1 Introduction

The goal of this assignment is to give an introduction to OpenFlow [1, 3]. 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 [1] and familiarize yourselves with the basic OpenFlow elements, before setting up the environment.

3 Setting up OpenFlow

There is plenty of documentation on how to install and setup OpenFlow and a step-by-step tutorial is provided in [4] under the OpenFlow Tutorial link. Additionally, [2] provides a more thorough Mininet walkthrough if desired.

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 from their website [4]. The link is provided under the Install Required Software section of the tutorial.

3.2 Installing VM in VirtualBox

The OpenFlow VM image already contains mininet and nox (i.e. for the controller part) installed. Therefore, all left is to load the VM in VirtualBox as described below:

• Open VirtualBox item Click New, then Continue

• Name your VM (e.g. acn13), select Linux as OS, Ubuntu as the version and click Continue

• Set the memory to 512MB and click Continue

• Check Use existing hard disk and select the extracted OpenFlowTutorial*.vdi image

• Installation is complete, press Done

• Additionally, go to the Settings → Network → Adapter 2, select the Enable adapter box and set it to host-only network. This allows you to easily access your VM through your host machine.

When pressing to start running your VM, enter the username and password, which are both openflow.

3.3 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 the username and password were introduced, you should now have access to the /home folder which contains several folders (i.e. nox, 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 openflow@[IP here]

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

3.4 Developing a network topology

All development takes place 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.

3.4.1 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 specific hosts, simply click Capture and then Interfaces and then select the desired host.

3.4.2 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. A few useful commands to use in the Mininet console are provided below

- **nodes** – see the list of available nodes
- **h1 ifconfig** – check h1’s IP address
- **h2 ping -c 1 h3** – pinging combined with wireshark is very useful in diagnosing controller behavior; if you view the wireshark output, you should be able to view the traffic in the network as a result of the ping
- **exit** – end the mininet session

### 3.4.3 Custom topologies

There are many custom examples of topologies that can be found on exploring `/mininet/examples`. A lot of the functions used in these topologies are located in `/mininet/mininet/net.py`. When looking at the code implementing topologies, such as `emptynet.py`, you will see that the default NOX controller is used. If you would want to replace the custom controller with another, then the following line should be changed in the code:

```python
net = Mininet( controller = Controller )
```

to, for example,

```python
net = Mininet( controller = lambda name: NOX( name, ACN_controller) )
```

**More information on NOX can be found at http://www.noxrepo.org/nox/about-nox/**.

### 3.4.4 Example controllers

A basic NOX controller is found under `/nox/src/nox/coreapps/examples/pyloop.py`. This is a basic controller shell that does not perform any forwarding at this point. What it does is illustrate how one would be able to install desired event handlers. If, for example, we want the controller to react to packet_in events, then we would install an event handle in the install function.

The hub controller is available under `/nox/src/nox/coreapps/tutorial/pytutorial.py`. The first item to notice is the install function. The controller has an event handler for packet_in. Whenever a packet arrives at a switch that does not match any rules, it is sent to the controller. Currently, packets that arrive at the controller are forwarded to everyone except the port from which it came from. Detailed steps and description of the hub controller are included in section 5 of the OpenFlow tutorial.

### 4 Assignment

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

#### 4.1 Learning NOX controller

In the first part of the assignment, you should implement a simple learning NOX controller. The 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.

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

Most of the code will go in one function, `learn_and_forward()`. You will need to add roughly 10 lines to make a learning switch.

### 4.2 NOX controllers

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 output.

In steps 1 and 2, you are going to 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 the NOX 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, as already described in section 3.4.4 (Example controllers). This is the default behavior of the controller code provided in `pytutorial.py`. 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.
4.2.2 Learning controller

Once again, all traffic arriving at the switches will be directed to a NOX controller. The NOX controller will perform MAC learning. You can re-use the code of the learning switch from the first part of the assignment. Analyze the network behavior with the MAC learning controller.

a) Run $H_1$ ping c 100 $H_2$. How much time does it take to ping the first packet? Is there any difference with the hub controller?

b) Run $H_1$ ping c 100 $H_5$. How long does it take? Compare with $H_1$ ping c 100 $H_4$.

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 $H_1$ ping c 100 $H_2$. 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.

4.3 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 follows

a) IP1 Switch1
b) IP2 Switch1
c) IP3 Switch2
d) IP4 Switch2

where IP1, IP2, IP3 and IP4 represent the IP addresses of the four end hosts.

