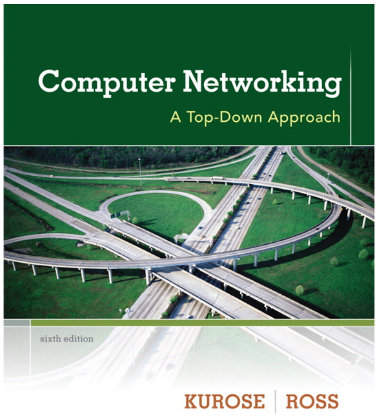
Wireshark Lab: HTTP v6.4

Supplement to *Computer Networking: A Top-Down Approach, 6th ed.,* J.F. Kurose and K.W. Ross

(With Modification by Rong Zheng@McMaster)

"Tell me and I forget. Show me and I remember. Involve me and I understand." Chinese proverb



One's understanding of network protocols can often be greatly deepened by "seeing protocols in action" and by "playing around with protocols" – observing the sequence of messages exchanged between two protocol entities, delving down into the details of protocol operation, and causing protocols to perform certain actions and then observing these actions and their consequences. This can be done in simulated scenarios or in a "real" network environment such as the Internet. In the Wireshark labs you'll be doing in this course, you'll be running various network applications in different scenarios using your own computer (or you can borrow a friends; let me know if you don't have access to a computer where you can install/run Wireshark). You'll observe the network protocols in your computer "in action," interacting and exchanging messages with protocol entities executing elsewhere in the Internet. Thus, you and your computer will be an integral part of these "live" labs. You'll observe, and you'll learn, by doing.

In this first Wireshark lab, you'll get acquainted with Wireshark, and make some simple packet captures and observations.

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. Recall from the discussion from section 1.5 in the text (Figure 1.24[1]) that 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. 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.

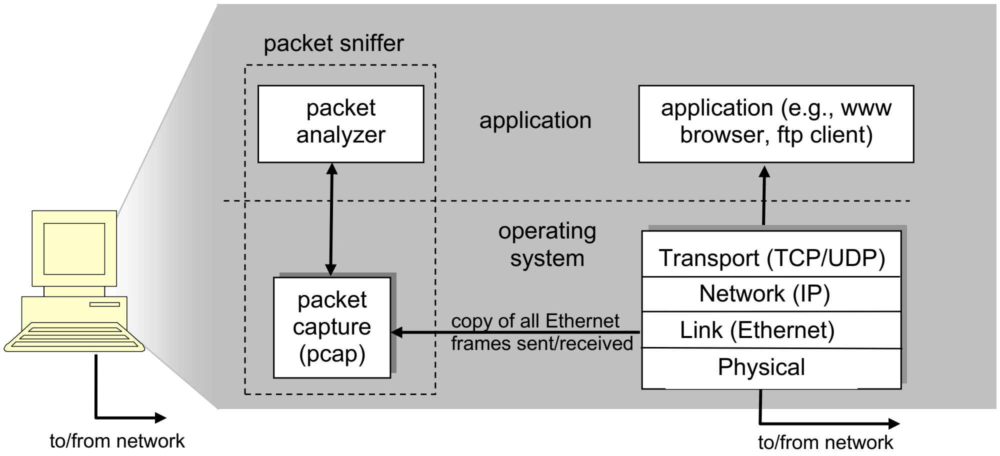


Figure 1: Packet sniffer structure

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," as shown in Figure 2.8 in the text.

We will be using the Wireshark packet sniffer [http://www.wireshark.org/] for these labs, allowing us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, Wireshark is a packet analyzer that uses a packet capture library in your computer). Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers. It's 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/), man pages (http://www.wireshark.org/docs/man-pages/), and a detailed FAQ

[1] References to figures and sections are for the 6th edition of our text, *Computer Networks, A Top-down Approach, 6th ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2012.*

(http://www.wireshark.org/faq.html), 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's running allows Wireshark to do so).

