CS6045 – Software Computer Networking
July-Nov. 2015, Prof. Krishna M. Sivalingam
Lab 3 - POX based Network Virtualization
Due Date: Sep. 30, 2015, 11 PM, on Moodle

NO Copying/Code-Sharing allowed from anywhere. Code written should be your own, except for use of Standard Templates. Students who violate this policy will receive an 'U' grade in the course

Based on Coursera course taught by Dr. Nick Feamster (Princeton Univ.);
`https://www.coursera.org/course/sdn1`

1 Network Virtualization

In this assignment, you will learn how to slice your OpenFlow network using the Pox OpenFlow controller. In the process, you will also learn more about the concept of flowspaces and how the centralized visibility and "layerless-ness" of OpenFlow enables flexible slicing.

The purpose of this exercise is to help motivate network virtualization and show you different ways in which network virtualization can be implemented.

1.1 Network Topology

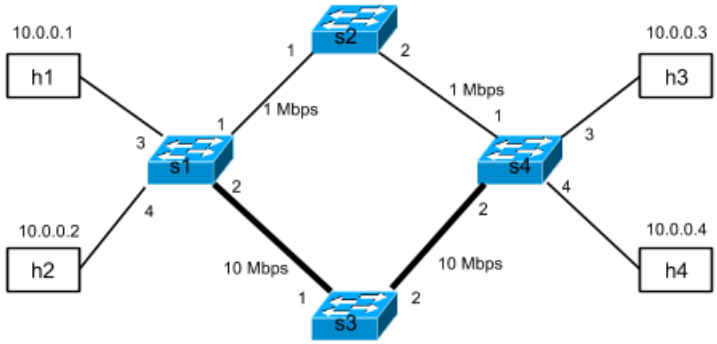

Figure 1: Network Topology.

In this assignment we will slice a wide-area network (WAN). The WAN shown in the Fig. 1 connects two sites. For simplicity, we will have each site represented by a single OpenFlow switch, s1 and s4, respectively. The sites, s1 and s4, have two paths between them: (i) a low-bandwidth path via switch s2; (ii) a high-bandwidth path via switch s3. Switch s1 has two hosts attached: h1 and h2, and s4 has two hosts attached: h3 and h4.

We will use a Mininet script, mininetSlice.py, to create a network with the topology provided above. The topology we have provided should correspond to the one shown in the figure. For the second part of the assignment, Pox's link-layer discovery protocol (LLDP) features are used to ensure that you can complete the assignment without ever explicitly referring to port numbers.

1.2 Code

To start this assignment update the course's Github repo (by default, `https://github.com/PrincetonUniversity/Coursera-SDN`) on your host machine using *git* or whatever you choose to use. Change to directory: `/vagrant/assignments/network-virtualization` with the following files:

1. mininetSlice.py: Mininet script to create the topology used for this assignment.
2. topologySlice.py: A skeleton class where you will implement simple topology-based slicing logic.

3. videoSlice.py: A skeleton class where you will implement advanced flowspace slicing logic.

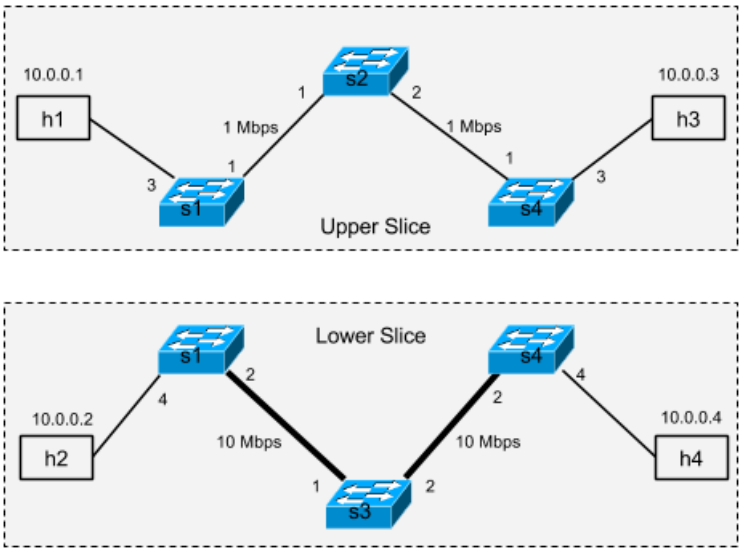

Figure 2: Topology based Slicing.

2 Topology Based Slicing

The goal of this part is to divide the network into two separate slices: **upper** and **lower**, as shown in Fig. 2. Users in different slices should not be able to communicate with each other. In practice, a provider may want to subdivide the network in this fashion to support multi-tenancy (and, in fact, the first part of this assignment provides similar functionality as ordinary VLANs).

To implement this isolation, we need to block communication between hosts in different slices. You will implement this functionality by judiciously inserting drop rules at certain network switches. For example, host h1 should not be able to communicate to host h2. To implement this restriction, you should write OpenFlow rules that provide this isolation. (Unlike previous assignment, this topology has multiple switches; each switch has its own flow table; the controller uses each switch's datapath ID to write flow rules to the appropriate switch.)

2.1 Code

The topologySlice.py file provides skeleton for this implementation. As in the previous assignment, the file includes a *launch()* function, which registers a new component. The *handle_ConnectionUp()* callback is invoked whenever a switch connects to the controller.

The code also launches the *discovery* and *spanning tree* modules, which compute a spanning tree on the topology, thus preventing any traffic that is flooded from looping (since the topology itself has a loop in it, computing a spanning tree is required).

2.2 Testing your code:

Copy your files to POX's directory, `pox/misc` under the *pox* main directory. Go to this main directory and launch the POX controller, as before.

```
$ cd ~/pox/
$ ./pox.py log.level --DEBUG misc.topologySlice
```

In a separate terminal, launch your Mininet script:

```
$ cd ~/pox/pox/misc
$ sudo python mininetSlice.py
```

Wait until the application indicates that the OpenFlow switch has connected and that all spanning tree computation has finished. Now, verify that the hosts in different slices are not able to communicate with each other using the *pingall* command at Mininet CLI.

3 Flowspace Based Slicing

In the previous part of the assignment, you learned how to slice a network based on the network's physical topology. It is also possible to slice the network in more interesting ways, such as based on the application that is sending the traffic. SDN networks in principle can be sliced on any attributes of flowspace.

Recall that the topology has two paths connecting two sites: a high-bandwidth path and a low-bandwidth path. Suppose that you want to prioritize video traffic in our network by sending all the video traffic over the high bandwidth path, and sending all the other traffic over the default low bandwidth path. File transfers won't be affected by the video traffic, and vice versa. In this part of the assignment, you'll use network slicing to implement this isolation.

Let's assume for simplicity that all video traffic goes on TCP port 80. In this assignment you are required to write the logic to create two slices, "video" and "non-video", as shown in Fig. 3.

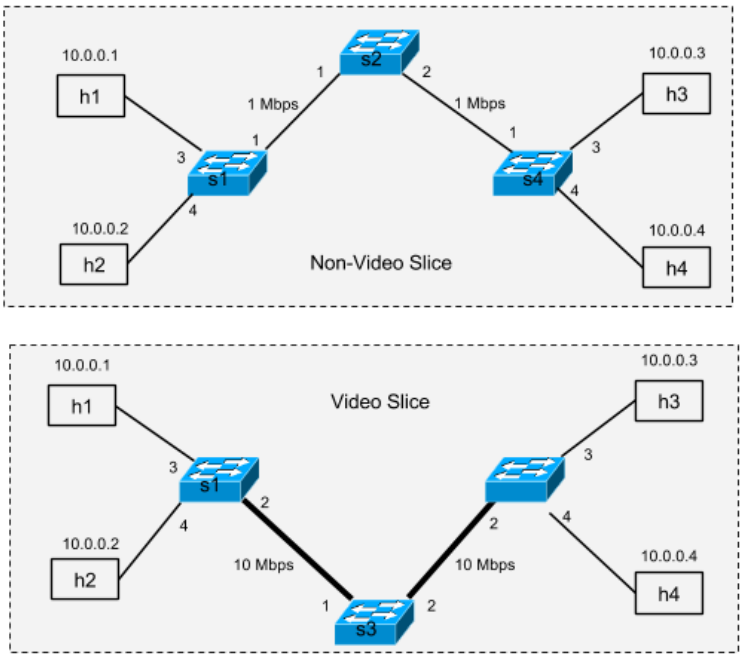

Figure 3: Video Traffic based Slicing.

3.1 Code

In `videoSlice.py`, you have a class called VideoSlice, with some of the logic to implement slicing. You should fill in the *portmap* data structure and the missing parts of the *forward* function. Also included is a line of the portmap data structure as a hint: the data structure should map a switch and a portion of flowspace to the *dpid* of the next switch. You will need to figure out how to implement a wildcard, in addition to explicit flowspace directives for port 80.

The structure of *self.portmap* is a four-tuple key and a string value. The type is:

```
(dpid string, src MAC addr, dst MAC addr, port (int))
```

The discovery module also passes information to the Pox controller about the topology, such as the switches that it knows about, and the ports of each switch that are connected to one another. This discovery protocol, link-layer discovery protocol (LLDP), is useful for part 2 of the assignment, but the code to do that discovery is already included. The `l2_multi.py` in the Pox distribution, which performs Layer 2 learning for multiple switches in a single topology, also makes use of LLDP.

Since the controller runs a spanning tree protocol (you should familiarize yourself with what this does), simply flooding from certain switches in the network as default behavior may not work. (In other words, if you find packets not getting through, consider being explicit in your portmap structure.)

3.2 Testing your Code

This is similar to what was done for Topology Slicing part.

You can now test that your slices work properly be ensuring that video (in this case, port 80) traffic traverses the 10 Mbps link and non-port 80 traffic traverses the 1 Mbps link. For example, you can test the two paths between h2 and h3 as below.

```
mininet> h3 iperf -s -p 80 &
mininet> h3 iperf -s -p 22 &
mininet> h2 iperf -c h3 -p 80 -t 2 -i 1
mininet> h2 iperf -c h3 -p 22 -t 2 -i 1
```

Here, we first run iperf server on h3 (on two different ports) and then two iperf clients that connect to these servers. You should observe about 10 Mbps port 80 traffic and about 1 Mbps throughput for port 22 traffic (or any port that is not port 80).

4 What to Submit

Create a file called CS1xGabc-Lab3.tar.gz that contains one sub-directory for each part of the assignment. In the sub-directory, copy the modified files ONLY (and screenshots/report for the Spanning Tree problem). Each sub-directory should contain a README file that describes how the program has to be run.

Then, upload the tarfile to Moodle.

5 Grading Criteria

- Topology Splicing: 20 points
- Video Splicing: 30 points

6 Misc

- It is possible to move many mininet command line options to the startup python file. See, for instance, `pox/samples/topo.py`.
- Make sure that you have gone through Openflow tutorials and the OpenFlow Wiki.
- Additionally, go through the FlowVisor tutorial and the OpenVirtex tutorials to better understand how network virtualization works.
- Start your work early.