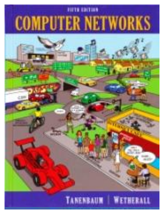
Lab Exercise – Protocol Layers



Objective

To learn how protocols and layering are represented in packets. They are key concepts for structuring networks that are covered in the text.

The trace for this lab is here:

http://scisweb.ulster.ac.uk/~kevin/com320/labs/wireshark/trace-protocol-layers.pcap

(although the main trace you will look at is from a site you pick such as www.ulster.ac.uk in the examples which follow).

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 for your personal computer. It is an ideal packet analyzer for our labs – it is stable, has a large user base and well-documented support that includes a user-guide http://www.wireshark.org/docs/wsug_html_chunked), and a detailed FAQ, rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface. It operates in computers using Ethernet, serial (PPP and SLIP), 802.11 wireless LANs, and many other link-layer technologies (if the OS on which it is running allows Wireshark to do so). It is already installed in the labs.

A quick help guide to Wireshark display filters is here: http://openmaniak.com/wireshark_filters.php

Wireshark is a core tool for any wireless 'man in the middle' or similar snooping attack. It is simply indispensable for those who wish to examine packets being transferred over a network – good or bad…..

Wireshark & Packet Sniffing Background

The basic tool for observing the messages exchanged between executing protocol entities is called a packet sniffer. As the name suggests, a packet sniffer captures (“sniffs”) messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a copy of packets that are sent/received from/by application and protocols executing on your machine.

Figure 1 shows the structure of a packet sniffer. At the right of Figure 1 are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniffer, shown within the dashed rectangle in Figure 1 is an addition to the usual software in your computer, and consists of two parts. The packet capture library receives a copy of every link-layer frame that is sent from or received by your computer. Messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable.

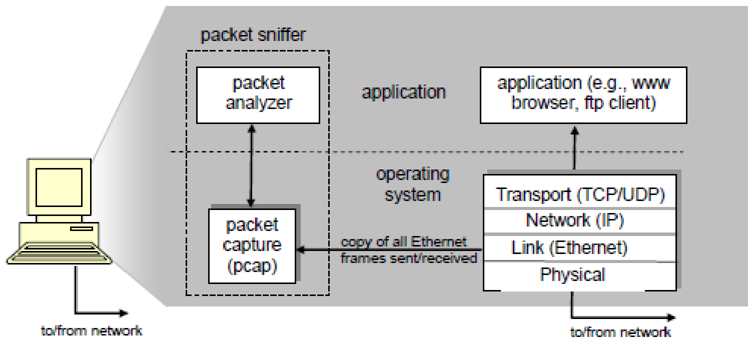


Figure 1: Packet Sniffer Structure

In Figure 1, the assumed physical media is an Ethernet, and so all upper-layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you all messages sent/received from/by all protocols and applications executing in your computer.

The second component of a packet sniffer is the packet analyzer, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer must “understand” the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol in Figure 1. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally, it understands the HTTP protocol and so, for example, knows that the first bytes of an HTTP message will contain the string “GET,” “POST,” or “HEAD,”.

Step 1: Capture a Trace

1. **Launching Wireshark**
   You can type ***Wireshark*** in the run box of main Windows 8 start screen. Press the Windows key on the keyboard and type "wireshark". If a problem launching it then see here[1].

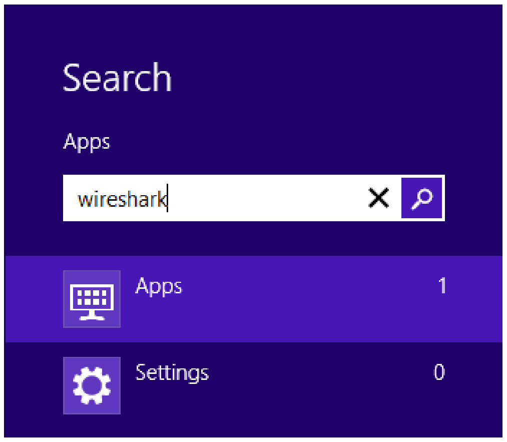


Figure 2: Wireshark in lab

2. Just **close the dialog box** which prompts you to install a new version. You will then see a startup screen, as shown next.

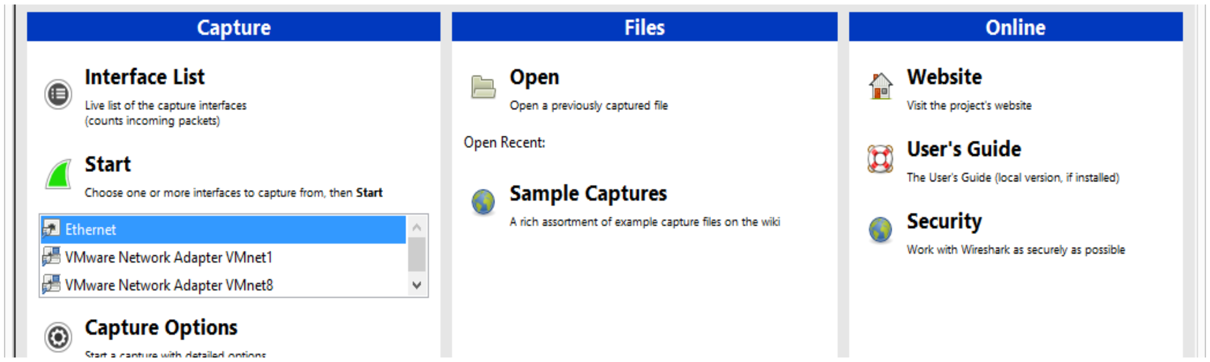
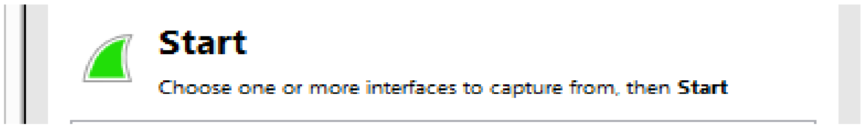

Capture
Interface List
Live list of the capture interfaces (counts incoming packets)
Start
Choose one or more interfaces to capture from, then Start
Ethernet
VMware Network Adapter VMnet1
VMware Network Adapter VMnet8
Capture Options
Start a capture with detailed options
Files
Open
Open a previously captured file
Open Recent:
Sample Captures
A rich assortment of example capture files on the wiki
Online
Website
Visit the project's website
User's Guide
The User's Guide (local version, if installed)
Security
Work with Wireshark as securely as possible

Figure 3: Initial Wireshark Screen

3. Take a look at the left hand side of the screen – you'll see an "Interface list". This is the list of network interfaces on your computer. Choose **Ethernet.**

[1] It should load but there can be a problem with the new lab configuration for Wireshark and npf driver. Therefore if this is not working…. then please do the next step. Launch Wireshark as follows. Click desktop icon on main windows screen & use the file explorer to browse to C:\local Disk (C)\Program Files\Wireshark. Finally, RIGHT CLICK on Wireshark as "Run as administrator".

