Assignment 1: Getting to know Click and D-ITG
Due 9/20/2015 11:59pm (i.e., Sunday night)

In this assignment, you will be using Click and D-ITG.

Background: Click

Click is a framework for experimenting with packet processing (data plane).

Elements – in click, an application is broken up into individual reusable building blocks which perform one stage of a packet processing pipeline. These are called elements. Click comes with many elements (see http://) and you can add your own by writing one in C++. In Part 1, you will only use elements that come with click, in part 2, you’ll extend that to write your own.

Click graph – to describe an entire packet processing pipeline, you describe in a special language the series of elements a packet is processed with. As an example:

```c
// instantiate elements. Put arguments in a comma separated list
// note: you can instantiate the element when describing the pipeline
// (e.g., fd0::FromDevice(eth0)->...)
Fd0::FromDevice(eth0)
Fd1::FromDevice(eth1)
// RandomSwitch does not have arguments
Sw0::RandomSwitch
q0::Queue
q1::Queue
// describe pipeline
// Fd0 -> [0] Sw0 means means connect Fd0 to input port 0 of Sw0.
// Sw0 [0] -> q0 means connect output port 0 to q0.
// q0 -> td0 means connect output of q0 to td0
// note, you can leave off the port number if the element
// only has 1 input or output port
Fd0 -> [0] Sw0 [0] -> q0 -> td0;
Fd1 -> [1] Sw0 [1] -> q1 -> td1;
```

Push/Pull – you’ll notice that the output of the switch was put in a queue. Some ports are push, meaning the upstream element pushes the packet to the downstream element whenever the upstream element has completed its processing. Some ports are pull, meaning the downstream element pulls the packet when the downstream element is ready to process a packet. Some ports are agnostic, meaning they will act as either push or pull, depending on what they’re connected to. Since the ports of ToDevice are pull and the ports of FromDevice and RandomSwitch are push, we need a Queue which has
a push input and a pull output.

Running – once you create a click graph saved in a file (say, myexp.click), you run with:
```
sudo click myexp.click
```

Notes:

1. Add the install/bin to your path and pass it in to sudo, or fully specify the click executable when running (e.g., `sudo ~/click-2.0.1/install/bin/click myexp.click`)
2. Look in `click-2.0.1/conf` for examples.
3. For testing, you can use elements such as InfiniteSource, TimedSource, etc. See `conf/test.click` for an example. You probably don’t need it, but for reference.

**Preparation: Install Click**

Download click to your home directory in emulab. Do this from a testbed machine (not users.emulab.net). Use the 2.0.1 release, so we’re all working from the same code.
```plaintext
http://www.read.cs.ucla.edu/click/click-2.0.1.tar.gz
```

To compile:
```
cd ~/click-2.0.1
mkdir install
./configure
--enable-local
--disable-linuxmodule
--prefix=$HOME/click-2.0.1/install
make install
```
(this will take a few minutes)

Note: you can use configure --help to see the options.

Another tool we’ll use is D-ITG. I have installed D-ITG in `/proj/ColoradoAdvNet15/tools/D-ITG-2.8.0-rc1/bin`.

Add both the click bin directory and the D-ITG bin directory to your path.
```
http://www.read.cs.ucla.edu/click/  (the Element’s page is especially useful)
http://traffic.comics.unina.it/software/ITG/
```

**Preparation: Emulab**

In the file a1.ns, I created a 3 node topology: `n0 <-> n1 <-> n2`. Use this for your Emulab experiment.
**Part 1: Bump in the wire**

In this, we’ll be creating a ‘bump-in-the-wire’ device, meaning it will not make any modifications to packets, but simply passes through, unmodified, any packets received.

In the file `bump.click`, I created an example that’s a bit more elaborate than just passing packets through (it’ll help with the next part). IMPORTANT: the device names may change for your experiment, and between swap-ins. Use `ifconfig` to see the ip addresses of each Ethernet device. Look at the ip addresses on n0, and compare – e.g., if n0’s eth1 is 10.1.2.2 and n1’s eth2 is 10.1.2.3, then eth2 is connected to n0 (same /24 subnet).

You’ll note that for the FromDevice, I set the parameters **SNIFFER false** and **PROMISC true**. **SNIFFER false** tells the element to not have Linux also process the packet (e.g., **SNIFFER true** would mean that the click application simply is sniffing the packets, but Linux is processing them). **PROMISC true** is important because we want the network card to pass all Ethernet frames to us, not just ones destined to us – since we want to act as a bump in the wire.

The “bit more elaborate” part is that I threw in a couple classifier elements. c0 will divide traffic – IP traffic out of port 0, other traffic (such as ARP) out of port 1. c1 divides the IP traffic further, by TCP port. All of c1’s outputs feed into a single queue. There are elements to print stuff out, and there is a BandwidthShaper which limits traffic to 1,000,000 Bytes / second (or 8Mbps).

