Print Your Name: ______________________

CSC 360 Lab 2

Preparations:

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

End of preparation.

TCP

In this lab, we'll investigate the behavior of TCP in detail. We'll do so by analyzing a trace of the TCP segments sent and received in transferring a 150KB file (containing the text of Lewis Carrol's *Alice's Adventures in Wonderland*) from your computer to a remote server. We'll study TCP's use of sequence and acknowledgement numbers for providing reliable data transfer; we'll see TCP's congestion control algorithm – slow start and congestion avoidance – in action; and we'll look at TCP's receiver-advertised flow control mechanism. We'll also briefly consider TCP connection setup and we'll investigate the performance (throughput and round-trip time) of the TCP connection between your computer and the server.

Before beginning this lab, you'll probably want to review the related sections in the textbook.

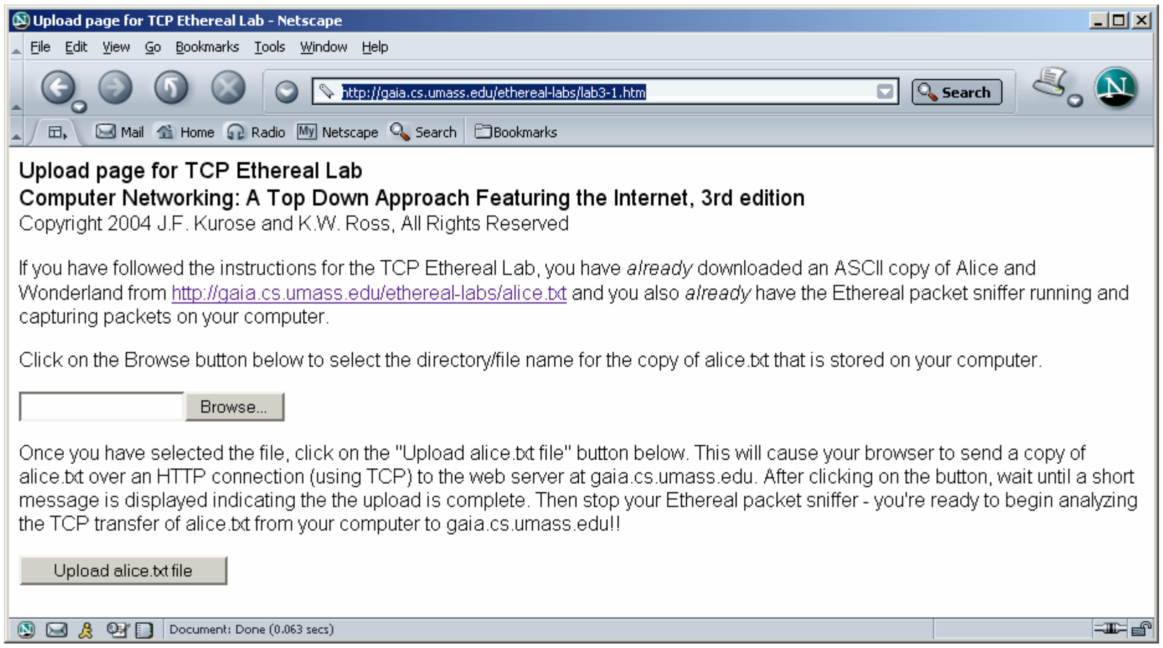
1. Capturing a bulk TCP transfer from your computer to a remote server

Before beginning our exploration of TCP, we'll need to use Wireshark to obtain a packet trace of the TCP transfer of a file from your computer to a remote server. You'll do so by accessing a Web page that will allow you to enter the name of a file stored on your computer (which contains the ASCII text of *Alice in Wonderland*), and then transfer the file to a Web server using the HTTP **POST** method (see the related section in the text). We're using the POST method rather than the GET method as we'd like to transfer a large amount of data *from* your computer to another computer. Of course, we'll be running Wireshark during this time to obtain the trace of the TCP segments sent and received from your computer.

Do the following:

- Start up your web browser. Go the http://gaia.cs.umass.edu/wireshark-labs/alice.txt your computer.

- Next go to http://gaia.cs.umass.edu/wireshark-labs/TCP-wireshark-file1.html.
- You should see a screen that looks like:


Upload page for TCP Ethereal Lab - Netscape
File Edit View Go Bookmarks Tools Window Help
http://gaia.cs.umass.edu/ethereal-labs/lab3-1.htm
Search
Mail Home Radio Netscape Search Bookmarks
Browse...
Upload alice.txt file
Document: Done (0.063 secs)

