Flowvisor Exercise

Jump to bottom

nikhilh edited this page on 15 Apr 2013 · 36 revisions

Table of Contents

In this exercise, you will learn how to slice your OpenFlow network and have each slice controlled by a separate controller. In the process, you will also learn the concept of flowspaces and how the centralized visibility and "layerless-ness" of OpenFlow enables flexible slicing.

For this exercise we will use an open source tool called the FlowVisor.

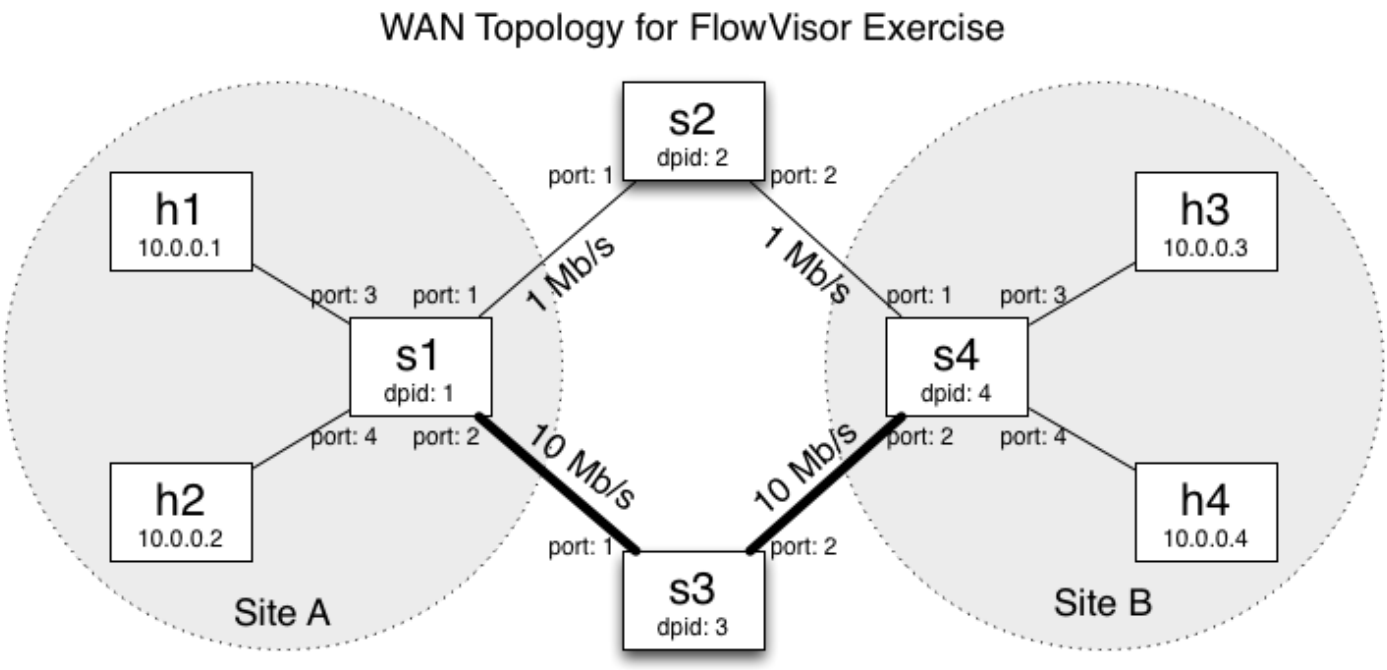
FlowVisor

FlowVisor is a special purpose OpenFlow controller that acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers.

FlowVisor creates rich slices of network resources and delegates control of each slice to a different controller. Slices can be defined by any combination of switch ports (layer 1), src/dst ethernet address or type (layer 2), src/dst IP address or type (layer 3), and src/dst TCP/UDP port or ICMP code/type (layer 4). FlowVisor enforces isolation between each slice, i.e., one slice cannot control another's traffic.

