Introduction To Software-Defined Networking (Winter 2014/2015)

Institute of Computer Science
Georg-August-Universität Göttingen

Lecturer:	Dr.David Koll
Author:	Hari Raghavendar Rao Bandari
	11334055
	h.bandari@stud.uni-goettingen.de
Date:	April 30, 2015

Contents

List of Diagrams

1 Exercise:7 Mininet and SDN Controllers

The Python API and Topologies

```
1. def runExp():# defines a function to run a mininet experiment.
2.     topo = customTopo( n=2 ) # Creates a customTopo with two
nodes.
3.     net = Mininet ( topo=topo )# Invoke mininet and pass
customTopo we created above.
4.     net.start() # net is calling function start, which starts
mininet emulator.
5.     CLI( net ) # Before net.stop() will escape the interactive
CLI before scripts terminates
6.     net.stop # net is calling the function stop, which stops the
mininet network.
7
8. class customTopo( Topo ): # define a class to create custom
network topology.
9.     def __init__( self, n, **kwargs ): # defines the constructor
of customTopo class with arguments such as self, n, kwargs.
10.        Topo.__init__( self, **kwargs ): # calling
the constructor of parent class.
11.        h1, h2 = self.addHost( h1 ), self.addHost( h2 ) # Adding
two hosts to the topology.
12.        s1 = self.addSwitch( s1) # Adding a
switch to the topology.
13.        for _ in range( n ):
14.            self.addLink( s1, h1 ) # Adding two-way link between
the switch s1 and host h1.
15.            self.addLink( s1, h2 ) # Adding two-way link between
the switch s1 and host h2.
16.
17. if __name__ == __main__: # Main function
18.     setLogLevel( info ) # To set up Mininets default output
```

```
level of logs.Notify useful information.
19.     runExp() # Executing function to run the mininet experiment
defined at step 1.
```

The POX controller

1.1 What do you observe after connecting the controller?

After Pox controller is enabled, mininet will accepts it as controller for particular mininet set-up and it will show status is up and start listening to updates in the network at same time it will show status is connected and starts controlling the hosts. where each host can ping each other.

1.2 Open one xterm window for each host and run tcp-dump on hosts 1 and 2. Then, in the xterm of host 3, try to ping the IP address of h1 or h2. Also try to ping a device that is not reachable. Please explain what you observe

```
$ tcpdump _XX -n -i h1-eth0( h1 terminal)
$ tcpdump _XX -n -i h2-eth0( h2 terminal)
$ ping -c1 10.0.0.1 (h3 terminal)
```

After running above commands every packet will be broadcasting to each and every host. So our hub is working correctly as hub forwards traffic out of all of its ports.

```
$ ping -c1 10.0.0.5(h3 terminal ping request to h5)
```

If we ping to unknown host, Then packet will be transmitted but it won't receive at any host in the network and packets will be loss.

1.3 Open up the file pox/pox/misc/of_tutorial.py in the editor. In the Python code you will see that we are currently operating our controller with the help of the act_like_hub() method

1. Modify the controller to use act_like_switch() instead.

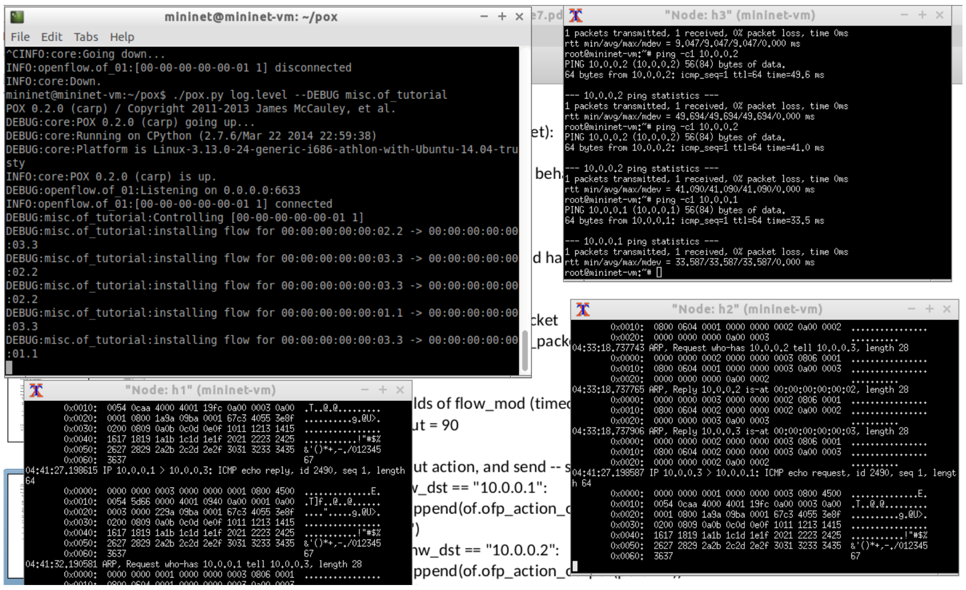
Solution :

```
 # self.act_like_hub(packet, packet_in)
   self.act_like_switch(packet, packet_in)
```

