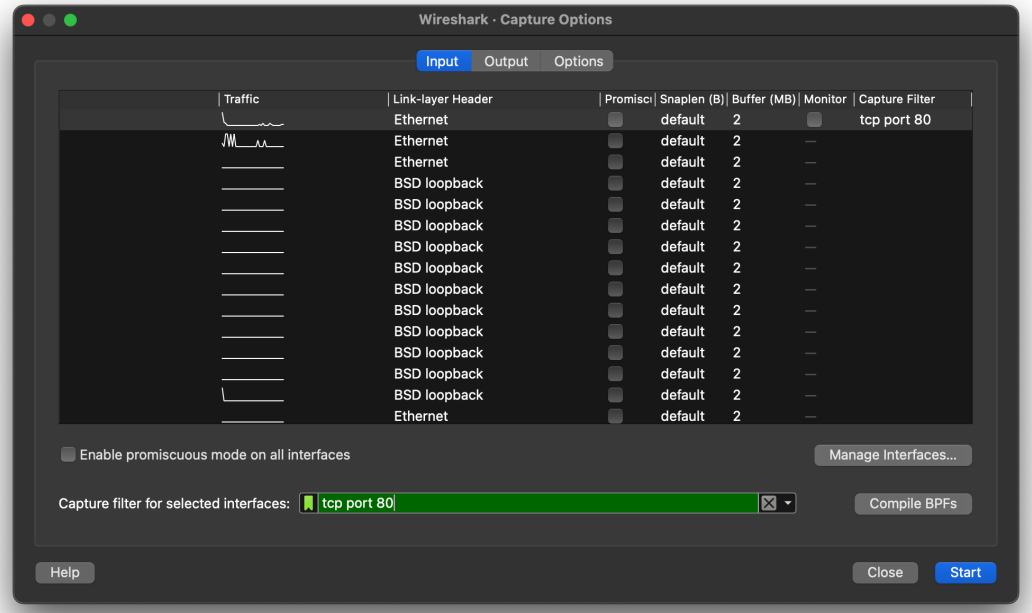
3) Perform a `traceroute` to the same remote server to check that you can discover information about the network path. On Windows, type, e.g., tracert www.uwa.edu.au. On Linux / Mac, type, e.g., `traceroute www.uwa.edu.au`. If you are on Linux / Mac and behind a NAT (as most home users or virtual machine users) then use the ?I option (that was a capital i) to `traceroute`, e.g., `traceroute ?I www.uwa.edu.au`. This will cause traceroute to send ICMP probes like `tracert` instead of its usual UDP probes; ICMP probes are better able to pass through NAT boxes. Save the output as you will need it for later steps. Note that `traceroute` may take up to a minute to run. Each line shows information about the next IP hop from the computer running `traceroute` towards the target destination. The lines with *'s indicate that there was no response from the network to identity that segment of the Internet path. Some unidentified segments are to be expected. However, if `traceroute` is not working correctly then nearly all the path will be *'s. In this case, try a different remote server, experiment with `traceroute`.

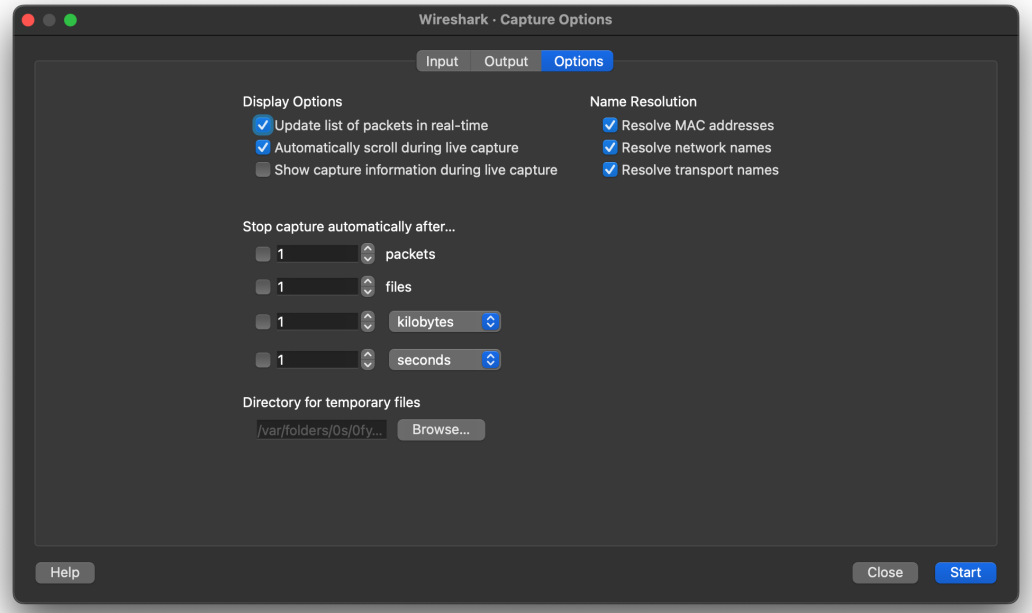
Solution:

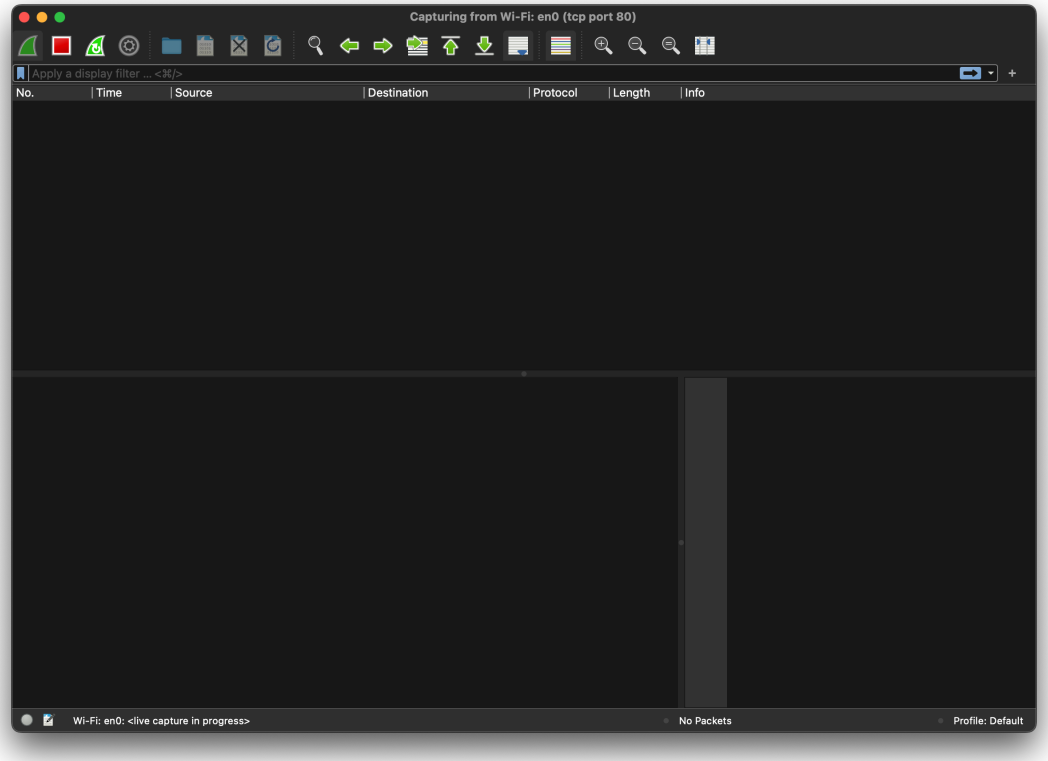


4) Launch Wireshark and start a capture with a filter of tcp port 80 and check enable network name resolution. This filer will record only standard web traffic and not other kinds of packets that your computer may send. The checking will translate the addresses of the computers sending and receiving packets into names, which should help you to recognize whether the packets are going to or from your computer. Select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful. Uncheck capture packets in promiscuous mode. This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets sent to/from your computer. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive. On Wireshark 1.8, the capture filter box is present directly on the options screen, but on Wireshark 1.9, you set a capture filter by double clicking on the interface.

Solution:

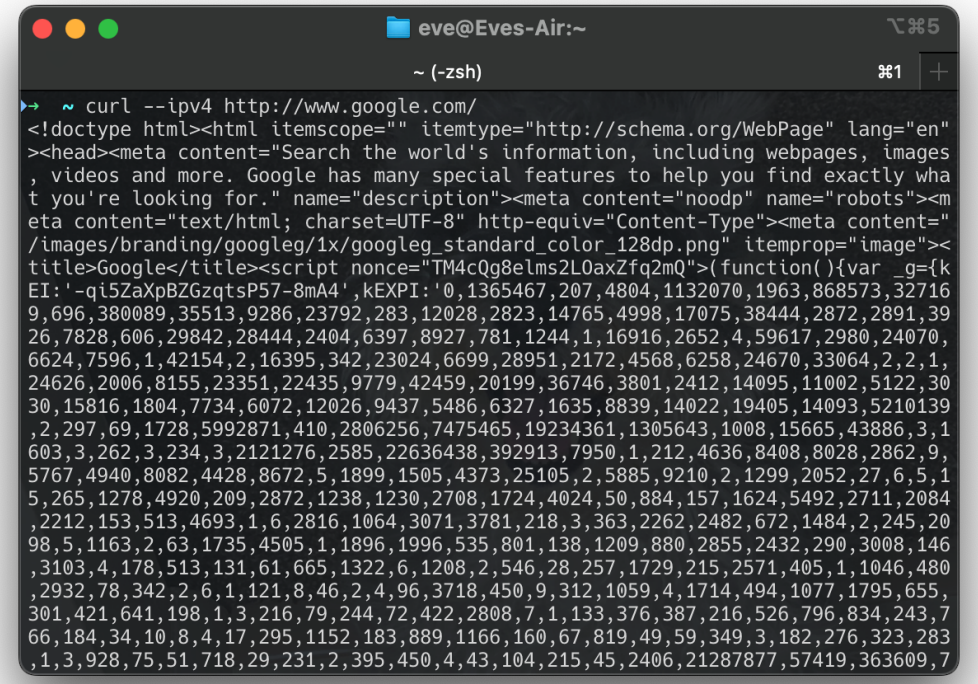

Wireshark · Capture Options
Input
Output
Options
Traffic
Link-layer Header
Promisc
Snaplen (B)
Buffer (MB)
Monitor
Capture Filter
Ethernet
default
2
tcp port 80
BSD loopback
Enable promiscuous mode on all interfaces
Manage Interfaces...
Capture filter for selected interfaces:
tcp port 80
Compile BPFs
Help
Close
Start

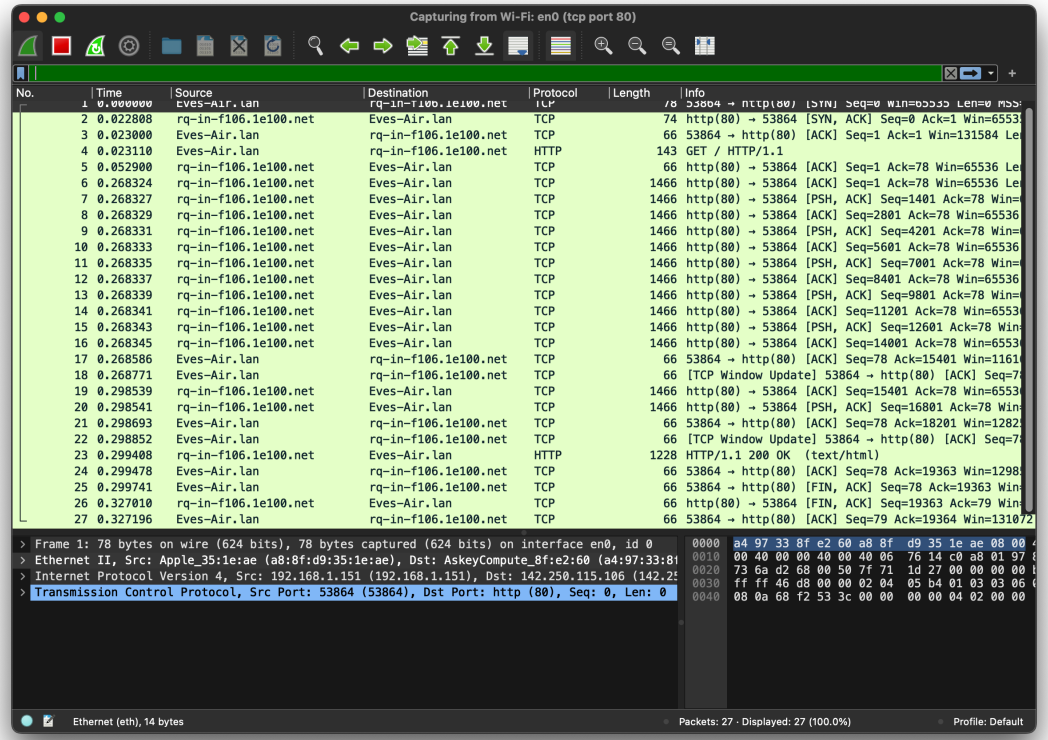

Wireshark · Capture Options
Input
Output
Options
Display Options
Update list of packets in real-time
Automatically scroll during live capture
Show capture information during live capture
Name Resolution
Resolve MAC addresses
Resolve network names
Resolve transport names
Stop capture automatically after...
1
packets
files
kilobytes
seconds
Directory for temporary files
/var/folders/0s/0fy...
Browse...
Help
Close
Start



5) When the capture is started, repeat the web fetch using `wget/curl` above. This time, the packets will be recorded by Wireshark as the content is transferred.

