



# THE UNIVERSITY OF WINNIPEG

## APPLIED COMPUTER SCIENCE

### ACS-3911-050 Computer Networks Winter 2020

#### Assignment 2/Lab HTTP and Lab DNS Due Date: 26<sup>th</sup> February, 2020

The assignment is in two parts, Part A - Questions and Answer, and Part B - Wireshark Lab HTTP and DNS. Part A is 25 marks, and Part B is 45 marks, a total of 70 marks for this assignment. Please complete by due date. No assignment will be accepted after the due date and the date/time stamp on the email will be used to determine if the assignment was on time or late.

Email your assignment in PDF format to 3911-050@acs.uwinnipeg.ca by 11:59pm Feb 26th, 2020. The name of your file should be named - <First name Last Name> - <Student ID> – Assignment <number>, e.g. John Doe – 12345 – Assignment 2. The subject line of your email must be: <your name><your student #>< course # with section #>Assignment 1.

## Part A – Questions and Answers

### Review Questions

#### Section 2.1

R5. What information is used by a process running on one host to identify a process running on another host? (2 marks)

#### Section 2.2 – 2.5

R11. Why do HTTP, FTP, SMTP, and POP3 run on top of TCP rather than on UDP? (1 mark)

R12. Consider an e-commerce site that wants to keep a purchase record for each of its customers. Describe how this can be done with cookies. (1 mark)

R13. Describe how Web caching can reduce the delay in receiving a requested object. Will Web caching reduce the delay for all objects requested by a user or for only some of the objects? Why? (1 mark)

R19. Is it possible for an organization's Web server and mail server to have exactly the same alias for a hostname (for example, foo.com)? What would be the type for the RR that contains the hostname of the mail server? (1 mark)

#### Section 2.5

R21. In BitTorrent, suppose Alice provides chunks to Bob throughout a 30-second interval. Will Bob necessarily return the favor and provide chunks to Alice in this same interval? Why or why not? (1 mark)

#### Section 2.6

R24. CDN typically adopt one of two different server placement philosophies. Name and briefly describe them. (1 mark)

#### Section 2.7

R27. For the client-server application over TCP described in Section 2.7, why must the server program be executed before the client program? For the client-server application over UDP, why may the client program be executed before the server program? (1 mark)

### Problems

P1. True or false? (5 marks)

- a. A user requests a Web page that consists of some text and three images. For this page, the client will send one request message and receive four response messages.
- b. Two distinct Web pages (for example, [www.mit.edu/research.html](http://www.mit.edu/research.html) and [www.mit.edu/students.html](http://www.mit.edu/students.html)) can be sent over the same persistent connection.
- c. With non-persistent connections between browser and origin server, it is possible for a single TCP segment to carry two distinct HTTP request messages.
- d. The Date: header in the HTTP response message indicates when the object in the response was last modified.
- e. HTTP response messages never have an empty message body.

P6. Obtain the HTTP/1.1 specification (RFC 2616). Answer the following questions: (4 marks)

- a. Explain the mechanism used for signaling between the client and server to indicate that a persistent connection is being closed. Can the client, the server, or both signal the close of a connection?
- b. What encryption services are provided by HTTP?
- c. Can a client open three or more simultaneous connections with a given server?
- d. Either a server or a client may close a transport connection between them if either one detects the connection has been idle for some time. Is it possible that one side starts closing a connection while the other side is transmitting data via this connection? Explain.

P16. Read the POP3 RFC, RFC 1939. What is the purpose of the UIDL POP3 command? (1 mark)

P32. What is the Apache Web Server? How much does it cost? What functionality does it currently have? You may want to look at Wikipedia to answer this question. (6 marks)

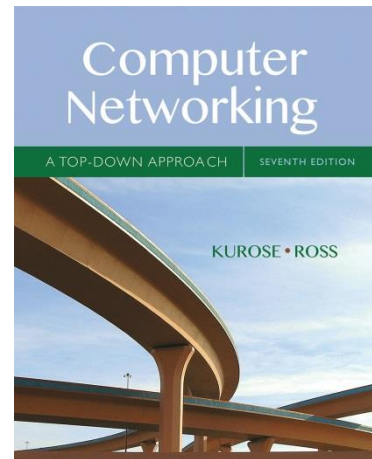
## Part B – Wireshark Lab

### Wireshark Lab: HTTP v7.0

Supplement to *Computer Networking: A Top-Down Approach*, 7<sup>th</sup> 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