Getting Wireshark

In order to run Wireshark, you will need to have access to a computer that supports both Wireshark and the *libpcap* or *WinPCap* packet capture library. The *libpcap* software will be installed for you, if it is not installed within your operating system, when you install Wireshark. See http://www.wireshark.org/download.html for a list of supported operating systems and download sites

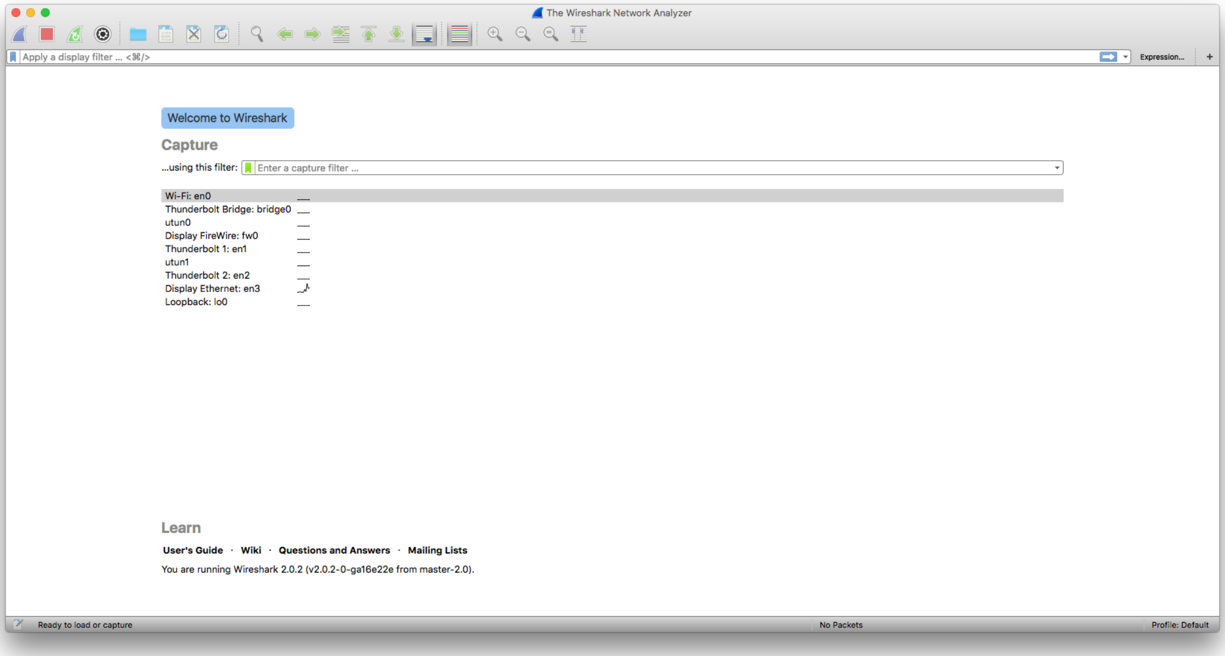
Download and install the Wireshark software:

- Go to http://www.wireshark.org/download.html and download and install the Wireshark binary for your computer (the subsequent example assumes Wireshark
.

The Wireshark FAQ has a number of helpful hints and interesting tidbits of information, particularly if you have trouble installing or running Wireshark.

Running Wireshark

When you run the Wireshark program, you'll get a startup screen, as shown below: (Note that GUI may differ depending on the version and OS you use)


The Wireshark Network Analyzer
Apply a display filter ... <⌘/>
Expression...
+
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists

Wi-Fi: en0
Thunderbolt Bridge: bridge0
utun0
Display FireWire: fw0
Thunderbolt 1: en1
utun1
Thunderbolt 2: en2
Display Ethernet: en3
Loopback: lo0

