Problem Set 3
Due: Start of Class, Thursday, September 28

Reading: Kurose & Ross, Sections 2.3, 2.4

Wireshark Lab [14]

The Domain Name System (DNS) translates hostnames to IP addresses. As discuss in class, 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. We examine these interactions from the clients perspective in these exercises.

1 nslookup

The Linux/Unix nslookup tool allows the hosts 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. You can type nslookup commands on in a terminal window on your Mac. For example, consider the following terminal session. The client host in my Mac located on Science Center, where the default local DNS server is Cheers. When running nslookup, if no DNS server is specified, then nslookup sends the query to the default DNS server. The first command nslookup www.smith.edu asks for the IP address of the host www.smith.edu. The response from this command provides two pieces of information: (1) the name and IP address of the DNS server that provides the answer (that would be server 149.130.10.16 who happens to be Cheers); and (2) the answer itself, which is the host name and IP address of www.smith.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,
The second command, `nslookup -type=NS smith.edu`, provides the “-type=NS” option and the domain “smith.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 smith.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 transcript, first indicates the DNS server that is providing the answer (which is Cheers, our default local DNS server) along with three five smith.edu nameservers. Each of these servers might be an authoritative
DNS server for the hosts on the Smith campus. However, \texttt{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 Smith DNS server. Finally, the answer also includes the IP addresses of the authoritative DNS servers at Smith. (Even though the type-NS query generated by \texttt{nslookup} did not explicitly ask for the IP addresses, the local DNS server returned this “for free” and \texttt{nslookup} displays the result.)

Finally consider the third command:

\begin{verbatim}
nslookup www.smith.edu babel.smith.edu.
\end{verbatim}

In this example, we indicate that we want to the query sent to the DNS server \texttt{babel.smith.edu} rather than to the default DNS server. Thus, the query and reply transaction takes place directly between our querying host and \texttt{babel.smith.edu}. In this example, the DNS server \texttt{babel.smith.edu} provides the IP address of the host \texttt{www.smith.edu}.

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

\begin{verbatim}
nslookup -option1 -option2 host-to-find dns-server
\end{verbatim}

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

\textbf{Exercise 1 [2]: nslookup}

\textbf{a} [1] Run \texttt{nslookup} to obtain the IP address of a Web server in Asia. What is the IP address of that server?

\textbf{b} [1] Run \texttt{nslookup} to determine the authoritative DNS servers for a university in Europe.

The \texttt{ifconfig} Linux/Unix tool (\texttt{ipconfig} for Windows\textsuperscript{1}) is an especially useful tool for debugging network issues. It can be used to show your current TCP/IP information, including your address, DNS server addresses, adapter type and so on. For example, simply typing \texttt{ifconfig} in a terminal window yields the following information:

\textsuperscript{1}The DNS lab on the authors’ website discusses \texttt{ipconfig} in more detail
Admittedly this is more than I want to know (we can and will refine our queries in later sessions). Among other things it tells me that I have a loopback on interface lo0, an active ethernet with mac address 40:6c:8f:0a:5a:07 on interface en0 with IP address 149.130.178.106, and that my WiFi is inactive (as it should be in preparation for Wireshark experiments) on interface fw0 with its own mac address 3c:07:54:ff:fe:df:79:e6.

2 Tracing DNS with Wireshark

Your computer will probably have a local cache of DNS records. This is good since it means that you need not dial directory information as often as you otherwise might. However, it is not so good for the following exercises since we will not be able to see DNS in action. Before we can trace DNS interactions we will need to clear this cache. To do this on OS X version 10.6 type: sudo dscacheutil -flushcache followed by your passwd.\(^2\)

\(^2\)A summary of the contents of your cache may be obtained using: dscacheutil -cachedump
Yosemite v10.10 through 10.10.3: `sudo discoveryutil mdnsflushcache`. For Mavericks, Mountain Lion, Lion and Yosemite v10.10.4: `sudo killall -HUP mDNSResponder`.\(^3\) If you have a Windows machine try: `ipconfig /flushdns`.

Fire up Wireshark and enter: `ip.addr==your_ip_address` into the filter, where `your_ip_address` is your IP address. On the MAC you can find your IP address by selecting Network under System Preferences ... in the apple menu. Alternatively, you can use either `ifconfig` (MAC) or `ipconfig` (Windows) as discussed above. Open your browser and clear the cache. Finally, open a terminal window and clear your local DNS cache as above. Start the Wireshark capture and type `http://www.ietf.org` into your browser. Stop the packet capture.\(^4\)

**Exercise 2 [8]: DNS query/response**

**a [1]** Locate the DNS query and response messages. Are they sent over UDP or TCP?

**b [1]** What is the destination port for the DNS query message? What is the source port of the DNS response message?

**c [1]** 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?

**d [1]** Examine the DNS query message. What Type of DNS query is it? Does the query message contain any answers?

**e [1]** Examine the DNS response message. How many answers are provided? What do each of these answers contain?

**f [2]** 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?

**g [1]:** This web page contains images. Before retrieving each image, does your host issue new DNS queries?

\(^3\)The **`sudo killall -INFO mDNSResponder`** command will send a snapshot summary of the internal state of your machine to `~/var/log/system.log` including the contents of your cache and, unfortunately, a whole lot more.

