Advanced Computer Networks
Assignment 3 – Software Defined Networking
Assignment given on: 21st Oct 2023
Due Date: 5th Nov 2023

(1% Penalty per 24-hour period)

Project Overview

In this project you will learn about Software Defined Networking (SDN). Using Virtualbox, Mininet, and Pox as the implementers of the OpenFlow protocol, you will build simple networks using SDN primitives.

- First you will learn to use mininet, a SDN-enabled network emulator.
- For the second portion you will be using POX to implement a simple L2 static firewall via OpenFlow rules determined by your SDN controller.

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 OpenVFlow controller. It provides python hooks to program mininet-based software emulated networks.

Assignment

Part 1: Mininet Primer

Learning Objectives (After this section, students will be able to…) :

- Use Mininet, a widely used self-contained network environment for research and experimentation, to build a test network topology.
- Use the basic debug tools built into Mininet to test connectivity between virtual hosts and get the state of the openflow switches in their topology.

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 environment 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 is 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

Mininet VM Installation With Vagrant

Please follow the instructions here to set up the Vagrant VM for this project. This VM is based on Ubuntu 20.04 and includes a default set of mininet binaries and example scripts. Once you are inside the VM after running `vagrant ssh`, proceed to the next part.

There are a well known set of issues in using virtualization. If you encounter issues, reach out online and we'll see how we can help. Some common issues and answers are:

- Is your computer really old? It might not be able to be virtualized – talk to the staff via Ed.
- Is your computer really new, and runs on a non-x86 architecture? Talk to the staff via Ed.
- Make sure that virtualization is enabled in your BIOS.

For more information about vagrant, you can checkout this supplemental doc

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 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.

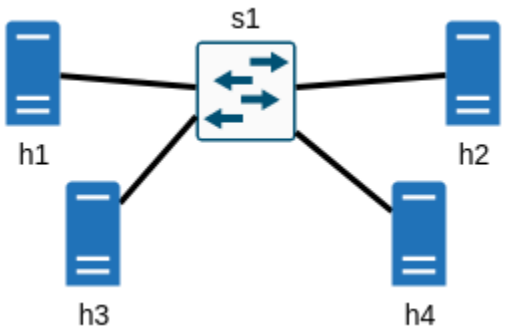
Programming Mininet Topologies

Mininet is also programmable using the python programming language. We have provided some sample topologies here, which should already be downloaded in the VM and unzipped into `~/project-1`.

In this directory, you'll find two different directories: topo and pox. Ignore the pox directory for now (it's used in part2). In the topo folder there are a variety of python files. These each define a topology for each of the following project portions. Run the project 1 file with `sudo python3 project-1/topos/part1.py`. It will drop you into the CLI with the network topology defined in the python script.

Deliverables:

Your task in part one is to modify part1.py to represent the following network topology:



After creating the above architecture, you submission should include the following items:

1. (10 Points) Your modified topos/part1.py file
2. (10 Points) Screenshots of the iperf h1 h4, dump, and pingall commands (from within the mininet cli in your running VM) in pdf format. These will help the TAs validate that your submission was working correctly on your machine if for some reason your submitted code does not work in the grading environment. Put your screenshots in the screenshots/part1 directory.

Part 2: SDN Controllers using POX

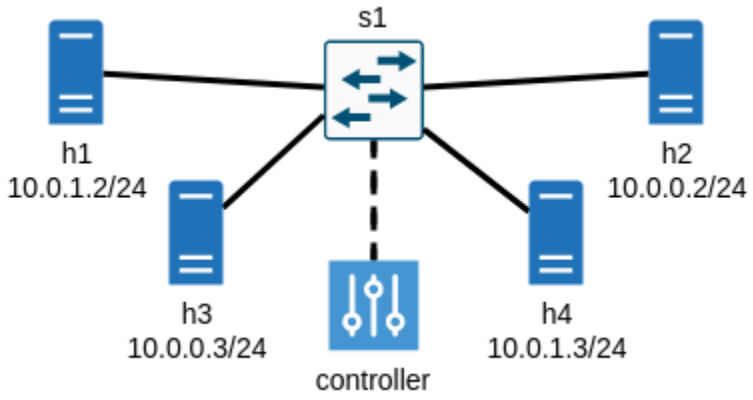
Learning Objectives (After this section, students will be able to…):

- Run their own POX openflow controller to add custom rules to switches in a mininet topology.
- Build an OpenFlow SDN controller program capable of connecting hosts together via a single programmable switch.
- Establish connectivity by flooding packets out switch ports.
- Create match –> action rules to block or allow particular types of traffic through a switch based on layer 2 and layer 3 header information.

In part 1, we experimented with Mininet using its internal controller. In this part, we will instead be using our own controller to send commands to the switches. We will be using the POX controller, which is written in Python.

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, we will provide you with the Mininet topology, part2.py, to set up your network which assumes a remote controller listening on the default IP address and port number 127.0.0.1:6633. You do not need to (and should not) modify this file. The topology that this script will setup is as follows. Note that h1 and h4 are on the same subnet and a different one from h2 and h3.



For part 2, we will also provide you with a skeleton POX controller: a1part2controller.py. This file will be where you will make your modifications to create the firewall. The bootstrap should have set up a link from the ~/pox/pox/misc directory to ~/project-1/pox/a1part2controller.py. If no link is present at ~/pox/pox/misc/a1part2controller.py, you can create a new one by running `ln -s ~/project-1/pox/* ~/pox/pox/misc/`. You can then launch the controller with the command `sudo ~/pox/pox.py misc.a1part2controller`. Once the controller is running, you can run mininet with `sudo python3 ~/project-1/topos/part2.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

Basically, your Firewall should allow all ARP and ICMP traffic to pass. However, any other type of traffic should be dropped. It as 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.

Deliverables:

1. (10 Points) A screenshot of the `pingall` command. Note that h1 and h4 should be able to ping each other (h2 and h3 as well), but not across subnets. Also, the `iperf h1 h4` command should fail (as you're blocking IP traffic). This is realized as the command hanging. Put your screenshot in the screenshots/part2 directory.
2. (10 Points) A screenshot of the output of the `dpctl dump-flows` command. This should contain all of the rules you've inserted into your switch(es). Put your screenshot in the screenshots/part2 directory.
3. (20 points) Your pox/a1part2controller.py file.

Submission

When you're ready to turn in your assignment, do the following:

1. Please read the questions carefully and answer all of them.
2. Update the README.txt file to contain the names and UW netid of the member(s) of your team.
3. Test well before submission. Follow some coding style uniformly. Provide proper comments in your code.
4. Double check you are uploading the latest version of your code and screenshots.
5. Archive your entire project-1 directory as a zip archive.
6. Submit the archive as <Your_Roll_Number>.zip to Moodle by **05-11-2023 (23:50 PM)**.