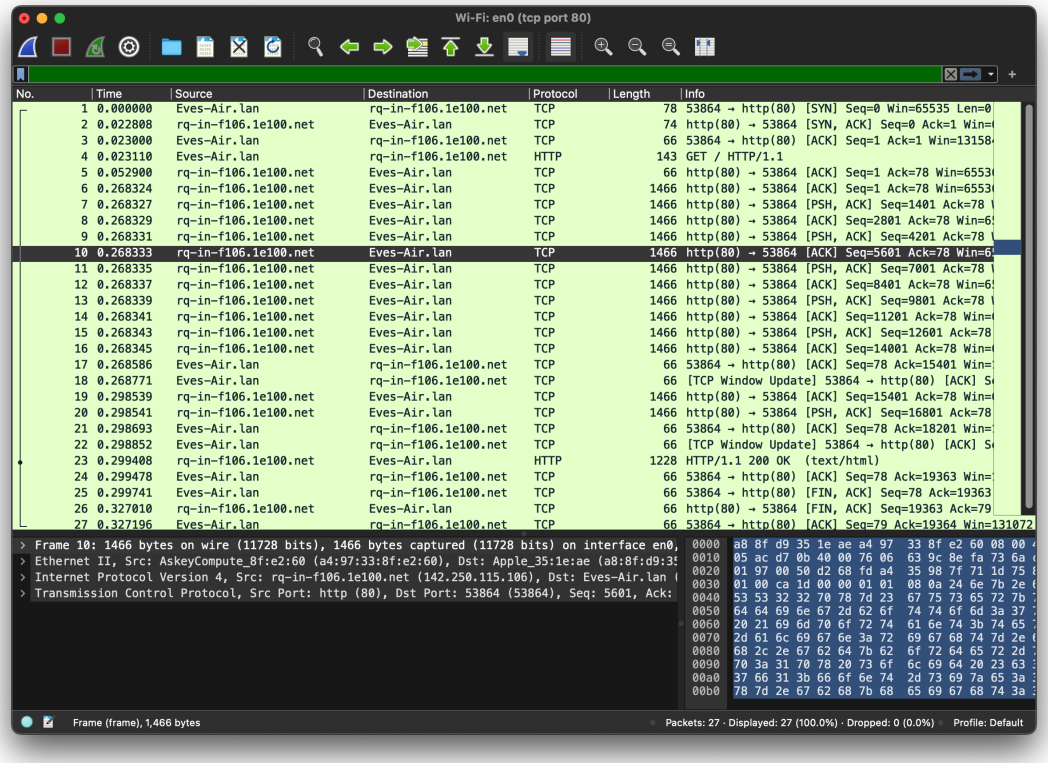
Solution:


eve@Eves-Air:~
~ (-zsh)
⌘1
curl --ipv4 http://www.google.com/
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta content="Search the world's information, including webpages, images, videos and more. Google has many special features to help you find exactly what you're looking for." name="description"><meta content="noodp" name="robots"><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</title><script nonce="TM4cQg8elms2LOaxZfq2mQ">(function(){var _g={kEI:'-qi5ZaXpBZGzqtsP57-8mA4',kEXPI:'0,1365467,207,4804,1132070,1963,868573,327169,696,380089,35513,9286,23792,283,12028,2823,14765,4998,17075,38444,2872,2891,3926,7828,606,29842,28444,2404,6397,8927,781,1244,1,16916,2652,4,59617,2980,24070,6624,7596,1,42154,2,16395,342,23024,6699,28951,2172,4568,6258,24670,33064,2,2,1,24626,2006,8155,23351,22435,9779,42459,20199,36746,3801,2412,14095,11002,5122,3030,15816,1804,7734,6072,12026,9437,5486,6327,1635,8839,14022,19405,14093,5210139,2,297,69,1728,5992871,410,2806256,7475465,19234361,1305643,1008,15665,43886,3,1603,3,262,3,234,3,2121276,2585,22636438,392913,7950,1,212,4636,8408,8028,2862,9,5767,4940,8082,4428,8672,5,1899,1505,4373,25105,2,5885,9210,2,1299,2052,27,6,5,15,265,1278,4920,209,2872,1238,1230,2708,1724,4024,50,884,157,1624,5492,2711,2084,2212,153,513,4693,1,6,2816,1064,3071,3781,218,3,363,2262,2482,672,1484,2,245,2098,5,1163,2,63,1735,4505,1,1896,1996,535,801,138,1209,880,2855,2432,290,3008,146,3103,4,178,513,131,61,665,1322,6,1208,2,546,28,257,1729,215,2571,405,1,1046,480,2932,78,342,2,6,1,121,8,46,2,4,96,3718,450,9,312,1059,4,1714,494,1077,1795,655,301,421,641,198,1,3,216,79,244,72,422,2808,7,1,133,376,387,216,526,796,834,243,766,184,34,10,8,4,17,295,1152,183,889,1166,160,67,819,49,59,349,3,182,276,323,283,1,3,928,75,51,718,29,231,2,395,450,4,43,104,215,45,2406,21287877,57419,363609,7


Capturing from Wi-Fi: en0 (tcp port 80)
No. Time Source Destination Protocol Length Info
2 0.022808 rq-in-f106.1e100.net Eves-Air.lan TCP 74 http(80) → 53864 [SYN, ACK] Seq=0 Ack=1 Win=65535
3 0.023000 Eves-Air.lan rq-in-f106.1e100.net TCP 66 53864 → http(80) [ACK] Seq=1 Ack=1 Win=131584
4 0.023110 Eves-Air.lan rq-in-f106.1e100.net HTTP 143 GET / HTTP/1.1
5 0.052900 rq-in-f106.1e100.net Eves-Air.lan TCP 66 http(80) → 53864 [ACK] Seq=1 Ack=78 Win=65536
6 0.268324 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [ACK] Seq=1 Ack=78 Win=65536
7 0.268327 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [PSH, ACK] Seq=1401 Ack=78
8 0.268329 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [ACK] Seq=2801 Ack=78 Win=65536
9 0.268331 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [PSH, ACK] Seq=4201 Ack=78
10 0.268333 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [ACK] Seq=5601 Ack=78 Win=65536
11 0.268335 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [PSH, ACK] Seq=7001 Ack=78
12 0.268337 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [ACK] Seq=8401 Ack=78 Win=65536
13 0.268339 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [PSH, ACK] Seq=9801 Ack=78
14 0.268341 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [ACK] Seq=11201 Ack=78
15 0.268343 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [PSH, ACK] Seq=12601 Ack=78
16 0.268345 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [PSH, ACK] Seq=14001 Ack=78
17 0.268586 Eves-Air.lan rq-in-f106.1e100.net TCP 66 53864 → http(80) [ACK] Seq=78 Ack=15401 Win=11616
18 0.268771 Eves-Air.lan rq-in-f106.1e100.net TCP 66 [TCP Window Update] 53864 → http(80) [ACK] Seq=78
19 0.298539 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [ACK] Seq=15401 Ack=78 Win=65536
20 0.298541 rq-in-f106.1e100.net Eves-Air.lan TCP 1466 http(80) → 53864 [PSH, ACK] Seq=16801 Ack=78
21 0.298693 Eves-Air.lan rq-in-f106.1e100.net TCP 66 53864 → http(80) [ACK] Seq=78 Ack=18201 Win=128256
22 0.298852 Eves-Air.lan rq-in-f106.1e100.net TCP 66 [TCP Window Update] 53864 → http(80) [ACK] Seq=78
23 0.299408 rq-in-f106.1e100.net Eves-Air.lan HTTP 1228 HTTP/1.1 200 OK (text/html)
24 0.299478 Eves-Air.lan rq-in-f106.1e100.net TCP 66 53864 → http(80) [ACK] Seq=78 Ack=19363 Win=129856
25 0.299741 Eves-Air.lan rq-in-f106.1e100.net TCP 66 53864 → http(80) [FIN, ACK] Seq=78 Ack=19363
26 0.327010 rq-in-f106.1e100.net Eves-Air.lan TCP 66 http(80) → 53864 [FIN, ACK] Seq=19363 Ack=79
27 0.327196 Eves-Air.lan rq-in-f106.1e100.net TCP 66 53864 → http(80) [ACK] Seq=79 Ack=19364 Win=131072
Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface en0, id 0
Ethernet II, Src: Apple_35:1e:ae (a8:8f:d9:35:1e:ae), Dst: AskeyCompute_8f:e2:60 (a4:97:33:8f
Internet Protocol Version 4, Src: 192.168.1.151 (192.168.1.151), Dst: 142.250.115.106 (142.25
Transmission Control Protocol, Src Port: 53864 (53864), Dst Port: http (80), Seq: 0, Len: 0
0000 a4 97 33 8f e2 60 a8 8f d9 35 1e ae 08 00
0010 00 40 00 00 40 00 40 06 76 14 c0 a8 01 97
0020 73 6a d2 68 00 50 7f 71 1d 27 00 00 00 00
0030 ff ff 46 d8 00 00 02 04 05 b4 01 03 03 06
0040 08 0a 68 f2 53 3c 00 00 00 00 04 02 00 00
Ethernet (eth), 14 bytes
Packets: 27 · Displayed: 27 (100.0%)
Profile: Default

6) After the fetch is successful, return to Wireshark and use the menus or buttons to stop the trace. If you have succeeded, the upper Wireshark window will show multiple packets, and most likely it will be full. How many packets are captured will depend on the size of the web page, but there should be at least 8 packets in the trace, and typically 20-100, and many of these packets will be colored green.

Solution:



Turn In: Screen capture your Wireshark trace and your `traceroute` and turn it in.

