Revised Problem Set 2

Due Thursday, May 6

This problem set will give you some hands-on experience with the Click modular router. We won't run Click in the kernel, though; instead, we'll run it at user level, on trace files. You will need access to a Unix machine (Linux, BSD, Sun, Mac OS X).

You'll turn in most of this problem set electronically. A tarball (`readable-ps2.tar.gz`) will be available on the course Web site. You should download and unpack this tarball and do your work inside the resulting `readable-ps2` directory. When you're done (by 5pm Thursday), package up the tarball and send me the result. You should also include a short writeup, either in the tarball in a file named `PS2` or, if you prefer, on paper (no staples please). One page will be fine.

You should be able to automatically check your work for some of the problems by running `make check`.

If bugs are found in the problem set, I'll distribute new tarballs. Therefore, *do not change files that came with the problem set*. Just add new files of your own.

Collaboration. Feel free to discuss the problem set with other students and on the mailing list, but do all the coding yourself. Include the names of your collaborators in your writeup.

0. Preparation. (No writeup necessary.)

Obtain Click from anonymous CVS (http://www.pdos.lcs.mit.edu/click/anoncvs.html) then compile and install it. If you have any problems, let me know immediately.

A quick rundown of the commands you'll use:

```
$ cd ~
$ cvs -d :pserver:anoncvs@cvs.pdos.lcs.mit.edu:/cvs login
CVS password: [press return]
$ cvs -z5 -d :pserver:anoncvs@cvs.pdos.lcs.mit.edu:/cvs co -d click click/release/one
...
$ cd click
$ ./configure --prefix=$HOME --enable-analysis --enable-test
     # '--prefix=$HOME' means 'install into my home directory'.
     # You'll need the elements included by --enable-analysis and --enable-test.
...
$ make install
$ ~/bin/click ~/click/conf/test.click
ok:   40 | 45000028 00000000 401177c3 01000001 02000002 13691369
ok:   40 | 45000028 00000000 401177c3 01000001 02000002 13691369
ok:   40 | 45000028 00000000 401177c3 01000001 02000002 13691369
ok:   40 | 45000028 00000000 401177c3 01000001 02000002 13691369
ok:   40 | 45000028 00000000 401177c3 01000001 02000002 13691369
$
```

The "`make install`" step installs a number of useful files, including manual pages for each of Click's element classes. Wherever you install Click, you'll probably want to alter your `PATH` and `MANPATH` variables accordingly. If you install into your home directory (recommended), you'll change `PATH` to include `$HOME/bin` and `MANPATH` to include `$HOME/man`.

Element class descriptions and other Click manual pages are also available on the Web at http://www.pdos.lcs.mit.edu/click/doc/

If you get a bus error when running Click on a Solaris or other non-x86 machine, try running your configuration through `click-align`: "`click-align` *`config`*`.click | click`".

1. Trace files [**2 points**]. Create a router configuration file called `prob1.click` that reads packets from the `tcpdump` trace file '`f1a.dump`' and writes those packets unchanged to the `tcpdump` trace file '`f1b.dump`'. The Click driver should exit when `f1a.dump` is out of packets.

Notes and hints: `tcpdump` is the canonical program for reading and storing packet traces. It can store packets read from Ethernets, ATM networks, and many other types of links. A `tcpdump` file's link type is called its *encapsulation*. We will be dealing only with *raw IP encapsulation*, where the files have no link-level headers: the first bytes in each packet will be an IPv4 header.

2. Routing [**2 points**]. Create a router configuration file called `prob2.click` that reads packets from the `tcpdump` trace file '`f2a.dump`' and *routes* those packets using the following routing table:

Destination prefix	**Output trace file**
131.0.0.0/8	`f2b.dump`
131.179.0.0/16	`f2c.dump`
18.0.0.0/8	`f2d.dump`
All others	`f2e.dump`

3. Error checking [**3 points**]. Create a router configuration file called `prob3.click`, based on `prob2.click`. The configuration should read from '`f3a.dump`' and write to '`f3*.dump`'. But this time, you should check for the following six kinds of errors, in this order.

