CS3506 - Lab 5: SDN.

In this lab, we will introduce you to Software Defined Networking (SDN) using the open-source Mininet emulator and Ryu controller. We will also use the miniNAM network visualisation utility for Mininet, thanks primarily to UCC Ph.D. student Ahmed Khalid.

The purpose of this lab is to familiarise student with the basic concepts of SDN:

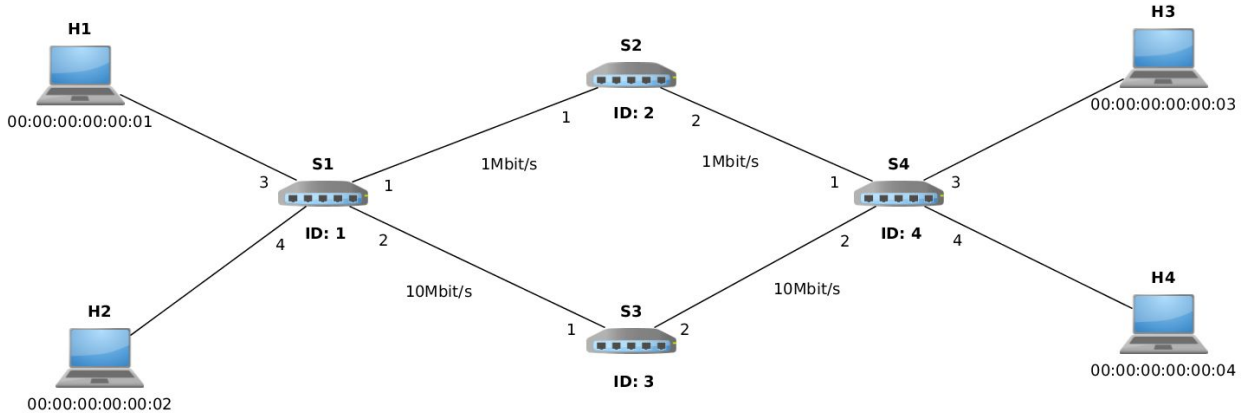
- Programming the controller logic (Ryu in this case)
- Configure each switch through OpenFlow protocol
- Test the new configuration

➔ Start the Windows and log in with your account
➔ Open a VirtualBox; go to File Import Appliance;
➔ Go to **C:\CS3506** and select **SDN-CS3506** image (importing image will take 2-3 minutes)
➔ Start the **SDN-CS3506** virtual machine

Link for downloading image (if you want to run locally on your own machine): SDN-CS3506_Lab_5

Log in with:
Username: *mn*
Password: *sdn*

We'll use the network shown in the figure below to experiment with network slices. Slices could be employed, for example, to share a network between different companies or ensure isolation between the various traffic types. Note that the Controller is not shown in the Figure, but is assumed to be able to communicate directly with each switch using the OpenFlow protocol.



We'll use a Mininet script, **mininetSlice.py**, to create a network with the topology provided above. Open a terminal (**ctrl-T**). Type commands:

cd ~/lab5;ls

You should see four files: **MiniNAM.py**, **mininetSlice.py**, **topoSlice.py** and **videoSlice.py**.

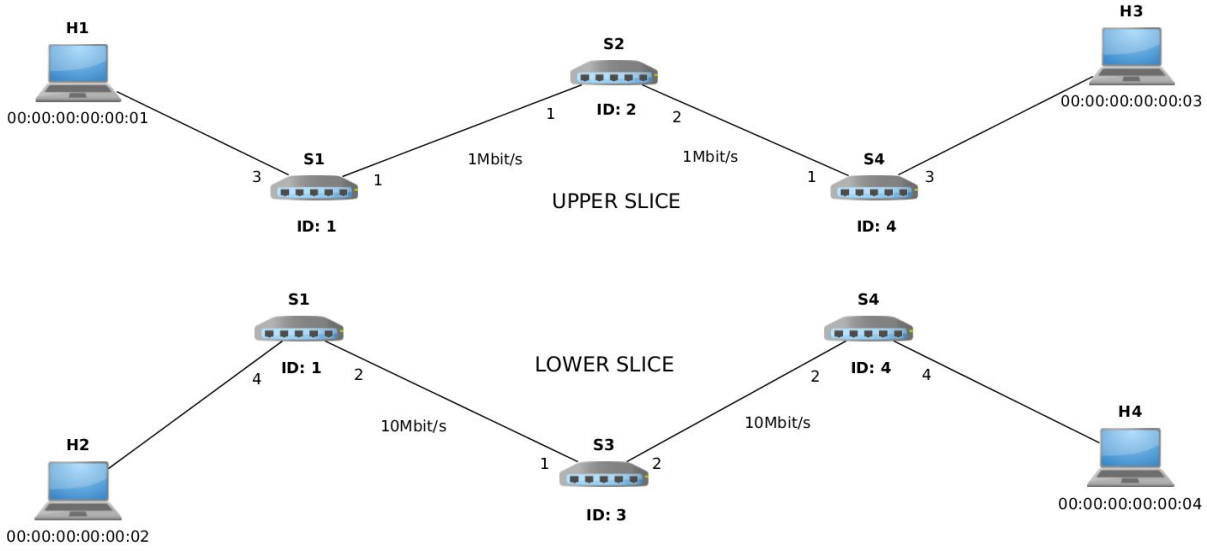
Run the mininetSlice.py script:

sudo python MiniNAM.py --custom mininetSlice.py

The command will create the network. A new window will open, and you should see network topology as demonstrated above.

- Now answer questions 1.1-1.3

Next, we need to divide the network into two slices as shown in figures below.



Hosts in different slices should not be able to communicate with each other. E.g., H1 should not reach H2 and H4. Because we are using an OpenFlow-enabled network, we can install flow rules into each switch and configure them appropriately.
When running mininetSlice.py script, our switches are 'empty.' To add flow rules, we will use Ryu controller (https://osrg.github.io/ryu/).

Open a file **topoSlice.py**.

In **topoSlice.py** file we have class **TopoSlice** with three methods: ***__init__***, ***switch_features_handler***, ***_packet_in_handler*** and ***add_flow***.

For our lab, we will use ***switch_features_handler*** and ***add_flow*** methods.

Method ***switch_features_handler*** is called at the beginning when the switch for the first time connects with the controller. We will leverage this process and install flow rules for each switch.

Each switch is identified by its ID (stored as dpid). Add logic to the method ***switch_features_handler*** writing OpenFlow rules that provide this isolation. Logic is already implemented for switches S2 and S3.

To test your implementation, open a new terminal and navigate to a folder where the scripts are stored. Type command to start a controller:

ryu-manager topoSlice.py

- Now answer questions 1.4-1.6

For the second part, we will implement slicing based on the type of application that is sending traffic. Let's assume that we want to send video traffic through high bandwidth links (lower slice) while all other traffic should be forwarded through the upper slice. We consider video traffic to be all the traffic that uses TCP and destination port 80. We are aware that all HTTP traffic uses destination port 80, but for simplicity in our lab, we will assume that the traffic is video.
Open a **videoSlice.py**. Go through **__init__** and **switch_features_handler** method. Implement logic for forwarding traffic. Test your implementation by running controller:

ryu-manager videoSlice.py

- Now answer questions 1.7-1.11

Submission - LAB 5: SDN

Please submit during this lab session or else at 4pm Thursday next week.

Student name: _____ ***Student ID:*** _____

Submission Date: _____

1.1 Looking at the output of the MininetSlice.py (in terminal) did the switches connect to a controller? Write down a line from the output.

1.2. On which port is the controller listening for the connection from the switches (**HINT:** look at the code of MininetSlice.py and https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#api)?

1.3. Run *pingall* inside mininet environment (*mininet> pingall*). How many packets are dropped? How many are sent?

1.4. Write down code for the flow rules at the S1

1.5. Write down code for the flow rules at the S4

1.6. Run *pingall* inside mininet environment (*mininet> pingall*). How many packets are dropped? How many are sent?

1.7. Write down code for the upper slice.

1.8. Write down code for the lower slice.

1.9 In mininet environment run these two commands: h3 iperf -s -p 80 & and h2 iperf -c h3 -p 80 -t 2 -i 1. Write down the output.

1.10 In mininet environment run these two commands: h3 iperf -s -p 22 & and h2 iperf -c h3 -p 22 -t 2 -i 1. Write down the output. (*iperf* is a network testing tool, usually used for generating and sending TCP or UDP packets. More information can be found at https://iperf.fr/iperf-doc.php)

1.11. Run *pingall* inside mininet environment (*mininet> pingall*). How many packets are dropped? How many are sent? Explain. (**HINT:** What rules are missing?)