CS 4121, Fall 2017 Dr. Zhiguang Xu

GENI Lab 3

Due: October 26, 2017

0. Introduction

0.1. Overview

This lab has two parts.

The first part gives students experience generating and analyzing TCP flows. Students will use `iperf` to create a flow and view the saw-tooth behavior. A second flow will then be introduced to show how TCP flows share a link.

The second part introduces some key tools for network traffic analysis. The `nc` and `tcpdump` tools are demonstrated and key options are highlighted as students capture TCP/UDP traffic on a real link.

0.2. Objectives

Upon completing this module you will:

- Be able to use `iperf` to generate TCP traffic
- Have an understanding of how TCP utilizes and shares a link's capacity
- Be able to use `tcpdump` to capture and analyze network traffic
- Be familiar with other network analysis and traffic generation tools such as `nc`

0.3. Key Commands

`ifconfig`, `iperf`, `nc`, and `tcpdump`

1. Lab Configurations

This lab assumes you have an active and **instrumentized** slice with two connected nodes and two SSH terminals accessing such nodes respectively like what we had in the GENI lab 0/1.

You may choose to use the same slice that you used for GENI lab 0/1 previously if it is alive, i.e. "**lab0-your-initial**". If necessary, resources might have to be renewed or re-reserved though.

Or, you may choose to create a new slice (please name the new slice "**lab3-your-initial**") and request resources there.

2. Lab Details, Part I: TCP Congestion Control

2.1. Terminate any `iperf -s` process that might be already running

In order to avoid any `iperf` server process that is already running in the background, we need to search and kill them (if any) before carrying out the rest of this GENI lab.

1. In the server SSH terminal, to search for the process id (pid) of the `iperf -s` process that might be running in the background, type:

   ```
   ps -ef | grep iperf
   ```

   Suppose the pid of the `iperf -s` process is 28464, type the following to terminate it:

   ```
   kill 28464
   ```

   If there are multiple such processes, terminate them all.

2. Do the same to kill the `iperf -s` process(es), if any, on the client side.

2.2. `iperf` in one direction

`iperf` is a tool for measuring TCP and UDP bandwidth performance. In this section, we establish an `iperf` flow from the client to the server.

1. If you have not already done so, open the "*total traffic*" graphs for the client and server in your GENI Desktop by following the steps below:

   - Click "Passive Graphs"
   - At the top of the "Passive Graphs" window,
     - Click the "Graphs" button. Uncheck all the items. Check "Total Traffic".
     - Click the "Nodes" button. Check both "client" and "server".
     - Click the "ApplyConfig" button.
     - Toggle the "LiveUpdate" button and make sure it is "On".

2. In the server SSH terminal, start the `iperf` server by typing:

   ```
   iperf -s &
   ```

   The node can now receive `iperf` traffic. The ampersand `&` allows the command to run in the **background** while you use the same console to enter more commands.

3. In the client SSH terminal, start the `iperf` client by typing:

   ```
   iperf -c server -i 10 -t 180 &
   ```

   This command opens a TCP connection to the `iperf` server on the server node and begins sending packets. The "-c server" indicates that `iperf` should run in client mode and it should connect to the node with the host named "server". If you are using different hostnames, replace "server" with the hostname or IP address of your server node. The "-i 10" tells `iperf` to print updates every 10 seconds and "-t 180" indicates that the client should run for 180 seconds before closing the connection.

4. Watch the network activity graphs in GENI Desktop. You should see the traffic ramp up to a maximum and then oscillate, creating a sawtooth pattern.

2.3. `iperf` in the other direction

While `iperf` is still running, about 90 second after you started the `iperf` flow from the client to the server, form a second `iperf` flow from the server node back to the client node. In this backwards scenario, the client node will be running the `iperf` server and the server node will act as a client.

1. In the client SSH terminal, type:

   ```
   iperf -s &
   ```

2. In the server SSH terminal, type:

   ```
   iperf -c client -i 10 -t 180 &
   ```

3. Again, watch the network activity graphs in GENI Desktop. You should see the initial flow pull back as it shares the link bandwidth with the second flow.