Step 2: Inspect the Trace

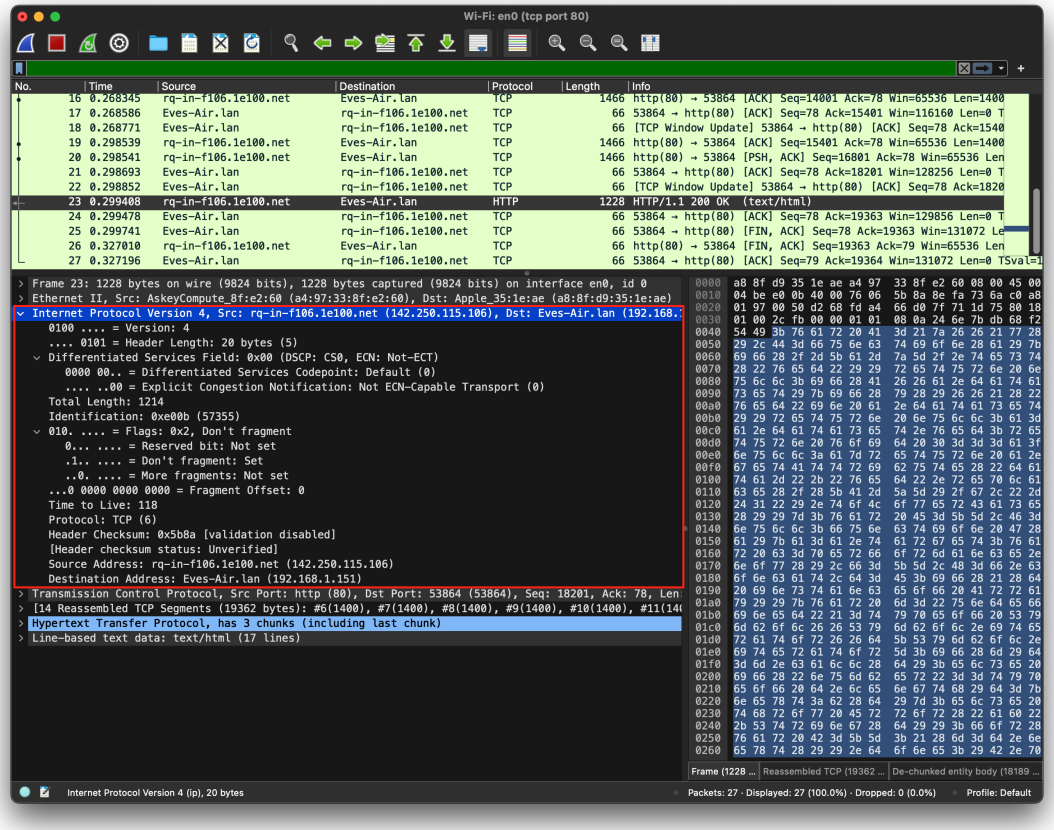
Select any packet in the trace and expand the IP header fields (using the '+' expander or icon) to see the details. You can simply click on a packet to select it (in the top panel). You will see details of its structure (in the middle panel) and the bytes that make up the packet (in the bottom panel). Our interest is the IP header, and you may ignore the other higher and lower layer protocols. When you click on parts of the IP header, you will see the bytes that correspond to the part highlighted in the bottom panel. Let us go over the fields in turn:

- The version field is set to 4. This is IPv4 after all.
- Then there is the header length field. Observe by looking at the bytes selected in the packet data that version and header length are both packed into a single byte.
- The Differentiated Services field contains bit flags to indicate whether the packet should be handled with quality of service and congestion indications at routers.

- Then there is the Total Length field.
- Next is the Identification field, which is used for grouping fragments, when a large IP packet is sent as multiple smaller pieces called fragments. It is followed by the Flags and the Fragment offset fields, which also relate to fragmentation. Observe they share bytes.
- Then there is the Time to live or TTL field, followed by the Protocol field.
- Next comes the header checksum. Is your header checksum carrying 0 and flagged as incorrect for IP packets sent from your computer to the remote server? On some computers, the operating system software leaves the header checksum blank (zero) for the NIC to compute and fill in as the packet is sent. This is called protocol offloading. It happens after Wireshark sees the packet, which causes Wireshark to believe that the checksum is wrong and flag it with a different color to signal a problem. A similar issue may happen for the TCP checksum. You can remove these false errors if they are occurring by telling Wireshark not to validate the checksums. Select?Preferences? from the Wireshark menus and expand the ?Protocols? area. Look under the list until you come to IPv4. Uncheck ?Validate checksum if possible?. Similarly, you may uncheck checksum validation for TCP if applicable to your case.
- The last fields in the header are the normally the source and destination address. It is possible for there to be IP options, but these are unlikely in standard web traffic.
- The IP header is followed by the IP payload. This makes up the rest of the packet, starting with the next higher layer header, TCP in our case, but not including any link layer trailer (e.g., Ethernet padding).

  Turn In: Screen capture your Wireshark packet structure screen. Modify your screen capture to indicate each of the components of the IPv4 packet.

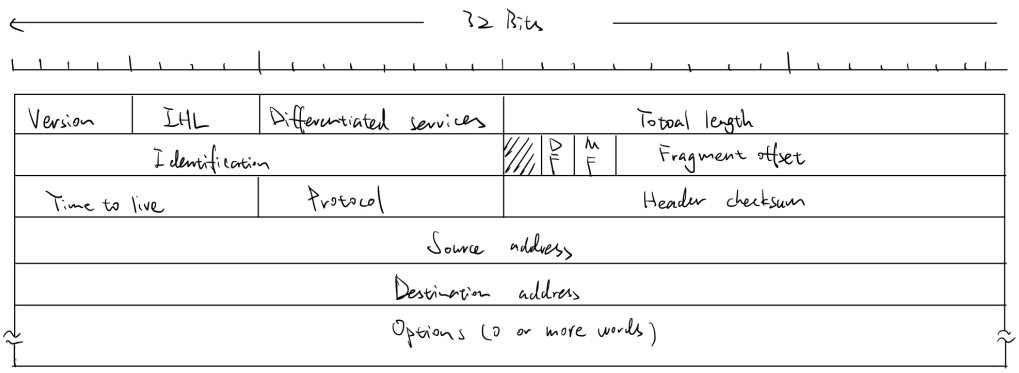
  Solution:



Step 3: IP Packet Structure

To show your understanding of IP, sketch a figure of an IP packet you studied. It should show the position and size in bytes of the IP header fields as you can observe using Wireshark. Since you cannot easily determine sub-byte sizes, group any IP fields that are packed into the same bytes. Your figure can simply show the frame as a long, thin rectangle. Try not to look at the figure of an IPv4 packet in your text; check it afterwards to note and investigate any differences.

Solution: [1]

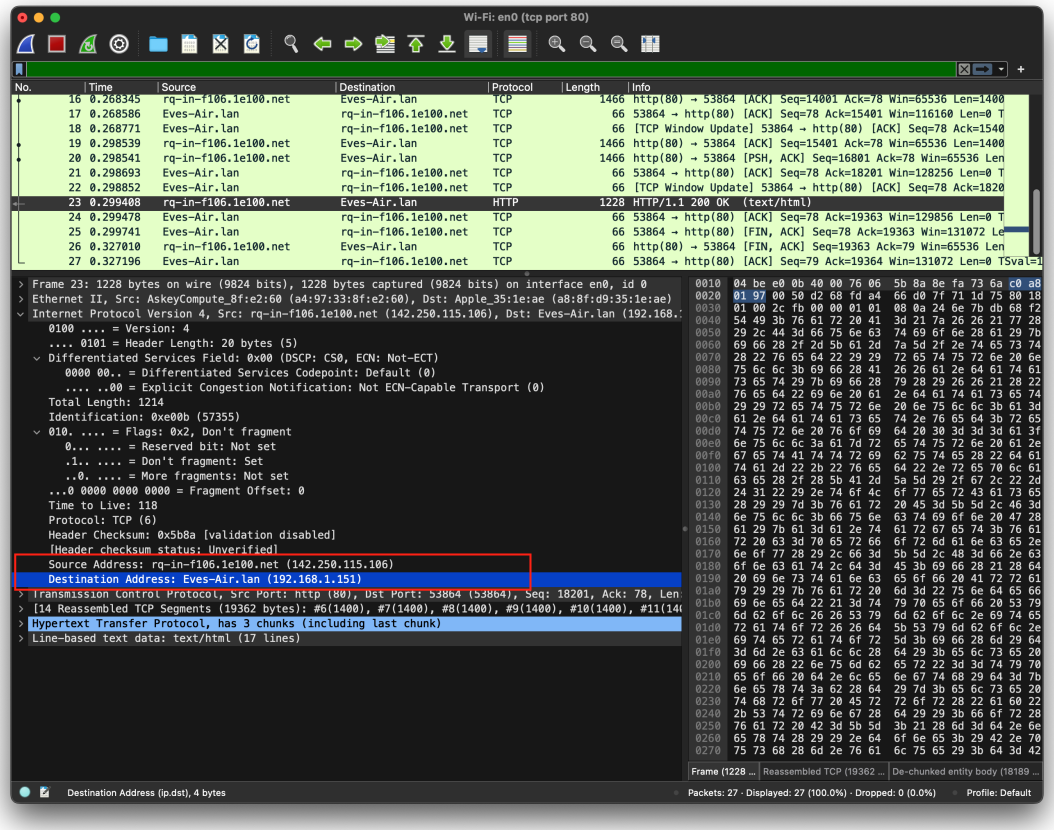


To work out sizes, observe that when you click on a protocol block in the middle panel (the block itself, not the '+' expander) Wireshark will highlight the corresponding bytes in the packet in the lower panel, and display the length at the bottom of the window. You may also use the overall packet size shown in the Length column or Frame detail block. Note that this method will not tell you sub-byte positions.

By looking at the IP packets in your trace, answer these questions:

1) What are the IP addresses of your computer and the remote server?

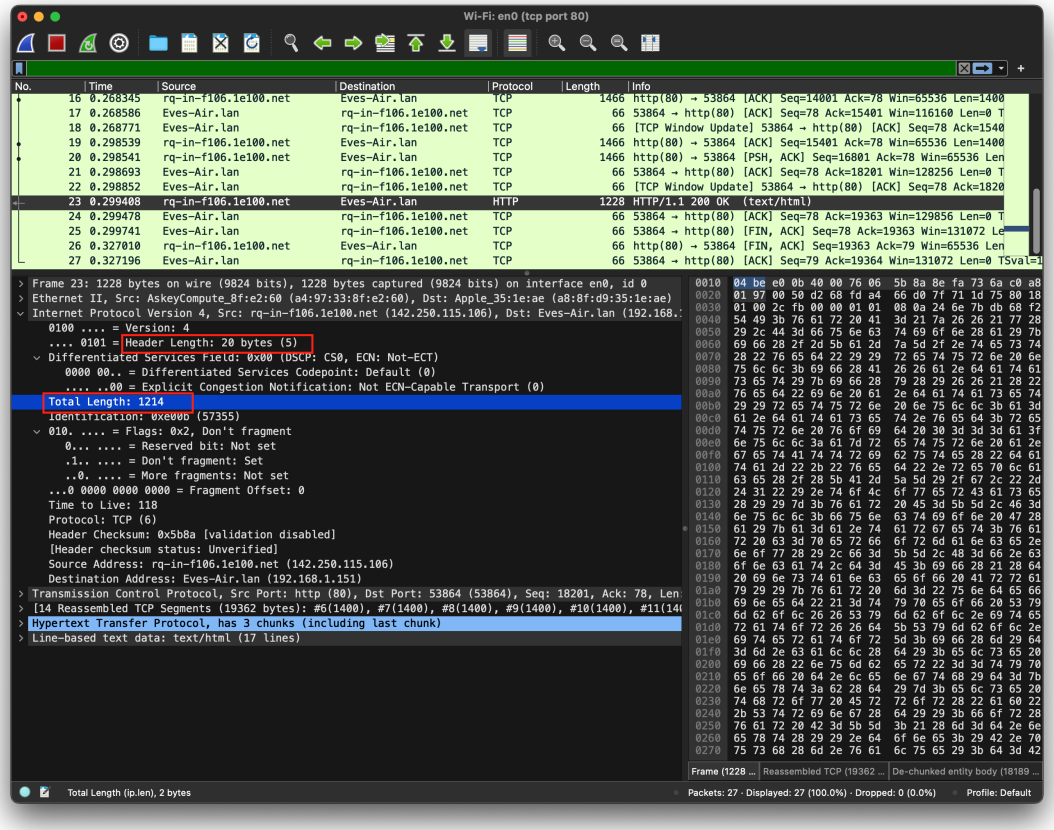
Solution:



The IP addresses of my computer is: 192.168.1.151.
The IP addresses of the remote server is 142.250.115.106 (source address) belonging to rq-in-f106.1e100.net, which is a public internet address of Google.

2) Does the Total Length field include the IP header plus IP payload, or just the IP payload?

