

Project 3: HTTP Wireshark (100 Pts)

Due: 23:59 Friday, November 20, 2020

CSE422 Computer Networks Fall 2020

Introduction

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 friend's; 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.

1 Getting Started

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) 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.

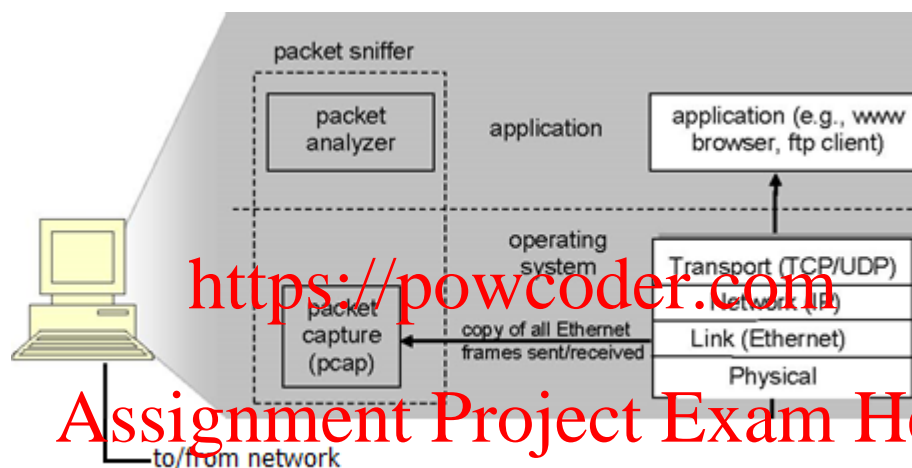


Figure 1 Packet Sniffer Structure

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](#), [man pages](#), a detailed [FAQ](#), rich functionality that includes the capability to analyze hundreds of protocols, and a well-designed user interface. It operates in computers using Ethernet, serial (PPP and SLIP), 802.11 wireless LANs, and many other link-layer technologies (if the OS on which it’s running allows Wireshark to do so).

1.1 Installing 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 [here](#) for a list of supported operating systems and download sites.

Download and install the Wireshark software:

- Go [here](#) and download and install the Wireshark binary for your computer.

Many linux distributions have wireshark directly available in their official package repositories (e.g. for Ubuntu systems it can be installed using `apt`).

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

1.2 Running Wireshark

When you run the Wireshark program, you'll get a startup screen, as shown below:

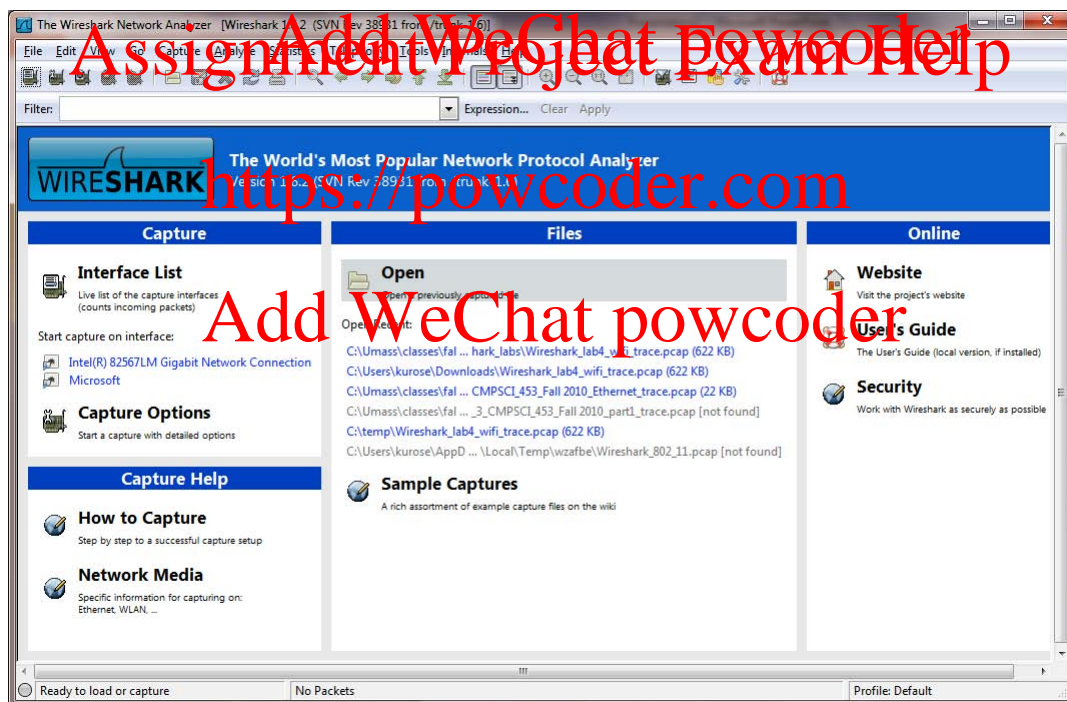


Figure 2: Initial Window Wireshark Screen

Take a look at the upper left hand side of the screen – you'll see an “Interface list”. This is the list of network interfaces on your computer. Once you choose an interface, Wireshark will capture all packets on that interface. In the example above, there is an Ethernet interface (Gigabit network Connection) and a wireless interface (“Microsoft”).

The startup screen when Wireshark is running on the MacOS looks slightly different. It immediately

displays a list of interfaces that you can select to start a capture on. Figure 3 shows the MacOS Wireshark startup screen.