© 2005-2016, J.F Kurose and K.W. Ross, All Rights Reserved



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 text.<sup>1</sup>

#### 1. The Basic HTTP GET/response interaction

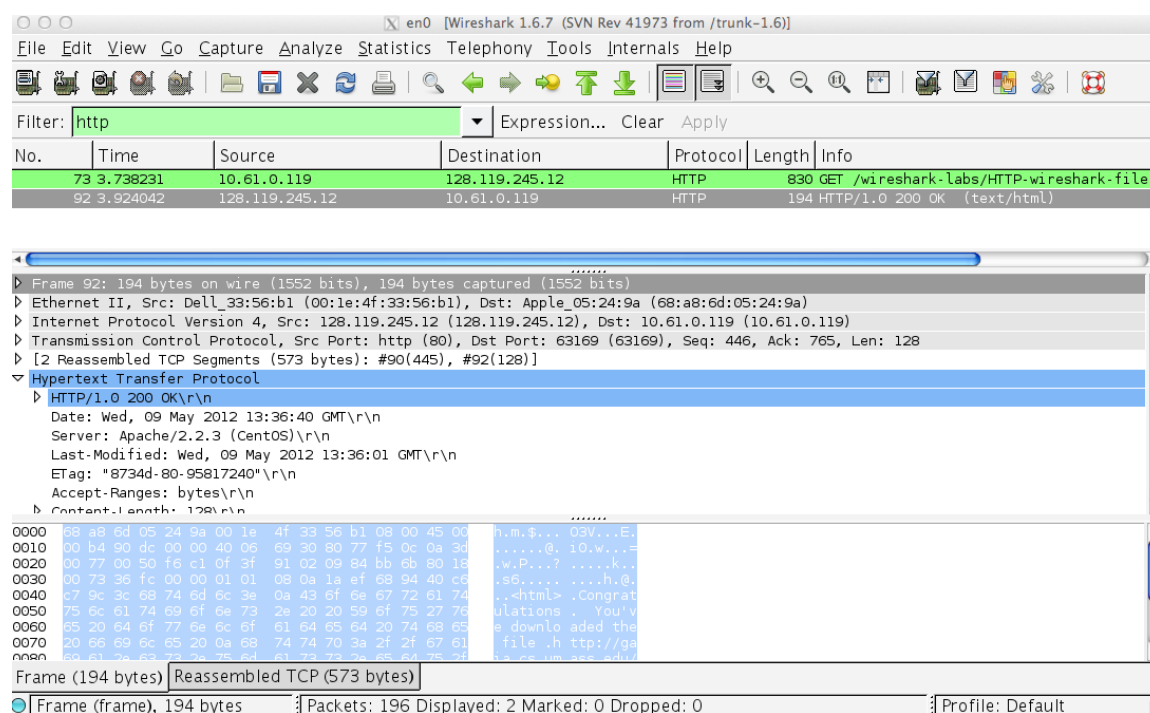
Let's begin our exploration of HTTP by downloading a very simple HTML file - one that is very short, and contains no embedded objects. Do the following:

1. Start up your web browser.
2. Start up the Wireshark packet sniffer, as described in the Introductory 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://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.

---

<sup>1</sup> References to figures and sections are for the 7<sup>th</sup> edition of our text, *Computer Networks, A Top-down Approach*, 7<sup>th</sup> ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2016.

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.<sup>2</sup>



**Figure 1:** Wireshark Display after [http://gaia.cs.umass.edu/wireshark-labs/ HTTP-wireshark-file1.html](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](http://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).

<sup>2</sup> Download the zip file <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip> and extract the file `http-ethereal-trace-1`. The traces in this zip file were collected by Wireshark running on one of the author's computers, while performing the steps indicated in the Wireshark lab. Once you have downloaded the trace, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the `http-ethereal-trace-1` trace file. The resulting display should look similar to Figure 1. (The Wireshark user interface displays just a bit differently on different operating systems, and in different versions of Wireshark).

(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 (e.g., for our classes, we ask that students markup paper copies with a pen, or annotate electronic copies with text in a colored font).

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 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 from Section 2.2.5 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://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.

### 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.
- (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-3 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.)