You can find more information about FlowVisor and source code at http://www.flowvisor.org

Exercise Topology



In this exercise we will slice a wide-area network (WAN) in two different ways. The WAN shown in the figure above connects two sites. For simplicity, we'll have each site represented by a single OpenFlow switch, s1 and s4, respectively. The sites, s1 and s4, have two paths between them:

- a low bandwidth path via switch s2
- a high bandwidth path via switch s3

s1 has two hosts attached: h1 and h2. s2 has two hosts attached: h3 and h4.

Environment Setup

Run Diamond Topology

First, start up the diamond topology for this exercise. Exit any running Mininet instance, and run:

```
$ sudo mn --custom ~/onstutorial/flowvisor_scripts/flowvisor_topo.py --topo fvtopo -
```

This will create a network in mininet with the WAN topology. To keep things simple, we have set static ARP entries for the Mininet hosts.

If you're curious about how to define a custom topology, check out the `flowvisor_topo.py` script; this contains Python code to define a topology named `fvtopo`, which then becomes accessible on the command line.

Start FlowVisor

Next we will configure FlowVisor, as it has not been run on your VM. First, **in a new terminal**, generate a flowvisor config:

```
$ sudo -u flowvisor fvconfig generate /etc/flowvisor/config.json
```

If prompted for the mininet sudo password, enter `mininet`. Leave the `fvadmin` password blank (just hit enter when prompted for password).

Start FlowVisor to be able to change the config:

```
$ sudo /etc/init.d/flowvisor start
```

Enable the FlowVisor topology controller using `fvctl`, which is a command-line utility to manage FlowVisor. The `-f` argument points to a password file. Your configured your FlowVisor instance to be passwordless, therefore the password file is `/dev/null`.

```
$ fvctl -f /dev/null set-config --enable-topo-ctrl
```

Restart FlowVisor:

```
$ sudo /etc/init.d/flowvisor restart
```

When FlowVisor started, all the OpenFlow switches we created earlier should have connected to it.

Next, ensure that FlowVisor is running by getting its configuration:

```
$ fvctl -f /dev/null get-config
```

If FlowVisor is running, you will see the FlowVisor configuration in JSON format like this:

```
$ fvctl -f /dev/null get-config
{
  "enable-topo-ctrl": true,
  "flood-perm": {
    "dpid": "all",
    "slice-name": "fvadmin"
  },
  "flow-stats-cache": 30,
  "flowmod-limit": {
    "fvadmin": {
      "00:00:00:00:00:00:00:01": -1,
      "00:00:00:00:00:00:00:02": -1,
      "00:00:00:00:00:00:00:03": -1,
      "00:00:00:00:00:00:00:04": -1,
      "any": null
    }
  },
  "stats-desc": false,
  "track-flows": false
}
```

List the existing slices using the following command:

```
$ fvctl -f /dev/null list-slices
```

Ensure that the only slice is the default `fvadmin` slice. The output should look like this:

```
$ fvctl -f /dev/null list-slices
Configured slices:
fvadmin         --> enabled
```

List the existing slices using the following command:

```
$ fvctl -f /dev/null list-flowspace
```

Ensure that there are no existing flowspaces. The output should look like this:

```
$  fvctl -f /dev/null list-flowspace
Configured Flow entries:
  None
```

Ensure that all the switches have connected by running the following command:

```
$ fvctl -f /dev/null list-datapaths
```

You might have to wait for a few seconds for the switches to connect before running the command. If all the switches are connected, you should see an output like this:

```
$ fvctl -f /dev/null list-datapaths
Connected switches:
  1 : 00:00:00:00:00:00:00:01
  2 : 00:00:00:00:00:00:00:02
  3 : 00:00:00:00:00:00:00:03
  4 : 00:00:00:00:00:00:00:04
```

Next, ensure that all the links are up by running the following command:

```
$ fvctl -f /dev/null list-links
```

You should see an output like this:

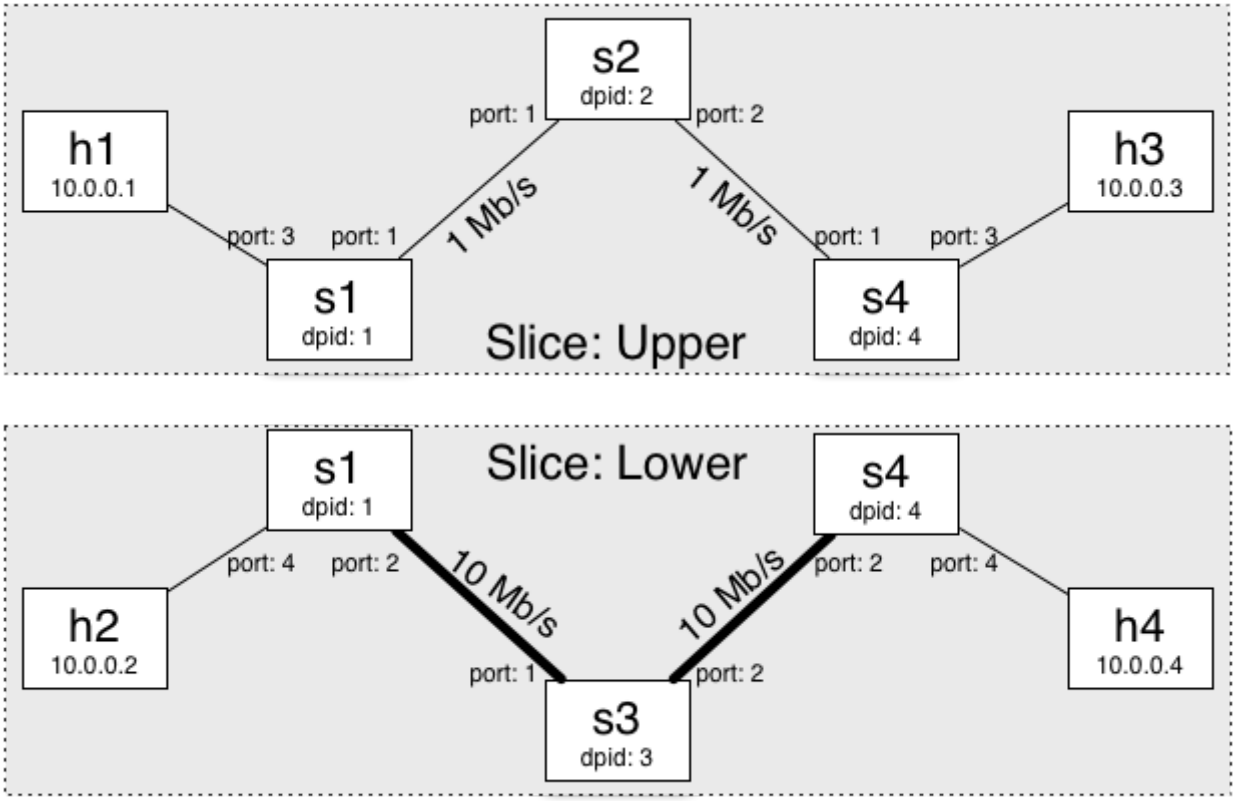
```
$ fvctl -f /dev/null list-links
[
  {
    "dstDPID": "00:00:00:00:00:00:00:01",
    "dstPort": "1",
    "srcDPID": "00:00:00:00:00:00:00:02",
    "srcPort": "1"
  },
  {
    "dstDPID": "00:00:00:00:00:00:00:02",
    "dstPort": "1",
    "srcDPID": "00:00:00:00:00:00:00:01",
    "srcPort": "1"
  },
  {
    "dstDPID": "00:00:00:00:00:00:00:04",
    "dstPort": "1",
    "srcDPID": "00:00:00:00:00:00:00:02",
    "srcPort": "2"
  },
```

```
  {
    "dstDPID": "00:00:00:00:00:00:00:03",
    "dstPort": "2",
    "srcDPID": "00:00:00:00:00:00:00:04",
    "srcPort": "2"
  },
  {
    "dstDPID": "00:00:00:00:00:00:00:01",
    "dstPort": "2",
    "srcDPID": "00:00:00:00:00:00:00:03",
    "srcPort": "1"
  },
  {
    "dstDPID": "00:00:00:00:00:00:00:04",
    "dstPort": "2",
    "srcDPID": "00:00:00:00:00:00:00:03",
    "srcPort": "2"
  },
  {
    "dstDPID": "00:00:00:00:00:00:00:02",
    "dstPort": "2",
    "srcDPID": "00:00:00:00:00:00:00:04",
    "srcPort": "1"
  },
  {
    "dstDPID": "00:00:00:00:00:00:00:03",
    "dstPort": "1",
    "srcDPID": "00:00:00:00:00:00:00:01",
    "srcPort": "2"
  }
]
```


