

Problem Set 1: Custom Packet Processing Software

*Instructor: Prof. Nick Feamster**College of Computing, Georgia Tech*

This problem set has three questions, each with several parts. 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 Georgia Tech honor code, posted on the course Web site).

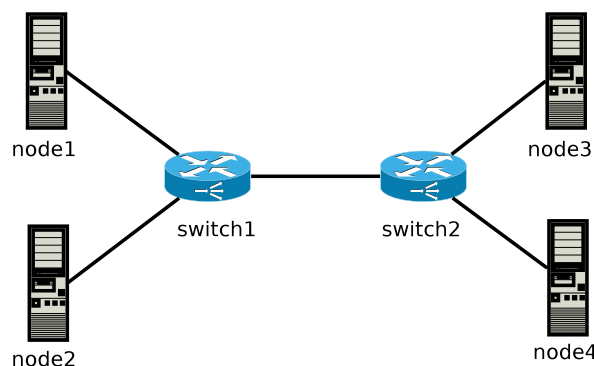
Turn in your solutions in on **February 9, 2010** by 11:59 p.m. -0500, to T-Square.

1 Warm Up

This problem will give you hand-on experience with Layer 2 and Layer 3 protocols, and with setting up network experiments in Emulab. The experimental configuration language is based on Tcl, which is the same as that which is used for *ns*, a network simulator.

Setting Up Your Experiment

1. **Emulab** Go to <http://www.emulab.net> and request an account. Join project 'cs8803', group 'cs8803'. *Do this early, as the the course staff will need to approve your membership!*
2. **Topology Setup.** Write a tcl procedure that takes an integer n and creates a dumbbell topology as shown in the figure below. 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. Name the nodes as in the Figure, as we will be using them in later parts of this problem.



Hint: Reading Emulab and *ns* documentation will help with this problem.

3. **Topology Creation.** Create your topology in Emulab. You can use the Emulab interface to get the DNS names and IP addresses of each of the nodes in your topology.

You should be also able to `ssh` to the nodes and send pings to and from each of the nodes at this point. We will now experiment on the nodes themselves.

Fun with ARP and Click

1. **LAN Setup** Put all of the nodes in a large LAN in your configuration file.
 - (a) Log into *node1* and print the ARP table (include the output in your hand-in). Ping *switch1*. Print the ARP table again. What happened?
 - (b) What does the “C” flag mean in the ARP table?
2. **Writing Your own Switch** In this part of the problem, you’ll write the switch table entries yourself using Click. Fortunately for you, Click already has an `EtherSwitch` element, so most of the “hard work” is done. You just have to figure out how to set it up!
 - (a) Modify your experiment script so that instead of all the nodes being placed on a single LAN, the nodes are simply set up as point-to-point links. You should be able to test this because you’ll only be able to ping adjacent nodes in the dumbbell topology.
 - (b) Download, compile, and install the Click (<http://www.read.cs.ucla.edu/click/>) modular router on *switch1* and *switch2* in your topology.

Possibly helpful hint: To save yourself the trouble of compiling on the nodes themselves, you can compile on a (possibly faster) home machine, and then use emulab’s `tb-set-node-os` command to install the proper OS on the Emulab node that will run your binary.
 - (c) Install and configure the Click elements on *switch1* and *switch2* so that all nodes in the topology can ping each other. The `EtherSwitch` element will likely be quite helpful for you in this regard; `ListenEtherSwitch` might also be helpful, particularly for debugging.

What to submit.

- Answers to the questions above.
- Emulab setup file for your dumbbell topology, entitled `ps1-1.ns`. We will test your configuration to make sure it instantiates the topology.
- Click configuration file for your learning switch, entitled `ps1-1.click`. We will test your switch in your Emulab configuration to ensure that the learning switch fully connects your topology.

2 Custom Click Element: Eliminating Duplicate Packets

In this part of the problem, you will build your own Click element that *eliminates duplicate packets*.

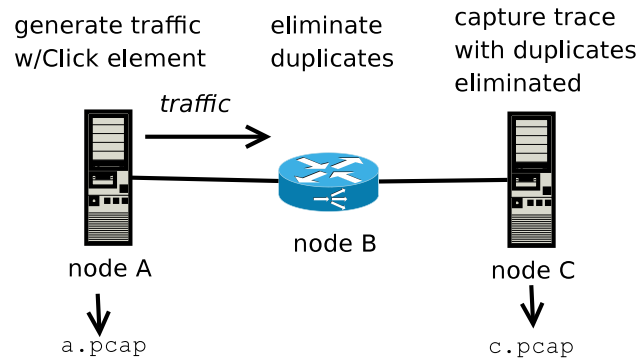
Emerging network architectures have recently seen a rise in the need to eliminate redundant traffic. For example, a network protocol that uses multipath routing might generate duplicate packets and send them along disjoint paths that converge closer to the destination. A network element at the point of convergence could eliminate the duplicate packets before they reach the receiver, saving network resources. For more information on redundancy elimination, this paper is a good starting point:

Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination
Ashok Anand, Archit Gupta, Aditya Akella, Srinivasan Seshan and Scott Shenker.
SIGCOMM 2008

To complete this part of the assignment, you will need to create your own Click element. To learn more about how to do that, please see the documentation on the Click Web page, here: <http://read.cs.ucla.edu/click/doxygen/>.

