Selected Topics on Software Defined Networking

EECE.7290

Instructor: *Prof. Yan Luo*

Lab 1: Experimenting OpenFlow with Mininet

Hand in Date: 21/02/17
Due Date: 21/02/17

By,
- Naga Ganesh Kurapati
ID:01592079

Table of contents:

a. Purpose

The main purpose this lab is to experiment Open-flow with mini-net software to understand the working of l2 learning switch. We need to make use of POX controller to design l2 learning switch in Python programming language. Open-flow packets are captured using Wireshark.

b. Lab Procedure

Act Like a Hub:

Before we start to program POX controller to act like a learning switch, we need to run it like a hub to troubleshoot the setup. By default, of_tutorial.py in *misc* folder of *pox* library is configured to flood the output ports of the switch for every packet it received. You can start the POX controller from the pox library by using the command ./pox.py log.level - -DEBUG misc.of_tutorial. It shown in the fig. below

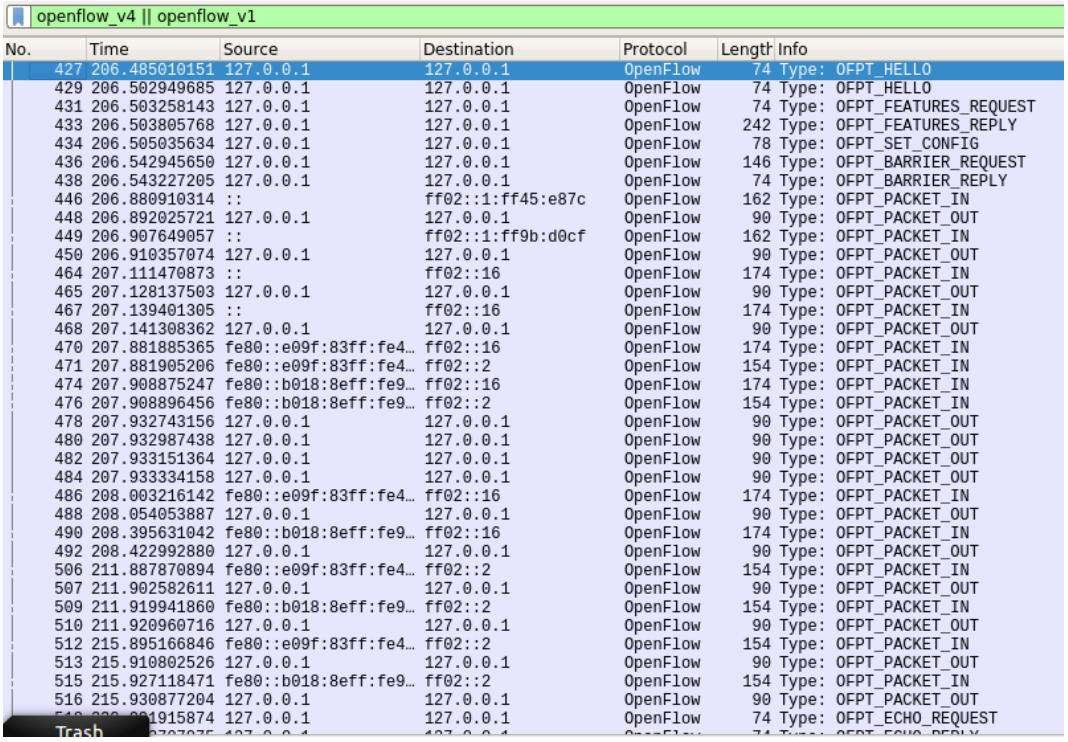
```
ganeshkurapati@ganeshkurapati-VirtualBox:~/pox$ ./pox.py log.level --DEBUG misc.
of_tutorial
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.12/Nov 19 2016 06:48:10)
DEBUG:core:Platform is Linux-4.4.0-62-generic-x86_64-with-Ubuntu-16.04-xenial
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 2]
```

Default topology from the mini-net can started using the command below. It contains two hosts, one switch and it is direct to connect to POX controller listening on port 6633 in the localhost. You can try to ping h2 from h1 using the command h1 ping h2.

```
ganeshkurapati@ganeshkurapati-VirtualBox:~$ sudo mn --controller=remote,ip=127.0.0.1,port=6633
[sudo] password for ganeshkurapati:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> h1 ping -c 1 h2
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=34.5 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 34.574/34.574/34.574/0.000 ms
mininet> h1 ping -c 1 h2
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=23.7 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 23.797/23.797/23.797/0.000 ms
mininet> h1 ping -c 1 h2
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=24.6 ms
```

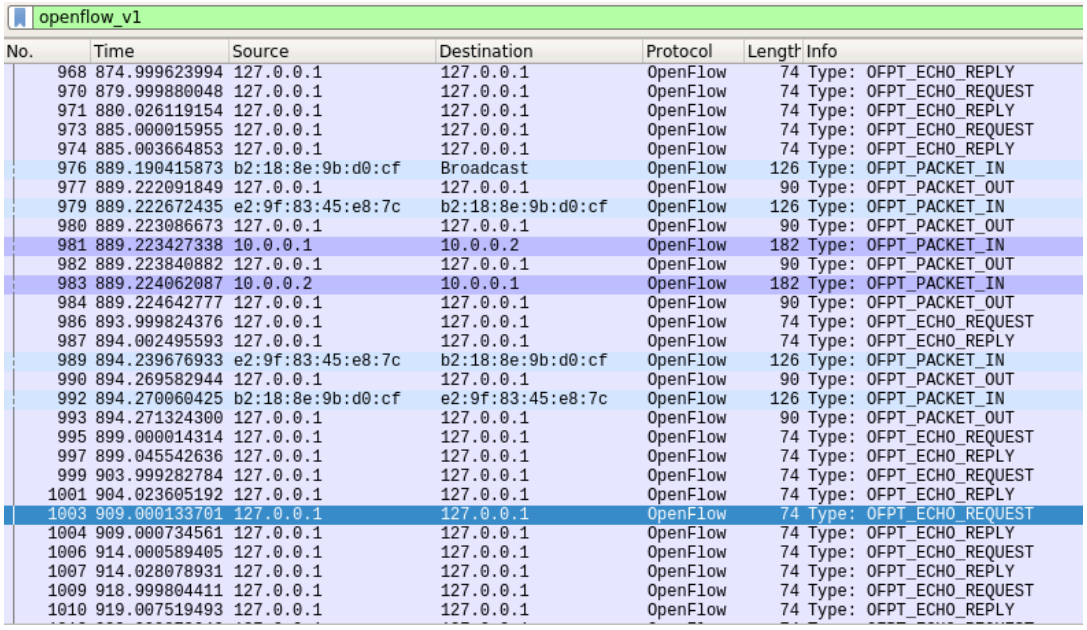


openflow_v4 || openflow_v1

No.	Time	Source	Destination	Protocol	Length	Info
427	206.485010151	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
429	206.502949685	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO
431	206.503258143	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
433	206.503805768	127.0.0.1	127.0.0.1	OpenFlow	242	Type: OFPT_FEATURES_REPLY
434	206.505035634	127.0.0.1	127.0.0.1	OpenFlow	78	Type: OFPT_SET_CONFIG
436	206.542945650	127.0.0.1	127.0.0.1	OpenFlow	146	Type: OFPT_BARRIER_REQUEST
438	206.543227205	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REPLY
446	206.880910314	::	ff02::1:ff45:e87c	OpenFlow	162	Type: OFPT_PACKET_IN
448	206.892025721	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
449	206.907649057	::	ff02::1:ff9b:d0cf	OpenFlow	162	Type: OFPT_PACKET_IN
450	206.910357074	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
464	207.111470873	::	ff02::16	OpenFlow	174	Type: OFPT_PACKET_IN
465	207.128137503	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
467	207.139401305	::	ff02::16	OpenFlow	174	Type: OFPT_PACKET_IN
468	207.141308362	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
470	207.881885365	fe80::e09f:83ff:fe4…	ff02::16	OpenFlow	174	Type: OFPT_PACKET_IN
471	207.881905206	fe80::e09f:83ff:fe4…	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
474	207.908875247	fe80::b018:8eff:fe9…	ff02::16	OpenFlow	174	Type: OFPT_PACKET_IN
476	207.908896456	fe80::b018:8eff:fe9…	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
478	207.932743156	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
480	207.932987438	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
482	207.933151364	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
484	207.933334158	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
486	208.003216142	fe80::e09f:83ff:fe4…	ff02::16	OpenFlow	174	Type: OFPT_PACKET_IN
488	208.054053887	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
490	208.395631042	fe80::b018:8eff:fe9…	ff02::16	OpenFlow	174	Type: OFPT_PACKET_IN
492	208.422992880	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
506	211.887870894	fe80::e09f:83ff:fe4…	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
507	211.902582611	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
509	211.919941860	fe80::b018:8eff:fe9…	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
510	211.920960716	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
512	215.895166846	fe80::e09f:83ff:fe4…	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
513	215.910802526	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
515	215.927118471	fe80::b018:8eff:fe9…	ff02::2	OpenFlow	154	Type: OFPT_PACKET_IN
516	215.930877204	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
	.001915874	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST

Trash

If above fig. you can see packet capture using Wireshark during the start of the topology. Its contains different Open-flow messages between the switch and the controller. OFPT_HELLO is used for version negotiation. Controller then sends OFPT_FEATURES_REQUEST, then switch responses with OFPT_FEATURES_REPLY with its capabilities.



openflow_v1

No.	Time	Source	Destination	Protocol	Length	Info
968	874.999623994	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
970	879.999880048	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
971	880.026119154	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
973	885.000015955	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
974	885.003664853	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
976	889.190415873	b2:18:8e:9b:d0:cf	Broadcast	OpenFlow	126	Type: OFPT_PACKET_IN
977	889.222091849	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
979	889.222672435	e2:9f:83:45:e8:7c	b2:18:8e:9b:d0:cf	OpenFlow	126	Type: OFPT_PACKET_IN
980	889.223086673	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
981	889.223427338	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
982	889.223840882	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
983	889.224062087	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
984	889.224642777	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
986	893.999824376	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
987	894.002495593	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
989	894.239676933	e2:9f:83:45:e8:7c	b2:18:8e:9b:d0:cf	OpenFlow	126	Type: OFPT_PACKET_IN
990	894.269582944	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
992	894.270060425	b2:18:8e:9b:d0:cf	e2:9f:83:45:e8:7c	OpenFlow	126	Type: OFPT_PACKET_IN
993	894.271324300	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
995	899.000014314	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
997	899.045542636	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
999	903.999282784	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
1001	904.023605192	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
1003	909.000133701	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
1004	909.000734561	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
1006	914.000589405	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
1007	914.028078931	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY
1009	918.999804411	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REQUEST
1010	919.007519493	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_ECHO_REPLY

In above fig. you can see, when I use h1 ping –c 1 h2. OFPT_PACKET_IN is send from switch to the controller because of a flow table entry miss. Controller responses with OFPT_PACKET_OUT, which

contains output port number. Other echo messages are used to exchange liveness, latency and bandwidth.

Now we try send more packets, using h1 ping –c 10 h2. Since the controller is acting like a hub, you can see the Wireshark capture below contains PACKET_IN and PACKET_OUT for every packet it reaches to the switch. Because controller is instructing the switch to flood the network, in result switch doesn't has any flow entries. So, it need to contact the controller for each transmission of packet

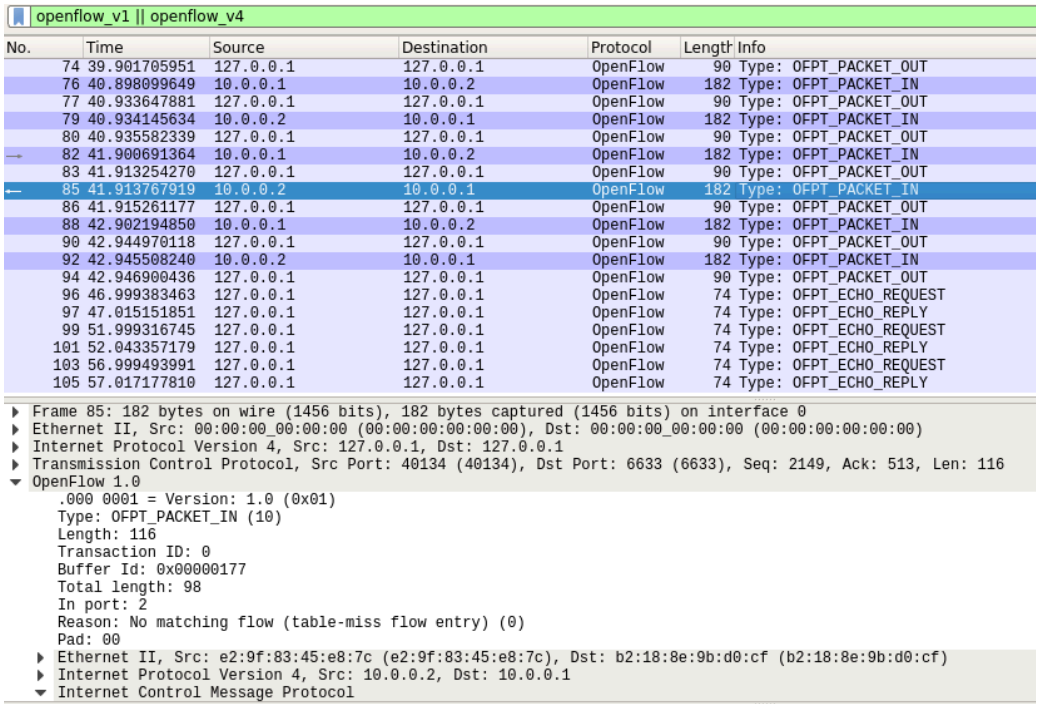
```
mininet> h1 ping -c 10 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=37.5 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=16.8 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=47.5 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=28.1 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=4.90 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=31.8 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=6.40 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=37.9 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=15.2 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=45.3 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9018ms
rtt min/avg/max/mdev = 4.909/27.173/47.504/14.714 ms
mininet>
```

openflow_v1 || openflow_v4

No.	Time	Source	Destination	Protocol	Length	Info
24	33.883552364	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
25	33.901798875	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
27	33.902375240	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
28	33.920367453	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
30	34.885653826	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
31	34.900219067	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
33	34.900978535	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
34	34.902044010	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
36	35.888074228	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
38	35.933092296	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
40	35.933603394	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
42	35.935079987	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
44	36.890096311	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
45	36.915863686	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
47	36.916326695	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
48	36.917653537	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
50	37.891734000	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
51	37.895151652	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
53	37.895517237	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
54	37.896136056	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
56	38.893491459	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
57	38.903541395	e2:9f:83:45:e8:7c	b2:18:8e:9b:d0:cf	OpenFlow	126	Type: OFPT_PACKET_IN
59	38.922519239	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
61	38.922687346	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
63	38.923500978	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
64	38.923515300	b2:18:8e:9b:d0:cf	e2:9f:83:45:e8:7c	OpenFlow	126	Type: OFPT_PACKET_IN
66	38.924633024	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
68	38.963238400	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
70	39.895916894	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
71	39.899795136	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
73	39.900591207	10.0.0.2	10.0.0.1	OpenFlow	182	Type: OFPT_PACKET_IN
74	39.901705951	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT
76	40.898099649	10.0.0.1	10.0.0.2	OpenFlow	182	Type: OFPT_PACKET_IN
77	40.933647881	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_PACKET_OUT

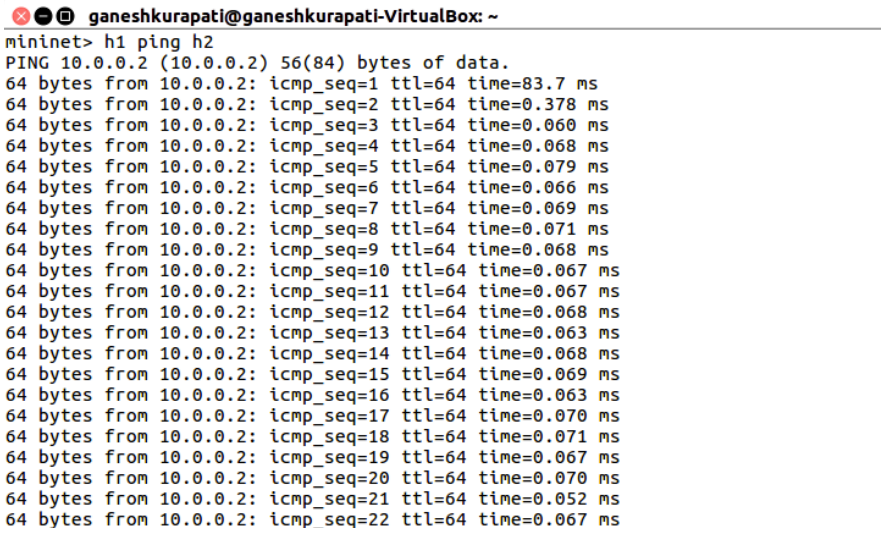
Above fig. shows packet capture for hub like behavior.



In above fig., you can observe the No match flow message is send in every PACKET_IN in hub like behavior.

Act Like a L2 Learning Switch:

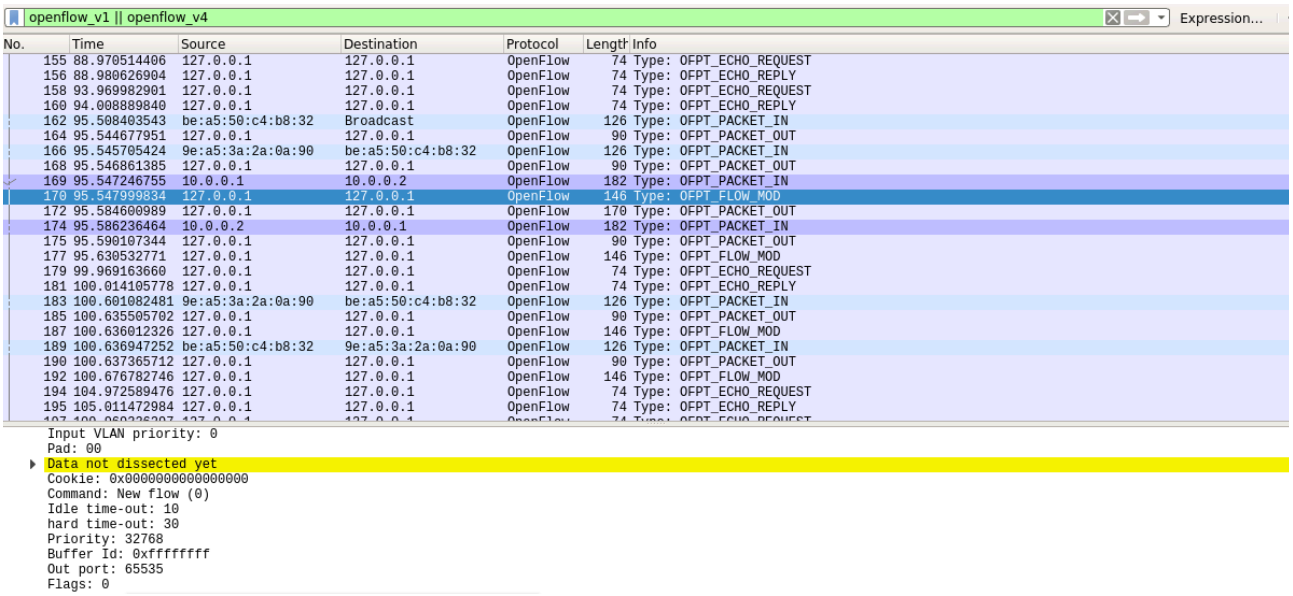
After configuring the controller to act like a switch, you can flow instructions given in above section to start the controller and topology. Also in this case, we use the same network topology. Then when we observe the packet capture, it contains few PACKET_IN, PACKET_OUT initially but after FLOW_MOD messages, you can't find any.



Shown in above fig., h1 pings h2.

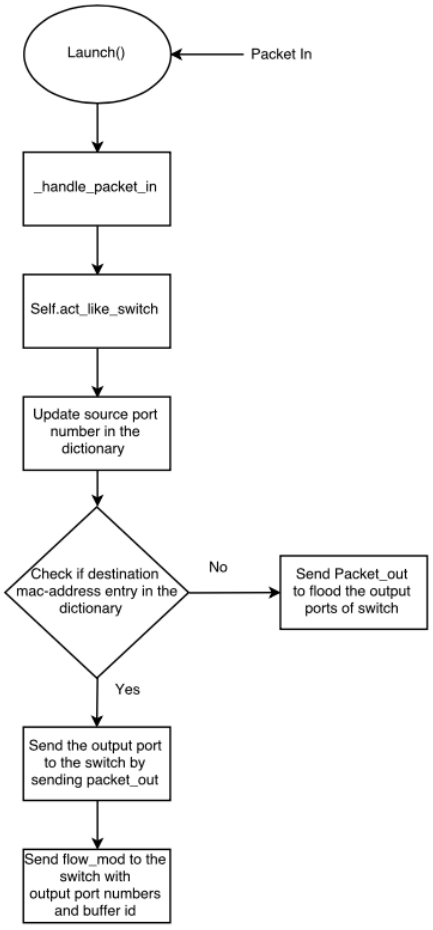
```
ganeshkurapati@ganeshkurapati-VirtualBox: ~/pox
ganeshkurapati@ganeshkurapati-VirtualBox:~$ cd pox
ganeshkurapati@ganeshkurapati-VirtualBox:~/pox$ ./pox.py log.level --DEBUG misc.
of_tutorial
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.12/Nov 19 2016 06:48:10)
DEBUG:core:Platform is Linux-4.4.0-62-generic-x86_64-with-Ubuntu-16.04-xenial
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 2]
DEBUG:misc.of_tutorial:Installing flow for 9e:a5:3a:2a:0a:90.2 -> be:a5:50:c4:b8
:32.1
DEBUG:misc.of_tutorial:Installing flow for be:a5:50:c4:b8:32.1 -> 9e:a5:3a:2a:0a
:90.2
DEBUG:misc.of_tutorial:Installing flow for 9e:a5:3a:2a:0a:90.2 -> be:a5:50:c4:b8
:32.1
DEBUG:misc.of_tutorial:Installing flow for 9e:a5:3a:2a:0a:90.2 -> be:a5:50:c4:b8
:32.1
DEBUG:misc.of_tutorial:Installing flow for be:a5:50:c4:b8:32.1 -> 9e:a5:3a:2a:0a
:90.2
DEBUG:misc.of_tutorial:Installing flow for be:a5:50:c4:b8:32.1 -> 9e:a5:3a:2a:0a
:90.2
```

Shown in the above fig., the controller is installing the flows in the switch by sending flow modification messages.



As shown above, you can see few PACKET_IN and PACKET_OUT. And FLOW_MOD messages also appear. If you see the flow modification message, it has time outs, buffer id etc.

c. Software



If observe the of_tutorial.py, launch () function calls the handle_packet_in (), which is configured to call another function act_like_switch(). In this lab, we need to write the code in this act_like_switch() in python program.

Now let us see the code line by line, which like the flow chart given above.

```
self.mac_to_port[packet.src]=packet_in.in_port
```

It checks for the source MAC address entry in the dictionary(mac_to_port). Then it updates the port number of the switch it received the packet.

```
if packet.dst in self.mac_to_port:
    self.resend_packet(packet_in, self.mac_to_port[packet.dst])
```

Check for the destination MAC address entry, if it has the entry then it sends the packet_out with the port number to send the packet. After it need to send the flow_mod message to update the flow entry of the switch.

```
log.debug("Installing flow for %s.%i -> %s.%i"
        % (packet.src, packet_in.in_port, packet.dst, self.mac_to_port[packet.dst]))
```
In above line we will log the flow entry

Below we generate flow modification message
```
msg = of.ofp_flow_mod()
msg.match = of.ofp_match.from_packet(packet)
```

To set the time out entries of flow modification message
```
msg.idle_timeout = 10
msg.hard_timeout = 30
```

To set the buffer id entry of flow modification message
```
msg.buffer_id = None
```

We bind the action to the message to establish the connection and send the message. Similar to resend_packet() function.
```
action =  of.ofp_action_output(port = self.mac_to_port[packet.dst])
msg.actions.append(action)
self.connection.send(msg)
```

If cannot find the destination mac-address entry, then packet_out to the switch saying it to flood the output ports. As shown below.
```
else:
    self.resend_packet(packet_in, of.OFPP_ALL)
```

You can find the source code in the appendix.

d. Trouble shooting:

First to check the mini-net, Pox controller setup, we need to check the default hub like behavior of the controller. We can capture the packets using Wireshark and visualize (see results section) the flow table entry using the command sudo ovs-ofctl dump-flows s1. We can cross check the data when we run learning switch.

Secondly, we can try to print the flow modification entry in the controller to see the source and destinations as shown below. And, we can see FLOW_MOD message in the Wireshark. This behavior different from the hub because it doesn't have flow mod messages in the packet captured.

```
DEBUG:misc.of_tutorial:Installing flow for 9e:a5:3a:2a:0a:90.2 -> be:a5:50:c4:b8
:32.1
DEBUG:misc.of_tutorial:Installing flow for be:a5:50:c4:b8:32.1 -> 9e:a5:3a:2a:0a
:90.2
DEBUG:misc.of_tutorial:Installing flow for 9e:a5:3a:2a:0a:90.2 -> be:a5:50:c4:b8
:32.1
DEBUG:misc.of_tutorial:Installing flow for 9e:a5:3a:2a:0a:90.2 -> be:a5:50:c4:b8
:32.1
DEBUG:misc.of_tutorial:Installing flow for be:a5:50:c4:b8:32.1 -> 9e:a5:3a:2a:0a
:90.2
DEBUG:misc.of_tutorial:Installing flow for be:a5:50:c4:b8:32.1 -> 9e:a5:3a:2a:0a
:90.2
```

Thirdly, if you see the code, we send packet_out as well as flow_mod if the dictionary has destination MAC address entry. Problem arises when we need to select buffer_id to send flow_mod. Buffer_id in this case should be None not the buffer_id in the packet_in because that Id does not exit since we send already packet_out. It throws error if we use buffer_id in the packet_in.

e. Results:

In the lab procedure section, we have seen that from the Wireshark packet capture that flow_mod messages are missing in hub like behavior, while it appeared in learning switch behavior. This message is responsible to install flows in the switch. And, we can see that after sending flow_mod, PACKET_IN & PACKET_OUT doesn't appear in learning switch whereas in hub PACKET_IN & PACKET_OUT appear for every packet send. We can also visualize the flow entries in the switch using ovs-ofctl command as shown below.

```
ganeshkurapati@ganeshkurapati-VirtualBox:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=3.651s, table=0, n_packets=4, n_bytes=392, idle_timeout=10
, hard_timeout=30, idle_age=0, icmp,vlan_tci=0x0000,dl_src=be:a5:50:c4:b8:32,dl_
dst=9e:a5:3a:2a:0a:90,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,icmp_
code=0 actions=output:2
 cookie=0x0, duration=3.611s, table=0, n_packets=3, n_bytes=294, idle_timeout=10
, hard_timeout=30, idle_age=0, icmp,vlan_tci=0x0000,dl_src=9e:a5:3a:2a:0a:90,dl_
dst=be:a5:50:c4:b8:32,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_
code=0 actions=output:1
```

Show in above fig., When the controller is acting like a learning switch, there are flow entries when start to ping (h1 ping h2). Whereas below fig., they are missing in hub like behavior since we are not sending flow_mod messages.

```
ganeshkurapati@ganeshkurapati-VirtualBox:~$ sudo ovs-ofctl dump-flows s1
[sudo] password for ganeshkurapati:
NXST_FLOW reply (xid=0x4):
ganeshkurapati@ganeshkurapati-VirtualBox:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
ganeshkurapati@ganeshkurapati-VirtualBox:~$
```

You can find the source code for the l2 learning switch in the git repository given below.
https://github.com/ganeshkurapati/Software-Defined-Networks---Spring-2017.git

f. Appendix:
Below is the code for the learning switch using POX controller.

of_tutorial.py

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

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

    if packet.dst in self.mac_to_port:
      # Send packet out the associated port
      self.resend_packet(packet_in, self.mac_to_port[packet.dst])

      # 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.)

      log.debug("Installing flow for %s.%i -> %s.%i"
                % (packet.src, packet_in.in_port, packet.dst,
                                  self.mac_to_port[packet.dst]))
      # Maybe the log statement should have source/destination/port?

      msg = of.ofp_flow_mod()
      #
      ## Set fields to match received packet
      msg.match = of.ofp_match.from_packet(packet)
      #
      #< Set other fields of flow_mod (timeouts? buffer_id?) >
      msg.idle_timeout = 10
      msg.hard_timeout = 30
      msg.buffer_id = None
      #Add an action to send to the specified port
      action =  of.ofp_action_output(port =
                          self.mac_to_port[packet.dst])
      msg.actions.append(action)

      # Send message to switch
      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)
```