2. Implement act_like_switch() so that your controller actually acts like a switch.

Solution :

```
def act_like_switch (self, packet, packet_in):
    """
    Implement switch-like behavior.
    """
    # Here's some psuedocode to start you off implementing a learning
    # switch.  You'll need to rewrite it as real
    Python code.

    # Learn the port for the source MAC
    self.mac_to_port[str(packet.src)] = packet_in.in_port

    if str(packet.dst) in self.mac_to_port:
      # Send packet out the associated port

      # Once you have the above working, try pushing a flow entry
      # instead of resending the packet (comment
       out the above and
      # uncomment and complete the below.)
      port = self.mac_to_port[str(packet.dst)]

      # Maybe the log statement should have source/destination/port?
      log.debug("installing flow for %s.%i -> %s.%i" %
             (packet.src, packet_in.in_port, packet.dst, port))
      msg = of.ofp_flow_mod()

      ## Set fields to match received packet
      #msg.match = of.ofp_match.from_packet(packet)
      msg.match.dl_dst = packet.dst
      msg.match.dl_src = packet.src

      #< Set other fields of flow_mod (timeouts? buffer_id?) >
      msg.idle_timeout = 10
      msg.hard_timeout = 30
      #< Add an output action, and send -- similar to resend_packet() >
      msg.actions.append(of.ofp_action_output(port = port))
      msg.buffer_id = packet_in.buffer_id
      self.connection.send(msg)
    else:
      # Flood the packet out everything but the input port
      # This part looks familiar, right?
      self.resend_packet(packet_in, of.OFPP_ALL)
```



```
^CINFO:core:Going down...
INFO:openflow.of_01:[00-00-00-00-00-01 1] disconnected
INFO:core:Down.
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:38)
DEBUG:core:Platform is Linux-3.13.0-24-generic-i686-athlon-with-Ubuntu-14.04-tru
sty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 1]
DEBUG:misc.of_tutorial:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00
:03.3
DEBUG:misc.of_tutorial:installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00
:02.2
DEBUG:misc.of_tutorial:installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00
:02.2
DEBUG:misc.of_tutorial:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00
:03.3
DEBUG:misc.of_tutorial:installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00
:01.1
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 9.047/9.047/9.047/0.000 ms
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=49.6 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 49.694/49.694/49.694/0.000 ms
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=41.0 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 41.090/41.090/41.090/0.000 ms
root@mininet-vm:~# ping -c1 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=33.5 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 33.587/33.587/33.587/0.000 ms
root@mininet-vm:~# 
```

```
        0x0010:  0800 0604 0001 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0000 0a00 0003                 ..........
04:33:18.737743 ARP, Request who-has 10.0.0.2 tell 10.0.0.3, length 28
        0x0000:  0000 0000 0002 0000 0000 0003 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0003 0a00 0003  ................
        0x0020:  0000 0000 0000 0a00 0002                 ..........
04:33:18.737765 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
        0x0000:  0000 0000 0003 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0003 0a00 0003                 ..........
04:33:18.737906 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
        0x0000:  0000 0000 0002 0000 0000 0003 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0003 0a00 0003  ................
        0x0020:  0000 0000 0002 0a00 0002                 ..........
04:41:27.198587 IP 10.0.0.3 > 10.0.0.1: ICMP echo request, id 2490, seq 1, lengt
h 64
        0x0000:  0000 0000 0001 0000 0000 0003 0800 4500  ..............E.
        0x0010:  0054 0caa 4000 4001 19fc 0a00 0003 0a00  .T..@.@.........
        0x0020:  0001 0800 1a9a 09ba 0001 67c3 4055 3e8f  ..........g.@U>.
        0x0030:  0200 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
```

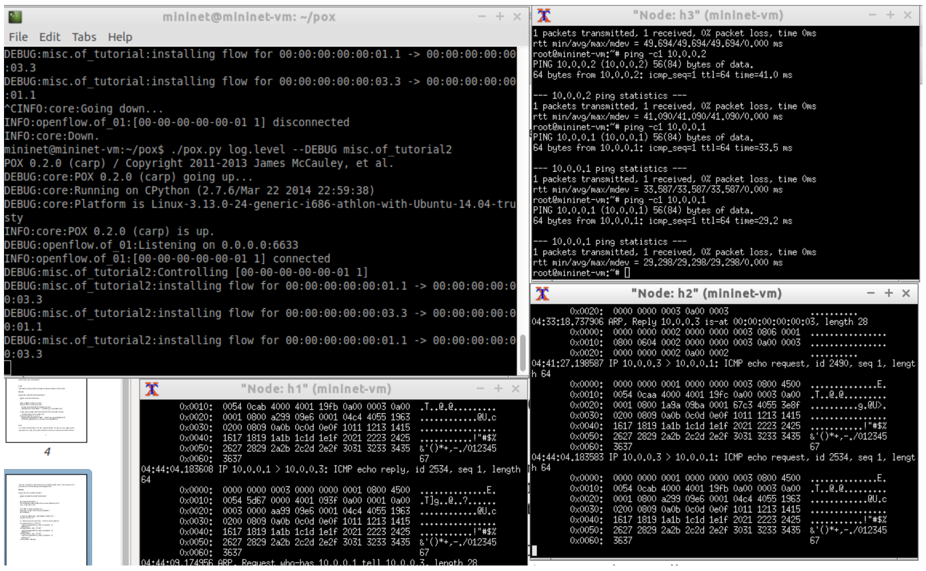
```
        0x0010:  0054 0caa 4000 4001 19fc 0a00 0003 0a00  .T..@.@.........
        0x0020:  0001 0800 1a9a 09ba 0001 67c3 4055 3e8f  ..........g.@U>.
        0x0030:  0200 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
04:41:27.198615 IP 10.0.0.1 > 10.0.0.3: ICMP echo reply, id 2490, seq 1, length
64
        0x0000:  0000 0000 0003 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 5d66 0000 4001 0940 0a00 0001 0a00  .T]f..@..@......
        0x0020:  0003 0000 229a 09ba 0001 67c3 4055 3e8f  ....".....g.@U>.
        0x0030:  0200 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
04:41:32.190581 ARP, Request who-has 10.0.0.1 tell 10.0.0.3, length 28
        0x0000:  0000 0000 0001 0000 0000 0003 0806 0001  ................
```

Figure 1: Screenshot of act-like-switch [17]

