Print Your Name: _____

CSC 360 Lab 1

Introduction to Wireshark

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

Preparations:

1. In this lab you will use a dedicated Kali VM. We will remotely access the VM from Firefox.
2. To access VM on or off campus, follow the instructions at https://docs.hpc.tcnj.edu/index.php/CSC360-Computer-Networking-with-Li .
3. Both username and password are **csc360**.
4. Open a terminal in the remote computer, also click the Firefox icon and Wireshark icon on the left panel of the window.

End of preparation.

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 these Wireshark labs, we'll take the latter approach. You'll be running various network applications in different scenarios using a computer on your desk, at home, or in a lab. 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.

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 consistsof 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 1.5.2 in the textbook (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.

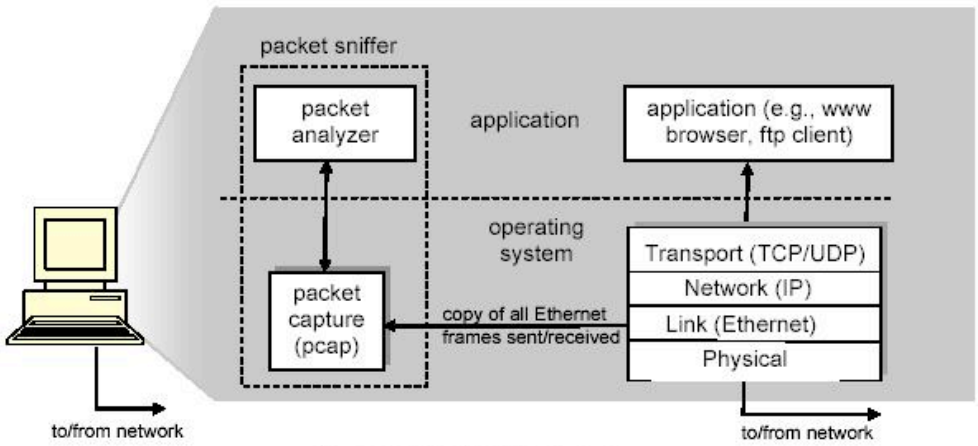


Figure 1: Packet sniffer structure

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 the textbook.

We will be using the Wireshark packet sniffer [the most recent version is Wireshark 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/), rich functionality that includes the capability to analyze more than 500 protocols, and a well-designed user interface. It operates in computers using Ethernet to connect to the Internet, as well as so-called point-to-point protocols such as PPP.

Getting Wireshark (for your home computer, the lab computers have already had Wireshark installed)

You may acquire a binary installer of Wireshark named something like: wireshark-setup-x.y.z.exe. The Wireshark installer includes WinPcap, so you don't need to download and install two separate packages.

Simply download the Wireshark installer from: http://www.wireshark.org/download.html and execute it. Beside the usual installer options like where to install the program, there are several optional components.

Running Wireshark

When you run the Wireshark program, the Wireshark graphical user interface shown in Figure 2 will de displayed. Initially, no data will be displayed in the various windows.