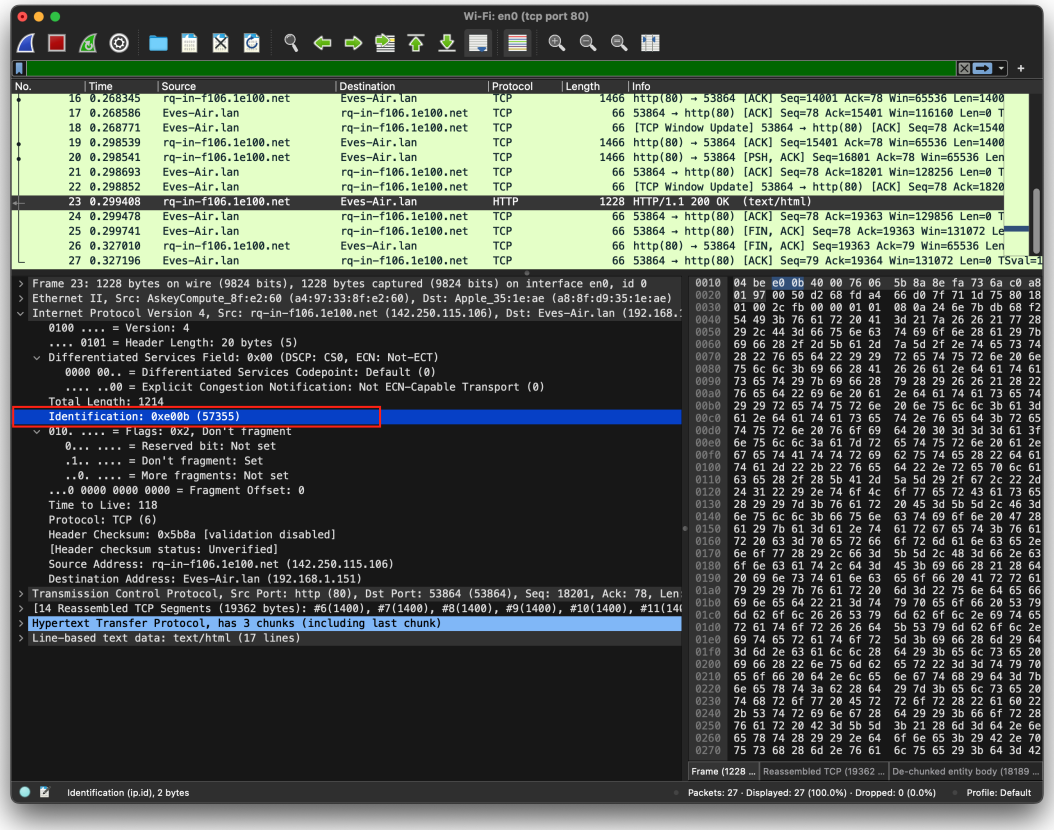
Solution:



The Total Length field value is 1214, which includes both the IP header and the IP payload. In this example, the IP header length is 20 bytes, making the IP payload length 1194 bytes.

3) How does the value of the Identification field change or stay the same for different packets? For instance, does it hold the same value for all packets in a TCP connection or does it differ for each packet? Is it the same in both directions? Can you see any pattern if the value does change?

Solution:



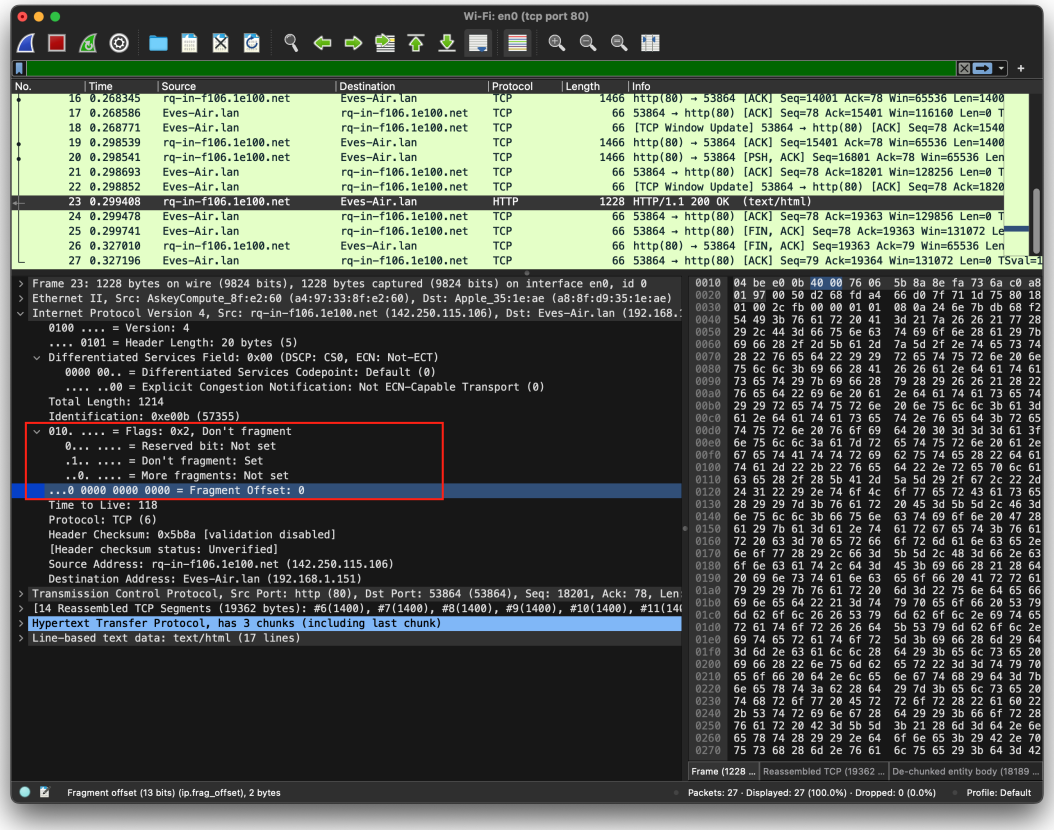
The Identification field value is 0xe00b (57355). For different packets within the same TCP connection, this value can change to assist the receiving end in reassembling fragmented packets. Since only one packet's information is provided here, cannot directly observe the pattern of change. However, typically, each IP packet would have a unique identification value, unless the packets are being fragmented.

4) What is the initial value of the TTL field for packets sent from your computer? Is it the maximum possible value, or some lower value?

Solution: The TTL field value is 118, which is not the maximum possible value but relatively high. The initial value of TTL depends on the operating system and network configuration, with common starting values including 64, 128, etc.

5) How can you tell from looking at a packet that it has not been fragmented? Most often IP packets in normal operation are not fragmented. But the receiver must have a way to be sure. Hint: you may need to read your text to confirm a guess.

Solution:



The packet's non-fragmentation can be determined by checking the "Don't Fragment" flag and the Fragment Offset field. In this example, the "Don't Fragment" flag is set (.1.. .... = Don't fragment: Set), and the Fragment Offset field value is 0 (...0 0000 0000 0000 = Fragment Offset: 0), indicating the packet has not been fragmented.

6) What is the length of the IP Header and how is this encoded in the header length field? Hint: notice that only 4 bits are used for this field, as the version takes up the other 4 bits of the byte. You may guess and check your text.

   Solution: The IP header length is 20 bytes (.... 0101 = Header Length: 20 bytes). This length is encoded in the header length field, with the value of 5 indicating the IP header occupies 5 32-bit words (5x4=20 bytes).

Turn In: Hand in your drawing of an IP packet and the answers to the questions above.

Step 4: Internet Paths

The source and destination IP addresses in an IP packet denote the endpoints of an Internet path, not the IP routers on the network path the packet travels from the source to the destination. traceroute is a utility for discovering this path. It works by eliciting responses (ICMP TTL Exceeded messages) from the router 1 hop away from the source towards the destination, then 2 hops away from the source, then 3 hops, and so forth until the destination is reached. The responses will identify the IP address of the router. The output from traceroute normally prints the information for one hop per line, including the measured round trip times

and IP address and DNS names of the router. The DNS name is handy for working out the organization to which the router belongs. Since traceroute takes advantage of common router implementations, there is no guarantee that it will work for all routers along the path, and it is usual to see ?*? responses when it fails for some portions of the path.

Using the traceroute output, sketch a drawing of the network path. If you are using the supplied trace, note that we have provided the corresponding traceroute output as a separate file. Show your computer (lefthand side) and the remote server (righthand side), both with IP addresses, as well as the routers along the path between them numbered by their distance on hops from the start of the path. You can find the IP address of your computer and the remote server on the packets in the trace that you captured. The output of traceroute will tell you the hop number for each router.

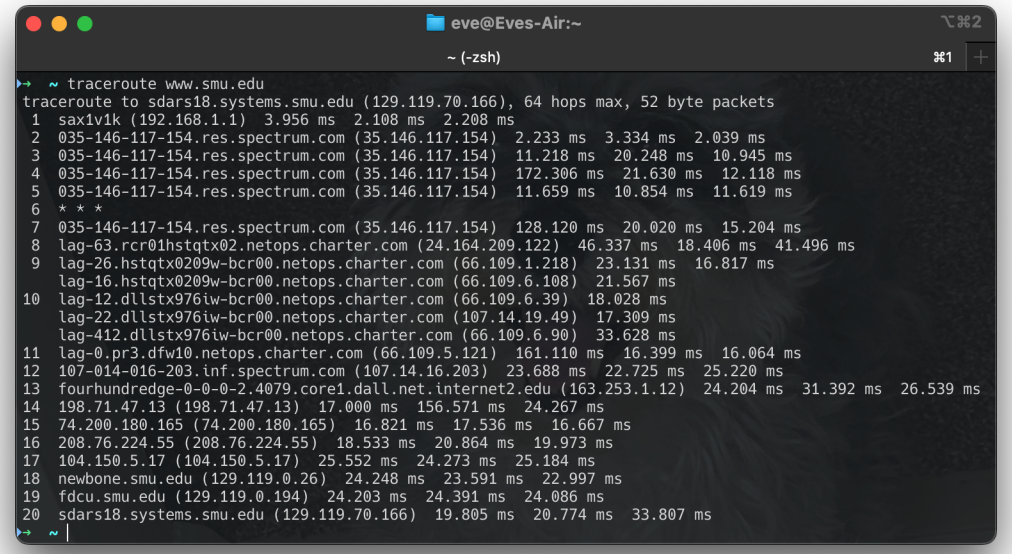
To finish your drawing, label the routers along the path with the name of the real-world organization to which they belong. To do this, you will need to interpret the domain names of the routers given by traceroute. If you are unsure, label the routers with the domain name of what you take to be the organization. Ignore or leave blank any routers for which there is no domain name (or no IP address).

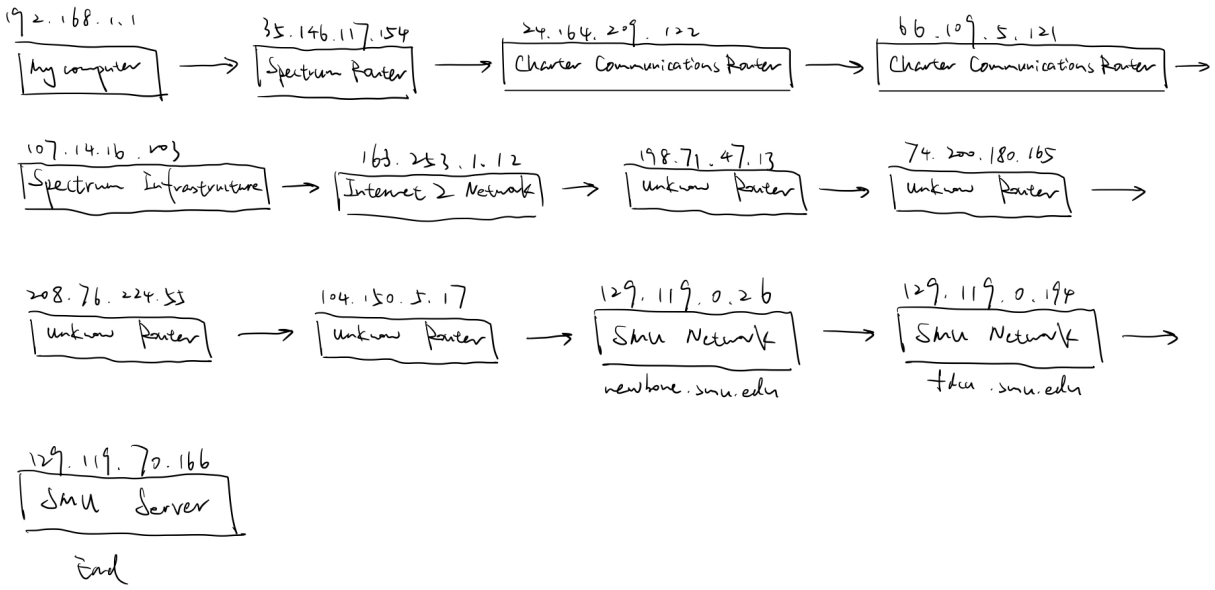
This is not an exact science, so we will give some examples. Suppose that traceroute identifies a router along the path by the domain name arouter.cac.washington.edu. Normally, we can ignore at least the first part of the name, since it identifies different computers in the same organization and not different organizations. Thus we can ignore at least ?arouter? in the domain name. For generic top-level domains, like ?.com? and ?.edu?, the last two domains give the domain name of the organization. So for our example, it is ?washington.edu?. To translate this domain name into the real-world name of an organization, we might search for it on the web. You will quickly find that washington.edu is the University of Washington. This means that ?cac? portion is an internal structure in the University of Washington, and not important for the organization name. You would write?*University of Washington* on your figure for any routers with domain names of the form `*.washington.edu`.