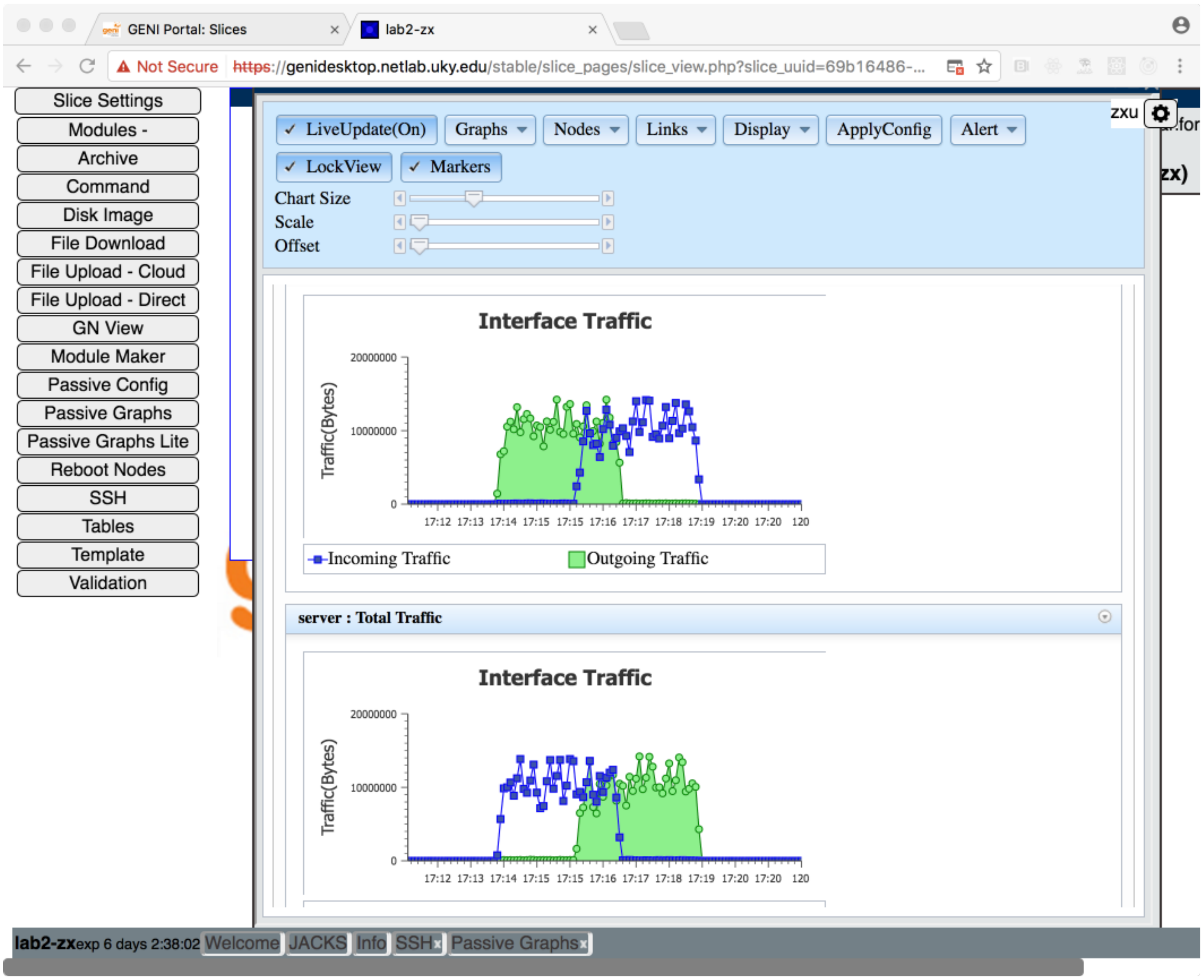
4. Once the initial flow stops, you should see the second flow grow to utilize the available bandwidth.

5. Terminate the `iperf -s` processes running on both the server and client sides. Refer to step 2.1 above.

2.3. Traffic graphs

Take a screenshot of the "*total traffic*" graphs of both the client and server in GENI Desktop and attach it to "*lab3-worksheet.docx*".

The following is a sample:



3. Lab Details, Part II: TCP/UDP Traffic Analysis

3.1. Introduction to `tcpdump`

`tcpdump` is an extremely valuable tool for network analysis. It is able to capture and display information on packets entering and/or leaving a network interface. The following steps will introduce you to some of the features of `tcpdump` and give you hands-on experience with it.

