PART A (W4_PE2A): OPENFLOW WITH MININET

The first part of this practical exercice is dedicated to the use of the Mininet simulator.

The Mininet open-source network simulator is designed to support research and education in the field of Software Defined Networking systems. Mininet creates a simulated network that runs real software on the components of the network so it can be used to interactively test networking software.

Software Defined Networking (SDN) is a relatively new technology, but it is already being deployed in some networks: most famously, in Google's internal network. Many companies are developing products to deploy and support networks using SDN technologies. In the following test drive, we will use Mininet to simulate and test some SDN scenarios and evaluate Mininet as network simulator.

Mininet was created by a group of professors at Stanford University to be used as a tool to research and to teach network technologies.

Mininet is designed to easily create virtual software-defined networks consisting of an OpenFlow controller, a flat Ethernet network of multiple OpenFlow-enabled Ethernet switches, and multiple hosts connected to those switches. It has built-in functions that support using different types of controllers and switches. We can create also complex custom scenarios using the Mininet Python API.

Mininet uses Linux network namespaces to create virtual nodes in the simulated network. This is a lightweight and fast way to create virtual nodes but it does not provide fully separated virtual machines and it is not possible to save configurations on each of the virtual nodes after the simulation is shut down.

controller processes in the VM's root namespace -- essentially they are

just processes running on the VM. It is also possible to set up the controllers and the switches each in their own network namespace so they operate as separate virtual machines networked to each other across virtual Ethernet interfaces.

In this example, we will just use the default configuration. This makes all virtual interfaces we set up in the simulation available to be monitored by programs like Wireshark so it simplifies the observation of events in the simulated network.

1. Setup a Simple Network

First, we will look at a very simple network scenario.

Mininet must be run as root. Become root and travel to root's homedir:

```
$ su
# cd
```

A special initialization script allows now to start the lab in a super clean state:

```
# ./prepare-clean-openflow-lab.sh
```

Mininet can be started with options that specify the network topology to be created. To see all available options run the Mininet command mn with the -h option to see the help menu.

```
# mn -h
```

The Mininet network topology can be specified using the --topo option. If started without any topology options, Mininet will create the minimal

connected to an OpenFlow switch, which is connected to an OpenFlow controller.

To create a network topology with four virtual machines connected to one switch, enter the command:

```
# mn --topo=single,4
```

In this scenario, we did not specify the type of controller so we are using Mininet's default controller, ovs-controller, which implements the functionality of a basic layer-2 MAC-learning switch. The controller learns the MAC addresses of the hosts connected to the network and programs the switch's flow tables to set up connections between hosts that are sending packets to each other.

You will see the following output as Mininet creates the simulated network topology:

```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Running terms on localhost:10.0
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

But this was only for testing :-). Actually for our first switching experiments we want only two machines. To this purpose we need to reset mininet and clean its internal data. First, quit mininet:

```
mininet> exit
```

...and clear any residual state or processes using:

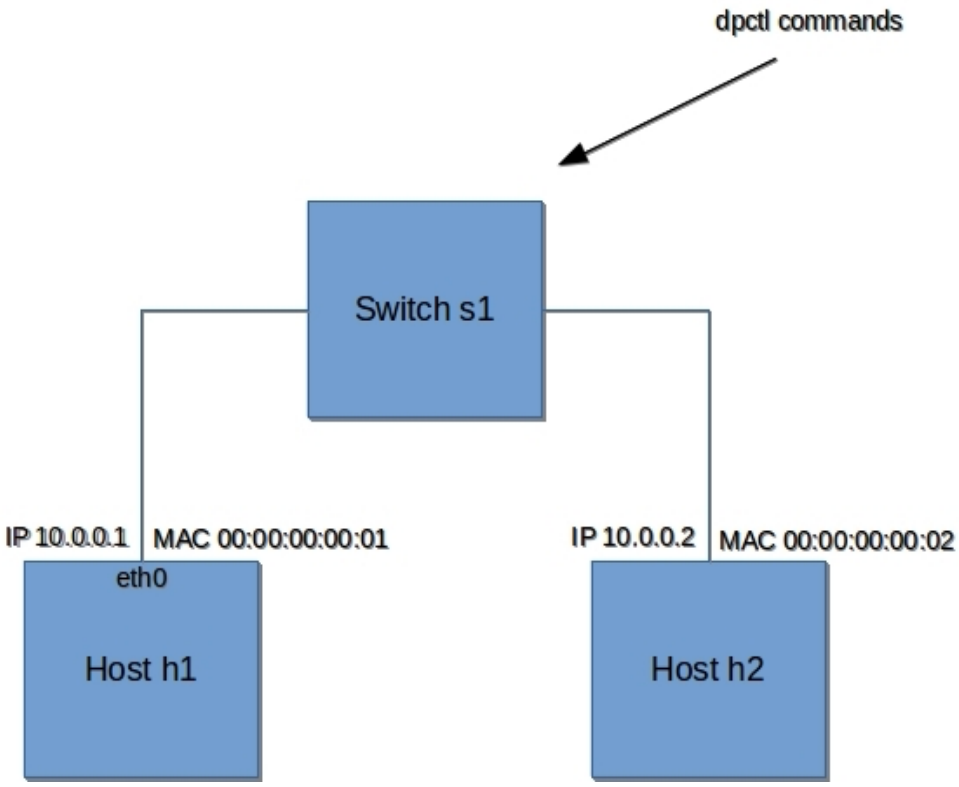
```
mn -c
```

Now prepare a simple topology with two machines (the --mac option creates MAC adresses mirroring the IP addresses):

```
# mn --mac --topo=single,2
```

You see that creating a network with mininet is as simple as typing a single command.

The layout is now the following:



Note: This version of mininet creates by default a parasitic flow that needs to be removed. For this purpose, while mininet is still running, use another root terminal, and type :

```
# ovs-ofctl del-flows s1
```

2. Dpctl Example Usage

dpctl is a utility that comes with the OpenFlow reference distribution and enables visibility and control over a single switch's flow table. It is especially useful for debugging, by viewing flow state and flow counters. You can run it either independantly or within mininet:

```
mininet> dpctl show
*** s1 ----------------------------------------------
------------------------
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS
QUEUE_STATS ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN
SET_DL_SRC SET_DL_DST
         SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC
SET_TP_DST ENQUEUE
 1(s1-eth1): addr:6e:c4:b2:0f:11:19
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:c2:35:4b:d7:53:ac
     config:     0
     state:      0
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:6a:6f:cc:f0:71:44
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal
miss_send_len=0
mininet>
```

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

A more useful command is:

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------
------------------------
NXST_FLOW reply (xid=0x4):
mininet>
```

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

3. Dpctl Flow Control Usage

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

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

QUESTION W4.PE2A.1 (1/1 point)

Do you get any replies ? (NA=1)

- Yes
- No ✔

Vous avez utilisé 2 essais sur 3

As you saw before, the switch flow table is empty. Besides that, there is no controller connected to the switch and therefore the switch doesn't know what to do with incoming traffic, leading to ping failure.

You'll use dpctl to manually install the necessary flows:

```
mininet> dpctl add-flow in_port=1,actions=output=2
*** s1 ------------------------------------------------
------------------------
mininet> dpctl add-flow in_port=2,actions=output=1
*** s1 ------------------------------------------------
------------------------
mininet>
```

This will forward packets coming at port 1 to port 2 and vice-versa. Verify by checking the flow-table:

```
mininet> dpctl dump-flows
```

Now run again the ping command:

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

QUESTION W4.PE2A.1.2 (1/1 point)

Do you get replies now ? (NA=1)

- (●) Yes ✔
- ( ) No

Correct: selected: You are correct, the flow that we installed works fine.

Vous avez utilisé 1 essais sur 3

NOTE: if you didn't see any ping replies coming through, it might be the case that the flow-entries expired before you start your ping test. When you do a "dpctl dump-flows" you can see an "idle_timeout" option for each entry, which defaults to 60s. This means that the flow will expire after 60secs if there is no incoming traffic. Run again respecting this limit, or install a flow-entry with longer timeout:

```
mininet> dpctl add-flow
in_port=1,idle_timeout=120,actions=output:2
```

4. Dpctl Wrapup

This manipulation demonstrates clearly the extreme flexibility of the OpenFlow architecture, but why exactly ? See below.

QUESTION W4.PE2A.3 (1/1 point)

Indicate the correct answers. Check all that apply. (NA=2)

- [x] The OpenFlow architecture introduces new flexibility by separating the data flow and the control flow.
- [ ] The OpenFlow architecture introduces new flexibility by providing a new protocol.
- [x] The OpenFlow architecture introduces new flexibility by leaving entirely to external sofware the control of the OpenFlow switch.

✔

Correct:
Yes, an approach that allows whole new perspectives.
Yes! A new protocol would not, by itself, be a proof of added flexibility.
Sure! This allows for entirely new possibilities for management.

Vous avez utilisé 2 essais sur 3

The above manipulation was interesting, however installing manually flows for each destination is quite cumbersome. Our next step is to automate the control by means of a software controller.