4. Next, click Start (as shown below).



Start

Choose one or more interfaces to capture from, then **Start**

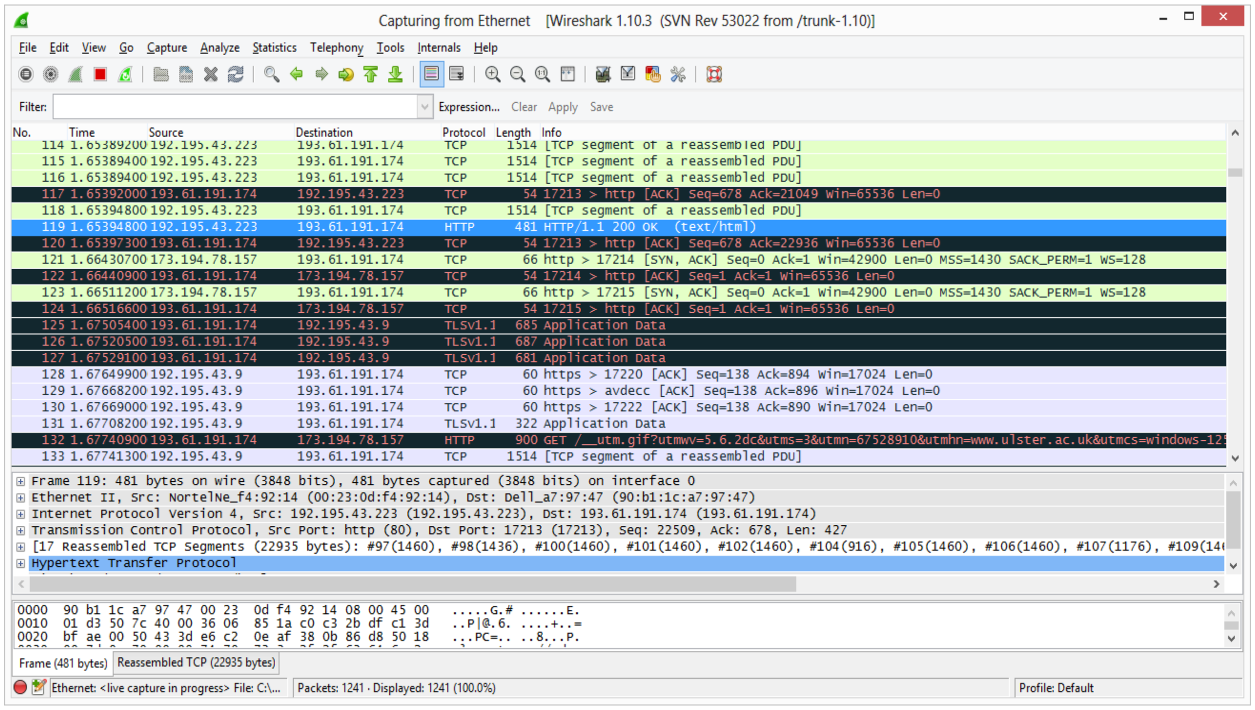
5. Wireshark will capture all packets on that interface. Click on the network card on the particular machine you are working on. In the example above, it is the **Ethernet Driver** to start packet capture (i.e., for Wireshark to begin capturing all packets being sent to/from that interface), a screen like the one below will be displayed, showing information about the packets being captured. Once you start packet capture, you can stop it by using the Capture pull down menu and selecting Stop.

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, could also serve as traffic.

6. *Open your browser, e.g. Chrome and pick a URL & fetch it e.g.* http://www.ulster.ac.uk ".

7. *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.

8. Now return to Wireshark and you will see a screen similar to below.



Capturing from Ethernet   [Wireshark 1.10.3 (SVN Rev 53022 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

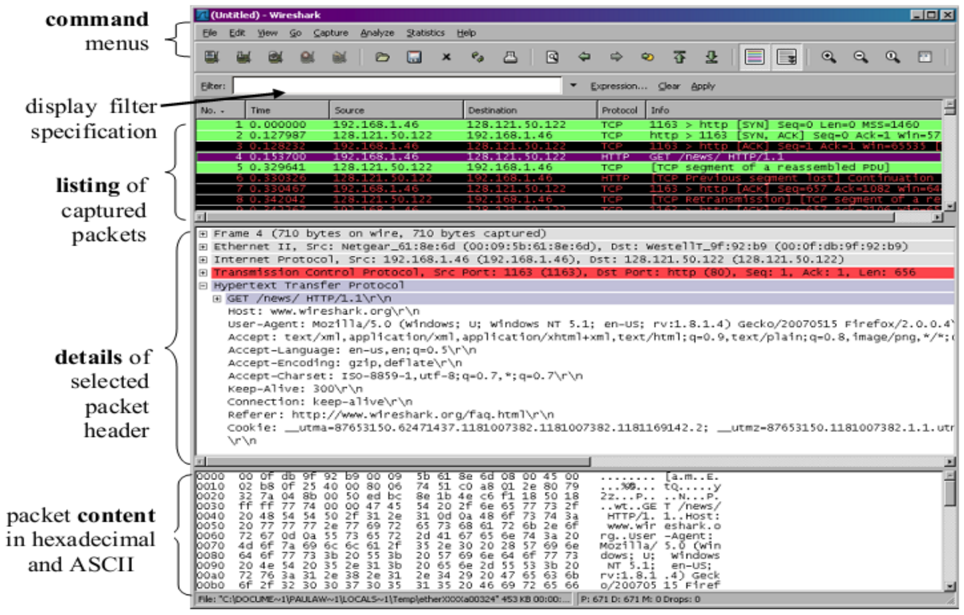
```
No.   Time       Source         Destination    Protocol Length Info
  114 1.65389200 192.195.43.223 193.61.191.174 TCP      1514 [TCP segment of a reassembled PDU]
  115 1.65389400 192.195.43.223 193.61.191.174 TCP      1514 [TCP segment of a reassembled PDU]
  116 1.65389400 192.195.43.223 193.61.191.174 TCP      1514 [TCP segment of a reassembled PDU]
  117 1.65392000 193.61.191.174 192.195.43.223 TCP        54 17213 > http [ACK] Seq=678 Ack=21049 Win=65536 Len=0
  118 1.65394800 192.195.43.223 193.61.191.174 TCP      1514 [TCP segment of a reassembled PDU]
  119 1.65394800 192.195.43.223 193.61.191.174 HTTP      481 HTTP/1.1 200 OK  (text/html)
  120 1.65397300 193.61.191.174 192.195.43.223 TCP        54 17213 > http [ACK] Seq=678 Ack=22936 Win=65536 Len=0
  121 1.66430700 173.194.78.157 193.61.191.174 TCP        66 http > 17214 [SYN, ACK] Seq=0 Ack=1 Win=42900 Len=0 MSS=1430 SACK_PERM=1 WS=128
  122 1.66440900 193.61.191.174 173.194.78.157 TCP        54 17214 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
  123 1.66511200 173.194.78.157 193.61.191.174 TCP        66 http > 17215 [SYN, ACK] Seq=0 Ack=1 Win=42900 Len=0 MSS=1430 SACK_PERM=1 WS=128
  124 1.66516600 193.61.191.174 173.194.78.157 TCP        54 17215 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
  125 1.67505400 193.61.191.174 192.195.43.9   TLSv1.1   685 Application Data
  126 1.67520500 193.61.191.174 192.195.43.9   TLSv1.1   687 Application Data
  127 1.67529100 193.61.191.174 192.195.43.9   TLSv1.1   681 Application Data
  128 1.67649900 192.195.43.9   193.61.191.174 TCP        60 https > 17220 [ACK] Seq=138 Ack=894 Win=17024 Len=0
  129 1.67668200 192.195.43.9   193.61.191.174 TCP        60 https > avdecc [ACK] Seq=138 Ack=896 Win=17024 Len=0
  130 1.67669000 192.195.43.9   193.61.191.174 TCP        60 https > 17222 [ACK] Seq=138 Ack=890 Win=17024 Len=0
  131 1.67708200 192.195.43.9   193.61.191.174 TLSv1.1   322 Application Data
  132 1.67740900 193.61.191.174 173.194.78.157 HTTP      900 GET /__utm.gif?utmwv=5.6.2dc&utms=3&utmn=67528910&utmhn=www.ulster.ac.uk&utmcs=windows-12
  133 1.67741300 192.195.43.9   193.61.191.174 TCP      1514 [TCP segment of a reassembled PDU]

⊞ Frame 119: 481 bytes on wire (3848 bits), 481 bytes captured (3848 bits) on interface 0
⊞ Ethernet II, Src: NortelNe_f4:92:14 (00:23:0d:f4:92:14), Dst: Dell_a7:97:47 (90:b1:1c:a7:97:47)
⊞ Internet Protocol Version 4, Src: 192.195.43.223 (192.195.43.223), Dst: 193.61.191.174 (193.61.191.174)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 17213 (17213), Seq: 22509, Ack: 678, Len: 427
⊞ [17 Reassembled TCP Segments (22935 bytes): #97(1460), #98(1436), #100(1460), #101(1460), #102(1460), #104(916), #105(1460), #106(1460), #107(1176), #109(14
⊞ Hypertext Transfer Protocol

0000  90 b1 1c a7 97 47 00 23  0d f4 92 14 08 00 45 00   .....G.# ......E.
0010  01 d3 50 7c 40 00 36 06  85 1a c0 c3 2b df c1 3d   ..P|@.6. ....+..=
0020  bf ae 00 50 43 3d e6 c2  0e af 38 0b 86 d8 50 18   ...PC=.. ..8...P.
```