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 messages 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?

## 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 gaia.cs.umass.edu web site. The image of the cover for our 5<sup>th</sup> edition (one of our favorite covers) is stored at the caite.cs.umass.edu server. (These are two different web servers inside cs.umass.edu).
- Stop Wireshark packet capture, and enter “http” in the display-filter-specification window, so that only captured HTTP messages will be displayed.
- (*Note:* If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-4 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:

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.

## 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](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](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.
- (Note: If you are unable to run Wireshark on a live network connection, you can use the http-ethereal-trace-5 packet trace to answer the questions below; see footnote 2. This trace file was gathered while performing the steps above on one of the author's computers.)

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](http://frontier.userland.com/stories/storyReader$2159)

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?
19. 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 (wireshark-students) and password (network) that you entered are encoded in the string of characters (d2lyZXNoYXJrLXN0dWRlbnRzOm5ldHdvcm0=) 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 <http://www.motobit.com/util/base64-decoder-encoder.asp> and enter the base64-encoded string d2lyZXNoYXJrLXN0dWRlbnRz and decode. *Voila!* You have translated from Base64 encoding to ASCII encoding, and thus should see your username! To view the password, enter

the remainder of the string Om5ldHdvcmS= and press decode. 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.

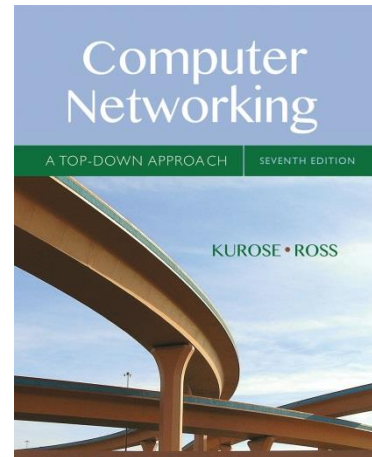
Fear not! As we will see in Chapter 8, there are ways to make WWW access more secure. However, we'll clearly need something that goes beyond the basic HTTP authentication framework!

# Wireshark Lab: DNS v7.0

Supplement to *Computer Networking: A Top-Down Approach*, 7<sup>th</sup> 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

© 2005-2016, J.F Kurose and K.W. Ross, All Rights Reserved



As described in Section 2.4 of the text<sup>3</sup>, the Domain Name System (DNS) translates hostnames to IP addresses, fulfilling a critical role in the Internet infrastructure. In this lab, we'll take a closer look at the client side of DNS. Recall that the client's role in the DNS is relatively simple – a client sends a *query* to its local DNS server, and receives a *response* back. As shown in Figures 2.19 and 2.20 in the textbook, much can go on “under the covers,” invisible to the DNS clients, as the hierarchical DNS servers communicate with each other to either recursively or iteratively resolve the client's DNS query. From the DNS client's standpoint, however, the protocol is quite simple – a query is formulated to the local DNS server and a response is received from that server.

Before beginning this lab, you'll probably want to review DNS by reading Section 2.4 of the text. In particular, you may want to review the material on **local DNS servers**, **DNS caching**, **DNS records and messages**, and the **TYPE field** in the DNS record.

