INSTITUTO SUPERIOR TÉCNICO

Traffic Engineering

Lab Project #4

Software-Defined Networking and OpenFlow

Introduction to Mininet (part I)

Fernando Mira da Silva

NOVEMBER 2017

1 Goal

In this lab project we will introduce a test environment for testing and deploying SDN networks and the OpenFlow protocol. Note that this guide should be completed on a single lab class; the full SDN lab (lab guides #4 and #5) should be completed in two lab classes. The report should cover both labs and must

Most of the simulation will be based on the Mininet suite. Mininet creates a **realistic virtual network**, running **real kernel, switch and application code**, on a single machine (VM, cloud or native), in seconds, with a single command.

Nearly every operating system virtualizes computing resources using a process abstraction. Mininet uses process-based virtualization to run many hosts and switches on a single OS kernel. Linux supports network namespaces, a lightweight virtualization feature that provides individual processes with separate network interfaces, routing tables, and ARP tables. The full Linux container architecture adds chroot() jails, process and user namespaces, and CPU and memory limits to provide full OS-level virtualization, but Mininet does not require these additional features. Mininet can create kernel or user-space OpenFlow switches, controllers to control the switches, and hosts to communicate over the simulated network. Mininet connects switches and hosts using virtual ethernet (veth) pairs. This enables to simulate several network components inside a single VM.

2 Required software

While Mininet can be built from scratch, it is also available in a fully operational pre-packaged Mininet/Ubuntu VM distribution, which already includes Mininet and OpenFlow binaries and tools.

In order to perform the planned tests, you'll need:

- The VM image (check https://github.com/mininet/mininet/wiki/Mininet-VM-Images);
- A Virtualization system which is able to run the VM image (VirtualBox, VMware,…);
- An X server running in your host system;
- An ssh client.

3 Testing the VM

1. Start the VM and log into the system. The default username/password is mininet/mininet.
2. Check the IP of the VM machine using ifconfig.
3. In your host system, start a X based command line terminal (e.g., xterm or equivalent) and type

```
ssh –Y mininet@[IP of your VM machine]
```

Check if you are able to log in the system via your X terminal.

You should get a prompt

```
mininet@mininet-vm:~$
```

or similar.

4. Check if X forwarding is running opening additional xterm X terminals in the mininet VM. They should appear on your X host system.
5. Open wireshark inside the mininet VM, using the command

   ```
   $ sudo wireshark &
   ```

   The Wireshark window should appear on your host X system.

4 Testing mininet

Type the following command to display a help message describing Mininet's startup options:

```
$ sudo mn -h
```

This walkthrough will cover typical usage of the majority of options listed.

4.1 Start Wireshark

To view control traffic using the OpenFlow Wireshark dissector, first open wireshark in the background:

```
$ sudo wireshark &
```

In the Wireshark filter box, enter this filter, then click `Apply`:

```
of
```

In Wireshark, click Capture, then Interfaces, then select Start on the loopback interface (`lo`).

For now, there should be no OpenFlow packets displayed in the main window.

4.2 Interact with Hosts and Switches

Start a minimal topology and enter the CLI:

```
$ sudo mn
```

The default topology is the `minimal` topology, which includes one OpenFlow kernel switch connected to two hosts, plus the OpenFlow reference controller. This topology could also be specified on the command line with `--topo=minimal`. Other topologies are also available out of the box; see the `--topo` section in the output of `mn -h`.

All four entities (2 host processes, 1 switch process, 1 basic controller) are now running in the VM. The Mininet CLI comes up.

In the Wireshark window, you should see the kernel switch connect to the reference controller.

Display Mininet CLI commands:

```
mininet> help
```

Display nodes:

```
mininet> nodes
```

Display links:

```
mininet> net
```

Dump information about all nodes:

```
mininet> dump
```

You should see the switch and two hosts listed.

If the first string typed into the Mininet CLI is a host, switch or controller name, the command is executed on that node. Run a command on a host process:

```
mininet> h1 ifconfig -a
```

You should see the host's `h1-eth0` and loopback (`lo`) interfaces. Note that this interface (`h1-eth0`) is not seen by the primary Linux system when `ifconfig` is run, because it is specific to the network namespace of the host process.

In contrast, the switch by default runs in the root network namespace, so running a command on the "switch" is the same as running it from a regular terminal:

```
mininet> s1 ifconfig -a
```

This will show the switch interfaces, plus the VM's connection out (`eth0`).

For other examples highlighting that the hosts have isolated network state, run `arp` and `route` on both `s1` and `h1`.

Note that *only* the network is virtualized; each host process sees the same set of processes and directories. For example, print the process list from a host process:

```
mininet> h1 ps -a
```

This should be the exact same as that seen by the root network namespace:

```
mininet> s1 ps -a
```

It would be possible to use separate process spaces with Linux containers, but currently Mininet doesn't do that. Having everything run in the "root" process namespace is convenient for debugging, because it allows you to see all of the processes from the console using `ps`, `kill`, etc.

4.3 Test connectivity between hosts

Now, verify that you can ping from host 0 to host 1:

```
mininet> h1 ping -c 1 h2
```

If a string appears later in the command with a node name, that node name is replaced by its IP address; this happened for h2.

You should see OpenFlow control traffic. The first host ARPs for the MAC address of the second, which causes a `packet_in` message to go to the controller. The controller then sends a `packet_out` message to flood the broadcast packet to other ports on the switch (in this example, the only other data port). The second host sees the ARP request and sends a reply. This reply goes to the controller, which sends it to the first host and pushes down a flow entry.

Now the first host knows the MAC address of the second, and can send its ping via an ICMP Echo Request. This request, along with its corresponding reply from the second host, both go the controller and result in a flow entry pushed down (along with the actual packets getting sent out).

Repeat the last `ping`:

```
mininet> h1 ping -c 1 h2
```

You should see a much lower `ping` time for the second try (< 100us). A flow entry covering ICMP `ping` traffic was previously installed in the switch, so no control traffic was generated, and the packets immediately pass through the switch.

An easier way to run this test is to use the Mininet CLI built-in `pingall` command, which does an all-pairs `ping`:

```
mininet> pingall
```

The alternative - better for running interactive commands and watching debug output - is to spawn an xterm for one or more virtual hosts. In the Mininet console, run:

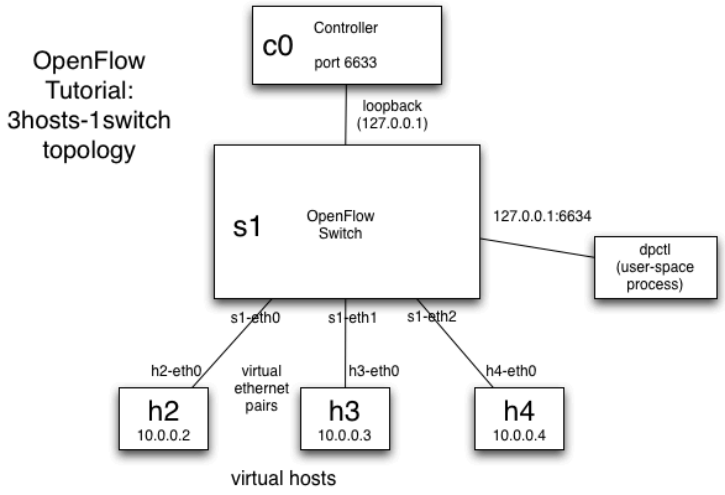
```
mininet> xterm h1 h2
```

You can close these windows now, as we'll run through most commands in the Mininet console.

5 Test the network

5.1 Network instantiation

The network for the first exercise includes 3 hosts and a switch.



To create this network in the VM, in an SSH terminal, enter:

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

This tells Mininet to start up a 3-host, single-(openvSwitch-based) switch topology, set the MAC address of each host equal to its IP, and point to a remote controller which defaults to the localhost. Note that in this case, the openvSwitch is created with an empty table (without any action rules).

Here's what Mininet just did:

Created 3 virtual hosts, each with a separate IP address.

Created a single OpenFlow software switch in the kernel with 3 ports.

Connected each virtual host to the switch with a virtual ethernet cable.

Set the MAC address of each host equal to its IP.

Configure the OpenFlow switch to connect to a remote controller.

5.2 ovs-ofctl Example Usage

The ovs-ofctl program is a command line tool for monitoring and administering OpenFlow switches using the OpenFlow API. It can also show the current state of an OpenFlow switch, including features, configuration, and table entries.

Create a second SSH window if you don't already have one, and run:

```
$ sudo ovs-ofctl show s1
```

The show command connects to the switch and dumps out its port state and capabilities.

Here's a more useful command:

```
$ ovs-ofctl dump-flows s1
```

Since we haven't started any controller yet, the flow-table should be empty.

5.3 Ping Test

Now, go back to the mininet console and try to ping h2 from h1. In the Mininet console:

```
mininet> h1 ping -c3 h2
```

Note that the name of host h2 is automatically replaced when running commands in the Mininet console with its IP address (10.0.0.2).

Do you get any replies? Why? Why not?

In your SSH terminal issue the following commands:

```
# ovs-ofctl add-flow s1 in_port=1,actions=output:2
```

```
# ovs-ofctl add-flow s1 in_port=2,actions=output:1
```

Check the flow-table

```
# ovs-ofctl dump-flows s1
```

Comment and explain the output.

Run the ping command again. In your mininet console:

```
mininet> h1 ping -c3 h2
```

Do you get replies now? Check the flow-table again and look the statistics for each flow entry. Is this what you expected to see based on the ping traffic?

Try pings in between other network hosts. What can you conclude?

Now try

```
sudo ovs-ofctl del-flows s1
```

Try again a ping between h1 and h2. What happens?

5.4 Start Controller and view Startup messages in Wireshark

Now, with the Wireshark dissector listening, start the OpenFlow reference controller. In your SSH terminal:

```
$ controller ptcp:6653
```

This starts a simple controller on port 6653 that acts as a learning switch without installing any flow-entries.

You should see a bunch of messages displayed in Wireshark, from the Hello exchange onwards. As an example, click on the Features Reply message. Click on the triangle by the 'OpenFlow Protocol' line in the center section to expand the message fields. Click the triangle by Switch Features to display datapath capabilities - feel free to explore.

These messages include:

Message	Type	Description
Hello	Controller->Switch	following the TCP handshake, the controller sends its version number to the switch.
Hello	Switch->Controller	the switch replies with its supported version number.
Features Request	Controller->Switch	the controller asks to see which ports are available.
Set Config	Controller->Switch	in this case, the controller asks the switch to send flow expirations.
Features Reply	Switch->Controller	the switch replies with a list of ports, port speeds, and supported tables and actions.
Port Status	Switch->Controller	enables the switch to inform that controller of changes to port speeds or connectivity. Ignore this one, it appears to be a bug.

Check all relevant OF messages in wireshark and identify them.

5.5 View OpenFlow Messages for Ping

Now, we'll view messages generated in response to packets.

Before that update your wireshark filter to ignore the echo-request/reply messages (these are used to keep the connection between the switch and controller alive): Type the following in your wireshark filter, then press apply:

```
 of and not (of10.echo_request.type or
of10.echo_reply.type)
```

Run a ping to view the OpenFlow messages being used. You will need to run this without having any flows between h1 and h2 already installed, e.g. from the "add-flows" command above:

```
sudo ovs-ofctl del-flows s1
```

It's also recommended to clean up ARP cache on both hosts, otherwise you might not see some ARP requests/replies as the cache will be used instead:

```
mininet> h1 ip -s -s neigh flush all
```

```
mininet> h2 ip -s -s neigh flush all
```

Do the ping in the Mininet console:

```
mininet> h1 ping -c1 h2
```

In the Wireshark window, you should see a number of new message types:

Message	Type	Description
Packet-In	Switch->Controller	a packet was received and it didn't match any entry in the switch's flow table, causing the packet to be sent to the controller.
Packet-Out	Controller->Switch	controller send a packet out one or more switch ports.
Flow-Mod	Controller->Switch	instructs a switch to add a particular flow to its flow table.
Flow-Expired	Switch->Controller	a flow timed out after a period of inactivity.

Explain what you see.

Re-run the ping, again from the Mininet console (hitting up is sufficient - the Mininet console has a history buffer):

```
mininet> h1 ping -c1 h2
```

If the ping takes the same amount of time, run the ping once more; the flow entries may have timed out while reading the above text.

This is an example of using OpenFlow in a *reactive* mode, when flows are pushed down in response to individual packets.

Alternately, flows can be pushed down before packets, in a *proactive* mode, to avoid the round-trip times and flow insertion delays.

5.6 Benchmark Controller w/iperf

iperf is a command-line tool for checking speeds between two computers.

Here, you'll benchmark the reference controller;

In the mininet console run :

```
mininet> iperf h1 h2
```

This Mininet command runs an iperf TCP server on one virtual host, then runs an iperf client on a second virtual host. Once connected, they blast packets between each other and report the results. Check the iperf for all host combination and report the results.

Exit Mininet:

```
mininet> exit
```