Lab Exercise – Protocol Layers and Ethernet

Objective

To learn how protocols and layering are represented in packets, as discussed in Lecture 3, and to explore the details of Ethernet frames. Ethernet is a popular link layer protocol that we covered in Lecture 6.

Requirements

Wireshark: This lab 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 UI 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 it from www.wireshark.org if it is not already installed on your computer.

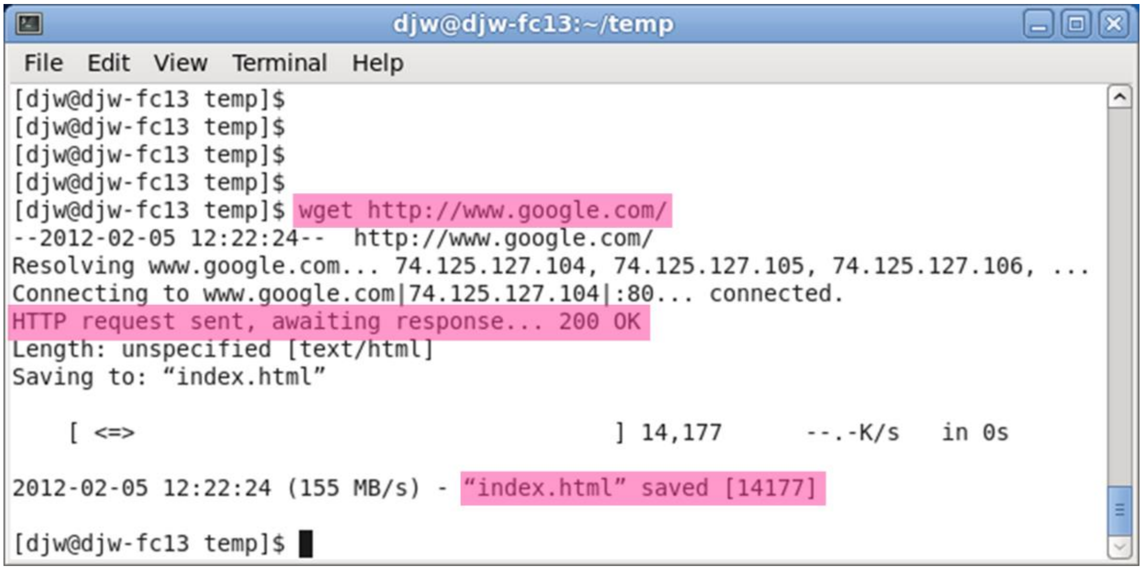
wget / curl: This lab uses `wget` (Linux and Windows) and `curl` (Mac) to fetch web resources. `wget` and `curl` are command-line programs that let you fetch a URL. Unlike a web browser, which fetches and executes entire pages, `wget` and `curl` give you control over exactly which URLs you fetch and 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*".

ping: This lab uses "`ping`" to send and receive messages. `ping` is a standard command-line utility for checking that another computer is responsive. It is widely used for network troubleshooting and comes pre-installed on Window, Linux, and Mac. While `ping` has various options, simply issuing the command "`ping www.bing.com`" will cause your computer to send a small number of ICMP ping requests to the remote computer (here www.bing.com), each of which should elicit an ICMP ping response.

Step 1: Capture a Trace

Proceed as follows to capture a trace of network traffic; alternatively, you may use a supplied trace. 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.