Frame (481 bytes)
Reassembled TCP (22935 bytes)
Ethernet: <live capture in progress> File: C:\...
Packets: 1241 · Displayed: 1241 (100.0%)
Profile: Default

That is all for now. Next you will read a little more about Wireshark and tracing packets.

Main Wireshark Interface

The Wireshark interface has five major components:


command menus
display filter specification
listing of captured packets
details of selected packet header
packet content in hexadecimal and ASCII
(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. .  Time       Source           Destination      Protocol  Info
  1 0.000000   192.168.1.46     128.121.50.122   TCP      1163 > http [SYN] Seq=0 Len=0 MSS=1460
  2 0.127987   128.121.50.122   192.168.1.46     TCP      http > 1163 [SYN, ACK] Seq=0 Ack=1 Win=57
  3 0.128232   192.168.1.46     128.121.50.122   TCP      1163 > http [ACK] Seq=1 Ack=1 Win=65535 [
  4 0.153700   192.168.1.46     128.121.50.122   HTTP     GET /news/ HTTP/1.1
  5 0.329641   128.121.50.122   192.168.1.46     TCP      [TCP segment of a reassembled PDU]
  6 0.330326   128.121.50.122   192.168.1.46     HTTP     [TCP Previous segment lost] Continuation
  7 0.330467   192.168.1.46     128.121.50.122   TCP      1163 > http [ACK] Seq=657 Ack=1082 Win=64
  8 0.342042   128.121.50.122   192.168.1.46     TCP      [TCP Retransmission] [TCP segment of a re
```

```
⊞ Frame 4 (710 bytes on wire, 710 bytes captured)
⊞ Ethernet II, Src: Netgear_61:8e:6d (00:09:5b:61:8e:6d), Dst: WestellT_9f:92:b9 (00:0f:db:9f:92:b9)
⊞ Internet Protocol, Src: 192.168.1.46 (192.168.1.46), Dst: 128.121.50.122 (128.121.50.122)
⊞ Transmission Control Protocol, Src Port: 1163 (1163), Dst Port: http (80), Seq: 1, Ack: 1, Len: 656
⊟ Hypertext Transfer Protocol
  ⊞ GET /news/ HTTP/1.1\r\n
    Host: www.wireshark.org\r\n
    User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;
    Accept-Language: en-us,en;q=0.5\r\n
    Accept-Encoding: gzip,deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    Referer: http://www.wireshark.org/faq.html\r\n
    Cookie: __utma=87653150.62471437.1181007382.1181007382.1181169142.2; __utmz=87653150.1181007382.1.1.utr
    \r\n
```

```
0000  00 0f db 9f 92 b9 00 09  5b 61 8e 6d 08 00 45 00   ........ [a.m..E.
0010  02 b8 0f 25 40 00 80 06  74 51 c0 a8 01 2e 80 79   ...%@... tQ.....y
0020  32 7a 04 8b 00 50 ed bc  8e 1b 4e c6 f1 18 50 18   2z...P.. ..N...P.
0030  ff ff 77 74 00 00 47 45  54 20 2f 6e 65 77 73 2f   ..wt..GE T /news/
0040  20 48 54 54 50 2f 31 2e  31 0d 0a 48 6f 73 74 3a    HTTP/1. 1..Host:
0050  20 77 77 77 2e 77 69 72  65 73 68 61 72 6b 2e 6f    www.wir eshark.o
0060  72 67 0d 0a 55 73 65 72  2d 41 67 65 6e 74 3a 20   rg..User -Agent:
0070  4d 6f 7a 69 6c 6c 61 2f  35 2e 30 20 28 57 69 6e   Mozilla/ 5.0 (Win
0080  64 6f 77 73 3b 20 55 3b  20 57 69 6e 64 6f 77 73   dows; U;  Windows
0090  20 4e 54 20 35 2e 31 3b  20 65 6e 2d 55 53 3b 20    NT 5.1;  en-US;
00a0  72 76 3a 31 2e 38 2e 31  2e 34 29 20 47 65 63 6b   rv:1.8.1 .4) Geck
00b0  6f 2f 32 30 30 37 30 35  31 35 20 46 69 72 65 66   o/200705 15 Firef
```

