Lab Assignment-1

Submit Assignment

Due Mar 6 by 11:59pm **Points** 50 **Submitting** a file upload **Available** after Feb 17 at 12am

Due: 03/06/2020

For your questions: thumbe at wisc dot edu, royyesudhas at wisc dot edu

Overview

In this assignment, you are going to implement the core functionalities of an Ethernet learning switch with Spanning Tree using the Switchyard framework (https://github.com/jsommers/switchyard). 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 will 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.

Spanning Tree (or STP) is a network protocol in Ethernet used to prevent broadcast storms, by converting a physical loop into a logical loop-free topology. This is done by electing a root port in the topology and designating ports as forwarding or blocking, depending on the proximity to the root switch.

Details

Part 1: Learning Switch

Your task is to implement the logic that is described in the flowchart here (https://github.com/jsommers/switchyard/blob/master/examples/exercises/learning_switch/learning_switch.rst#ethernet-learning-switch-operation). A more elaborate flowchart has been described in the FAQ section. 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In addition to these, you will also implement two mechanisms to purge the outdated/stale entries from the forwarding table.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.

Given below is the replacement mechanisms that you need to implement in a file named **myswitch_lru.py**.

- 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.You will implement this mechanism in a Python file named **myswitch_lru.py.**

NOTE: There is an example of a switch without learning implemented in switchyard-master/examples/exercises/learning_switch/ which will likely be useful to get you started.

Also please keep an eye on the FAQ section below which will be updated with information regarding the most common issues that arise during implementation.

Part 2: Spanning Tree implementation

Helper code can be downloaded here (SpanningTreeMessage.py). For the second part of the assignment you will build on top of your learning switch implementation. STP packets will be handled in a separate code path whereas the other packets will continue as per the behavior of the learning switch. Create a copy of the above file (myswitch_lru.py) and name it as **myswitch_stp.py.** You will be implementing a simplified version of the Spanning Tree Protocol for the purpose of this assignment. The problem description has been modified from here **(https://github.com/jsommers/switchyard/blob/master/examples/exercises/learning_switch/learning_switch.rst#implement-a-simplified-spanning-tree-computation-algorithm)** to suit our requirements as described below. (**NOTE:** This is not the actual STP working but a simplified version which can be implemented within the restrictions of the framework capabilities.)

If you attempt to run your switch on multiple nodes within a virtual network using Mininet, and if there is a physical *loop* in the network, you will observe that packets will circulate infinitely. Oops. An interesting and challenging extension to the learning switch is to implement a simplified spanning tree computation algorithm. Once the spanning tree is computed among switches in the network, traffic will only be forwarded along links that are part of the spanning tree, thus eliminating the loop and preventing packets from circulating infinitely.

To implement a spanning tree computation, you'll need to do the following:

1. Add to your switch the notion of an *id*, which can just be an integer (or even a string). The point is that each switch will need its own unique id, and there should be a simple way to compare ids. (**NOTE:** For our implementation the ID of the switch will be equal to the smallest MAC address among all it's ports MAC addresses.)
2. Create a new packet header type that includes three attributes: the id of the *root* node in the spanning tree, the number of observed hops to the root, and the id of the switch that forwards the spanning tree packet (**NOTE**: For this part of the assignment you can refer to API example given here **(https://jsommers.github.io/switchyard/advanced_api.html#one-more-example)** . Also note that (root id = switch id) when the message is generated at the root.
3. The implementation has been packaged and given in the starter code under **spanningtreemessage.py**. Just import that in your myswitch_stp.py). The source MAC in the Ethernet header can be anything as we are not going to learn any MAC table information from STP packets but the destination MAC should be broadcast address (“ff:ff:ff:ff:ff:ff”).
4. Add a capability to each switch so that when it starts up, it floods out spanning tree packets to determine and/or find out what node is the root. Each switch needs to store a few things: the id of the current root (which is initialized to the switch's own id), the number of hops to the root (initialized to 0), and the time at which the last spanning tree message was received. Each non-root node also needs to remember
    1. the interface on which spanning tree message from the perceived root arrives => ***root_interface***