1. On the client SSH terminal, issue the command:

   ```
   sudo tcpdump -i <interface name> -nn
   ```

   where "-i " is the name of the interface on which to listen. This will be the same as the interface name used in the previous section (i.e. "eth1"). The "-nn" instructs tcpdump to use the numerical IP addresses rather than using DNS to resolve the symbolic addresses. It also instructs tcpdump to use the numerical ports rather than substituting them with commonly used protocol strings. You should see a few lines of text appear. Tcpdump is now ready to print information on network packets which enter or leave through the given interface.

2. Now, switch to the server SSH terminal and ping the client by typing:

   ```
   ping client
   ```

   You should see something like the following printed in the client terminal for each ping:

   ```
   10:58:37.662843 IP 10.10.1.1 > 10.10.1.2: ICMP echo request, id 7748, seq 3, leng
   th 64
   10:58:37.762902 IP 10.10.1.2 > 10.10.1.1: ICMP echo reply, id 7748, seq 3, length
    64
   ```

   Each line of the output corresponds to a packet that has been sent or received on the specified interface. The first column contains the timestamp for the event, with microsecond precision. Next, the protocol is listed (i.e. ARP or IP). For IP packets, the source and destination IP addresses are listed, followed by the layer 3 control plane protocol name and associated information. In this case, the attributes of the ICMP packets used by the ping tool are shown. Notice that an echo request is sent from 10.10.1.1 to 10.10.1.2 and shortly thereafter an echo reply is sent in the opposite direction.

3. Type Ctrl-c in the server SSH terminal to stop pinging.

4. Type Ctrl-c in the client SSH terminal to stop `tcpdump`.

5. Restart `tcpdump`, but add "-S" to the options and "tcp" at the end as in:

   ```
   sudo tcpdump -i <interface name> -nn -S tcp
   ```

   The "-S" option tells `tcpdump` to print absolute TCP sequence numbers rather than converting them to small sequence numbers that are easier for humans to read. This will allow us to see the raw sequence numbers in our initial analysis. "tcp" adds a filter to tcpdump such that only TCP traffic will be printed to the screen.

6. In the server SSH terminal, type:

```
nc -l 44444
```

   This command uses the "netcat" tool to start listening on the given port for a connection.

7. In GENI Desktop, open **another** SSH terminal to the client. (Make sure client is selected and click SSH then Open SSH. We need this second client terminal because the first one is still running tcpdump.)

8. In this **new** client SSH terminal, type:

```
nc server 44444
```

   This command establishes a TCP connection with the listener on the server. Notice that tcpdump has now printed something like the following:

```
zxu@client:~$ sudo tcpdump -i eth1 -nn -S tcp

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes

15:48:29.783583 IP 10.10.1.2.42294 > 10.10.1.1.44444: Flags [S], seq 2702194805,
win 29200, options [mss 1460,sackOK
,TS val 192075922 ecr 0,nop,wscale 7], length 0

15:48:29.784088 IP 10.10.1.1.44444 > 10.10.1.2.42294: Flags [S.], seq 164913660,
ack 2702194806, win 28960, options
[mss 1460,sackOK,TS val 192080712 ecr 192075922,nop,wscale 7], length 0

15:48:29.784132 IP 10.10.1.2.42294 > 10.10.1.1.44444: Flags [.], ack 164913661, w
in 229, options [nop,nop,TS val 192
075922 ecr 192080712], length 0
```

   As before, the timestamp, protocol, and IP addresses are present, but an extra number after the IP addresses indicates the port number. In this case, a packet was sent from port 42294 on 10.1.1.2 to the port 44444 of 10.1.1.1. The remainder of the each line (after the ":") gives the details of the TCP packet. The TCP flags set, if any, are indicated within the square brackets ([]) after the word Flags. The S indicates a SYN packet and a period (".") indicates an ack. The rest of the potential attributes include "seq" for the packet's sequence number, "ack" for the acknowledgement number, "win" for the source's TCP window size, "length" for payload/data length, and several optional fields after "options".