```
File: "C:\DOCUME~1\PAULAW~1\LOCALS~1\Temp\etherXXXXa00324" 453 KB 00:00:...   P: 671 D: 671 M: 0 Drops: 0
```

Figure 4: Wireshark Graphical User Interface, during packet capture and analysis

- The command menus are standard pull down menus located at the top of the window. The File menu allows you to save captured packet data or open a file containing previously captured packet data. The Capture menu allows you to begin packet capture.

- The packet-listing window displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is not a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest-level protocol that sent or received this packet, i.e., the protocol that is source or ultimate sink for this packet.

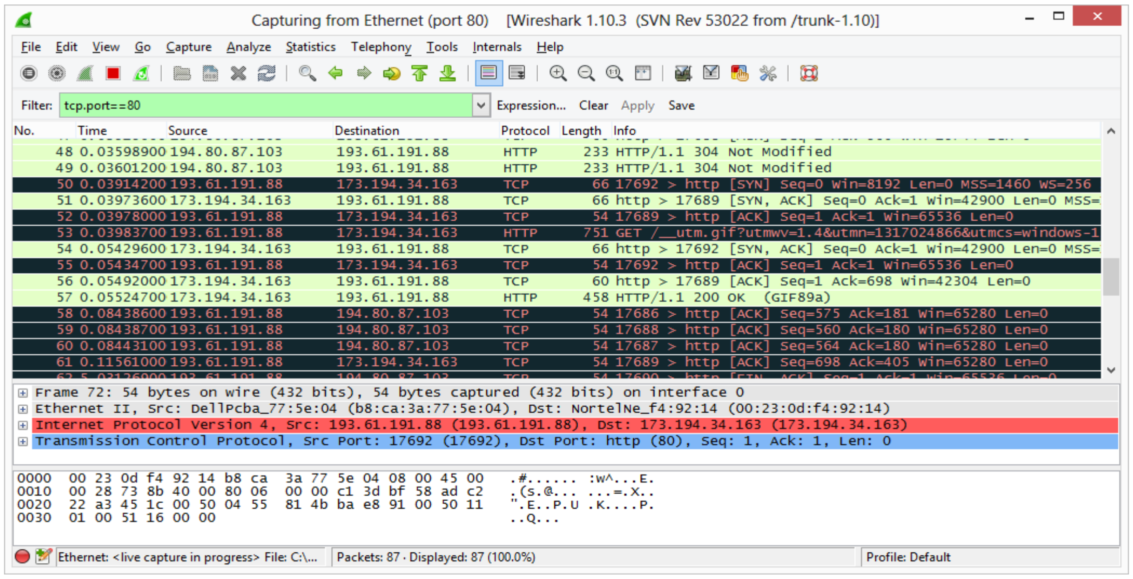
- The packet-header details window provides details about the packet selected (highlighted) in the packet-listing window. (To select a packet in the packet-listing window, place the cursor over the packet's one-line summary in the packet-listing window and click with the left mouse button.). These details include information about the Ethernet frame (assuming the packet was sent/received over an Ethernet interface) and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the plus minus boxes to the left of the Ethernet frame or IP datagram line in the packet details window.

- The packet-contents window displays the entire contents of the captured frame, in both ASCII and hexadecimal format. Towards the top of the Wireshark graphical user interface, is the packet display filter field, into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window.

Filtering Traffic

Now return to your Wireshark trace captured earlier. We will look to see how we can isolate the Web Page traffic from the other protocol packets.

9. *You have two options. You can simply type "tcp.port==80" in the main filter box on the main screen like below.*



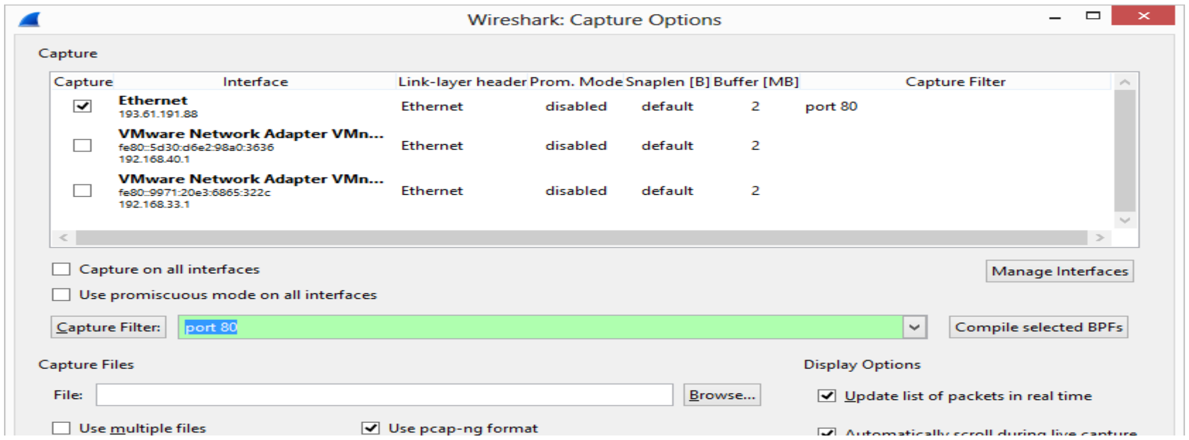
Capturing from Ethernet (port 80) [Wireshark 1.10.3 (SVN Rev 53022 from /trunk-1.10)]

Filter: tcp.port==80

```
No.    Time       Source          Destination     Protocol Length Info
    48 0.03598900 194.80.87.103   193.61.191.88   HTTP     233 HTTP/1.1 304 Not Modified
    49 0.03601200 194.80.87.103   193.61.191.88   HTTP     233 HTTP/1.1 304 Not Modified
    50 0.03914200 193.61.191.88   173.194.34.163  TCP       66 17692 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256
    51 0.03973600 173.194.34.163  193.61.191.88   TCP       66 http > 17689 [SYN, ACK] Seq=0 Ack=1 Win=42900 Len=0 MSS=
    52 0.03978000 193.61.191.88   173.194.34.163  TCP       54 17689 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
    53 0.03983700 193.61.191.88   173.194.34.163  HTTP     751 GET /__utm.gif?utmwv=1.4&utmn=1317024866&utmcs=windows-1
    54 0.05429600 173.194.34.163  193.61.191.88   TCP       66 http > 17692 [SYN, ACK] Seq=0 Ack=1 Win=42900 Len=0 MSS=
    55 0.05434700 193.61.191.88   173.194.34.163  TCP       54 17692 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
    56 0.05492000 173.194.34.163  193.61.191.88   TCP       60 http > 17689 [ACK] Seq=1 Ack=698 Win=42304 Len=0
    57 0.05524700 173.194.34.163  193.61.191.88   HTTP     458 HTTP/1.1 200 OK  (GIF89a)
    58 0.08438600 193.61.191.88   194.80.87.103   TCP       54 17686 > http [ACK] Seq=575 Ack=181 Win=65280 Len=0
    59 0.08438700 193.61.191.88   194.80.87.103   TCP       54 17688 > http [ACK] Seq=560 Ack=180 Win=65280 Len=0
    60 0.08443100 193.61.191.88   194.80.87.103   TCP       54 17687 > http [ACK] Seq=564 Ack=180 Win=65280 Len=0
    61 0.11561000 193.61.191.88   173.194.34.163  TCP       54 17689 > http [ACK] Seq=698 Ack=405 Win=65280 Len=0

⊞ Frame 72: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
⊞ Ethernet II, Src: DellPcba_77:5e:04 (b8:ca:3a:77:5e:04), Dst: NortelNe_f4:92:14 (00:23:0d:f4:92:14)
⊞ Internet Protocol Version 4, Src: 193.61.191.88 (193.61.191.88), Dst: 173.194.34.163 (173.194.34.163)
⊞ Transmission Control Protocol, Src Port: 17692 (17692), Dst Port: http (80), Seq: 1, Ack: 1, Len: 0