Alternatively, consider a router with a domain name like arouter.syd.aarnet.net.au. Again, we ignore at least the `arouter` part as indicating a computer within a specific organization. For country-code top-level domains like '.au' (for Australia) the last three domains in the name will normally give the organization. In this case the organization?s domain name is aarnet.net.au. Using a web search, we find this domain represents AARNET, Australia?s research and education network. The 'syd' portion is internal structure, and a good guess is that it means the router is located in the Sydney part of AARNET. So for all routers with domain names of the form *.aarnet.net.au, you would write 'AARNET' on your figure. While there are no guarantees, you should be able to reason similarly and at least give the domain name of the organizations near the ends of the path.

Turn In: Hand in your drawing and `traceroute` output.

Solution:





Exercise 3 : Wireshark DNS

The goal of this exercise is to become familiar with the Domain Name System (DNS).

This exercise uses the Wireshark software tool to capture and examine a packet trace.

This exercise uses a web browser to find or fetch pages as a workload. Any web browser will do.

This exercise uses `dig` to issue DNS requests and observe DNS responses. `dig` is a flexible, command-line tool for querying remote DNS servers that replaces the older `nslookup` program. It comes installed on Mac OS. On Window, you can download `dig` from ISC?s BIND web site as part of the bind download. (Note that there may be some dependencies. Check for online instructions to set up *dig* on Windows.)

On Linux, install `dig` with your package manager. It is normally part of a `dnsutils` or `bindutils` package.

Perform each of the steps below.

Step 1: Manual Name Resolution

Before we look at how your computer uses the DNS, we will see how a local nameserver resolves a DNS name, i.e., we will interact with remote nameservers. To do this exercise, you will pretend to be the local nameserver and issue requests to remote nameservers using the `dig` tool.

Pick a domain name to resolve, such as that of your web server. We will use www.smu.edu. Find the IP address of one of the root nameservers by searching the web. For example, the Wikipedia article on root name servers includes the IP address of the root nameservers a through m. Any one of these should do, as they hold replicated information. You need this information to begin the name resolution process, and nameservers are provided with it as part of their configuration.

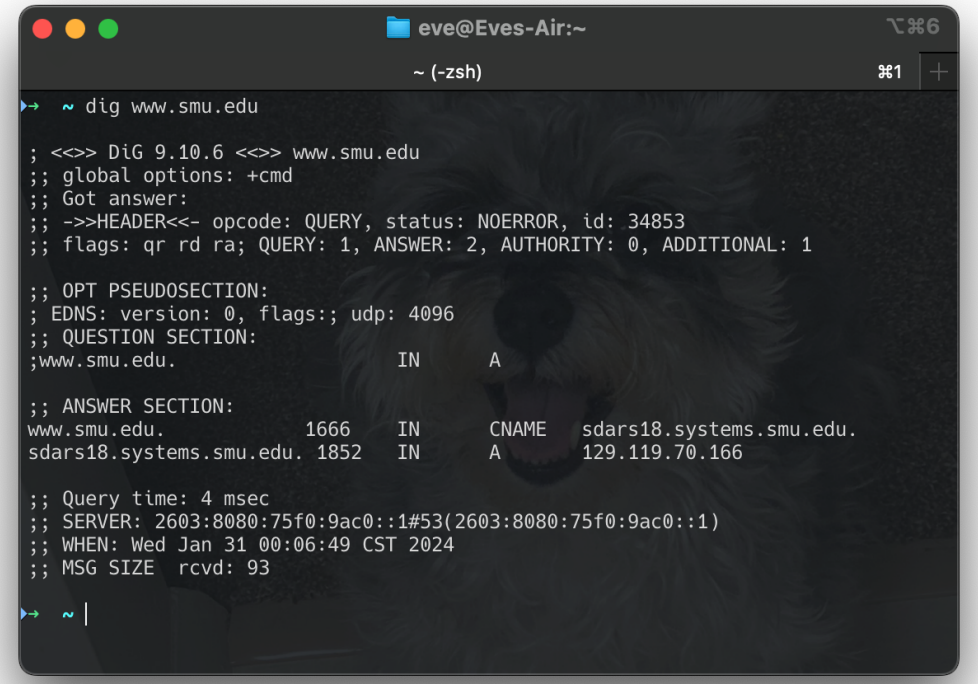
Use dig to issue a request to a root nameserver to perform the first step of the resolution. You are assuming that you have no cached information that will let you begin a resolution below the root. The format of a dig command is dig @aa.bb.cc.dd domainname. It instructs dig to send a request to a nameserver at a given IP address (or name) for the given domain name. The reply from the root does not provide the full name resolution, but it does tell us about nameservers closer to having the information for you to contact.

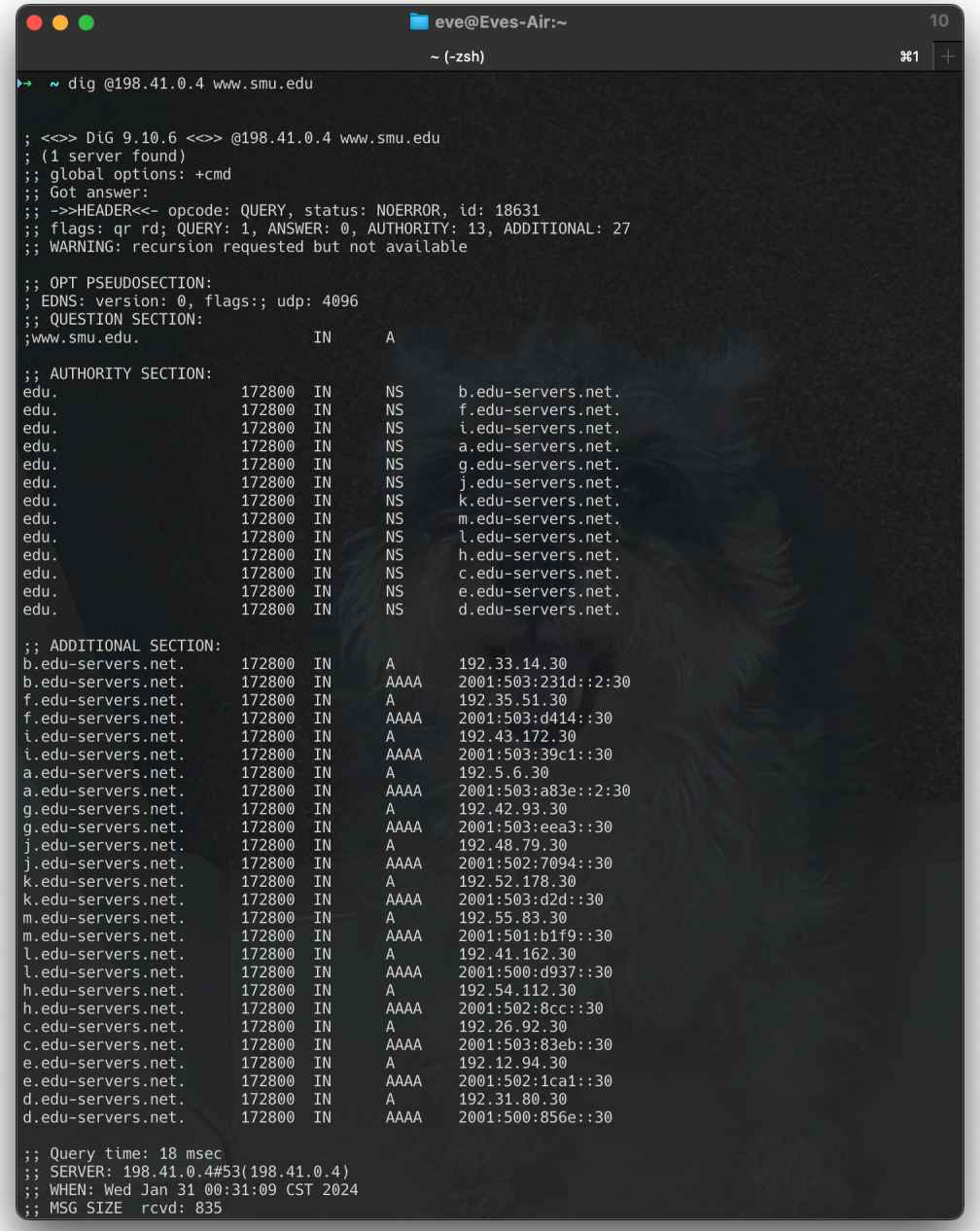
Continue the resolution process with dig until you complete the resolution. When you have alternatives to choose, prefer IPv4 nameservers and select the first one in alphabetical order. If this nameserver has multiple IP addresses then select the numerically smallest IP address. You can complete the resolution without these tie-breaking rules and will likely obtain the same result since the DNS information is replicated. Keep these dig commands handy, as you will repeat them in the next step when you capture a trace.

Draw a figure that shows the sequence of remote nameservers that you contacted and the domain for which they are responsible. Note that future name resolutions are likely to be a much shorter sequence because they can use cached information. For example, if you looked up a domain name in ?.edu? then when you look up a different domain name in ?.edu? you already know the name of the ?.edu? nameserver. Thus you can start there, or even closer to the final nameserver depending on what you have cached; you do not need to start again at the root nameserver.

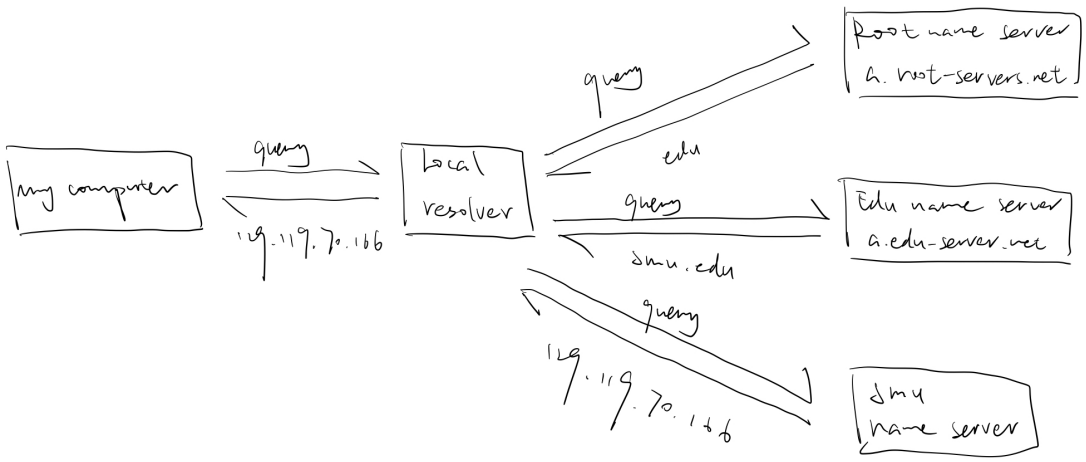
Turn In: Hand in your drawing.

