

Selected Topics on Software Defined Networking

EECE.7290

Instructor: Prof. Yan Luo

Lab 2: Programming OpenDayLight

Hand in Date: 20/03/17

Due Date: 20/03/17

By,

- Naga Ganesh Kurapati

ID:01592079

Table of contents:

- a. Purpose
- b. Lab Procedure
- c. Software
- d. Trouble Shooting
- e. Results
- f. Appendix

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 Open daylight controller to design L2 learning switch in java programming language. Open-flow packets are captured using Wireshark.

b. Lab Procedure

Act Like a Hub:

Before we start to program Open daylight controller to act like a learning switch, we need to run it like a hub to troubleshoot the setup. You can find SDNHub tutorial in location `/home/ubuntu/SDNHub_OpenDaylight_tutorial`. Update the local repository using the command `git pull --rebase`.

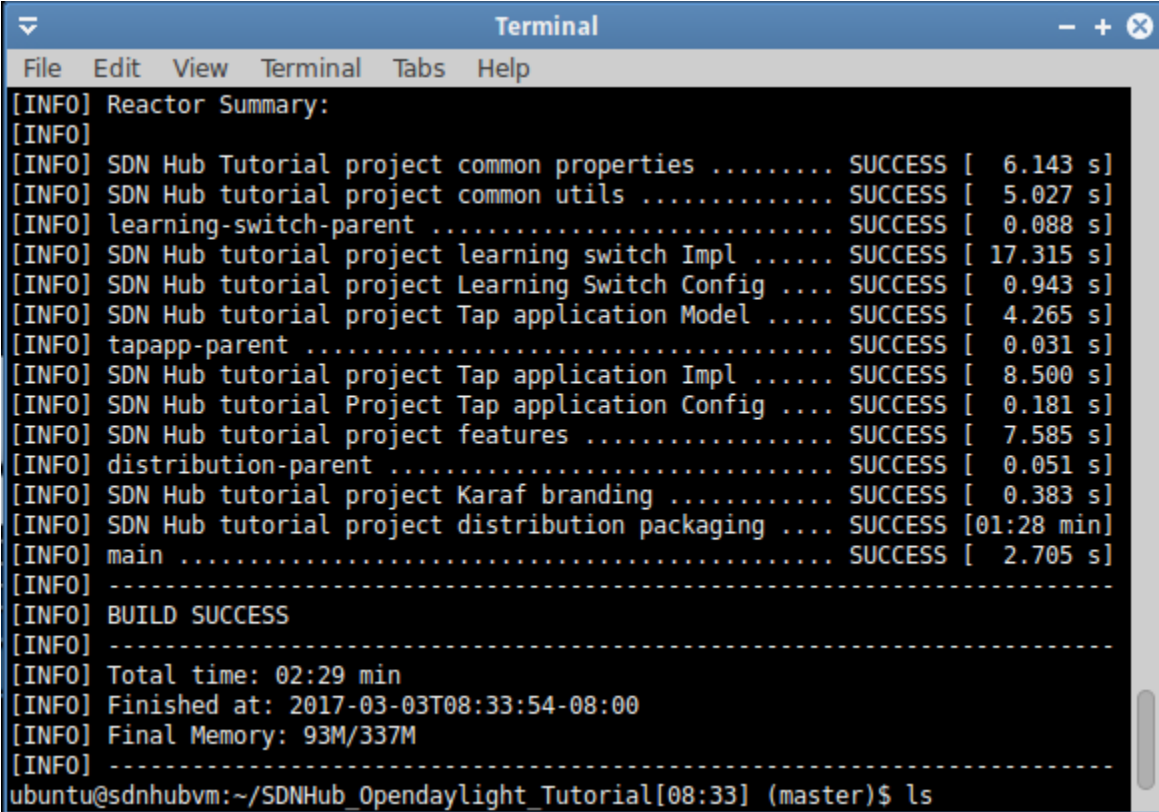
Before moving further, we need to know about

Maven: It is building automation tool uses .xml script files to build the project

Karaf: It is OSGi based runtime to load different modules.