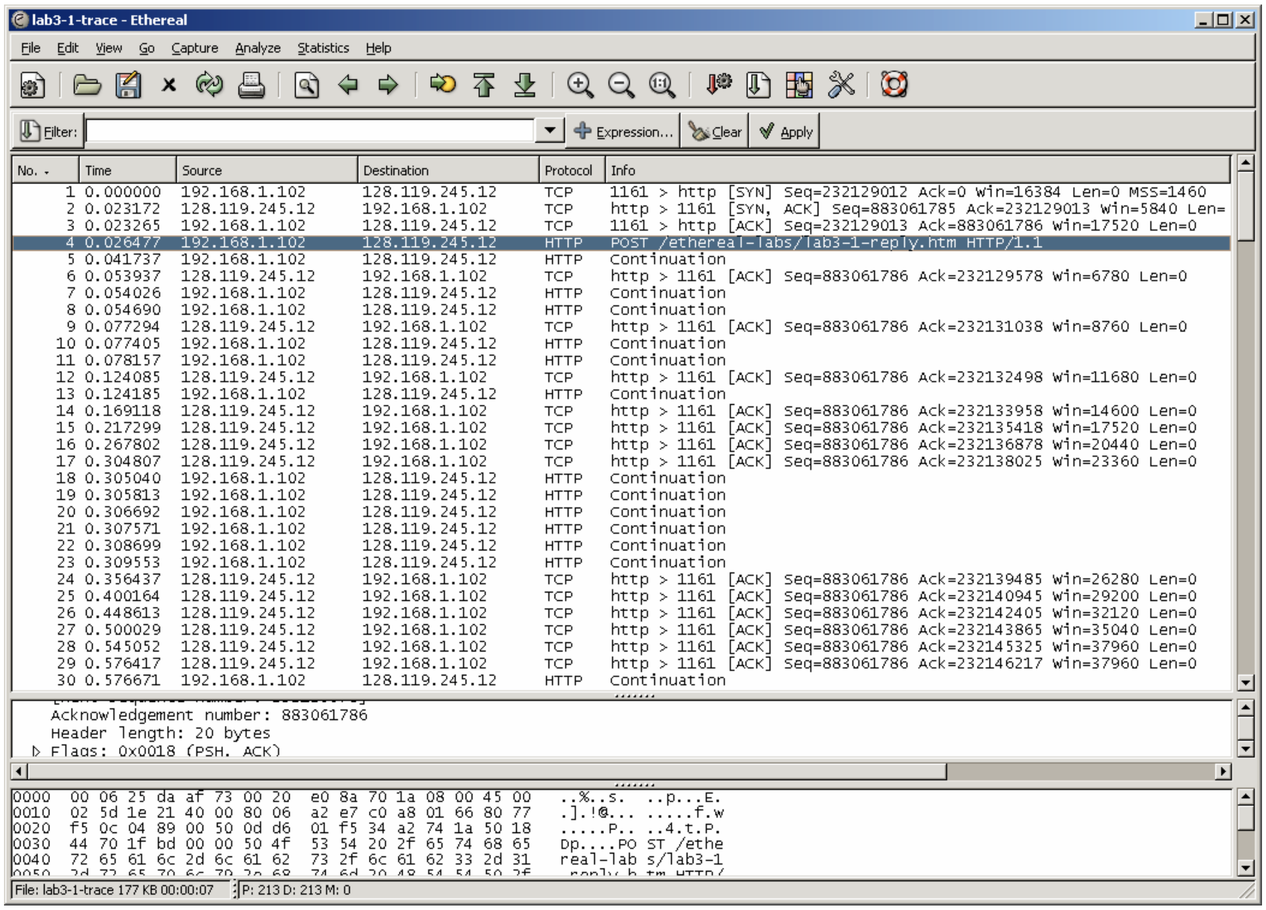
Upload page for TCP Ethereal Lab
Computer Networking: A Top Down Approach Featuring the Internet, 3rd edition
Copyright 2004 J.F. Kurose and K.W. Ross, All Rights Reserved

If you have followed the instructions for the TCP Ethereal Lab, you have *already* downloaded an ASCII copy of Alice and Wonderland from http://gaia.cs.umass.edu/ethereal-labs/alice.txt and you also *already* have the Ethereal packet sniffer running and capturing packets on your computer.

Click on the Browse button below to select the directory/file name for the copy of alice.txt that is stored on your computer.

Once you have selected the file, click on the "Upload alice.txt file" button below. This will cause your browser to send a copy of alice.txt over an HTTP connection (using TCP) to the web server at gaia.cs.umass.edu. After clicking on the button, wait until a short message is displayed indicating the the upload is complete. Then stop your Ethereal packet sniffer - you're ready to begin analyzing the TCP transfer of alice.txt from your computer to gaia.cs.umass.edu!!

- Use the *Browse* button in this form to enter the name of the file (full path name) on your computer containing *Alice in Wonderland* (or do so manually). Don't yet press the "*Upload alice.txt file*" button.
- Now start up Wireshark and begin packet capture *(Capture->Start)* and then press *OK* on the Wireshark Packet Capture Options screen (we'll not need to select any options here).
- Returning to your browser, press the "*Upload alice.txt file*" button to upload the file to the gaia.cs.umass.edu server. Once the file has been uploaded, a short congratulations message will be displayed in your browser window.
- Stop Wireshark packet capture. Your Wireshark window should look similar to the window shown below.



```
No. -  Time      Source          Destination     Protocol Info
     1 0.000000  192.168.1.102   128.119.245.12  TCP      1161 > http [SYN] Seq=232129012 Ack=0 Win=16384 Len=0 MSS=1460
     2 0.023172  128.119.245.12  192.168.1.102   TCP      http > 1161 [SYN, ACK] Seq=883061785 Ack=232129013 Win=5840 Len=
     3 0.023265  192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232129013 Ack=883061786 Win=17520 Len=0
     4 0.026477  192.168.1.102   128.119.245.12  HTTP     POST /ethereal-labs/lab3-1-reply.htm HTTP/1.1
     5 0.041737  192.168.1.102   128.119.245.12  HTTP     Continuation
     6 0.053937  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232129578 Win=6780 Len=0
     7 0.054026  192.168.1.102   128.119.245.12  HTTP     Continuation
     8 0.054690  192.168.1.102   128.119.245.12  HTTP     Continuation
     9 0.077294  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232131038 Win=8760 Len=0
    10 0.077405  192.168.1.102   128.119.245.12  HTTP     Continuation
    11 0.078157  192.168.1.102   128.119.245.12  HTTP     Continuation
    12 0.124085  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232132498 Win=11680 Len=0
    13 0.124185  192.168.1.102   128.119.245.12  HTTP     Continuation
    14 0.169118  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232133958 Win=14600 Len=0
    15 0.217299  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232135418 Win=17520 Len=0
    16 0.267802  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232136878 Win=20440 Len=0
    17 0.304807  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232138025 Win=23360 Len=0
    18 0.305040  192.168.1.102   128.119.245.12  HTTP     Continuation
    19 0.305813  192.168.1.102   128.119.245.12  HTTP     Continuation
    20 0.306692  192.168.1.102   128.119.245.12  HTTP     Continuation
    21 0.307571  192.168.1.102   128.119.245.12  HTTP     Continuation
    22 0.308699  192.168.1.102   128.119.245.12  HTTP     Continuation
    23 0.309553  192.168.1.102   128.119.245.12  HTTP     Continuation
    24 0.356437  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232139485 Win=26280 Len=0
    25 0.400164  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232140945 Win=29200 Len=0
    26 0.448613  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232142405 Win=32120 Len=0
    27 0.500029  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232143865 Win=35040 Len=0
    28 0.545052  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232145325 Win=37960 Len=0
    29 0.576417  128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232146217 Win=37960 Len=0
    30 0.576671  192.168.1.102   128.119.245.12  HTTP     Continuation
```

```
   Acknowledgement number: 883061786
   Header length: 20 bytes
 ▷ Flags: 0x0018 (PSH, ACK)
```

```
0000  00 06 25 da af 73 00 20  e0 8a 70 1a 08 00 45 00   ..%..s.  ..p...E.
0010  02 5d 1e 21 40 00 80 06  a2 e7 c0 a8 01 66 80 77   .].!@... .....f.w
0020  f5 0c 04 89 00 50 0d d6  01 f5 34 a2 74 1a 50 18   .....P.. ..4.t.P.
0030  44 70 1f bd 00 00 50 4f  53 54 20 2f 65 74 68 65   Dp....PO ST /ethe
0040  72 65 61 6c 2d 6c 61 62  73 2f 6c 61 62 33 2d 31   real-lab s/lab3-1
```

```
File: lab3-1-trace 177 KB 00:00:07   P: 213 D: 213 M: 0
```

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 textbook author's computers[2]. You may find it valuable to download this trace even if you've captured your own trace and use it, as well as your own trace, when you explore the questions below.

2. A first look at the captured trace

Before analyzing the behavior of the TCP connection in detail, let's take a high level view of the trace.

- First, filter the packets displayed in the Wireshark window by entering "ip.addr = = 192.168.2.168 && tcp" (lowercase, no quotes, and don't forget to press return after entering! also replace the IP address with your computer IP address) into the display filter specification window towards the top of the Wireshark window.

What you should see is series of TCP and HTTP messages between your computer and gaia.cs.umass.edu. You should see the initial three-way handshake containing a SYN

[2] Download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the file tcp-Wireshark-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 tcp-Wireshark-trace-1 trace file.

message. You should see an HTTP POST message and a series of "HTTP Continuation" messages being sent from your computer to gaia.cs.umass.edu. Recall from our discussion in the earlier HTTP Wireshark lab, that is no such thing as an HTTP Continuation message – this is Wireshark's way of indicating that there are multiple TCP segments being used to carry a single HTTP message. You should also see TCP ACK segments being returned from gaia.cs.umass.edu to your computer.

Answer the following questions, by opening the Wireshark captured packet file *tcp-Wireshark-trace-1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip (that is download the trace and open that trace in Wireshark; see footnote 2). Answer the question.