\(^4\)If you cannot clear your cache, you still have one shot at this. But it will only work the first time.
Exercise 3 [4]: Capturing an nslookup request

Restart Wireshark capture and enter the following command: `nslookup -type=MX mit.edu`.

Answer the following questions:

**a [1]** To what IP address is the DNS query message sent? Is this the IP address of your default local DNS server?

**b [1]** Examine the DNS query message. What “Type” of DNS query is it? Does the query message contain any “answers”?

**c [2]** Examine the DNS response message. What MIT nameservers does the response message provide? Does this response message also provide IP addresses of MIT nameservers?
Problems

Here’s your chance to amaze your friends and relatives (well maybe not amaze, but amuse anyway). This exercise set has two parts. In the first you will use telnet to manually send mail through an SMTP mail server. In the second part, you will write a simple Python or Java program that performs the same action. Later when we learn more about socket programming you will improve upon this simple mail user agent.

In order to limit spam, some mail servers do not accept TCP connection from arbitrary sources. For the experiment described below you should send your email both manually and through your program from the cs server, tempest. You may also try making your connection both from your home. Even with tempest on your side the receiving mail server might classify your e-mail as junk. Make sure you check the junk/spam folder when you look for the e-mail sent from your client.

Problem 1 [4]: Only spoofing When you do this problem, you should try to send email to yourself. This means you need to know the host name of the mail server for your mail domain. To find out this information, you can query DNS for the MX record that maintains information about your mail domain. For example, rshull@wellesley.edu has mail domain wellesley.edu. The following command queries DNS for the mail servers responsible for delivering mail to this domain:

```
nslookup -type=MX wellesley.edu
```

For the response to this command, there may be several mail servers (there will be) that will deliver mail to mailboxes in the domain wellesley.edu. Suppose that the name of one of them is mx1.someschool.edu. In this case, the following command will establish a TCP connection to this mail server. (Notice that the port number 25 is specified on the command line.)

```
telnet mx1.wellesley.edu 25
```

At this point, the telnet program will allow you to enter SMTP commands, and will display the responses from the mail server. For example, the following sequence of commands would send email to Randy from Randy (and passed the SPAM filter which means we could do some spoofing – but that would be wrong):

```
HELO wellesley.edu

MAIL FROM:<rshull@wellesley.edu>
```
RCPT TO:<rshull@wellesley.edu>

DATA

FROM: Randy Shull <rshull@wellesley.edu>
TO: Randy Shull <rshull@wellesley.edu>
SUBJECT: Saying Hello

This is a message

QUIT

The SMTP protocol was originally designed to allow people to manually interact with mail servers in a conversational manner. For this reason, if you enter a command with incorrect syntax, or with unacceptable arguments, the server will return a message stating this, and will allow you to try again.

To complete this part of the problem, you should send an email message to yourself and verify that it was delivered. Then send one to your instructor (whom you may spoof if you wish, but let me know in the body that you are only spoofing.)

Problem 2 [10]: Automated email  Your task is to develop a simple mail client that sends email to any recipient.\(^5\) Your client will need to connect to a mail server, dialogue with the mail server using the SMTP protocol, and send an email message to the mail server. Both Python and Java provide a API's for interacting with the Internet mail system, which are called \texttt{smtplib} and \texttt{JavaMail}. However, we will not be using these API's, because they hide the details of SMTP and socket programming. Instead, you should write either a Python or Java program that establishes a TCP connection with a mail server through the socket interface,\(^6\) and sends an email message. What follows are two sets of skeleton code for you to fill in. Select one, and fill in the places that marked with Fill in \texttt{start} and Fill in \texttt{end}. Each place may require one or more lines of code.

Skeleton Python Code for the Mail Client

You can place all of your code into a file called \texttt{EmailSender.py}. This

\(^5\)As in the previous exercise you will need to run your program from \texttt{tempest}. Google is a very suspicious animal and won’t accept messages from just anyone.