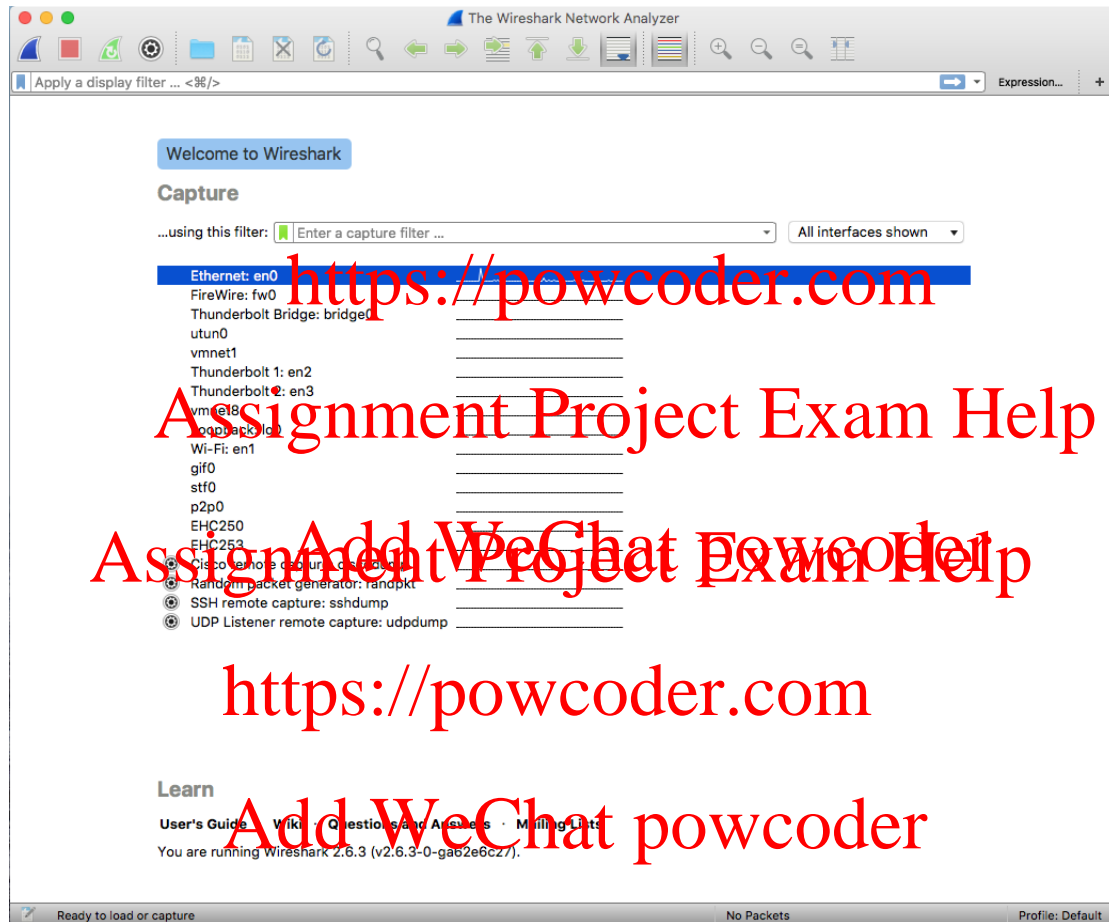


Figure 3: MacOS Wireshark Startup Screen

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.

