

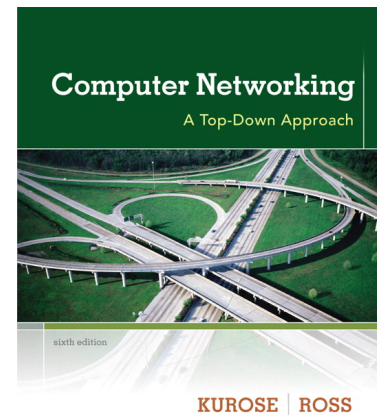
# Wireshark Lab: HTTP v6.4

Supplement to *Computer Networking: A Top-Down Approach*, 6<sup>th</sup> 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

© 2005-21012, J.F Kurose and K.W. Ross, All Rights Reserved



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<sup>1</sup>) 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.

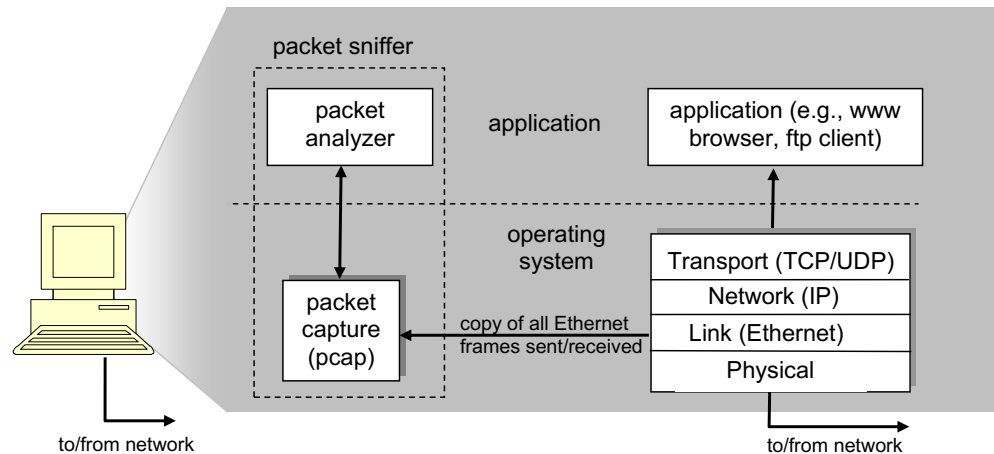


Figure 1: Packet sniffer structure

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/](http://www.wireshark.org/docs/wsug_html_chunked/)), man pages (<http://www.wireshark.org/docs/man-pages/>), and a detailed FAQ

<sup>1</sup> References to figures and sections are for the 6<sup>th</sup> edition of our text, *Computer Networks, A Top-down Approach*, 6<sup>th</sup> 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

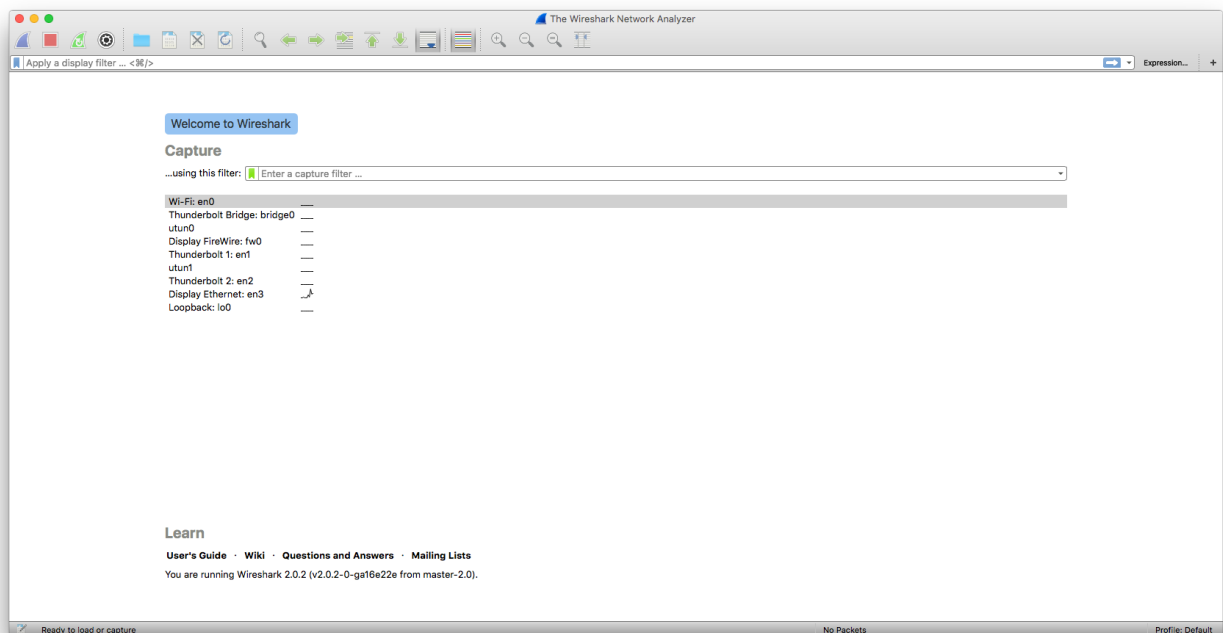
Download and install the Wireshark software:

- 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

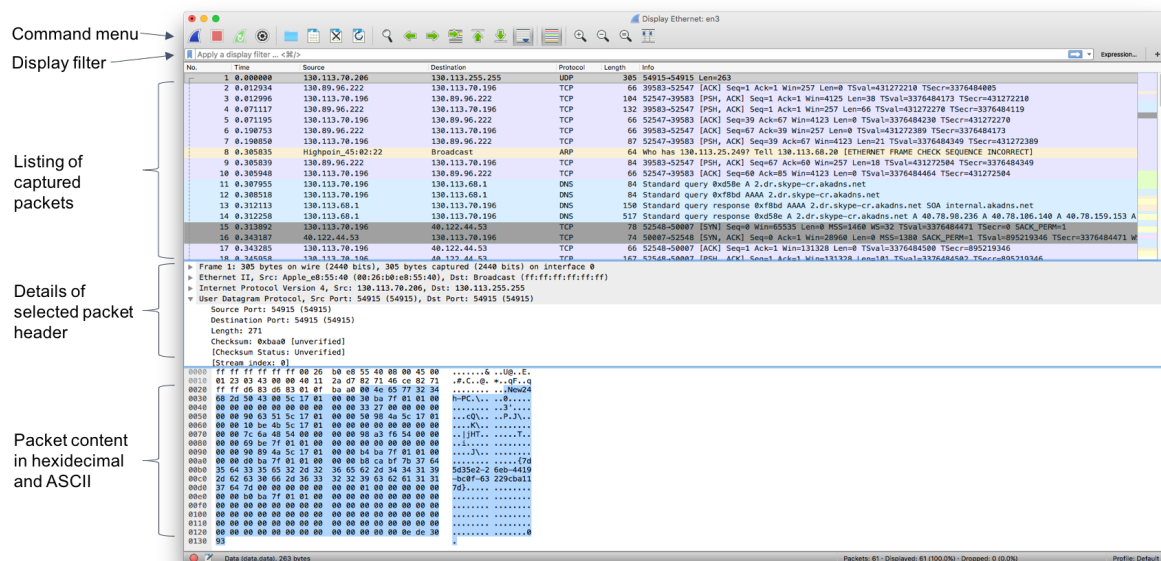
When you run the Wireshark program, you'll get a startup screen, as shown below: (Note that GUI may differ depending on the version and OS you use)



**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.



**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.<sup>2</sup>

**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

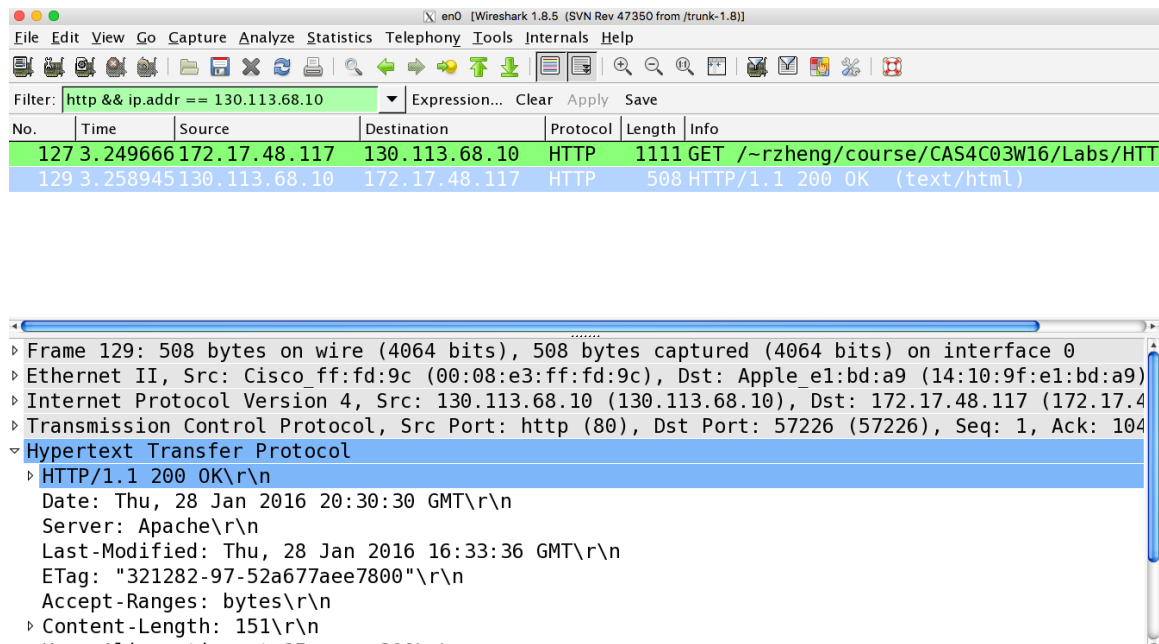
---

<sup>2</sup> References to figures and sections are for the 6<sup>th</sup> edition of our text, *Computer Networks, A Top-down Approach*, 6<sup>th</sup> 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.
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.

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.



**Figure 1:** Wireshark Display after  
<http://www.cas.mcmaster.ca/~rzheng/course/CAS4C03W16/Labs/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](http://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 in 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 languages (if any) does your browser indicate that it can accept to the server?
3. What is the IP address of your computer? Of the www.cas.mcmaster.ca server?
4. What is the status code returned from the server to your browser?
5. When was the HTML file that you are retrieving last modified at the server?
6. How many bytes of content are being returned to your browser?
7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one.

**NOTE:** In your answer to question 5 above, you might find that the document you just retrieved was last modified within a minute before you downloaded the document. This is because the www.cas.mcmaster.ca server sets the file's last-modified time to be the current time once per minute. Thus, if you wait a minute between accesses, the file will appear to have been recently modified, and hence your browser will download a "new" copy of the document.

## 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**. (To do this under Firefox, select *Tools->Clear Recent History* and check the Cache box, or for Internet Explorer, select *Tools->Internet Options->Delete File*; these actions will remove cached files from your browser's cache.) Now 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-file2.html>

Your browser should display a very simple five-line HTML file.

- Quickly enter the same URL into your browser again (or simply select 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:

8. How many HTTP GET requests were sent?
9. How many HTTP response messages were received?
10. If multiple response messages were received, what are the difference between them?

### 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, followed by a multiple-packet TCP response to your HTTP GET request. This multiple-packet response deserves a bit of explanation. Recall from Section 2.2 (see Figure 2.9 in the text) that the HTTP response message consists of a status line, followed by header lines, followed by a blank line, followed by the entity body. In the case of our HTTP GET, the entity body in the response is the *entire* requested HTML file. In our case here, the HTML file is rather long, and at 4500 bytes is too large to fit in one TCP packet. The single HTTP response message is thus broken into several pieces by TCP, with each piece being contained within a separate TCP segment (see Figure 1.24 in the text). In recent versions of Wireshark, Wireshark indicates each TCP segment as a separate packet, and the fact that the single HTTP response was fragmented across multiple TCP packets is indicated by the “TCP segment of a reassembled PDU” in the Info column of the Wireshark display. Earlier versions of Wireshark used the “Continuation” phrase to



indicated that the entire content of an HTTP message was broken across multiple TCP segments.. We stress here that there is no “Continuation” message in HTTP!

Answer the following questions:

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

## 4. HTML Documents with Embedded Objects

Now that we’ve seen how Wireshark displays the captured packet traffic for large HTML files, we can look at what happens when your browser downloads a file with embedded objects, i.e., a file that includes other objects (in the example below, image files) that are stored on another server(s).

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-file4.html>

- Your browser should display a short HTML file with two images. These two images are referenced in the base HTML file. That is, the images themselves are not contained in the HTML; instead the URLs for the images are contained in the downloaded HTML file. As discussed in the textbook, your browser will have to retrieve these logos from the indicated web sites. Our publisher’s logo is retrieved from the [www.aw-bc.com](http://www.aw-bc.com) web site. The image of the cover for our 5<sup>th</sup> edition (one of our favorite covers) is stored at the [manic.cs.umass.edu](http://manic.cs.umass.edu) server.
- Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed.

Answer the following questions:

15. How many HTTP GET request messages did your browser send? To which Internet addresses were these GET requests sent?
16. Can you tell whether your browser downloaded the two images serially, or whether they were downloaded from the two web sites in parallel? Explain.

## 5 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:

17. What is the server's response (status code and phrase) in response to the initial HTTP GET message from your browser?
18. 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.