3. It is somewhat inefficient to let the controller decide the fate of every single packet. Implement an act_like_flow_switch() method in which your controller instead installs flow-rules into the switch for all packets that are initially flow-table misses. This will improve the performance of the forwarding on subsequent packets.

```
def act_like_flow_switch (self, packet, packet_in):
    """
    Implement switch-like behavior.
    """

    # Here's some psuedocode to start you off implementing a learning
    # switch.  You'll need to rewrite it as real
    Python code.

    # Learn the port for the source MAC
    self.mac_to_port[str(packet.src)] = packet_in.in_port
```

```
if str(packet.dst) in self.mac_to_port:
  # Send packet out the associated port

  # Once you have the above working, try pushing a flow entry
  # instead of resending the packet (comment
   out the above and
  # uncomment and complete the below.)

  port = self.mac_to_port[str(packet.dst)]

  # Maybe the log statement should have source/destination/port?
  log.debug("installing flow for %s.%i -> %s.%i" %
         (packet.src, packet_in.in_port, packet.dst, port))
  msg = of.ofp_flow_mod()

  ## Set fields to match received packet
  #msg.match = of.ofp_match.from_packet(packet)
  msg.match.dl_dst = packet.dst
  msg.match.dl_src = packet.src

  #< Set other fields of flow_mod (timeouts? buffer_id?) >
  msg.idle_timeout = 10
  msg.hard_timeout = 30
  #< Add an output action, and send -- similar to resend_packet() >
  msg.actions.append(of.ofp_action_output(port = port))
  msg.buffer_id = packet_in.buffer_id
  self.connection.send(msg)
else:
  # Flood the packet out everything but the input port
  # This part looks familiar, right?
  self.resend_packet(packet_in, of.OFPP_ALL)
```



```
DEBUG:misc.of_tutorial:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00
:03.3
DEBUG:misc.of_tutorial:installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00
:01.1
^CINFO:core:Going down...
INFO:openflow.of_01:[00-00-00-00-00-01 1] disconnected
INFO:core:Down.
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.of_tutorial2
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:38)
DEBUG:core:Platform is Linux-3.13.0-24-generic-i686-athlon-with-Ubuntu-14.04-tru
sty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:misc.of_tutorial2:Controlling [00-00-00-00-00-01 1]
DEBUG:misc.of_tutorial2:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:0
0:03.3
DEBUG:misc.of_tutorial2:installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:0
0:01.1
DEBUG:misc.of_tutorial2:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:0
0:03.3
```

```
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 49.694/49.694/49.694/0.000 ms
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=41.0 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 41.090/41.090/41.090/0.000 ms
root@mininet-vm:~# ping -c1 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=33.5 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 33.587/33.587/33.587/0.000 ms
root@mininet-vm:~# ping -c1 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=29.2 ms

--- 10.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 29.298/29.298/29.298/0.000 ms
root@mininet-vm:~# 
```

```
        0x0020:  0000 0000 0003 0a00 0003                 ..........
04:33:18.737906 ARP, Reply 10.0.0.3 is-at 00:00:00:00:00:03, length 28
        0x0000:  0000 0000 0002 0000 0000 0003 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0003 0a00 0003  ................
        0x0020:  0000 0000 0002 0a00 0002                 ..........
04:41:27.198587 IP 10.0.0.3 > 10.0.0.1: ICMP echo request, id 2490, seq 1, lengt
h 64
        0x0000:  0000 0000 0001 0000 0000 0003 0800 4500  ..............E.
        0x0010:  0054 0caa 4000 4001 19fc 0a00 0003 0a00  .T..@.@.........
        0x0020:  0001 0800 1a9a 09ba 0001 67c3 4055 3e8f  ..........g.@U>.
        0x0030:  0200 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
04:44:04.183583 IP 10.0.0.3 > 10.0.0.1: ICMP echo request, id 2534, seq 1, lengt
h 64
        0x0000:  0000 0000 0001 0000 0000 0003 0800 4500  ..............E.
        0x0010:  0054 0cab 4000 4001 19fb 0a00 0003 0a00  .T..@.@.........
        0x0020:  0001 0800 a299 09e6 0001 04c4 4055 1963  ............@U.c
        0x0030:  0200 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
```

```
        0x0010:  0054 0cab 4000 4001 19fb 0a00 0003 0a00  .T..@.@.........
        0x0020:  0001 0800 a299 09e6 0001 04c4 4055 1963  ............@U.c
        0x0030:  0200 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
04:44:04.183608 IP 10.0.0.1 > 10.0.0.3: ICMP echo reply, id 2534, seq 1, length
64
        0x0000:  0000 0000 0003 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 5d67 0000 4001 093f 0a00 0001 0a00  .T]g..@..?......
        0x0020:  0003 0000 aa99 09e6 0001 04c4 4055 1963  ............@U.c
        0x0030:  0200 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
04:44:09.174956 ARP, Request who-has 10.0.0.1 tell 10.0.0.3, length 28
```

Figure 2: Screenshot of act-like-flow-switch[18]

2 Exercise 8 Mininet and FlowVisor

2.1 Create your own FlowVisor topology

Using the Mininet Python API, create the FlowVisor WAN topology (which you may know from earlier exercises) in a file mini-fw-topo.py:

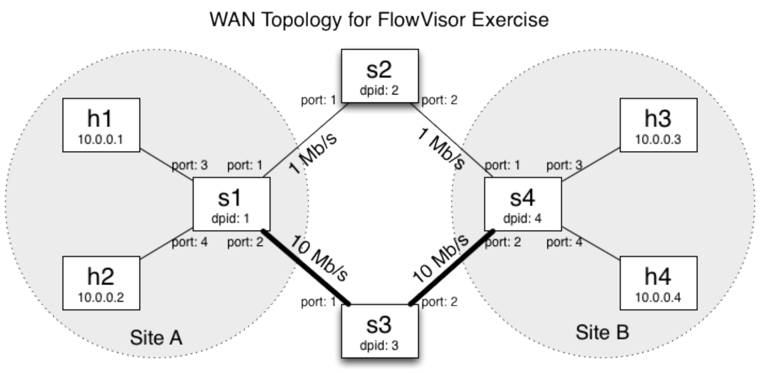


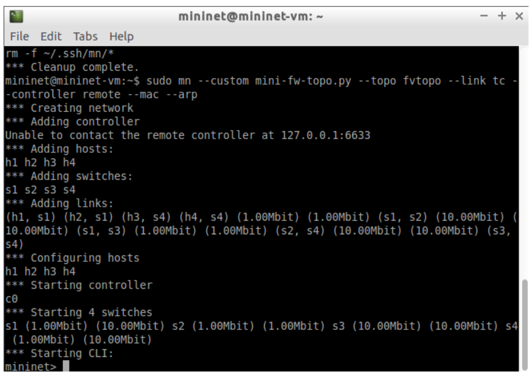
Figure 3: WAN Topology for FlowVisor[19]

