Programming Assignment 2: Mininet and POX

In this assignment, you will learn how to create a virtual network on your local machine and install flow rules with OpenFlow.

Setup

1. Download the required components.
    a. Download Virtualbox:
    https://www.virtualbox.org/wiki/Downloads

    b. Download Mininet:
    https://github.com/mininet/mininet/wiki/Mininet-VM-Images

    c. Configure the Virtualbox so that you can access it through SSH:
    https://github.com/mininet/openflow-tutorial/wiki/Set-up-Virtual-Machine
    From this point on, you can minimize Virtualbox and interact with it purely through SSH. Don't forget `-x` option for graphical forwarding.

2. Learn how to run a basic topology.
    a. Run a controller:
    ```
    ./pox.py log.level --DEBUG misc.of_tutorial
    ```
    This loads the controller in ~/pox/pox/misc/of_tutorial.py. This version acts like a dumb switch and floods all packets.

    b. At the same time, set up a topology that connects to the controller:
    ```
    sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --controller remote
    ```
    This loads the topology ~/mininet/custom/ topo-2sw-2host.py. This version creates a simple topology with one switch connected to two hosts.

Resources

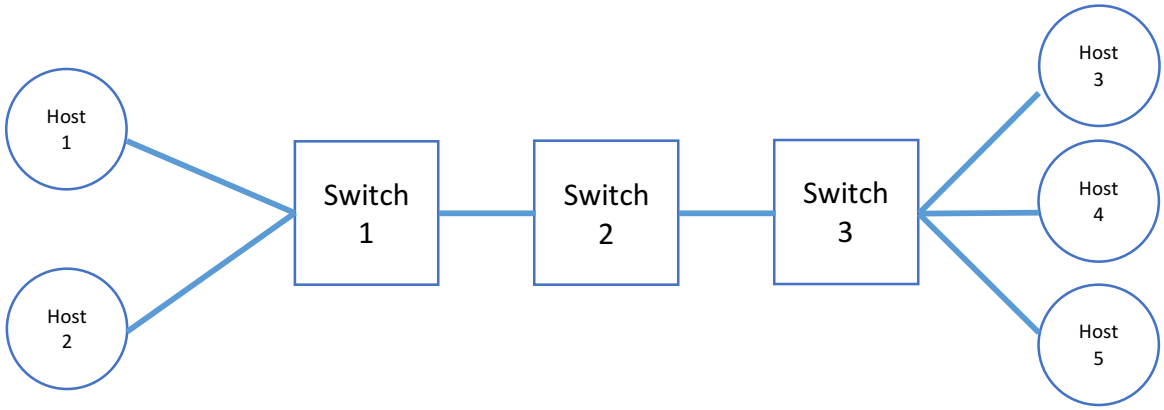
This is only a barebones guide to the setup. You may find the following resources useful (and in fact are encouraged to look throught them):
https://github.com/mininet/openflow-tutorial/wiki
https://openflow.stanford.edu/display/ONL/POX+Wiki

Some handy commands:

- Ping between hosts h1 and h2: `h1 ping h2`
- Any command you want to send to host h1: `h1 cmd`
- Open up a new terminal for host h1: `xterm h1`
- Analyze network traffic: `wireshark, tcpdump`
- Print the rules currently installed on switch s1: `ovs-ofctl dump-flows s1`

Instructions

First, set up up the following topology:



In Parts A and B, all traffic will be directed to the network controller, which will make decisions on behalf of the switches. In Parts C and D, the controller will install rules on the switches so they can forward traffic themselves.

Please submit your Python code for each step in an Appendix.

Part A: Hub controller

All traffic arriving on a switch will be forwarded to the controller. The controller then instructs the switch to forward the packet on all ports except the one it arrived on. This is the default behavior of the of_tutorial.py controller.

1. Have `h1 ping h2` and `h1 ping h5`. How long did it take to ping? What is the difference? Which of the hosts and switches observe traffic?
2. Run `iperf h1 h2` and `iperf h1 h5`. What is the throughput? What is the difference?
3. Run `pingall` to verify connectivity and dump the output.

Part B: MAC learning controller

All traffic arriving on a switch will be forwarded to the controller. Modify the default controller so that, for each switch, it learns the mapping between MAC addresses and ports. It then instructs the switch which port to forward the packet on.

1. Have `h1 ping h2` and `h1 ping h5`. How long did it take to ping? What is the difference? Which of the hosts and switches observe traffic? How does this compare to the hub controller?
2. Run `iperf h1 h2` and `iperf h1 h5`. What is the throughput? What is the difference? How does this compare to the hub controller?
3. Run `pingall` to verify connectivity and dump the output.

Part C: MAC learning switch

Now, we will try to make the switch a bit smarter. Modify the controller from Part B so that when it learns a mapping, it installs a flow rule on the switch to handle future packets. Thus, the only packets that should arrive at the controller are those that the switch doesn't have a flow entry for.

1. Have `h1 ping h2` and `h1 ping h5`. How long did it take to ping? What is the difference? Which of the hosts and switches observe traffic? How does this compare to Part A, and why?
2. Run `iperf h1 h2` and `iperf h1 h5`. What is the throughput? What is the difference? How does this compare to Part A?
3. Run `pingall` to verify connectivity and dump the output.
4. Dump the output of the flow rules using `ovs-ofctl dump-flows`. How many rules are there, and why?

Bonus: if you used the `of.ofp_match.from_packet` function to create the flow rule, that might explain why there are so many rules. Can you think of a way to reduce the number of rules, and implement this in the controller?

Part D: Simplified IP router

Finally, we will do a bit of layer-3 routing. This is a simplified version of an IP router where we ignore several aspects such as TTL, checksum, and ARP. Modify the topology so the hosts H1-5 have IP addresses 10.0.0.1, 10.0.0.2, 10.0.1.1, 10.0.1.2, 10.0.1.3 respectively (but are still on the same subnet, e.g. /16). Install IP-matching rules on switch 2 (hint: use `ovs-ofctl` and match on /24). Let switches 1 and 3 stay as MAC learning switches.

1. Have h1 ping h2 and h1 ping h5. How long did it take to ping? What is the difference? Which of the hosts and switches observe traffic? How does this compare to the previous controllers?
2. Run iperf h1 h2 and iperf h1 h5. What is the throughput? What is the difference? How does this compare to the previous controllers?
3. Run pingall to verify connectivity and dump the output.
4. Dump the output of the flow rules using ovs-ofctl dump-flows. How many rules
1. are there, and why?
5. How does this network compare to your previous controllers? Which is better, and why?

Based on Jennifer Rexford's COS561 programming assignment.