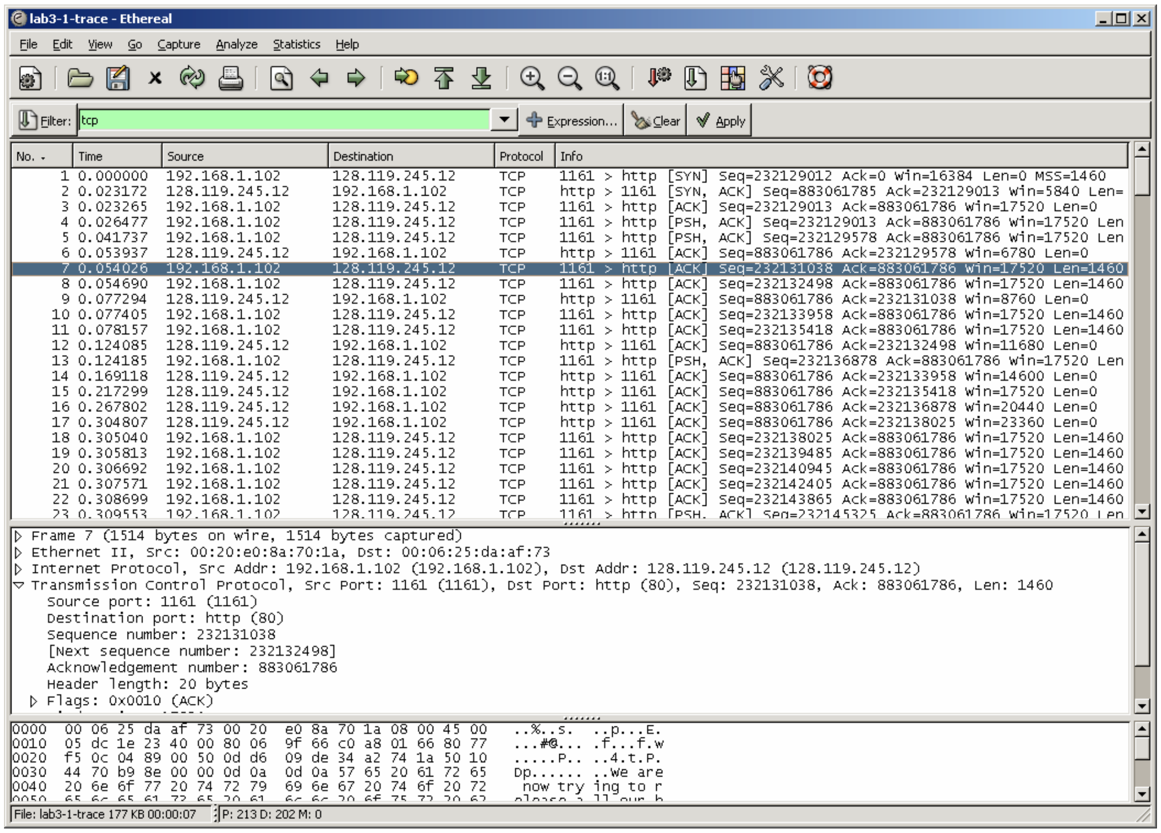
1. What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? To answer this question, it's probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the "details of the selected packet header window" (refer to Figure 2 in the "Getting Started with Wireshark" Lab if you're uncertain about the Wireshark windows.

2. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?

If you have been able to create your own trace, answer the following question:

3. What is the IP address and TCP port number used by your client computer (source) to transfer the file to gaia.cs.umass.edu?

Since this lab is about TCP rather than HTTP, let's change Wireshark's "listing of captured packets" window so that it shows information about the TCP segments containing the HTTP messages, rather than about the HTTP messages. To have Wireshark do this, select *Analyze->Enabled Protocols.* Then uncheck the HTTP box and select *OK*. You should now see an Wireshark window that looks like:


lab3-1-trace - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter: tcp
Expression... Clear Apply

```
No. - Time     Source          Destination     Protocol Info
    1 0.000000 192.168.1.102   128.119.245.12  TCP      1161 > http [SYN] Seq=232129012 Ack=0 Win=16384 Len=0 MSS=1460
    2 0.023172 128.119.245.12  192.168.1.102   TCP      http > 1161 [SYN, ACK] Seq=883061785 Ack=232129013 Win=5840 Len=
    3 0.023265 192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232129013 Ack=883061786 Win=17520 Len=0
    4 0.026477 192.168.1.102   128.119.245.12  TCP      1161 > http [PSH, ACK] Seq=232129013 Ack=883061786 Win=17520 Len
    5 0.041737 192.168.1.102   128.119.245.12  TCP      1161 > http [PSH, ACK] Seq=232129578 Ack=883061786 Win=17520 Len
    6 0.053937 128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232129578 Win=6780 Len=0
    7 0.054026 192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232131038 Ack=883061786 Win=17520 Len=1460
    8 0.054690 192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232132498 Ack=883061786 Win=17520 Len=1460
    9 0.077294 128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232131038 Win=8760 Len=0
   10 0.077405 192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232133958 Ack=883061786 Win=17520 Len=1460
   11 0.078157 192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232135418 Ack=883061786 Win=17520 Len=1460
   12 0.124085 128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232132498 Win=11680 Len=0
   13 0.124185 192.168.1.102   128.119.245.12  TCP      1161 > http [PSH, ACK] Seq=232136878 Ack=883061786 Win=17520 Len
   14 0.169118 128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232133958 Win=14600 Len=0
   15 0.217299 128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232135418 Win=17520 Len=0
   16 0.267802 128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232136878 Win=20440 Len=0
   17 0.304807 128.119.245.12  192.168.1.102   TCP      http > 1161 [ACK] Seq=883061786 Ack=232138025 Win=23360 Len=0
   18 0.305040 192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232138025 Ack=883061786 Win=17520 Len=1460
   19 0.305813 192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232139485 Ack=883061786 Win=17520 Len=1460
   20 0.306692 192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232140945 Ack=883061786 Win=17520 Len=1460
   21 0.307571 192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232142405 Ack=883061786 Win=17520 Len=1460
   22 0.308699 192.168.1.102   128.119.245.12  TCP      1161 > http [ACK] Seq=232143865 Ack=883061786 Win=17520 Len=1460
   23 0.309553 192.168.1.102   128.119.245.12  TCP      1161 > http [PSH, ACK] Seq=232145325 Ack=883061786 Win=17520 Len
```

```
▷ Frame 7 (1514 bytes on wire, 1514 bytes captured)
▷ Ethernet II, Src: 00:20:e0:8a:70:1a, Dst: 00:06:25:da:af:73
▷ Internet Protocol, Src Addr: 192.168.1.102 (192.168.1.102), Dst Addr: 128.119.245.12 (128.119.245.12)
▽ Transmission Control Protocol, Src Port: 1161 (1161), Dst Port: http (80), Seq: 232131038, Ack: 883061786, Len: 1460
    Source port: 1161 (1161)
    Destination port: http (80)
    Sequence number: 232131038
    [Next sequence number: 232132498]
    Acknowledgement number: 883061786
    Header length: 20 bytes
  ▷ Flags: 0x0010 (ACK)
```

```
0000  00 06 25 da af 73 00 20  e0 8a 70 1a 08 00 45 00   ..%..s.  ..p...E.
0010  05 dc 1e 23 40 00 80 06  9f 66 c0 a8 01 66 80 77   ...#@... .f...f.w
0020  f5 0c 04 89 00 50 0d d6  09 de 34 a2 74 1a 50 10   .....P.. ..4.t.P.
0030  44 70 b9 8e 00 00 0d 0a  0d 0a 57 65 20 61 72 65   Dp...... ..We are
0040  20 6e 6f 77 20 74 72 79  69 6e 67 20 74 6f 20 72    now try ing to r
```

```
File: lab3-1-trace 177 KB 00:00:07   P: 213 D: 202 M: 0
```