   2. ID of the switch connected to the ***root_interface*** => ***root_switch_id***
   3. The information about which ports are blocked.
   4. The time at which the last spanning tree message was received from the **root_interface =>*last_stp_time***
5. Let's call the interface through which STP message is received => ***incoming_interface***
6. Only root nodes generate STP packets periodically. Initially, a node assumes that it is the root. These packets (root_id, number of hops, switch id) are initialized as (switch_id, 0, switch id). The root node should emit new spanning tree packets every 2 seconds. Once a node gets to know that it is not the root, it should stop generating spanning tree messages.(NOTE: update **last_stp_time** for root node as soon as it generates the stp packet)
7. When a node receives a spanning tree packet it examines the interface through which the message was received and the root attribute of the packet.
8. If ***incoming_interface*** is same as ***root_interface*** or the root ID in the received packet is *smaller* than the ID that the node currently thinks is the root, the switch updates its information(as described in point 4) and forwards the STP packets taking information update into account.(**NOTE:** Also update ***last_stp_time*** in this case)
9. If the root ID in the received packet is greater than the id that node currently thinks is the root, then remove ***incoming_interface*** from the list of blocked interfaces (if present).
10. If the root ID in the received packet is *the same* as the ID that the node currently thinks is the root, it examines the number of hops to the root value:
    1. (If the number of hops to the root + 1 is less than the value that the switch has stored) or (If the number of hops to the root + 1 is equal to the value that the switch has stored and the ***root_switch_id*** is greater than the switch_id of the packet), then
        1. switch removes the ***incoming_interface*** from the list of blocked interfaces(if present), block the original ***root_interface*** and update ***root_interface*** = ***incoming_interface*** and other relevant information(as described in point 4). Finally forward the STP packets taking the information update into account.
    2. Otherwise, block the ***incoming_interface***

11. If a non root node doesn't receive STP messages for more than 10 seconds i.e *current_time - **last_stp_time**>10*, then re-initialize the root id to switch's own id, hop count to 0 and remove all interfaces from the set of blocking interfaces.
12. Lastly, the learning switch forwarding algorithm changes a bit in the context of a spanning tree. Instead of flooding a frame with an unknown destination Ethernet address out *every* port (except the one on which the frame was received), a switch only floods a frame out every port except the input port and the ports corresponding to the set of interfaces that are blocked.

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 **(https://cs.colgate.edu/~jsommers/switchyard/2017.01/test_scenario_creation.html)** . 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:

```
./swyard.py -c mytestscenario.py
```

To run the test scenario with your switch implementation:

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

You can find more detailed information on compiling test scenarios and running in the test environment in the Switchyard documentation.

(Optional) In addition to these, you should also try running your switch in Mininet. You can find more information on this here **(https://github.com/jsommers/switchyard/blob/master/examples/exercises/learning_switch/learning_switch.rst#testing-and-deploying-your-switch)** .

NOTE: LRU implementation can be tested using the compiled code. But for testing the STP implementation directly pass the python file.

```
./swyard.py -t testfile.py myswitch_stp.py
```

Test Scenarios

You can find example test scenarios here.

Myswitchlru_test.srpy - compiled test file for testing lru implementation

Myswitchstp_test.py - test file for testing stp algorithm

Note that these tests are not comprehensive and are provided to help you debug simple issues. We will be running additional tests for grading your assignment.

Handing it in

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

- **SpanningTreeMessage.py:** Already provided. Leave it as is
- **myswitch_lru.py**: Your learning switch with LRU based entry removal
- **myswitch_stp.py**: Your learning switch with spanning tree implementation and the required LRU code
- **README** : Include names of team members and your netids. Only one of the group members should submit the assignment.

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

You **DON'T have to turn in your test scenarios** for this project. However, you should still write test scenarios that test all aspects of your code, since the test scenarios provided are not comprehensive.

Development notes

- 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. After logging in the VM you can find a switchyard_master folder. The tests and switch python file should be located and executed from that folder.

- You can find the VM image here (http://pages.cs.wisc.edu/~seanm/assets/Switchyard.ova) . (user name: **cs640user** - password: **cs640**)
- You can learn more about importing a VM image in VirtualBox here (https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html) .CSL machines (link (https://csl.cs.wisc.edu/services/remote-access/instructional-linux-computers)) already have VirtualBox installed so you should be able to use the image there without any problems. You can also install VirtualBox in your machines following this link (https://www.virtualbox.org/wiki/Downloads) .
- 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 (https://github.com/jsommers/switchyard#installation) . Here's a f (http://pages.cs.wisc.edu/~seanm/projects/sy_install.txt) i (http://pages.cs.wisc.edu/~karthikc/CS640S19/P1/sy_install.txt) le (http://pages.cs.wisc.edu/~seanm/projects/sy_install.txt) with commands that can be used to setup the Switchyard environment on Ubuntu 14.04. This might or might not be useful for you depending on your environment. (**NOTE** Grading however will be done using the VM provided above. If you build the code using a different environment, ensure that your code runs in the VM provided).
- Documentation for Switchyard is available at switchyard (http://cs.colgate.edu/~jsommers/switchyard/2017.01/)
- FAQ will be updated 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 email to get clarification.

FAQs

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

A: Assuming that the leftmost entry is the most recently used entry:

[h1]-->[h1,h2]-->[h1,h3, h2]-->[h1,h3, h2]-->[h1, h4, h3, h2]-->[h1, h5, h4, h3, h2]--->[h1,h6,h5, h4, h3]---> [h4, h1, h6, h5, h3] --> [h6,h2,h4, h1, h5]

Note: The ordering here is given to illustrate the behavior of entry & access.

Q Should the entries in the LRU switch be removed based on a timeout condition?

A: No, you shouldn't remove the entries based on a timeout. The entries should only be removed based on the LRU condition.

Q If the state of the mac table is [h5, h4, h3, h2, h1] and a packet (h6,h1) comes to the switch what should be the final state of the table?

A: Assuming that there are ordered starting with most recently (h5) used to least recently used(h1), you need to remove h1 and insert h6. Thus the table becomes [h6,h5,h4,h3,h2]. Now, since we don't have the entry for h1, you should broadcast the packet along all port except the incoming port.

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

A:

LRU switch: 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! When updating the entry for a particular host, do not update its LRU order information. (i.e if the host was third entry in the LRU queue it should still be the third entry after the update). LRU order changes only based on addition and access of entry.

STP switch:

We will be first testing if spanning tree protocol is running correctly on the switches. Once this is ensured we will check if LRU forwarding for this switch is working as expected (For instance: packets should not forwarded along the blocked ports). The idea of the stp switch is to learn the blocked ports.

Once the blocked ports are learnt through spanning tree algorithm, you just have to do LRU forwarding taking blocked ports into account and not worry about learning the spanning tree again(for the purpose of this assignment).

However **while** learning the spanning tree you have to consider topology changes into account. For instance, if the switch was connected to root switch with a distance of 2 hops, but there was a change in topology later, where the same switch has a direct connection to root, then the hop count of the switch to the root should be updated to 1.

Q What MAC addresses should STP packets contain when being flooded or forwarded?

1. **A**. STP packets are generated at the node itself, so whenever a STP packet is to be sent, it should be sent with src MAC of the node (for convenience hard-code the src mac to the ethernet address of **eth0** and destination should be broadcast always: "ff:ff:ff:ff:ff:ff"

Grading Rubric

1) 10 points for turning in required files that have required content and documentation

2) 10 points for code that runs the test cases provided

3) 15 points for code that successfully runs switching tests

4) 15 points for code that successfully runs spanning tree tests