Lab – Hands-on Experience with SDN Tools and Applications

Problem 1: Discover Mininet

In this problem, we will run **Mininet** for the first time and see some of the commands to interact with it.

Mininet is a network emulator that can create a virtual network consisting of virtual hosts, virtual switches and virtual links connecting them. **Mininet** runs on Linux, and all the virtual hosts share the same filesystem as the computer running **Mininet**, and as a result, can run any program that is installed on the host.

The **Mininet** virtual switches are by default **Open vSwitch** SDN switches that are **OpenFlow** enabled switches that can be managed using version 1.0 of **OpenFlow**.

First, let's start **Mininet** using the following command in Putty:

- **sudo mn**

After launching this command, Mininet will create a default topology and then will provide a command-line interface. We can get information about this topology using the following commands:

- **nodes**: gives the different nodes (hosts, switches, and controllers) emulated in Mininet.
- **links**: gives the different links between nodes emulated in Mininet.
- **net**: gives for each node the different network interfaces and to which interface of other nodes they are connected.

Questions:

1. How many nodes are there in the default topology?
2. How many switches are there in the default topology? Is there a controller for those switches?
3. Sketch the topology of the default network emulated by Mininet.

We can launch any Linux command from any host or switch, for that, we need to prepend the command by the name of the host/switch:

- Launch an ifconfig on h1 using the following command: **h1 ifconfig**
- Launch a ping from h1 to h2 using the following command: **h1 ping h2**
- Launch **wireshark** on the switch s1 using the following command: **s1 wireshark &**

You can exit Mininet using the **exit** command.

Problem 2: Manual configuration of the switches

In this problem, we will run Mininet on a second topology without a controller. We will then manually add flow rules in the switches in order to handle the traffic:

- Launch Mininet using the following command:

sudo mn --controller=none --topo=single,3 --mac

Questions:

4. Sketch the topology of the emulated network
5. Perform a ping from host h1 to host h2. Does it work? Explain.

We will now configure the switch s1 in order to have connectivity between h1 and h2.

The Open vSwitch switches emulated by Mininet have instead of an IP routing table a flow table containing a list of flows that are matching rules associated to an action. The matching rules can match on several fields from layer 1 to layer 4:

- Layer 1: for example, the input and output ports of the switch
- Layer 2: for example, the source and destination Ethernet MAC addresses
- Layer 3: for example, the source and destination IP addresses
- Layer 4: for example, the source and destination TCP ports

When a rule matches a packet, the action specified in the rule is applied to this packet. These are some of the actions that are possible:

- Flood i.e., broadcast the packet to all the ports of the switch except the port where the packet came from
- Forward the packet to a specific port
- Drop the packet
- Send the packet to the controller for further processing
- Perform the normal IP processing on the packet (i.e., check the IP routing table)

In order to configure the flow table of the switch s1, we will use the **ovs-ofctl** command:

- Launch the following command: **sh ovs-ofctl dump-flows s1**

This command will display the content of the flow table of the switch s1.

Questions:

6. What is in the flow table of s1? (provide a screenshot)
7. Why is the flow table of s1 empty?

Let's now add flows to the flow table of s1 in order to permit bidirectional communication between h1 and h2. For this, we will use the **ovs-ofctl add-flow** command. The syntax of this command is as follows:

sh ovs-ofctl add-flow <switch_name> <matching criteria>,actions=<action>

The **matching criteria** parameter is a comma-separated list of statements of the type **<header field>=<value>**. You can get the complete list of possible header fields by typing the command **sh man ovs-ofctl** and reading the **Flow Syntax** section.

Question:

8. Cite three header fields that can be used on the **ovs-ofctl add-flow** command and explain what field of which protocol they check.

The action parameter can take the following values (we present here only the actions that will be used in this lab, for a complete list of the actions, please refer to the man page of ovs-ofctl):

- **flood**: to flood the packet
- **output,<port number>**: to forward the packet on the specified output port number
- **drop**: to drop the packet

Let's execute the following commands:

- **sh ovs-ofctl add-flow s1 in_port=1,actions=output:2**
- **sh ovs-ofctl add-flow s1 in_port=2,actions=output:1**

Questions:

9. Explain what is done by the flows we added using the two previous commands.
10. Perform a ping from h1 to h2. Does it work? (provide a screenshot)

Problem 3: Manual configuration of a layer 2 forwarding

In this problem, we will manually configure the switch in order to perform a layer 2 forwarding (i.e., based on the MAC addresses).

In order to do a layer 2 forwarding on switch s1, we will add flows that will match on the destination MAC address, and then forward the packet to the appropriate output port.

Provide the list of commands you ran to configure the requested layer 2 forwarding behavior, as well as your test methodology and results.

Problem 4: Manual configuration of a layer 3 forwarding

In this problem, we will manually configure the switch in order to perform a layer 3 forwarding (i.e., based on the IP addresses).

In order to do a layer 3 forwarding on switch s1, we will add flows that will match on the destination IP address, and then forward the packet to the appropriate output port.

Provide the list of commands you ran to configure the requested layer 3 forwarding behavior, as well as your test methodology and results.

Problem 5: Manual configuration of a layer 4 forwarding

In this problem, we want to configure the switch in order to have the HTTP traffic routed towards host h3.

Provide the list of commands you ran to configure the requested layer 4 forwarding behavior, as well as your test methodology and results.

In order to test that the switch has now the required behavior, you can start a web server on h3 and do a **wget** from h1:

- **h3 python –m SimpleHTTPServer 80 &**
- **h1 wget h3**

Problem 6: Manual configuration of a layer 3 forwarding with a firewall

In this problem, we want to configure the switch in order to have a layer 3 forwarding (similarly to what was done in problem 4), but we want the HTTP traffic destined to host h3 to be blocked, while the HTTP traffic destined to hosts h1 and h2 stays allowed.

Provide the list of commands you ran to configure the requested behavior, as well as your test methodology and results.

Problem 7: Using a remote controller

In this problem, we will connect the switches emulated by Mininet to a remote controller called POX, and examine how it works.

First, let's run the following commands to launch the layer 2 learning application of POX:

- **cd /home/mininet/pox**
- **./pox.py log.level -DEBUG forwarding.l2_learning**

Open then another terminal to the Mininet VM and create the simple topology:

sudo mn --controller=remote --topo=single,3 --mac

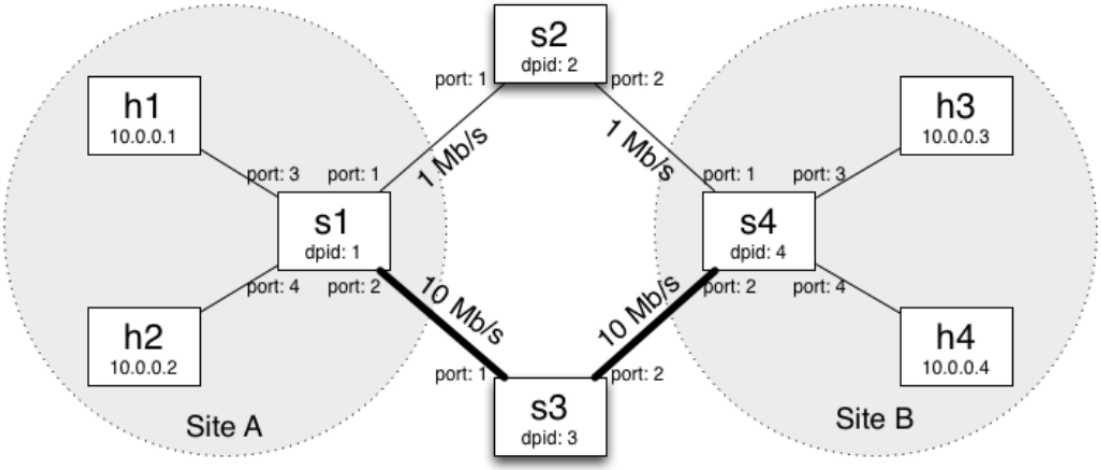
Notice that we have specified the parameter **--controller=remote** that tells Mininet to connect to a remote controller. By default, Mininet will try to connect to a controller listening on ports 6653 or 6633 in localhost.

Run **wireshark** on the switch and run a packet capture while testing the connectivity between the hosts. Describe the different OpenFlow packets you see and explain their role.

By looking at the code of the **l2_learning.py** controller application, explain how the controller algorithm works, and what network behavior it implements.

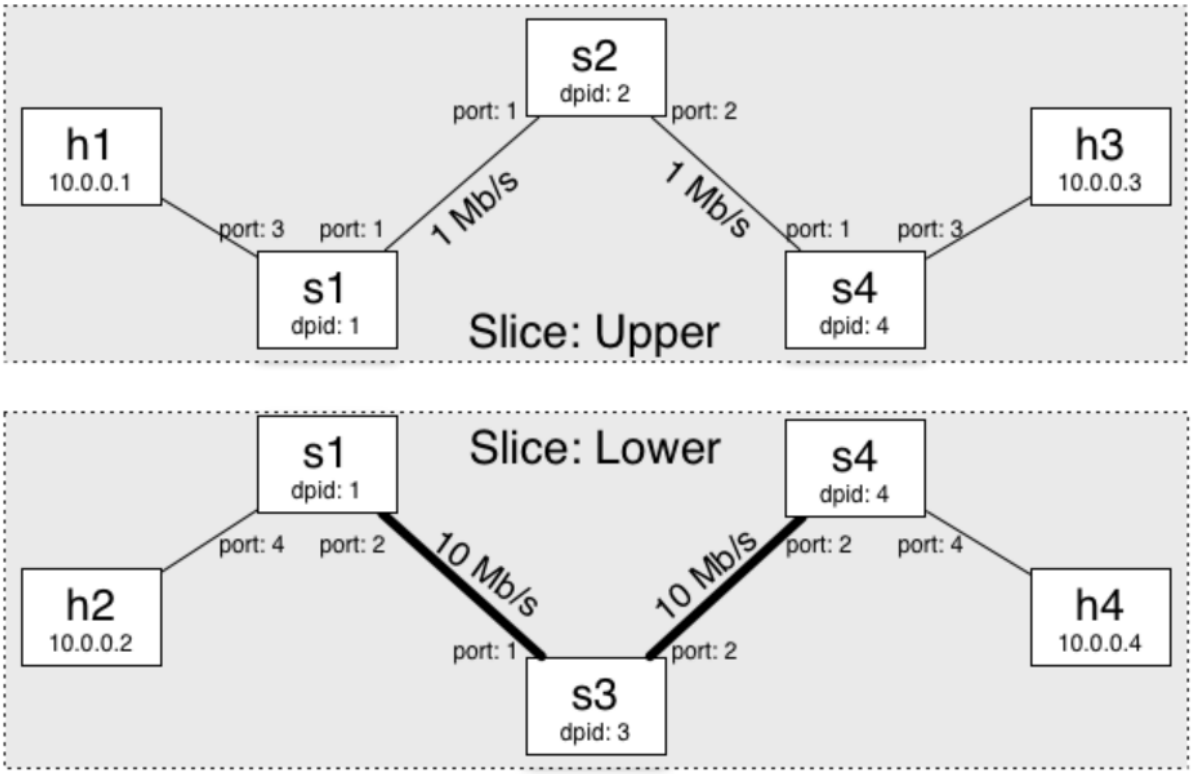
Problem 8: Create a custom topology

Using the Mininet documentation, create a python script that will configure Mininet to emulate the following topology:



Problem 9: Network slicing using flowvisor

Using flowvisor, run the custom topology created in the previous problem, and perform topology slicing as follows:



Each slice must be managed by a separate controller (you can use the layer 2 learning application of POX).

Provide the list of commands you ran to configure the requested behavior, as well as your test methodology and results.

Problem 10: Custom network behavior

Using everything you learned in the previous problems, choose a custom network behavior (e.g., load balancer) and implement it on **Mininet**. You can create a small POX network application that will dynamically configure the switched.

Describe the network behavior you implemented, provide the list of commands and scripts you used to configure it and explain your test methodology and results.