Slicing Exercise

Part1: Simple Topology-based Slicing

FlowVisor can slice a network in many ways. Here, we are going to explore the simplest way--slicing by switch ports. An example use case is a WAN provider who wants to slice her network to provide dedicated links to each of her tenants. In this part, we will create two physical slices of our WAN, naming them upper and lower as shown in the figure below.



Create Slices

Each slice will be managed by a separate controller which will control all the traffic in its slice. As an example (no need to type this one), commands to create a slice would use the `add-slice` command in `fvctl`:

```
fvctl add-slice [options] <slicename> <controller-url> <admin-email>
```

The controller URL is of the form `tcp:hostname:port`. The admin email is used for administrative purposes if there is a problem with the slice. You can learn more about the command by typing:

```
$ fvctl add-slice -h
```

Now, create a slice named `upper` connecting to a controller listening on `tcp:localhost:10001` by running the following command:

```
$ fvctl -f /dev/null add-slice upper tcp:localhost:10001 admin@upperslice
```

Leave the Slice password empty by hitting Enter when prompted.

Similarly, create a slice named `lower` connecting to a controller listening on `tcp:localhost:10002`.

```
$ fvctl -f /dev/null add-slice lower tcp:localhost:10002 admin@lowerslice
```

Leave the Slice password empty by hitting Enter when prompted.

Ensure that the slices were added by running the following command:

```
$ fvctl -f /dev/null list-slices
```

You should now see both the `upper` and `lower` slices in addition to the default `fvadmin` slice, all enabled.

Create Flowspaces

Flowspaces associate packets of a particular type in the network to specific slices. Commands to create flowspaces (no need to type this one) would use the `add-flowspace` command:

```
$ fvctl add-flowspace [options] <flowspace-name> <dpid> <priority> <match> <slice-pe
```

- When a packet matches multiple flowspaces, FlowVisor assigns it to the flowspace with the highest `priority` number.
- `match` describes a flow or flows. Such flow descriptions comprise a series field=value assignments, separated by commas.
- `slice-perm` is a comma-separated list of slices that have control over a specific FlowSpace. `slice-perm` is of the form "slicename1=perm[slicename2=perm[...]]". Each slice can have three types of permissions over a flowspace: *DELEGATE*, *READ*, and *WRITE*. Permissions are a bitmask specified as an integer, with *DELEGATE*=1, *READ*=2, *WRITE*=4.

You can learn more about the command by typing:

```
$ fvctl add-flowspace -h
```

Now, create a flowspace named `dpid1-port1` (with priority value 1) that maps all the traffic on port 1 of switch s1 to the upper slice by running the following command:

```
$ fvctl -f /dev/null add-flowspace dpid1-port1 1 1 in_port=1 upper=7
```

Here we gave the upper slice all permissions: *DELEGATE*, *READ*, and *WRITE*.

Similarly, create a flowspace named `dpid1-port3` that maps all the traffic on port 3 of switch s1 to the upper slice:

```
$ fvctl -f /dev/null add-flowspace dpid1-port3 1 1 in_port=3 upper=7
```

We can create a flowspace for all the traffic at a switch by using the match value of `any`. Use that technique to add switch s2 to the upper slice:

```
$ fvctl -f /dev/null add-flowspace dpid2 2 1 any upper=7
```

Now, create flowspaces to add ports 1 and 3 of switch s4 to the upper slice:

```
$ fvctl -f /dev/null add-flowspace dpid4-port1 4 1 in_port=1 upper=7
$ fvctl -f /dev/null add-flowspace dpid4-port3 4 1 in_port=3 upper=7
```

Ensure that the flowspaces are correctly added:

```
$ fvctl -f /dev/null list-flowspace
```

You should see all the flowspaces (5 in all) that you just added.