0000  00 23 0d f4 92 14 b8 ca  3a 77 5e 04 08 00 45 00   .#...... :w^...E.
0010  00 28 73 8b 40 00 80 06  00 00 c1 3d bf 58 ad c2   .(s.@... ...=.X..
0020  22 a3 45 1c 00 50 04 55  81 4b ba e8 91 00 50 11   ".E..P.U .K....P.
0030  01 00 51 16 00 00                                  ..Q...
```

Ethernet: <live capture in progress> File: C:\... Packets: 87 · Displayed: 87 (100.0%) Profile: Default

10. *Instead of the method above, you can also filter by typing "port 80" in the filter box like below.* 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 will be similar to the one pictured in next screenshot.



Wireshark: Capture Options

Capture

Capture	Interface	Link-layer header	Prom. Mode	Snaplen [B]	Buffer [MB]	Capture Filter
☑	**Ethernet** 193.61.191.88	Ethernet	disabled	default	2	port 80
☐	**VMware Network Adapter VMn...** fe80::5d30:d6e2:98a0:3636 192.168.40.1	Ethernet	disabled	default	2	
☐	**VMware Network Adapter VMn...** fe80::9971:20e3:6865:322c 192.168.33.1	Ethernet	disabled	default	2	

☐ Capture on all interfaces

☐ Use promiscuous mode on all interfaces

Capture Filter: port 80

Capture Files

File:

☐ Use multiple files ☑ Use pcap-ng format

Display Options

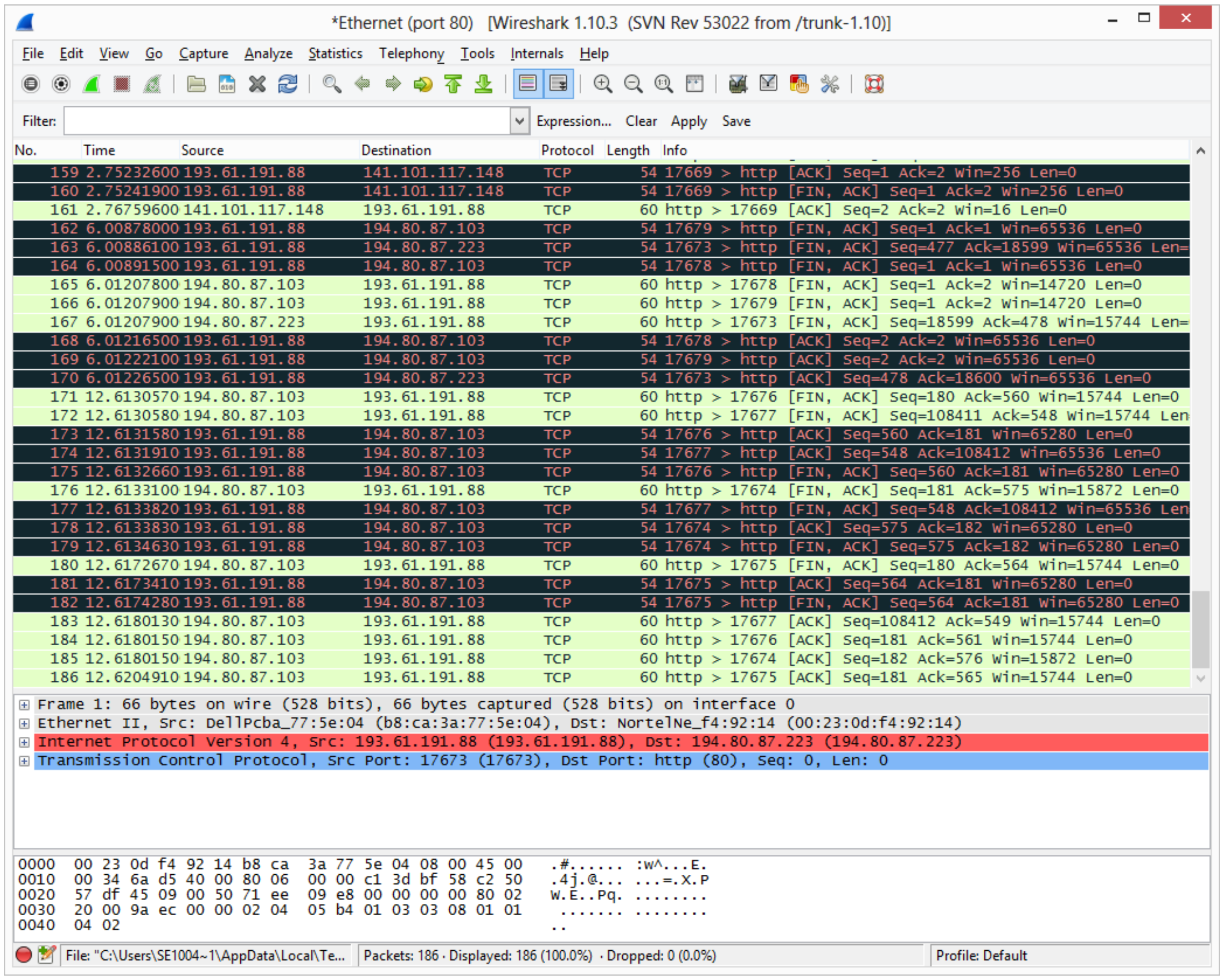
☑ Update list of packets in real time

☑ Automatically scroll during live capture

Figure 5: Setting up the capture options

11. *When the capture is started, visit a site which uses http (not https).* This time, the packets will be recorded by Wireshark as the content is transferred. Try again, http://www.ulster.ac.uk in your browser.

12. *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. I recommend you stop recording by pressing the Red Stop button on the inter-face as this will make it easier for you to scroll back to the start of the request to find the "200 OK" packet. An example is shown below. Congratulations, you have captured a trace!


*Ethernet (port 80) [Wireshark 1.10.3 (SVN Rev 53022 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time        Source          Destination      Protocol Length Info
 159 2.75232600 193.61.191.88   141.101.117.148  TCP      54 17669 > http [ACK] Seq=1 Ack=2 Win=256 Len=0
 160 2.75241900 193.61.191.88   141.101.117.148  TCP      54 17669 > http [FIN, ACK] Seq=1 Ack=2 Win=256 Len=0
 161 2.76759600 141.101.117.148 193.61.191.88    TCP      60 http > 17669 [ACK] Seq=2 Ack=2 Win=16 Len=0
 162 6.00878000 193.61.191.88   194.80.87.103    TCP      54 17679 > http [FIN, ACK] Seq=1 Ack=1 Win=65536 Len=0
 163 6.00886100 193.61.191.88   194.80.87.223    TCP      54 17673 > http [FIN, ACK] Seq=477 Ack=18599 Win=65536 Len=
 164 6.00891500 193.61.191.88   194.80.87.103    TCP      54 17678 > http [FIN, ACK] Seq=1 Ack=1 Win=65536 Len=0
 165 6.01207800 194.80.87.103   193.61.191.88    TCP      60 http > 17678 [FIN, ACK] Seq=1 Ack=2 Win=14720 Len=0
 166 6.01207900 194.80.87.103   193.61.191.88    TCP      60 http > 17679 [FIN, ACK] Seq=1 Ack=2 Win=14720 Len=0
 167 6.01207900 194.80.87.223   193.61.191.88    TCP      60 http > 17673 [FIN, ACK] Seq=18599 Ack=478 Win=15744 Len=
 168 6.01216500 193.61.191.88   194.80.87.103    TCP      54 17678 > http [ACK] Seq=2 Ack=2 Win=65536 Len=0
 169 6.01222100 193.61.191.88   194.80.87.103    TCP      54 17679 > http [ACK] Seq=2 Ack=2 Win=65536 Len=0
 170 6.01226500 193.61.191.88   194.80.87.223    TCP      54 17673 > http [ACK] Seq=478 Ack=18600 Win=65536 Len=0
 171 12.6130570 194.80.87.103   193.61.191.88    TCP      60 http > 17676 [FIN, ACK] Seq=180 Ack=560 Win=15744 Len=0
 172 12.6130580 194.80.87.103   193.61.191.88    TCP      60 http > 17677 [FIN, ACK] Seq=108411 Ack=548 Win=15744 Len
 173 12.6131580 193.61.191.88   194.80.87.103    TCP      54 17676 > http [ACK] Seq=560 Ack=181 Win=65280 Len=0
 174 12.6131910 193.61.191.88   194.80.87.103    TCP      54 17677 > http [ACK] Seq=548 Ack=108412 Win=65536 Len=0
 175 12.6132660 193.61.191.88   194.80.87.103    TCP      54 17676 > http [FIN, ACK] Seq=560 Ack=181 Win=65280 Len=0
 176 12.6133100 194.80.87.103   193.61.191.88    TCP      60 http > 17674 [FIN, ACK] Seq=181 Ack=575 Win=15872 Len=0
 177 12.6133820 193.61.191.88   194.80.87.103    TCP      54 17677 > http [FIN, ACK] Seq=548 Ack=108412 Win=65536 Len
 178 12.6133830 193.61.191.88   194.80.87.103    TCP      54 17674 > http [ACK] Seq=575 Ack=182 Win=65280 Len=0
 179 12.6134630 193.61.191.88   194.80.87.103    TCP      54 17674 > http [FIN, ACK] Seq=575 Ack=182 Win=65280 Len=0
 180 12.6172670 194.80.87.103   193.61.191.88    TCP      60 http > 17675 [FIN, ACK] Seq=180 Ack=564 Win=15744 Len=0
 181 12.6173410 193.61.191.88   194.80.87.103    TCP      54 17675 > http [ACK] Seq=564 Ack=181 Win=65280 Len=0
 182 12.6174280 193.61.191.88   194.80.87.103    TCP      54 17675 > http [FIN, ACK] Seq=564 Ack=181 Win=65280 Len=0
 183 12.6180130 194.80.87.103   193.61.191.88    TCP      60 http > 17677 [ACK] Seq=108412 Ack=549 Win=15744 Len=0
 184 12.6180150 194.80.87.103   193.61.191.88    TCP      60 http > 17676 [ACK] Seq=181 Ack=561 Win=15744 Len=0
 185 12.6180150 194.80.87.103   193.61.191.88    TCP      60 http > 17674 [ACK] Seq=182 Ack=576 Win=15872 Len=0
 186 12.6204910 194.80.87.103   193.61.191.88    TCP      60 http > 17675 [ACK] Seq=181 Ack=565 Win=15744 Len=0
```

```
⊞ Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
⊞ Ethernet II, Src: DellPcba_77:5e:04 (b8:ca:3a:77:5e:04), Dst: NortelNe_f4:92:14 (00:23:0d:f4:92:14)
⊞ Internet Protocol Version 4, Src: 193.61.191.88 (193.61.191.88), Dst: 194.80.87.223 (194.80.87.223)
⊞ Transmission Control Protocol, Src Port: 17673 (17673), Dst Port: http (80), Seq: 0, Len: 0
```

```
0000  00 23 0d f4 92 14 b8 ca  3a 77 5e 04 08 00 45 00   .#...... :w^...E.
0010  00 34 6a d5 40 00 80 06  00 00 c1 3d bf 58 c2 50   .4j.@... ...=.X.P
0020  57 df 45 09 00 50 71 ee  09 e8 00 00 00 00 80 02   W.E..Pq. ........
0030  20 00 9a ec 00 00 02 04  05 b4 01 03 03 08 01 01    ....... ........
0040  04 02                                              ..
```

```
File: "C:\Users\SE1004~1\AppData\Local\Te...  Packets: 186 · Displayed: 186 (100.0%) · Dropped: 0 (0.0%)  Profile: Default
```

Figure 6: Packet trace of web browsing traffic

Step 2: Inspect the Trace

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.

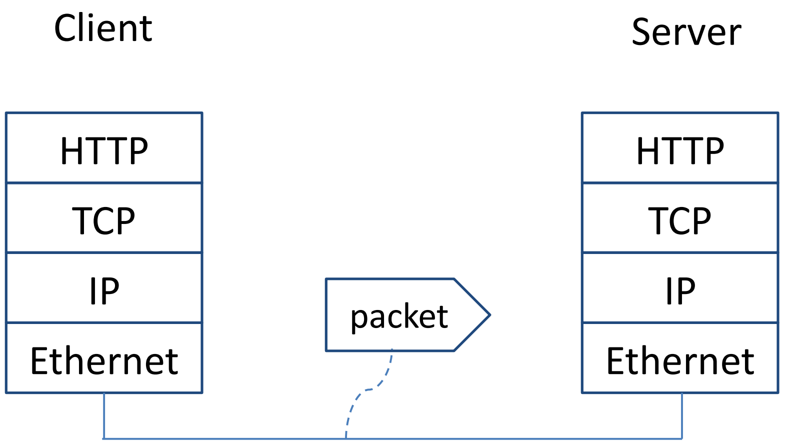


Figure 7: Protocol stack for a web fetch

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". 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.

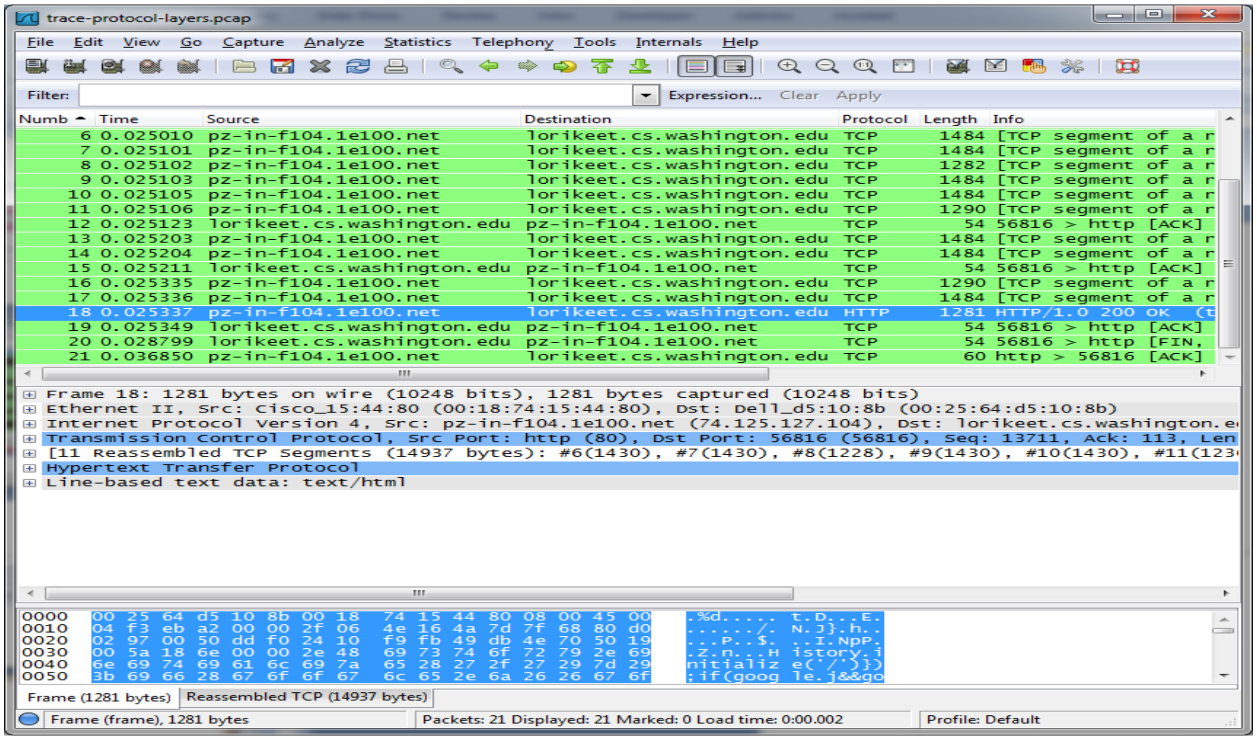
- Then come IP, TCP, and HTTP, which are just as we wanted. Note that the order is from the bottom of the protocol stack upwards. This is because as packets are passed down the stack, the header information of the lower layer protocol is added to the front of the information from the higher layer protocol, as in Fig. 1-15 of your text. That is, the lower layer protocols come first in the packet "on the wire".

Step 3: Inspect the Trace Again

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 something like “[... 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
```

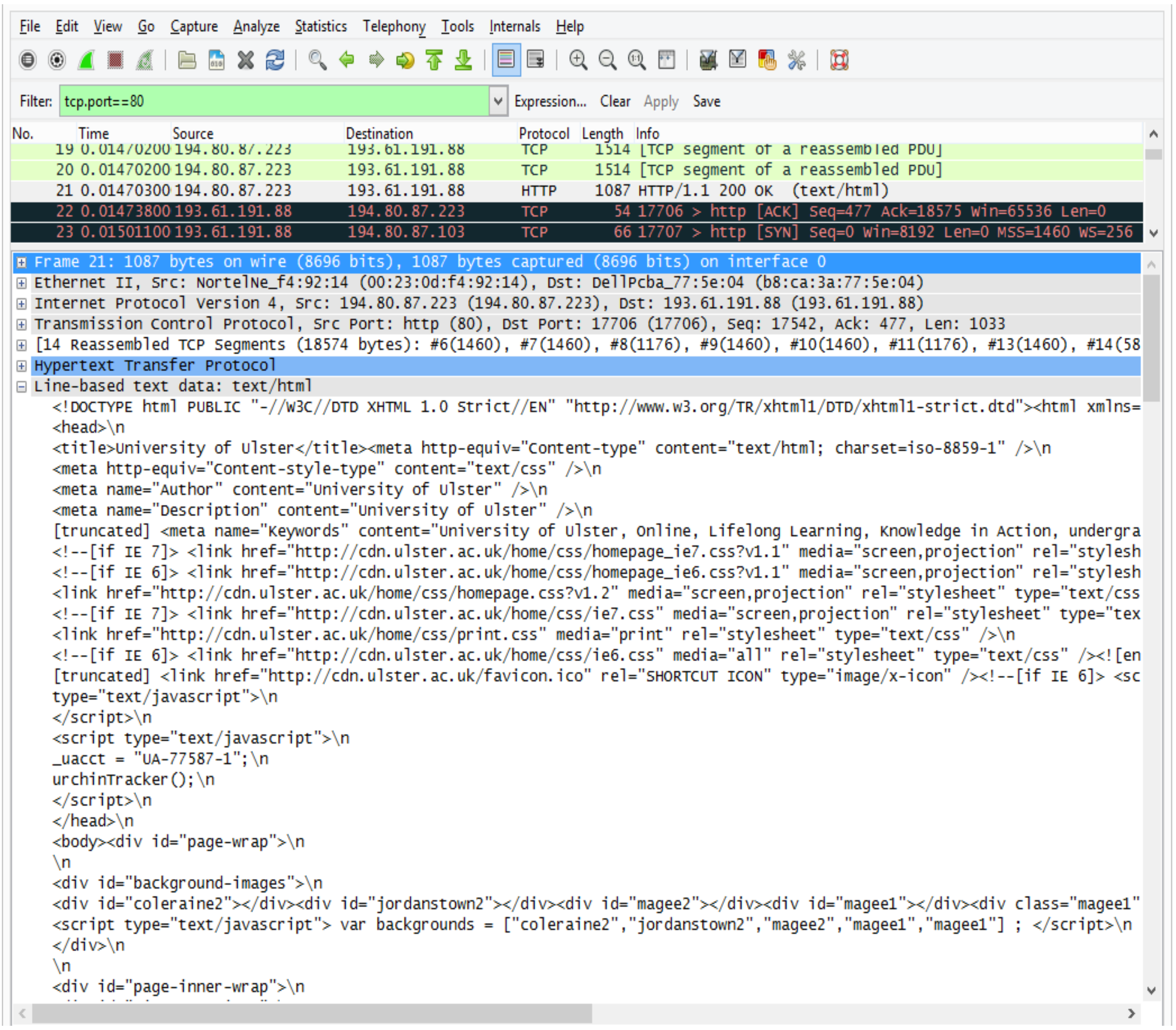
```
0000  00 25 64 d5 10 8b 00 18  74 15 44 80 08 00 45 00   .%d..... t.D...E.
0010  04 f3 eb a2 00 00 2f 06  4e 16 4a 7d 7f 68 80 d0   ....../. N.J}.h..
0020  02 97 00 50 dd f0 24 10  f9 fb 49 db 4e 70 50 19   ...P..$. ..I.NpP.
0030  00 5a 18 6e 00 00 2e 48  69 73 74 6f 72 79 2e 69   .Z.n...H istory.i
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 8: Inspecting a HTTP “200 OK” response

1. Locate the **HTTP GET** packet which has **“200 OK (text/html)”** in the Info field, denoting a successful fetch. See below for instance in the third line in white.


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

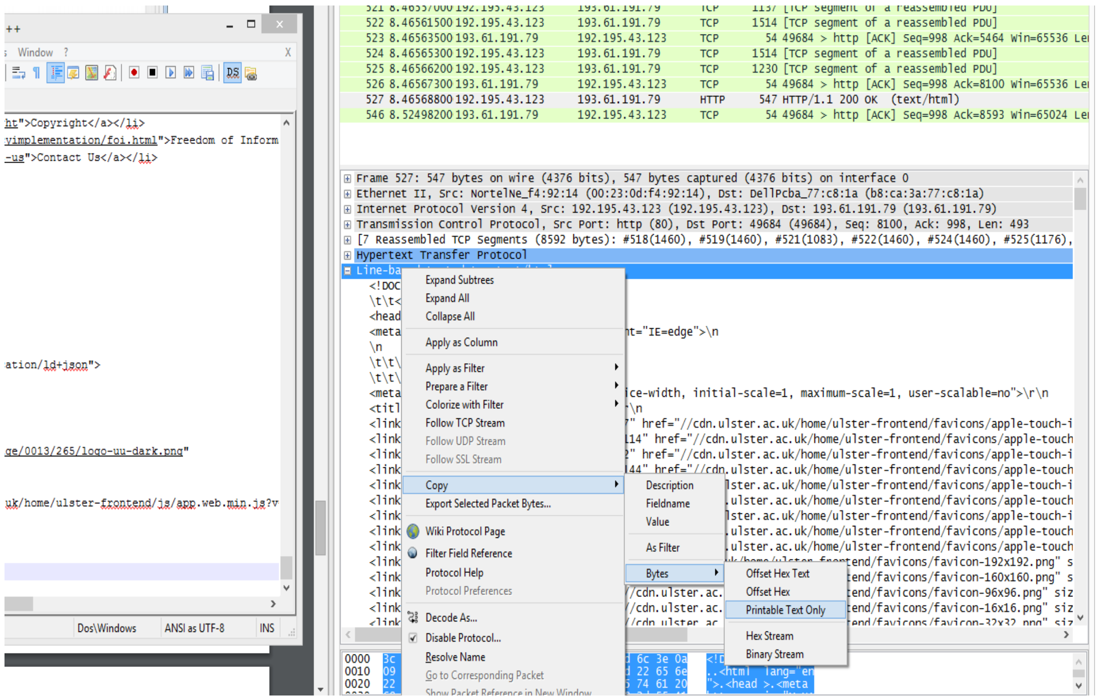
Filter: tcp.port==80

```
No.   Time        Source         Destination    Protocol Length Info
   19 0.01470200 194.80.87.223   193.61.191.88   TCP      1514 [TCP segment of a reassembled PDU]
   20 0.01470200 194.80.87.223   193.61.191.88   TCP      1514 [TCP segment of a reassembled PDU]
   21 0.01470300 194.80.87.223   193.61.191.88   HTTP     1087 HTTP/1.1 200 OK  (text/html)
   22 0.01473800 193.61.191.88   194.80.87.223   TCP        54 17706 > http [ACK] Seq=477 Ack=18575 Win=65536 Len=0
   23 0.01501100 193.61.191.88   194.80.87.103   TCP        66 17707 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256
```

```
⊞ Frame 21: 1087 bytes on wire (8696 bits), 1087 bytes captured (8696 bits) on interface 0
⊞ Ethernet II, Src: NortelNe_f4:92:14 (00:23:0d:f4:92:14), Dst: DellPcba_77:5e:04 (b8:ca:3a:77:5e:04)
⊞ Internet Protocol Version 4, Src: 194.80.87.223 (194.80.87.223), Dst: 193.61.191.88 (193.61.191.88)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 17706 (17706), Seq: 17542, Ack: 477, Len: 1033
⊞ [14 Reassembled TCP Segments (18574 bytes): #6(1460), #7(1460), #8(1176), #9(1460), #10(1460), #11(1176), #13(1460), #14(58
⊞ Hypertext Transfer Protocol
⊟ Line-based text data: text/html
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html xmlns=
    <head>\n
    <title>University of Ulster</title><meta http-equiv="Content-type" content="text/html; charset=iso-8859-1" />\n
    <meta http-equiv="Content-style-type" content="text/css" />\n
    <meta name="Author" content="University of Ulster" />\n
    <meta name="Description" content="University of Ulster" />\n
    [truncated] <meta name="Keywords" content="University of Ulster, Online, Lifelong Learning, Knowledge in Action, undergra
    <!--[if IE 7]> <link href="http://cdn.ulster.ac.uk/home/css/homepage_ie7.css?v1.1" media="screen,projection" rel="stylesh
    <!--[if IE 6]> <link href="http://cdn.ulster.ac.uk/home/css/homepage_ie6.css?v1.1" media="screen,projection" rel="stylesh
    <link href="http://cdn.ulster.ac.uk/home/css/homepage.css?v1.2" media="screen,projection" rel="stylesheet" type="text/css
    <!--[if IE 7]> <link href="http://cdn.ulster.ac.uk/home/css/ie7.css" media="screen,projection" rel="stylesheet" type="tex
    <link href="http://cdn.ulster.ac.uk/home/css/print.css" media="print" rel="stylesheet" type="text/css" />\n
    <!--[if IE 6]> <link href="http://cdn.ulster.ac.uk/home/css/ie6.css" media="all" rel="stylesheet" type="text/css" /><![en
    [truncated] <link href="http://cdn.ulster.ac.uk/favicon.ico" rel="SHORTCUT ICON" type="image/x-icon" /><!--[if IE 6]> <sc
    type="text/javascript">\n
    </script>\n
    <script type="text/javascript">\n
    _uacct = "UA-77587-1";\n
    urchinTracker();\n
    </script>\n
    </head>\n
    <body><div id="page-wrap">\n
    \n
    <div id="background-images">\n
    <div id="coleraine2"></div><div id="jordanstown2"></div><div id="magee2"></div><div id="magee1"></div><div class="magee1"
    <script type="text/javascript"> var backgrounds = ["coleraine2","jordanstown2","magee2","magee1","magee1"] ; </script>\n
    </div>\n
    \n
    <div id="page-inner-wrap">\n
```