Solution:


eve@Eves-Air:~
⌥⌘6
~ (-zsh)
⌘1
→ ~ dig www.smu.edu
; <<>> DiG 9.10.6 <<>> www.smu.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34853
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.smu.edu. IN A
;; ANSWER SECTION:
www.smu.edu. 1666 IN CNAME sdars18.systems.smu.edu.
sdars18.systems.smu.edu. 1852 IN A 129.119.70.166
;; Query time: 4 msec
;; SERVER: 2603:8080:75f0:9ac0::1#53(2603:8080:75f0:9ac0::1)
;; WHEN: Wed Jan 31 00:06:49 CST 2024
;; MSG SIZE rcvd: 93
→ ~


eve@Eves-Air:~
~ (-zsh)
~ dig @198.41.0.4 www.smu.edu
; <<>> DiG 9.10.6 <<>> @198.41.0.4 www.smu.edu
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18631
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
;; WARNING: recursion requested but not available
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.smu.edu. IN A
;; AUTHORITY SECTION:
edu. 172800 IN NS b.edu-servers.net.
edu. 172800 IN NS f.edu-servers.net.
edu. 172800 IN NS i.edu-servers.net.
edu. 172800 IN NS a.edu-servers.net.
edu. 172800 IN NS g.edu-servers.net.
edu. 172800 IN NS j.edu-servers.net.
edu. 172800 IN NS k.edu-servers.net.
edu. 172800 IN NS m.edu-servers.net.
edu. 172800 IN NS l.edu-servers.net.
edu. 172800 IN NS h.edu-servers.net.
edu. 172800 IN NS c.edu-servers.net.
edu. 172800 IN NS e.edu-servers.net.
edu. 172800 IN NS d.edu-servers.net.
;; ADDITIONAL SECTION:
b.edu-servers.net. 172800 IN A 192.33.14.30
b.edu-servers.net. 172800 IN AAAA 2001:503:231d::2:30
f.edu-servers.net. 172800 IN A 192.35.51.30
f.edu-servers.net. 172800 IN AAAA 2001:503:d414::30
i.edu-servers.net. 172800 IN A 192.43.172.30
i.edu-servers.net. 172800 IN AAAA 2001:503:39c1::30
a.edu-servers.net. 172800 IN A 192.5.6.30
a.edu-servers.net. 172800 IN AAAA 2001:503:a83e::2:30
g.edu-servers.net. 172800 IN A 192.42.93.30
g.edu-servers.net. 172800 IN AAAA 2001:503:eea3::30
j.edu-servers.net. 172800 IN A 192.48.79.30
j.edu-servers.net. 172800 IN AAAA 2001:502:7094::30
k.edu-servers.net. 172800 IN A 192.52.178.30
k.edu-servers.net. 172800 IN AAAA 2001:503:d2d::30
m.edu-servers.net. 172800 IN A 192.55.83.30
m.edu-servers.net. 172800 IN AAAA 2001:501:b1f9::30
l.edu-servers.net. 172800 IN A 192.41.162.30
l.edu-servers.net. 172800 IN AAAA 2001:500:d937::30
h.edu-servers.net. 172800 IN A 192.54.112.30
h.edu-servers.net. 172800 IN AAAA 2001:502:8cc::30
c.edu-servers.net. 172800 IN A 192.26.92.30
c.edu-servers.net. 172800 IN AAAA 2001:503:83eb::30
e.edu-servers.net. 172800 IN A 192.12.94.30
e.edu-servers.net. 172800 IN AAAA 2001:502:1ca1::30
d.edu-servers.net. 172800 IN A 192.31.80.30
d.edu-servers.net. 172800 IN AAAA 2001:500:856e::30
;; Query time: 18 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Wed Jan 31 00:31:09 CST 2024
;; MSG SIZE rcvd: 835

```
eve@Eves-Air:~
~ (-zsh)
➜  ~ dig @192.5.6.30 www.smu.edu

; <<>> DiG 9.10.6 <<>> @192.5.6.30 www.smu.edu
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29792
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 3, ADDITIONAL: 4
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.smu.edu.                   IN      A

;; AUTHORITY SECTION:
smu.edu.                172800  IN      NS      pony.cis.smu.edu.
smu.edu.                172800  IN      NS      xpony.smu.edu.
smu.edu.                172800  IN      NS      epony.smu.edu.

;; ADDITIONAL SECTION:
pony.cis.smu.edu.       172800  IN      A       129.119.64.10
xpony.smu.edu.          172800  IN      A       129.119.64.8
epony.smu.edu.          172800  IN      A       128.42.182.100

;; Query time: 183 msec
;; SERVER: 192.5.6.30#53(192.5.6.30)
;; WHEN: Wed Jan 31 00:33:43 CST 2024
;; MSG SIZE  rcvd: 151
```



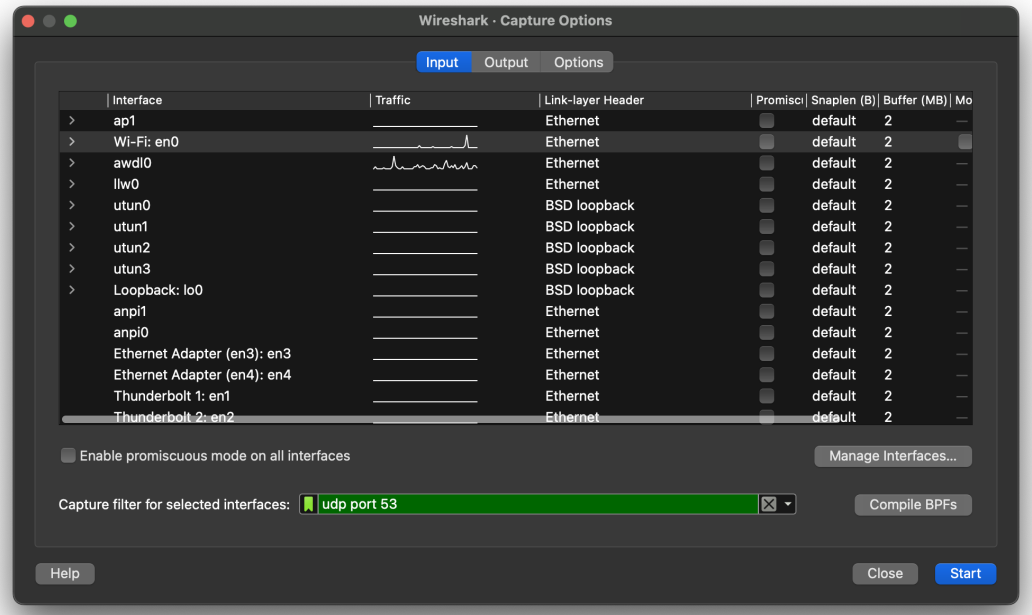
Step 2: Capture a Trace

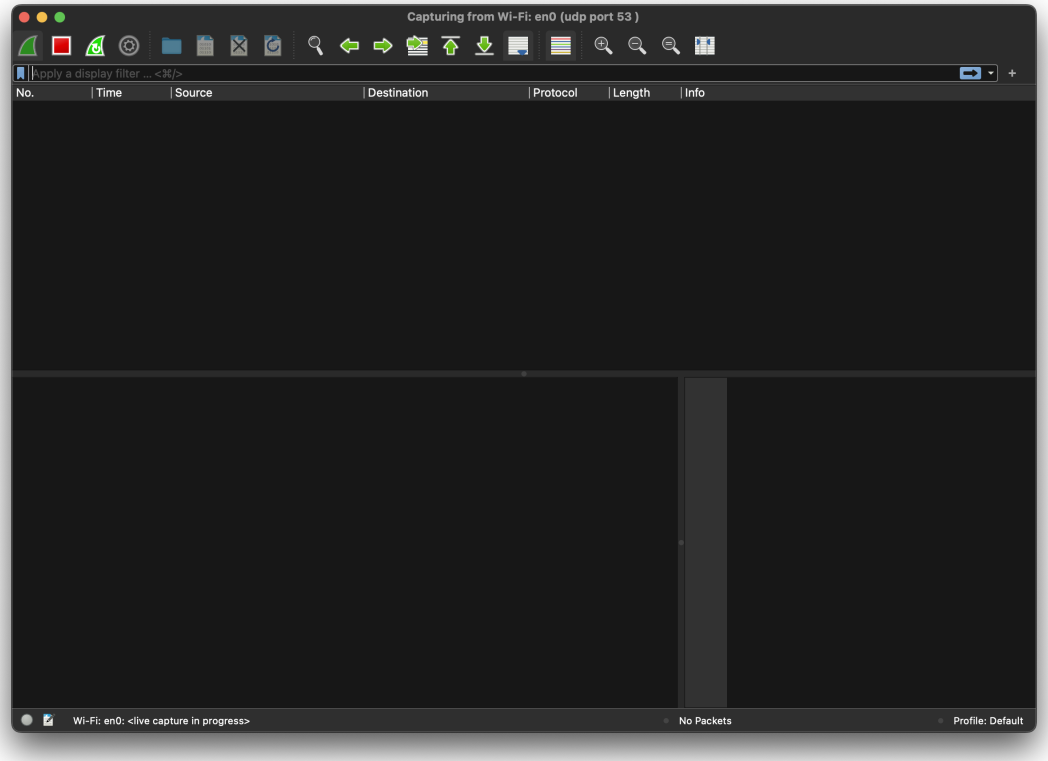
Capture a trace of your browser making DNS requests as follows. Now that we are familiar with the process of name resolution, we will inspect the details of DNS traffic. To generate DNS traffic you will both repeat the dig commands and browse web sites.

1. Close all unnecessary browser tabs and windows. Browsing web sites will generate DNS traffic as your browser resolves domain names to connect to remote servers. We want to minimize browser activity initially so that we capture only the intended DNS traffic.

2. Launch Wireshark and start a capture with a filter of udp port 53. We use this filter because there is no shorthand for DNS, but DNS is normally carried on UDP port 53. Select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful. Uncheck 'capture packets in promiscuous mode'. This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets sent to/from your computer. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive. On Wireshark 1.8, the capture filter box is present directly on the options screen, but on Wireshark 1.9, you set a capture filter by double-clicking on the interface.

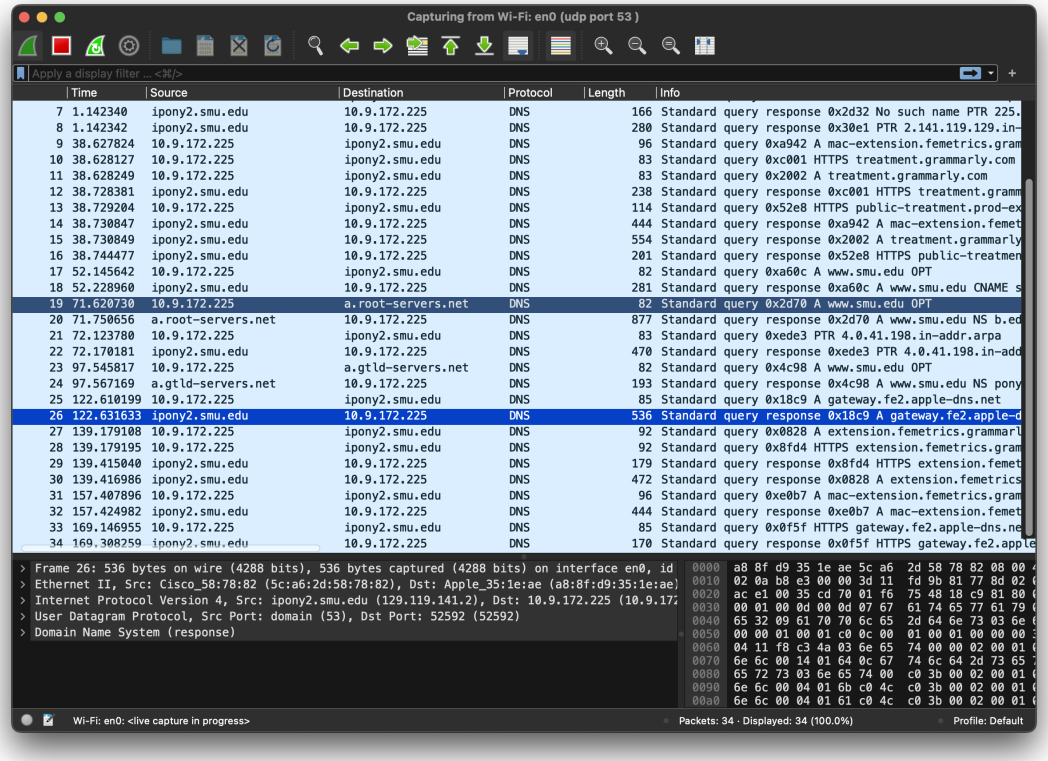
Solution:





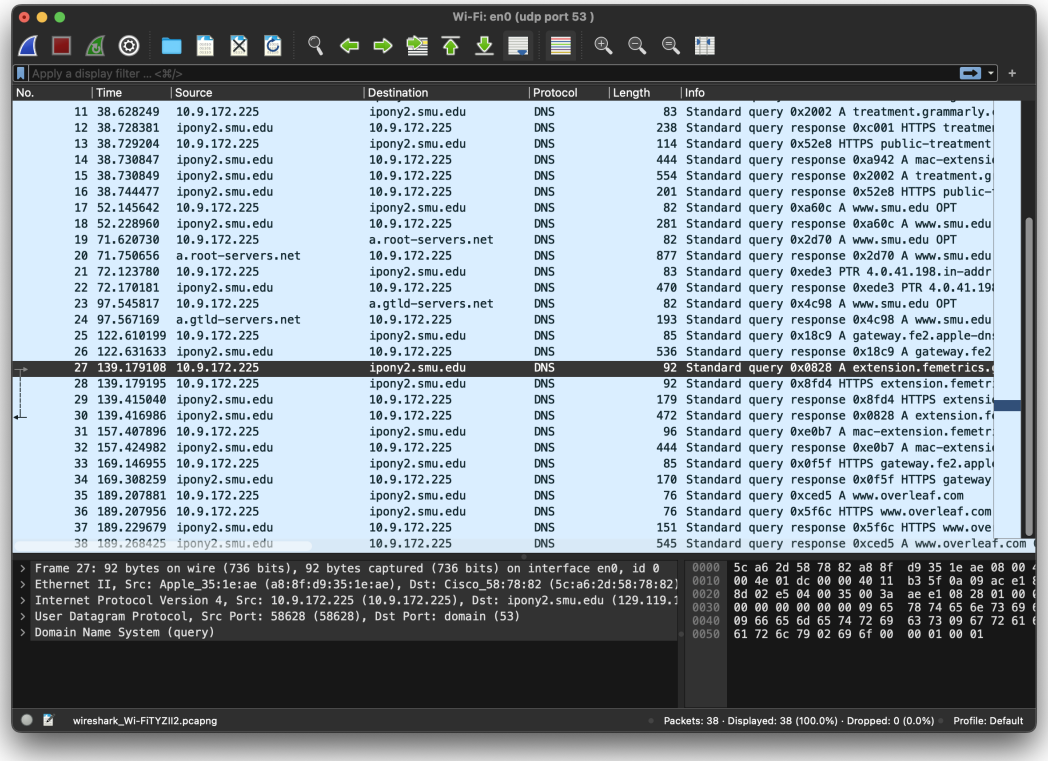
3. Repeat the dig commands from the previous step. This time, you should see the DNS request and reply packets that correspond to your commands captured in the trace window. Note that there may be some background DNS traffic originating from your computer if any process needs to resolve names to make a network connection. We are assuming that there will be little of this traffic so that you can

   Solution:



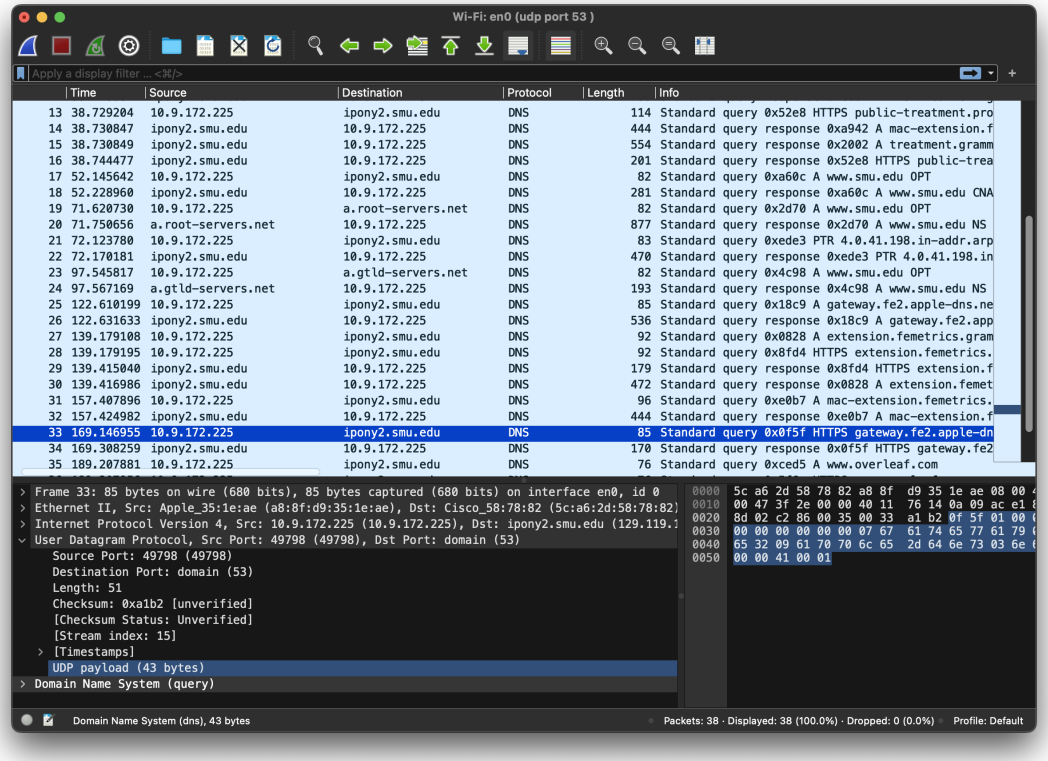
4. Wait 10 seconds, then open your browser and browse a variety of sites. Using your browser will generate DNS traffic as you visit new domains, and also as your browser runs its background tasks such as auto-completion. Unlike the dig traffic, this will be DNS traffic between your computer and the local nameserver.

   Solution:



5. Stop the capture when you have a good sample of DNS traffic. We would like enough traffic to see a variety of behavior. DNS traffic is generated fairly quickly as you browse so it should only take a short while to collect this DNS traffic.

   Solution:

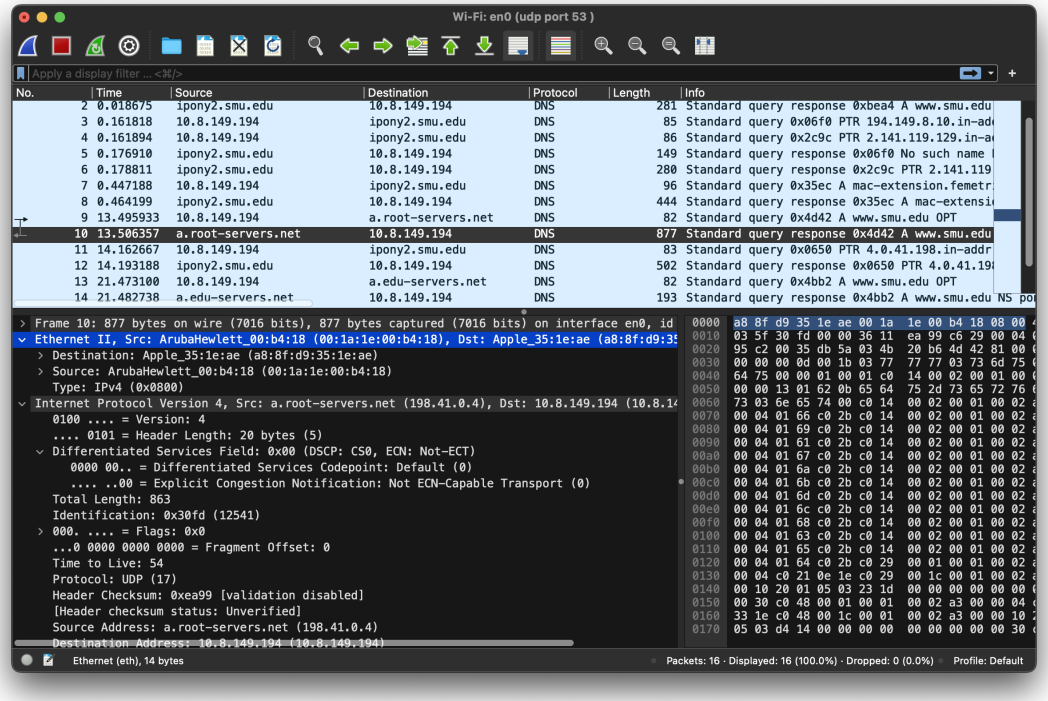


Step 3: Inspect the Trace

To explore the details of DNS packets, select a DNS query expand its Domain Name System block (by using the '+' expander or icon). The first packets should correspond to your dig commands, followed by DNS traffic produced by your browser.

Select the first DNS query that corresponds to your dig commands and expand its DNS block. Likely this query is the first packet in your trace, with the first several packets corresponding to your dig commands, followed by other DNS traffic produced by your browser. To check, see if there are several queries that list the domain you chose in the Info column, each followed by a response. We will use these DNS messages to study the details of the DNS protocol. Sometimes there may be other DNS traffic interspersed with these queries due to background activity; you should ignore these extraneous packets.

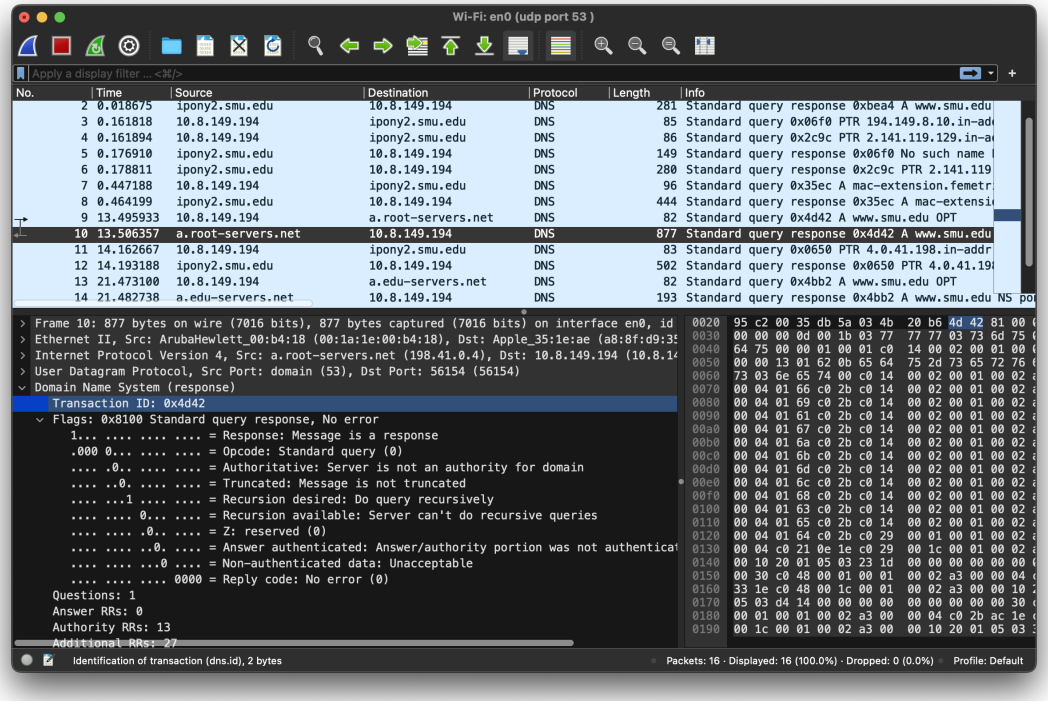
Solution:



Look at the DNS header, and answer the following questions:

1) How many bits long is the Transaction ID? Based on this length, take your best guess as to how likely it is that concurrent transactions will use the same transaction ID.