Now, create flowspaces for the lower slice:

```
$ fvctl -f /dev/null add-flowspace dpid1-port2 1 1 in_port=2 lower=7
$ fvctl -f /dev/null add-flowspace dpid1-port4 1 1 in_port=4 lower=7
$ fvctl -f /dev/null add-flowspace dpid3 3 1 any lower=7
$ fvctl -f /dev/null add-flowspace dpid4-port2 4 1 in_port=2 lower=7
$ fvctl -f /dev/null add-flowspace dpid4-port4 4 1 in_port=4 lower=7
```

Ensure that the flowspaces are correctly added:

```
$ fvctl -f /dev/null list-flowspace
```

Connect each slice to a different instance of the FVexercise controller app. The FVexercise app reactively installs routes based on the destination MAC address, and it is provided as an executable. Open **two fresh terminal tabs** and run the following:

- Terminal 1

```
$ cd ~/onstutorial/beacon-fvexercise-controller1
$ ./beacon
```

- Terminal 2

```
$ cd ~/onstutorial/beacon-fvexercise-controller2
$ ./beacon
```

This will launch two instances of FVexercise Beacon controller, listening on port 10001 and 10002 respectively.

After a short delay, both controllers should connect. Verify that h1 can ping h3 but not h2 and h4 (and vice versa). In the Mininet console run the following commands (note: it takes some time for the first ping to go through, as a new route is being installed):

```
mininet> h1 ping -c1 h3
mininet> h1 ping -c1 -W1 h2
mininet> h1 ping -c1 -W1 h4
```

Verify that h2 can ping h4 but not h1 and h3 (and vice versa). In the mininet console run the following commands:

```
mininet> h2 ping -c1 h4
mininet> h2 ping -c1 -W1 h1
mininet> h2 ping -c1 -W1 h3
```

Cleanup

Remove both the slices that you created:

```
$ fvctl -f /dev/null remove-slice upper
$ fvctl -f /dev/null remove-slice lower
```

Ensure that all the slices are correctly removed:

```
$ fvctl -f /dev/null list-slices
```

You should see only the default `fvadmin` slice remaining. Once the slices are removed, the associated flowspaces are also automatically removed. Ensure that all the flowspaces were correctly removed:

```
$ fvctl -f /dev/null list-flowspace
```

Finally, kill the two instances of beacon controller you are running.

Quiz

- Could you have implemented this kind of slicing in a traditional network? Highlight the hidden text below for the answer.

