

Problem Set 2

Instructor: Prof. Bruce Maggs

Computer Science Department, Duke University

This problem set has three questions. Answer them as clearly and concisely as possible. You may discuss ideas with others in the class, but your solutions and presentation must be your own. Do not look at anyone else's solutions or copy them from anywhere. (Please refer to the Duke University honor code).

Turn in your solutions in on **March 3, 2007** by 11:59 p.m.

1 Netflow

In this problem, you'll analyze Netflow flow records captured over a 5-minute window from Abilene's (Internet2) router in New York, NY on December 31, 2005 at 11:55pm GMT.

1. First, download the Netflow flow records here from here: <http://www.cs.duke.edu/courses/spring08/cps214/hw/ps2/ft-v05.2005-12-31.235500+0000.gz>.

Next, download and compile the `flow-tools` package, which you'll need to parse the flow records (which is in a gzipped binary format). To get a flavor for the flow records, try, for example, the following commands:

```
% zcat ft-v05.2005-12-31.235500+0000 | flow-cat | flow-export -f2 | head
#:unix_secs,unix_nsecs,sysuptime,exaddr,dpkts,doctets,first,last,engine_type,
engine_id,srcaddr,dstaddr,nexthop,input,output,srcport,dstport,prot,tos,
tcp_flags,src_mask,dst_mask,src_as,dst_as
1136073300,0,105262938,198.32.10.249,2,1725,105196926,105206958,0,0,
128.103.184.0,222.28.80.0,198.32.8.82,66,88,2077,4670,6,0,24,16,12,1742,4538
1136073300,0,105262938,198.32.10.249,2,2600,105205566,105218421,0,0,
218.196.192.0,134.155.136.0,198.32.11.51,88,56,3268,1057,6,0,16,13,16,4538,553
1136073300,0,105262938,198.32.10.249,1,44,105193056,105193056,0,0,
218.196.192.0,134.155.136.0,198.32.11.51,88,56,3268,1057,6,0,24,13,16,4538,553
...
```

which outputs most of the interesting bits of the flow records in a script-friendly format. Note that, to preserve anonymity, Abilene “zeros” the bottom ten bits of every IP address.

To get a slightly more human-readable (but less script-friendly) summary, try the following:

```
% zcat ft-v05.2005-12-31.235501+0000 |
/usr/local/netflow/bin/flow-cat
| /usr/local/netflow/bin/flow-print | head
srcIP          dstIP          prot  srcPort  dstPort  octets  packets
138.87.216.0   130.49.8.0     6     3591     6346    80      2
130.132.216.0  130.14.24.0    6     29393    80      52      1
```

2. Provide a distribution of the fraction of bytes and packets by destination port number. What is the most common port in this traffic?
3. What is the average length of flows of each type, in packets and in bytes? What artifacts of Netflow might cause this estimate to be inaccurate?
4. What are the top 2 /24s (and the organizations they correspond to) to which Duke University sends traffic (bytes) out of Abilene via New York? (Hint: You will have to create a filter to match a /24 prefix and Duke network address).
5. What are the top 2 /24s (and the organizations they correspond to) that send traffic (bytes) to Duke University via the Abilene router in New York? (Again, use prefix match to figure this out.)

You can use any of the the flow-tools available with Netflow (also you may use sort or other Linux commands). Provide the commands you used to answer the previous questions.

2 A Few of My Favorite Pings

Scriptroute is a network measurement tool that allows users to easily write network measurement scripts and, optionally, to perform distributed measurement by invoking these measurements on remote servers. Developing customized probing mechanisms with scriptroute is relatively easy in comparison to hacking one's own socket code.

1. Download and install scriptroute.<http://www.scriptroute.org/source/>.

Note: check the appendix for installation instructions

2. We will modify `sr-ping` so that we can measure packet loss rates under arbitrary conditions (*e.g.*, , different sending rates, etc.), and observe other fields in the IP header.
 - (a) Modify `sr-ping` so that it takes:
 - a `-r --rate` option, in packets per second, and
 - a `-n --number` option, which specifies the number of packets
 - (b) Also modify `sr-ping` so that it prints out the ICMP echo reply in the output. For example:

```
% sr-ping www.cs.duke.edu
1 152.3.140.5 11.656 ms [id=11223]
2 152.3.140.5 9.802 ms [id=11224]
3 152.3.140.5 10.341 ms [id=11225]
4 152.3.140.5 9.562 ms [id=11226]
...
```

- (c) Execute `./sr-ping -n 10 -r 0.2 www.cs.duke.edu`. This should send out an ICMP echo request about once every five seconds. Include the output of this program. How is the IP ID field being incremented?

Now, repeat the above command, but *also* run a normal ping simultaneously to the same machine. What's going on?