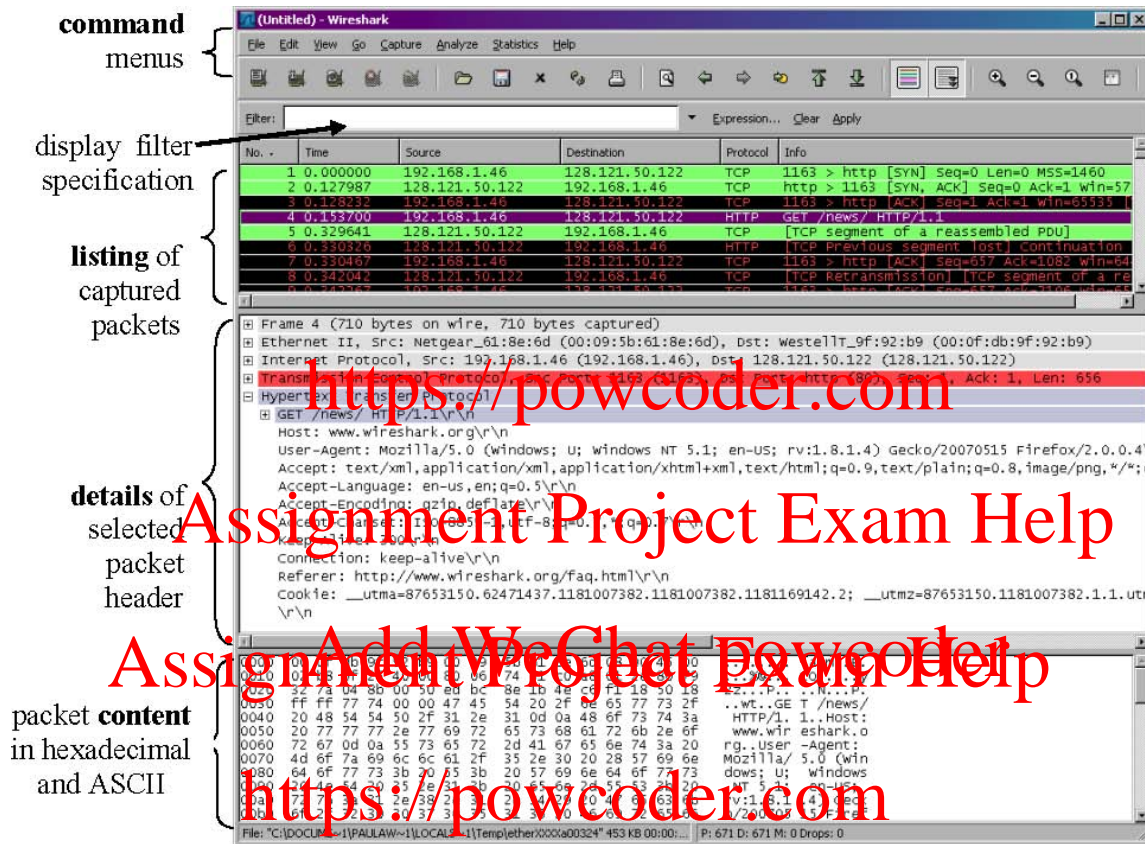


Figure 4: Wireshark Graphical User Interface, during packet capture and analysis

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.

1.3 Taking Wireshark for a Test Run

The best way to learn about any new piece of software is to try it out. I will assume that your computer is connected to the Internet via a wired Ethernet interface. Indeed, I recommend that you do this first lab on a computer that has a wired Ethernet connection, rather than just a wireless connection.

Do the following

- Start up your favorite web browser which will display your selected homepage.
- Start up the Wireshark software. You will initially see a window similar to that shown in Figure 2. Wireshark has not yet begun capturing packets.
- To begin packet capture, select the Capture pull down menu and select *Interfaces*. This will cause the “Wireshark: Capture Interfaces Window” to be displayed, as shown in Figure 5.

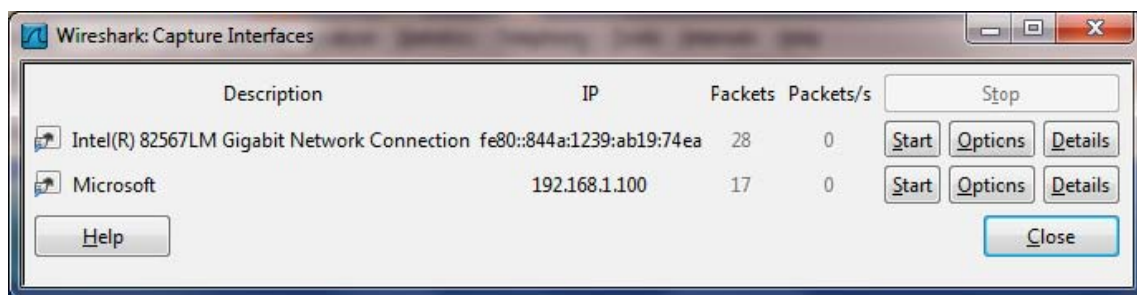


Figure 5: Wireshark Capture Interface Window

On MacOS simply select the interface from the startup screen to start a capture.

- You'll see a list of the interfaces on your computer as well as a count of the packets that have been observed on that interface so far. Click on *Start* for the interface on which you want to begin packet capture (in the case, the Gigabit network Connection). Packet capture will now begin -Wireshark is now capturing all packets being sent/received from/by your computer!
- Once you begin packet capture, a window similar to that shown in Figure 3 will appear. This window shows the packets being captured. By selecting *Capture* pulldown menu and selecting *Stop*, you can stop packet capture. But don't stop packet capture yet. Let's capture some interesting packets first. To do so, we'll need to generate some network traffic. Let's do so using a web browser, which will use the HTTP protocol that we will study in detail in class to download content from a website.
- While Wireshark is running, enter the URL:

<https://powcoder.com>

<http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html>

Assignment Project Exam Help

- After your browser has displayed the INTRO-wireshark-file1.html page (it is a simple one line of congratulations), stop Wireshark packet capture by selecting stop in the Wireshark capture window. The main Wireshark window should now look similar to Figure 3. 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 3). 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 text! For now, you should just be aware that there is often much more going on than "meet's the eye" 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 text. The Ethernet frames containing these HTTP messages (as well as all other frames passing through your Ethernet adapter) will be captured by Wireshark.
- Type in "http" (without the quotes, and in lower case – all protocol names are in lower case in Wireshark) into the display filter specification window at the top of the main Wireshark window. 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.
- Find the HTTP GET message that was sent from your computer to the gaia.cs.umass.edu HTTP server. (Look for an HTTP GET message in the "listing of captured packets" portion of the Wireshark window (see Figure 3) that shows "GET" followed by the gaia.cs.umass.edu URL that you entered. 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. By clicking on '+' and '-' right-pointing and down-pointing arrow-heads 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).

- Exit Wireshark.

Congratulations! You've now completed the first lab.