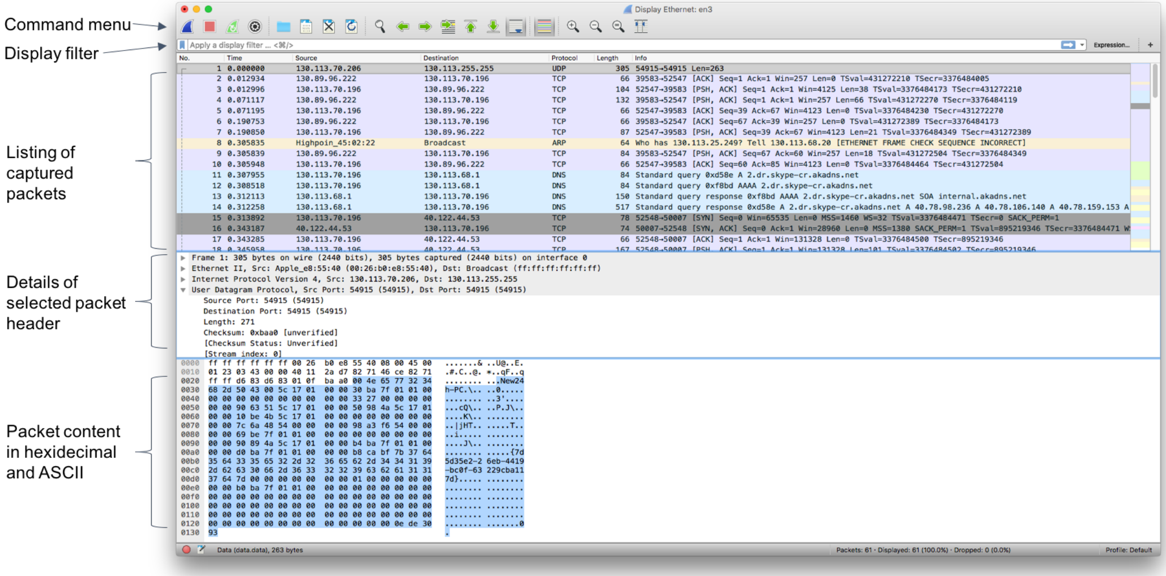
You are running Wireshark 2.0.2 (v2.0.2-0-ga16e22e from master-2.0).

Ready to load or capture | No Packets | Profile: Default

Figure 2: Initial Wireshark Screen

Take a look at the center the screen – you'll see a list of network interfaces on your computer. Once you click on an interface, Wireshark will capture all packets on that interface. In the example above, there are multiple interfaces, some corresponding to real network devices (e.g., WiFi:en0, Display Ethernet:en3), and some corresponding to virtual interfaces (e.g, loopback interface lo0, utun0 for VPN connections).

If you click on one of these interfaces 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 or click the red square on the main toolbar.