To Run Diamond Topology

$ sudo mn –custom mini-fw-topo.py –topo fvtopo –link tc –controller remote –mac –arp

Above will create a network in mininet with the WAN topology. Along with it we have set static ARP entries for the Mininet hosts.

output:



```
rm -f ~/.ssh/mn/*
*** Cleanup complete.
mininet@mininet-vm:~$ sudo mn --custom mini-fw-topo.py --topo fvtopo --link tc -
-controller remote --mac --arp
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s1) (h3, s4) (h4, s4) (1.00Mbit) (1.00Mbit) (s1, s2) (10.00Mbit) (
10.00Mbit) (s1, s3) (1.00Mbit) (1.00Mbit) (s2, s4) (10.00Mbit) (10.00Mbit) (s3,
s4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 (1.00Mbit) (10.00Mbit) s2 (1.00Mbit) (1.00Mbit) s3 (10.00Mbit) (10.00Mbit) s4
 (1.00Mbit) (10.00Mbit)
*** Starting CLI:
mininet>
```

Figure 4: A network in mininet with WAN Topology [20]

2.2 Slice the Network

Now, slice your network so that it supports the following slices: In short, this slice

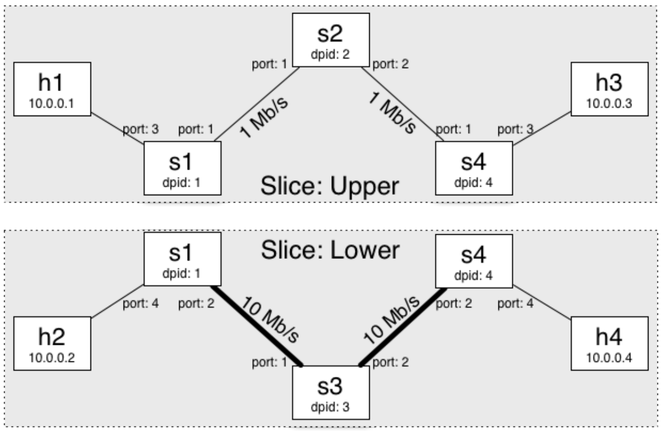


Figure 5: Simple Topology-based Slicing [21]

arrangement allows traffic to be sent from h1 to h3 and h2 to h4 (and viceversa) only, even though the topology itself (i.e., without slicing) would allow sending traffic between arbitrary pairs of hosts. For slicing a network with FlowVisor in general, you need to take the following steps. First, make sure you set up the flowvisor package correctly. To configure FlowVisor, use:

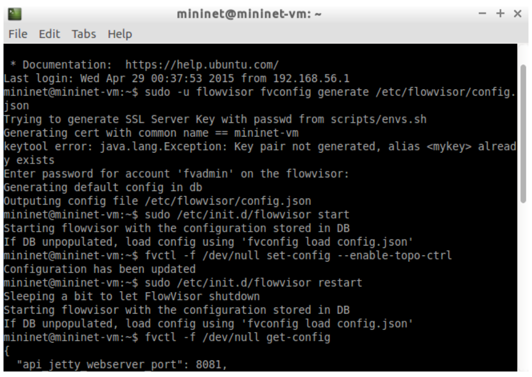
$ sudo -u flowvisor fvconfig generate /etc/flowvisor/config.json

Then, start flowvisor in a new terminal:
$ sudo /etc/init.d/flowvisor start

We have to enable topology control for flowvisor as well: $ fvctl -f /dev/null set-config –enable-topo-ctrl

Similar to ovs-ofctl, fvctl is the control channel that we will use for flowvisor. The option f refers to the flowvisor password file. Since we have set the password to be empty, we can hand it /dev/null. This part will be present in all the following fvctl calls.

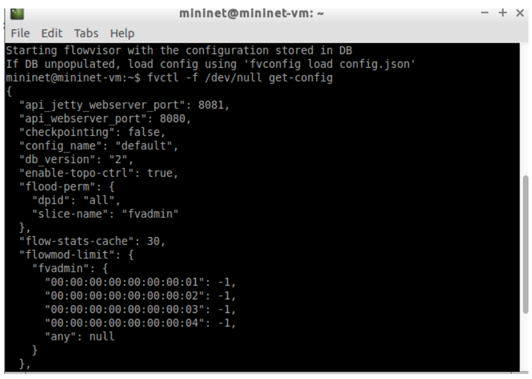
Restart flowvisor:
$ sudo /etc/init.d/flowvisor start



```
 * Documentation:  https://help.ubuntu.com/
Last login: Wed Apr 29 00:37:53 2015 from 192.168.56.1
mininet@mininet-vm:~$ sudo -u flowvisor fvconfig generate /etc/flowvisor/config.
json
Trying to generate SSL Server Key with passwd from scripts/envs.sh
Generating cert with common name == mininet-vm
keytool error: java.lang.Exception: Key pair not generated, alias <mykey> alread
y exists
Enter password for account 'fvadmin' on the flowvisor:
Generating default config in db
Outputing config file /etc/flowvisor/config.json
mininet@mininet-vm:~$ sudo /etc/init.d/flowvisor start
Starting flowvisor with the configuration stored in DB
If DB unpopulated, load config using 'fvconfig load config.json'
mininet@mininet-vm:~$ fvctl -f /dev/null set-config --enable-topo-ctrl
Configuration has been updated
mininet@mininet-vm:~$ sudo /etc/init.d/flowvisor restart
Sleeping a bit to let FlowVisor shutdown
Starting flowvisor with the configuration stored in DB
If DB unpopulated, load config using 'fvconfig load config.json'
mininet@mininet-vm:~$ fvctl -f /dev/null get-config
{
  "api_jetty_webserver_port": 8081,
```