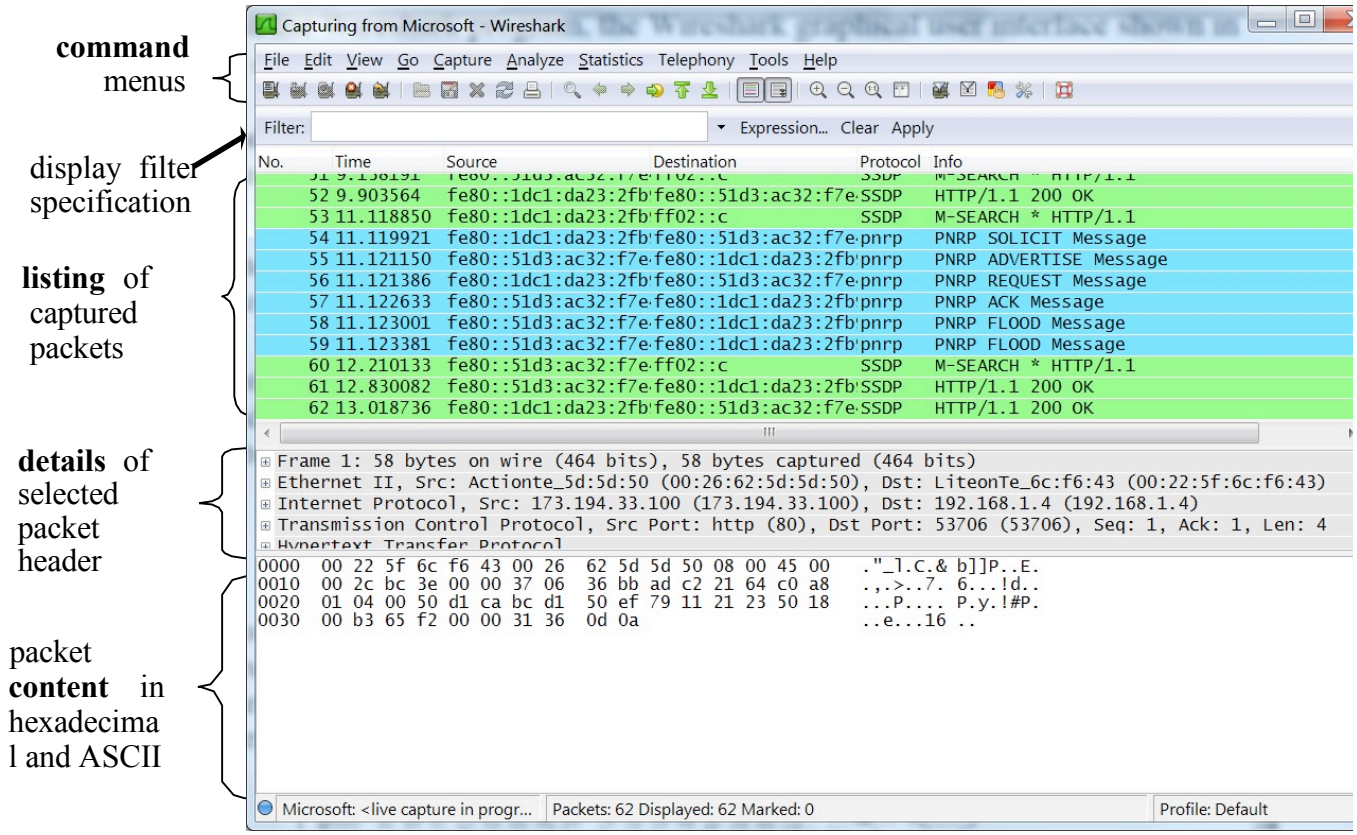


Figure 2: Wireshark Graphical User Interface

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 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 right-pointing or down-pointing arrowhead 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.

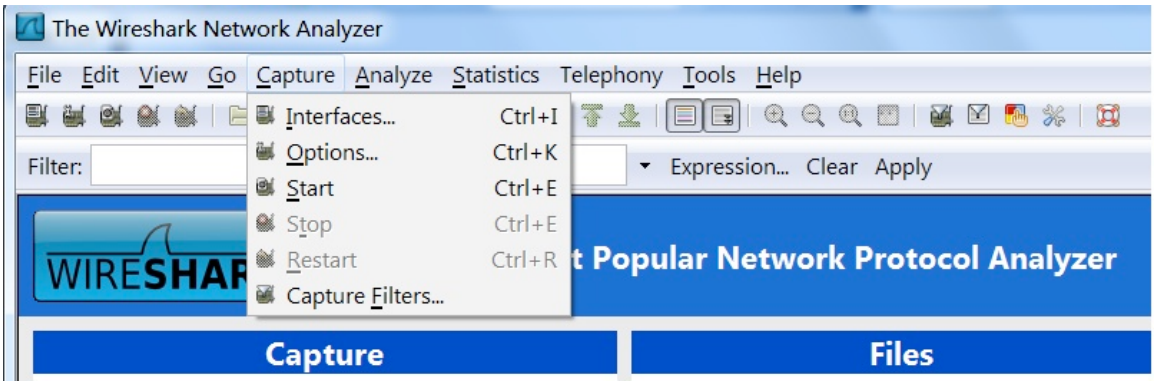
Taking Wireshark for a Test Run

The best way to learn about any new piece of software is to try it out! Do the following