To actually get this to work, we’ll need to edit the routing tables of n0 and n2 – to use a /16 instead of a /24. Some sample from my run

```bash
n0:~/click/click-2.0.1/conf> ip route
10.1.2.0/24 dev eth1  proto kernel  scope link  src 10.1.2.2
155.98.36.0/22 dev eth0  proto kernel  scope link  src 155.98.36.156
default via 155.98.36.1 dev eth0
n0:~/click/click-2.0.1/conf> sudo ip route del 10.1.2.0/24
n0:~/click/click-2.0.1/conf> sudo ip route add 10.1.0.0/16 dev eth1 src 10.1.2.2

n2:~/a1> ip route
10.1.1.0/24 dev eth1  proto kernel  scope link  src 10.1.1.3
155.98.36.0/22 dev eth0  proto kernel  scope link  src 155.98.36.121
default via 155.98.36.1 dev eth0
n2:~/a1> sudo ip route del 10.1.1.0/24
n2:~/a1> sudo ip route add 10.1.0.0/16 dev eth1 src 10.1.1.3
```
Run click, perform some pings, etc.

To hand in: A drawing for what bump.click is.

Part 2: Queuing

In this part, we’ll play around with queues and prioritizing traffic and see what effect that has. For example, we might consider VoIP traffic more critical, so we give it a higher priority Queue. There are all kinds of papers on this, but we’ll look at two simple examples – a strict priority, and a round robin.

Step 2A: In part 1, we fed each of the IPClassifier’s outputs to a single Queue. For this, modify them to feed into separate queues (feed into one queue both the non-IP traffic, port 1 of c0, and traffic that is not tcp dest 9501, 9502, or 9503, i.e., port 3 of c1). Then use a scheduler element (create one click graph with PrioSched and one with RoundRobinSched) – the non-IP and non-port 9501-9503 traffic should be highest priority (input port 0 of the scheduler). Now, extend the click graph with the RoundRobinSched to rate limit traffic to tcp dest 9503 by using BandwidthShaper(10000) (note: there will still be the other BandwidthShaper element rate limiting all traffic). Delete all of the print elements (Print and IPPrint), as they will slow down processing.

We’ll now run traffic through, varying the rate at which we send. Think about what you expect to happen in each case. I included four files: itg-script1, itg-script2, and itg-script3, itg-script4.

D-ITG is a tool to perform network measurements. You run a Sender on one machine, a receiver on another. D-ITG tracks every packet, and you can calculate the delay, throughput, and loss.

For each experiment and each script (so a total of 12 times), run the following from your home directory or subdirectory, which is a networked file system, so the files are all accessible on all machines (note the n2 or n0 before the command says what machine to run it on – n1 is running click):

```
n2: ITGRecv
n0: ITGSend <script> -l send.log -x recv.log
(once complete, ctrl-c on n2 to close out of ITGRecv)
n0: ITGDec recv.log
```

We’re interested in the lines that say “Average bitrate” (one for each flow and one for the total, don’t worry about the total) – this is the actual throughput. Note, you can run ITGSend several times, specifying a different recv.log – then you can go back and process each of them afterwards with ITGDec.

To hand in: 1) the three click files, 2) 3 plots with y-axis (bit rate), x-axis (script number, which is an increase in send rate) – one line for each flow, one plot for each click file used.

Step 2B: Next, we’ll look at the effect a large queue can have. Modify bump.click and change Queue q0’s size to be 10000 (meaning it can hold 10,000 packets) – again, remove the Print and IPPrint elements. We’re now going to send UDP traffic at a rate faster than the “bump” can handle – so, greater than 8Mbps, which is what our BandwidthShaper limits to). Use itg-script5 this time. Run
“ITGDec recv.log -d 500 delay.log”. This will print out the average delay for each 500ms interval. Note that it’s increasing. Huge buffers are not a good idea.

To hand in: the output of ITGDec (delay.log).

Part 3: RST Injection

In this part, we won’t rate limit the traffic, but we’ll act like a certain ISP did, and periodically reset the TCP connection. To do this, I created skeleton code for a module called TCPReseter. I also will supply the click graph tcprst.click which uses it. Your job is to fill in TCPReseter. As with bump.click, be sure to modify the device names as necessary. What TCPReseter does is with some probability $p$, send packets out of port 0 unmodified (which is then allowed to be sent to n2), and with some probability $1-p$, drops the received packet and sends a TCP RST out of port 1 (which is then sent back to n0). Note: in the skeleton, I use an int for $p$, so it can be 0 to 100 (representing 0.0 to 1.0). This is configured in the click graph with the number in parentheses).

Read FAQ in the click directory to learn how to add your own module (search for “Q. How can I add my own element class to Click?”). Put it in elements/local

In supplement_code.txt I added some code which you can take to realize this functionality (hint: it’s just about everything you need).

To test it, on n0 in one terminal, use tcpdump to monitor the traffic (sudo tcpdump -i eth1 -v -x). Then, also on n0, in a different terminal, run ITGSend with itg-script6 (running ITGRecv on n2 and click on n1 first, of course).

To Hand In: One line from the tcpdump output which shows the reset (you’ll see Flags [R]). And TCPReseter.cc and .hh