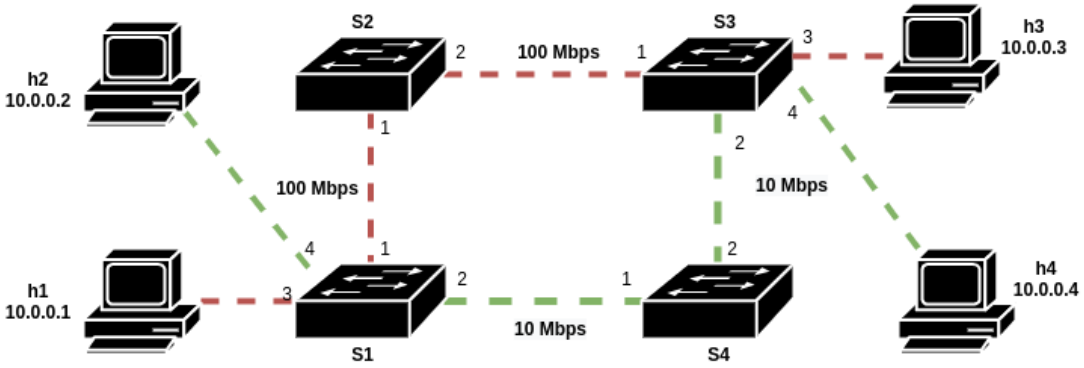
Practice 3: Introduction to network virtualization

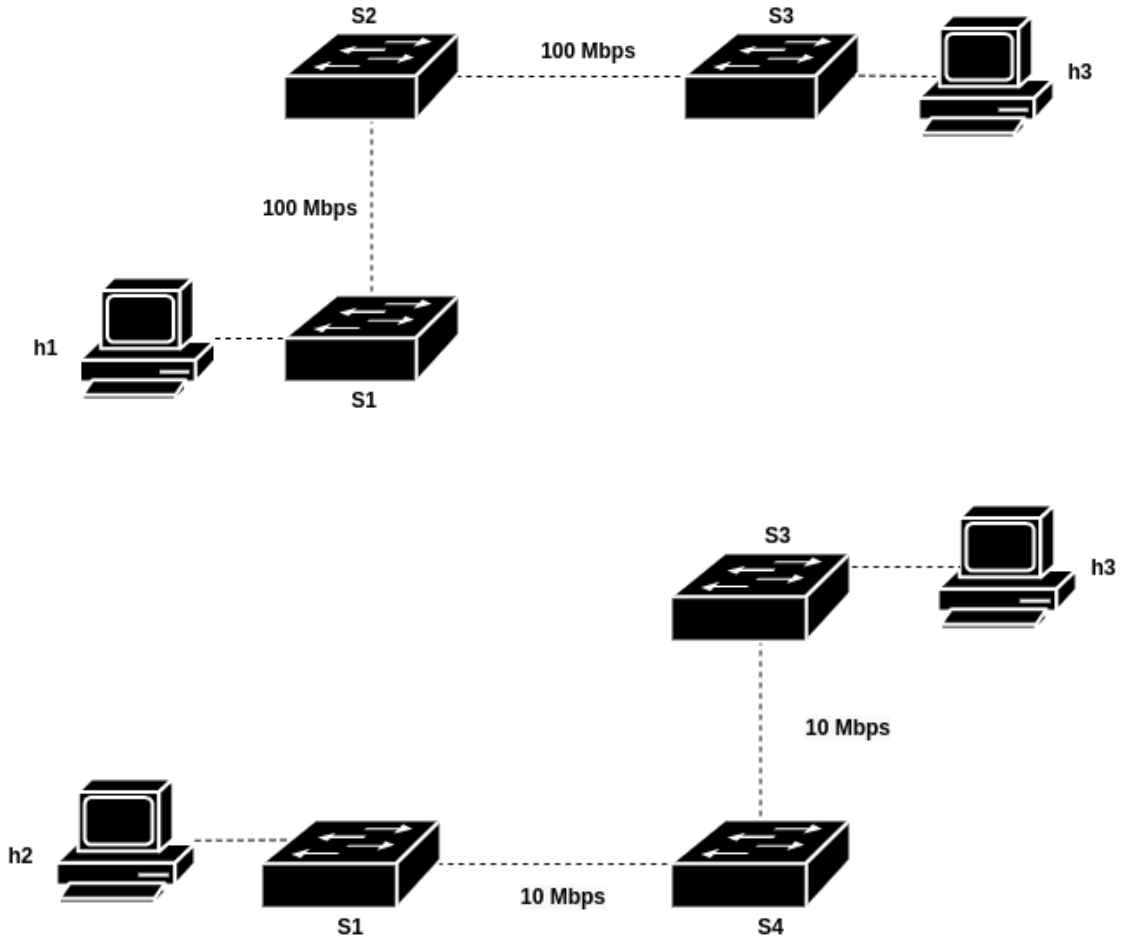
In this practice, we will continue to explore the capabilities of the POX controller. In this exercise, you will use POX to segment an OpenFlow network in order to have a first approach to the concept of network virtualization. There are tools such as FlowVisor, which allows the operation of several controllers over different portions of the network. These tools normally have to be installed and the network operator has to learn how to use a completely new configuration environment.