1. Start up your favorite web browser, which will display your selected homepage.

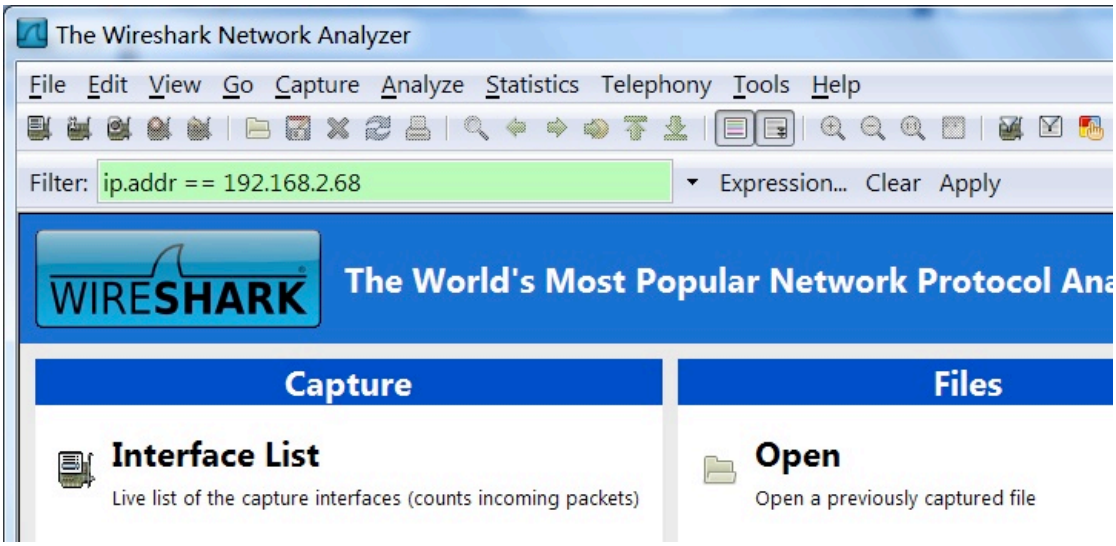
2. Start up the Wireshark software. You will initially see a window similar to that shown in Figure 2, except that no packet data will be displayed in the packet- listing, packet-header, or packet-contents window, since Wireshark has not yet begun capturing packets.

3. Select the **Capture** pull down menu and select *Interfaces.* You will see a pop up, which lists the **IP address** of your computer.



Write down the IP address displayed in the popup. You may see several IP address. Select the one used by your Ethernet card. If your computer IP address is 192.168.2.68 (you are likely to have a different one), type the following content into the **filter**.
IP.addr = = 192.168.2.68, then type carriage return.

This filter will make software display packets from or to this IP address. If you do not type in such filter information, Wireshark will show you a lot of broadcasting packets, which can be quite annoying.



4. To begin packet capture, select the Capture pull down menu and select *Options*. This will cause the "Wireshark: Capture Options" window to be displayed, as shown in Figure

