CS 538: Advanced Computer Networks Spring 2016

OpenFlow functionality with Mininet

Assignment 2 Due: 2:00 PM CT, Thursday March 17, 2016

1 Introduction

The main goal of the assignment is to get hands-on experience with OpenFlow and Mininet.

Data. The package is available for download at compass 2g. Inside, you will find a file: `firewall.py`.

Submission instructions. This assignment is due at 2pm CT, Thursday March 17, 2016. TA (Wenyu Ren; wren3@illinois.edu) will collect assignments on compass 2g. Acceptable formats are PDF (preferred), or plain text, with attached figures and code files. (Word .doc is not preferred. Please export to PDF.)

You will create a zip file with your NetID as the name, including all the files required and then upload it to compass 2g.

Collaboration policy. You're encouraged to discuss the assignment, solution strategies, and coding strategies with your classmates. However, your solution and submission must be coded and written by yourself. Please see the policy on cheating stated in the course syllabus.

2 OpenFlow functionality with Mininet

Experimentation is an important part of networking research. However, large-scale experiments can sometimes be hard to achieve, e.g., due to lack of machines. In this section, you will learn how to use Mininet[1], a relatively new experimental platform that can scale to hundreds or more emulated "nodes" running on a single machine. Mininet takes advantage of Linux support for *network namespaces*[2] to virtualize the network on a single machine, so that different processes on the same machine can see their own network environments (like network interfaces, ARP tables, routing tables, etc.), distinct from other processes. Combined with the Mininet software, this enables a single machine to emulate a network of switches and hosts. The emulated processes, however, do see the same real/physical file system.

Mininet is designed with OpenFlow[3] in mind. In this exercise, you will gain a basic understanding of OpenFlow and create a custom OpenFlow controller to control your switches. Quite simply, OpenFlow allows for "programmable" network devices, e.g., switches. With Mininet, each switch will connect to the controller specified when the switch is launched. When the switch receives an Ethernet frame, it consults its forwarding table for what to do with the frame. If it cannot determine what to do with the frame, the switch sends the frame (and some extra information such as the input switch port) to the controller, which will then instruct the switch on what to do with the frame. To avoid this extra work on every such frame, the controller can install a new rule/match in the switch's forwarding table, so that the switch can forward future similar frames without having to contact the controller.

[1] `http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet`
[2] `http://lwn.net/Articles/219794/`
[3] `http://www.openflow.org/`

2.1 Prepare the Mininet VM and OpenFlow Controller

1. Install VirtualBox from `https://www.virtualbox.org/wiki/Downloads`. VMware should also work; adjust your VM configurations accordingly. (It is possible to install Mininet directly on your Linux system, but for simplicity we'll use the virtual machine here.)
2. Download and unzip the VM with Mininet already installed from `https://github.com/mininet/mininet/wiki/Mininet-VM-Images`
3. In VirtualBox, create (not "Add") a new VirtualBox VM
   (a) For "Operating System" and "Version," select "Linux" and "Ubuntu," respectively.
   (b) At the "Virtual Hard Disk" page, select "Use existing hard disk," and select the `Mininet-VM.vmdk` file just unzipped.
   (c) For the newly created machine, go to "Settings" → "Network" and make "Adapter 1" a "NAT" (Network Address Translation). If your VM is allowed to obtain an IP address from your local network, you can alternatively use "Bridge Adapter." For more information on networking with VirtualBox, see `http://www.virtualbox.org/manual/ch06.html`
4. Start the VM
5. Log in with **mininet** for both username and password
6. Make sure `eth0` is up:
   (a) run the command:
   ```
   ifconfig eth0
   ```
   (b) check the `inet addr` field. If it does not have an IP address, then run the command:
   ```
   sudo dhclient eth0
   ```
   and repeat step (a).
7. Install POX, a platform that allows you to write your own OpenFlow controller using Python:
   ```
   git clone http://github.com/noxrepo/pox
   ```
   Newer version of the image has pox with it. So this step can be skipped. For more information on POX, see `https://openflow.stanford.edu/display/ONL/POX+Wiki`
8. Install a GUI in the VM:
   (a) Install the GUI
   ```
   sudo apt-get update
   sudo apt-get install openbox xinit -y
   ```
   (b) Start it
   ```
   startx
   ```
   (c) To create a new terminal, right-click on the desktop and select "Terminal emulator"

   Alternately you may use SSH to log in to the VM remotely, with GUI (X11) forwarding. With SSH, you will need to enable X-forwarding (e.g., `ssh -X` on *NIX hosts) when you ssh into the VM. NOTE: this requires you have an X server running on the host. See a description of how to do this on various platforms at `http://www.openflow.org/wk/index.php/OpenFlow_Tutorial#Download_Files`. Alternative for some versions of Mac OS X: install the Developer Tools (a free download from the App Store) and open `/Applications/Utilities/X11`.

2.2 Create a hub in Mininet using POX

In this exercise you will create a Mininet network with 3 hosts connecting via a switch. Using POX, you will program the switch to behave like a hub, which simply forwards incoming packets to every port except the one on which it entered.