This is what we're looking for - a series of TCP segments sent between your computer and gaia.cs.umass.edu. We will use the packet trace that you have captured (and/or the packet trace *tcp-Wireshark-trace-1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip) to study TCP behavior in the rest of this lab.

3. TCP Basics

Answer the following questions for the TCP segments:

4. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?

5. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the ACKnowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?

6. What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Wireshark window, looking for a segment with a "POST" within its DATA field.

7. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in theTCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the EstimatedRTT value (see page 249 in text) after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 237 for all subsequent segments.

> *Note:* Wireshark has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the gaia.cs.umass.edu server. Then select: *Statistics->TCP Stream Graph->Round Trip Time Graph.*

8. What is the length of each of the first six TCP segments?[3]

9. What is the minimum amount of available buffer space advertised at the received for the entire trace? Does the lack of receiver buffer space ever throttle the sender?

10. Are there any retransmitted segments in the trace file? What did you check for (in

the trace) in order to answer this question?

11. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment.

12. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

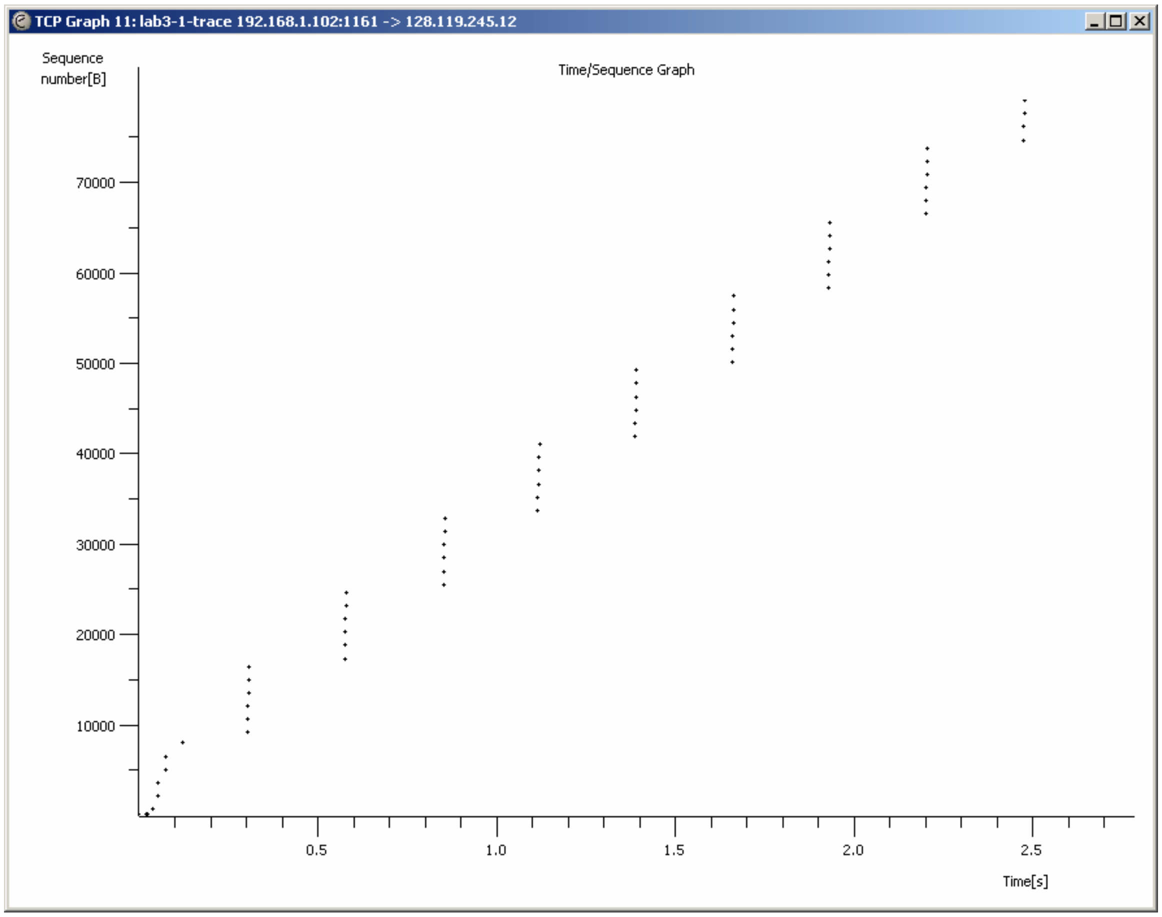
4. TCP congestion control in action

Let's now examine the amount of data sent per unit time from the client to the server. Rather than (tediously!) calculating this from the raw data in the Wireshark window, we'll use one of Wireshark's TCP graphing utilities - *Time-Sequence-Graph(Stevens*) - to plot out data.

- Select a TCP segment in the Wireshark's "listing of captured-packets" window. Then select the menu: *Statistics->TCP Stream Graph-> Time-Sequence- Graph(Stevens*). You should see a plot that looks similar to the following plot, which was created from the captured packets in the packet trace *tcp-Wireshark-*

[3] The TCP segments in the tcp-Wireshark-trace-1 trace file are all less that 1460 bytes. This is because the computer on which the trace was gathered has an Ethernet card that limits the length of the maximum IP packet to 1500 bytes (40 bytes of TCP/IP header data and 1460 bytes of TCP payload). This 1500 byte value is the standard maximum length allowed by Ethernet. If your trace indicates a TCP length greater than 1500 bytes, and your computer is using an Ethernet connection, then Wireshark is reporting the wrong TCP segment length; it will likely also show only one large TCP segment rather than multiple smaller segments. Your computer is indeed probably sending multiple smaller segments, as indicated by the ACKs it receives. This inconsistency in report edsegment lengths is due to the interaction between the Ethernet driver and the Wireshark software. We recommend that if you have this inconsistency, that you perform this lab using the provided trace file.

trace-1 in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip (see footnote 2):



Here, each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stacked above each other represents a series of packets that were sent back-to-back by the sender.
Answer the following questions for the TCP segments the packet trace *tcp-Wireshark- trace-1* in http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip

13. Use the *Time-Sequence-Graph(Stevens)* plotting tool to view the sequence number versus time plot of segments being sent from the client to the gaia.cs.umass.edu server. Can you identify where TCP's slowstart phase begins and ends, and where congestion avoidance takes over? Note that in this "real- world" trace, not everything is quite as neat and clean as in Figure in the textbook (also note that the y-axis labels for the *Time-Sequence-Graph(Stevens)* plotting tool and Figure in the textbook are different).

14. Comment on ways in which the measured data differs from the idealized behavior of TCP that we've studied in the text.

Wireshark Lab: DNS

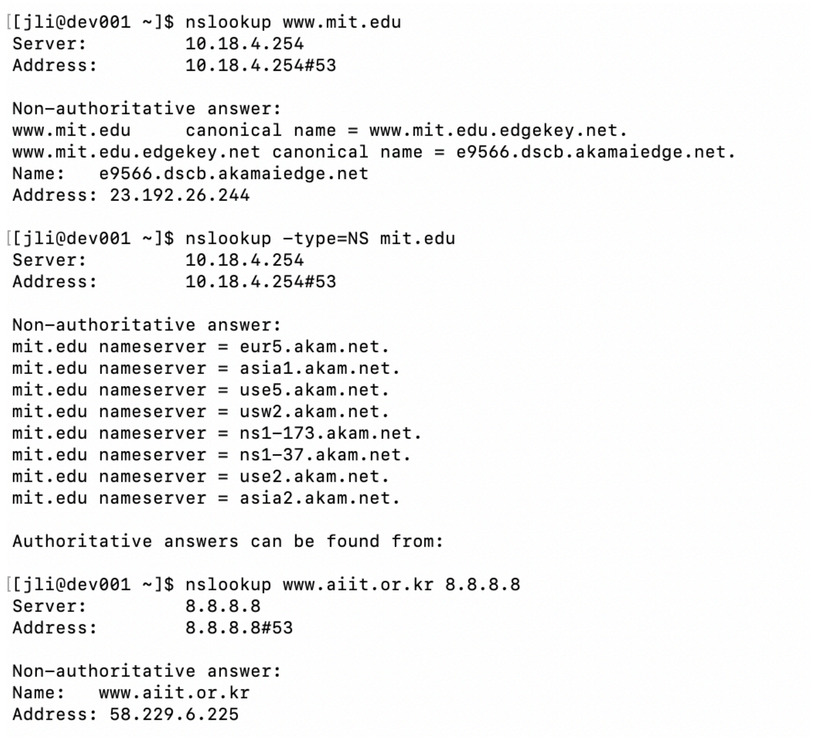
As described in Section 2.5 of the textbook, the Domain Name System (DNS) translates hostnames to IP addresses, fulfilling a critical role in the Internet infrastructure. In this lab, we'll take a closer look at the client side of DNS. Recall that the client's role in the DNS is relatively simple – a client sends a *query* to its local DNS server, and receives a *response* back. As shown in Figures 2.21 and 2.22 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 2.5 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 it is 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.



```
[[jli@dev001 ~]$ nslookup www.mit.edu
 Server:         10.18.4.254
 Address:        10.18.4.254#53

 Non-authoritative answer:
 www.mit.edu     canonical name = www.mit.edu.edgekey.net.
 www.mit.edu.edgekey.net canonical name = e9566.dscb.akamaiedge.net.
 Name:   e9566.dscb.akamaiedge.net
 Address: 23.192.26.244

[[jli@dev001 ~]$ nslookup -type=NS mit.edu
 Server:         10.18.4.254
 Address:        10.18.4.254#53

 Non-authoritative answer:
 mit.edu nameserver = eur5.akam.net.
 mit.edu nameserver = asia1.akam.net.
 mit.edu nameserver = use5.akam.net.
 mit.edu nameserver = usw2.akam.net.
 mit.edu nameserver = ns1-173.akam.net.
 mit.edu nameserver = ns1-37.akam.net.
 mit.edu nameserver = use2.akam.net.
 mit.edu nameserver = asia2.akam.net.

 Authoritative answers can be found from:

[[jli@dev001 ~]$ nslookup www.aiit.or.kr 8.8.8.8
 Server:         8.8.8.8
 Address:        8.8.8.8#53

 Non-authoritative answer:
 Name:   www.aiit.or.kr
 Address: 58.229.6.225
```

The above screenshot shows the results of three independent *nslookup* commands. In this example, the client host is located on a cluster computer of the campus of TCNJ, where the default local DNS server is `master.cm.cluster (10.18.4.254)`. When running *nslookup*, if no DNS server is specified, then *nslookup* sends the query to the default DNS server, which is `10.18.4.254`. 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 `10.18.4.254`, it is quite possible that this local DNS server iteratively contacted several other DNS servers to get the answer, as described in Section 2.5 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.

Now finally consider the third command:

nslookup www.aiit.or.kr 8.8.8.8

In this example, we indicate that we want to the query sent to the DNS server 8.8.8.8 (DNS service provided by Google) rather than to the default DNS server (`10.18.4.254`). Thus, the query and reply transaction takes place directly between our querying host and 8.8.8.8. In this example, the DNS server 8.8.8.8 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.
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.

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 Windows Command Prompt. Or

```
ifconfig
```

into Linux/Mac/Unix terminal.

ipconfig is also very useful for managing the DNS information stored in your host. 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.

=========== Mac Users=============
If you want to get the DNS server information on **Linux/Unix/Mac** system, you can type:

```
cat /etc/resolv.conf
```

For Mac OSX 10.9 user, to flush DNS, type this command:

```
dscacheutil -flushcache
```

To reload DNS, type this command:

```
sudo killall -HUP mDNSResponder
```

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[1]. Answer the following questions:

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?

