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. <u>Purpose</u>

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.

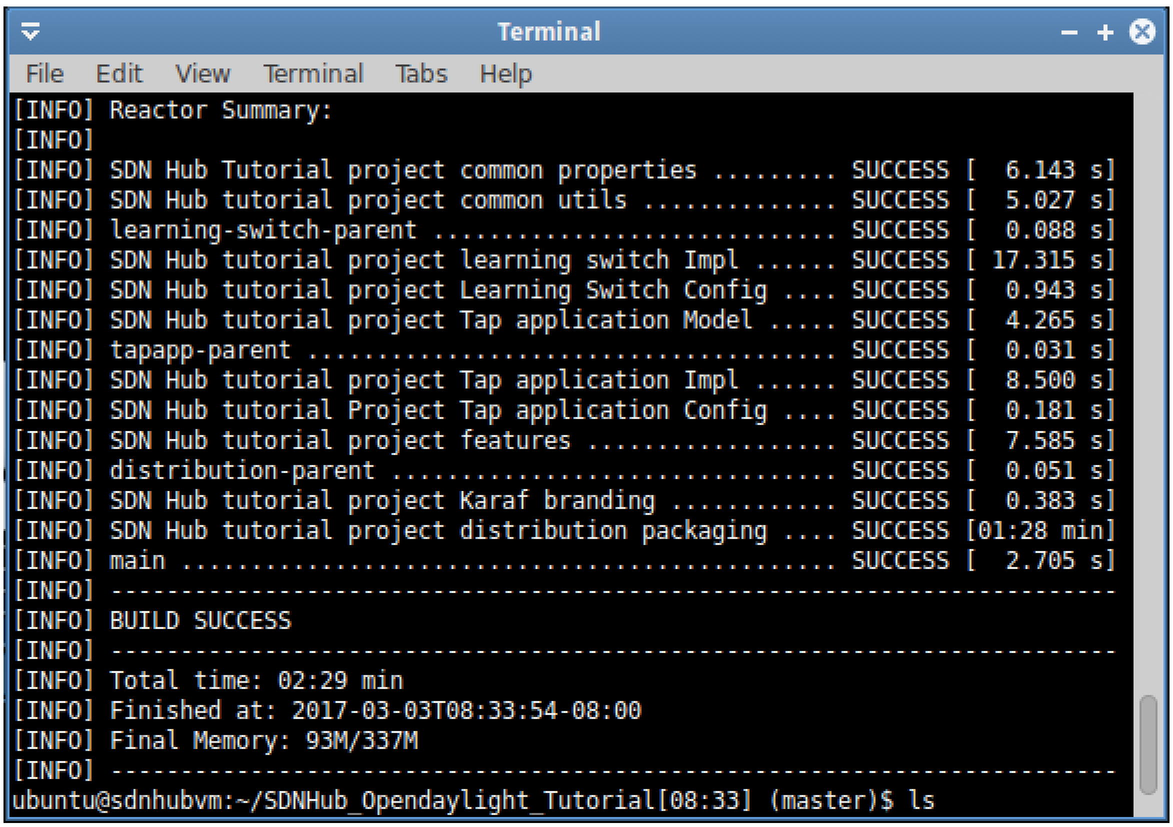
b. <u>Lab Procedure</u>

<u>Act Like a Hub:</u>

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`.