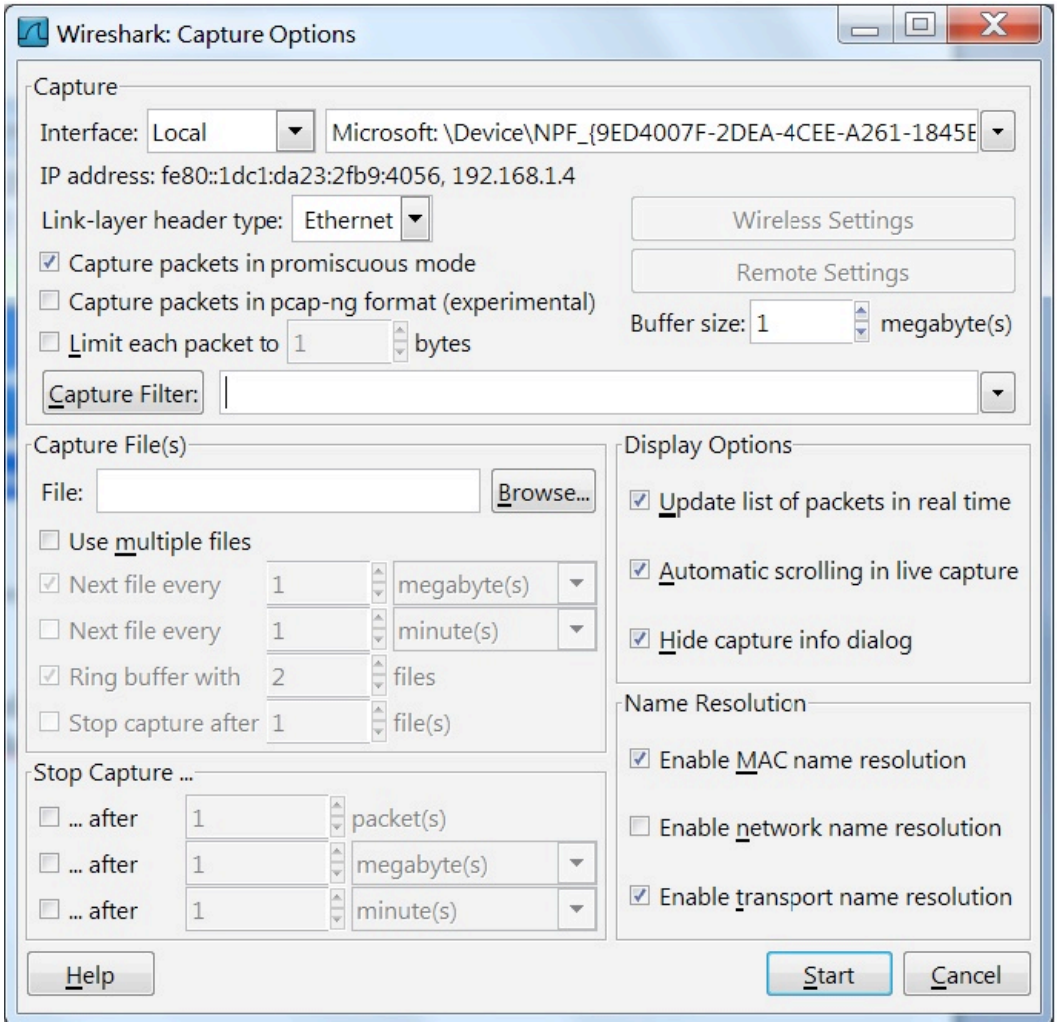


Figure 3: Wireshark Capture Options Window

5. Except network **interface**, you can use all of the default values in this window. The network interfaces (i.e., the physical connections) that your computer has to the network will be shown in the Interface pull down menu at the top of the Capture Options window. In case your computer has more than one active network interface (e.g., if you have both a wireless and a wired Ethernet connection), you will need to select the interface that is **being used to send and receive packets** (in our lab, it will be an Ethernet card). Please note that some computers in our lab have two Ethernet cards, **only one** card is active, you need to select the active one. After selecting the network interface (or using the default interface chosen by Wireshark), click Start. Packet capture will now begin - all packets being sent/received from/by your computer are now being captured by Wireshark!

6. Once you begin packet capture, the captured packets will be displayed in Wireshark in real time. Let Wireshark keep running.

7. While Wireshark is running, enter the URL: **http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html** and have that page displayed in your browser. In order to display this page, your browser will contact the HTTP server at gaia.cs.umass.edu and exchange HTTP messages with the server in order to download this page, as discussed in section 2.2 of the textbook. The Ethernet frames containing these HTTP messages will be captured by Wireshark.

8. After your browser has displayed the INTRO-wireshark-file1.html page, stop Wireshark packet capture by selecting stop in the Wireshark capture window. This will cause the Wireshark capture window to disappear and the main Wireshark window to display all packets captured since you began packet capture. The main Wireshark window should now look similar to Figure 2. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP message exchanges with the gaia.cs.umass.edu web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well (see, e.g., the many different protocol types shown in the *Protocol* column in Figure 2). Even though the only action you took was to download a web page, there were evidently many other protocols running on your computer that are unseen by the user. We'll learn much more about these protocols as we progress through the textbook! For now, you should just be aware that there is often much more going on than "meet's the eye"!

