Introduction to Mininet and Software Defined Networking

In this exercise you will be introduced to Software Defined Networking (SDN).

You will learn:

- how to use mininet, an SDN-enabled network emulator.
- how to use different cmd tools to inspect and debug a network.
- build a simple SDN controller that implements a firewall.

You can find the code repository for this exercise here

Background

Software Defined Networking and OpenFlow

Software-Defined Networking (SDN) is a recently proposed networking paradigm in which the data and control planes are decoupled from one another. One can think of the control plane as being the network's "brain", i.e., it is responsible for making all decisions, for example, how to forward data, while the data plane is what actually moves the data. In traditional networks, both the control- and data planes are tightly integrated and implemented in the forwarding devices that comprise a network. The SDN control plane is implemented by the "controller" and the data plane by "switches". The controller acts as the "brain" of the network, and sends commands ("rules") to the switches on how to handle traffic. OpenFlow has emerged as the de facto SDN standard and specifies how the controller and the switches communicate as well as the rules controllers install on switches.

Mininet

Mininet is a software stack that creates a virtual network on your computer/laptop. It accomplishes this task by creating host namespaces (h1, h2, etc) and connecting them through virtual interfaces. So when we run ping between the linux namespaces h1 and h2, the ping will run from h1s namespace through a virtual interface pair created for h1 and h2, before it reaches h2. If h1 and h2 are connected through a switch as shown in the python code in the Mininet walkthrough, the ping will transit multiple virtual interface pairs. The switches that we will be using are running OpenVSwitch (OVS), a software-defined networking stack. Mininet will connect additional virtual interfaces between each virtual port on the switch with each connected host. The host name space allows each host to see the same file system, but operates as its own process that will run separately from each of the other host processes. The OVS version running on the Ubuntu image supports OpenFlow.

POX

Pox is a research/academic implementation of an OpenFlow controller. It provides python hooks to program mininet-based software emulated networks.

Assignment I

Virtualbox and Vagrant Installation

Mininet is a tool for building emulated network topologies on a Linux host for experimentation with SDN. To ensure everyone is using the same enviornment and prevent you from messing up the networking on your computer, we'll run mininet inside of a virtual machine.

To manage the virtual machine, we'll be using a tool called Vagrant. Vagrant supports multiple "virtualization providers", but for compatibility with the Ubuntu image we'll use to run mininet, we suggest using VirtualBox. VirtualBox an open source and freely available virtualization system.

You will need to install both Vagrant and VirtualBox for this assignment. See their respective documentation for your platform for installation instructions:

- Vagrant: installation docs, downloads
- VirtualBox: installation docs, downloads

Depending on your operating system, you may be able to install the above through your package manager.

Mininet VM Installation with Vagrant

Please follow the instructions here to set up the Vagrant VM. Once you connect to the VM via ssh, you can proceed to the next part.

(Alternative) Pre-baked VM Image

In case you have trouble using Vagrant, we also provide the same VM image for Virtualbox. You can download it from here and then import it directly into Virtualbox. You can then connect to the default user with the following credentials (user:`vagrant`, password: `vagrant`).

Using Mininet

Using mininet is described here. You can run it by typing: `sudo mn`.

There is also a very helpful Mininet community wiki with lots of good up-to-date information about how to use mininet effectively.

Inside of the the mininet CLI, try running other commands like `help`, `net`, `nodes`, `links` and `dump`. Mininet starts with a default network that you can poke at. Find the mac address of the hosts, the ethernet ports connected, and the hostnames in the system.

Programming Mininet Topologies

Mininet is also programmable using the python programming language. We have provided some sample topologies in `./topos/`. To use them inside the VM, ssh into it and clone the repo:

```
vagrant ssh -- -A
git clone <path_to_repo>
```

In the `topo` folder there are two python files. You can run these files with `sudo python3 exe1/topos/topo1.py`. It will drop you into the CLI with the network topology defined in the python script.

Tasks:

- Modify `topo1.py` to create four hosts (h1-h4) connected to the same switch.
- Run `h1 ping h4`, `iperf h1 h4`, `dump`, and `pingall`.
- Provide screenshots for the execution in your report.

Using an OpenFlow Controller

Unlike the previous exercise, in which Mininet used its default controller, in this exercise you will use a remote controller. The controller you will use can be found, inside the VM, at `mnvm:~/pox/pox/forwarding/l2_learning.py`.

From the `~/pox` directory you can run the controller with `./pox.py forwarding.l2_learning`.

After the controller starts, start mininet by running the second topology `sudo python3 exe1/topos/topo2.py`. This topology includes 4 hosts across two different subnets (h1 and h4 in one and h2 and h3 in another). Also, not that in this setting mininet is configured with static ARP. You should not modify this file.

You can monitor the traffic in any of the 4 interfaces (s1-eth1, s1-eth2, s1-eth3, s1-eth4) using `tcpdump`. You can observe the OpenFlow messages sent between the controller and the switch by running `tcpdump` on the `lo` interface, i.e., `tcpdump -i lo`.

Tasks:

- Run `h1 ping h4`.
- Run `dpctl dump-flows` and inspect the rules the controller installed. Provide screenshots in your report.
- Briefly explain how `l2_learning.py` works. When you run `h1 ping h4` is there any chance you receive an `ICMP` packet in h2? Explain your answer.

Impelementing a simple firewall

For this assignment you will create a simple firewall using OpenFlow-enabled switches. The term “firewall” is derived from building construction: a firewall

is a wall you place in buildings to stop a fire from spreading. In the case of networking, it is the act of providing security by not letting specified traffic pass through the firewall. This feature is good for minimizing attack vectors and limiting the network "surface" exposed to attackers.

In this part, you will use the same topology as in the previous exercise `topo2.py`. For this exercise, though, you need to comment out the static ARP configuration as hinted in the comments of the python script.

For this part, we also provide you with a skeleton POX controller: `./pox/exe1-controller.py`. This file will be where you will make your modifications to create the firewall. After doing your modifications you need to place this file in `~/pox/pox/misc`. Alternative you can create a symlink. You can run this controller with `~/pox/pox.py misc.exe1-controller`. Once the controller is running, you can run mininet, as before, with `sudo python3 exe1/topos/topo2.py`.

The rules s1 will need to implement are as follows:

src ip	dst ip	protocol	action
any ipv4	any ipv4	icmp	accept
any	any	arp	accept
any ipv4	any ipv4	-	drop

Your Firewall should allow all ARP and ICMP traffic to pass. However, any other type of traffic should be dropped. It is acceptable to flood the allowable traffic out all ports. Be careful! Flow tables match the rule with highest priority first, where priority is established based on the order rules are placed in the table. When you create a rule in the POX controller, you need to also have POX "install" the rule in the switch. This makes it so the switch "remembers" what to do for a few seconds. Do not handle each packet individually inside of the controller! Hint: To do this, look up `ofp_flow_mod`. The OpenFlow tutorial (specifically "Sending OpenFlow messages with POX") and the POX Wiki are both useful resources for understanding how to use POX.

Tasks:

- Run `pingall`. `h1` and `h4` should be able to ping each other and `h2` and `h3` as well. Provide screenshots in your report.
- Run `dpctl dump-flows` to see the ruls you inserted. Provide the equivalent screenshot.

Deliverables

You should create `.zip` archive that includes a report in pdf with the requested screenshots from the execution and the explanation for the second exercise, the

`topo1.py` file from the first exercise, and the `exe1-controller.py` file from the third exercise.