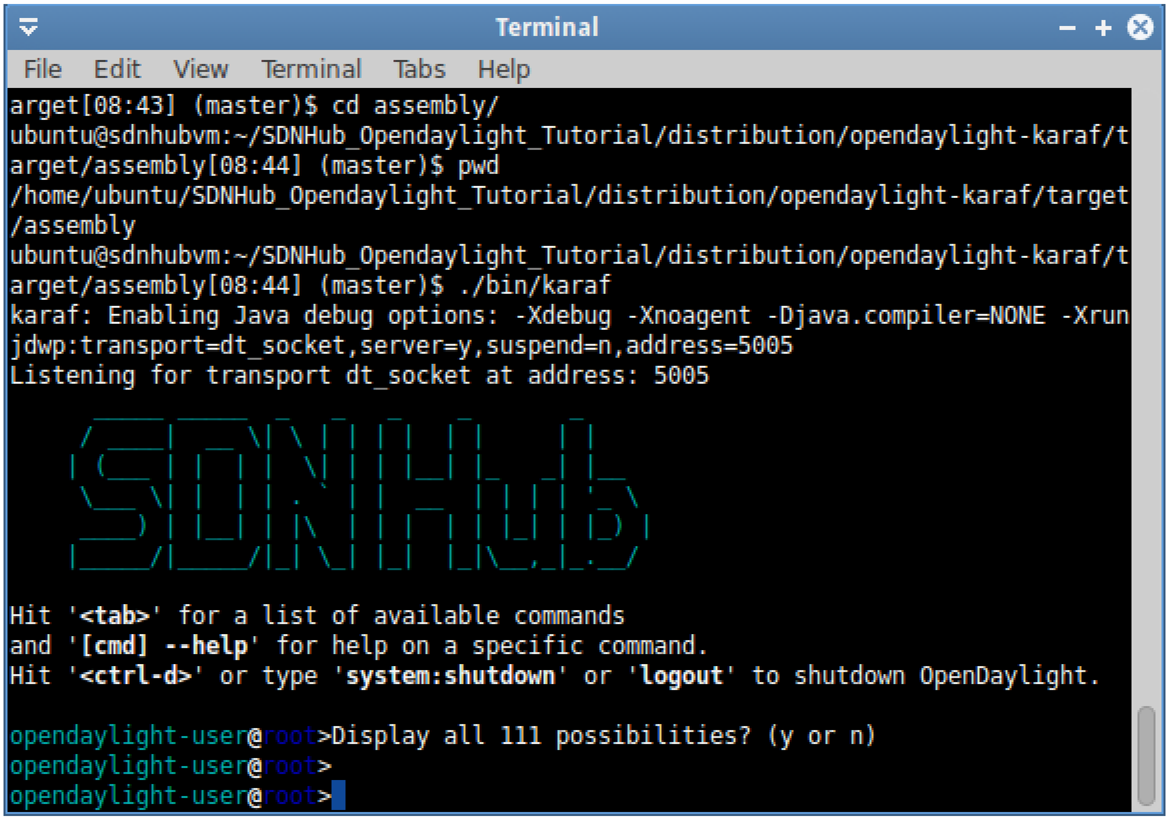
Before moving further, we need to know about
Maven: It is building automation tool uses .xml script files to build the project
Karaf: It is OSGi based runtime to load different modules.

Next we need to use the Maven command to clean and build the project as shown in the fig. below
`mvn install -nsu`



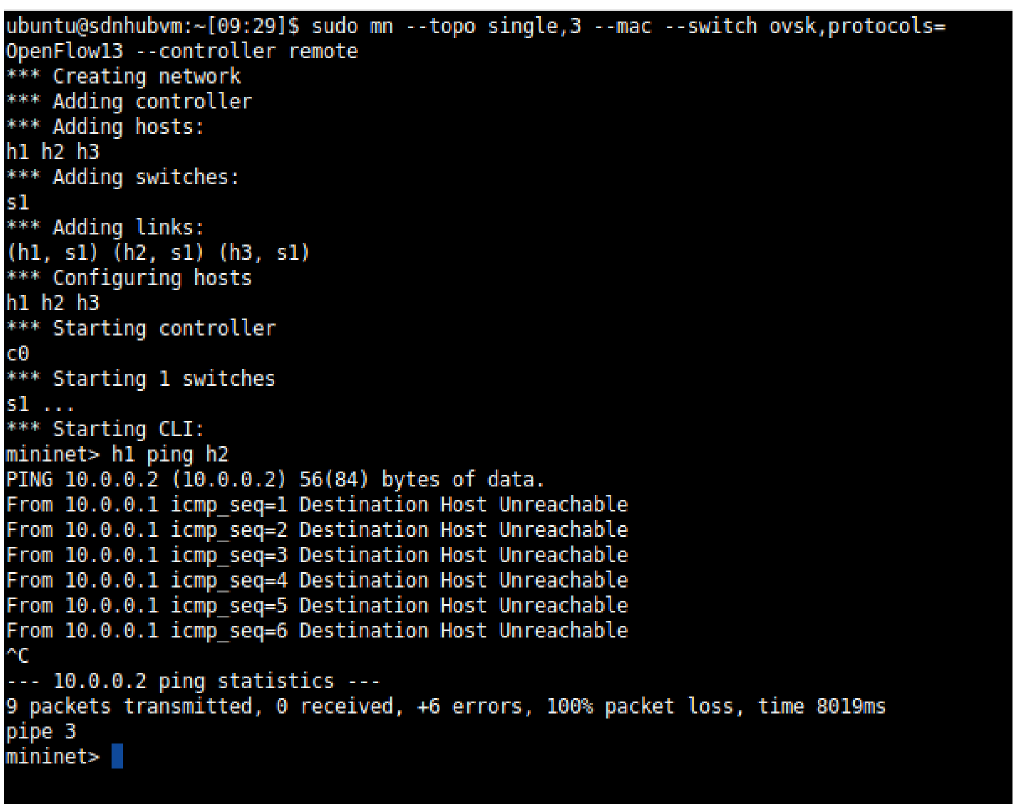
Next we need to start the karaf in the location
`cd distribution/opendaylight-karaf/target/assembly`