9. Inside the filter field, type in "&& http" **after** "ip.addr = = 192.168.2.68" (without the quotes, and in lower case – all protocol names are in lower case in Wireshark). Then select *Apply* (to the right of where you entered "http"). This will cause only HTTP message to be displayed in the packet-listing window.

10. Select the first http message shown in the packet-listing window. This should be the HTTP GET message that was sent from your computer to the gaia.cs.umass.edu HTTP server. When you select the HTTP GET message, the Ethernet frame, IP datagram, TCP segment, and HTTP message header information will be displayed in the packet-header window[2]. By clicking on right- pointing and down-pointing arrowsheads to the left side of the packet details window, *minimize* the amount of Frame, Ethernet, Internet Protocol, and Transmission Control Protocol information displayed. *Maximize* the amount information displayed about the HTTP protocol. Your Wireshark display should now look roughly as shown in Figure 5 (Note, in particular, the minimized amount of protocol information for all protocols except HTTP, and the maximized amount of protocol information for HTTP in the packet-header window).

[2] Recall that the HTTP GET message that is sent to the gaia.cs.umass.edu web server is contained within a TCP segment, which is contained (encapsulated) in an IP datagram, which is encapsulated in an Ethernet frame. If this process of encapsulation isn't quite clear yet, review section 1.5.2 in the textbook