Additionally, your controller should enforce specific paths for the traffic going from one particular host to another, as follows

a) IP1 IP4 SW3
b) IP2 IP4 SW3

which means that any packets travelling from IP1 or IP2 to IP4 is required to traverse switch 3. For all other traffic (i.e. including the one from IP4 to IP1 or IP2) 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.
5 Hand-In Instructions

- Create a new .py file for each of the controllers implemented. Dump the outputs from the ping runs and, preferably, create a separate folder for each controller, placing the code, logs and measurements (where required) inside. Similar to assignment 8, you will use SVN to handin this assignment.

- You are going to use SVN (http://subversion.apache.org) to handin your assignments for this course. Make sure you have it installed on your computer. On Ubuntu, you can install it by typing 'sudo apt-get install subversion'.

- We have set up a directory for each student in the course and the directories are named according to NETHZ User IDs. Clone your SVN folder to your computer using the following command (Replace the NETHZ_USERNAME with your own NETHZ User ID and enter the whole command in a single line).

  `svn co --username NETHZ_USERNAME https://svn.inf.ethz.ch/svn/systems/acn13_students/trunk/NETHZ_USERNAME`

- Create a new directory called assignment9 inside your directory.
- Copy your solution file into the newly created assignment folder and then type:

  `svn add assignment9`

  This will add your assignment folder and its contents into to your working copy and schedule them for addition to the SVN repository. They will be uploaded and added to the repository on your next commit when you type:

  `svn commit -m "checking in assignment 9"`

  Note that the string you provide after the -m switch does not have to be the same as the one provided here. It is intended to be a message for keeping a history on how the document evolves.

- After the handin, if you discover any mistakes in your solution and want to submit a revised copy, place your new solution files into the assignment folder and type:

  `svn commit -m "checking in revised assignment 9"`

- For more information about how to use SVN, start with typing `svn --help`
References


6 Hints

There is no great nox API around, so there are several files that you will need to look into in order to find the appropriate function declarations to use. Included in this list of hints are other Python functions that you may find useful. This list should cover all the basic functionality that will be necessary to complete the assignment.

6.1 References

/nox/src/nox/lib/core.py contains most of the higher level functions (e.g. sending packets, installing flows).
/nox/src/nox/packet/packet_utils.py
/nox/src/nox/packet/ethernet.py
/nox/src/nox/packet/packet_base.py contain some convenient functions for packet header analysis/parsing.
/nox/src/include/openflow/openflow/openflow.h contains many of the openflow variables that you may need to use.
/mininet/examples/ and /mininet/mininet/ contain examples for custom topologies and their function definitions.

6.2 Event registration

register_for_packet_in(handler) – no matching flow rules come to controller
register_for_datapath_leave(handler) – switch down
register_for_datapath_join(handler) – switch join
post_callback(handler) – custom event handler

6.3 Packet analysis

In the learn_and_forward() function, the arguments packet and packet.arr are passed. packet contains parsed header information that is easy to grab using functions found in the packet library.
packet.arr is the actual buffer data that needs to be modified if you wish to send this buffer.

6.4 Printing controller information

In contrast to using the typical print statements, it is much more useful to print to a log. You may find it useful to declare the following at the top of your file:

```python
logger = logging.getLogger(nox.[path\_to\_code\_file].noxtest)
logger.info(XYZ)
```

To view the output, you can take a look at the controller log file that can be found at /tmp/c0.log.

6.5 Installing dataflow rules

Installing dataflow rules is quite straightforward. First, you should specify a flow rule, followed by the action associated with it.

```python
flow = extract_flow(packet)
flow[core.IN\_PORT] = import
actions = [[openflow.OFPAT\_OUTPUT, [0,prt[0]]]]
self.install_datapath_flow(dpid, flow, CACHE\_TIME\_OUT, openflow.OFP\_FLOW\_PERMANT, actions, bufid, openflow.OFP\_DEFAULT\_PRIORITY, import, buf)
```

One can perform more actions by simply adding to the list (e.g. [[openflow.XYZ, args], [openflow.WXY, args]]). Please consult with core.py for more details.

6.6 Sending a packet

Sending a single packet is a good starting point before attempting to install dataflow rules. Please try to see if you can forward on a specific port once you have learned the MAC.

```python
self.send\_openflow(dpid, bufid, buf, ?, inport)
```

6.7 Switch identifier

The switch ID (dpid) will be its MAC address if specified in the Mininet topology (e.g. If MAC = 00:01:02:03:04:01, then its dpid will be 0x000102030401).