And use the command to start it as shown in the fig. below
`./bin/karaf`



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
```



If you try to ping h2 from h1 as shown above, it will be unreachable because the karaf did not loaded 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

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
```

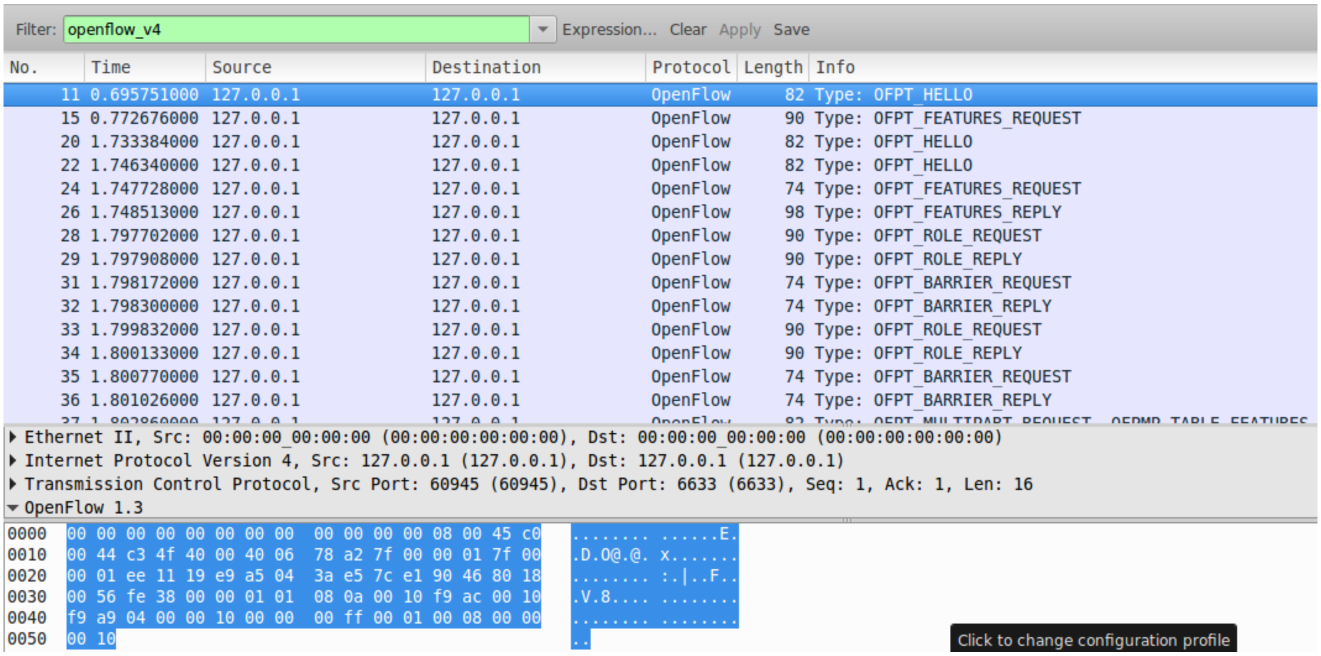
If you try to ping again it fails because we need to install the flows in the switch manually. So, use the command

`s1 ovs-ofctl add-flow tcp:127.0.0.1:6634 -OOpenFlow13 priority=1,action=output:controller`