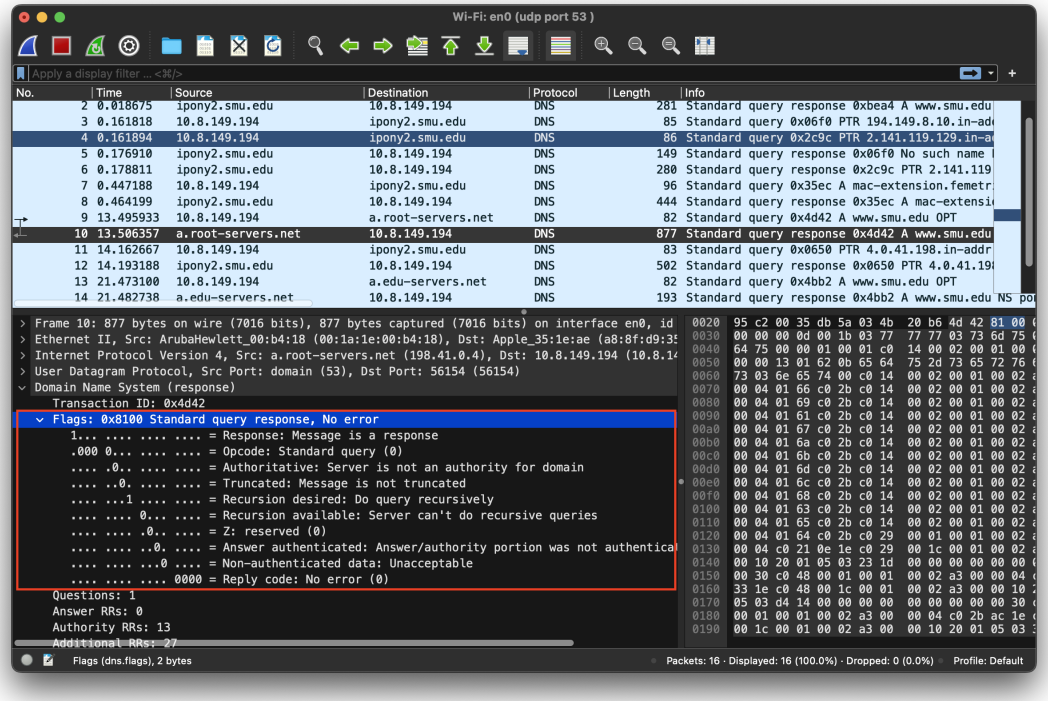
   Solution:



Transaction ID: 0x4d42. This ID is represented in hexadecimal format. Since each hexadecimal digit represents 4 bits, and there are 4 digits (4d42), the Transaction ID is 16 bits long. With 16 bits, there are 65,536 possible unique IDs, making the likelihood of concurrent transactions using the same ID relatively low.

2) Which flag bit and what values signifies whether the DNS message is a query or response?

Solution:



The Flags field is 0x8100. The most significant bit of this field (the first bit in the sequence) determines if the message is a query or a response. In this case, the bit is 1, signifying that the message is a response. For queries, this bit would be 0.

3) How many bytes long is the entire DNS header? Use information in the bottom status line when you select parts of the packet and the bottom panel to help you work this out.

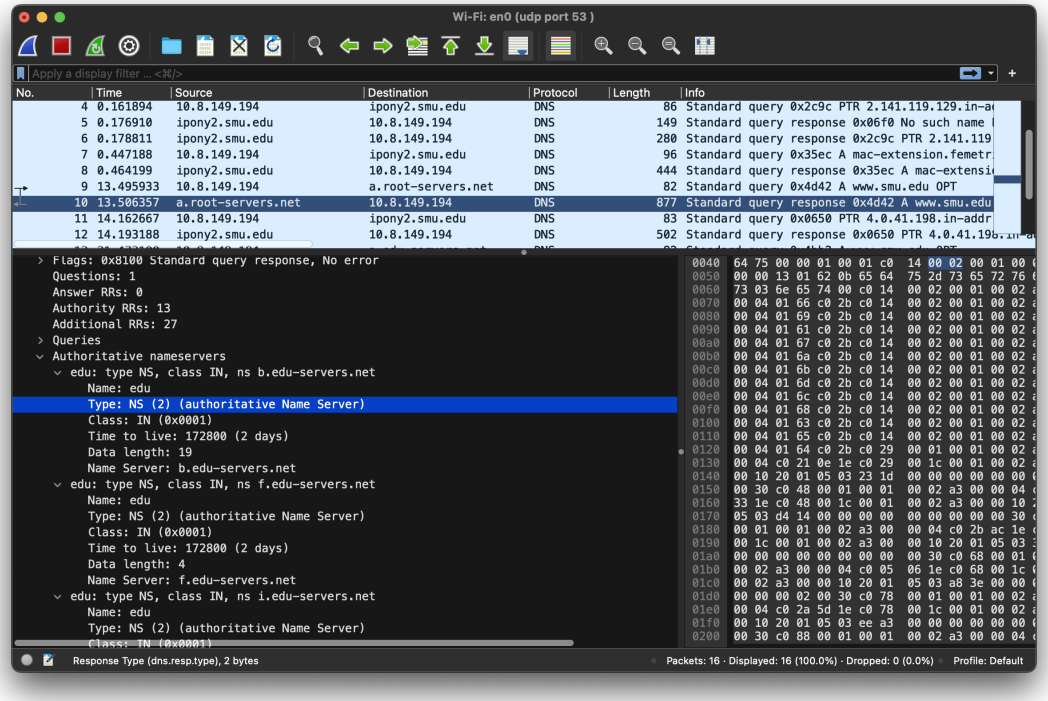
Solution: The DNS header typically includes the Transaction ID, Flags, Questions, Answer RRs, Authority RRs, Additional RRs, and some additional fields. However, the provided capture does not specify the explicit byte length of the DNS header. Normally, a DNS header is 12 bytes long.

Now examine the responses to the dig DNS queries you made. The initial response should have provided another nameserver one step closer to the nameserver, but not the final answer. You should find that it includes the original query in its Query section. It will also include records with both the name of the nameservers to contact next, and the IP addresses of those nameservers. The final response in this series will include the IP address of the domain name ? this is the answer to the query.

Look at the body of the DNS response messages, and answer the following questions:

4) For the initial response, in what section are the names of the nameservers carried? What is the Type of the records that carry nameserver names?

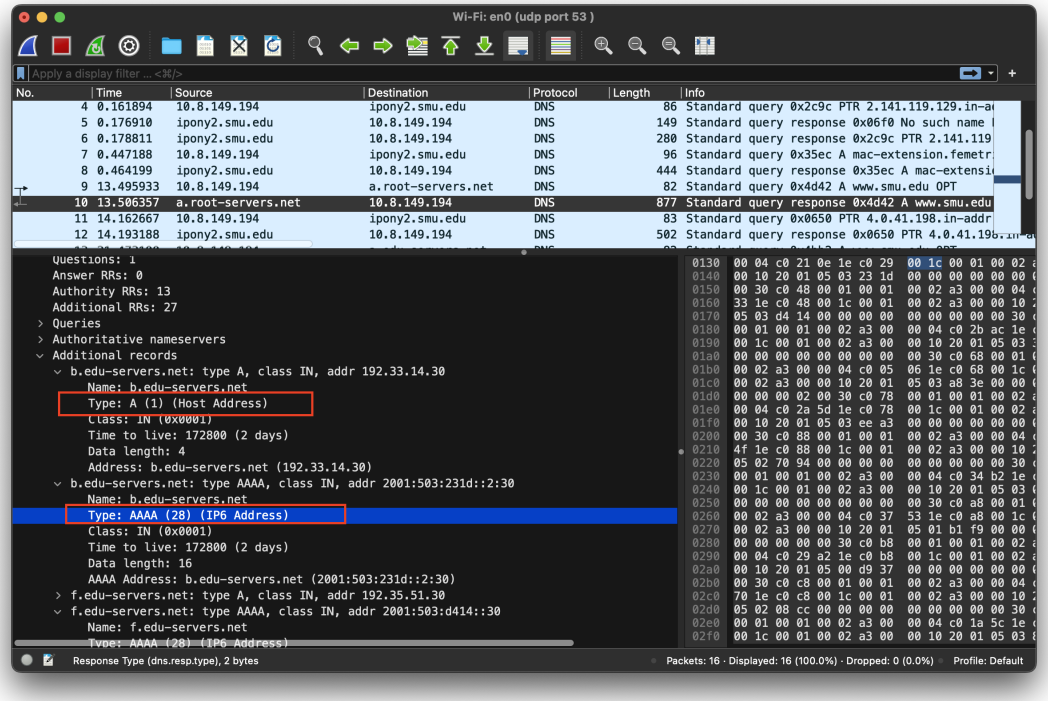
Solution:



The names of the nameservers are carried in the ”Authoritative nameservers” section. The Type of the records carrying the nameserver names is NS (Name Server), with a value of 2.

5) Similarly, in what section are the IP addresses of the nameservers carried, and what is the Type of the records that carry the IP addresses?

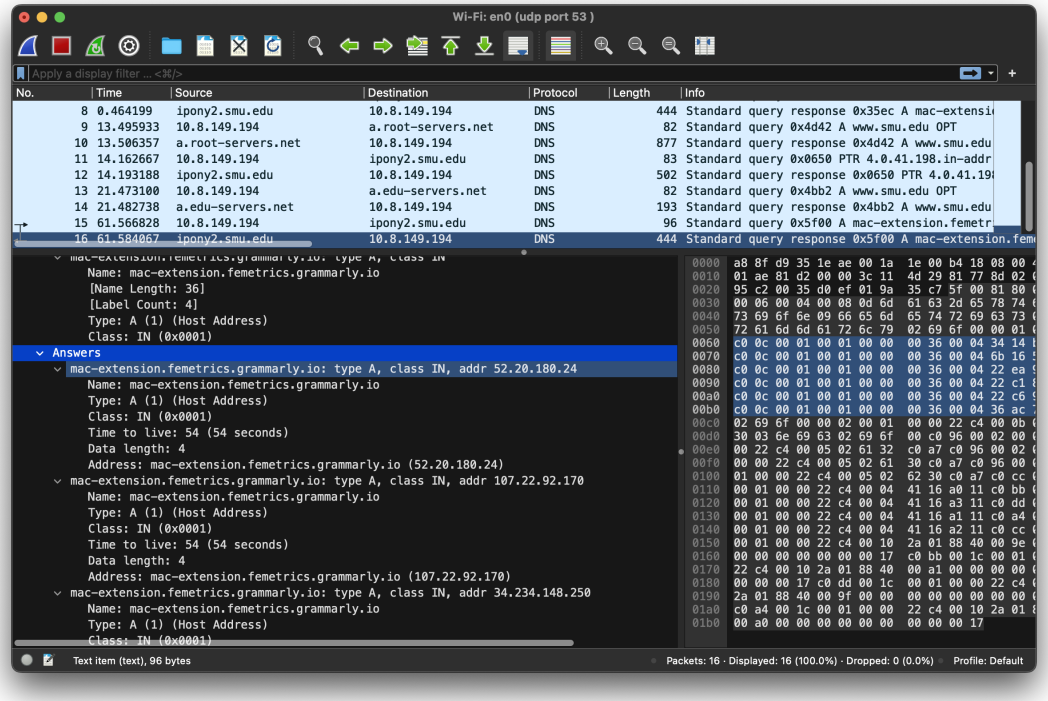
Solution:



The IP addresses of the nameservers are carried in the ”Additional records” section. The Type of the records carrying the IP addresses is A (Host Address) for IPv4 addresses, with a value of 1, and AAAA (IP6 Address) for IPv6 addresses, with a value of 28.

6) For the final response, in what section is the IP address of the domain name carried?

Solution: In this particular capture, the final response section carrying the IP address of the domain name is not provided. Typically, this information would be found in the ”Answer” section of the DNS response, and the record Type would be A (for IPv4) or AAAA (for IPv6), similar to the types seen in the ”Additional records” section for the nameserver IPs. But in the last response,



Domain Name is: mac-extension.femetrics.grammarly.io And Record Type: A (Host Address), with a value of 1, indicating these are IPv4 addresses. List of IP Addresses: 52.20.180.24; 107.22.92.170; 34.234.148.250; 34.193.138.38; 34.198.145.166; 54.172.120.202; These records provide the IPv4 addresses for the domain name mac-extension.femetrics.grammarly.io. Each address is marked as type A, indicating they are host addresses. These addresses represent the final response to the DNS query, meaning they are the IP addresses associated with the domain name at the time of the query

Turn In: Hand in your answers to the above questions.

References

[1] A. S. Tanenbaum, N. Feamster, and D. Wetherall, Computer networks, Sixth edition, Global edition. Harlow, United Kingdom: Pearson, 2021.