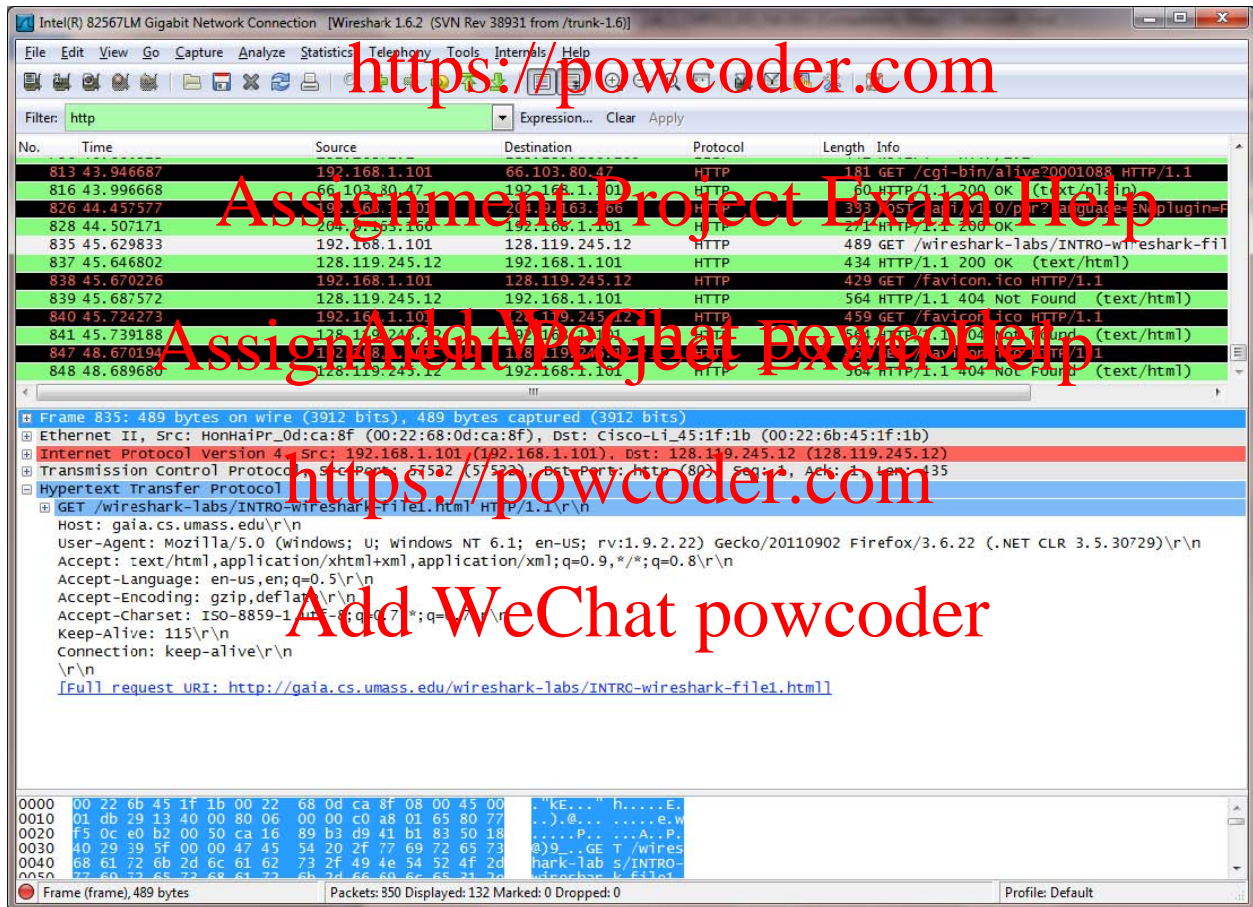


Figure 6: Windows Wireshark window after browser connects to webpage.