The TA may also be able to provide you an example of a custom element.

This problem has three parts. First, you will first create an example topology, as you did in Part 1 of this assignment. Second, you will create a Click setup that generates duplicate packets to test your duplicate elimination element. Finally, you will implement an element to eliminate duplicate packets and run it on your setup.



1. **Setup.** Configure a three-node Emulab topology as a single LAN, as shown in the figure above. You are welcome to use the `make-lan` operation in Emulab to permit this.
2. **Configure a duplicate traffic generator.** Set up a click configuration on node *A* that generates duplicate packets and sends them to node *B*. (The rate of the traffic generation is not important, but you may find the `RatedSource` element helpful and fun to play with.)
3. **Implement a custom Click element to eliminate duplicate packets.** Create a new Click element that detects duplicate packets *in a single traffic flow*. Define a *flow* as a collection of packets that have a common five-tuple: (source IP, destination IP, source port, destination port, and protocol type).
Then, your element should keep track of packets that have traversed any given flow and drop packets that have payloads that have already appeared on that flow.
4. **Test your duplicate elimination element.** Connect your traffic generation element on node *A* and run multiple traffic flows that generate duplicate packets through node *B*. Please configure your test harness to generate a traffic trace on node *A* and a traffic trace on node *C*.

What to submit.

- **Your traffic traces from node *A* and node *C*.** We will run a script against your trace to ensure that the trace captured on node *A* has duplicate packets and the trace on node *C* does not. Your traces should be generated by (and readable with) `libpcap` programs (e.g., `tcpdump`, `ethereal`). Your trace files should be called `a.pcap` and `c.pcap`.

- **Code for your duplicate elimination element.** Please name your element DupElim so that we can run it in our own test harness.
- **Emulab configuration for your three-node setup.** Call this file ps1-2.ns.
- **Your Click configuration for the traffic generation element and the duplicate elimination element.** We will want to set up your three-node configuration in Emulab and test it. We will also run our own duplicate traffic generator and ensure that your Click element produces the correct output against the input that we generate.

3 Part 3: Network Policy with OpenFlow

In this part of the assignment, you will familiarize yourself with OpenFlow, a new paradigm for configuring network switches. For more information about OpenFlow, please see this overview paper:

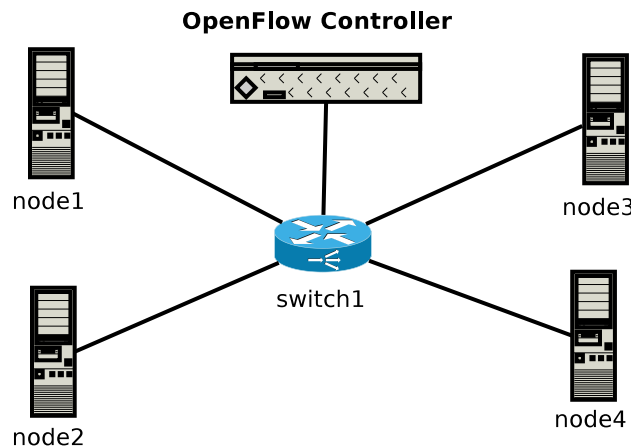
OpenFlow: Enabling innovation in campus networks. McKeown, N. and Anderson, T. and Balakrishnan, H. and Parulkar, G. and Peterson, L. and Rexford, J. and Shenker, S. and Turner, J. *ACM SIGCOMM Computer Communication Review*.

The OpenFlow Web site (<http://www.openflowswitch.org/>) has more details about OpenFlow.

1. **Setup.** Follow the instructions at http://noxrepo.org/manual/vm_environment.html to set up a virtual test environment using QEMU. *Note:* If you do this on a Debian or Ubuntu Linux setup, the course staff can help you. If you do this for another Linux distribution, you are on your own.
2. **Basic topology and policy configuration.** Configure an OpenFlow topology in your QEMU setup as shown in the figure below.

Modify the switch or pyswitch component to filter or restrict communication between two end hosts. Create a module that takes a file as input with one of two formats:

- IP1 [indicating that all traffic from IP1 should be dropped]
- IP2 IP3 [indicating that traffic between IP2 and IP3 should be dropped]



What to submit.

- **Your modified switch element.** We will run it in the same QEMU environment as you have set up in the first part of this problem with various policy configurations on the same topology as shown in the figure to ensure that your switch element works properly.