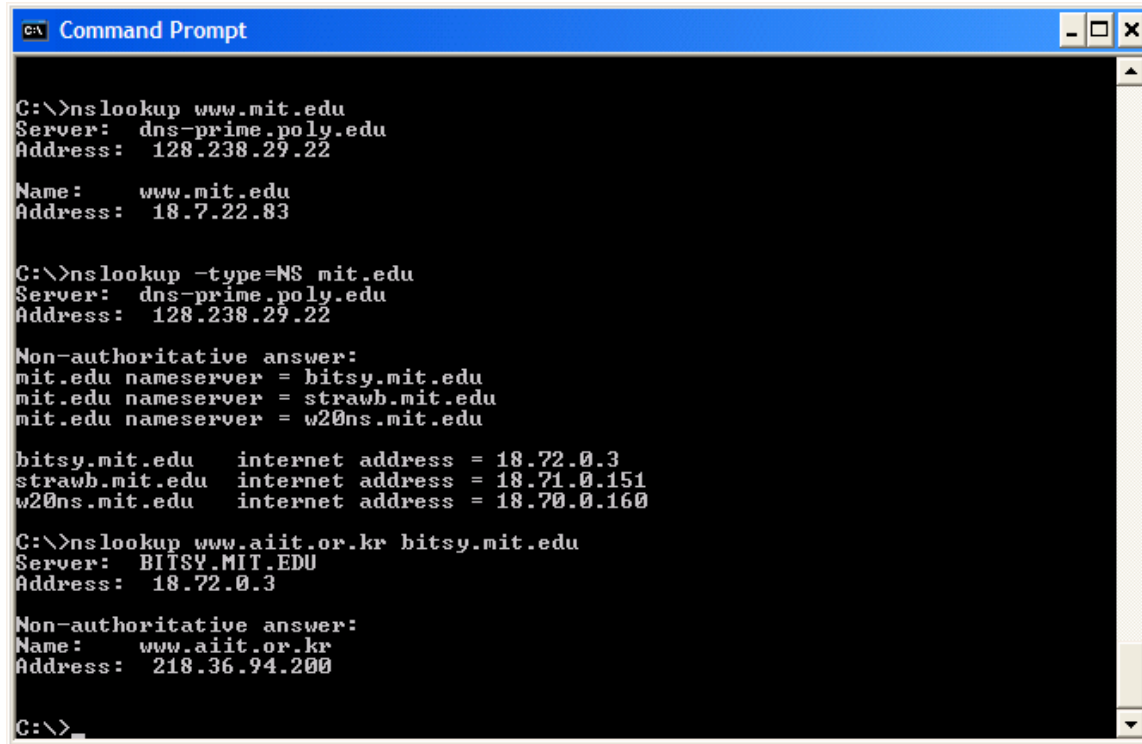
## 1. nslookup

In this lab, we'll make extensive use of the *nslookup* tool, which is available in most Linux/Unix and Microsoft platforms today. To run *nslookup* in Linux/Unix, you just type the *nslookup* command on the command line. To run it in Windows, open the Command Prompt and run *nslookup* on the command line.

In its most basic operation, *nslookup* tool allows the host running the tool to query any specified DNS server for a DNS record. The queried DNS server can be a root DNS server, a top-level-domain DNS server, an authoritative DNS server, or an intermediate DNS server (see the textbook for definitions of these terms). To accomplish this task, *nslookup* sends a DNS query to the specified DNS server, receives a DNS reply from that same DNS server, and displays the result.

---

<sup>3</sup> References to figures and sections are for the 7<sup>th</sup> edition of our text, *Computer Networks, A Top-down Approach*, 7<sup>th</sup> ed., J.F. Kurose and K.W. Ross, Addison-Wesley/Pearson, 2016.



```
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:\>
```

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 we want 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):

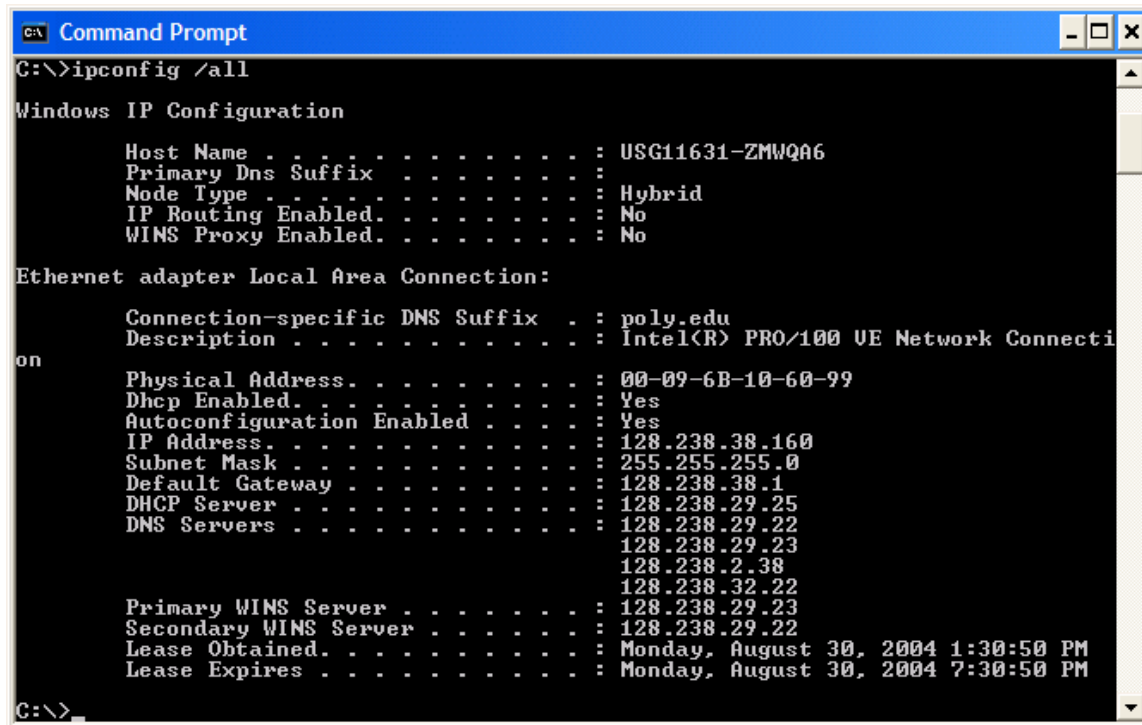
1. Run *nslookup* to obtain the IP address of a Web server in Asia. What is the IP address of that server?
2. Run *nslookup* to determine the authoritative DNS servers for a university in Europe.
3. Run *nslookup* so that one of the DNS servers obtained in Question 2 is queried for the mail servers for Yahoo! mail. What is its IP address?