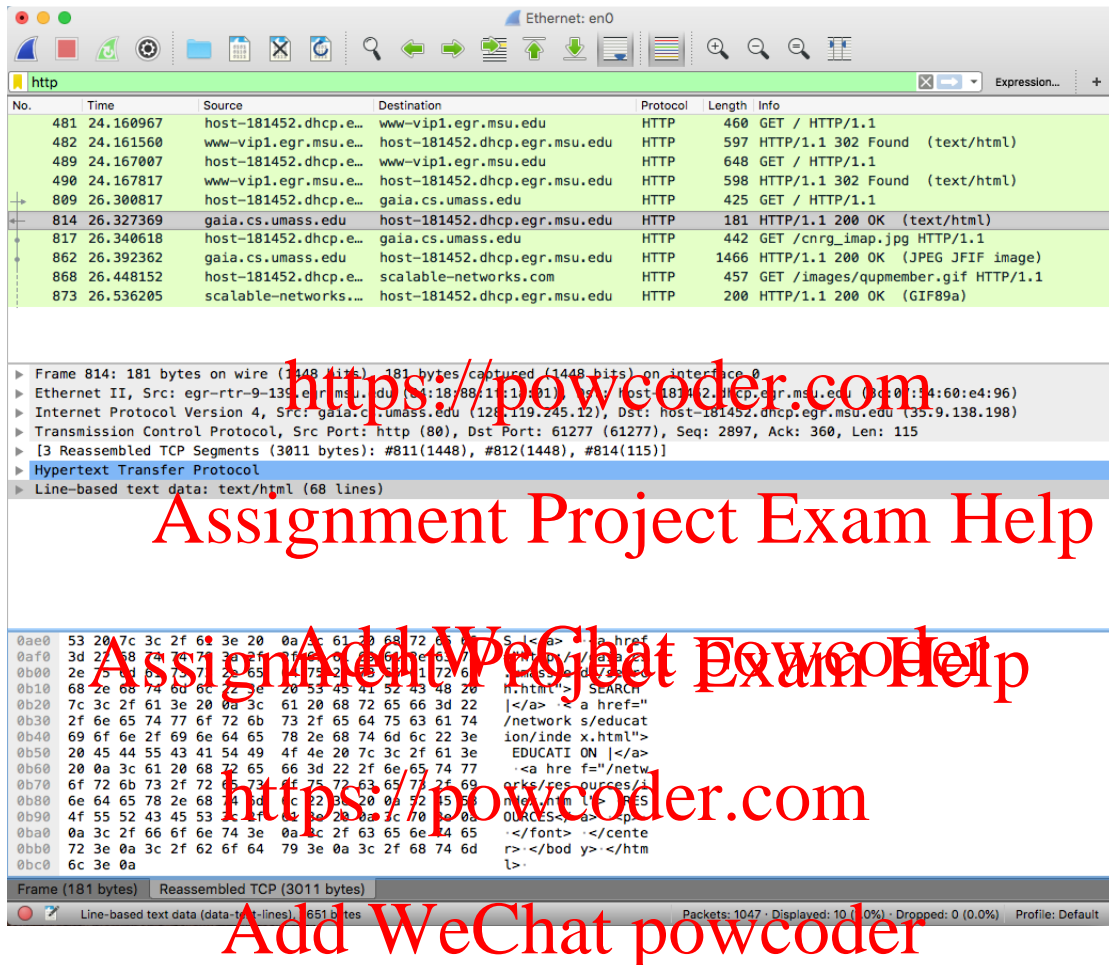


Figure 7: MacOS Capture Screen

1.4 What to hand in for this section

The goal of this first part 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 3 different protocols that appear in the protocol column in the unfiltered packet-listing window in step 7 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 wwwnet.cs.umass.edu)? What is the Internet address of your computer?

4. Print the two HTTP messages (GET and OK) referred to in question 2 above. To do so, select *Print* from the Wireshark *File* command menu, and select the “*Selected Packet Only*” and “*Print as displayed*” radial buttons, and then click OK.

2 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 part, 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 this part, you might want to review Section 2.2 of the text.

2.1 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:

- Start up your web browser.
- Start up the Wireshark packet sniffer, as described in the previous section (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).
- Wait a bit more than one minute (we’ll see why shortly), and then begin Wireshark packet capture.
- 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.
- Stop Wireshark packet capture.

Your Wireshark window should look similar to the window shown in Figure 1.

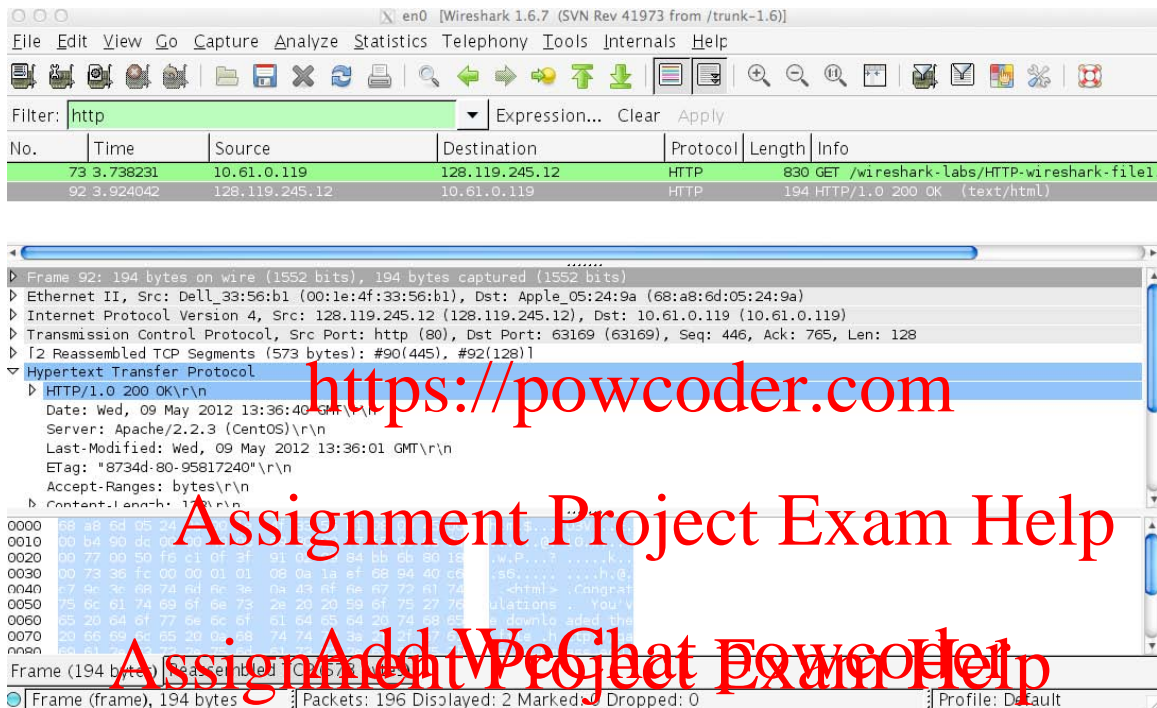


Figure 8: Wireshark Display after <http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file1.html> has been retrieved by your browser.

