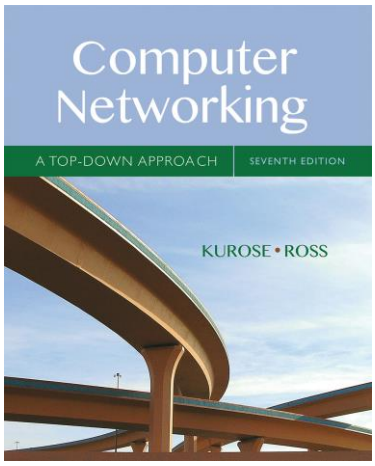
Wireshark Lab: HTTP



This lab is an abbreviated version of:
- Wireshark Getting Started v7.0
- Wireshark HTTP v7.0

Supplements to *Computer Networking: A Top-Down Approach, 7th ed.*, J.F. Kurose and K.W. Ross

"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". In the Wireshark labs, you'll observe the network protocols in your computer "in action".

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. The packet sniffer receives a copy of every link-layer frame that is sent from or received by your computer. Recall 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. Capturing all link-layer frames thus gives you all messages sent/received from/by all protocols and applications executing in your computer.

We will be using the Wireshark packet sniffer [https://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. Wireshark is a free network protocol analyzer that runs on Windows, Mac, and Linux/Unix computer.

Getting Wireshark

Download and install the Wireshark software:

- Go to https://www.wireshark.org/download.html and download and install Wireshark for your computer.
- You'll also need a packet capture library. It will be installed for you, when you install Wireshark. Make sure to keep this option selected during installation!

Running Wireshark

When you run the Wireshark program, you'll get a startup screen that looks something like the screen below. Different versions of Wireshark will have different startup screens – so don't panic if yours doesn't look exactly like the screen below!

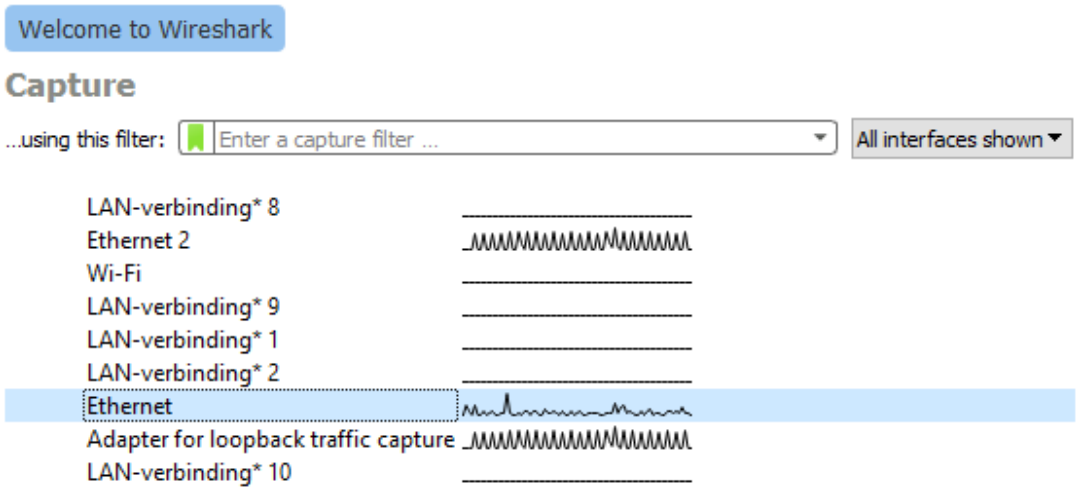


Figure 2: Initial Wireshark Screen

Note that under the Capture section, there is a list of so-called interfaces. The computer we're taking these screenshots from has just two real interfaces – Ethernet and Wi-Fi. All packets to/from this example computer will pass through the Ethernet interface, so it's here where we want to capture packets. Choose the interface through which you are getting Internet connectivity, e.g., mostly likely a Wi-Fi or Ethernet interface, and select that interface.

Let's take Wireshark out for a spin! 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.

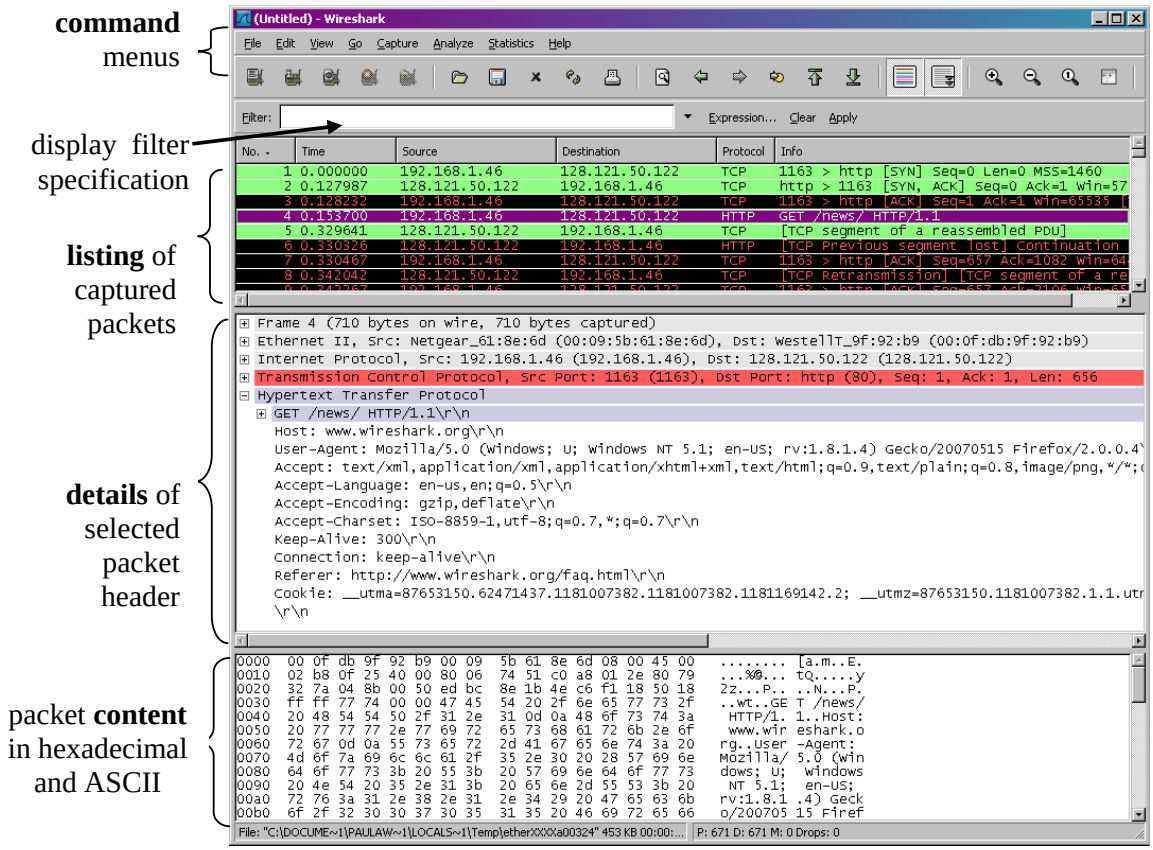


Figure 3: Wireshark Graphical User Interface, during packet capture and analysis

This looks more interesting! The Wireshark interface has five major components:

- The **command menus** are standard pulldown menus located at the top of the window. The Capture menu allows you to begin packet capture.
- The **packet-listing window** displays a one-line summary for each packet captured, including the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information.
- The **packet-header details window** provides details about the packet selected (highlighted) in the packet-listing window. These details include information about the Ethernet 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. 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 can be entered to filter the information displayed in the packet-listing window In the example below, we'll use the display filter to display only packets hat correspond to HTTP messages.

We're now ready to use Wireshark to investigate protocols in operation. In this lab, we'll explore several aspects of the HTTP protocol: the basic GET/response interaction, HTTP message formats and retrieving HTML files with embedded object.

1. The Basic HTTP GET/response interaction

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 (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://gaia.cs.umass.edu/wireshark-labs/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.

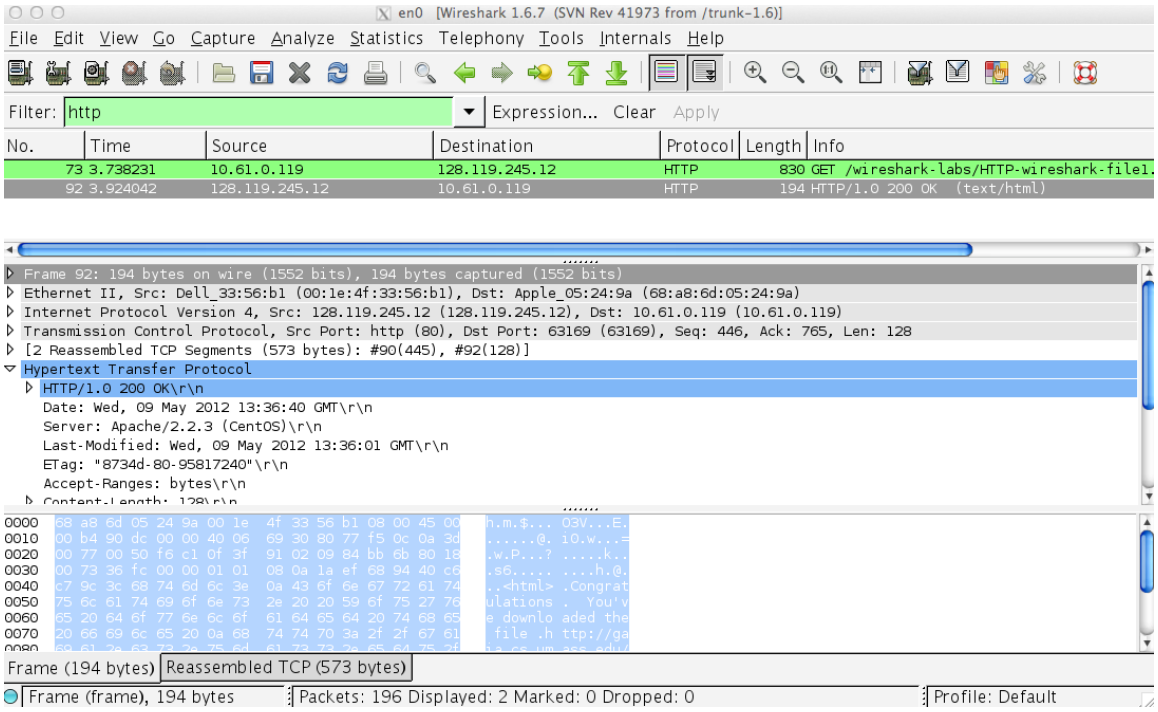


Figure 1: Wireshark Display after http://gaia.cs.umass.edu/wireshark-labs/ HTTP-wireshark-file1.html has been retrieved by your browser

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 gaia.cs.umass.edu 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.

1. Is your browser running HTTP version 1.0 or 1.1, or newer? 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 gaia.cs.umass.edu 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?

In your answer to question 5 above, you might have been surprised to find that the document you just retrieved was last modified within a minute before you downloaded the document. That's because (for this particular file), the gaia.cs.umass.edu server is setting the file's last-modified time to be the current time, and is doing so 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 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 (See: How to clear your cache on any browser). 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://gaia.cs.umass.edu/wireshark-labs/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:

6. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE" line in the HTTP GET?
7. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?
8. Now inspect the contents of the second HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE:" line in the HTTP GET? If so, what information follows the "IF-MODIFIED-SINCE:" header?
9. What is the HTTP status code and phrase returned from the server in response to this second HTTP GET? Did the server explicitly return the contents of the file? Explain.

3. HTML Documents with Embedded Objects

Now 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://gaia.cs.umass.edu/wireshark-labs/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 gaia.cs.umass.edu web site. The image of the cover for our 5th edition (one of our favorite covers) is stored at the caite.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:

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

4. Non-existing pages

Finally, we will see what happens when your browser requests a file that does not exist on the server.

Do the following:

- Start up your web browser
- Start up the Wireshark packet sniffer
- Enter the following URL into your browser
  http://gaia.cs.umass.edu/wireshark-labs/idontexist.html
  Your browser should display an error message.
- 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 question:

12. What is the status code returned from the server to your browser?