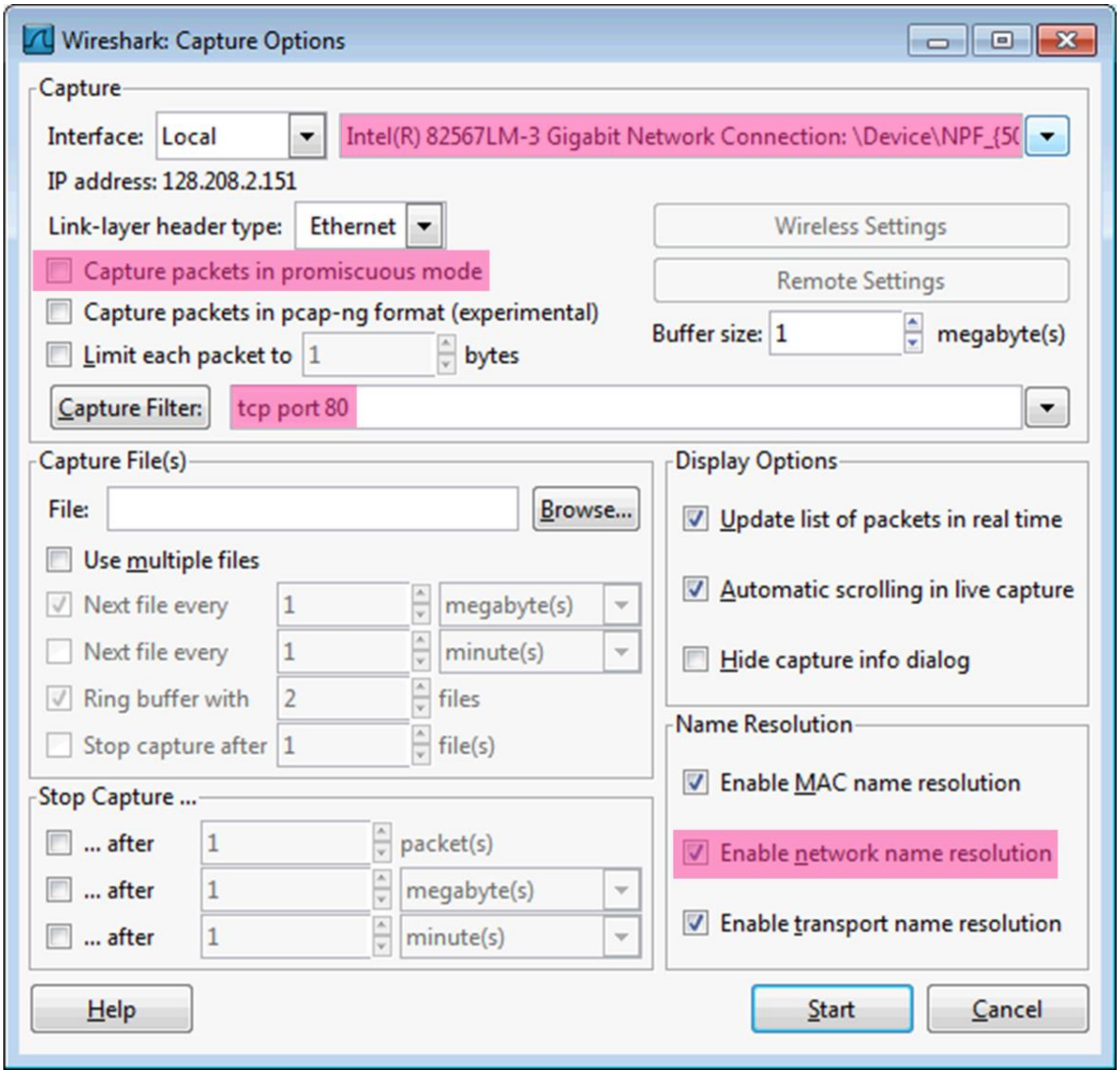
1. *Pick a URL and fetch it with* `wget` *or* `curl`. For example, "`wget http://www.google.com`" or "`curl http://www.google.com`". This will fetch the resource and either write it to a file (`wget`) or to the screen (`curl`). You are checking to see that the fetch works and retrieves some content. A successful example is shown below (with added highlighting) for `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.



```
[djw@djw-fc13 temp]$
[djw@djw-fc13 temp]$
[djw@djw-fc13 temp]$
[djw@djw-fc13 temp]$
[djw@djw-fc13 temp]$ wget http://www.google.com/
--2012-02-05 12:22:24--  http://www.google.com/
Resolving www.google.com... 74.125.127.104, 74.125.127.105, 74.125.127.106, ...
Connecting to www.google.com|74.125.127.104|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: “index.html”

    [ <=>                                  ] 14,177      --.-K/s   in 0s

2012-02-05 12:22:24 (155 MB/s) - “index.html” saved [14177]

[djw@djw-fc13 temp]$ 
```

Figure 1: Using `wget` to fetch a URL

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 filter 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. Your capture window should be similar to the one pictured below, other than our highlighting. 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.



Wireshark: Capture Options

Capture

Interface: Local Intel(R) 82567LM-3 Gigabit Network Connection: \Device\NPF_{5(

IP address: 128.208.2.151

Link-layer header type: Ethernet

□ Capture packets in promiscuous mode

□ Capture packets in pcap-ng format (experimental)

□ Limit each packet to 1 bytes

Buffer size: 1 megabyte(s)

Capture Filter: tcp port 80

Capture File(s)

File:

□ Use multiple files

☑ Next file every 1 megabyte(s)

□ Next file every 1 minute(s)

☑ Ring buffer with 2 files

□ Stop capture after 1 file(s)

Display Options

☑ Update list of packets in real time

☑ Automatic scrolling in live capture

□ Hide capture info dialog

Stop Capture ...

□ ... after 1 packet(s)

□ ... after 1 megabyte(s)

□ ... after 1 minute(s)

Name Resolution

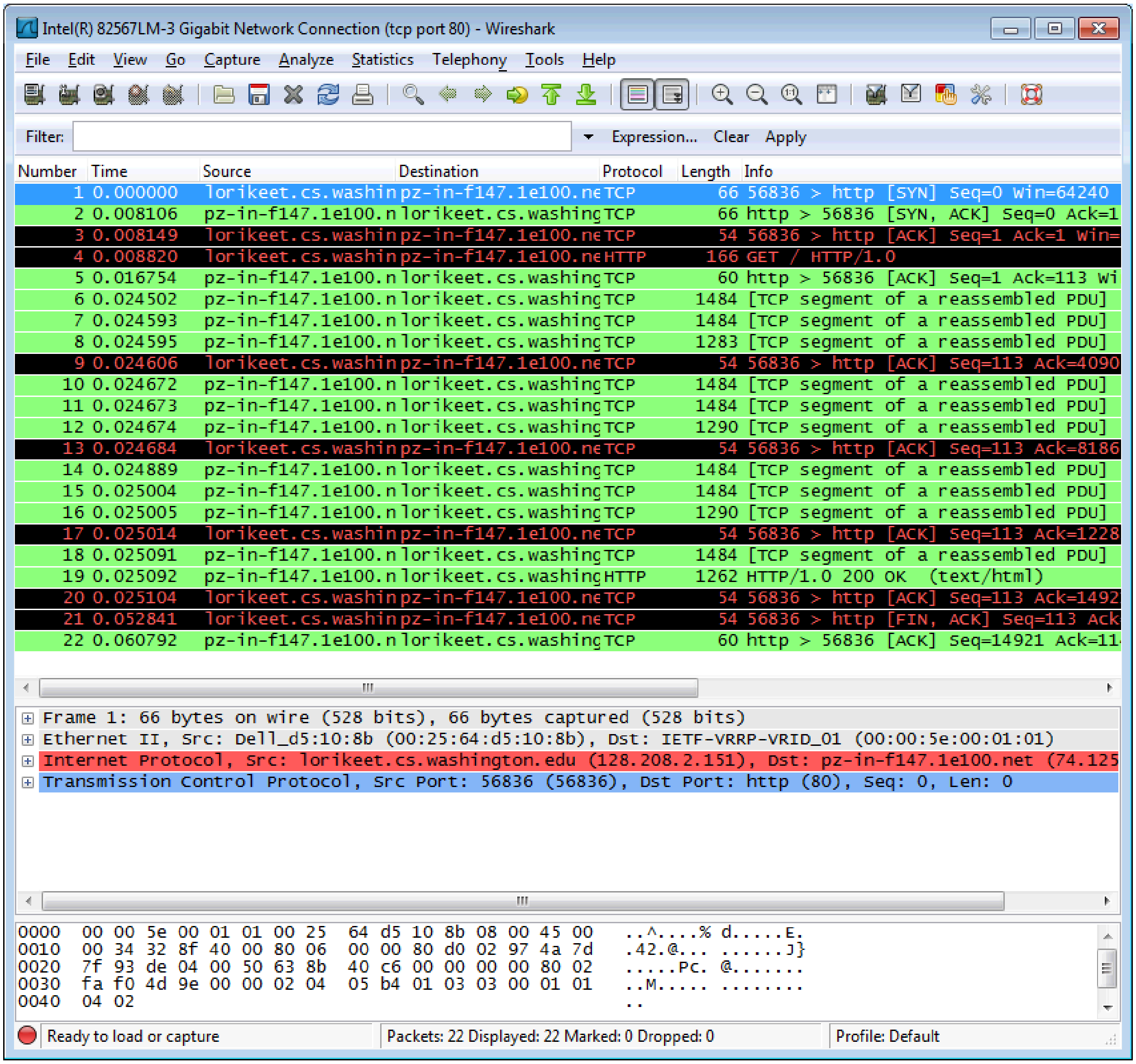
☑ Enable MAC name resolution

☑ Enable network name resolution

☑ Enable transport name resolution

Figure 2: Setting up the capture options

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.
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. An example is shown below. Congratulations, you have captured a trace!



Number	Time	Source	Destination	Protocol	Length	Info
1	0.000000	lorikeet.cs.washin	pz-in-f147.1e100.ne	TCP	66	56836 > http [SYN] Seq=0 Win=64240
2	0.008106	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	66	http > 56836 [SYN, ACK] Seq=0 Ack=1
3	0.008149	lorikeet.cs.washin	pz-in-f147.1e100.ne	TCP	54	56836 > http [ACK] Seq=1 Ack=1 Win=
4	0.008820	lorikeet.cs.washin	pz-in-f147.1e100.ne	HTTP	166	GET / HTTP/1.0
5	0.016754	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	60	http > 56836 [ACK] Seq=1 Ack=113 Wi
6	0.024502	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	1484	[TCP segment of a reassembled PDU]
7	0.024593	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	1484	[TCP segment of a reassembled PDU]
8	0.024595	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	1283	[TCP segment of a reassembled PDU]
9	0.024606	lorikeet.cs.washin	pz-in-f147.1e100.ne	TCP	54	56836 > http [ACK] Seq=113 Ack=4090
10	0.024672	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	1484	[TCP segment of a reassembled PDU]
11	0.024673	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	1484	[TCP segment of a reassembled PDU]
12	0.024674	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	1290	[TCP segment of a reassembled PDU]
13	0.024684	lorikeet.cs.washin	pz-in-f147.1e100.ne	TCP	54	56836 > http [ACK] Seq=113 Ack=8186
14	0.024889	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	1484	[TCP segment of a reassembled PDU]
15	0.025004	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	1484	[TCP segment of a reassembled PDU]
16	0.025005	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	1290	[TCP segment of a reassembled PDU]
17	0.025014	lorikeet.cs.washin	pz-in-f147.1e100.ne	TCP	54	56836 > http [ACK] Seq=113 Ack=1228
18	0.025091	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	1484	[TCP segment of a reassembled PDU]
19	0.025092	pz-in-f147.1e100.n	lorikeet.cs.washing	HTTP	1262	HTTP/1.0 200 OK (text/html)
20	0.025104	lorikeet.cs.washin	pz-in-f147.1e100.ne	TCP	54	56836 > http [ACK] Seq=113 Ack=1492
21	0.052841	lorikeet.cs.washin	pz-in-f147.1e100.ne	TCP	54	56836 > http [FIN, ACK] Seq=113 Ack
22	0.060792	pz-in-f147.1e100.n	lorikeet.cs.washing	TCP	60	http > 56836 [ACK] Seq=14921 Ack=11

```
⊞ Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
⊞ Ethernet II, Src: Dell_d5:10:8b (00:25:64:d5:10:8b), Dst: IETF-VRRP-VRID_01 (00:00:5e:00:01:01)
⊞ Internet Protocol, Src: lorikeet.cs.washington.edu (128.208.2.151), Dst: pz-in-f147.1e100.net (74.125
⊞ Transmission Control Protocol, Src Port: 56836 (56836), Dst Port: http (80), Seq: 0, Len: 0
```

```
0000   00 00 5e 00 01 01 00 25  64 d5 10 8b 08 00 45 00   ..^....% d.....E.
0010   00 34 32 8f 40 00 80 06  00 00 80 d0 02 97 4a 7d   .42.@... ......J}
0020   7f 93 de 04 00 50 63 8b  40 c6 00 00 00 00 80 02   .....Pc. @.......
0030   fa f0 4d 9e 00 00 02 04  05 b4 01 03 03 00 01 01   ..M..... ........
0040   04 02                                              ..
```

Ready to load or capture | Packets: 22 Displayed: 22 Marked: 0 Dropped: 0 | Profile: Default

Figure 3: Packet trace of `wget` traffic

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 as shown below. 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 (shown) if your computer is wired, or 802.11 (not shown) if your computer is wireless.

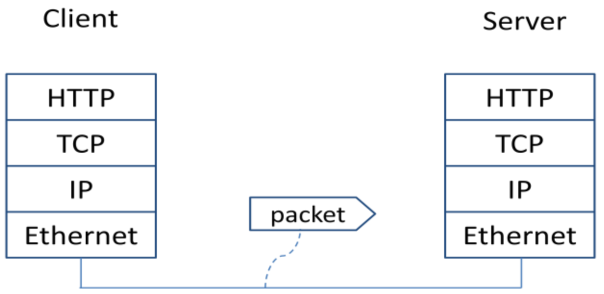


Figure 4: Protocol stack for a web fetch

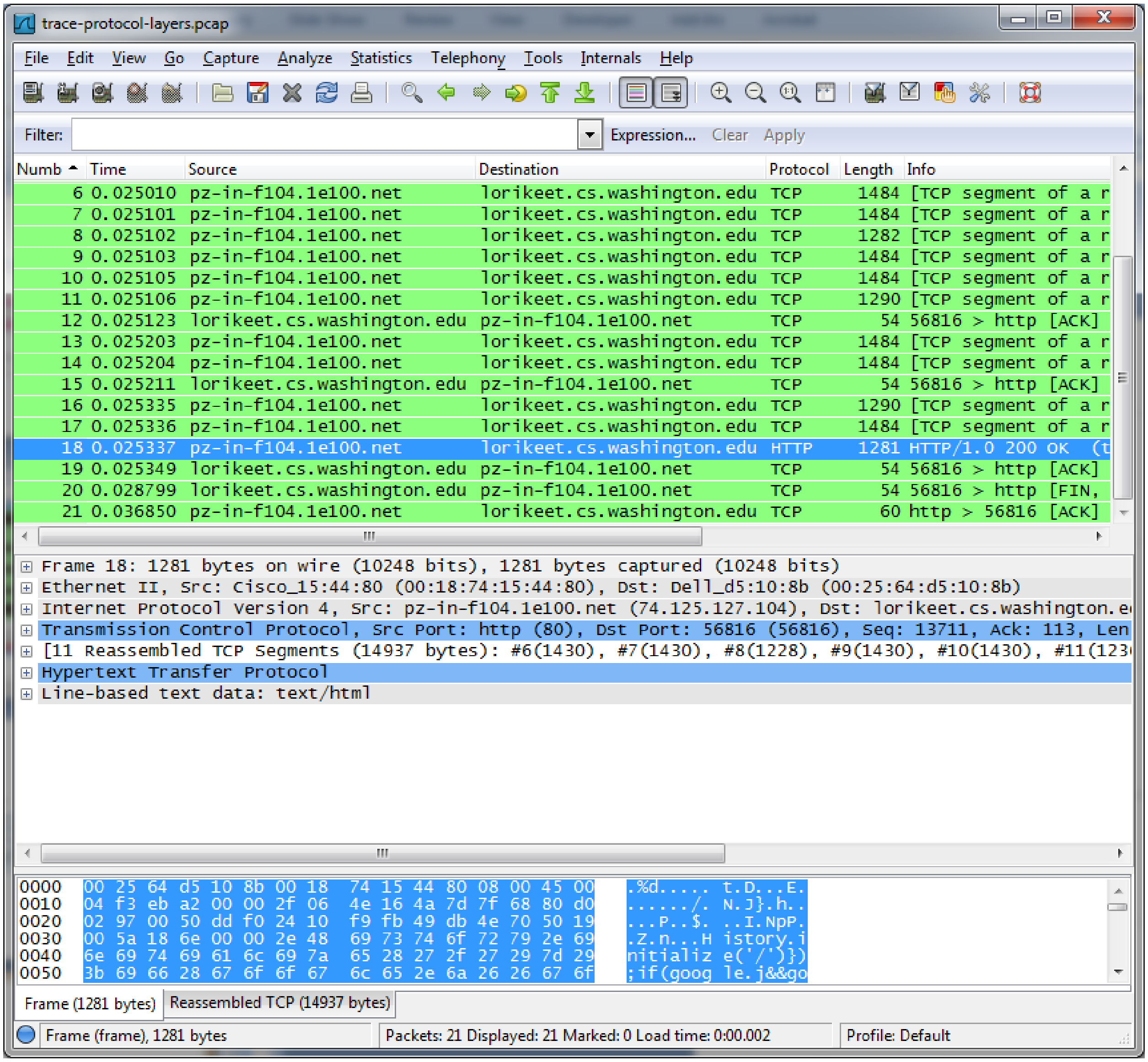
With the HTTP GET packet selected, look closely to see the similarities and differences between it and our protocol stack as described next. The protocol blocks are listed in the middle panel. You can expand each block (by clicking on the "+" expander or icon) to see its details.