\(^6\)You guessed it, more on this later.
means you will include in your code the details of the particular email message you are trying to send. Run your Python program on `tempest` using the command:

```
python EmailSender.py
```

No compilation is necessary. Here is a skeleton of the code you’ll need to write:

```python
from socket import *

# Write you own message. Something a bit more imaginative than below:
msg = 
endmsg = 

# Choose a mail server (e.g. Google mail server) and call it mailserver.
mailserver = #Fill in start #Fill in end
port = #Fill in start #Fill in end

# Create socket called clientSocket
clientSocket = socket(AF_INET, SOCK_STREAM)

# and establish a connection with the mailserver
clientSocket.connect((mailserver, port))

recv = clientSocket.recv(1024)
print recv if recv[:3] != '220':
    print '220 reply not received from server.'

# Send HELO command.
heloCommand = 'HELO Alice\n'
clientSocket.send(heloCommand)

# Get back and print the response
recv1 = clientSocket.recv(1024)
print recv1 if recv1[:3] != '250':
    print '250 reply not received from server.'

# Send MAIL FROM command and print server response.
# Fill in start

# Fill in end

# Send RCPT TO command and print server response.
# Fill in start

# Fill in end

# Send DATA command and print server response.
# Fill in start

# Fill in end

# Send message data.
# Fill in start

# Fill in end

# Message ends with a single period.
# Fill in start

# Fill in end

# Send QUIT command and get server response.
# Fill in start

# Fill in end

Skeleton Java Code for the Mail Client

You can place all of your code into the main method of a class called EmailSender.java. Compile and run your program:

    javac EmailSender.java
java EmailSender

This means you will include in your code the details of the particular email message you are trying to send. Here is a skeleton of the code you'll need to write:

```java
import java.io.*;
import java.net.*;

public class EmailSender
{
    public static void main(String[] args) throws Exception
    {
        // Define mailserver and port number

        // Establish a TCP connection with the mail server.
        Socket socket = new Socket(mailserver, port);

        // Create a BufferedReader to read a line at a time.
        InputStream is = socket.getInputStream();
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);

        // Read greeting from the server.
        String response = br.readLine();
        System.out.println(response);
        if (!response.startsWith("220")) {
            throw new Exception("220 reply not received from server.");
        }

        // Get a reference to the socket’s output stream.
        OutputStream os = socket.getOutputStream();

        // Send HELO command and get server response.
        String command = "HELO alice\r\n";
        System.out.print(command);
        os.write(command.getBytes("US-ASCII"));
    }
}
```
response = br.readLine();
System.out.println(response);
if (!response.startsWith("250")) {
    throw new Exception("250 reply not received from server.");
}

// Send MAIL FROM command.

// Send RCPT TO command.

// Send DATA command.

// Send message data.

// End with line with a single period.

// Send QUIT command.

// Close the socket.
socket.close();
}
Problem 3 [2]: Protocols Consider an HTTP client that wants to retrieve a Web document at a given URL. The IP address of the HTTP server is initially unknown. The Web document at the URL has one embedded GIF image that resides at the same server as the original document. What transport and application-layer protocols besides HTTP are needed in this scenario?

Problem 4 [4]: DNS Hierarchy On Unix and Linux host the dig tool can be used to explore the hierarchy of DNS servers. Recall that DNS servers higher in the hierarchy delegate DNS queries to DNS servers lower in the hierarchy, by sending back to the DNS client the name of that lower-lever DNS server. A typical invocation of dig looks like:

\[
\text{dig +norecurse @server type name}
\]

where server is the name or IP address of the name server to query. This can be an IPv4 address in dotted-decimal notation or an IPv6 address in colon-delimited notation. When the supplied server argument is a hostname, dig resolves that name before querying that name server. If no server argument is provided, dig consults /etc/resolv.conf and queries the name servers listed there. The reply from the name server that responds is displayed. Here name is the name of the resource record that is to be looked up and type indicates what type of query is required (any, a, mx, sig, etc.) type can be any valid query type. If no type argument is supplied, dig will perform a lookup for an “a” record. The “any” type tells all. The +norecurse toggles off recursive queries.

You are encouraged to read the man page for dig or visit your favorite online source, e.g.,

https://www.lifewire.com/dig-linux-command-4096131

for more information.

Starting with a root DNS server (for one of the root servers [a-m].root-servers.net), initiate a sequence of queries for the IP address for the cs department’s web server, cs.wellesley.edu. Show the list of names of DNS servers in the delegation chain in answering your query.

Problem 5 [6]: To whom are we talking? The NSA isn’t the only one who knows to whom we are talking.

a [2]: Troubling Suppose you can access the caches in the local DNS servers of your department. Can you propose a way to roughly determine
the Web Servers (outside your department) that are most popular among the users in your department?

b [4]: A little more troubling Of course most of us do not have direct access to the cache of our local DNS server. Still, it might be possible for Joan Average to determine if an external Web site was very likely accessed from a computer in her department a couple of seconds ago. How?