- (d) Now modify `sr-ping` to send TCP syn probes to arbitrary ports; your code should use standard ICMP pings unless the `-p --port` option is specified at the command line.
- (e) Modify `sr-ping` so that it keeps track of some of the same statistics as a normal ping program:
- minimum, average, maximum round trip times; and
 - average loss rates
3. Find a server that will respond to pings both via ICMP and port TCP SYNs that is at least 60 milliseconds away.¹
- Take measurements of minimum, maximum, and average round-trip times for 1,000 probes, sent at a rate of two pings per second. Also, plot the CDF of round-trip times for both the TCP ping and the ICMP ping. Also take note of the maximum RTT values for each. Finally report the number of packets lost.
- What conclusions, if any, can you draw from your data? (If your data is inconclusive, what conclusions might you expect to draw?)
4. In this sub-problem, we'll look at some more benefits of TCP-based probing over ICMP-based probing.
- (a) Try pinging `www.ebay.com`. What happens?
- (b) Now try the same experiment, but with `--port 8000`. What's happening? Give two reasons why TCP-based probing is often more effective and accurate than ICMP-based probing.
5. Extend your ping program to "round robin" through a sequence of IP addresses. For example, your output should look something like:

```
% sr-ping www.cs.duke.edu www.cc.gatech.edu www.eecs.mit.edu
1 152.3.140.5 9.258 ms [id=11270]
1 130.207.7.80 119.373 ms [id=3332]
1 18.62.0.96 29.021 ms [id=65364]
2 152.3.140.5 10.909 ms [id=11271]
2 130.207.7.80 120.598 ms [id=40726]
2 18.62.0.96 25.560 ms [id=65366]
...
```

Don't forget to include your `sr-ping` source code with your answers.

3 Simulation vs. Emulation

This problem will give you experience running performance tests in simulation and emulation. Really, it is an excuse to get you familiar with the *ns* network simulator, and with Emulab. Both experimental setups use the same configuration language (based on Tcl), so you will be able to reuse much of your code for both the simulation and emulation parts of this problem.

¹Take care with this measurement, particularly the TCP ping measurements, which might look like a SYN flood if you send them at too high of a rate. I intentionally did not provide you a server because

3.1 Setup

ns Download and install the network simulator *ns*. Download ns-allinone from the ns-2 homepage. This version of ns contains all the components you need, including the Tcl interpreter. Visit <http://www.isi.edu/nsnam/ns/ns-build.html#allinone> for instructions to install ns.

Once you have installed em ns, you can run any simulation by typing `ns sim.ns`, where `sim.ns` is the Tcl script that specifies your simulation.

Emulab We'll let you know how to get an Emulab account.

3.2 Experiments

1. Write a tcl procedure that takes an integer n and creates a dumbbell topology. You can test your code on emulab or in *ns*. Make all “edge” links in the topology 100Mbps/10ms duplex links, with DropTail queues, and the link between R_1 and R_2 a 1Mbps/10ms duplex link. *Hint:* Reading Emulab and ns documentation will help with this problem.
2. Create a four-node topology (a dumbbell topology with $n = 1$) using the same parameters as above. Create a *TCP Reno* session with FTP traffic between the source and the sink.

To record the output of your experiment, you will have to use *ns*'s tracing facility. Here is an example code snippet to help:

```
#Create a simulator object
set ns [new Simulator]

#Open the Trace file
set tf [open out.tr w]
$ns trace-all $tf

#Define a 'finish' procedure
proc finish {} {
    global ns tf
    $ns flush-trace
    #Close the Trace file
    close $tf
    exit 0
}
```

3. Use the output trace `out.tr` (1) to plot the evolution of sequence numbers over time and (2) to compute the “goodput” (i.e., the amount of the file successfully transferred) of the link in steady state.
4. Repeat the experiment, but install a 5% packet loss rate on the link between R_1 and R_2 .

5. Create the same four-node topology in Emulab, and run the same experiment on the Emulab testbed. For this experiment, use the `iperf` binary (<http://sourceforge.net/projects/iperf>) to create a traffic source and sink, and `tcpdump` to record the packet trace.
 - (a) You could record packet traces at either the sender or the receiver. Explain how, for each case, you would compute the goodput differently.
 - (b) Plot the sequence number evolution and compute the goodput of the transmission in the Emulab setup with the same CBR traffic stream (1kbps, 100-byte packets), for both no packet loss and 5% packet loss.

Do your results differ from the same setup in ns? List two reasons why you might expect your results to differ.

A Scriptroute Installation Notes

Note: I had some problems installing this code.

Most of the dependencies required by scriptroute were already installed on my Linux box. However, the `libpcap` and `tcpdump` versions were much newer than those indicated in the scriptroute installation notes and `libpcap` was causing some conflicts during the scriptroute build process. The quickest fix to this is to uninstall the version you already have and let scriptroute install the version it needs. You should be able to reinstall `libpcap` later with your favorite package management software. Read the `INSTALL` and `README` notes that come with the scriptroute sources. I am running Ruby version 1.8 and it works fine with scriptroute. You will need root privileges to install scriptroute.

You may have to take the following steps.

First, you may have to manually create `/var/lib/scriptroute/web/var/log/scriptroute`, or wherever you've configured scriptroute to install its log directory. Mine did not build automatically.

Second, you may have to manually create the file: `/etc/scriptroute/ratelimits_whitelist` (as well as its parent directory). Again, the scriptroute install script did not create this file or directory for me.

Third, you will need to start up the local scriptroute server on the local machine. I did this with the following command: `/usr/local/sbin/scriptrouted`
`--config=/usr/local/etc/scriptrouted.conf`, although you may have to do this differently if you do not have root access on your local machine.