Command menu
Display filter
Listing of captured packets
Details of selected packet header
Packet content in hexidecimal and ASCII
Display Ethernet: en3
Apply a display filter ... <⌘/>
Expression...
+
Data (data.data), 263 bytes
Packets: 61 · Displayed: 61 (100.0%) · Dropped: 0 (0.0%)
Profile: Default

```
No.  Time      Source           Destination      Protocol Length Info
   1 0.000000  130.113.70.206   130.113.255.255  UDP      305 54915→54915 Len=263
   2 0.012934  130.89.96.222    130.113.70.196   TCP       66 39583→52547 [ACK] Seq=1 Ack=1 Win=257 Len=0 TSval=431272210 TSecr=3376484005
   3 0.012996  130.113.70.196   130.89.96.222    TCP      104 52547→39583 [PSH, ACK] Seq=1 Ack=1 Win=4125 Len=38 TSval=3376484173 TSecr=431272210
   4 0.071117  130.89.96.222    130.113.70.196   TCP      132 39583→52547 [PSH, ACK] Seq=1 Ack=1 Win=257 Len=66 TSval=431272270 TSecr=3376484119
   5 0.071195  130.113.70.196   130.89.96.222    TCP       66 52547→39583 [ACK] Seq=39 Ack=67 Win=4123 Len=0 TSval=3376484230 TSecr=431272270
   6 0.190753  130.89.96.222    130.113.70.196   TCP       66 39583→52547 [ACK] Seq=67 Ack=39 Win=257 Len=0 TSval=431272389 TSecr=3376484173
   7 0.190850  130.113.70.196   130.89.96.222    TCP       87 52547→39583 [PSH, ACK] Seq=39 Ack=67 Win=4123 Len=21 TSval=3376484349 TSecr=431272389
   8 0.305835  Highpoin_45:02:22 Broadcast       ARP       64 Who has 130.113.25.249? Tell 130.113.68.20 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
   9 0.305839  130.89.96.222    130.113.70.196   TCP       84 39583→52547 [PSH, ACK] Seq=67 Ack=60 Win=257 Len=18 TSval=431272504 TSecr=3376484349
  10 0.305948  130.113.70.196   130.89.96.222    TCP       66 52547→39583 [ACK] Seq=60 Ack=85 Win=4123 Len=0 TSval=3376484464 TSecr=431272504
  11 0.307955  130.113.70.196   130.113.68.1     DNS       84 Standard query 0xd58e A 2.dr.skype-cr.akadns.net
  12 0.308518  130.113.70.196   130.113.68.1     DNS       84 Standard query 0xf8bd AAAA 2.dr.skype-cr.akadns.net
  13 0.312113  130.113.68.1     130.113.70.196   DNS      150 Standard query response 0xf8bd AAAA 2.dr.skype-cr.akadns.net SOA internal.akadns.net
  14 0.312258  130.113.68.1     130.113.70.196   DNS      517 Standard query response 0xd58e A 2.dr.skype-cr.akadns.net A 40.78.98.236 A 40.78.106.140 A 40.78.159.153 A
  15 0.313892  130.113.70.196   40.122.44.53     TCP       78 52548→50007 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=3376484471 TSecr=0 SACK_PERM=1
  16 0.343187  40.122.44.53     130.113.70.196   TCP       74 50007→52548 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1380 SACK_PERM=1 TSval=895219346 TSecr=3376484471 W
  17 0.343285  130.113.70.196   40.122.44.53     TCP       66 52548→50007 [ACK] Seq=1 Ack=1 Win=131328 Len=0 TSval=3376484500 TSecr=895219346
  18 0.345958  130.113.70.196   40.122.44.53     TCP      167 52548→50007 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=101 TSval=3376484502 TSecr=895219346
```

```
▶ Frame 1: 305 bytes on wire (2440 bits), 305 bytes captured (2440 bits) on interface 0
▶ Ethernet II, Src: Apple_e8:55:40 (00:26:b0:e8:55:40), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▶ Internet Protocol Version 4, Src: 130.113.70.206, Dst: 130.113.255.255
▼ User Datagram Protocol, Src Port: 54915 (54915), Dst Port: 54915 (54915)
    Source Port: 54915 (54915)
    Destination Port: 54915 (54915)
    Length: 271
    Checksum: 0xbaa0 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
```

```
0000  ff ff ff ff ff ff 00 26  b0 e8 55 40 08 00 45 00   .......& ..U@..E.
0010  01 23 03 43 00 00 40 11  2a d7 82 71 46 ce 82 71   .#.C..@. *..qF..q
0020  ff ff d6 83 d6 83 01 0f  ba a0 00 4e 65 77 32 34   ........ ...New24
0030  68 2d 50 43 00 5c 17 01  00 00 30 ba 7f 01 01 00   h-PC.\.. ..0.....
0040  00 00 00 00 00 00 00 00  00 00 33 27 00 00 00 00   ........ ..3'....
0050  00 00 90 63 51 5c 17 01  00 00 50 98 4a 5c 17 01   ...cQ\.. ..P.J\..
0060  00 00 10 be 4b 5c 17 01  00 00 00 00 00 00 00 00   ....K\.. ........
0070  00 00 7c 6a 48 54 00 00  00 00 98 a3 f6 54 00 00   ..|jHT.. .....T..
0080  00 00 69 be 7f 01 01 00  00 00 00 00 00 00 00 00   ..i..... ........
0090  00 00 90 89 4a 5c 17 01  00 00 b4 ba 7f 01 01 00   ....J\.. ........
00a0  00 00 d0 ba 7f 01 01 00  00 00 b8 ca bf 7b 37 64   ........ .....{7d
00b0  35 64 33 35 65 32 2d 32  36 65 62 2d 34 34 31 39   5d35e2-2 6eb-4419
00c0  2d 62 63 30 66 2d 36 33  32 32 39 63 62 61 31 31   -bc0f-63 229cba11
00d0  37 64 7d 00 00 00 00 00  00 00 01 00 00 00 00 00   7d}..... ........
00e0  00 00 b0 ba 7f 01 01 00  00 00 00 00 00 00 00 00   ........ ........
00f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0100  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0110  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0120  00 00 00 00 00 00 00 00  00 00 00 00 00 0e de 30   ........ .......0
0130  93                                                 .
```

Figure 3: Wireshark Graphical User Interface, during packet capture and analysis

The Wireshark interface has five major components:

- The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data, and exit the Wireshark application. 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 the 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. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest-level protocol that sent or received this packet are also provided.
- 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 (and hence the packet-header and packet-contents windows). In the example below, we'll use the packet-display filter field to have Wireshark hide (not display) packets except those that correspond to HTTP messages.