Next we need to use the Maven command to clean and build the project as shown in the fig. below

```
mvn install -nsu
```



```
Terminal
File Edit View Terminal Tabs Help
[INFO] Reactor Summary:
[INFO]
[INFO] SDN Hub Tutorial project common properties ..... SUCCESS [ 6.143 s]
[INFO] SDN Hub tutorial project common utils ..... SUCCESS [ 5.027 s]
[INFO] learning-switch-parent ..... SUCCESS [ 0.088 s]
[INFO] SDN Hub tutorial project learning switch Impl ..... SUCCESS [ 17.315 s]
[INFO] SDN Hub tutorial project Learning Switch Config .... SUCCESS [ 0.943 s]
[INFO] SDN Hub tutorial project Tap application Model ..... SUCCESS [ 4.265 s]
[INFO] tapapp-parent ..... SUCCESS [ 0.031 s]
[INFO] SDN Hub tutorial project Tap application Impl ..... SUCCESS [ 8.500 s]
[INFO] SDN Hub tutorial Project Tap application Config .... SUCCESS [ 0.181 s]
[INFO] SDN Hub tutorial project features ..... SUCCESS [ 7.585 s]
[INFO] distribution-parent ..... SUCCESS [ 0.051 s]
[INFO] SDN Hub tutorial project Karaf branding ..... SUCCESS [ 0.383 s]
[INFO] SDN Hub tutorial project distribution packaging .... SUCCESS [01:28 min]
[INFO] main ..... SUCCESS [ 2.705 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:29 min
[INFO] Finished at: 2017-03-03T08:33:54-08:00
[INFO] Final Memory: 93M/337M
[INFO] -----
ubuntu@sdnhubvm:~/SDNHub_Opendaylight_Tutorial[08:33] (master)$ ls
```

Next we need to start the karaf in the location

```
cd distribution/opendaylight-karaf/target/assembly
```

And use the command to start it as shown in the fig. below

```
./bin/karaf
```

```
Terminal
File Edit View Terminal Tabs Help
arget[08:43] (master)$ cd assembly/
ubuntu@sdnhubvm:~/SDNHub_OpenDaylight_Tutorial/distribution/opendaylight-karaf/t
arget/assembly[08:44] (master)$ pwd
/home/ubuntu/SDNHub_OpenDaylight_Tutorial/distribution/opendaylight-karaf/target
/assembly
ubuntu@sdnhubvm:~/SDNHub_OpenDaylight_Tutorial/distribution/opendaylight-karaf/t
arget/assembly[08:44] (master)$ ./bin/karaf
karaf: Enabling Java debug options: -Xdebug -Xnoagent -Djava.compiler=NONE -Xrun
jdwp:transport=dt_socket,server=y,suspend=n,address=5005
Listening for transport dt_socket at address: 5005

SDNHub

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>Display all 111 possibilities? (y or n)
opendaylight-user@root>
opendaylight-user@root>
```

Now the controller is running. Next we need to start the mininet with a topology as shown below

```
sudo mn --topo single,3 --mac --switch ovsk,protocols=OpenFlow13 --controller remote
```

```
ubuntu@sdnhubvm:~[09:29]$ sudo mn --topo single,3 --mac --switch ovsk,protocols=
OpenFlow13 --controller remote
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable
From 10.0.0.1 icmp_seq=6 Destination Host Unreachable
^C
--- 10.0.0.2 ping statistics ---
9 packets transmitted, 0 received, +6 errors, 100% packet loss, time 8019ms
pipe 3
mininet>
```

If you try to ping h2 from h1 as shown above, it will be unreachable because the karaf did not load the learning switch module (in this case it is acting like hub). Use the command `feature:install sdnhub-tutorial-learning-switch` in the controller command line to activate it.

```
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 11:11 min
[INFO] Finished at: 2017-03-03T09:24:58-08:00
[INFO] Final Memory: 123M/495M
[INFO] -----
ubuntu@sdnhubvm:~/SDNHub_OpenDaylight_Tutorial[09:25] (master)$ cd distribution/
ubuntu@sdnhubvm:~/SDNHub_OpenDaylight_Tutorial/distribution/opendaylight-karaf/t
karaf: Enabling Java debug options: -Xdebug -Xnoagent -Djava.compiler=NONE -Xrun
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512m; sup
Listening for transport dt_socket at address: 5005

SDNHub

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>feature:install sdnhub-tutorial-learning-switch
```