Problem	**Action**
Invalid IP header or checksum	Discard packet
Invalid TCP header or checksum	Discard packet
Invalid UDP header or checksum	Discard packet
Invalid ICMP header or checksum	Discard packet
Expired IP TTL	Generate appropriate ICMP error message, send message to '`f3f.dump`'
Packet longer than 1500 bytes	Discard packet

4. Handlers [**2 points**]. Create a router configuration file called `prob4.click`, based on `prob3.click`. It should read from '`f4a.dump`' and write to '`f4*.dump`'. But this time, in addition to checking for errors, running the configuration should generate a file '`f4.drops`' containing the following 6 lines:

1. The number of packets with invalid IP headers or checksums;
2. The number of packets with invalid TCP headers or checksums;
3. The number of packets with invalid UDP headers or checksums;
4. The number of packets with invalid ICMP headers or checksums;
5. The number of packets with expired IP TTLs; and
6. The number of packets longer than 1500 bytes.

Each packet will show up in at most one of these counts.

Hint: Check out *DriverManager*.

5. Compound elements [**3 points**]. Create a partial router configuration file called `prob5.click`. This file should define a compound element named '`ErrorChecker`' with one input and one

output that implements all the error checks from Problem 3. All erroneous packets should be dropped (not written to a file).

6. Compound element overloading [**3 points**]. Create a partial router configuration file called `prob6.click`, based on `prob5.click`. This time, the `ErrorChecker` compound element should support three different use patterns. If it is used with one input and one output, it should behave like in Problem 5. If it is used with one input and two outputs, then all erroneous packets should be emitted on the second output (not dropped). If it is used with one input and seven outputs, then the first output is for correct packets, and the following six outputs are used for the six different errors.

7. Scheduling [**5 points**]. Create a router configuration file called `prob7.click`. It should read packets from '`f7a.dump`', *maintaining the timing of the dump file*. (This means that if two adjacent packets in the dump file have timestamps that are 0.5 seconds apart, the packets will be emitted into the configuration 0.5 seconds apart.) The packets should be written into '`f7b.dump`', except that:

- Packets should be written to the dump at a maximum rate of 384 Kb/s. Since the input rate might be more than 384 Kb/s, this means you will need *queuing* and *traffic shaping* in your configuration.
- TCP should always get up to 75% of the bandwidth (up to 288 Kb/s). This means that if the TCP input rate is 288 Kb/s and the UDP input rate is also 288 Kb/s, TCP's output rate will equal 288 Kb/s and UDP's output rate will be limited to 96 Kb/s. If the TCP input rate is less than 288 Kb/s, UDP and other kinds of traffic can take up the remainder.

Packet timestamps in the output file should reflect the 384 Kb/s rate limit.

8. More error handling [**2 points**]. Add error handling à la Problem 4 to the configuration of Problem 7. The router configuration file should be called `prob8.click`, it should read from '`f8a.dump`' and write to '`f8b.dump`', it should discard IP TTL-expired packets rather than generate errors, and '`f8.drops`' should report any packets dropped from queues after the errors already mentioned.

9. Trace analysis [**6 points**]. Write an executable script called `prob9` that takes a single command line argument, a trace file; reads that trace file; and writes the following information about the trace to the standard output:

1. Total number of packets.
2. Average packet size (rounded down to the nearest integer).
3. Average TCP packet size (rounded down to the nearest integer).
4. Average UDP packet size (rounded down to the nearest integer).

5–261. Number of bytes sent to each of the 256 8-bit prefixes.
(For example, packets sent to any IP address "169.x.y.z" should be counted towards the 8-bit prefix "169".)
Don't print lines where the count is zero.

262–518. Number of bytes sent *by* each of the 256 8-bit subnets.

Use an output format like this (don't include the parenthetical comments):

```
10          (Packet count)
765         (Average packet size)
1500        (Average TCP packet size)
```

```
30                  (Average UDP packet size)
dst 169 7650        (Prefix 169 received 7650 bytes)
src 18 7500         (Prefix 18 sent 7500 bytes)
src 19 150          (Prefix 19 sent 150 bytes)
```

10. Element implementation [EXTRA CREDIT up to 6 points]. Create an element of your own. For example, implement a scheduler, such as a Smoothed Round Robin scheduler (Guo Chuanxiong, "SRR: An $O(1)$ Time Complexity Packet Scheduler for Flows in Multi-Service Packet Networks", *Proc. SIGCOMM 2001*); or an active queue management scheme, such as FPQ (Robert Morris, "Scalable TCP Congestion Control", *Proc. INFOCOM 2000*); or a packet source, such as an element that reads a file, then sends 1500-byte TCP packets containing that file's contents as if TCP had sent them; or a packet sink, such as an element that sends the corresponding TCP acknowledgement for each TCP packet received. (The TCP packet source and sink examples can be made pretty interesting if you implement more-or-less-correct TCP behavior.) Be creative!