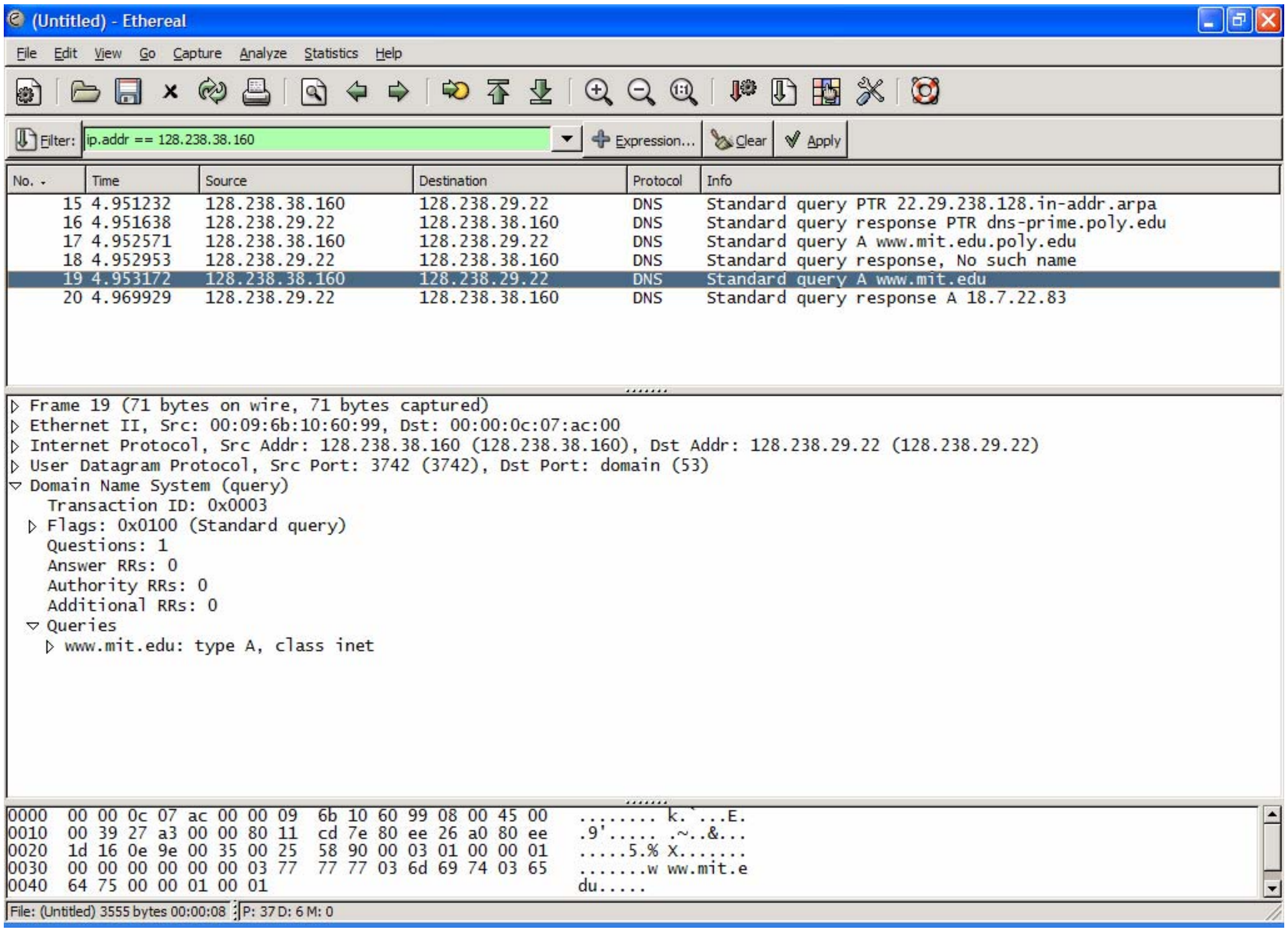
9. Consider the subsequent TCP SYN packet sent by your host. Does the destination IP address of the SYN packet correspond to any of the IP addresses provided in the DNS response message?

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

Now let's play with *nslookup*[2].

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

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



Filter: ip.addr == 128.238.38.160

```
No. -  Time       Source           Destination      Protocol  Info
   15 4.951232   128.238.38.160   128.238.29.22    DNS       Standard query PTR 22.29.238.128.in-addr.arpa
   16 4.951638   128.238.29.22    128.238.38.160   DNS       Standard query response PTR dns-prime.poly.edu
   17 4.952571   128.238.38.160   128.238.29.22    DNS       Standard query A www.mit.edu.poly.edu
   18 4.952953   128.238.29.22    128.238.38.160   DNS       Standard query response, No such name
   19 4.953172   128.238.38.160   128.238.29.22    DNS       Standard query A www.mit.edu
   20 4.969929   128.238.29.22    128.238.38.160   DNS       Standard query response A 18.7.22.83
```

```
▷ Frame 19 (71 bytes on wire, 71 bytes captured)
▷ Ethernet II, Src: 00:09:6b:10:60:99, Dst: 00:00:0c:07:ac:00
▷ Internet Protocol, Src Addr: 128.238.38.160 (128.238.38.160), Dst Addr: 128.238.29.22 (128.238.29.22)
▷ User Datagram Protocol, Src Port: 3742 (3742), Dst Port: domain (53)
▽ Domain Name System (query)
    Transaction ID: 0x0003
  ▷ Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▽ Queries
    ▷ www.mit.edu: type A, class inet
```

```
0000  00 00 0c 07 ac 00 00 09  6b 10 60 99 08 00 45 00   ........ k.`...E.
0010  00 39 27 a3 00 00 80 11  cd 7e 80 ee 26 a0 80 ee   .9'..... .~..&...
0020  1d 16 0e 9e 00 35 00 25  58 90 00 03 01 00 00 01   .....5.% X.......
0030  00 00 00 00 00 00 03 77  77 77 03 6d 69 74 03 65   .......w ww.mit.e
0040  64 75 00 00 01 00 01                               du.....
```

File: (Untitled) 3555 bytes 00:00:08 P: 37 D: 6 M: 0

[1] Download the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip and extract the file dns- Wireshark-trace-1. The traces in this zip file were collected by Wireshark running on one of the textbook 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-Wireshark-trace-1 trace file.

[2] If you are unable to run Wireshark and capture a trace file, use the trace file dns-Wireshark-trace-2 in the zip file http://gaia.cs.umass.edu/wireshark-labs/wireshark-traces.zip

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?

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

nslookup –type=NS mit.edu

Answer the following questions[3] :

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"?

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

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 namesers?

Submission

1. I need this paper back after you finish.
2. Hand in today, if you can finish. If you can not, hand in next class.