In this case, the task to be developed is quite simple, using POX you will create an application that creates multiple network slices (in layer 2) and each one will be used to carry flows of particular applications. Again, we recommend that you test your application on your virtual machine installation at home and then submit it for assessment.

To start, we propose an activity where you must split an initial topology into two paths: one of high capacity and one of low capacity. In this case, you will use the following topology in Mininet:



You must configure in POX an application that allows you to split the network in one path with a capacity of 100 Mbps and another path with a capacity of 10 Mbps:



Users of different network slices will not be able to communicate with each other. This could be similar to the situation where an operator wants to isolate a part of the topology that a particular customer will use or to the VLAN implementation in the traditional networks.

For this, we will give you a file (Topo.py) that will contain different topologies that we will use in the following labs. Each class in this file represents a topology that can be invoked from the key in the dictionary at the end of the script:

```
topos = { 'p3-2':  CustomTopo, 'p3-1': TopoSimple, 'p4-1': TopoP4,
'p4-2':RandomTopo }
```

If you want to start a Mininet instance with the topology of the aforementioned exercise you must use the command:

```
sudo mn --custom Topo.py --topo p3-1 --controller remote,port=6633
--link tc
```

The --topo option allows loading the topology associated with the class used in the file to create the network.

You can use the file topologySlice.py to put your code and try your solution. In this second file, you will use the **_handle_ConnectionUp()** method, which will be activated every time that a switch connects to the POX controller. Knowing the topology and the ports used to connect switches and hosts (configured in the Mininet file), you must construct the OpenFlow messages that will be sent to the switches in order to achieve the network division.

In this situation, there are a few things to keep in mind: The topology has closed loops, so processes such as ARP will generate broadcast storms, these events are not desirable and must be prevented through the construction of a Spanning Tree.

POX includes a module that builds a spanning tree over a topology using another module that takes advantage of LLDP(Link Layer Discovery Protocol) messages to detect how OpenFlow switches are connected. In this case, within the skeleton that we provide, in the launch() method we include both applications:

```
def launch():
    pox.openflow.discovery.launch()
    pox.openflow.spanning_tree.launch()
```

To perform the exercise, you must open two consoles, in the first one we will start Mininet with the designed topology, as specified before, after that, you must run your network application:

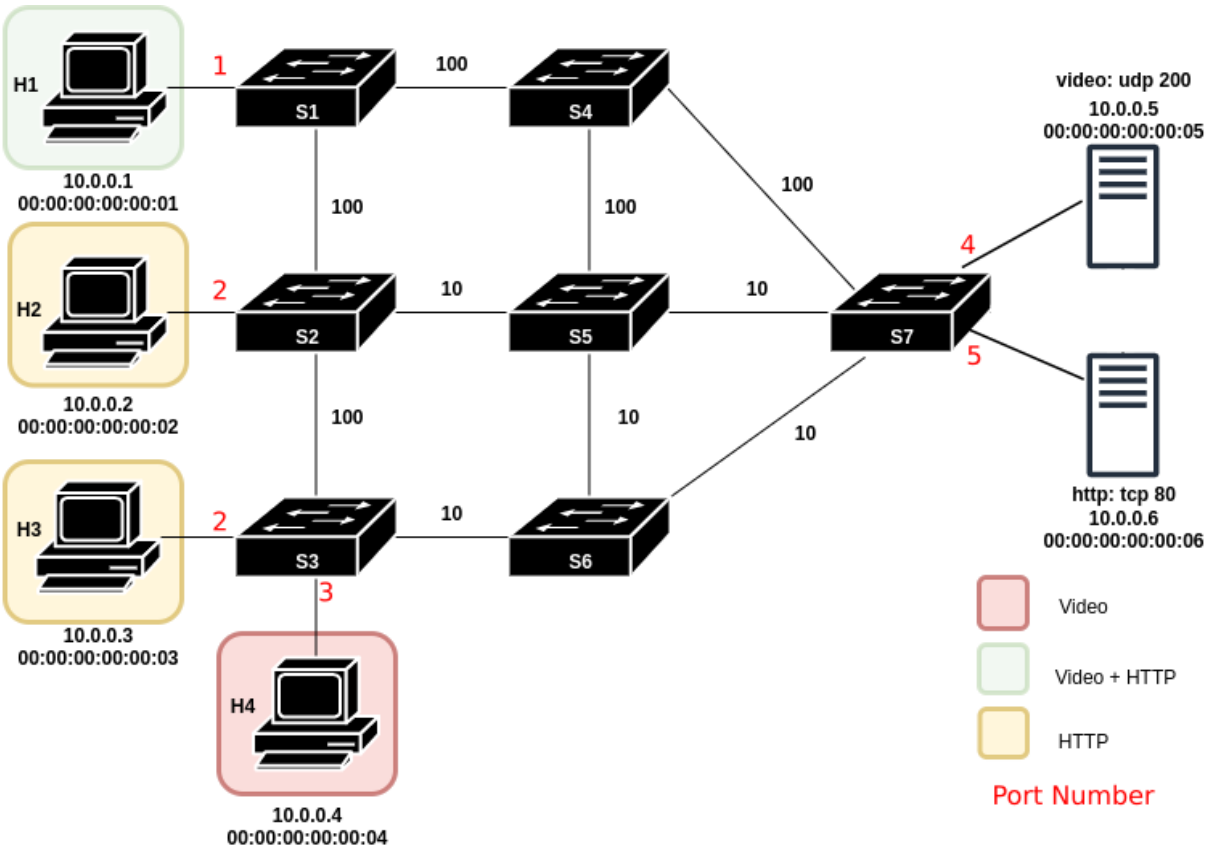
```
sudo ./pox.py log.level --DEBUG pox.misc.topologySlice
```

Activity

The previous activity consisted in performing a division based on the physical distribution of the devices. Now, we will perform a division based on the application that is used. Additionally, we will limit the hosts' access to applications. We will make use of the Discovery module in order to know (in the controller) the ports through which the switches are connected, using the _handle_LinkEvent() method. The_handle_PacketIn() method will be used to send an OpenFlow instruction when a switch sends a message for which it does not have an entry in its OF table.

The topology has 10 and 100 Mbps links and two servers, the first one, running the video-on-demand service, will use port 200 (UDP) and the second one will be a web service. Ports from switches to hosts are indicated in red. The video service must

always use a portion that has only 100 Mbps links, while the web service must transit through 10 Mbps links. The hosts have access to the services, as shown in the following figure:



Again, you will use two files, the first one (Topo.py) to run the topology and the file Skeleton.py, to build your POX app. To run the topology you can use the following command:

```
sudo mn --custom Topo.py --topo p3-2 --controller remote,port=6633
--link tc
```

After finishing your code, verify that all the hosts are able to communicate with each other. This is a simple sanity check to make sure that your logic is not affecting connectivity in general.

```
mininet> pingall
```

You should see connectivity among all hosts.

You can now test that your slices work properly by ensuring that video (in this case, port 200) traffic traverses the 100 Mbps link and HTTP (port 80) traffic traverses the 10 Mbps links. For example, you can test the two paths between h1-h5 and h3-h6 as below. (Your code should work for all pairwise paths.)

```
mininet> h5 iperf -s -p 200 &
------------------------------------------------------------
Server listening on TCP port 200
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
mininet> h6 iperf -s -p 80 &
------------------------------------------------------------
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
mininet> h1 iperf -c h5 -p 200 -t 2 -i 1
------------------------------------------------------------
Client connecting to 10.0.0.5, TCP port 200
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 3] local 10.0.0.1 port 41320 connected with 10.0.0.5 port 200
[ ID] IntervalTransferBandwidth
[3]0.0- 1.0 sec11.9 MBytes99.6 Mbits/sec
[3]1.0- 2.0 sec10.8 MBytes90.2 Mbits/sec
[3]0.0- 2.0 sec22.8 MBytes95.2 Mbits/sec
mininet> h3 iperf -c h6 -p 80 -t 2 -i 1
```

```
------------------------------------------------------------
Client connecting to 10.0.0.6, TCP port 80
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[3] local 10.0.0.3 port 57070 connected with 10.0.0.6 port 80
[ID] IntervalTransferBandwidth
[3]0.0- 1.0 sec1.62 MBytes13.6 Mbits/sec
[3]1.0- 2.0 sec1.25 MBytes10.5 Mbits/sec
[3]0.0- 2.0 sec3.00 MBytes12.3 Mbits/sec
```