If you try to ping again it fails because we need to install the flows in the switch manually. So, use the command `s1 ovs-ofctl add-flow tcp:127.0.0.1:6634 -OOpenFlow13 priority=1,action=output:controller` from the mininet command line. If you try to ping now, it will be successful as shown below.

```
mininet> h1 ping 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=43.6 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=13.3 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=6.03 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=8.40 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=7.17 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=7.86 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=9.46 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=16.1 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=5.77 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=9.55 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=8.79 ms
^C
--- 10.0.0.2 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10021ms
rtt min/avg/max/mdev = 5.771/12.382/43.610/10.300 ms
mininet>
```

If you observe the 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. It is shown below.

No.	Time	Source	Destination	Protocol	Length	Info
11	0.695751000	127.0.0.1	127.0.0.1	OpenFlow	82	Type: OFPT_HELLO
15	0.772676000	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_FEATURES_REQUEST
20	1.733384000	127.0.0.1	127.0.0.1	OpenFlow	82	Type: OFPT_HELLO
22	1.746340000	127.0.0.1	127.0.0.1	OpenFlow	82	Type: OFPT_HELLO
24	1.747728000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_FEATURES_REQUEST
26	1.748513000	127.0.0.1	127.0.0.1	OpenFlow	98	Type: OFPT_FEATURES_REPLY
28	1.797702000	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_ROLE_REQUEST
29	1.797908000	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_ROLE_REPLY
31	1.798172000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
32	1.798300000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REPLY
33	1.799832000	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_ROLE_REQUEST
34	1.800133000	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_ROLE_REPLY
35	1.800770000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REQUEST
36	1.801026000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REPLY
37	1.803860000	127.0.0.1	127.0.0.1	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE_FEATURES
▶ Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)						
▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)						
▶ Transmission Control Protocol, Src Port: 60945 (60945), Dst Port: 6633 (6633), Seq: 1, Ack: 1, Len: 16						
▼ OpenFlow 1.3						
0000	00 00 00 00 00 00 00 00	00 00 00 00 08 00 45 c0E.			
0010	00 44 c3 4f 40 00 40 06	78 a2 7f 00 00 01 7f 00	.D.0@.@. x.....			
0020	00 01 ee 11 19 e9 a5 04	3a e5 7c e1 90 46 80 18: ..F..			
0030	00 56 fe 38 00 00 01 01	08 0a 00 10 f9 ac 00 10	.V.8....			
0040	f9 a9 04 00 00 10 00 00	00 ff 00 01 00 08 00 00			
0050	00 10		..			

Now we try send more packets, using h1 ping 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 as shown below.

No.	Time	Source	Destination	Protocol	Length	Info
852	105.972307000	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_MULTIPART_REQUEST, OFPMP_METER
853	105.972807000	127.0.0.1	127.0.0.1	OpenFlow	82	Type: OFPT_MULTIPART_REPLY, OFPMP_METER
855	106.393763000	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_IN
857	106.400760000	127.0.0.1	127.0.0.1	OpenFlow	204	Type: OFPT_PACKET_OUT
858	106.402139000	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_IN
859	106.404986000	127.0.0.1	127.0.0.1	OpenFlow	204	Type: OFPT_PACKET_OUT
861	106.972821000	127.0.0.1	127.0.0.1	OpenFlow	441	Type: OFPT_PACKET_OUT
863	107.398824000	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_IN
864	107.401682000	127.0.0.1	127.0.0.1	OpenFlow	204	Type: OFPT_PACKET_OUT
866	107.403645000	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_IN
867	107.406366000	127.0.0.1	127.0.0.1	OpenFlow	204	Type: OFPT_PACKET_OUT
869	108.400905000	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_IN
870	108.401994000	127.0.0.1	127.0.0.1	OpenFlow	204	Type: OFPT_PACKET_OUT
872	108.402752000	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_IN
873	108.403821000	127.0.0.1	127.0.0.1	OpenFlow	204	Type: OFPT_PACKET_OUT
▶ Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)						
▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)						
▶ Transmission Control Protocol, Src Port: 60945 (60945), Dst Port: 6633 (6633), Seq: 1, Ack: 1, Len: 16						
▼ OpenFlow 1.3						
0000	00 00 00 00 00 00 00 00	00 00 00 00 08 00 45 c0E.			
0010	00 44 c3 4f 40 00 40 06	78 a2 7f 00 00 01 7f 00	.D.0@.@. x.....			
0020	00 01 ee 11 19 e9 a5 04	3a e5 7c e1 90 46 80 18: ..F..			
0030	00 56 fe 38 00 00 01 01	08 0a 00 10 f9 ac 00 10	.V.8....			
0040	f9 a9 04 00 00 10 00 00	00 ff 00 01 00 08 00 00			
0050	00 10		..			

