Assignment 2

(Fake) Router Exercise with POX

Author: BA
Edition: 2 (20221105)

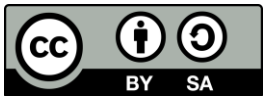


Table of Contents

Table of Contents 1

General Information 2

Changelog 2

Description 2

Part 1 – Creating topology and setting up hosts 3

Part 2 – Setup the controller 4

Some variables 5

Handling incoming packets 5

Limitations 6

Part 3 – Testing your router 6

References 7

Assignment 2

(Fake) Router Exercise with POX

General Information

- **Assignment Type** **:** Individual
- **Deadline** **:** November 23rd 2022, 22:00 WIB (SCeLE Time)
- **Output** **:** Report
- **File Name Format** **:** Assignment2_[NPM] (example: Assignment2_1006181721)

Changelog

Edition 1 (20211005-0000): First Release

Edition 2 (20221105-0000): change some information details

Description

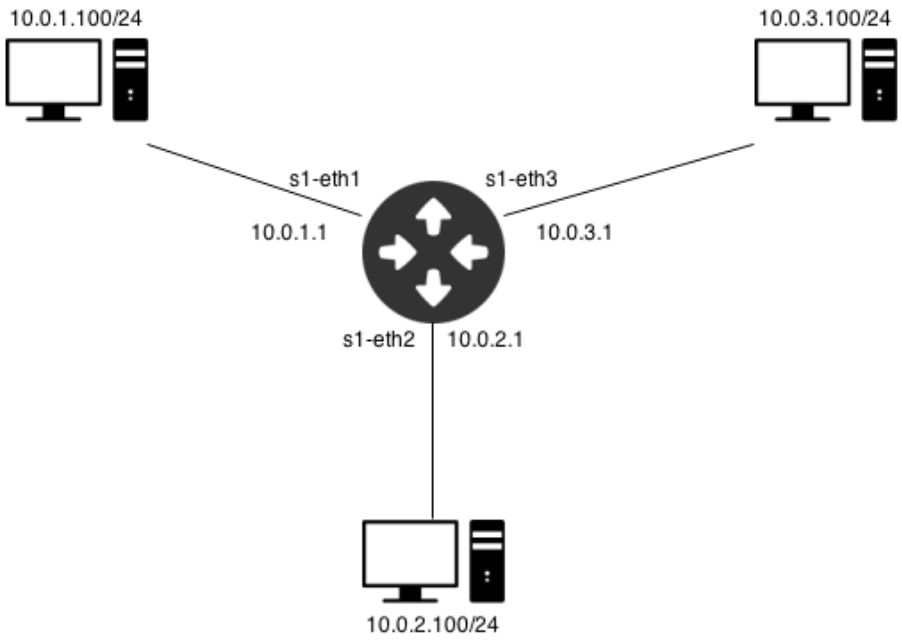
In this assignment, we will try to implement some basic functionalities of a router into a virtual switch. This is called a "Fake" router because essentially it only implements forwarding in Layer 3 without routing function (i.e. static and dynamic routing). Furthermore, it does not decrement the IP TTL and recompute the checksum at each hop (so traceroute won't work).

To simplify the assignment, we will use a simple topology with one router and some hosts, where each interface of the router is connected to a host and corresponds to different subnet. Therefore, there are only two cases for testing the connectivity with ping, (1) a host send an ICMP Echo Request (ping) to any router interface; (2) a host send a ping to any other host in different subnet. In the first case, the router will reply with an ICMP Echo Reply message to the sender, while in the second case the router will forward the packet to the correct interface/port number to reach the destination host (which is one hop away from the router).

There are several parts that we have to accomplish to make this assignment up and running properly, i.e. part 1 (create topology) and 2 (program the controller). Note that, the `router_exercise.py` is adapted from `pox/forwarding/l3_learning.py` which you can find under the home directory. This assignment is also based on the one given in https://github.com/mininet/openflow-tutorial/wiki/Router-Exercise with some modifications.

Part 1 – Creating topology and setting up hosts

The topology for this assignment looks like the following figure:



The topology is basically based on single topology, but it is slightly different in the gateway IP address of each host which corresponds to router interface a host is connected to. To setup the host, we need to configure the IP address, subnet mask and default gateway. It is important to note that although each interface of an actual router should be configured with an IP address corresponding to a subnet, we cannot configure it directly in Mininet, so we will do it "indirectly" from POX controller. Here is the snippet and hint to create the topology in Mininet (for this example, let's name it `mytopo.py`):

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."
    def __init__( self, k=2 ):
        "Create custom topo."
        # Initialize topology
        Topo.__init__(self)

        # Add hosts and switches
        self.k = k

        switch = self.addSwitch('s1')
        for h in range(k):
            host = self.addHost('h%s' % (h+1), ip="x.x.%s.x/24" % (h+1),
                        defaultRoute = "via x.x.%s.x" % (h+1))
            self.addLink(host, switch)

topos = { 'singlerouter':  MyTopo }
```

NOTE: you need to change the "x" in the **ip** and **defaultRoute** arguments according to the example topology in the given picture.

According to the snippet, the only difference with the original single (switch) topology is, the host is configured programmatically here. As stated earlier, the host configuration consists of host's IP address with subnet prefix (indicating its netmask) and default route or gateway, along with the host name (e.g. h1, h2, etc). Also, 'singlerouter' written in the last line is defined as the topology name and it will be used as the value for `--topo` option when Mininet command is run. Again, you can modify it as you please. Now, let us do the following:

1. Open a terminal and bring the Openflow VM up (either by using VirtualBox or vagrant).
2. Create `mytopo.py` and store it at your VM home directory (e.g. `/home/mininet` if you are using mininet VM or `/home/vagrant` if you are using vagrant).
3. Run mininet by using the following command to use our custom topology:
   `$ sudo mn --custom mytopo.py --topo singlerouter,3 --mac --controller remote --switch ovsk`
4. Open another terminal and run POX controller with default configuration or let say with the default `of_tutorial` just to try the connectivity of this topology:
   `$ ./pox/pox.py log.level --DEBUG` or
   `$ ./pox/pox.py log.level --DEBUG misc.of_tutorial`
   Make sure the vSwitch is connected to the controller (from the POX DEBUG log)
5. Verify that the topology shown in the topology figure earlier has been created by using `dump` and `net` command in the mininet CLI. `mininet> dump` and `mininet> net`
6. Try ping from h1 to other host or its default gateway from the mininet CLI:
   `mininet> h1 ping -c2 h2` or
   `mininet> h1 ping -c2 10.0.1.1`
   what is the `ping` result?

Part 2 – Setup the controller

Here, you need to create a code that enables forwarding in layer 3 (layer network) and store it under the `~/pox/pox/samples` directory. **HINT:** you can get some inspiration and simplify the code from https://github.com/noxrepo/pox/blob/gar-experimental/pox/forwarding/l3_learning.py or `~/pox/forwarding/l3_learning.py` (under the home directory). In this file, main functionalities of the (fake) router reside within the `l3_switch` class.

Some variables

Before that, let us first explain some important variables:

- `fakeways`: a Python set that consists of IP addresses in each router's interface which correspond to the default gateway of each host. This value is set as one of the options upon starting up the POX controller.
- `dpid`: stands for datapath ID. It is actually a unique ID for each switch controlled by POX controller. (see: https://noxrepo.github.io/pox-doc/html/#dpids-in-pox).
- `Entry`: a Python class that consist of switch's `port` number and its corresponding MAC `address`. It will be used in the ARP table that corresponds to a gateway's IP address.
- `arpTable`: a 2-D Python dictionary that consists of the following Key-Value dictionary:
  - 1st dictionary's Key: IP address in an ARP table, Value: `Entry (mac and port)`
  - 2nd dictionary's Key: `dpid`, Value: 1st dictionary

Handling incoming packets

So far so good. Now, we need to know that the main function that handles all incoming packets to the switch is called `_handle_PacketIn (self, event)`. The incoming packet can be parsed from the `event` argument of the `_handle_PacketIn` function and it contains all headers from the outermost header (e.g. Ethernet header), followed by the inner header directly next to it (e.g. ARP or IPv4) and so on (e.g. ICMP, TCP, etc). One way to extract the packet from the outermost header towards the innermost header in POX is by using the following methods:

`packet = event.parsed` packet contains the ethernet frame

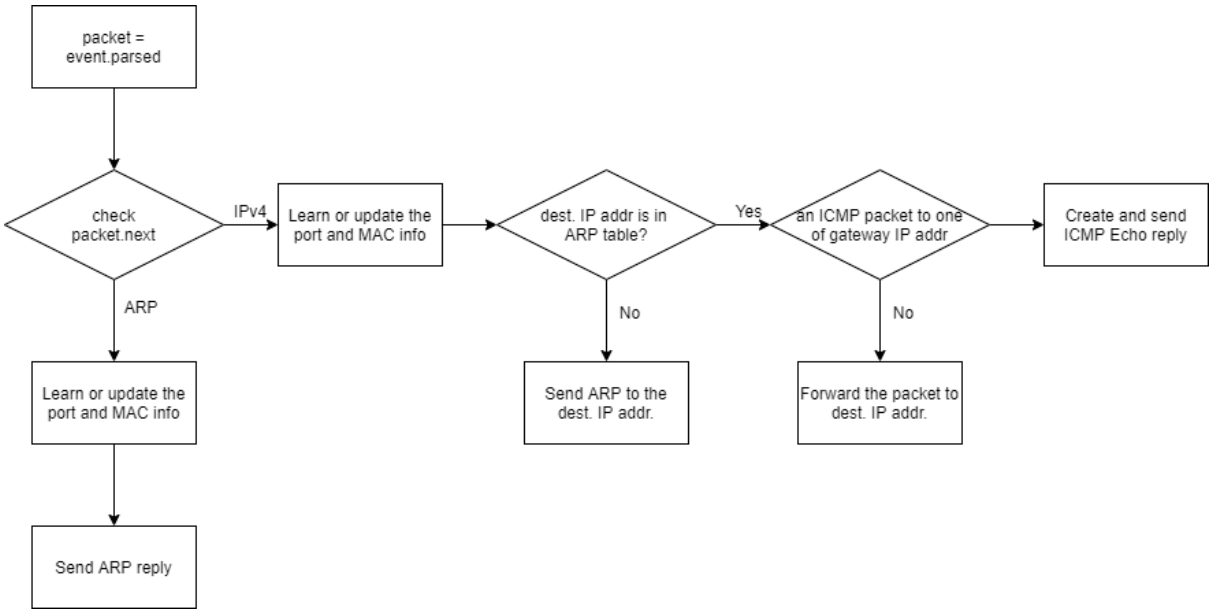
`packet.next` strips the ethernet header and contains the IPv4 or ARP packet

`packet.next.next` strips the header after ethernet and contains the underlying packet, e.g. ICMP

… and so on

Another way to do so is by using `packet.payload.payload` … method.

The figure below illustrates the high level process handling the incoming packet.



If you read the `l3_learning.py`, the **check packet.next** conditional statement in the figure above is done by two conditional if statements, namely: `isinstance(packet.next, ipv4)` and `isinstance(packet.next, arp)`.

Limitations

Some limitations to the implemented code here are as follows:

1. Routing table is not implemented (even the static one)
2. The fake router floods the ARP request message to all router interfaces except the sender when it doesn't know the MAC address of the packet's destination IP address. In the real router, the ARP request is only sent to an interface that corresponds to a subnet whom the destination IP address belongs to. While it is not possible here because no routing table is maintained.
3. The fake router also floods the ARP reply, upon receiving an ARP reply which corresponds to an ARP request message that was originating from itself.
4. Unable to send ICMP unreachable host or network in case the destination address of an incoming ICMP echo request does not exists (although the subnet is there) or the subnet of the destination address of an incoming ICMP echo request is unreachable by this router.

Part 3 – Testing your router

1. Stop the POX controller that you run previously

2. Store the file from Part 2 under the `~/pox/pox/samples` directory, with any name, e.g. `mycode.py`.
3. Then, we can run the POX controller with the following command (assuming the `mycode.py` is stored in the `~/pox/pox/samples` directory):

   ```
   $ ./pox/pox.py log.level --DEBUG samples.mycode --fakeways=10.0.1.1,10.0.2.1,10.0.3.1
   ```

   Note that in the above example, the fakeways consists of IP address of router's interfaces separated by comma (the code uses comma to split the IP address values).
4. Launch `xterm s1 h1` within the mininet CLI
5. In `s1 xterm`, run `tcpdump -XX -n -i s1-eth1`
6. In h1 xterm, run ping command to any of gateways IP address, e.g. `ping -c2 10.0.3.1`
7. In h1 xterm, run another ping to any host IP address, e.g. `ping -c2 10.0.2.100`
8. Check the ping results from h1 xterm as well as in the DEBUG log of the controller.

Your Tasks

1. Write a code to create a topology and setting up hosts (e.g. mytopo.py).
2. Write a code to enable layer 3 forwarding in POX controller (e.g. mycode.py or l3_forwarding.py or something else).
3. Write some explanations about the two codes in a single report.
4. Perform some experiments by following the instructions in Part 1 and Part 3, then write the results in the report along with the screen shots.

Submissions

Submit a report (`Assignment2_[NPM].pdf`) and the codes in single **zip file**.

References

https://github.com/mininet/openflow-tutorial/wiki/Router-Exercise

https://github.com/noxrepo/pox/blob/gar-experimental/pox/forwarding/l3_learning.py

https://noxrepo.github.io/pox-doc/html/