Assignment 2 - Switching

In this assignment, we will be be creating two switches that implement forwarding within a L2 network. First, we will create a static switch which will show how switches forward. This will use a much smaller configuration (you'll see why!). Second, we will create learning switch for a much larger topology wherein switches learn which port to forward traffic to get to any end host.

For this assignment, we will be using the Pyretic controller to program the switches. Much of the difficult parts are abstracted out in the helper file included within the assignment, but there are some features that you will need to read up on. See the wiki and this page in particular to get a feel for the language

Before getting started, you will need to update your CS6250 repository using the following commands from the 2015-Summer-OMS6250 directory:

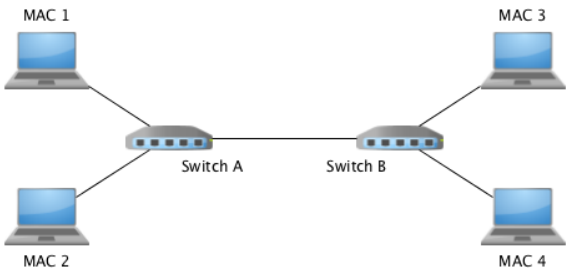
```
git commit -a -m "Saving work"
git pull --rebase
```

In the new `assignment-2` directory, there are six files. You can read up on `__init__.py` here and here. `helpers.py` contains helper functions. You will need to use the print routines in it to output your results in a format that we can grade. The files that end with `-topo.py` are the topology files that you will use to run the assignments. You will not have to modify them. The final two files are the assignment files.

Part 1 - Static Switching

Static switching is where we manually fill in switch tables matching destination MAC address to the port the packets should go out of. It is similar to how static routing works, except at the MAC layer rather than IP layer. There are serious problems, however. First, it doesn't scale well - for each end host, each switch must have a forwarding path programmed in. Second, it cannot handle dynamic changes - if a host moves from Switch A to Switch B, many, possibly all, switches have to have the forwarding path updated. Third, it's highly error prone.

What it does do is provide us with a good example of how to deal with Pyretic, in particular manipulating packets and creating forwarding behaviour. It also shows us how forwarding behaviour is implemented by keying off of the destination MAC address to make forwarding decisions.



The topology we are using is above. In code, Switch 1 is Switch A in the diagram above, and Switch 2 is Switch B. This is to simplify descriptions using the diagram, and limitation Mininet.

1. Look in the file `static-forwarding-topo.py`. This is a description of the topology. It's very simple. You can start this by, in one terminal, running the command `sudo python static-forwarding-topo.py`. If you see a warning about `Unable to contact the remote controller`, you can ignore that. As long as you get the `mininet>` prompt, it's working. (You may see an error,

`Exception: Error creating interface pair: RTNETLINK answers: File exists` on subsequent runs. You can run `sudo mn -c` to clear this.)

From the `mininet>` prompt, you can run commands on hosts/switches by giving the name of the host/switch followed by the command. For example, `h1 ifconfig` will run ifconfig on host h1. This can be useful for querying the topology for details about the hosts/switches.

1. Now, you will need to make changes to `static-forwarding.py` in the sections marked `TODO`. As it stands, this file will not run without your modifications. There is significant example code that you can base your code on.

It will be useful to know that MAC addresses are assigned to the hosts as `00:00:00:00:00:01` to the first host (i.e., the first call to `addHost()`), `00:00:00:00:00:02` to the second, and so forth...

You also need to make sure the broadcast MAC `ff:ff:ff:ff:ff:ff` is forwarded to all ports. Some starter code for this is provided.

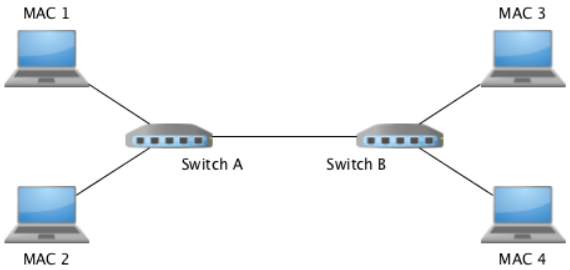
1. When you written up your forwarding table, you *must* write out your forwarding configuration. Use the functions at the beginning of `helpers.py`. First, open a file with the name `static-forwarding.log` using the `open_log()` function, write out all the forwarding table entries (order doesn't matter), then close the file with `finish_log()'.