Figure 6: Flowvisor Configuration, Start and Restart [22]

Now, have a look at the FlowVisor configuration:
$ fvctl -f /dev/null get-config



```
Starting flowvisor with the configuration stored in DB
If DB unpopulated, load config using 'fvconfig load config.json'
mininet@mininet-vm:~$ fvctl -f /dev/null get-config
{
  "api_jetty_webserver_port": 8081,
  "api_webserver_port": 8080,
  "checkpointing": false,
  "config_name": "default",
  "db_version": "2",
  "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
    }
  },
```

Figure 7: Flowvisor Configuration in JSON format [23]

This also has the purpose of making sure that flowvisor is actually running and that all the switches have indeed a connection to flowvisor. The configuration should show this.

a. Which part of the configuration file tells you that all four switches have connected to flowvisor?

Solution:

```
"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
    }
  }
```

In the lecture, you also got a brief overview over the major flowvisor commands. Now, make use of these commands to
b. List the currently existing slices.

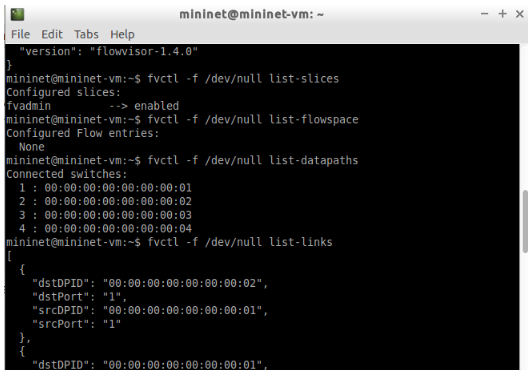
```
$ fvctl -f /dev/null list-slices
        Configured slices:\newline
             fvadmin  --> enabled
```

c. List the currently existing flowspaces.

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

d. List the currently connected switches.

```
$ 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
```

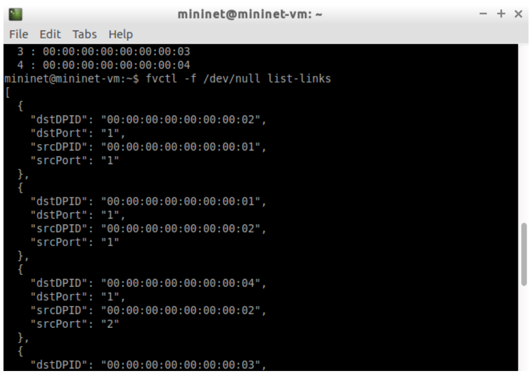


```
  "version": "flowvisor-1.4.0"
}
mininet@mininet-vm:~$ fvctl -f /dev/null list-slices
Configured slices:
fvadmin         --> enabled
mininet@mininet-vm:~$ fvctl -f /dev/null list-flowspace
Configured Flow entries:
  None
mininet@mininet-vm:~$ fvctl -f /dev/null list-datapaths
Connected switches:
  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
mininet@mininet-vm:~$ fvctl -f /dev/null list-links
[
  {
    "dstDPID": "00:00:00:00:00:00:00:02",
    "dstPort": "1",
    "srcDPID": "00:00:00:00:00:00:00:01",
    "srcPort": "1"
  },
  {
    "dstDPID": "00:00:00:00:00:00:00:01",
```

Figure 8: Screenshot of List of slices/flowspace/datapaths [24]

e. List the currently existing links.

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



```
  3 : 00:00:00:00:00:00:00:03
  4 : 00:00:00:00:00:00:00:04
mininet@mininet-vm:~$ fvctl -f /dev/null list-links
[
  {
    "dstDPID": "00:00:00:00:00:00:00:02",
    "dstPort": "1",
    "srcDPID": "00:00:00:00:00:00:00:01",
    "srcPort": "1"
  },
  {
    "dstDPID": "00:00:00:00:00:00:00:01",
    "dstPort": "1",
    "srcDPID": "00:00:00:00:00:00:00:02",
    "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",
```

Figure 9: Screenshot of List of links [25]

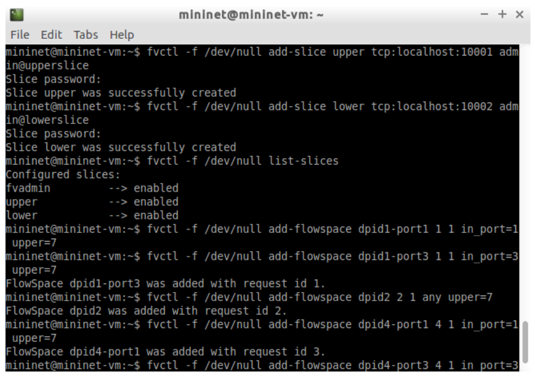
Afterwards, proceed with slicing your topology: f. Create the appropriate slices.

