ASSIGNMENT 5

Implementing the L2 forwarding and L3 static routing protocol

Arijit Panigrahy | 14CS30005

Aniket Choudhary | 14CS30004

19.03.2017

Computer Networks Laboratory

OBJECTIVE

To understand the L2 forwarding and L3 routing mechanism and implement the forwarding and routing protocols as a part of POX controller.

STEPS FOLLOWED

This assignment consists of 5 major parts (or stages) as follows :-

- L2 forwarding
  This step involves writing the controller protocol for the layer 2 switches. The switch works based on the following algorithm:-
  1. When a packet is received, the source mac and the in-port is stored in the mac_to_interface table.
  2. Then, if destination mac is already present in the mac_to_interface table, we create a flow rule corresponding to source_mac->destination_mac as well as destination_mac->source_mac with some timeout values.
  3. Else, if we don't know the destination mac, we simply FLOOD the packet across the network. No flow rules are created here.

  The above algorithm is implemented in the code l2.py. All the layer 2 switches in our topology will be registered to this controller.

- L3 forwarding
  This step involves writing the controller protocol for the layer 3 routers. The router works based on the following algorithm:-
  1. Initialize the routing table from the config.json based on the dpid values and listen to the core.openflow (i.e. wait for the packet to arrive)
  2. If the received packet is of LLDP type, simply drop it.
  3. If an ARP packet is received, store the source_mac and source_ip in the mac_to_ip table. Try sending the waiting packets (the packets for which we did not know the mac and enqueued to the lost_buffer queue), based on the mac we inserted in the table now. If the packet is an ARP request, and the request is for one of the interfaces of the router, then we construct the arp packet and send it to back to the requestor.
  4. If any other packet is received, we store the source_mac and store_ip in mac_to_ip table and try sending the waiting packets based on the

information we learnt now. Now, if the destination IP is in the mac_to_ip table, we simply construct a flow rule, which instructs the router to send the packet out of the corresponding port (based on the ROUTING Table i.e. we are sending to the Next Hop). If the destination IP is not in the table, means we have no information about the destination MAC. So, we have to send an (rather flood) ARP request in order to get the next hop MAC. We enqueue the packet to the lost buffers list while waiting for the ARP reply. Once, it is received, we send the packet to the corresponding port .

The above algorithm is implemented in the code l3.py. All the layer 3 routers in our topology will be registered to this controller.

- Configuration file
  This file consists the {dpid,{destination, next_hop, output_port}} in .json format. The file is parsed and put in the routing table once the l3 controller starts.

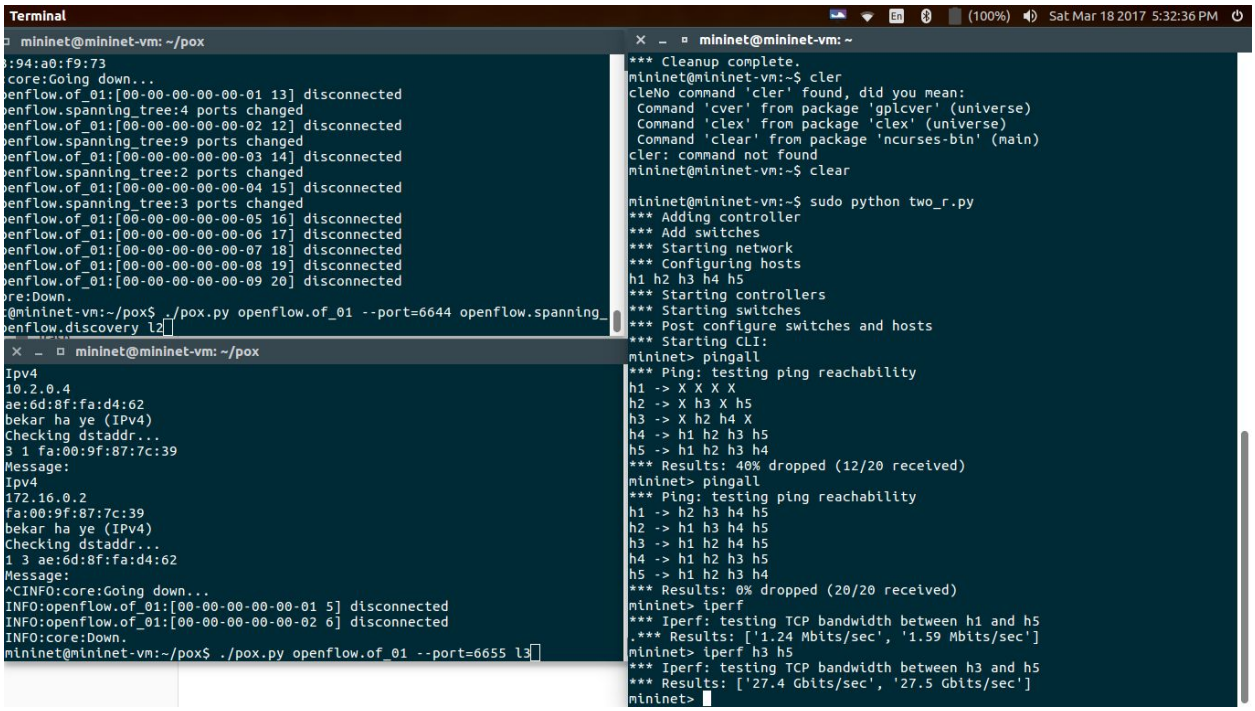
- Custom Topology
  The custom topology is created which specifies the default gateways of the host as well as declares the custom controllers. Moreover, the custom topo allocates the port on which the switches and routers should listen so as to register them to different controllers.

- Testing the controller
  In order to run the topology and hook them up to the custom controllers, we have follow the following steps:-
  1. Open 3 terminals and ssh it to mininet. Place the controller files (i.e. l2.py and l3.py) in pox/ext directory. Place the custom topology in the mininet home directory.
  2. In terminal 1:
     cd pox
     sudo su
     ./pox.py openflow.of_o1 --port=6644 openflow.spanning_tree openflow.discovery l2
     We are using port=6644 because we have specified in our topo that switches listen on TCP port 6644.
  3. In terminal 2:
     cd pox

sudo su
./pox.py openflow.of_01 --port=6655 l3
In our topo, routers are listening on 6655.

4. In terminal 3:
   sudo python topo.py
   Pingall
5. We are now ready to test our controller with the custom topo we made. Any command to test say iperf/pingall/running a ftp server, will suffice.

OBSERVATIONS AND DISCUSSIONS



The first image is the screenshot of the iperf and pingall statements on the custom topo (in the second image) using the l2.py and l3.py as its Level 2 and Level 3 control protocol respectively.

Following are my observations :-

1. The TCP bandwidth decreases substantially when the hosts are not in the same subnet. As we can see from the 1st pic, the TCP bandwidth from h3->h5 is 27.5 Gbps and from h1->h5 is 1.25 Mbps.
   The reason behind this is that the path from h1->h5 involves 2 routers, while the path from h3->h5, involves no router. As we know, the switches are much faster than routers because of the route table lookup cost involved in L3 Processing while this is not the case for L2 Processing.
2. The first time when I ran pingall, approximately 40% packets got dropped. But, when I did it next time, it was not the case. There was no loss of packets observed in subsequent pingall calls.
   The reason behind this is that for the first pingall, there is a delay to set up the paths and populating the tables as a result of which packets get dropped due to buffer overflow.