2. To run your code, you will need to copy it over to the pyretic directory, then run it with Pyretic. To do this, you'll need to use the following commands.
`cp static-forwarding.py helpers.py ~/pyretic/pyretic/modules; cd ~/pyretic; python pyretic.py -m p0 pyretic.modules.static-forwa`
or use the provided run script for more convenient use.

Part 2 - Learning Switch

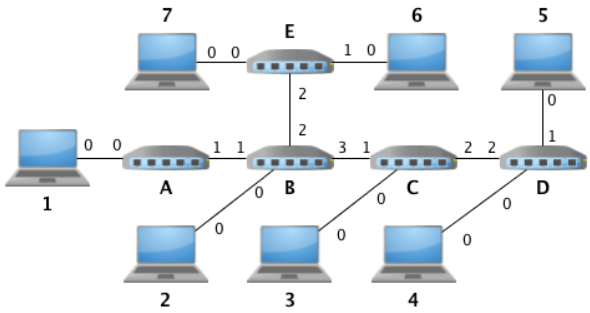
Learning Switch Review

We're going to go over how learning switches work again, just in case, using the topology from the first part of the assignment.



When the topology has just come up, the switch tables are empty. When a packet goes from Host 1 (with MAC address 1, for simplicity's sake), and is destined for Host 2, it first goes to Switch A. Switch A will record which port Host 1 came in on. Since Switch A does not know where the MAC address for 2 is, will flood and send a copy of the packet to both remaining ports. It will reach Host 2, but it will also reach Switch 2. Switch 2 will save off how to get to Host 1 (via Switch 1), and flood to hosts 3 and 4.

If, afterward, Host 3 is trying to send a packet to Host 1, Switch B learn how to get to Host 3, will not flood and send it directly to Switch A. Switch A will also learn how to get to Host 3 (via Switch B), and forward directly to Host 1.



Above is the topology provided for the assignment. There are significantly more switches than in the review, however this allows for more test cases possible. You will be able to use both this topology and the one for the static part of the assignment for testing purposes.

Completing this part of the assignment is similar to the first half.

1. Look in the file `learning-switch-topo.py`. It's much more complicated than `static-forwarding-topo.py` but you can run in the same way. You can (and should!) use both topologies for your own testing.

2. In `learning-switch.py` there are significantly more TODOs than in the first half. The biggest difference is filling in the `learn_route()` function. You have lots of different ways of handling the forwarding table, and it's up to you to decide on the right thing to do. We've placed suggestions in the file that you can follow or ignore if you have other ideas. Logging is different this time. At the beginning, you must open the file as before with the filename `learning-switch.log`. You should fill in `print_switch_tables()` by logging all entries in the log file as before, then calling `next_entry()` once logged each entry in the switch table. This is so we get a view over time of the changes in the switch forwarding table.

You also need to make sure the broadcast MAC `ff:ff:ff:ff:ff:ff` is forwarded to all ports. Refer to the starter code from the static forwarding portion to see how this can be done.

1. Running your code is similar to before. To do this, you'll need to use the following commands.
`cp learning-switch.py helpers.py ~/pyretic/pyretic/modules; cd ~/pyretic; python pyretic.py -m p0 pyretic.modules.learning-switc`
or use the provided run script for more convenient use.

Note that if you test with ping, the first few pings (usually ~3) may not go through while the switch learns, so allow several pings before you conclude it's not working... However, once a path is learned, subsequent pings should go through okay.

What to turn in

You will need to turn in two files:

1. `static-forwarding.py`
2. `learning-switch.py`

The autograder will run each piece of code and verify its output file. BE SURE TO USE THE CORRECT FILE NAME OR YOU WILL RECEIVE A ZERO.

For the static forwarding part, we will use the exact same topology as you did.

For the learning switch, we will be using a different topology so that you can be sure that your code is actually *learning*.

For the learning switch, do not use outside code. There is plenty of it existing - which you can review, but not use.

This page was last edited on 2015/05/26 12:31:51.