The example in Figure 8 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 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 print out the GET and response messages (see the introductory Wireshark lab for an explanation of how to do this) and indicate where in the message you've found the information that answers the following questions. When

you hand in your assignment, annotate the output so that it's clear where in the output you're getting the information for your answer.

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 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?
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.

In your answer to question 7, do you perhaps 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.

2.2 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; 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.
- (Note: If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-2 packet trace to answer the questions below; see footnote 1. This trace file was gathered while performing the steps above on one of the author’s computers.)

Answer the following questions:

8. 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?
9. Inspect the contents of the server response. Did the server explicitly return the contents of the file? How can you tell?
10. 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?
11. 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.

2.3 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 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 indicate 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:

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

2.4 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://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 www.aw-bc.com web site. The image of the cover for our 5th edition (one of our favorite covers) is stored at the 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:

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

<https://powcoder.com>

2.5 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://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.

<https://powcoder.com>

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

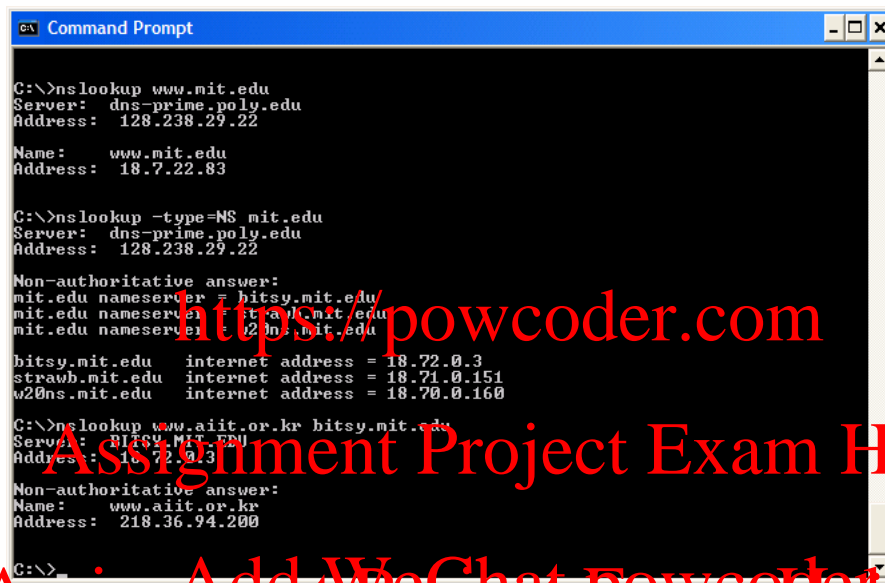
- Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window.

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” [here](#).

Answer the following questions:

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

specified DNS server, receives a DNS reply from that same DNS server, and displays the result.



```
C:\>nslookup www.mit.edu
Server: dns-prime.poly.edu
Address: 128.238.29.22

Name: www.mit.edu
Address: 18.7.22.83

C:\>nslookup -type=NS mit.edu
Server: dns-prime.poly.edu
Address: 128.238.29.22

Non-authoritative answer:
mit.edu nameserver = bitsy.mit.edu
mit.edu nameserver = strawb.mit.edu
mit.edu nameserver = w20ns.mit.edu

bitsy.mit.edu internet address = 18.72.0.3
strawb.mit.edu internet address = 18.71.0.151
w20ns.mit.edu internet address = 18.70.0.160

C:\>nslookup www.aiit.or.kr bitsy.mit.edu
Server: bitsy.mit.edu
Address: 18.72.0.3

Non-authoritative answer:
Name: www.aiit.or.kr
Address: 218.36.94.200

C:\>
```

Figure 9: The above screenshot shows the results of three independent `nslookup` commands (displayed in the Windows Command Prompt).

In this example, the client host is located on the campus of Polytechnic University in Brooklyn, where the default local DNS server is `dns-prime.poly.edu`. When running `nslookup`, if no DNS server is specified, then `nslookup` sends the query to the default DNS server, which in this case is `dns-prime.poly.edu`. Consider the first command:

```
nslookup www.mit.edu
```

In words, this command is saying “please send me the IP address for the host `www.mit.edu`”. As shown in the screenshot, the response from this command provides two pieces of information: (1) the name and IP address of the DNS server that provides the answer; and (2) the answer itself, which is the host name and IP address of `www.mit.edu`. Although the response came from the local DNS server at Polytechnic University, it is quite possible that this local DNS server iteratively contacted several other DNS servers to get the answer, as described in Section 2.4 of the textbook. Now consider the second command:

```
nslookup --type=NS mit.edu
```

In this example, we have provided the option “`-type=NS`” and the domain “`mit.edu`”. This causes `nslookup` to send a query for a type-NS record to the default local DNS server. In words, the query is saying, “please send me the host names of the authoritative DNS for `mit.edu`”. (When the `-type` option is not used, `nslookup` uses the default, which is to query for type A records.) The answer, displayed in the above screenshot, first indicates the DNS server that is providing the answer (which

is the default local DNS server) along with three MIT nameservers. Each of these servers is indeed an authoritative DNS server for the hosts on the MIT campus. However, **nslookup** also indicates that the answer is “non-authoritative,” meaning that this answer came from the cache of some server rather than from an authoritative MIT DNS server. Finally, the answer also includes the IP addresses of the authoritative DNS servers at MIT. (Even though the type-NS query generated by **nslookup** did not explicitly ask for the IP addresses, the local DNS server returned these “for free” and **nslookup** displays the result.)