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.

```
mininet> h1 ping 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=43.6 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=13.3 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=6.03 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=8.40 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=7.17 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=7.86 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=9.46 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=16.1 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=5.77 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=9.55 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=8.79 ms
^C
--- 10.0.0.2 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10021ms
rtt min/avg/max/mdev = 5.771/12.382/43.610/10.300 ms
mininet>
```

Shown in above fig., h1 pings h2.

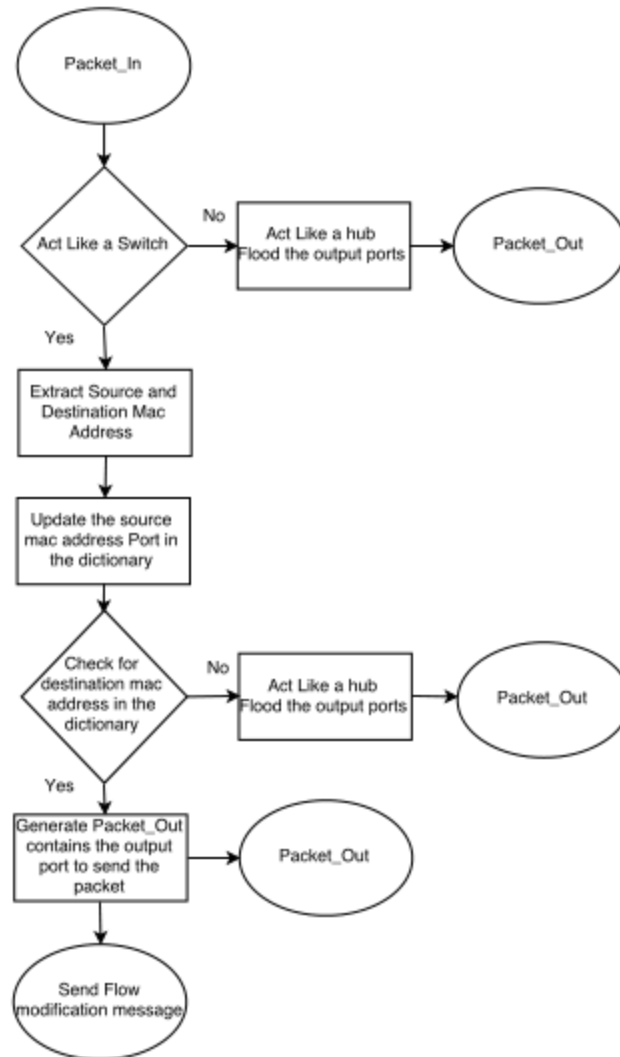
No.	Time	Source	Destination	Protocol	Length	Info
403	54.155449000	127.0.0.1	127.0.0.1	OpenFlow	82	Type: OFPT_MULTIPART_REPLY, OFPMP_METER
405	55.301678000	127.0.0.1	127.0.0.1	OpenFlow	150	Type: OFPT_PACKET_IN
407	55.331646000	127.0.0.1	127.0.0.1	OpenFlow	148	Type: OFPT_PACKET_OUT
408	55.332245000	127.0.0.1	127.0.0.1	OpenFlow	150	Type: OFPT_PACKET_IN
410	55.470159000	127.0.0.1	127.0.0.1	OpenFlow	148	Type: OFPT_PACKET_OUT
411	55.470999000	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_IN
413	55.510541000	127.0.0.1	127.0.0.1	OpenFlow	204	Type: OFPT_PACKET_OUT
414	55.511198000	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_IN
416	55.538881000	127.0.0.1	127.0.0.1	OpenFlow	204	Type: OFPT_PACKET_OUT
418	55.595636000	127.0.0.1	127.0.0.1	OpenFlow	354	Type: OFPT_FLOW_MOD
420	55.596120000	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_BARRIER_REQUEST
422	55.606414000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REPLY
423	55.606502000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REPLY
424	55.606538000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REPLY
425	57.022454000	127.0.0.1	127.0.0.1	OpenFlow	132	Type: OFPT_MULTIPART_REQUEST, OFPMP_FLOW

Frame 1: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface 0
▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▶ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 60839 (60839), Seq: 1, Ack: 1, Len: 56
▶ OpenFlow 1.3

0000 00 00 00 00 00 00 00 00 00 00 08 00 45 00E.
0010 00 6c 21 b2 40 00 40 06 1a d8 7f 00 00 01 7f 00 .l!.@.@
0020 00 01 19 e9 ed a7 4e 4e a5 85 51 de 10 6e 80 18NN ..Q..n..
0030 0c ff fe 60 00 00 01 01 08 0a 00 b7 af af 00 b7
0040 ac fd 04 12 00 38 00 00 01 28 00 01 00 00 00 008..(.
0050 00 00 ff 00 00 00 ff ff ff ff ff ff ff 00 00
[Click to change configuration profile](#)

Above you can see the Wireshark capture during act like switch behavior of the Open Daylight controller. After Flow modification, you can't find any of the PACKET_IN, PACKET_OUT messages for h1 ping h2.

c. Software



Now we will see the code line by line. We need to write the code in the file TutorialL2Forwarding.java under SDN hub tutorial. Projects are visible in Eclipse. When the packet_in comes in

If it is acting like a switch it need to payload, from that it need to abstract destination and mac address as shown below

Extract payload

```
byte[] payload = notification.getPayload();
```

Extract destination and source MAC address from the that payload

```
byte[] dstMacRaw = PacketParsingUtils.extractDstMac(payload);  
byte[] srcMacRaw = PacketParsingUtils.extractSrcMac(payload);
```

Next we need to convert that raw mac addresses to string form so that it can used for comparisons and storage in the program. It is shown below


```
String srcMac = PacketParsingUtils.rawMacToString(srcMacRaw);
String dstMac = PacketParsingUtils.rawMacToString(dstMacRaw);
```

We are using Hash Map in java as a lookup table

```
private Map<String, NodeConnectorId> macTable = new HashMap <String, NodeConnectorId>();
```

Here the key is Mac address (String form) and the storage element is the port number.

First we need to learn the port number of the source mac address, for that we need to update entry of that mac table.

Learn source MAC address

```
this.macTable.put(srcMac, ingressNodeConnectorId);
```

Now look for destination mac address entry

```
NodeConnectorId egressNodeConnectorId = this.macTable.get(dstMac);
```

If the entry doesn't exist, we need send the packet_out contains output port and the flow modification message as well

```
packetOut(ingressNodeRef, InventoryUtils.getNodeConnectorRef(egressNodeConnectorId), payload);
```

Flow modification message:

```
programL2Flow(ingressNodeId, dstMac, ingressNodeConnectorId, egressNodeConnectorId);
```

If entry doesn't exist we need to flood the output ports,

```
packetOut(ingressNodeRef, floodNodeConnectorRef, payload);
```

We see the design of programL2Flow (Flow modification function) in more detail 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 received packets using tcpdump. We can cross check the data when we run learning switch.

When controller is acting like a hub, in the topology we have 3 hosts h1, h2, h3. If we open xterm and initiate tcpdump using the command tcpdump -i h2-eth0 at host h2 and tcpdump -i h3-eth0 at host h3 to capture incoming packets. We can observe the packets capture for h1 ping h2 in both h2 and h3, you can the flooding as shown below.

```

"Node: h1"
root@sdnhubvm:~[11:46]$ ping 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=35.7 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=11.2 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=14.5 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=4.11 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=19.2 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=6.71 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=5.05 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=28.7 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=16.8 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=7.97 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=19.0 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=5.60 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=8.13 ms
^C
--- 10.0.0.2 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12019ms
rtt min/avg/max/mdev = 4.111/14.083/35.733/9.329 ms
root@sdnhubvm:~[11:47]$
```

```

"Node: h2"
11:47:47.678145 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7094, seq 11, length 64
11:47:48.662461 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7094, seq 12, length 64
11:47:48.662508 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7094, seq 12, length 64
11:47:48.665860 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7094, seq 12, length 64
11:47:49.664118 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7094, seq 13, length 64
11:47:49.664148 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7094, seq 13, length 64
11:47:49.670141 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7094, seq 13, length 64
```

```

"Node: h3"
64
11:47:46.778183 LLDP, length 71: openflow:1
11:47:47.663312 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7094, seq 11, length 64
11:47:47.678150 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7094, seq 11, length 64
11:47:48.662463 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7094, seq 12, length 64
11:47:48.665862 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7094, seq 12, length 64
11:47:49.664121 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7094, seq 13, length 64
11:47:49.670143 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7094, seq 13, length 64
```

For the controller acting like a switch, things are different. We can see the packets in the host h2 window but not in h3 window. We can see those screen shots in results section. We can see that, in Wireshark capture, we can see packet_in, packet_out for each packet transmission in hub like behavior but not in switch like behavior.

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 tcpdump capture as described in troubleshooting section for switch like behavior below. It equivalent to h1 ping h2

```
Node: h1
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:8 errors:0 dropped:0 overruns:0 frame:0
TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:896 (896.0 B) TX bytes:896 (896.0 B)

root@sdnhubvm:~# ping 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=0.378 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.181 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.128 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.012 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.151 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.157 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.151 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.144 ms
^C
--- 10.0.0.2 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8005ms
rtt min/avg/max/mdev = 0.012/0.154/0.378/0.093 ms
root@sdnhubvm:~#
```

```
Node: h2
64
15:04:45.914603 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
15:04:45.916122 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01 (oui Ethernet), length 28
15:04:46.089871 LLDP, length 71: openflow:1
15:04:46.903472 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 21985, seq 7, length 64
15:04:46.903525 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 21985, seq 7, length 64
15:04:47.902547 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 21985, seq 8, length 64
15:04:47.902576 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 21985, seq 8, length 64
15:04:48.904045 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 21985, seq 9, length 64
15:04:48.904086 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 21985, seq 9, length 64
```

```

"Node: h3"
root@sdnhubvm:~[14:38]$ tcpdump -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:04:31.089717 LLDP, length 71: openflow:1
15:04:36.089273 LLDP, length 71: openflow:1
15:04:41.091307 LLDP, length 71: openflow:1
15:04:46.089933 LLDP, length 71: openflow:1
15:04:51.089056 LLDP, length 71: openflow:1
15:04:56.092771 LLDP, length 71: openflow:1
15:05:01.088934 LLDP, length 71: openflow:1
15:05:06.089227 LLDP, length 71: openflow:1
15:05:11.089306 LLDP, length 71: openflow:1
15:05:16.088232 LLDP, length 71: openflow:1

```

You can see in the h3 host xterm window, in switch like behavior we can say that packets are not flooding. And in the lab procedure section we can find packet_in and packet_out are missing after flow modification message.

No.	Time	Source	Destination	Protocol	Length	Info
403	54.153449000	127.0.0.1	127.0.0.1	OpenFlow	82	Type: OFPT_MULTIPART_REPLY, OFPMP_METER
405	55.301678000	127.0.0.1	127.0.0.1	OpenFlow	150	Type: OFPT_PACKET_IN
407	55.331646000	127.0.0.1	127.0.0.1	OpenFlow	148	Type: OFPT_PACKET_OUT
408	55.332245000	127.0.0.1	127.0.0.1	OpenFlow	150	Type: OFPT_PACKET_IN
410	55.470159000	127.0.0.1	127.0.0.1	OpenFlow	148	Type: OFPT_PACKET_OUT
411	55.470999000	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_IN
413	55.510541000	127.0.0.1	127.0.0.1	OpenFlow	204	Type: OFPT_PACKET_OUT
414	55.511198000	127.0.0.1	127.0.0.1	OpenFlow	206	Type: OFPT_PACKET_IN
416	55.538881000	127.0.0.1	127.0.0.1	OpenFlow	204	Type: OFPT_PACKET_OUT
418	55.595636000	127.0.0.1	127.0.0.1	OpenFlow	354	Type: OFPT_FLOW_MOD
420	55.596120000	127.0.0.1	127.0.0.1	OpenFlow	90	Type: OFPT_BARRIER_REQUEST
422	55.606414000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REPLY
423	55.606502000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REPLY
424	55.606538000	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_BARRIER_REPLY
426	57.022451000	127.0.0.1	127.0.0.1	OpenFlow	172	Type: OFPT_MULTIPART_REQUEST, OFPMP_FLOW

Filter: openflow_v4 Expression... Clear Apply Save
 Frame 1: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface 0
 ▶ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
 ▶ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
 ▶ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 60839 (60839), Seq: 1, Ack: 1, Len: 56
 ▶ OpenFlow 1.3

```

0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
0010  00 6c 21 b2 40 00 40 06 1a d8 7f 00 00 01 7f 00  .!!.@. ....
0020  00 01 19 e9 ed a7 4e 4e a5 85 51 de 10 6e 80 18  ....NN ..Q.n..
0030  0c ff fe 60 00 00 01 01 08 0a 00 b7 af af 00 b7  ....'.....
0040  ac fd 04 12 00 38 00 00 01 28 00 01 00 00 00 00  ....8.. .(.....
0050  00 00 ff 00 00 00 ff ff ff ff ff ff ff 00 00  .....

```

Click to change configuration profile

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/blob/master/Lab2/TutorialL2Forwarding.java>

f. Appendix:

Below is the code for the learning switch using Open daylight controller.

```
/*-----Function 1-----*/
public void onPacketReceived(PacketReceived notification) {
    LOG.trace("Received packet notification {}", notification.getMatch());

    NodeConnectorRef ingressNodeConnectorRef = notification.getIngress();
    NodeRef ingressNodeRef = InventoryUtils.getNodeRef(ingressNodeConnectorRef);
    NodeConnectorId ingressNodeConnectorId =
        InventoryUtils.getNodeConnectorId(ingressNodeConnectorRef);
    NodeId ingressNodeId = InventoryUtils.getNodeId(ingressNodeConnectorRef);

    // Useful to create it beforehand
    NodeConnectorId floodNodeConnectorId =
        InventoryUtils.getNodeConnectorId(ingressNodeId, FLOOD_PORT_NUMBER);
    NodeConnectorRef floodNodeConnectorRef =
        InventoryUtils.getNodeConnectorRef(floodNodeConnectorId);
    //Ignore LLDP packets, or you will be in big trouble
    byte[] etherTypeRaw =
        PacketParsingUtils.extractEtherType(notification.getPayload());
    int etherType = (0x0000ffff & ByteBuffer.wrap(etherTypeRaw).getShort());
    if (etherType == 0x88cc) {
        return;
    }

    // Hub implementation
    if (function.equals("hub")) {

        //flood packet (1)
        packetOut(ingressNodeRef, floodNodeConnectorRef, notification.getPayload());
    } else {
        LOG.debug("****Act Like a Switch****");
        //TODO: Extract payload
        byte[] payload = notification.getPayload();
        //TODO: Extract MAC address (2.1)
        byte[] dstMacRaw = PacketParsingUtils.extractDstMac(payload);
        byte[] srcMacRaw = PacketParsingUtils.extractSrcMac(payload);

        String srcMac = PacketParsingUtils.rawMacToString(srcMacRaw);
        String dstMac = PacketParsingUtils.rawMacToString(dstMacRaw);

        //TODO: Learn source MAC address (2.2)
        this.macTable.put(srcMac, ingressNodeConnectorId);
        //TODO: Lookup destination MAC address in table (2.3)
        NodeConnectorId egressNodeConnectorId = this.macTable.get(dstMac);

        //TODO: If found (2.3.1)
        if (egressNodeConnectorId != null) {
            //TODO: 2.3.1.1 perform FLOW_MOD for that dst_mac through the target node connector
            programL2Flow(ingressNodeId, dstMac, ingressNodeConnectorId, egressNodeConnectorId);
            //TODO: 2.3.1.2 perform PACKET_OUT of this packet to target node connector
            packetOut(ingressNodeRef,
                InventoryUtils.getNodeConnectorRef(egressNodeConnectorId), payload);
        } else {
            //2.3.2 Flood packet
            packetOut(ingressNodeRef, floodNodeConnectorRef, payload);
        }
    }
}
```

```

/*-----Function2-----*/
private void programL2Flow(NodeId nodeId, String dstMac, NodeConnectorId
ingressNodeConnectorId, NodeConnectorId egressNodeConnectorId) {

    //Creating match object
    MatchBuilder matchBuilder = new MatchBuilder();
    MatchUtils.createEthDstMatch(matchBuilder, new MacAddress(dstMac), null);
    MatchUtils.createInPortMatch(matchBuilder, ingressNodeConnectorId);

    // Instructions List Stores Individual Instructions
    InstructionsBuilder isb = new InstructionsBuilder();
    List<Instruction> instructions = Lists.newArrayList();
    InstructionBuilder ib = new InstructionBuilder();
    ApplyActionsBuilder aab = new ApplyActionsBuilder();
    ActionBuilder ab = new ActionBuilder();
    List<Action> actionList = Lists.newArrayList();

    // Set output action
    OutputActionBuilder output = new OutputActionBuilder();
    output.setOutputNodeConnector(egressNodeConnectorId);
    output.setMaxLength(65535); //Send full packet and No buffer
    ab.setAction(new OutputActionCaseBuilder().setOutputAction(output.build()).build());
    ab.setOrder(0);
    ab.setKey(new ActionKey(0));
    actionList.add(ab.build());

    // Create Apply Actions Instruction
    aab.setAction(actionList);
    ib.setInstruction(new
        ApplyActionsCaseBuilder().setApplyActions(aab.build()).build());
    ib.setOrder(0);
    ib.setKey(new InstructionKey(0));
    instructions.add(ib.build());

    // Create Flow
    String flowId = "L2_Rule_" + dstMac;
    FlowBuilder flowBuilder = new FlowBuilder();
    flowBuilder.setMatch(matchBuilder.build());
    flowBuilder.setId(new FlowId(flowId));
    FlowKey key = new FlowKey(new FlowId(flowId));
    flowBuilder.setBarrier(true);
    flowBuilder.setTableId((short)0);
    flowBuilder.setKey(key);
    flowBuilder.setPriority(32768);
    flowBuilder.setFlowName(flowId);
    flowBuilder.setHardTimeout(0);
    flowBuilder.setIdleTimeout(0);
    flowBuilder.setInstructions(isb.setInstruction(instructions).build());

    // Perform transaction to store rule

    InstanceIdentifier<Flow> flowIID = InstanceIdentifier.builder(Nodes.class)
        .child(Node.class, new NodeKey(nodeId))
        .augmentation(FlowCapableNode.class)
        .child(Table.class, new TableKey(flowBuilder.getTableId()))
        .child(Flow.class, flowBuilder.getKey())
        .build();
    GenericTransactionUtils.writeData(dataBroker, LogicalDatastoreType.CONFIGURATION,
        flowIID, flowBuilder.build(), true);
}
}

```