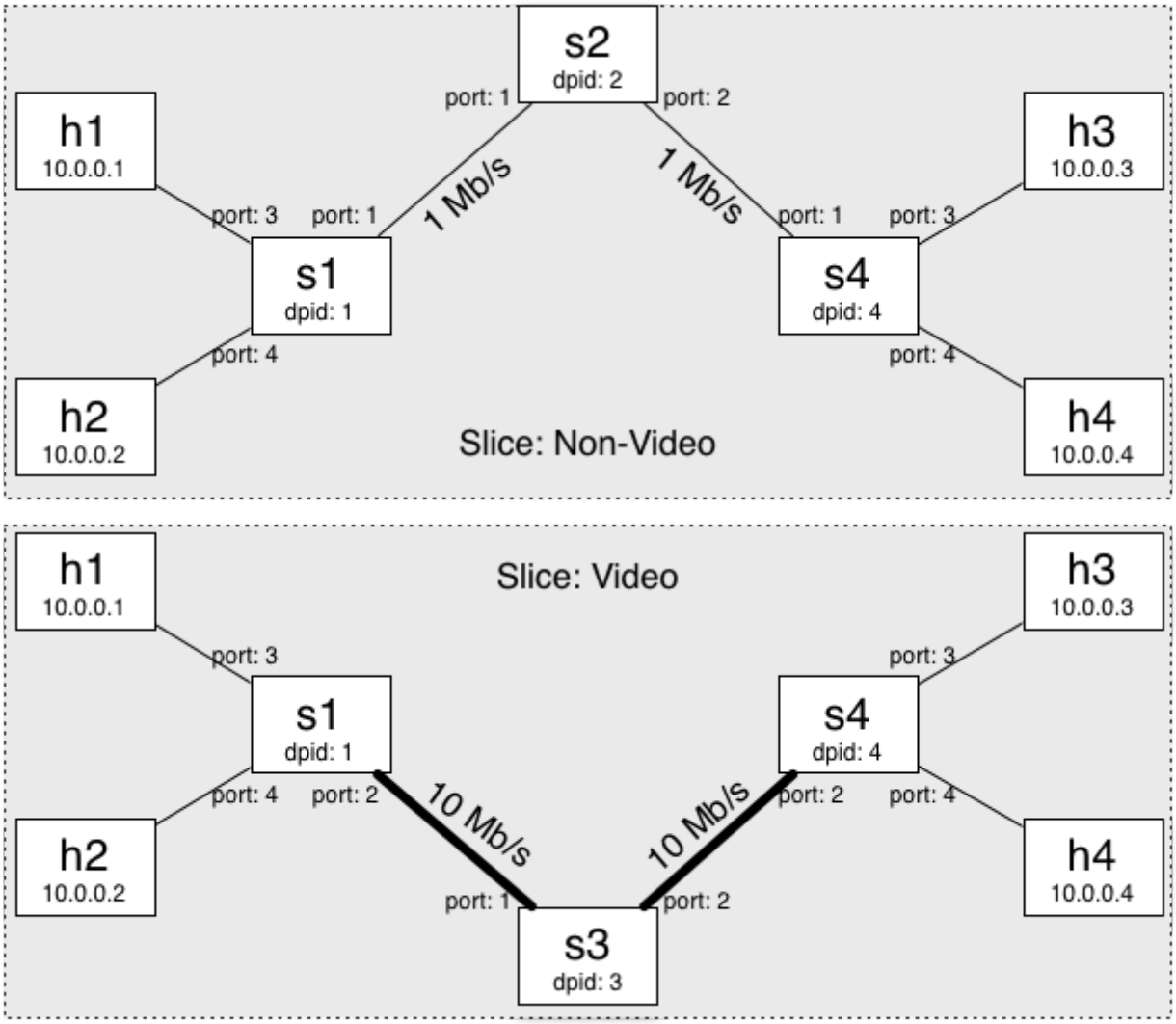
** Answer: Yes, with a VLAN! Port-based slicing is not so new.

Part2: Advanced Flowspace Slicing

The previous part of the exercise showed how we can slice an OpenFlow network based on the physical topology of the network. With the centralized visibility and "layerless-ness", we can slice the network in more interesting ways. At its core, slicing is simply delegating the control of flows to different slices. OpenFlow allows us to define flows in flexible ways. This allows us to slice our network in more interesting ways. We will explore one such way in this part of the exercise.

Go back to our WAN topology. We have two paths connecting our two sites: a high bandwidth path and a low bandwidth path. As administrators of our WAN, we decide that we want to give special treatment to video traffic in our network. In particular, we want to send all video traffic over the high bandwidth path, and send all the other traffic over the default low bandwidth path. This way, our Skype connections won't be impacted by the video traffic, and vice versa. We will see how we can achieve this using slicing.

To keep things simple, we will assume that all video traffic goes on TCP port 9999. We will create two slices, `video` and `non-video`, as shown below.



The video slice controls traffic on TCP port 9999. The non-video slice controls everything else. Note that the video slice controls the high-bandwidth part of network.