Now finally consider the third command:

```
nslookup www.aiit.or.kr bitsy.mit.edu
```

In this example, we indicate that the query sent to the DNS server bitsy.mit.edu rather than to the default DNS server (dns-prime.poly.edu). Thus, the query and reply transaction takes place directly between our querying host and bitsy.mit.edu. In this example, the DNS server bitsy.mit.edu provides the IP address of the host www.aiit.or.kr, which is a web server at the Advanced Institute of Information Technology (in Korea).

Now that we have gone through a few illustrative examples, you are perhaps wondering about the general syntax of **nslookup** commands. The syntax is

```
nslookup --option1 --option2 host-to-find dns-server
```

In general, **nslookup** can be run with zero, one, two or more options. And as we have seen in the above examples, the dns-server is optional as well; if it is not supplied, the query is sent to the default local DNS server.

Now that we have provided an overview of **nslookup**, it is time for you to test drive it yourself. Do the following (and write down the results):

1. Run **nslookup** to obtain the IP address of a Web server in Asia. What is the hostname and IP address of that server?
2. Run **nslookup** to determine the authoritative DNS servers for a university in Europe. What University did you query for and what was the domain name you used? List two of the hostnames and IP addresses of the servers indicated in the response.
3. Run **nslookup** so that one of the DNS servers obtained in Question 2 is queried for the mail servers for Yahoo! mail. What kind of response do you get?

3.2 ipconfig/ifconfig

ipconfig (for Windows) and **ifconfig** (for Linux/Unix/MacOS) are among the most useful little utilities in your host, especially for debugging network issues. Here we'll only describe **ipconfig**, although the Linux/Unix/MacOS **ifconfig** is very similar. **ipconfig** can be used to show your current TCP/IP information, including your address, DNS server addresses, adapter type and so on. For example, if you want all this information about your host simply by entering


```
ipconfig /all
```

into the Command Prompt, as shown in the following screenshot (or `ifconfig -a`).

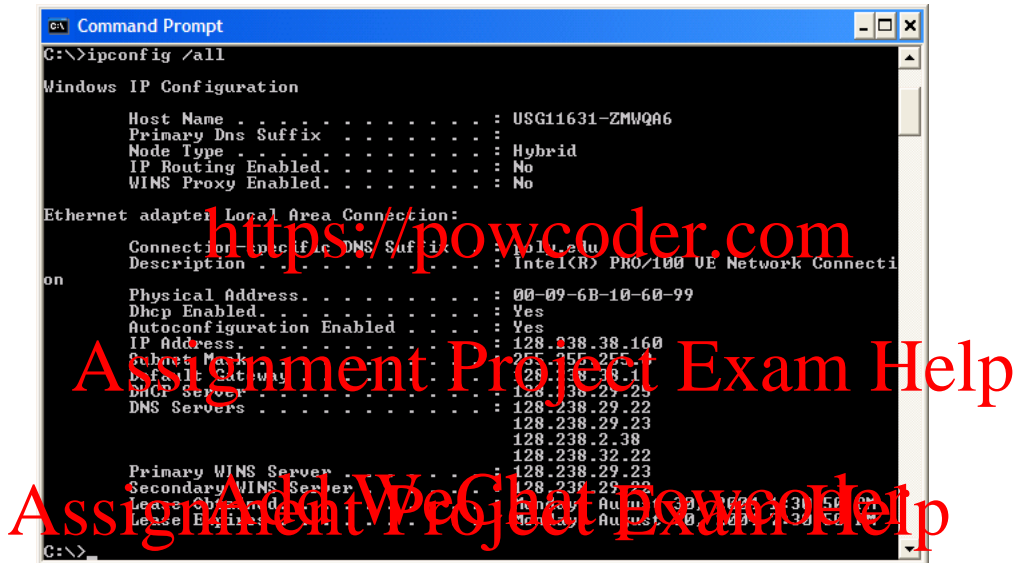


Figure 10: Screenshot of `ipconfig /all`

`ipconfig` is also very useful for managing the DNS information stored in your host. In Section 2.5 we learned that a host can cache DNS records it recently obtained. To see these cached records, provide the following command:

```
ipconfig /displaydns
```

Displaying the cache entries on the latest version of the MacOS is more difficult. The following steps should allow you to display the dns cache entries.

- Open two command terminals.
- In window 1 type the following command line:

```
log stream --predicate process == 'mDNSResponder' --info
```
- In window 2 type the following command line:

```
sudo killall -INFO mDNSResponder
```

The DNS cache entries should be displayed in the first command window. Each entry shows the remaining Time to Live (TTL) in seconds. To clear the cache, enter

```
ipconfig /flushdns
```

on MacOS use

```
sudo dscacheutil -flushcache
```

on Linux use

```
sudo /etc/init.d/nscd restart
```

Flushing the DNS cache clears all entries and reloads the entries from the hosts file.

3.3 Tracing DNS with Wireshark

Now that we are familiar with `nslookup` and `ipconfig`, we're ready to get down to some serious business. Let's first capture the DNS packets that are generated by ordinary Web-surfing activity.

- Use `ipconfig` or `dscacheutil` or `/etc/init.d/nscd` to empty the DNS cache in your host.
- Open your browser and empty your browser cache.
- Open Wireshark and enter "`ip.addr == your_IP_address`" into the filter, where you obtain your IP address with `ipconfig/ifconfig`. This filter removes all packets that neither originate nor are destined to your host.
- Start packet capture in Wireshark. With your browser, visit the Web page: <http://www.ietf.org>
- Stop packet capture.