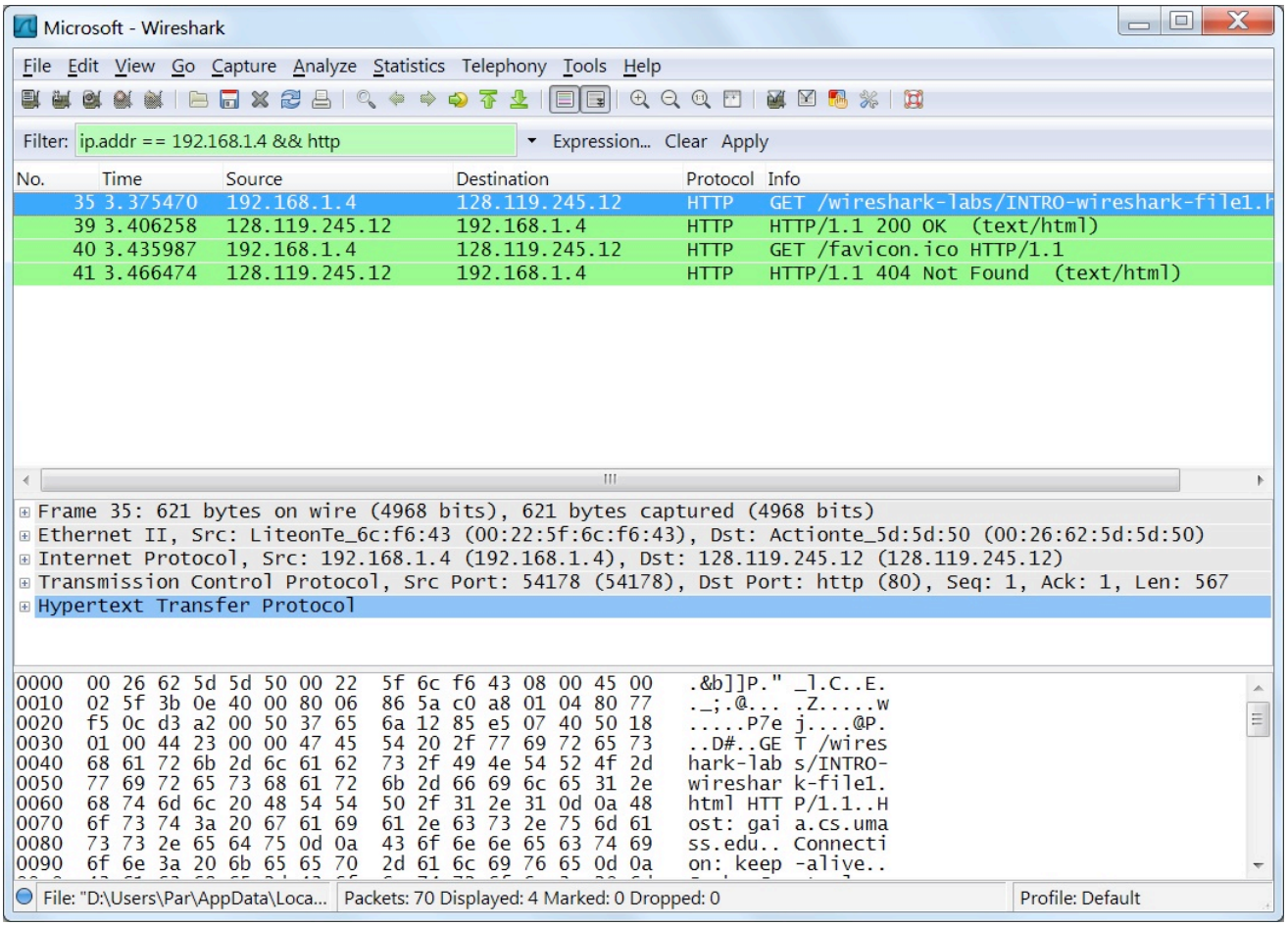


Figure 5: Wireshark display after step 9

Congratulations! You've now completed the first part of the lab.

What to **hand in**

The goal of this first lab was primarily to introduce you to Wireshark. The following questions will demonstrate that you've been able to get Wireshark up and running, and have explored some of its capabilities. Answer the following questions, based on your Wireshark experimentation:

1. List the different protocols that appear in the protocol column in the unfiltered packet-listing window in step 8 above.

2. How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packet-listing window is the amount of time, in seconds, since Wireshark tracing began. To display the Time field in time-of-day format, select the Wireshark *View* pull down

menu, then select Time *Display Format*, then select *Time-of-day*.)

3. What is the Internet address of the gaia.cs.umass.edu (also known as www-net.cs.umass.edu)? What is the Internet address of your computer?

HTTP

Having gotten our feet wet with the Wireshark packet sniffer in the introductory lab, 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, 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 textbook.

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, as described in the Introductory part (but don't yet begin packet capture). Enter "ip.addr = = 192.168.2.68 && 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, and then begin Wireshark packet capture.
4. Enter the following to your browser **http://gaia.cs.umass.edu/wireshark-labs/INTRO-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 6.

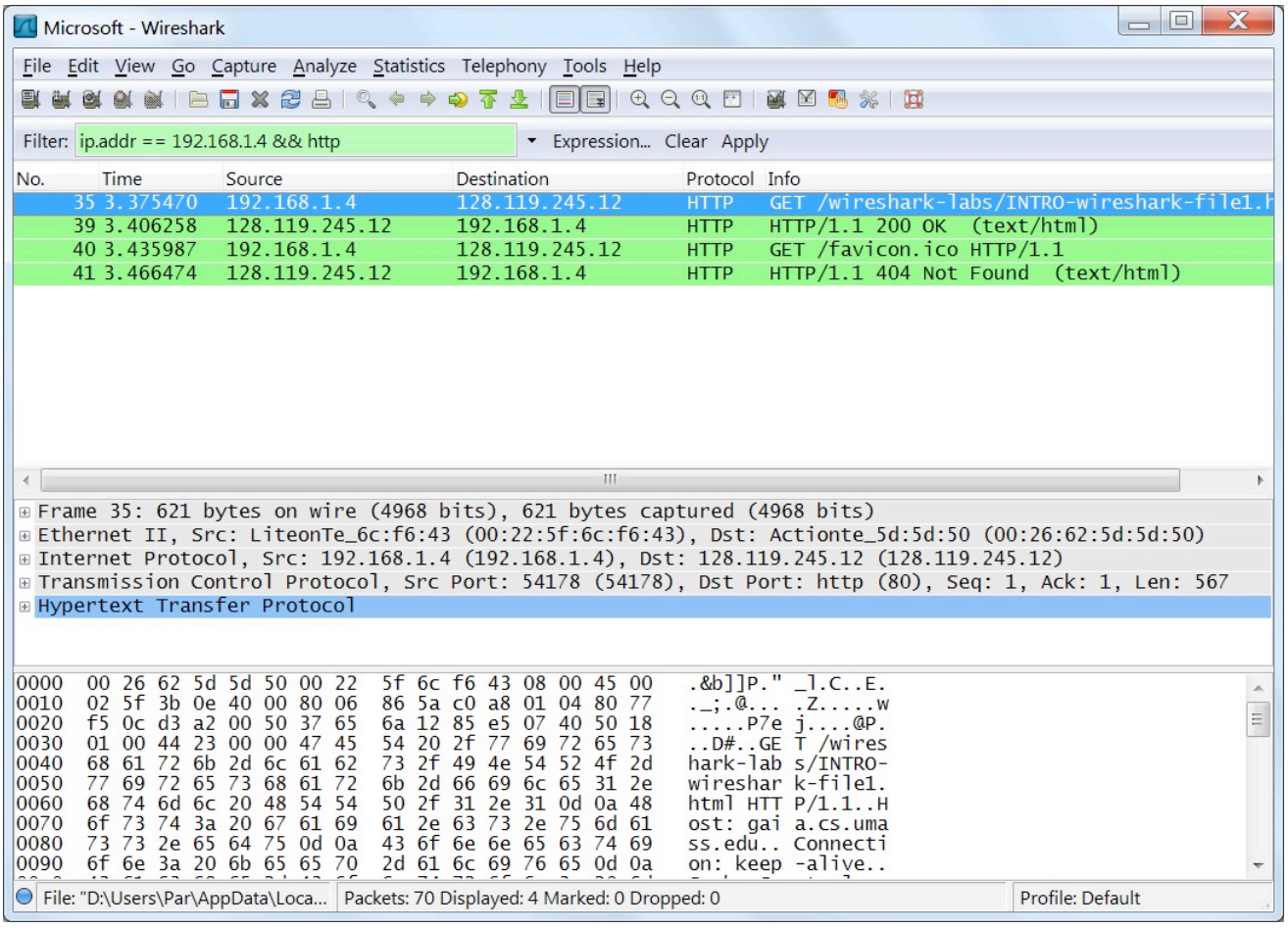


Figure 6: Wireshark Display after http://gaia.cs.umass.edu/wireshark-labs/ HTTP-wireshark-file1.html has been retrieved by your browser

The example in Figure 6 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 GET 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 right-pointing arrowhead (which means there is hidden, undisplayed information), and the HTTP line has a down-pointing arrowhead (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 print out the GET and response messages (see the introductory part for an explanation of how to do this) and 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 status code returned from the server to your browser?

4. How many bytes of content are being returned to your browser?

5. 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.

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 from Section 2.2.5 of the textbook, 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 Netscape 7.0, select *Edit->Preferences->Advanced->Cache* and clear the memory and disk cache.
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://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. 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://gaia.cs.umass.edu/wireshark-labs/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 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 textbook) 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 1.22 in the textbook). Each TCP segment is recorded as a separate packet by Wireshark, and the fact that the single HTTP response was fragmented across multiple TCP packets is indicated by the "Continuation" phrase displayed by Wireshark. We stress here that there is no "Continuation" message in HTTP!

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://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html** is password protected. The username is "wireshark-students" (without the quotes), and the password is "network" (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://gaia.cs.umass.edu/wireshark-labs/protected_pages/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. You might want to first read up on HTTP authentication by reviewing the easy-to-read material on "HTTP Access Authentication Framework" at **http://frontier.userland.com/stories/storyReader$2159**

Answer the following questions:

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

11. 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 (eth-students) and password (network) that you entered are encoded in the string of characters (ZXRoLXN0dWRlbnRzOm5ldHdvcmtz) 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 ZXRoLXN0dWRlbnRzOm5ldHdvcmtz and press decode. *Voila!* You have translated from Base64 encoding to ASCII encoding, and thus should see both your username and 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.

Fear not! As we will see in Chapter 7, there are ways to make WWW access more secure. However, we'll clearly need something that goes beyond the basic HTTP authentication framework!

Submission

1. I need your paper back when you leave.
2. If you can not finish it today, hand in the answer to me by next class.