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
```

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



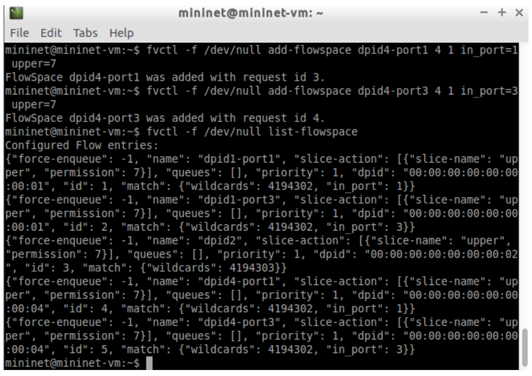
```
mininet@mininet-vm:~$ fvctl -f /dev/null add-slice upper tcp:localhost:10001 adm
in@upperslice
Slice password:
Slice upper was successfully created
mininet@mininet-vm:~$ fvctl -f /dev/null add-slice lower tcp:localhost:10002 adm
in@lowerslice
Slice password:
Slice lower was successfully created
mininet@mininet-vm:~$ fvctl -f /dev/null list-slices
Configured slices:
fvadmin         --> enabled
upper           --> enabled
lower           --> enabled
mininet@mininet-vm:~$ fvctl -f /dev/null add-flowspace dpid1-port1 1 1 in_port=1
 upper=7
mininet@mininet-vm:~$ fvctl -f /dev/null add-flowspace dpid1-port3 1 1 in_port=3
 upper=7
FlowSpace dpid1-port3 was added with request id 1.
mininet@mininet-vm:~$ fvctl -f /dev/null add-flowspace dpid2 2 1 any upper=7
FlowSpace dpid2 was added with request id 2.
mininet@mininet-vm:~$ fvctl -f /dev/null add-flowspace dpid4-port1 4 1 in_port=1
 upper=7
FlowSpace dpid4-port1 was added with request id 3.
mininet@mininet-vm:~$ fvctl -f /dev/null add-flowspace dpid4-port3 4 1 in_port=3
```

Figure 10: Screenshot After Slicing–List of slices [26]

g. Create the appropriate flowspaces. solution: Creating flowspaces for upper Slice:

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

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

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

$ 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
```

```
Creating flowspaces for 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
```



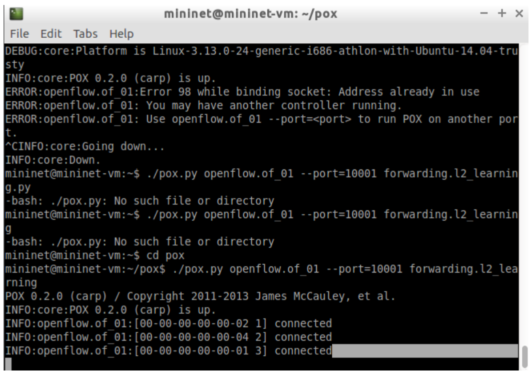
```
mininet@mininet-vm:~$ fvctl -f /dev/null add-flowspace dpid4-port1 4 1 in_port=1
 upper=7
FlowSpace dpid4-port1 was added with request id 3.
mininet@mininet-vm:~$ fvctl -f /dev/null add-flowspace dpid4-port3 4 1 in_port=3
 upper=7
FlowSpace dpid4-port3 was added with request id 4.
mininet@mininet-vm:~$ fvctl -f /dev/null list-flowspace
Configured Flow entries:
{"force-enqueue": -1, "name": "dpid1-port1", "slice-action": [{"slice-name": "up
per", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00
:00:01", "id": 1, "match": {"wildcards": 4194302, "in_port": 1}}
{"force-enqueue": -1, "name": "dpid1-port3", "slice-action": [{"slice-name": "up
per", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00
:00:01", "id": 2, "match": {"wildcards": 4194302, "in_port": 3}}
{"force-enqueue": -1, "name": "dpid2", "slice-action": [{"slice-name": "upper",
"permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00:00:02
", "id": 3, "match": {"wildcards": 4194303}}
{"force-enqueue": -1, "name": "dpid4-port1", "slice-action": [{"slice-name": "up
per", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00
:00:04", "id": 4, "match": {"wildcards": 4194302, "in_port": 1}}
{"force-enqueue": -1, "name": "dpid4-port3", "slice-action": [{"slice-name": "up
per", "permission": 7}], "queues": [], "priority": 1, "dpid": "00:00:00:00:00:00
:00:04", "id": 5, "match": {"wildcards": 4194302, "in_port": 3}}
mininet@mininet-vm:~$
```

Figure 11: Screenshot After Slicing–flowspaces [27]

h. (10P) Connect an instance of the POX controller to each of your slices. solution:

```
$ ./pox.py openflow.of_01 --port=10001 forwarding.l2_learning
```

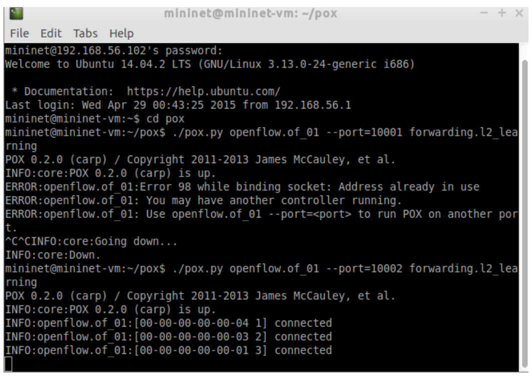
It is connected to S1, S2, S4



```
DEBUG:core:Platform is Linux-3.13.0-24-generic-i686-athlon-with-Ubuntu-14.04-tru
sty
INFO:core:POX 0.2.0 (carp) is up.
ERROR:openflow.of_01:Error 98 while binding socket: Address already in use
ERROR:openflow.of_01: You may have another controller running.
ERROR:openflow.of_01: Use openflow.of_01 --port=<port> to run POX on another por
t.
^CINFO:core:Going down...
INFO:core:Down.
mininet@mininet-vm:~$ ./pox.py openflow.of_01 --port=10001 forwarding.l2_learnin
g.py
-bash: ./pox.py: No such file or directory
mininet@mininet-vm:~$ ./pox.py openflow.of_01 --port=10001 forwarding.l2_learnin
g
-bash: ./pox.py: No such file or directory
mininet@mininet-vm:~$ cd pox
mininet@mininet-vm:~/pox$ ./pox.py openflow.of_01 --port=10001 forwarding.l2_lea
rning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-02 1] connected
INFO:openflow.of_01:[00-00-00-00-00-04 2] connected
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
```

Figure 12: Screenshot of POX Controller [28]

It is connected to S1, S3, S4

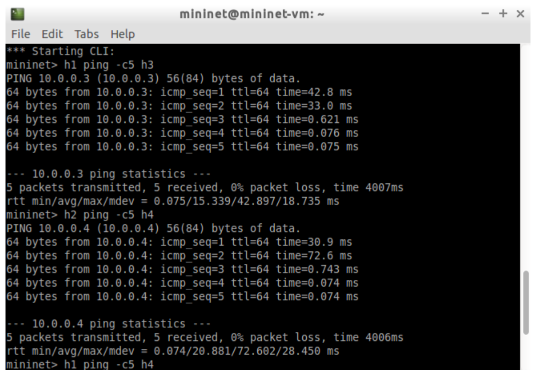


```
mininet@192.168.56.102's password:
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-24-generic i686)

 * Documentation:  https://help.ubuntu.com/
Last login: Wed Apr 29 00:43:25 2015 from 192.168.56.1
mininet@mininet-vm:~$ cd pox
mininet@mininet-vm:~/pox$ ./pox.py openflow.of_01 --port=10001 forwarding.l2_lea
rning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
ERROR:openflow.of_01:Error 98 while binding socket: Address already in use
ERROR:openflow.of_01: You may have another controller running.
ERROR:openflow.of_01: Use openflow.of_01 --port=<port> to run POX on another por
t.
^C^CINFO:core:Going down...
INFO:core:Down.
mininet@mininet-vm:~/pox$ ./pox.py openflow.of_01 --port=10002 forwarding.l2_lea
rning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-04 1] connected
INFO:openflow.of_01:[00-00-00-00-00-03 2] connected
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
```

Figure 13: Screenshot of POX Controller [29]

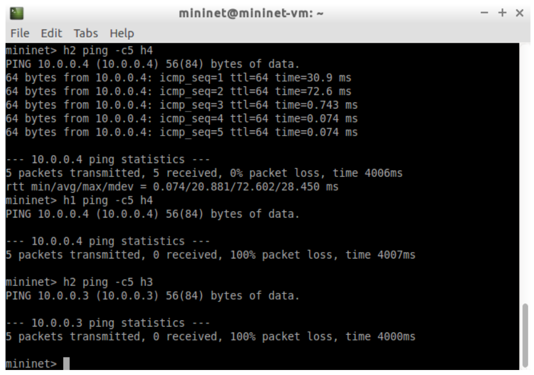
i. (10P) In Mininet, verify that your slicing works properly, i.e., h1 can reach h3 but not h2 and h4, and h2 can reach h4, but not h1 and h3.

Solution:



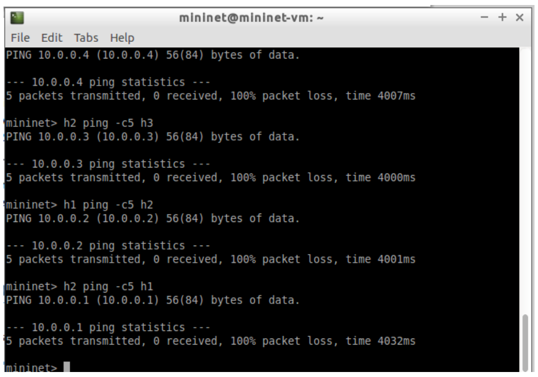
```
*** Starting CLI:
mininet> h1 ping -c5 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=42.8 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=33.0 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.621 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.076 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.075 ms

--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4007ms
rtt min/avg/max/mdev = 0.075/15.339/42.897/18.735 ms
mininet> h2 ping -c5 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=30.9 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=72.6 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.743 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.074 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.074 ms

--- 10.0.0.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.074/20.881/72.602/28.450 ms
mininet> h1 ping -c5 h4
```

Figure 14: Screenshot of Mininet slicinig work properly [30]



```
mininet> h2 ping -c5 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=30.9 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=72.6 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.743 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.074 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.074 ms

--- 10.0.0.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.074/20.881/72.602/28.450 ms
mininet> h1 ping -c5 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.

--- 10.0.0.4 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4007ms

mininet> h2 ping -c5 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

--- 10.0.0.3 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4000ms

mininet> 
```

Figure 15: Screenshot of Mininet slicinig work properly[31]



```
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.

--- 10.0.0.4 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4007ms

mininet> h2 ping -c5 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

--- 10.0.0.3 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4000ms

mininet> h1 ping -c5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4001ms

mininet> h2 ping -c5 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.

--- 10.0.0.1 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4032ms

mininet> 
```

Figure 16: Screenshot of Mininet slicinig work properly [32]

3 Paper Review

3.1 Paper Information

Title: OpenVirteX: Make Your Virtual SDNs Programmable
Type of Paper: Short Paper

3.2 Summary of the paper

OpenVirteX is a network virtualization platform which enables tenants to create and manage their own virtual network software defined networks with customised topology and have control on vSDNs, it allows tenants to utilize complete physical infrastructure resources but in the form of virtual. Therefore OpenVirteX provides full virtualization to tenants. Tenants will be interacting with the virtual network which resembles as real physical network. Its architecture enables the introduction of features and enhancements such as link resilience to tenant networks.In other words, OpenVirteX creates multiple virtual software defined networks out of one. It gives more flexibility and efficiency to switch the traffic in fraction of seconds whenever network is down.

Therefore, Due to its dynamic nature it provides tenants to have programmable Virtual Networks. It delivers Infrastructure as service(IaaS) in the context of Software defined networks. Therefore, It sits in between virtual network controller and physical hardware it provides to create isolated virtual network with a customized topology so tenant can view his own physical network in any topology to decrease the complexity of the network, Such virtual networks could be instantiated along with the compute resources to deliver true infrastructure on-demand.Here tenants can have own network operating system on demand.

OpenVirteX as a network virtualization platform provide address virtualization to keep tenant traffic separate because every tenant will have fully virtualized network featuring a tenant specified topology and a full header space. Tenants can change their own network at runtime , As each virtual element is simply a pointer onto some real network element such as we can disable, enable, modify and/or reorganize virtual network elements at runtime. It automatically recovers from physical failures. According topology virtualization they have limitations by design, the only limitation enforced by OVX is that a single physical switch cannot be partitioned into multiple virtual switches plus Address Virtualization Evidently, OVX cannot support the scenario where all tenants want to use the entire IP or MAC space since OVX reserve some bits in those headers to encode the tenant identifier.

3.3 Contributions of the Paper

They discussed issues about Flowvisor cannot provide fully virtualization and also its doesn't provide isolation if it is effected slice and cannot provide completely separate and independent address space. It does not allow a slice to have an virtual network topology. OpenVirtex provides above features to tenant and it is very important for tenants to have fully virtualization network and dynamically they can change their virtual topology i.e. Topology virtualization and other hand full address virtualization which provide isolation and security. It gives tenants to enable each virtual network to have its own NOS.The degree of novelty,creativity is they provide custom topology to view physical network and impact is simple own vSDN architecture for each tenant. This paper didnt explained technically in depth but explained in theoretically.

3.4 Strengths of the Paper

1.Innovate Idea tenants can configure their own NOS without effecting their physical infrastructure. Customized Topology and clone of physical network 2. paper presentation is clearly explained what they have implemented. 3. The main reason is evaluation performance is compared among all virtulization controller where OVX takes "Virtual network provisioning time in seconds". OVX works on all different type of networks

3.5 Weaknesses of the Paper

Limitation : 1. Tenants cannot have entire MAC address or IP Address. 2. OVX limitation is i.e. a single physical switch cannot be partitioned into multiple virtual switches. 3. This implementation doesn't scale or provide any significant performance.

3.6 Open Issues and Future Work

1. Snapshotting and migration of vSDNs. 2. Evolving beyond OF1.0 3. vSDN-based Quality of Service above three are very important if the network is virtualized.

3.7 Your Questions to the Authors

1. what kind of rules for the reference about the IP address changing form Physical IP to OVXIP and viceversa? 2. what if tenants wants to reconfigure of virtual network.

4 Paper Review

4.1 Paper Information

Title: CoVisor: A Compositional Hypervisor for Software-Defined Networks
Type of Paper: Full Paper

4.2 Summary of the paper

Covisor is a novel kind of network hypervisor that enables a single network, which has many controllers written in different programming languages and operating on different platforms. Therefore, Implementation is depends on two-phased compilation process. 1. Covisor will Assemble all these controller policies together delivers single policy to physical network, which required only a single administrator to manage controller applications in combination or separately to desire traffic. 2. Covisor also provides custom virtual topologies, this will allow administrators to define access controls that regulate the packets a given controller may modify, monitor,Or reroute, protection against misbehaving controllers can be restrict by limiting functionality of the virtual switches exposed to other controller for example firewall controllers should not be allowed to modify packets and MAC learner should not able to inspect IP or TCP headers.

The technical contribution of the composition is about efficient algorithms for composing controller policies, for compiling virtual networks into concrete OpenFlow rules, and for efficiently processing controller rule updates. The abstract topology of covisor deals with one physical switch to many logical switches. They developed a new algorithm for data plane policies. They are 1. parallel: It allow multiple controllers to act independently on the same packets at the same time, 2. sequentially: It allow one controller to process certain traffic before another,and by 3. overriding: It allow one controller to choose to act or to defer control to another controller. Covisor is independent of the specific languages, libraries, or controller platforms used to construct client applications and to match administrator specified composition policies with the openflow messages it requires to implement a new incremental algorithms for processing rule updates which again depends on action and parallel operators and sequential operators and override operation. Therefore, administrator can configure the policies based on these operator when assembling the controller applications get the priority based on these actions only.similarly for incremental policy compilation.

Therefore, covisor solution was compared with the strawman solution is to assign priorities to rules in the composed implementation from bottom to top and so every time when a new is updated it has to recompile whole policy then only priority will change it disadvantage it takes more compilation time.

In conclusion functionalities of the covisor uses a novel algorithm to incrementally compose applications with rule priorities to calculate priorities for new rules using convenient Algebra . Therefore , It removes the need to recompile from scratch for every rule update and it translates the composed policy into rules for physical topology. (one physical switch mapped to multiple virtual switches). After compiling the policy, covisor sends the necessary rule to update to switches this will allow to reduce the compilation time.

4.3 Contributions of the Paper

The major issues addressed in this paper is how to assign priority when the new rule is updated whether it has to recompile whole policy to assign new priority and as it assembles all controller policies gives one policy to the physical network and allow each and every controller has its own logical topology. Therefore, challenge was to optimise the compilation process and flow table compilation efficiency. It very important to decrease the compilation time and they solved above problems by introducing as their novelty/creation Incremental compilation where it provides Incremental priority assignment and advanced indexing for fast rule lookup. The impact of this implementation reduces compilation time and this paper explained in depth how rule priorities will be allotted to the controllers for compilation.

4.4 Strengths of the Paper

1.This paper explained theoretically in depth with implementation results. 2.Its clear enough the performance of covisor is faster than naive implementation . 3. They introduced Compositional mode operations.

4.5 Weaknesses of the Paper

It's not clear that the rule update level is the best place to implement the composition abstraction in the long term.

4.6 Open Issues and Future Work

1.supporting OpenFlow 1.3 protocols, compiling a policy in OpenFlow 1.0 into flow tables that support OpenFlow 1.3, and supporting arbitrary nesting of topology virtualization and policy composition in a single instance. The implementation required extensive use of priority to implement composition, so we'll need to manually change priorities of existing application or provide a mapping.

4.7 Your Questions to the Authors

How the isolation and security will handle when assemble multiple controller policies?

References

[1] *Ali Al-Shabibi, M.De Leenheer, M. Gerola,A.Koshibe,G.Parulkar, E. Salvadori, and B.Snow , OpenVirteX: Make Your Virtual SDNs Programmable, ACM SIGCOMM HotSDN workshop,August 2014.*

[2] *Xin Jin ,Jennifer Gossels,Jennifer Rexford,David Walker Princeton University CoVisor: A Compositional Hypervisor for Software-Defined Networks, USENIX NSDI May 4-6 2015.*