Answer the following questions. Whenever possible, when answering a question below, you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. Annotate the printout to explain your answer. To print a packet, use *File-Print*, choose *Selected packet only*, choose *Packet summary line*, and select the minimum amount of packet detail that you need to answer the question.

4. Locate the DNS query and response messages. Are they sent over UDP or TCP?
5. What is the destination port for the DNS query message? What is the source port of DNS response message?
6. To what IP address is the DNS query message sent? Use `ipconfig` to determine the IP address of your local DNS server. Are these two IP addresses the same?
7. Examine the DNS query message. What "Type" of DNS query is it? Does the query message contain any "answers"?

8. Examine the DNS response message. How many “answers” are provided? What do each of these answers contain?
9. Consider the subsequent TCP SYN packet sent by your host. Does the destination IP address of the SYN packet correspond to any of the IP addresses provided in the DNS response message?
10. This web page contains images. Before retrieving each image, does your host issue new DNS queries?

Now lets play with nslookup.

<https://powcoder.com>

- Start packet capture.
- Do a nslookup on www.mit.edu
- Stop packet capture.

Assignment Project Exam Help

You should get a trace that looks something like the following:

Assignment Project Exam Help

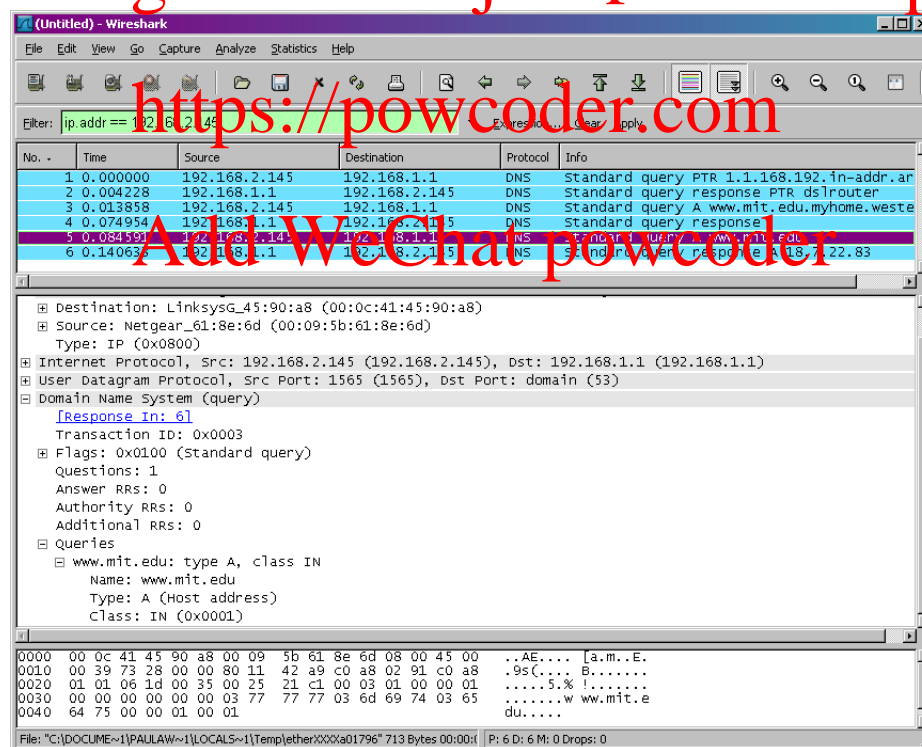


Figure 11: Trace of nslookup

We see from the above screenshot that nslookup actually sent three DNS queries and received three DNS responses. For the purpose of this assignment, in answering the following questions,

ignore the first two sets of queries/responses, as they are specific to `nslookup` and are not normally generated by standard Internet applications. You should instead focus on the last query and response messages.

11. What is the destination port for the DNS query message? What is the source port of DNS response message?
12. To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server?
13. Examine the DNS query message. What “Type” of DNS query is it? Does the query message contain any “answers”?
14. Examine the DNS response message. How many “answers” are provided? What do each of these answers contain?
15. Provide a screenshot.

Now repeat the previous experiment, but instead issue the command:

```
nslookup --type=NS mit.edu
```

Answer the following questions:

16. To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server?
17. Examine the DNS query message. What “Type” of DNS query is it? Does the query message contain any “answers”?
18. Examine the DNS response message. What MIT nameservers does the response message provide? Does this response message also provide the IP addresses of the MIT nameservers?
19. Provide a screenshot.

4 Grading

You should turn in your answers with annotated printouts to the indicated questions to the appropriate crowdmark assignment. Each question within each section is worth the same number of points. The summary for how the points / questions are distributed between the sections is described in the following table.

Section	Num Questions	Points
1 – Getting Started	4	10
2 – HTTP	19	45
3 – DNS	19	45

5 Office Hours

You can contact TA Jonathon Fleck at fleckjo1@msu.edu if you have any questions about the project or need to setup additional office hours. The planned office hours for this project are as follows (all times are in EST).

Week	Tuesday	Wednesday	Friday
11/9 - 11/13	2-3	2-3	NA
11/16 - 11/20	2-3	2-3	2-3

<https://powcoder.com>

Assignment Project Exam Help

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder