Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Spring Term 2017

ADVANCED COMPUTER NETWORKS
Assignment 7: Introduction to OpenFlow

Assigned on: **06 April 2017**
Due by: **02 May 2017, 23:59**

1 Introduction

The goal of this assignment is to give an introduction to OpenFlow [2, 4]. OpenFlow proposes a way for researchers to run experimental protocols in the networks they use every day. It is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. OpenFlow is a pragmatic compromise: on one hand, it allows researchers to run experiments on heterogeneous switches in a uniform way at line-rate and with high port-density; while on the other hand, vendors do not need to expose the internal workings of their switches. In addition to allowing researchers to evaluate their ideas in real-world traffic settings, OpenFlow could serve as a useful campus component in proposed large-scale testbeds like GENI.

2 More on OpenFlow

OpenFlow exploits the fact that most modern Ethernet switches and routers contain flow-tables (typically built from TCAMs) that run at line-rate to implement firewalls, NAT, QoS, and to collect statistics. An OpenFlow Switch consists of at least three parts:

- a **flow table**, which keeps an entry for every flow and tells each switch how to process the flow
- a **secure channel** that connects the switch to a remote control process, namely the **controller** – that adds and removes flow entries from the flow table for different experiments –, allowing commands and packets to be sent between a controller and the switch by using
- a **protocol**, which provides an open and standard way for a controller to communicate with a switch

In the context of OpenFlow, a flow can be a TCP connection, or all packets from a particular MAC address or IP address, or all packets with the same VLAN tag, or all packets from the same switch port. Every flow entry in the flow table has 3 basic actions associated with it:

- Forward the flow's packets to a given port or ports, which means packets are to be routed through the network

- Encapsulate and forward the flow's packets to a controller, which either processes them or decides if the flow needs to be added as a new entry to the flow table (i.e. if the packet is the first in a new flow)
- drop the flow's packets, which can be used for security issues, to curb denial of service attacks and so on

Read the Openflow whitepaper [2] and familiarize yourself with the basic OpenFlow elements, before setting up the environment.

3 Setting up OpenFlow

There is plenty documentation on how to install and setup OpenFlow and a step-by-step tutorial is provided in [5]. Key components of interest in the documentation are :

- **Mininet:** Mininet is a network simulator that creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native). We will use mininet to create different network topologies to study their behavior. Mininet has an ability to connect to any of the many available openflow controller implementations. Read the mininet walkthrough [3] to understand its capabilities and the command line interface.
- **POX:** POX is a python framework that is used to connect and interact with openflow capable switches. It contains many useful functions and primitives that can be used to implement a simple openflow controller. It already comes with some default implementation. See Section 5.1 of the openflow tutorial for setting up POX controller with mininet. For further details consult POX wiki [6].

3.1 Necessary downloads

First, you should download VirtualBox at **https://www.virtualbox.org/wiki/Downloads**. VMWare will work too, but VirtualBox is free, For those using Ubuntu, the Synaptic Package Manager already provides VirtualBox for installation. Second, you need to download the OpenFlow VM image from their website [5]. The link is provided on the page **Installing Required Software** of the tutorial. Import this VM image in VirtualBox and follow the instructions on the OpenFlow tutorial to setup the VM. Additionally, go to the **Settings** → **Network** → **Adapter 2**, select the **Enable adapter** box and set it to **host-only Adaptor**. This allows you to easily access your VM through your host machine. Note that you may have to create a host-adaptor first in VirtualBox by going in `File -> Preferences -> Network -> Host-only Networks -> Add Host-only network`.

The user on the VM image is **mininet** with password **mininet**.

3.2 Configuring VirtualBox for SSH

The VM image provided is only command line. You will need to SSH and use X Forwarding in order to load certain applications, such as **xterm** and **wireshark**. There are subtle differences in this step between Mac/Linux and Windows, so please follow the specific instructions for your machine, explained in detail in the tutorial.

Once your VM image has started and you have logged in, you should have access to the `/home` folder which contains several folders (i.e. `pox`, `mininet` and so on). To be able to connect through

SSH from your machine to the VM, first you need to check whether any of the network interfaces has IP addresses already assigned to them. You can easily check by issuing the command

```
$ ifconfig -a
```

If no IP addresses have been assigned, then run the command

```
$ sudo dhclient ethX
```

where you would replace `ethX` with any of the `eth0, eth1, ...` interface names.

Once an IP address was assigned, open a console on your machine and run

```
$ ssh -Y mininet@[IP here]
```

For specific instructions on Windows machines and more details on Mac/Linux setup, please read the **Access VM via SSH** tutorial section.

3.3 Developing a network topology

All development takes places through a SSH session to the VM. Thus, you will need an X Forwarding client running in order to have any graphical interaction through the session. On Mac/Linux systems this means checking that you have the `-Y` option when running SSH commands. On Windows machines, check that your X Forwarding application is running.

3.3.1 Wireshark analyzer

Wireshark is a great tool to help you analyze traffic flowing through nodes in the network. You would use Wireshark to analyze network behavior with the command

```
$ sudo wireshark &
```

This should open a graphical pop-up window (if there is an error saying a window cannot be created, your X Forwarding client may not be running). In the Wireshark filter box, enter `of` and then click **Apply**. Click **Capture**, then **Interfaces**, then **Start** on the loopback interface (lo). All packets flowing through the controller will show up including flow modifications. If you want to capture specific interfaces such as specifics hosts, simply click **Capture** and then **Interfaces** and then select the desired host. These details are also given in Section 4.5 of the tutorial.

Also, you may want to try out the command line tool `tcpdump` [7] which can capture packets in plain-text, and is easy to automate and script.

3.3.2 POX Controller

Start the POX framework with a simple controller

```
$ cd ~/pox
$ ./pox.py log.level --DEBUG misc.of_tutorial
```

3.3.3 Exploring the default topology

The default topology, discussed in the tutorial, includes 3 hosts and 1 switch. To create this network, issue the command

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

This essentially tells Mininet to start up a 3-host, single-switch topology, set the MAC address of each host equal to its IP address and point to a remote controller which defaults to the localhost. Once you start mininet, all packets flowing through the controller will show up in Wireshark/tcpdump. A few useful commands to use in the Mininet console are provided below

- `nodes` – see the list of available nodes
- `net` – see topology
- `xterm h1` – open xterm on host 1 (useful to run iperf, etc)
- `h1 ifconfig` – check h1's IP address
- `h2 ping -c 1 h3` – pinging combined with Wireshark/tcpdump is very useful in diagnosing controller behavior; if you view the Wireshark/tcpdump output, you should be able to view the traffic in the network as a result of the ping
- `dpctl dump-tables` – dump all tables from switches
- `dpctl dump-flows` – dump all flow rules from switches
- `s1 tcpdump -XX -n -i s1-eth1` – dump traffic from port 1 of switch 1.
- `exit` – end the mininet session

See mininet walkthrough to understand how to generate switches, hosts, and connections [3] to create a custom topology required later in the assignment.

4 Assignment

After getting familiar with the hub controller, the assignment consists of 3 parts.

You *should* put your custom POX controllers in the `ext` directory and can then launch them using names like `myfirstcontroller` (for a controller named myfirstcontroller in the ext directory).

4.1 Learning POX controller

In the first part of the assignment, you should implement a simple learning POX controller. This is a fairly simple exercise to familiarize yourself with POX framework. The learning algorithm consists of the following steps:

a) Look at the incoming port number and MAC source of the incoming packet and store it in a data structure, which means you associate the MAC address to the port number (you can use Python dictionary). If a packet arrives with destination to this MAC, then the port to forward the packet to is known.

b) If the data structure contains an entry with the MAC destination of the packet, then the port to forward to is known.

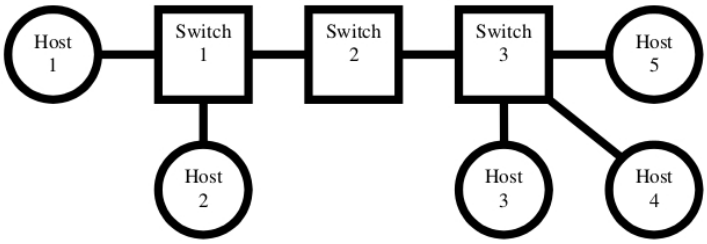

Figure 1: Topology for 2nd part of the assignment

c) Otherwise, the packet is flooded to all ports except for the port from which the packet came in.

4.2 Custom Network Topology I

In the second part of the assignment, you will use the topology shown in Figure 1. To create the new topology, you should follow the tutorial on developing a network topology (in section 7 of the OpenFlow tutorial) and create the topology in mininet. Verify that the topology is correct by analyzing the Wireshark/tcpdump output.

In steps 1 and 2, you are going to be direct all network traffic to the controller, that will then decide how each packet should be directed. In step 3, you will be installing microflow rules, so that the majority of network traffic will be handled by the switch as opposed to the controller.

4.2.1 Hub Controller

All traffic arriving at the switches will be directed to your POX controller. You should begin with a simple Ethernet hub where all network traffic is flooded on all ports except the port that it arrived in. Analyze what occurs on the network topology above with the Ethernet hub controller.

a) Run `H1 ping -c 100 H2`. How long does it take to ping? Which of the hosts and switches observe traffic as a result of the ping?

b) Run `H1 ping -c 100 H5`. How long does this take? Is there a difference?

c) Perform a `pingall` and copy the output, verifying that all hosts are pingable.

d) Run `iperf` instance between a pair of hosts and report the throughput.

4.2.2 Learning Controller

Once again, all traffic arriving at the switches will be directed to a POX controller. Your POX controller will perform MAC learning. You should reuse the code of the learning switch from the first part of the assignment. Analyze the network behavior with the MAC learning controller.

a) Run `H1 ping c 100 H2`. How much time does it take to ping the first packet? Is there any difference with the hub controller?

b) Run `H1 ping c 100 H5`. How long does it take? Compare with `H1 ping c 100 H4`.

c) Run `iperf` instance between a pair of hosts and report the throughput.

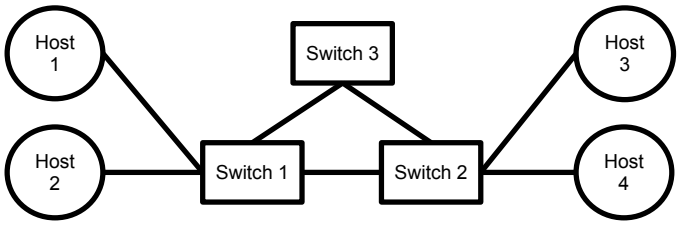

Figure 2: Topology for 3rd part of the assignment

4.2.3 MAC Learning Switch via MicroFlow Rules

Modify the MAC learning controller to install microflow rules in the OpenFlow switch reactively to incoming traffic. Thus, the controller will only receive packets for which there are no matching microflow rules in the switch.

a) Run `H1 ping c 100 H2`. How does it compare to the learning switch and hub from earlier?

b) Perform a pingall and dump the output. Verify that all hosts are pingable. Also dump the microflow rules that are installed in each switch.

c) Run `iperf` between a pair of hosts for which you have rules installed. Compare the obtained throughput with the throughput from the learning controller in the last section.

4.3 Custom Network Topology II: Controller with policies

In the final part of the assignment, consider the topology shown in Figure 2. The policy to implement requires that traffic to or from certain end hosts needs to go through a particular switch, as

a) H1 H4 SW3

b) H2 H4 SW3

which means that any packets traveling from H1 or H2 to H4 are required to traverse switch 3. For all other traffic (i.e. including the one from H4 to H1 or H2) the default paths will be the shortest paths. To check the controller enforces the policies, run *ping* between any 2 hosts and dump the traces in a log file. For this exercise you should not assume that your controller knows the topology from the beginning. You can however specify your policies with IPs as identifiers for the hosts.

5 Git Instructions

- You are going to use Git to handin your assignments for this course. Make sure you have it installed on your computer.
- Clone your private repository by typing:

  ```
  git clone git@gitlab.inf.ethz.ch:COURSE-ACN/NETHZ_USERNAME.git
  ```

  or

```
git clone https://gitlab.inf.ethz.ch/COURSE-ACN/NETHZ_USERNAME.git
```

You can access the repository online `https://gitlab.inf.ethz.ch/COURSE-ACN/NETHZ_USERNAME`

- For more information about how to use Git, look at [1]

6 Hand-In Instructions

- You should use following directory structure to submit your solution:

```
assignment7
    |-- hubController
    |   |-- t1_controller.py
    |   |-- your_plots_here
    |   |-- output.log
    |   `-- explanation.txt
    |-- learningController
    |   |-- t2_controller.py
    |   |-- t2_topology.py
    |   |-- your_plots_here
    |   |-- output.log
    |   `-- explanation.txt
    |-- microFlowController
    |   |-- t3_controller.py
    |   |-- t3_topology.py
    |   |-- your_plots_here
    |   |-- output.log
    |   `-- explanation.txt
    |-- policyController
    |   |-- t4_controller.py
    |   |-- t4_topology.py
    |   |-- your_plots_here
    |   |-- output.log
    |   `-- explanation.txt
    `-- README.txt
```

- Create a separate, and self contained solution for each part of the assignment, and put it in the separate folder. Here are some tips for formatting your solution:
  - If you are copying your code around, be generous and write proper comments, and cleanup the dead code.
  - OpenFlow code must work with/without mac and id when changed. The code should learn the topology.
  - If you have issues which might be related to the local system, then use `explanation.txt` note down expected performance and why.
  - Always install rules with timeout and run longer tests to see the difference in performance after the timeout.
  - Avoid strange hooks and hacks. Do what is asked cleanly and clearly.
  - plot ping traces. It will help you see the bumps and drops in the performance.

- Dump the output of your testing commands like `ping`, `pingall`, `iperf` in the output file. You can create multiple output files for different tools. Your output files should show following:
  - What you were trying to test?
  - What commands did you use? (These should be detailed enough that we can reproduce your results)
  - Actual output
  - Make sure that your output includes the OpenFlow rules which were installed during the experiment.
  - Did it work as you expected?
  - Did you see any anomalies?
- The `explanation.txt` should explain how you created the topology, How your solution is learning about it, why you get the behavior that your tests are showing? (both expected and unexpected).
- Use `README.txt` file to tell us about any other modifications you have done to the system which we should be aware of, or to explain if we need to do something extra in order to run your solution.

References

[1] Git tutorial. `https://git-scm.com/docs/gittutorial`.

[2] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[3] Mininet Walkthrough. `http://mininet.org/walkthrough/`, 2014. [Last visited on 2015-05-06].

[4] 2011. http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[5] OpenFlow tutorial. `https://github.com/mininet/openflow-tutorial/wiki`, 2015. [Last visited on 2016-04-11].

[6] POX Wiki. `https://openflow.stanford.edu/display/ONL/POX+Wiki`, 2014. [Last visited on 2015-05-06].

[7] A tcpdump Primer with Examples. `https://danielmiessler.com/study/tcpdump/`. [Last visited on 2015-05-06].