CS 6250: Computer Networking September 8, 2010

Problem Set 1

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 **September 29, 2010** by 11:59pm. *Please upload your solutions to T-Square. Other forms of submission will not be accepted!*

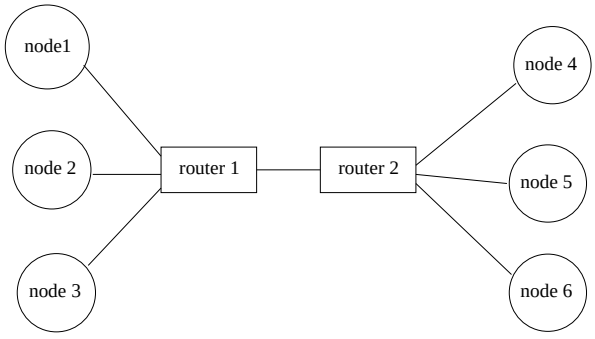
You are welcome to work with one other partner on this problem if you want, provided you (1) work out all the problems yourself; (2) type up your own assignment; (3) list on your assignment hand-in the name of the person you worked with.

Experience with ARP and OSPF in Emulab

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 'cs4251, group 'cs4251'. *Do this early, as I will need to approve your membership!*
2. **Topology Setup.** Write a tcl procedure that takes an integer n and creates a dumbell 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.

4. **Assigning IP addresses to your nodes.** By default, the Emulab experiment manager assigns hosts each Ethernet segment with IP addresses from distinct subnets; it will then install routing tables in the kernel and try to route packets based using the kernel routing table. Since you are doing an experiment that involves constructing a layer-2 topology, you will want to remove these routing table entries, assign hosts IP addresses from the same subnet, or both. The instructions at `https://users.emulab.net/trac/emulab/wiki/Nscommands#IPAddressCommands` explain how to automate IP address assignment in the experiment script itself.

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 *router1*. 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 *router1* and *router2* 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 *router1* and *router2* 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.

3. **Quick and Dirty Performance Evaluation**

   (a) Log into *node4* and start a tcpdump that looks for ARP packets.

   (b) Log into *node1* and begin pinging *node4*.

   (c) Show the packet trace excerpt containing the ARP query and response.

   (d) Modify your experiment setup so that the interface on *node4* that faces *router2* fails. Explain what you did to do this. Also give one other way you might have simulated this kind of effect. (*Hint:* You can do this manually on the node, automatically using Emulab, with Click, etc.). Reinstate the link after you're done with the next part of the problem.

(e) How long does it take before *node4* begins responding to pings after you have reinstated the link?

Fun with OSPF

In this part of the problem, you will configure the Quagga (`http://www.quagga.net/`) software router's OSPF module to connect the nodes in your topology.

1. Uninstall Click, but keep the point-to-point Emulab topology. Your topology should now be as before: each node should be able to ping its immediate neighbors, but no other nodes in the topology.
2. Install and configure OSPF on each of the nodes in the topology. (If you want to save work, it's fine if you only install OSPF on *node1*, *node4*, *router1* and *router2*, but you will see more of the flooding effects if you install everywhere. You can also save work by using a Perl, Ruby, or Python script to generate your configuration.)
3. How often do you see OSPF "Hello" messages? How often do you see an LSA message for any single link?
4. Repeat the "quick and dirty performance evaluation" above where you fail a link in the topology and then bring it back. How long does it take before *node1*'s routing table entries reflect that node4 is "dead"?