## 2. ipconfig

*ipconfig* (for Windows) and *ifconfig* (for Linux/Unix) 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 *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 all this information about your host simply by entering

```
ipconfig \all
```

into the Command Prompt, as shown in the following screenshot.



```
C:\>ipconfig /all

Windows IP Configuration

    Host Name . . . . . : USG11631-ZMWQA6
    Primary Dns Suffix . . . . . : poly.edu
    Node Type . . . . . : Hybrid
    IP Routing Enabled. . . . . : No
    WINS Proxy Enabled. . . . . : No

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix . : poly.edu
    Description . . . . . : Intel(R) PRO/100 VE Network Connecti
on
    Physical Address. . . . . : 00-09-6B-10-60-99
    Dhcp Enabled. . . . . : Yes
    Autoconfiguration Enabled . . . . : Yes
    IP Address. . . . . : 128.238.38.160
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 128.238.38.1
    DHCP Server . . . . . : 128.238.29.25
    DNS Servers . . . . . : 128.238.29.22
                           128.238.29.23
                           128.238.2.38
                           128.238.32.22
    Primary WINS Server . . . . . : 128.238.29.23
    Secondary WINS Server . . . . . : 128.238.29.22
    Lease Obtained. . . . . : Monday, August 30, 2004 1:30:50 PM
    Lease Expires . . . . . : Monday, August 30, 2004 7:30:50 PM

C:\>
```

*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, after the prompt C:\> provide the following command:

```
ipconfig /displaydns
```

Each entry shows the remaining Time to Live (TTL) in seconds. To clear the cache, enter

```
ipconfig /flushdns
```

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

### 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* to empty the DNS cache in your host.
- Open your browser and empty your browser cache. (With Internet Explorer, go to Tools menu and select Internet Options; then in the General tab select Delete Files.)
- Open Wireshark and enter "ip.addr == your\_IP\_address" into the filter, where you obtain your\_IP\_address with *ipconfig*. 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.

If you are unable to run Wireshark on a live network connection, you can download a packet trace file that was captured while following the steps above on one of the author's computers<sup>4</sup>. 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<sup>5</sup> 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 then 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?

---

<sup>4</sup> Download the zip file <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip> and extract the file *dns-ethereal-trace-1*. The traces in this zip file were collected by Wireshark running on one of the author's computers, while performing the steps indicated in the Wireshark lab. Once you have downloaded the trace, you can load it into Wireshark and view the trace using the *File* pull down menu, choosing *Open*, and then selecting the *dns-ethereal-trace-1* trace file.