We're now ready to use Wireshark to investigate protocols in operation. Next, we'll explore several aspects of the HTTP protocol: the basic GET/response interaction, HTTP message formats, retrieving large HTML files, retrieving HTML files with embedded objects, and HTTP authentication and security. Before beginning these labs, you might want to review Section 2.2 of the text.[2]

When you hand in your assignment, include screenshots and annotate the outputs so that it's clear where in the output you're getting the information for your answer. Marks will be deducted if no explanation is given or the timestamps are unreasonable.

1. The Basic HTTP GET/response interaction

[2] References to figures and sections are for the 6th edition of our text, *Computer Networks, A Top-down Approach, 6th ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2012.*

Let's begin our exploration of HTTP by downloading a very simple HTML file - one that is very short, and contains no embedded objects. Do the following:

1. Start up your web browser. For this assignment, we recommend that you use Google Chrome for more consistent behavior.
2. Start up the Wireshark packet sniffer, as described in the Introductory lab (but don't yet begin packet capture). Enter "http" (just the letters, not the quotation marks) in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window. (We're only interested in the HTTP protocol here, and don't want to see the clutter of all captured packets).
3. Wait a bit more than one minute (we'll see why shortly), and then begin Wireshark packet capture.
4. Enter the following to your browser

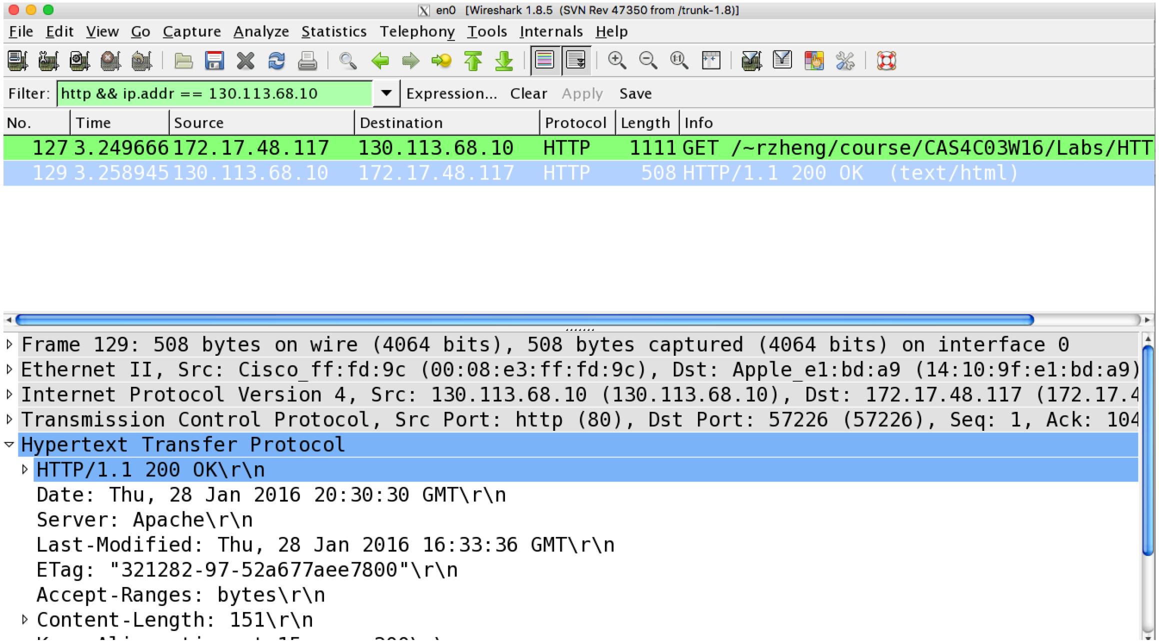
http://www.cas.mcmaster.ca/~rzheng/course/wireshark/HTTP/HTTP-wireshark-file1.html

   Your browser should display the very simple, one-line HTML file.

5. Stop Wireshark packet capture.

!! Make sure the URL uses HTTP not HTTPS!

Your Wireshark window should look similar to the window shown in Figure 1. If you are unable to run Wireshark on a live network connection, you can download a packet trace that was created when the steps above were followed.


en0 [Wireshark 1.8.5 (SVN Rev 47350 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

Filter: http && ip.addr == 130.113.68.10

```
No.  Time      Source         Destination    Protocol Length Info
 127 3.249666 172.17.48.117  130.113.68.10  HTTP       1111 GET /~rzheng/course/CAS4C03W16/Labs/HTT
 129 3.258945 130.113.68.10  172.17.48.117  HTTP        508 HTTP/1.1 200 OK  (text/html)
```

```
▷ Frame 129: 508 bytes on wire (4064 bits), 508 bytes captured (4064 bits) on interface 0
▷ Ethernet II, Src: Cisco_ff:fd:9c (00:08:e3:ff:fd:9c), Dst: Apple_e1:bd:a9 (14:10:9f:e1:bd:a9)
▷ Internet Protocol Version 4, Src: 130.113.68.10 (130.113.68.10), Dst: 172.17.48.117 (172.17.4
▷ Transmission Control Protocol, Src Port: http (80), Dst Port: 57226 (57226), Seq: 1, Ack: 104
▽ Hypertext Transfer Protocol
  ▷ HTTP/1.1 200 OK\r\n
    Date: Thu, 28 Jan 2016 20:30:30 GMT\r\n
    Server: Apache\r\n
    Last-Modified: Thu, 28 Jan 2016 16:33:36 GMT\r\n
    ETag: "321282-97-52a677aee7800"\r\n
    Accept-Ranges: bytes\r\n
  ▷ Content-Length: 151\r\n
```

Figure 1: Wireshark Display after http://www.cas.mcmaster.ca/~rzheng/course/wireshark/HTTP/HTTP-wireshark-file1.html has been retrieved by your browser

The example in Figure 1 shows in the packet-listing window that two HTTP messages were captured: the GET message (from your browser to the www.cas.mcmaster.ca web server) and the response message from the server to your browser. The packet-contents window shows details of the selected message (in this case the HTTP OK message, which is highlighted in the packet-listing window). Recall that since the HTTP message was carried inside a TCP segment, which was carried inside an IP datagram, which was carried within an Ethernet frame, Wireshark displays the Frame, Ethernet, IP, and TCP packet information as well. We want to minimize the amount of non-HTTP data displayed (we're interested in HTTP here, and will be investigating these other protocols is later labs), so make sure the boxes at the far left of the Frame, Ethernet, IP and TCP information have a plus sign or a right-pointing triangle (which means there is hidden, undisplayed information), and the HTTP line has a minus sign or a down-pointing triangle (which means that all information about the HTTP message is displayed).

> (*Note:* You should ignore any HTTP GET and response for favicon.ico. If you see a reference to this file, it is your browser automatically asking the server if it (the server) has a small icon file that should be displayed next to the displayed URL in your browser. We'll ignore references to this pesky file in this lab.).

By looking at the information in the HTTP GET and response messages, answer the following questions. When answering the following questions, you should indicate where in the message you've found the information that answers the following questions.

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?
2. What is the IP address of your computer? Of the www.cas.mcmaster.ca server?
3. What is the status code returned from the server to your browser?
4. When was the HTML file that you are retrieving last modified at the server?
5. How many bytes of content are being returned to your browser?

2. The HTTP CONDITIONAL GET/response interaction

Recall from Section 2.2.6 of the text, that most web browsers perform object caching and thus perform a conditional GET when retrieving an HTTP object. Before performing the steps below, make sure your browser's cache is empty. (In Chrome, Firefox, or Microsoft Edge, open the browser menu, select Settings, navigate to Privacy & Security, and clear Cached images and files. In Safari, open Settings, go to Privacy, and choose Manage Website Data, then remove cached data.) Now do the following:

a. Start up your web browser, and make sure your browser's cache is cleared, as discussed above.
b. Start up the Wireshark packet sniffer
c. Enter the following URL into your browser
d. http://www.cas.mcmaster.ca/~rzheng/course/wireshark/HTTP/HTTP-wireshark-file2.html
    a. Your browser should display a very simple five-line HTML file.
e. Click the refresh button on your browser

- Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.

Answer the following questions:

6. How many HTTP GET requests were sent?
7. How many HTTP response messages were received?
8. If multiple response messages were received, what are the difference between them?
9. If instead of clicking the “refresh” button in Step e, you re-enter the URL in the browser or simply enter return when the URL is highlighted, answer Q6 – Q8.

3. Retrieving Long Documents

In our examples thus far, the documents retrieved have been simple and short HTML files. Let’s next see what happens when we download a long HTML file. Do the following:

- Start up your web browser, and make sure your browser’s cache is cleared, as discussed above.
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser

http://www.cas.mcmaster.ca/~rzheng/course/wireshark/HTTP/HTTP-wireshark-file3.html

Your browser should display the rather lengthy US Bill of Rights.

- Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed.

In the packet-listing window, you should see your HTTP GET message and one HTTP response message. The requested HTML file is too large to fit into a single TCP segment, so TCP divides the HTTP response across multiple segments for transmission. When using an http display filter, Wireshark typically shows a single HTTP response message because its TCP dissector reassembles the TCP byte stream internally before handing it to the HTTP dissector. Evidence of this segmentation can be seen by selecting the HTTP response in the packet-list pane and examining the packet content pane, which indicates that the message was reassembled from multiple TCP segments (*look for “Reassembled TCP segments”*). We emphasize that HTTP itself does not define continuation or fragmentation.

Answer the following questions:

10. How many HTTP GET request messages did your browser send? Which packet number in the trace contains the GET message for the Bill or Rights?
11. Which packet number in the trace contains the status code and phrase associated with the response to the HTTP GET request?
12. What is the status code and phrase in the response?
13. How many data-containing TCP segments were needed to carry the single HTTP response and the text of the Bill of Rights?

4 HTTP Authentication

Finally, let's try visiting a web site that is password-protected and examine the sequence of HTTP message exchanged for such a site. The URL http://www.cas.mcmaster.ca/~rzheng/course/wireshark/HTTP/HTTP-wireshark-file5.html is password protected. The username is "4C03_students" (0 as zero not "O", without the quotes), and the password is "4C03" (again, without the quotes). So let's access this "secure" password-protected site. Do the following:

- Make sure your browser's cache is cleared, as discussed above, and close down your browser. Then, start up your browser
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser

http://www.cas.mcmaster.ca/~rzheng/course/wireshark/HTTP/HTTP-wireshark-file5.html

Type the requested user name and password into the pop up box.

- Stop Wireshark packet capture, and enter "http" in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.

Now let's examine the Wireshark output. Answer the following questions:

14. What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?
15. When your browser's sends the HTTP GET message for the second time, what new field is included in the HTTP GET message?

The username (4C03_students) and password (4C03) that you entered are encoded in the string of characters (NEMwM19zdHVkZW50czo0QzAz)following the "`Authorization: Basic`" header in the client's HTTP GET message. While it may appear that your username and password are encrypted, they are simply encoded in a format known as Base64 format. The username and password are *not* encrypted! To see this, go to https://www.base64decode.org and enter the base64-encoded string NEMwM19zdHVkZW50czo0QzAz and decode. *Voila!* You have translated from Base64 encoding to ASCII encoding, and thus should see your username:password! Since anyone can download a tool like Wireshark and sniff packets (not just their own) passing by their network adaptor, and anyone can translate from Base64 to ASCII (you just did it!), it should be clear to you that simple passwords on WWW sites are not secure unless additional measures are taken.

Submission

Include your answers and screen shots if any in a single PDF file named YOUR_MAC_ID_A1.pdf. Submission should be done through Avenue.