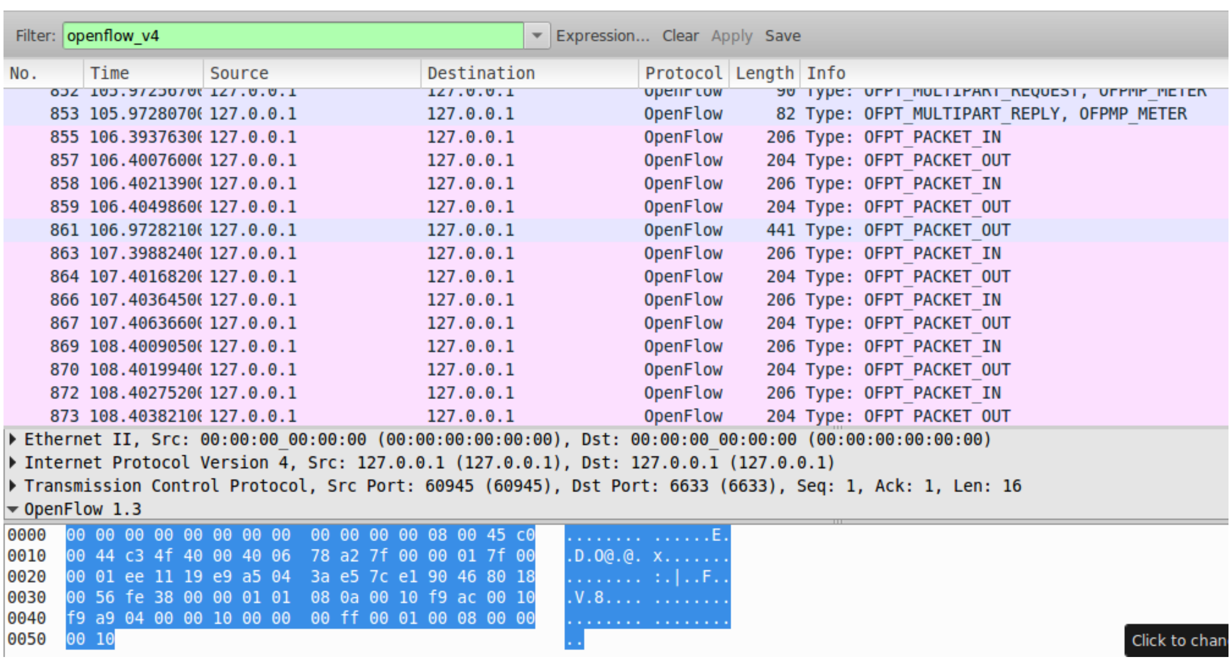
from the mininet command line. If you try to ping now, it will be successful as shown below.

```
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.

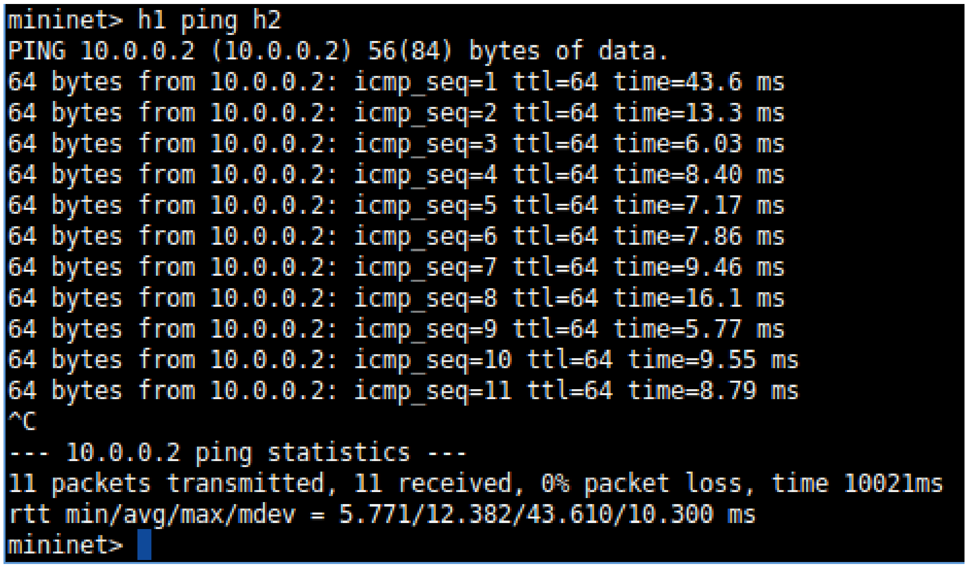


Now we try send more packets, using h1 ping h2. Since the controller is acting like a hub, you can see the Wireshark capture below contains PACKET_IN and PACKET_OUT for every packet it reaches to the switch. Because controller is instructing the switch to flood the network, in result switch doesn’t has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

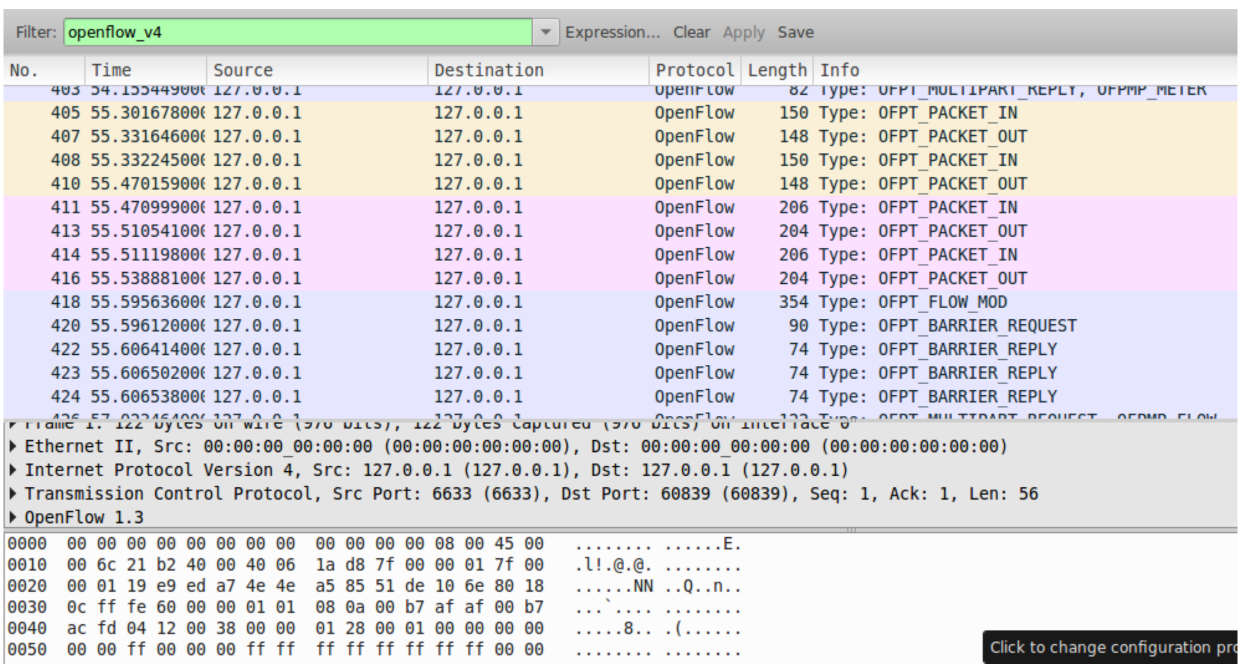


Act Like a L2 Learning Switch:

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.

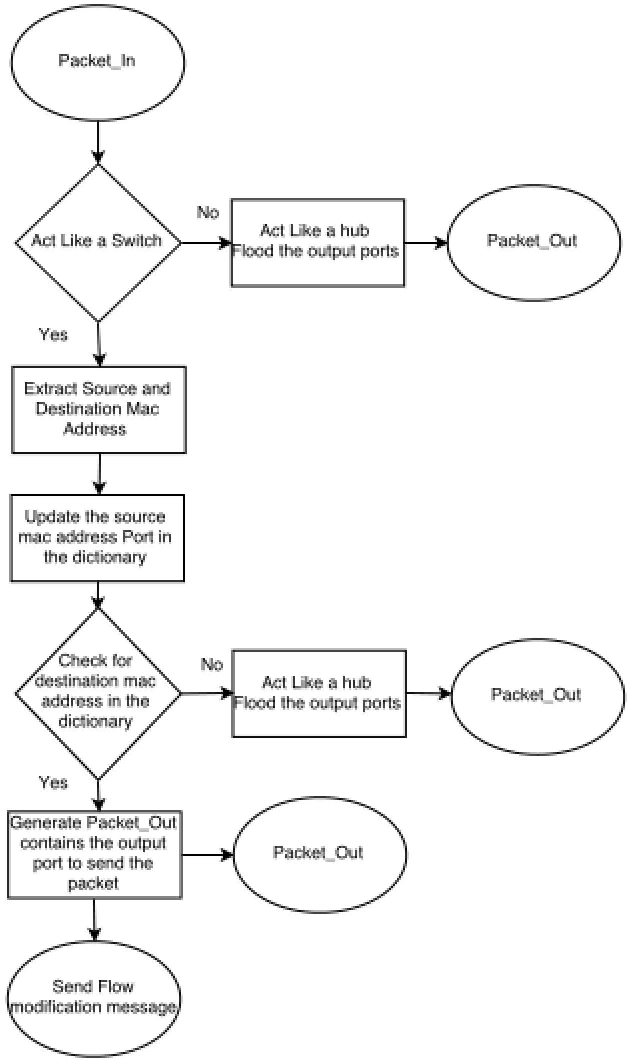


Shown in above fig., h1 pings h2.



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 exit we need to flood the output ports,

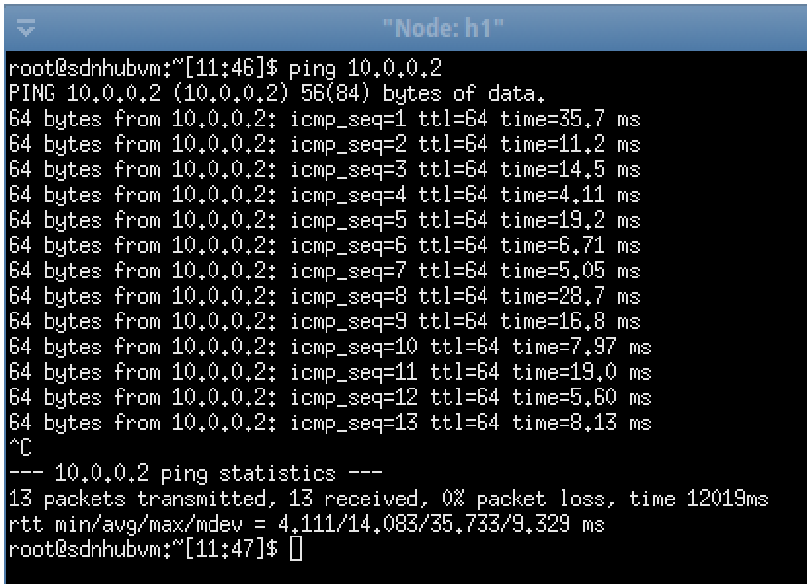
```
packetOut(ingressNodeRef, floodNodeConnectorRef, payload);
```

We see the design of programL2Flow (Flow modification function) in more detail in the appendix.

d. Trouble shooting:

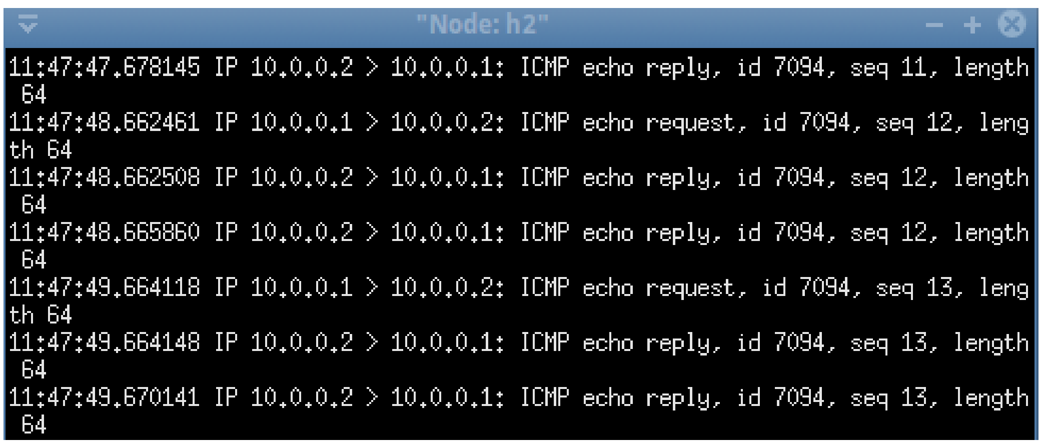
First to check the mini-net, Pox controller setup, we need to check the default hub like behavior of the controller. We can capture the packets using Wireshark and visualize (see results section) the received packets using tcpdump. We can cross check the data when we run learning switch.

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 h2, you can the flooding as shown below.



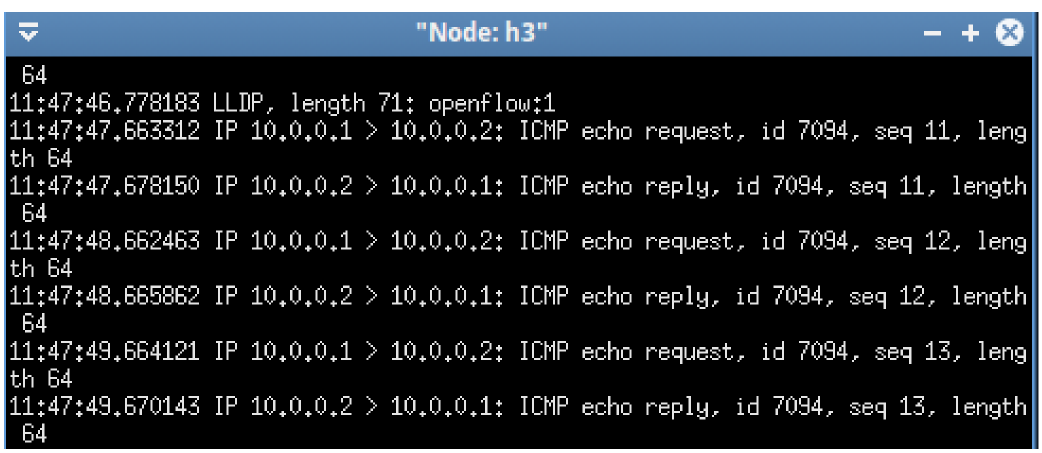
"Node: h1"

```
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, leng
th 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, leng
th 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
```



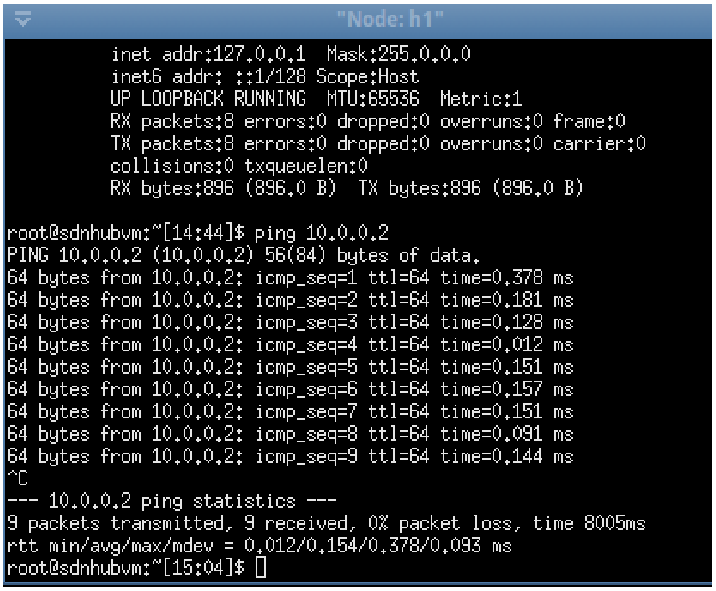
"Node: h3"

```
 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, leng
th 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, leng
th 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, leng
th 64
11:47:49.670143 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7094, seq 13, length
 64
```

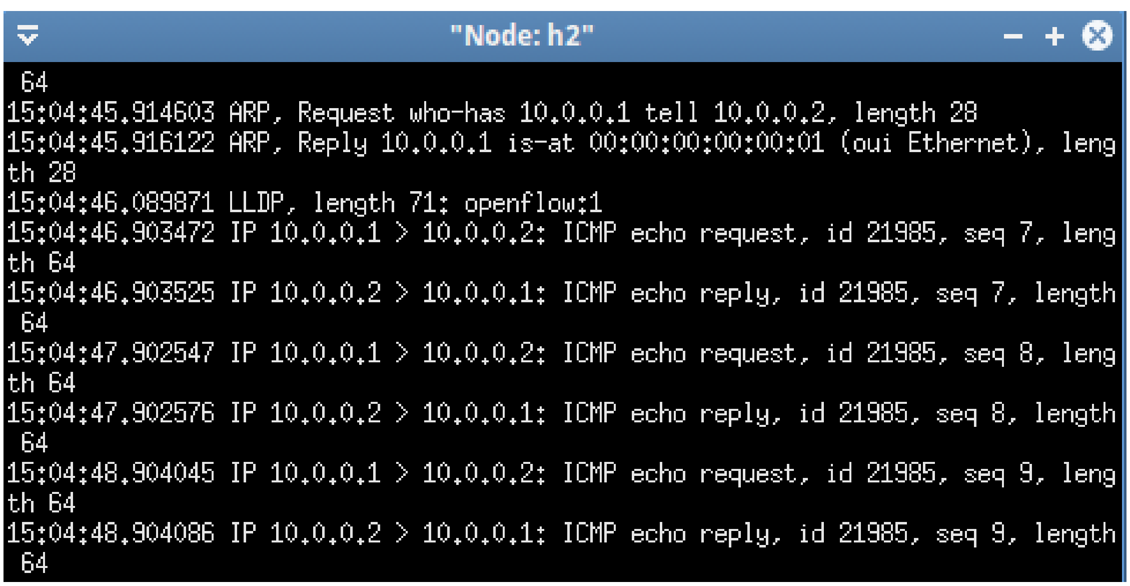
For the controller acting like a switch, things are different. We can see the packets in the host h2 window but not in h3 window. We can see those screen shots in results section. We can see that, in Wireshark capture, we can see packet_in,packet_out for each packet transmission in hub like behavior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

e. Results:

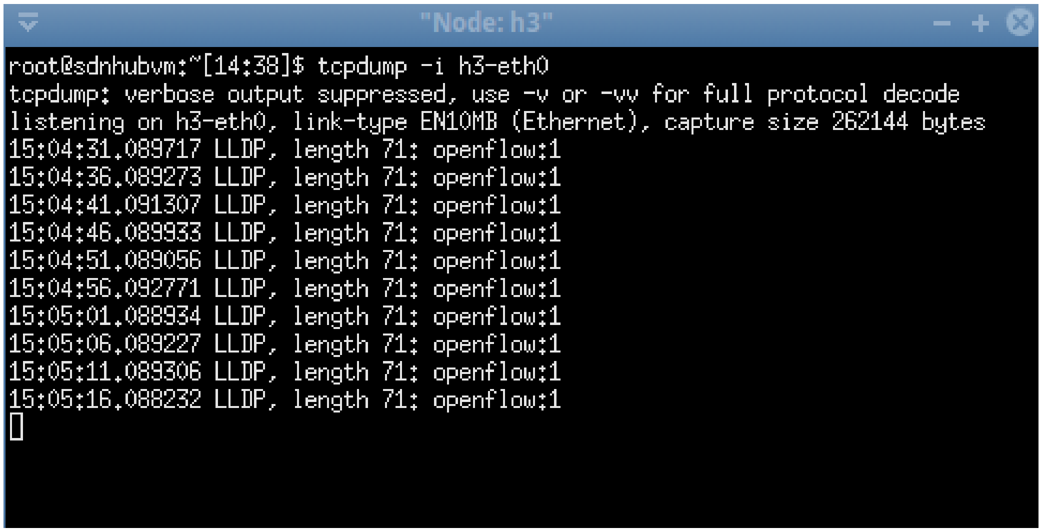
In the lab procedure section, we have seen that from the Wireshark packet capture that flow_mod messages are missing in hub like behavior, while it appeared in learning switch behavior. This message is responsible to install flows in the switch. And, we can see that after sending flow_mod, PACKET_IN & 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



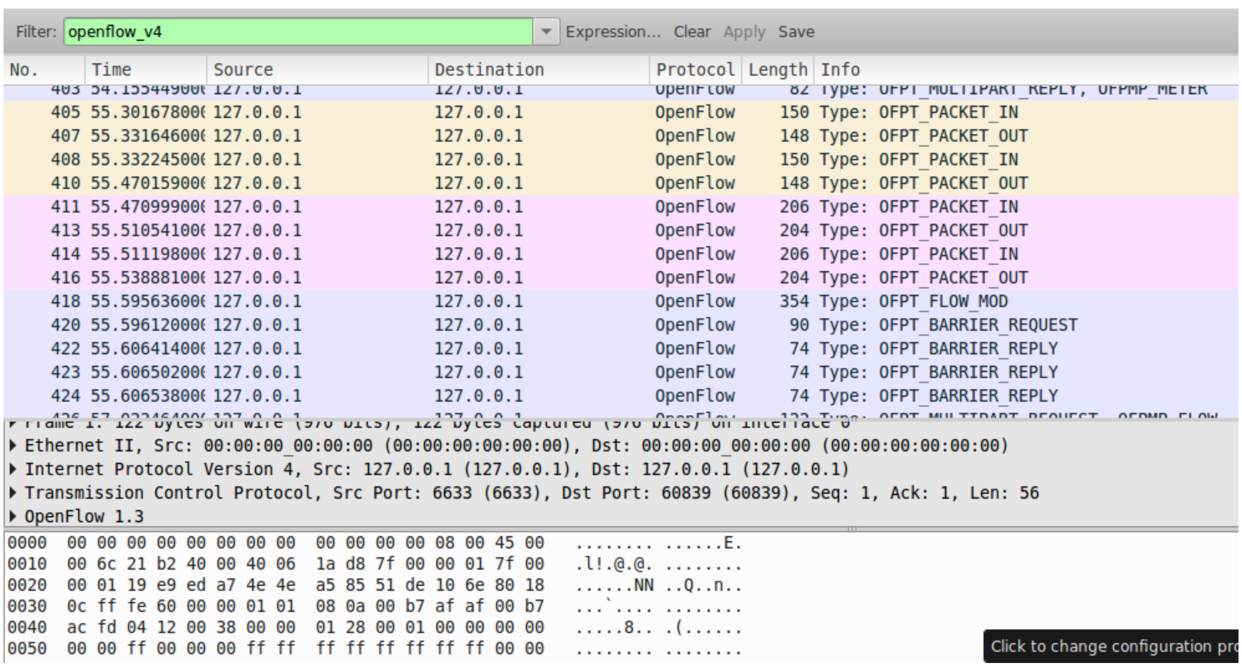
```
"Node: h1"
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0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

root@sdnhubvm:~[14:44]$ 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:~[15:04]$
```



```
"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), leng
th 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, leng
th 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, leng
th 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, leng
th 64
15:04:48.904086 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 21985, seq 9, length
 64
```



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.



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. <u>Appendix:</u>

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);
             }
        }
    }
```

```java
/*-------------------------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);
    }
}
```