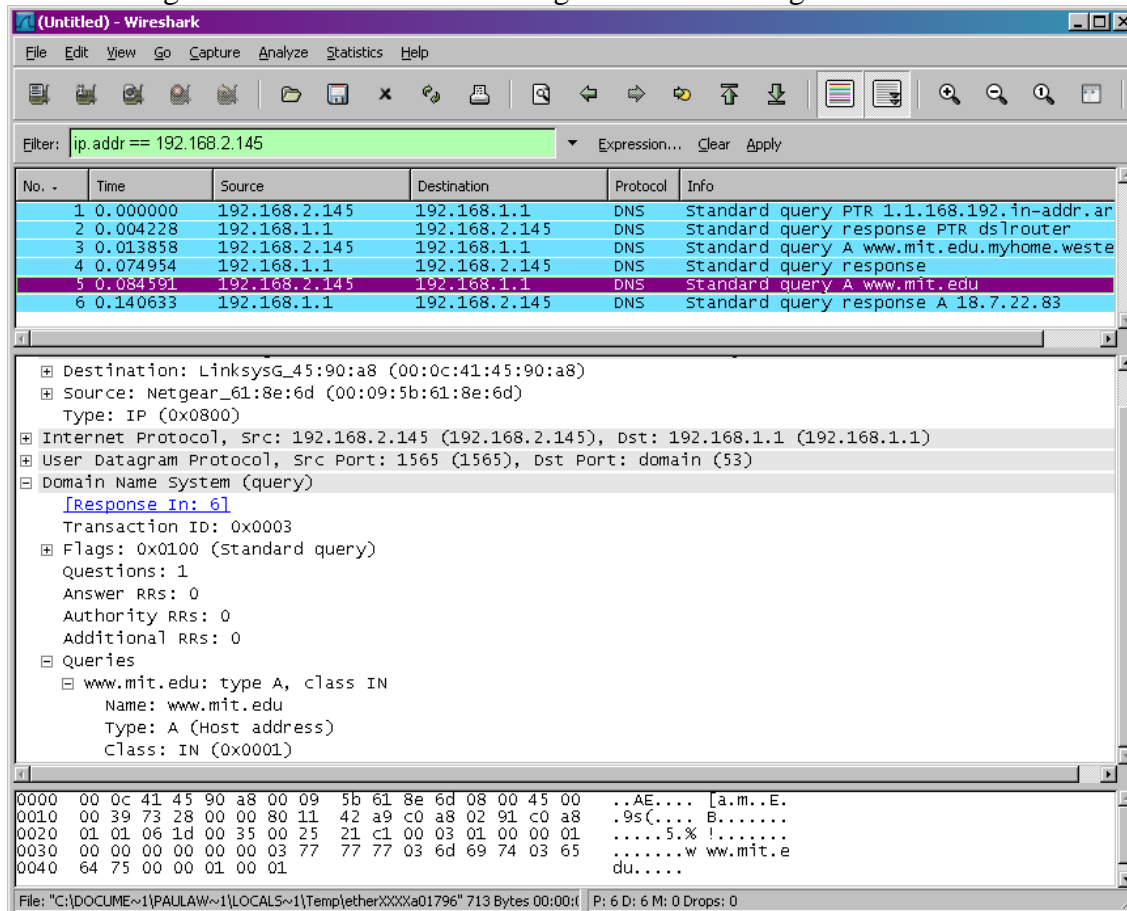
<sup>5</sup> What do we mean by "annotate"? If you hand in a paper copy, please highlight where in the printout you've found the answer and add some text (preferably with a colored pen) noting what you found in what you've highlight. If you hand in an electronic copy, it would be great if you could also highlight and annotate.

10. This web page contains images. Before retrieving each image, does your host issue new DNS queries?

Now let's play with *nslookup*<sup>6</sup>.

- Start packet capture.
- Do an *nslookup* on `www.mit.edu`
- Stop packet capture.

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



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.

<sup>6</sup> If you are unable to run Wireshark and capture a trace file, use the trace file `dns-ethereal-trace-2` in the zip file <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip>



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<sup>7</sup> :

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.

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

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

Answer the following questions<sup>8</sup>:

20. To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server? If not, what does the IP address correspond to?
21. Examine the DNS query message. What “Type” of DNS query is it? Does the query message contain any “answers”?
22. Examine the DNS response message. How many “answers” are provided? What does each of these answers contain?
23. Provide a screenshot.

---

<sup>7</sup> If you are unable to run Wireshark and capture a trace file, use the trace file dns-ethereal-trace-3 in the zip file <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip>

<sup>8</sup> If you are unable to run Wireshark and capture a trace file, use the trace file dns-ethereal-trace-4 in the zip file <http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip>