SOFTWARE DEFINED NETWORKING

LAB 3: MANUALLY ADDING FORWARDING RULES FOR END TO END CONNECTIVITY

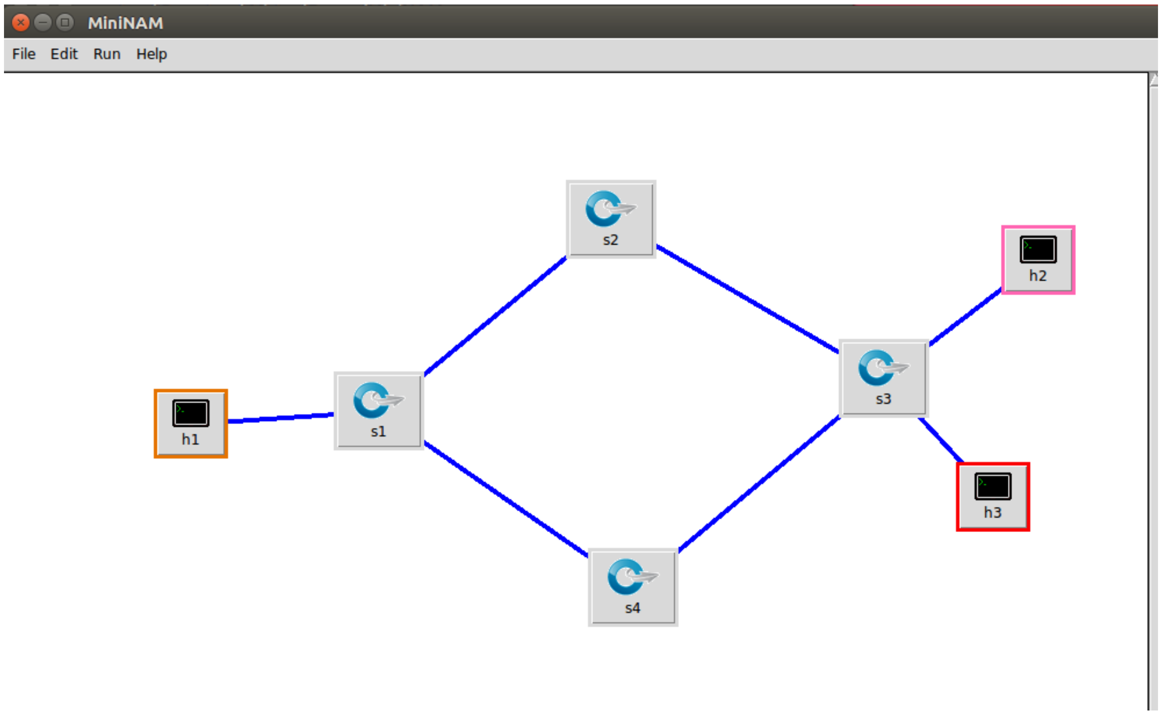
In last two labs we learnt how to develop a Mininet topology using python script and to apply bandwidth constraint to limit the bandwidth between the links. In this lab, we will learn various Mininet CLI commands to manually enter the flows and path in the topology. We won't be using Ryu controller in this lab since we will be manually adding the flow entries at the CLI whereas the controller will dynamically assign the path and solve the STP problem.
In this lab, we are going to use "ovs-ofctl" which is a command line tool for monitoring and administering OpenFlow switches. Even if OVS isn't configured for centralized administration, ovs-ofctl can be used to show the current state of OVS including features, configuration, and table entries.
Below are common show commands:

sh ovs-ofctl show <bridge> : Shows OpenFlow features and port descriptions.
sh ovs-ofctl snoop <bridge> : Snoops traffic to and from the bridge and prints to console.
sh ovs-ofctl dump-flows <bridge> <flow> : Prints flow entries of specified bridge. With the flow specified, only the matching flow will be printed to console. If the flow is omitted, all flow entries of the bridge will be printed.
sh ovs-ofctl dump-ports-desc <bridge> : Prints port statistics. This will show detailed information about interfaces in this bridge, include the state, peer, and speed information. Very useful
sh ovs-ofctl dump-tables-desc <bridge> : Similar to above but prints the descriptions of tables belonging to the stated bridge. ovs-ofctl dump-ports-desc is useful for viewing port connectivity. This is useful in detecting errors in your NIC to bridge bonding.

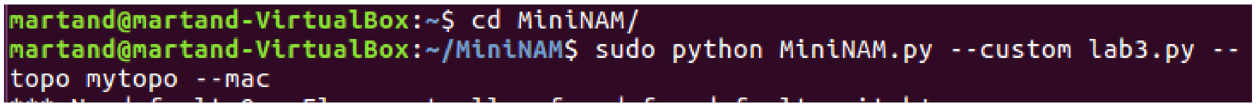
Below are the common configurations used with the ovs-ofctl tool:
sh ovs-ofctl add-flow <bridge> <flow> : Add a static flow to the specified bridge. Useful in defining conditions for a flow (i.e. prioritize, drop, etc).
sh ovs-ofctl del-flows <bridge> <flow> : Delete the flow entries from flow table of stated bridge. If the flow is omitted, all flows in specified bridge will be deleted.

The above commands can take many arguments regarding different field to match. They can be used for simple source/destination flow additions to complex L3 rewriting you can even build a functional router with them.

Step 1: Using the sample code attached with this lab file <code.py> create the topology give below by adding more switches and links.

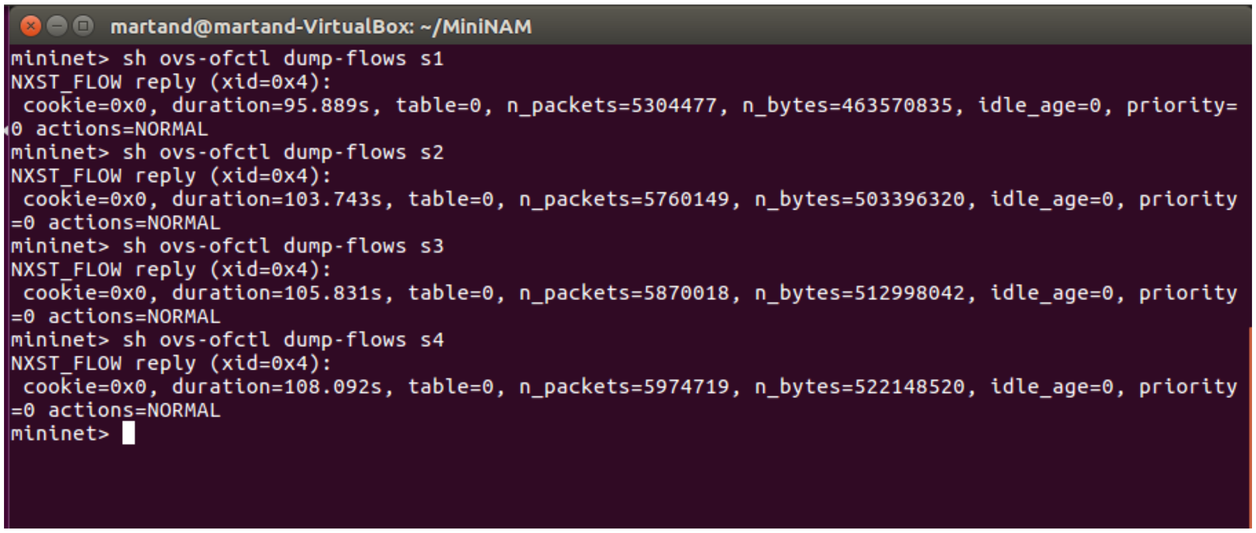


Step 2: Run the topology using the custom topology command given below. We don't need the ryu-controller like mentioned above. Please make sure you pass the **<--mac>** command which will dynamically assign the mac address for the hosts starting with 00:00:00:00:01 and IP address starting with 10.0.0.1. (**Paste the screenshot of it**)



```
martand@martand-VirtualBox:~$ cd MiniNAM/
martand@martand-VirtualBox:~/MiniNAM$ sudo python MiniNAM.py --custom lab3.py --
topo mytopo --mac
```

Step 3: Use <pingall> command to check connectivity between the hosts the ping should fail. Now using the ovs-ofctl command check if any flows are present. It should give you the table shown below which is not flow entry but just stating that the port is alive, and it keeps forwarding control packets which we cannot stop since we are not using controller with STP. Once we manually add the flow entries in the next step, we can view the flows using same command.

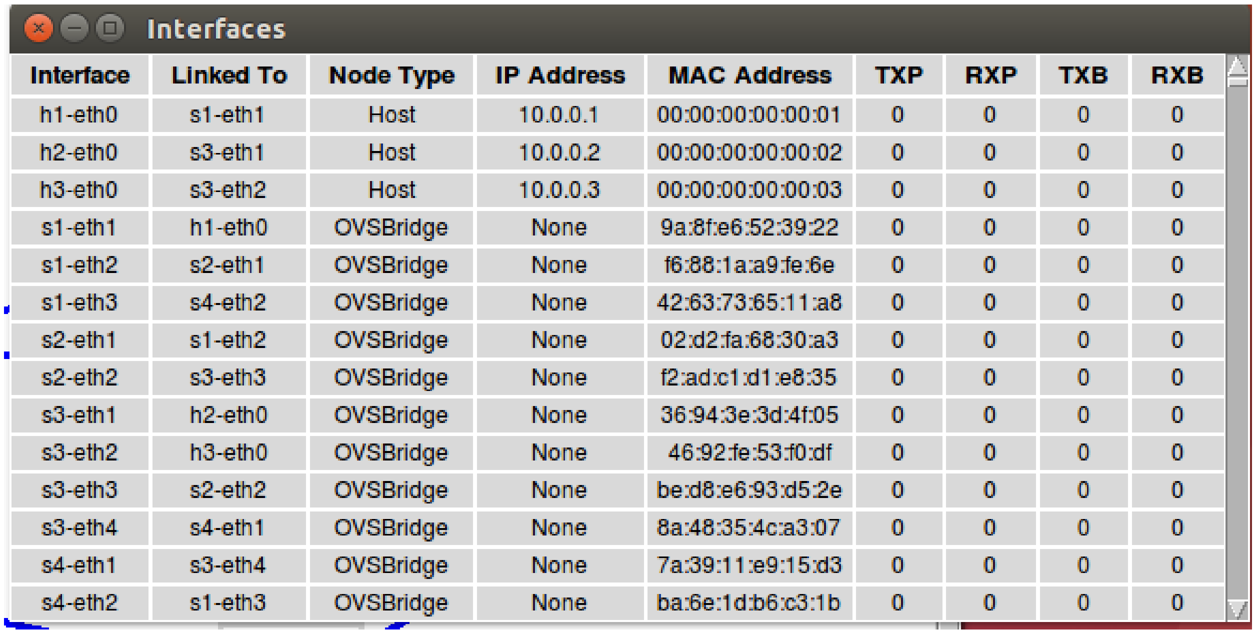


```
martand@martand-VirtualBox: ~/MiniNAM
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=95.889s, table=0, n_packets=5304477, n_bytes=463570835, idle_age=0, priority=
0 actions=NORMAL
mininet> sh ovs-ofctl dump-flows s2
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=103.743s, table=0, n_packets=5760149, n_bytes=503396320, idle_age=0, priority
=0 actions=NORMAL
mininet> sh ovs-ofctl dump-flows s3
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=105.831s, table=0, n_packets=5870018, n_bytes=512998042, idle_age=0, priority
=0 actions=NORMAL
mininet> sh ovs-ofctl dump-flows s4
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=108.092s, table=0, n_packets=5974719, n_bytes=522148520, idle_age=0, priority
=0 actions=NORMAL
mininet> 
```

Step 4: Now we will manually add the flow entries from H1 to H2 and H1 to H3. Let's understand the command used for adding flows manually. The syntax is as follows:

sh ovs-ofctl add-flow <switch_name> priority=<value>,in_port=<port_no>,actions=output:<port_no>

sh : Used for running commands from the system shell and not the Mininet CLI.
ovs-ofctl: Described above.
Priority: Optional. In flow table, flows are added in ascending order of the priority. Thus, assigning higher priority will add the flow above the flow with the lower priority and higher priority flows are executed first.
in_port: Port from which traffic is entering the switch. To check port number, you can go to the MiniNAM topology window. Under <Run> tab, check OVS interface summary, you will get following window.

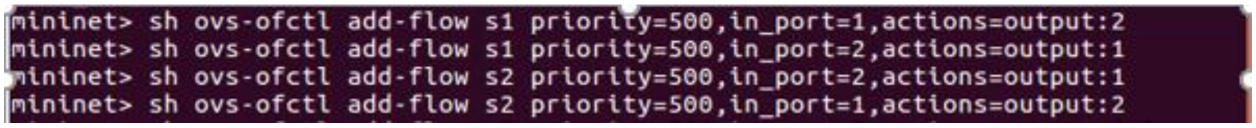


Interfaces

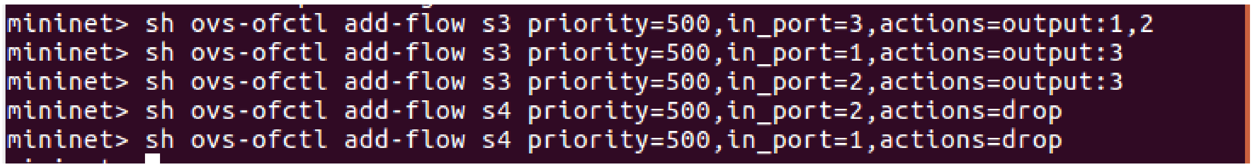
Interface	Linked To	Node Type	IP Address	MAC Address	TXP	RXP	TXB	RXB
h1-eth0	s1-eth1	Host	10.0.0.1	00:00:00:00:00:01	0	0	0	0
h2-eth0	s3-eth1	Host	10.0.0.2	00:00:00:00:00:02	0	0	0	0
h3-eth0	s3-eth2	Host	10.0.0.3	00:00:00:00:00:03	0	0	0	0
s1-eth1	h1-eth0	OVSBridge	None	9a:8f:e6:52:39:22	0	0	0	0
s1-eth2	s2-eth1	OVSBridge	None	f6:88:1a:a9:fe:6e	0	0	0	0
s1-eth3	s4-eth2	OVSBridge	None	42:63:73:65:11:a8	0	0	0	0
s2-eth1	s1-eth2	OVSBridge	None	02:d2:fa:68:30:a3	0	0	0	0
s2-eth2	s3-eth3	OVSBridge	None	f2:ad:c1:d1:e8:35	0	0	0	0
s3-eth1	h2-eth0	OVSBridge	None	36:94:3e:3d:4f:05	0	0	0	0
s3-eth2	h3-eth0	OVSBridge	None	46:92:fe:53:f0:df	0	0	0	0
s3-eth3	s2-eth2	OVSBridge	None	be:d8:e6:93:d5:2e	0	0	0	0
s3-eth4	s4-eth1	OVSBridge	None	8a:48:35:4c:a3:07	0	0	0	0
s4-eth1	s3-eth4	OVSBridge	None	7a:39:11:e9:15:d3	0	0	0	0
s4-eth2	s1-eth3	OVSBridge	None	ba:6e:1d:b6:c3:1b	0	0	0	0

In this, we can check the in port and out port. For example, H1-eth0 is connected to S1-eth1, thus port 0 of H1 is connected to port 1 of S1. Like that, S1-eth2 is connected to S2-eth1, thus port 2 of S1 is connected to port 1 of S2. Make sure you label all the ports before you go further since it will help you understand the topology and write flow entries.

Actions: This will give the action you want to do with the flow like either drop or forward. So, **<output:port_number>** will forward the traffic on that port. We can also set the multiple ports for output thus reducing the flow entries like we did in this example. The commands given below are the flow entries for **H1 to H2 using route H1-S1-S2-S3-H2 and H1 to H3 using route H1-S1-S2-S3-H3.**



```
mininet> sh ovs-ofctl add-flow s1 priority=500,in_port=1,actions=output:2
mininet> sh ovs-ofctl add-flow s1 priority=500,in_port=2,actions=output:1
mininet> sh ovs-ofctl add-flow s2 priority=500,in_port=2,actions=output:1
mininet> sh ovs-ofctl add-flow s2 priority=500,in_port=1,actions=output:2
```



```
mininet> sh ovs-ofctl add-flow s3 priority=500,in_port=3,actions=output:1,2
mininet> sh ovs-ofctl add-flow s3 priority=500,in_port=1,actions=output:3
mininet> sh ovs-ofctl add-flow s3 priority=500,in_port=2,actions=output:3
mininet> sh ovs-ofctl add-flow s4 priority=500,in_port=2,actions=drop
mininet> sh ovs-ofctl add-flow s4 priority=500,in_port=1,actions=drop
```

Similarly, using the above information and screen-shots, add flow from **H1 to H2 using route H1-S1-S4-S3-H2 and H1 to H3 using route H1-S1-S4-S3-H3.** Note that we can only add routes using one route otherwise if we add routes in both switch it create loop and generate broadcast storm. One more thing is to add reverse flow otherwise it will not ping since we need flow for the reply message.

Step 5: Once you have added the flow entries from H1 to H2 and H1 to H3, ping from H1 to H2 and from H1 to H3. (Use ping for 4 count using command <**h1 ping h2 -c 4**>) (Paste the screenshot of the CLI)

Note: When you ping the from H1 to H2 or H1 to H3, on the switch S3 you will see packet going to H3 if you ping to H2 and vice versa. This is due to arp request sent to the host for MAC learning. Since we are not using controller there is no MAC learning feature. You can check this on Wireshark.

Step 6: Open the Wireshark from using new terminal window. Start packet tracing on the S3-Eth2 and ping from H1 to H2 using 4 count command. Check then packets you receive on the port. You will only able to see Request packet and Arp request. (Paste screenshot of Wireshark window)

Step 7: Now delete all the flows using command (**sh ovs-ofctl del-flows <switch>)** Now, dump all the flows using command (**sh ovs-ofctl dump-flows <switch>**). Now paste **screenshot of all the flow table of all switch (**It should be empty**).** Now using all the information provided about the flow entries, create flow such that when you ping from **H1 to H2 and H1 to H3, the request message of ping should follow the path H1-S1-S2-S3-H2/H3 and the reply message should follow path H2/H3-S3-S4-S1-H1. (Paste the screenshot of all the flow entries using dump-flows command)**

Hint: To implement above flow, add forward path following towards S2 and instead of adding reverse path in same switch, add the reverse path in S4 switch.

This concludes your lab 3.
Thank you.