- 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". This matches our diagram! Note that you may have taken a trace on a computer using 802.11 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, as in Fig. 1-15 of your text. 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 our trace, there are two extra blocks in the detail panel as seen in the next figure.

- 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.


trace-protocol-layers.pcap
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply

```
Numb ▲ Time     Source                     Destination                Protocol Length Info
     6 0.025010 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP        1484 [TCP segment of a r
     7 0.025101 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP        1484 [TCP segment of a r
     8 0.025102 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP        1282 [TCP segment of a r
     9 0.025103 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP        1484 [TCP segment of a r
    10 0.025105 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP        1484 [TCP segment of a r
    11 0.025106 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP        1290 [TCP segment of a r
    12 0.025123 lorikeet.cs.washington.edu pz-in-f104.1e100.net       TCP          54 56816 > http [ACK]
    13 0.025203 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP        1484 [TCP segment of a r
    14 0.025204 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP        1484 [TCP segment of a r
    15 0.025211 lorikeet.cs.washington.edu pz-in-f104.1e100.net       TCP          54 56816 > http [ACK]
    16 0.025335 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP        1290 [TCP segment of a r
    17 0.025336 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP        1484 [TCP segment of a r
    18 0.025337 pz-in-f104.1e100.net       lorikeet.cs.washington.edu HTTP       1281 HTTP/1.0 200 OK  (t
    19 0.025349 lorikeet.cs.washington.edu pz-in-f104.1e100.net       TCP          54 56816 > http [ACK]
    20 0.028799 lorikeet.cs.washington.edu pz-in-f104.1e100.net       TCP          54 56816 > http [FIN,
    21 0.036850 pz-in-f104.1e100.net       lorikeet.cs.washington.edu TCP          60 http > 56816 [ACK]
```

```
⊞ Frame 18: 1281 bytes on wire (10248 bits), 1281 bytes captured (10248 bits)
⊞ Ethernet II, Src: Cisco_15:44:80 (00:18:74:15:44:80), Dst: Dell_d5:10:8b (00:25:64:d5:10:8b)
⊞ Internet Protocol Version 4, Src: pz-in-f104.1e100.net (74.125.127.104), Dst: lorikeet.cs.washington.e
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 56816 (56816), Seq: 13711, Ack: 113, Len
⊞ [11 Reassembled TCP Segments (14937 bytes): #6(1430), #7(1430), #8(1228), #9(1430), #10(1430), #11(123
⊞ Hypertext Transfer Protocol
⊞ Line-based text data: text/html
```

```
0000  00 25 64 d5 10 8b 00 18  74 15 44 80 08 00 45 00   .%d..... t.D...E.
0010  04 f3 eb a2 00 00 2f 06  4e 16 4a 7d 7f 68 80 d0   ....../. N.J}.h..
0020  02 97 00 50 dd f0 24 10  f9 fb 49 db 4e 70 50 19   ...P..$. ..I.NpP.
0030  00 5a 18 6e 00 00 2e 48  69 73 74 6f 72 79 2e 69   .Z.n...H istory.i
0040  6e 69 74 69 61 6c 69 7a  65 28 27 2f 27 29 7d 29   nitializ e('/')})
0050  3b 69 66 28 67 6f 6f 67  6c 65 2e 6a 26 26 67 6f   ;if(goog le.j&&go
```

Frame (1281 bytes)
Reassembled TCP (14937 bytes)
Frame (frame), 1281 bytes
Packets: 21 Displayed: 21 Marked: 0 Load time: 0:00.002
Profile: Default

Figure 5: Inspecting a HTTP “200 OK” response

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 lower 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.

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.

Step 5: Demultiplexing Keys

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:

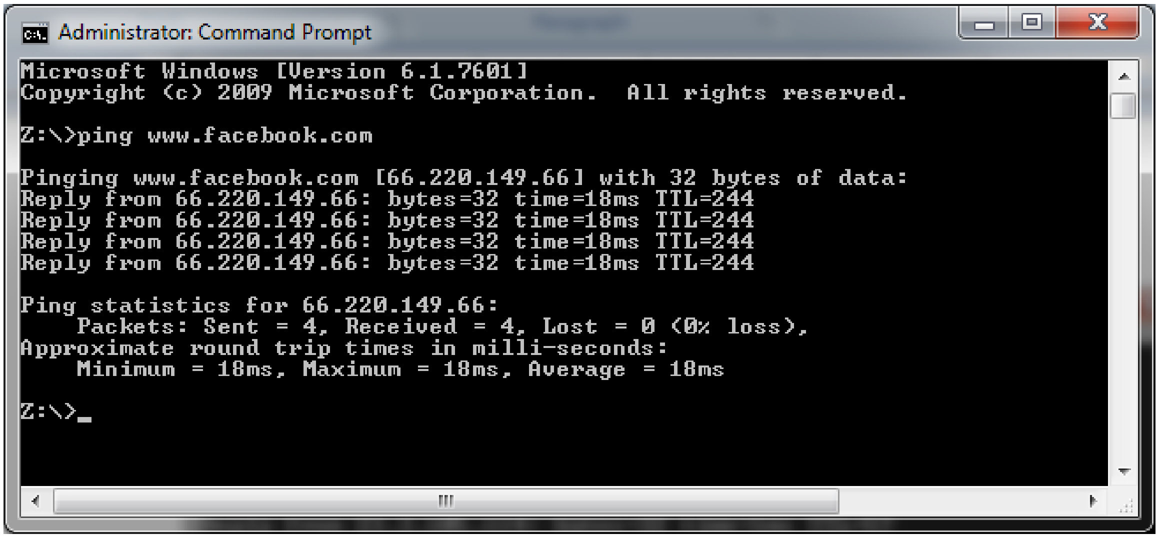
1. *Which Ethernet header field is the demultiplexing key that tells it the next higher layer is IP? What value is used in this field to indicate "IP"?*

2. *<u>Which IP header field is the demultiplexing key</u> that tells it the next higher layer is TCP? What value is used in this field to indicate "TCP"?*

Step 6: Capture the Trace for a Ping

Proceed as follows to capture a trace of ping packets; alternatively you may use a supplied trace. We will use ping simply as an easy way to collect a small trace.

1. *Pick a remote web server or other publicly reachable Internet host and use* `ping` *to send some ping messages and check that it sends replies*. For example, "`ping www.bing.com`". You should see several replies indicating that the pings reached the remote host and were returned. The figure below shows a successful example. Note that some versions of `ping` will continue to bounce messages off of a remote server until you tell the program to stop by signaling it with ^C. If your ping test does not succeed then try another server.

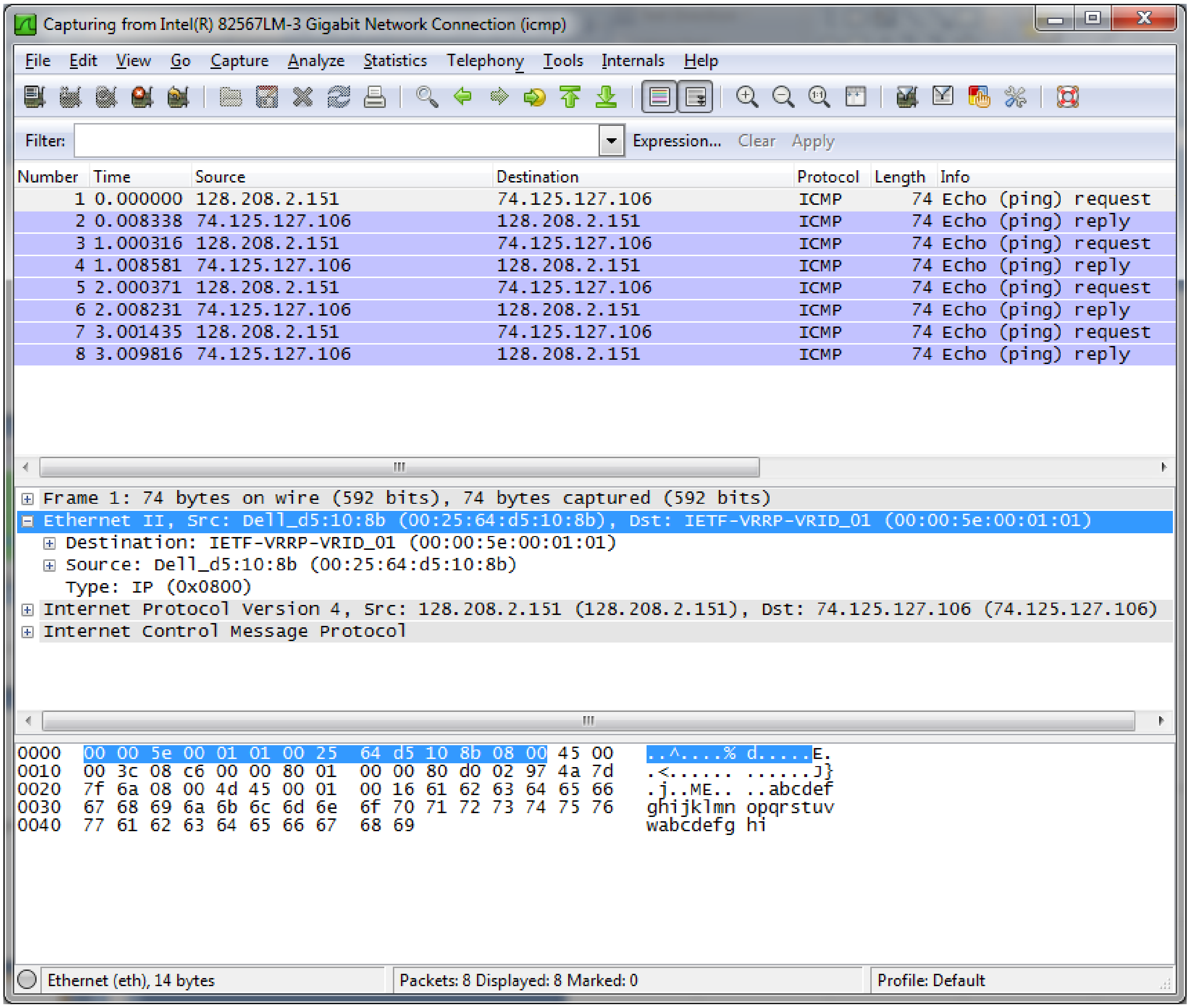


```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

Z:\>ping www.facebook.com

Pinging www.facebook.com [66.220.149.66] with 32 bytes of data:
Reply from 66.220.149.66: bytes=32 time=18ms TTL=244
Reply from 66.220.149.66: bytes=32 time=18ms TTL=244
Reply from 66.220.149.66: bytes=32 time=18ms TTL=244
Reply from 66.220.149.66: bytes=32 time=18ms TTL=244

Ping statistics for 66.220.149.66:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 18ms, Maximum = 18ms, Average = 18ms

Z:\>_
```

Figure 6: Using `ping` to bounce messages off a remote host

2. *Launch Wireshark and start a capture of Ethernet frames with a filter of* "`icmp`"*, making sure that "enable MAC name resolution" is checked*. The latter will translate Ethernet (MAC) addresses to provide vendor information. Also check that the Link-layer header type pulldown says "Ethernet".

3. *When the capture is started, repeat the* `ping` *command above.* This time, the packets will also be recorded by Wireshark.
4. *After the* `ping` *command is complete, return to Wireshark and use the menus or buttons to stop the trace.* You should now have a short trace similar to that shown in the figure below. If you do not succeed in capturing a trace then use the supplied one. Note that the trace we supply begins with ping messages, and then has other kinds of Ethernet frames.



```
Number Time     Source          Destination     Protocol Length Info
     1 0.000000 128.208.2.151   74.125.127.106  ICMP         74 Echo (ping) request
     2 0.008338 74.125.127.106  128.208.2.151   ICMP         74 Echo (ping) reply
     3 1.000316 128.208.2.151   74.125.127.106  ICMP         74 Echo (ping) request
     4 1.008581 74.125.127.106  128.208.2.151   ICMP         74 Echo (ping) reply
     5 2.000371 128.208.2.151   74.125.127.106  ICMP         74 Echo (ping) request
     6 2.008231 74.125.127.106  128.208.2.151   ICMP         74 Echo (ping) reply
     7 3.001435 128.208.2.151   74.125.127.106  ICMP         74 Echo (ping) request
     8 3.009816 74.125.127.106  128.208.2.151   ICMP         74 Echo (ping) reply
```

```
⊞ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
⊟ Ethernet II, Src: Dell_d5:10:8b (00:25:64:d5:10:8b), Dst: IETF-VRRP-VRID_01 (00:00:5e:00:01:01)
  ⊞ Destination: IETF-VRRP-VRID_01 (00:00:5e:00:01:01)
  ⊞ Source: Dell_d5:10:8b (00:25:64:d5:10:8b)
    Type: IP (0x0800)
⊞ Internet Protocol Version 4, Src: 128.208.2.151 (128.208.2.151), Dst: 74.125.127.106 (74.125.127.106)
⊞ Internet Control Message Protocol
```

```
0000  00 00 5e 00 01 01 00 25  64 d5 10 8b 08 00 45 00   ..^....% d.....E.
0010  00 3c 08 c6 00 00 80 01  00 00 80 d0 02 97 4a 7d   .<...... ......J}
0020  7f 6a 08 00 4d 45 00 01  00 16 61 62 63 64 65 66   .j..ME.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

```
Ethernet (eth), 14 bytes | Packets: 8 Displayed: 8 Marked: 0 | Profile: Default
```

Figure 7: Trace of `ping` traffic, showing Ethernet details of the first packet

Step 7: Inspect the Trace

Select any packet in the trace (in the top panel) to see details of its structure (in the middle panel) and the bytes that make up the packet (in the bottom panel). Now we can inspect the details of the packets. In the figure, we have selected the first packet in the trace. Note that we are using the term "packet" in a loose way. Each record captured by Wireshark more correctly corresponds to a single frame in Ethernet format that carries a packet as its payload; Wireshark interprets as much structure as it can.

In the middle panel, expand the Ethernet header fields (using the "+" expander or icon) to see their details. Our interest is the Ethernet header, and you may ignore the higher layer protocols (which are IP and ICMP in this case). You can click on the Ethernet header to see the bytes that correspond to it in the packet highlighted in the bottom panel. We have performed both steps in the figure.

If you are capturing traffic over an 802.11 interface, you may wonder why you have an Ethernet header at all, instead of an 802.11 header. This happens because we asked Wireshark to capture packets in

Ethernet format on the capture options (in Figure 2). In this case, the OS software converted the real 802.11 header into a pseudo-Ethernet header. We are seeing the pseudo-Ethernet header.

Compare the fields you see with the picture of an Ethernet frame in Fig. 4-14 of your text. You will see both similarities and differences:

- There are two kinds of Ethernet shown in the slides, IEEE 802.3 and DIX Ethernet. The latter is rare. The frames in the figure and your capture are DIX, called “Ethernet II” in Wireshark.
- There is no preamble in the fields shown in Wireshark. The preamble is a physical layer mechanism to help the NIC identify the start of a frame. It carries no useful data and is not received like other fields.
- There is a destination address and a source address. Wireshark is decoding some of these bits in the OUI (Organizationally Unique Identifier) portion of the address to tell us the vendor of the NIC, e.g., Dell for the source address.
- There is a Type field. For the ping messages, the Ethernet type is IP, meaning the Ethernet payload carries an IP packet. (There is no Length field as in the IEEE 802.3 format. Instead, the length of a DIX Ethernet frame is determined by the hardware of a receiving computer, which looks for valid frames that start with a preamble and end with a correct checksum, and passed up to higher layers along with the packet.)
- There is no Data field per se – the data starts with the IP header right after the Ethernet header.
- There is no pad. A pad will be present at the end if the frame would otherwise be less than 64 bytes, the minimum Ethernet frame size.
- There is no checksum in most traces, even though it really does exist. Typically, Ethernet hardware that is sending or receiving frames computes or checks this field and adds or strips it. Thus it is simply not visible to the OS or Wireshark in most capture setups.
- There are also no VLAN fields such as the Tag shown in Fig. 4-49. If VLANs are in use, the VLAN tags are normally added and removed by switch ports so they will not be visible at host computers using the network.

Step 8: Ethernet Frame Structure

To show your understanding of the Ethernet frame format, draw a figure of the ping message that shows the position and size in bytes of the Ethernet header fields. Your figure can simply show the frame as a long, thin rectangle. The leftmost fields come first in the packet and are sent on the wire first. On this drawing, show the range of the Ethernet header and the Ethernet payload. Add a dashed box at the end to represent the 4-byte checksum; we know it is there even if Wireshark does not show us this field.

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 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.

Step 9: Scope of Ethernet Addresses

Each Ethernet frame carries a source and destination address. One of these addresses is that of your computer. It is the source for frames that are sent, and the destination for frames that are received. But what is the other address? Assuming you pinged a remote Internet server, it cannot be the Ethernet address of the remote server because an Ethernet frame is only addressed to go within one LAN. Instead, it will be the Ethernet address of the router or default gateway, such as your AP in the case of 802.11. This is the device that connects your LAN to the rest of the Internet. In contrast, the IP addresses in the IP block of each packet do indicate the overall source and destination endpoints. They are your computer and the remote server.

Draw a figure that shows the relative positions of your computer, the router, and the remote server. Label your computer and the router with their Ethernet addresses. Label your computer and the remote server with their IP addresses. Show where the Ethernet and the rest of the Internet fit on the drawing.

Step 10: Broadcast Frames

The trace that you gathered above captured unicast Ethernet traffic sent between a specific source and destination, e.g., your computer to the router. It is also possible to send multicast or broadcast Ethernet traffic, destined for a group of computers or all computers on the Ethernet, respectively. We can tell from the address whether it is unicast, multicast, or broadcast. Broadcast traffic is sent to a reserved Ethernet address that has all bits set to “1”. Multicast traffic is sent to addresses that have a “1” in the first bit sent on the wire; broadcast is a special case of multicast. Broadcast and multicast traffic is widely used for discovery protocols, e.g., a packet sent to everyone in an effort to find the local printer.

Start a capture for broadcast and multicast Ethernet frames with a filter of “`ether multicast`”, *wait up to 30 seconds to record background traffic, and then stop the capture. If you do not capture any packets with this filter then use the trace that we supplied.* On most Ethernets, there is a steady chatter of background traffic as computers exchange messages to maintain network state, which is why we try to capture traffic without running any other programs. The capture filter of “`ether multicast`” will capture both multicast and broadcast Ethernet frames, but not regular unicast frames. You may have to wait a little while for these packets to be captured, but on most LANs with multiple computers you will see at least a packet every few seconds.

Examine the multicast and broadcast packets that you captured, looking at the details of the source and destination addresses. Most likely one has the broadcast Ethernet address, as broadcast frames tend to be more common than multicast frames. Look at a broadcast frame to see what address is used for broadcast by Ethernet. Expand the Ethernet address fields of either broadcast or multicast frames to see which bit is set to distinguish broadcast/multicast or group traffic from unicast traffic.

Answer the following questions:

1. *What is the broadcast Ethernet address, written in standard form as Wireshark displays it?*
2. *Which bit of the Ethernet address is used to determine whether it is unicast or multicast/broadcast?*

Explore on your own

We encourage you to explore protocols and layering once you have completed this lab. Some ideas:

- Look at a short TCP packet that carries no higher-layer data. To what entity is this packet destined? After all, if it carries no higher-layer data then it does not seem very useful to a higher layer protocol such as HTTP!
- In a classic layered model, one message from a higher layer has a header appended by the lower layer and becomes one new message. But this is not always the case. Above, we saw a trace in which the web response (one HTTP message comprised of an HTTP header and an HTTP payload) was converted into multiple lower layer messages (being multiple TCP packets). Imagine that you have drawn the packet structure (as in step 2) for the first and last TCP packet carrying the web response. How will the drawings differ?
- In the classic layered model described above, lower layers append headers to the messages passed down from higher layers. How will this model change if a lower layer adds encryption?
- In the classic layered model described above, lower layers append headers to the messages passed down from higher layers. How will this model change if a lower layer adds compression?