First, you can familiarize yourself with Mininet by following `http://mininet.org/walkthrough/`. To start Mininet with the topology we want:

- First clean up the network:
  `sudo mn -c`
- Then create a network with the topology we want:
  `sudo mn --topo single,3 --mac --switch ovsk --controller remote`

This will create a network with the following topology:

```
host h1 ------switch s1 ---- controller c1
host h2 -------/ /
host h3 --------/
```

After you create this network, you will be entering the Mininet console. You can type `help` in the console to see a list of commands provided by Mininet. We will later use some of these commands.

Now let's run POX controller. Create another terminal (right-click on the desktop and select "Terminal emulator"). Go to the directory you installed POX in this new terminal, and then start POX with basic hub function:

```
pox/pox.py log.level --DEBUG forwarding.hub
```

The argument `log.level --DEBUG` enables verbose logging and `forwarding.hub` asks POX to start the hub component. It takes up to 15 seconds for switches to connect to the controller. When a OpenFlow switch has connected, POX will print something like:

```
    INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
```

To verify the hub behavior, we use `tcpdump`, a common packet analyzer that intercepts and prints packet information. To do this, we first create an xterm (terminal emulator in X Window System) for each host in Mininet and view packets in each. To start an xterm for each host, type the following command in the Mininet console:

```
xterm h1 h2 h3
```

You may want to arrange xterms properly so that you can see them on the screen at once. You may need to reduce the terminal height to fit a laptop screen. In the xterms for h1 and h2, run `tcpdump` to capture and print all the packets:

```
tcpdump -XX -n -i h1-eth0
```

and

```
tcpdump -XX -n -i h2-eth0
```

In the xterm for h3, send a ping to h1:

```
ping -c1 10.0.0.1
```

The ping packets are going to the controller, which floods the packet out all interfaces but the received one. Because of this hub behavior, you should see identical ARP and ICMP packets in both xterms running tcpdump.

Question #1: What will happen if you ping a non-existent host that doesn't reply ICMP requests? For example, do the following command in the xterm for h3:

```
ping -c1 10.0.0.9
```

Submit and explain the results.

Now let's take a look at the hub code at `pox/pox/forwarding/hub.py`. Make sure to get familiar with the code because many POX API functions used here will help you answer the later questions. We describe several important API functions here, and you can find more information about POX APIs at `https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-POXAPIs`.

- `connection.send()` function sends an OpenFlow message to a switch.
  When the connection between a switch and the controller established, the code will invoke `_handle_ConnectionUp()` function that implements the hub logic.
- `ofp_flow_mod` OpenFlow message
  This tells a switch to install a flow entry, which matches some fields of incoming packet headers and executes some actions on matching packets. Important fields include:
  - `actions:` A list of actions that apply to matching packets (e.g., `ofp_action_output` described below).
  - `match:` An `ofp_match` object (described below).
  - `priority:` When a packet matches on more than one non-exact flow entry, only the highest priority entry will be used. Here, higher values are higher priority.
- `ofp_action_output` class
  This is an action for use with `of.ofp_flow_mod`. You can use it to assign a switch port that you want to send the packet out of. It can also take "special" port numbers, e.g., we use `OFPP_FLOOD` to send the packet out all ports but the received one.
- `ofp_match` class (not used in the hub code but is useful in the assignment) This is an object that specifies packet header fields and input port to match on. All fields here are optional, i.e., if you do not specify a field, it becomes a "wildcard" field and will match on anything. Some important objects in this class:
  - `dl_src:` The data link layer (MAC) source address
  - `dl_dst:` The data link layer (MAC) destination address
  - `in_port:` The packet input switch port

  Example to match packets with source MAC address 00:00:00:00:00:01 in a OpenFlow message `msg`:
  `msg.match.dl_src = EthAddr("00:00:00:00:00:01")`

2.3 Create a firewall

A firewall is used as a barrier to protect networked computers by blocking the malicious network traffic generated by viruses and worms. In this assignment, you are asked to implement a data link layer firewall to block certain traffic.

To start this, you will find a skeleton class file at `firewall.py`. This skeleton class is currently not blocking any traffic and you will need to modify this skeleton code to add your own logic later. To test the firewall, put the `firewall.py` in the `pox/pox/misc` directory and run the POX controller:

```
./pox.py log.level --DEBUG forwarding.hub misc.firewall
```

After the connection between the controller and the switch is established, we can verify the connectivity between all pairs of hosts by typing `pingall` in the Mininet console. Note that when ping cannot get through a pair of hosts, you need to wait for the timeout, which takes about 10 seconds.

Question #2: Modify the firewall (`firewall.py`) to block traffic with source MAC address 00:00:00:00:00:02 and destination MAC address 00:00:00:00:00:03. To show the result, you can use the command `pingall` and copy the output to your report. (Hint 1: this only takes a few lines of code. Hint 2: if you did not specify any action in a OpenFlow message, then matching packets will be dropped.)

What to submit: The assignment will be graded out of 100 pts. your package should include two files:

1. Your completed `firewall.py` (30pts)
2. A report including results and your interpretation/explanation for both questions (70pts).
   - Question 1 (35pts)
   - Question 2 (35pts)

Note:

1. To get your files off the VM, you can `scp` or `ftp` them to some other machine. Or you can install the GUI (instructions in PDF) and then ”sudo apt-get install firefox” and then launch the GUI and use firefox to upload/email the files off the machine.