What was just printed by tcpdump is the 3-way TCP handshake that initiated the connection. Notice that the payload "length" of each of the three messages is 0. No application data has been transferred, but a connection has been established. First, the client (10.1.1.2) sent a SYN (indicated by [S]) to the server (10.1.1.1), communicating its initial client-side sequence number of 2702194805. Then, the server (10.1.1.1) replied with a SYN/ACK (indicated by [S.]), giving its initial server-side sequence number of 164913660, and acknowledging that all data before 2702194806 (one more than the client-side sequence number) has been received. Finally, an ACK is sent from the client (10.1.1.2) to the server (10.1.1.1) to complete the 3-way handshake, acknowledging that all data before 164913661 (one more than the server-side sequence number) has been received. The TCP connection is now ready to be used.

9. In the server SSH terminal, type:

```
Hello World
```

and press "Enter". The `nc` (netcat) tool packages this input into a TCP message and sends it to the server, where it is unpackaged and displayed. Check tcpdump again, and you should see something like the following:

```
16:00:30.078745 IP 10.10.1.1.44444 > 10.10.1.2.42294: Flags [P.], seq 164913661:1
64913673, ack 2702194806, win 227,
options [nop,nop,TS val 192260793 ecr 192075922], length 12

16:00:30.078826 IP 10.10.1.2.42294 > 10.10.1.1.44444: Flags [.], ack 164913673, w
in 229, options [nop,nop,TS val 192
255996 ecr 192260793], length 0
```

Notice that a TCP message with 12 bytes of data (length 12) was sent from the server (10.1.1.1) to the client (10.1.1.2). This is one byte for each of the 11 characters in your message plus the carriage return. Also, notice that the sequence number is now 164913661:164913673, indicating the range of sequence numbers for the data being sent. The "P" flag may be present. It stands for "push" and tells TCP on the receiving side of the connection to immediately push all buffered data to the application. Finally, the client responds with an ACK, acknowledging the receipt of the data.

10. Now, send a message in the opposite direction by typing it in the client SSH terminal and pressing "Enter". Verify that the sequence number and ack fields of the TCP messages printed by tcpdump increment as expected.

11. Type "Ctrl-c" to quit out of both `nc` sessions and `tcpdump`.

3.2. More `tcpdump` options

This section will describe some more useful features of `tcpdump` ; however, for a full treatment, you are encouraged to find a good online tutorial.

1. `tcpdump` is able to write what it captures to disk rather than just displaying it on the screen. However, care should be taken to honor privacy when using tcpdump. It is customary to only capture a certain number of bytes of each packet's header. This is done by setting a snap length of N. Only the data in the first N bytes is recorded by tcpdump. Restart tcpdump in the client SSH terminal, but use the "-w" option to write to a file and add a snaplength of 96 bytes with the "-s" option:

   ```
   sudo tcpdump -i <interface name> -nn -S tcp -s 96 -w <filename.pcap>
   ```

   The format of the resulting file will be the pcap (for packet capture) format. It is customary to give these files the extension ".pcap".

2. In the server SSH terminal, check if there is an `iperf` server running in the background already. If so, kill it. Then start the iperf server by typing:

   ```
   iperf -s
   ```

   The node can now receive iperf traffic.

3. In a second client SSH terminal, start the `iperf` client by typing:

   ```
   iperf -c server -t 1
   ```

4. After 1 second, the client should have completed sending its traffic. Type "Ctrl-c" in the `tcpdump` window and the `iperf` server window to stop `tcpdump` and the `iperf` server.

5. `tcpdump` can display and filter previously captured pcap files. To display the first 10 packets of the file just captured, use the count ("-c") and the read ("-r") options, typing:

   ```
   tcpdump -nn -r <filename.pcap> -c 10
   ```

   Notice we no longer need sudo because we are reading the packets from a file.

6. Now, display only packets whose source is the server (with IP address 10.10.1.1). To do so, we add an "and" and a "src" attribute to the filter string, like this:

   ```
   tcpdump -nn -r <filename.pcap> -c 10 tcp and src 10.10.1.1
   ```

This will display only the first 10 tcp packets whose source IP address is 10.10.1.1. In this way you can build up complex filters for your traffic. Many other filtering commands are available, but their description are outside the scope of this lab.

7. Take a screenshot of the the client SSH terminal and attach it to "*lab3-worksheet.docx*". **Make sure it includes the `tcpdump` command and its outputs listed in step 6 above.**

4. What to Turn in?

Submit the following file:

- lab3-worksheet.docx