Create Slices

Just like the previous part, create two slices called `video` and `non-video` connecting to controllers listening at `tcp:localhost:10001` and `tcp:localhost:10002`, respectively:

```
$ fvctl -f /dev/null add-slice video tcp:localhost:10001 admin@videoslice
$ fvctl -f /dev/null add-slice non-video tcp:localhost:10002 admin@nonvideoslice
```

Leave the Slice password empty by hitting Enter when prompted.

Ensure that the slices were added by running the following command:

```
$ fvctl -f /dev/null list-slices
```

You should now see both the `video` and `non-video` slices in addition to the default `fvadmin` slice, all enabled.

Create Flowspaces

FlowSpace creation in this case is slightly more involved, especially at the edge ports where hosts connect to the switches, as the traffic there is split between the two slices. At the edge ports, the `video` slice needs to handle all the traffic with source or destination port of 9999, and the `non-video` slice needs to handle the rest. We will achieve it using different `priority` values.

Let's start with (switch s1, port 3) that connects s1 to h1. First, create two high priority flowspaces that match all traffic with source or destination port of 9999 at that port, and assign it to the `video` slice:

```
$ fvctl -f /dev/null add-flowspace dpid1-port3-video-src 1 100 in_port=3,dl_type=0x0
$ fvctl -f /dev/null add-flowspace dpid1-port3-video-dst 1 100 in_port=3,dl_type=0x0
```

We have arbitrarily chosen a high priority value of 100. Next, create a low-priority flowspace that matches all traffic at that port, and assign it to the `non-video` slice:

```
$ fvctl -f /dev/null add-flowspace dpid1-port3-non-video 1 1 in_port=3 non-video=7
```

This results in all the video traffic at (switch s1, port 3) being handled by the `video` slice, and all the non-video traffic at being handled by the `non-video` slice.

Similarly, create flowspaces to segregate video and non-video traffic at (switch s1, port 4):

```
$ fvctl -f /dev/null add-flowspace dpid1-port4-video-src 1 100 in_port=4,dl_type=0x0
$ fvctl -f /dev/null add-flowspace dpid1-port4-video-dst 1 100 in_port=4,dl_type=0x0
$ fvctl -f /dev/null add-flowspace dpid1-port4-non-video 1 1 in_port=4 non-video=7
```

Similarly, create flowspaces for the other two edge ports (at switch s4). Create flowspaces for (switch s4, port 3):

```
$ fvctl -f /dev/null add-flowspace dpid4-port3-video-src 4 100 in_port=3,dl_type=0x0
$ fvctl -f /dev/null add-flowspace dpid4-port3-video-dst 4 100 in_port=3,dl_type=0x0
$ fvctl -f /dev/null add-flowspace dpid4-port3-non-video 4 1 in_port=3 non-video=7
```

Create flowspaces for (switch s4, port 4):

```
$ fvctl -f /dev/null add-flowspace dpid4-port4-video-src 4 100 in_port=4,dl_type=0x0
$ fvctl -f /dev/null add-flowspace dpid4-port4-video-dst 4 100 in_port=4,dl_type=0x0
$ fvctl -f /dev/null add-flowspace dpid4-port4-non-video 4 1 in_port=4 non-video=7
```

We are now just left with the internal ports. Creating flowspaces for them is simple, as traffic at each internal port is exclusively handled by one slice. Create flowspaces for the internal ports just the way you did part 1 (remember to use the correct slice name).

Assign (switch s1, port 2) to `video`:

```
$ fvctl -f /dev/null add-flowspace dpid1-port2-video 1 100 in_port=2 video=7
```

Assign (switch s3, all ports) to `video`:

```
$ fvctl -f /dev/null add-flowspace dpid3-video 3 100 any video=7
```

Assign (switch s4, port 2) to `video`:

```
$ fvctl -f /dev/null add-flowspace dpid4-port2-video 4 100 in_port=2 video=7
```

Assign (switch s1, port 1) to `non-video`:

```
$ fvctl -f /dev/null add-flowspace dpid1-port1-non-video 1 1 in_port=1 non-video=7
```

Assign (switch s2, all ports) to `non-video`:

```
$ fvctl -f /dev/null add-flowspace dpid2-non-video 2 1 any non-video=7
```

Assign (switch s4, port 1) to `non-video`:

```
$ fvctl -f /dev/null add-flowspace dpid4-port1-non-video 4 1 in_port=1 non-video=7
```

Now, we are done with creating all the flowspaces. Ensure that the Flowspaces were correctly added:

```
$ fvctl -f /dev/null list-flowspace
```

You should see all the flowspaces that you just added (18 in all).

Connect both slices to two different instances of the FVExercise controller app, each running in a terminal window:

- Terminal 1

```
$ cd ~/onstutorial/beacon-fvexercise-controller1
$ ./beacon
```

- Terminal 2

```
$ cd ~/onstutorial/beacon-fvexercise-controller2
$ ./beacon
```

We will next verify that we correctly sliced our network. If we correctly sliced our network, a video flow (running on TCP port 9999) should go over the high bandwidth path and see a throughput of 10 Mb/s. A non-video flow (running on any other TCP port) should go over the low bandwidth path and see a throughput of 1 Mb/s.

Verify connectivity. In the Mininet console, run:

```
mininet> pingall
```

Note that the network has full connectivity now, as compared to before, where ports were isolated from each other.

Now, we will verify that video and non-video flows take the right paths. Open terminals for hosts h1 and h3. In the Mininet console run:

```
mininet> xterm h1 h3
```

Note: to copy text into an xterm, one way is to click the middle mouse button. In the popped up terminal of "Node: h1" run:

```
# iperf -s -p 9999
```

Start a video flow by running the following command in the popped up terminal of node h3:

```
# iperf -c 10.0.0.1 -p 9999 -t 10 -i 1
```

You should see a throughput of roughly around 10 Mb/s. You can run this same command in h4, and get the same result, whereas before, there was no connectivity.

Next open terminals for hosts h2 and h4. In the Mininet console run:

```
mininet> xterm h2 h4
```

In the popped up terminal of "Node: h2" run:

```
# iperf -s -p 8888
```

Start a non-video flow by running the following command in the popped up terminal of node h4:

```
# iperf -c 10.0.0.2 -p 8888 -t 10 -i 1
```

You should see a throughput of roughly around 1 Mb/s.

Cleanup

Stop both the controllers.

Remove both the slices that you created:

```
$ fvctl -f /dev/null remove-slice video
$ fvctl -f /dev/null remove-slice non-video
```

Ensure that all the slices are correctly removed:

```
$ fvctl -f /dev/null list-slices
```

You should see only the default `fvadmin` slice remaining. Once the slices are removed, the associated flowspaces too are automatically removed. Ensure that all the flowspaces are correctly removed:

```
$ fvctl -f /dev/null list-flowspace
```

Quiz

Highlight the hidden text below each question for its answer.

- Could this division of the network have been implemented on a more traditional network?
- Short answer: Of course, SDN doesn't do anything you couldn't do before.
- Long answer: In this case, an administrator might manually define access control rules to direct traffic to the video server or video port along a non-default path, logging into each switch. The difference is that with SDN, and in particular FlowVisor, those policies can be more flexible, as the controller for the video slice can make more dynamic traffic-routing and prioritization decisions for more dynamically defined slices of traffic.
- If static ARP entries were not set, which slice would have handled the ARP traffic? (Hint: The way the flowvisor handles ARPs is that it takes an ARP request "who has $ip? tell $mac" and sends that to the slice that has flowspace for nw_dst=$ip.)

** Answer: The ARP traffic would still be handled by the non-video slice.

Takeaway

Recall that throughout the exercise we used the same topology and the same controller. By just changing the way we sliced our network, we completely altered the behavior of the network. This is an example of the power of the flexibility and generality of OpenFlow.

Since our network slicer is a program, with an API, we could even integrate the slicing functionality with server orchestration tools, or implement a higher-level language to define our policies.

Congratulations! You have successfully completed the Slicing exercise.

7: Credits

Clone this wiki locally

```
https://github.com/onstutorial/onstutorial.wiki.git
```