2. Scroll down in the middle pane to **[+] Line-based text data: text/html** and right click with your mouse. (See above)

3. From the context menu which now appears, select **Copy -→ Bytes → Printable Text Only** You should see a window similar to the following popup window below.



```
522 8.46561500 192.195.43.123   193.61.191.79    TCP    1514 [TCP segment of a reassembled PDU]
523 8.46563500 193.61.191.79    192.195.43.123   TCP      54 49684 > http [ACK] Seq=998 Ack=5464 Win=65536 Le
524 8.46565300 192.195.43.123   193.61.191.79    TCP    1514 [TCP segment of a reassembled PDU]
525 8.46566200 192.195.43.123   193.61.191.79    TCP    1230 [TCP segment of a reassembled PDU]
526 8.46567300 193.61.191.79    192.195.43.123   TCP      54 49684 > http [ACK] Seq=998 Ack=8100 Win=65536 Le
527 8.46568800 192.195.43.123   193.61.191.79    HTTP    547 HTTP/1.1 200 OK  (text/html)
546 8.52498200 193.61.191.79    192.195.43.123   TCP      54 49684 > http [ACK] Seq=998 Ack=8593 Win=65024 Le
```

```
⊞ Frame 527: 547 bytes on wire (4376 bits), 547 bytes captured (4376 bits) on interface 0
⊞ Ethernet II, Src: NortelNe_f4:92:14 (00:23:0d:f4:92:14), Dst: DellPcba_77:c8:1a (b8:ca:3a:77:c8:1a)
⊞ Internet Protocol Version 4, Src: 192.195.43.123 (192.195.43.123), Dst: 193.61.191.79 (193.61.191.79)
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 49684 (49684), Seq: 8100, Ack: 998, Len: 493
⊞ [7 Reassembled TCP Segments (8592 bytes): #518(1460), #519(1460), #521(1083), #522(1460), #524(1460), #525(1176),
⊞ Hypertext Transfer Protocol
⊟ Line-ba
```

4. This copies all the code for the Ulster home page.

5. Open up a text editor such as notepad++ and paste this HTML code into a new document. Save as **UlsterHomePage.html** in a directory such as W: or downloads. Next open up explorer and scroll to where you have saved the file and open it in the default browser by clicking on it.

6. Note how much of the Ulster home page actually gets sent over the wire to your PC.

Step 4: Packet Structure

Examine a HTTP GET packet that shows the position and size in bytes of the TCP, IP and Ethernet protocol headers. Note the range of the Ethernet header and the Ethernet payload that IP passes to Ethernet to send over the network. Note also the range of the IP header and the IP payload.

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.

Answer

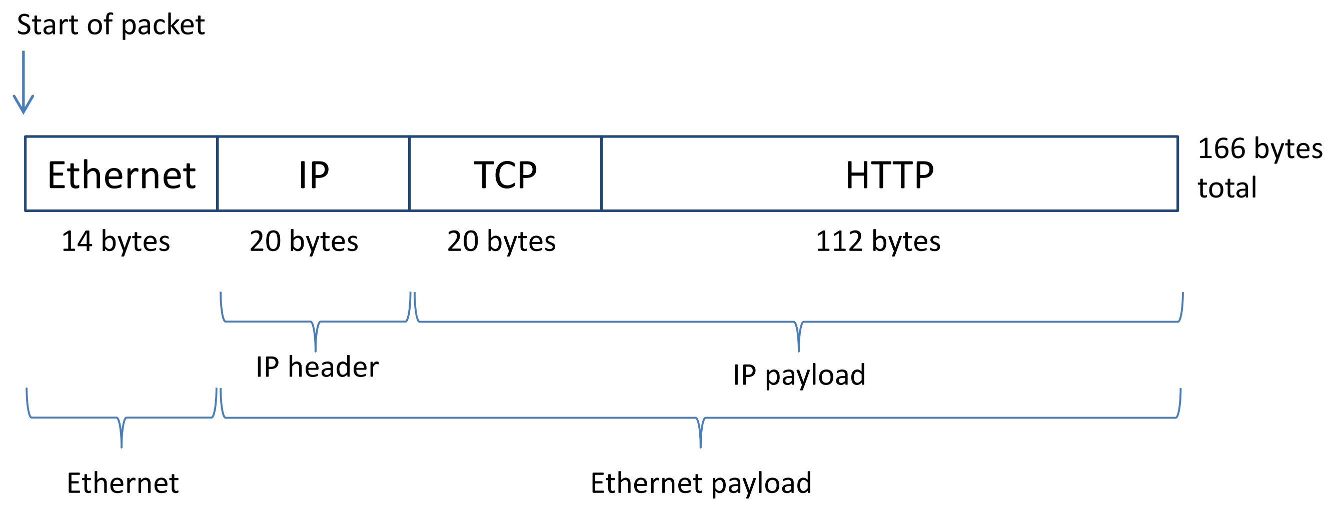


Figure 9: Protocol layer structure of the HTTP GET packet

There are several features to note:

- The order of the headers (Ethernet, IP, TCP, HTTP) is the protocol stack from the bottom upwards because the lower layers are outermost in the packet as it travels through the network.
- Observant students will note some differences between the Ethernet header size in Wireshark and in the text that will be explored in later labs.
- The size of the IP and TCP headers is normally around 20 bytes each, but it may be larger in some cases depending on the OS, e.g., IPv6 instead of IPv4 and optional TCP header fields might double these numbers.
- The size of the HTTP message will vary depending on what tool and URL is used to send the web request. For wget/curl, it is likely to be around 100-300 bytes.
- The Ethernet payload comprises everything beyond the Ethernet header. That is, Ethernet does not understand the IP / TCP / HTTP internal structure; it is up to higher layers to determine their headers and message boundaries.
- Similarly, the IP payload comprises everything beyond the IP header. Note that neither the IP header nor payload covers the Ethernet header.

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"?*

   Answer. The demultiplexing key for Ethernet is the Type field. It holds 0x800 when the higher layer is 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"?*

   Answer: The demultiplexing key for IP is the Protocol field. It has value 6 when the higher layer is TCP.