Project 1: Learning Switch

Due: Thursday 10/20 by 5pm.

For your questions: `ecalp at cs dot wisc dot edu`

Overview

In this assignment, you are going to implement the core functionalities of an Ethernet learning switch using the Switchyard framework. An Ethernet switch is a layer 2 device that uses packet switching to receive, process and forward frames to other devices (end hosts, other switches) in the network. A switch has a set of interfaces (ports) through which it sends/receives Ethernet frames. When Ethernet frames arrive on any port, the switch process the header of the frame to obtain information about the destination host. If the switch knows that the host is reachable through one of its ports, it sends out the frame from the appropriate output port. If it does not know where the host is, it floods the frame out of all ports except the input port.

Details

Your task is to implement the logic that is described in the flowchat here. As it is described in the last paragraph of the "Ethernet Learning Switch Operation" section, your switch will also handle the frames that are intended for itself and the frames whose Ethernet destination address is the broadcast address `FF:FF:FF:FF:FF:FF`.

In addition to these, you will also implement three different mechanisms to purge the outdated/stale entries from the forwarding table. This will allow your learning switch to adapt to changes in the network topology.

These mechanisms are:

- Remove an entry from the forwarding table after 10 seconds have elapsed
- Remove the least recently used (LRU) entry from the forwarding table. For this functionality assume that your table can only hold 5 entries at a time. If a new entry comes and your table is full, you will remove the entry that has not been matched with a Ethernet frame destination address for the longest time.
- Remove the entry that has the least traffic volume. For this functionality assume that your table can only hold 5 entries at a time. Traffic volume for an entry is the number of frames that the switch received where `Destination MAC address == MAC address of entry`.

You will implement these mechanisms in three separate Python files. The core functionalities that are explained above will be the same for these implementations.

Testing your code

Once you develop your learning switch, you should test the correctness of your implementation. Switchyard allows you to write scenarios to test your implementation. You can find more detailed information on creating test scenarios here. You can also find a simple test scenario in `switchyard/examples/hubtests.py`. Once you understand and get comfortable with the framework, make sure that you test your switch implementations meticulously. Do not forget to consider corner cases. Make sure that your entry purging mechanisms work as expected.

Once you prepare your test scenario, you can compile it as follows:

```
./srpy.py -c -s mytestscenario.py
```

To run the test scenario with your switch implementation:

```
./srpy.py -t -s mytestscenario.srpy myswitchimplementation.py
```

You can find more detailed information on compiling test scenarios and running in the test environment in the Switchyard documentation. In addition to these, you should also try running your switch in Mininet. You can find more information on this here.

Handing it in

You will submit a **.tar.gz** file that will include:

- **myswitch_to.py**: Your learning switch with timeout based entry removal
- **myswitch_lru.py**: Your learning switch with LRU based entry removal
- **myswitch_traffic.py**: Your learning switch with traffic volume based entry removal
- **README.txt**: This file will contain a line for each student in your team: [name_of_student][single whitespace][10-digit UWID]

IMPORTANT: The file names in your submission package has to **exactly match the file names above**. Otherwise, you will lose points!

In addition to these files, you will also submit your test scenario files. Please submit your Python files, **not the compiled version with the .srpy extension**.

Notes

1. We are providing you with a Ubuntu 14.04 (64-bit) VM image for this assignment. This image has Switchyard, Mininet and Wireshark installed so you do not need to worry about setting up the environment.
    - You can find the VM image here. (user name: **cs640user** - password: **cs640**)
    - You can learn more about importing a VM image in VirtualBox here. CSL machines already have VirtualBox installed so you should be able to use the image there without any problems. You are also free to use your favorite virtuaization software for importing the image but you will most probably have to deal with the possible issues on your
2. If you are a free soul and want to setup Switchyard in a different environment you are welcome to do that as well. You can find some useful information here. I prepared a file with commands that I have used to setup the Switchyard environment on Ubuntu 14.04. This might or might not be useful for you depending on your environment.
3. Documentation for Switchyard is available at http://cs.colgate.edu/~jsommers/switchyard.
4. Instructions for submitting the assignment will be announced later.
5. I will update the FAQ if multiple people run into the same problems, so it might be useful to regularly check the FAQ. Otherwise, you can always shoot me an e-mail.

FAQ

1. **Q:** I get the following error when I try testing my switch implementation in Mininet: ***NameError: name 'devname' is not defined (switchy_real.py, line 261 in send_packet())***. How can I fix it?

   A: This issue is now fixed. Please go ahead and pull down the updated version of Switchyard from the GitHub repository.

2. **Q:** Let's assume that the table in my switch has 5 entries: [h2, h3, h4, h5, h1] where h2 is the entry that has not been matched the longest while h1 is the most recently matched entry. If a new packet(src=h6,dest=h2) arrives, how is my switch supposed to handle this packet in the LRU-based entry removal implementation (assuming that the network topology does not change)?

   A: Whenever you receive a new packet, you will assess the state of the switch as if you don't know about the new packet and make decisions accordingly. So when your switch receives (h6, h2), it is going to add an entry for h6 since it is not in the table. However, since the table is full(5 entries) it will need to remove the LRU entry, which is h2. So your table is going to look like this: [h3, h4, h5, h1, h6] and your switch will broadcast the incoming packets on all ports except for the incoming port since it does not have information about h2 anymore. In other words, your switch (upon receiving the packet) is not going to update the table to [h3,h4,h5,h1,h2], remove h3 and add h6 to get [h4,h5,h1,h6,h2] and output the packet on a single port, which goes to h2.

3. **Q:** How do the entry removal mechanisms work?

   A: Flow chart for LRU, flow chart for traffic volume, flow chart for timeout

   Note that the flow chart for timeout based mechanism does not show when/how to purge the stale entries. Your implementation will obviously handle this as well. Keep in mind that there is not a limit on the number of entries that the table can hold for this mechanism.

4. **Q:** How would the table look for the following sequence of packets in the LRU-based implementation: (h1,h4), (h2,h1), (h3,h1), (h4,h1), (h5,h1), (h6.h7), (h4,h5)? (assuming that the network topology does not change)

   A: Assuming that the leftmost entry is the most recently used and the rightmost is the least recently used: [h1]-->[h1,h2]-->[h1,h3,h2]-->[h1,h4,h3,h2]-->[h1,h5,h4,h3,h2]-->[h6,h1,h5,h4,h3]-->[h5,h6,h1,h4,h3]

5. **Q:** Should our switch implementations be aware of changes in the topology?

   A: Your learning switch has to be aware of the changes in the topology. More specifically, if the switch receives a packet from host A on its interface 1 (i1) it will record this in its table {a->i1}. Later, if the switch receives another packet from host A but on a different interface (say i2), and if the entry {a->i1} is still present, it will be updated to {a->i2}. There will not be two different entries for the same host in your table! Reflecting the topological changes in your implementations will differ slightly:

   Timeout-based: When updating the entry for a particular host, reset its timer to 0 (this will be equivalent to refreshing the entry for that host).
   LRU-based: When updating the entry for a particular host, **do not** update its LRU information.
   Traffic volume-based: When updating the entry for a particular host, keep the same traffic volume count for the host. **Do not** set it to 0.