PART B (W4_PE2B): EXPERIMENTING A POX CONTROLLER

The second part of this practical exercice aims at experimenting a POX controller.

1. POX Controller

POX is a Python-based SDN controller platform geared towards research and education. As a platform, it features a rich python API allowing to program easily any SDN scenario. In the lab context, we will test two basic POX applications: a simple hub (this section) and a learning switch (see below).

First, clean any remains of the previous experiment:

```
mininet> exit
# mn -c
```

Open a new xterm, become root on it and go to the pox folder:

```
$ su
# cd /root/pox
```

Start POX in verbose mode with the hub application:

```
./pox.py log.level --DEBUG misc.of_tutorial
```

On the initial terminal, start wireshark (still as root) as a background task:

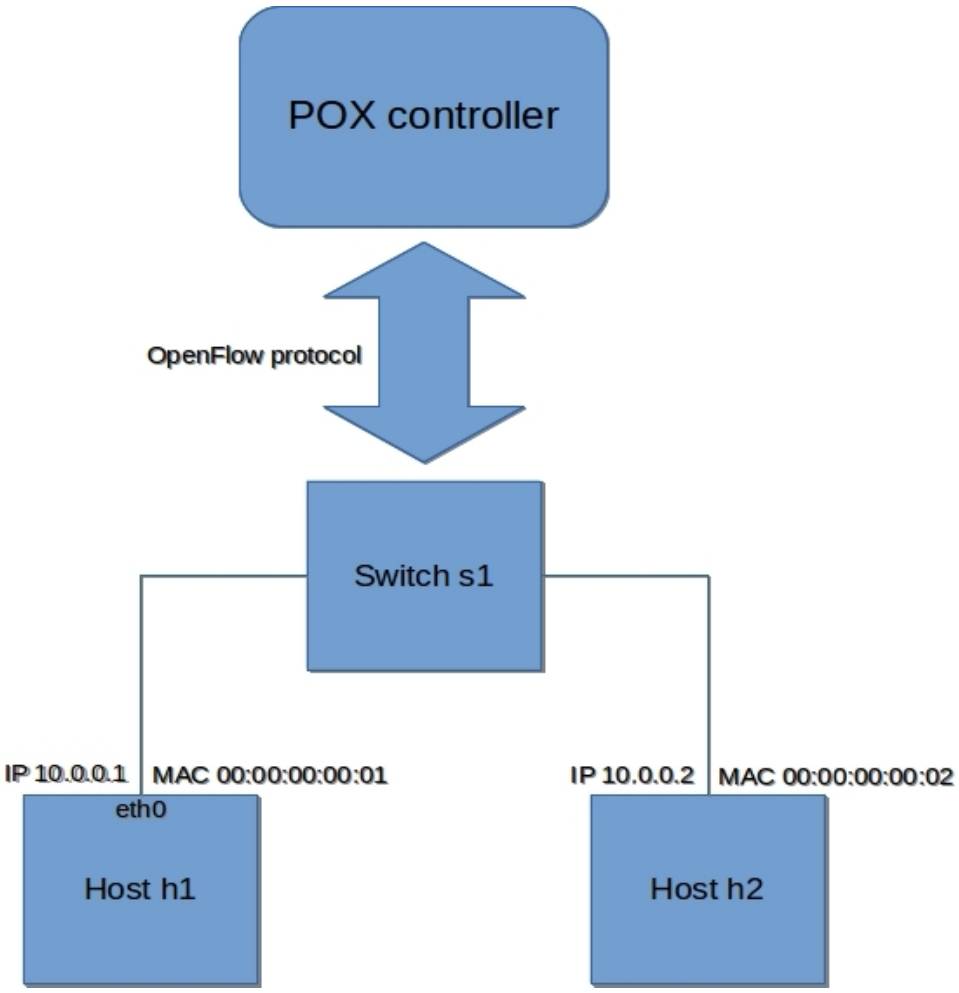
```
# wireshark &
```

In wireshark, start a capture on the loopback interface lo: and position the OpenFlow filtering by typing the "openflow_v1" keyword in the filter input field.

Start mininet with a default topology, but this time require explicitly to connect to a remote controller:

```
# mn --mac --controller=remote
```

The layout is now:



The POX process should show that it is now connected to the switch:

```
INFO:openflow.of_01:[00-00-00-00-00-01 10] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01
10]
```

Now try a single ping:

```
h1 ping -c1 h2
```

Wireshark should then show two OFPT_PACKET_IN PDUs with "Reason: no matching flow (table-miss flow entry)", respectively followed by OFPT_PACKET_OUT PDUs. This corresponds to twice the following events:

- The switch receives a data packet (ICMP ping) and does not know what to do with it, since it has no installed flows. It sends therefore a OFPT_PACKET_IN request to the controller for further instructions.
- The controller implements a hub behaviour, the unique answer OFPT_PACKET_OUT is to instruct the switch to forward the data packet to all its outputs.
- The switch receives this answer and forwards the data packet.

Note: Take the time to explore the of_tutorial.py file, which is located in ./pox/pox/misc/of_tutorial.py

2. POX-based Learning Switch

To further demonstrate the flexibility of OpenFlow, we will now set up a learning switch application. Basically a learning switch behaves like a hub (flooding to all outputs) until it *learns* which mac addresses are connected to which ports. Once it has this knowledge, it forwards the data only to the unique destination port, which is naturally much more efficient.

First, stop and clean mininet:

```
mininet> exit
# mn -c
```

Stop the controller by means of "control-C" and restart the wireshark capture.

Start POX in verbose mode, this time with the learning switch application:

```
./pox.py log.level --DEBUG forwarding.l2_learning
```

Restart mininet but with the --mac option which sets conveniently MAC adresses:

```
# mn --mac --controller=remote
```

h1 has now the MAC address 00:00:00:00:00:01, h2 has 00:00:00:00:00:02.

Fire a single ping:

```
h1 ping -c1 h2
```

This time the controller application exhibits a new OpenFlow action: the installation of flows. During the learning process, the controller has acquired knowledge about the ethernet destination ports and programs the OpenFlow switch accordingly:

```
DEBUG:forwarding.l2_learning:installing flow for
00:00:00:00:00:01.1 -> 00:00:00:00:00:02.2
DEBUG:forwarding.l2_learning:installing flow for
00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
```

The status of these flows can be confirmed on the switch by means of:

```
mininet> dpctl dump-flows
*** s1 -------------------------------------------------------
------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2.058s, table=0, n_packets=1,
n_bytes=98, idle_timeout=10,
 hard_timeout=30, idle_age=2,
priority=65535,icmp,in_port=2,vlan_tci=0x0000,

dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_sr
c=10.0.0.2,nw_dst=10.0.0.1,
 nw_tos=0,icmp_type=0,icmp_code=0 actions=output:1
 cookie=0x0, duration=2.062s, table=0, n_packets=1,
n_bytes=98, idle_timeout=10,
 hard_timeout=30, idle_age=2,
priority=65535,icmp,in_port=1,vlan_tci=0x0000,
 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,

nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,ic
mp_code=0 actions=output:2
```

Please note that the idle_timeout parameter is set to a very short duration of 10 seconds, after which the switch will "forget" this flow. In this case the flow installation process restarts.

Note: Take the time to explore the l2_learning.py file, which is

3. POX Wrapup

The POX platform with its rich API allows to program virtually any kind of behaviour, so it seems. Check below:

QUESTION W4.PE2B.1 (1 point possible)

The POX platform introduces a whole set of possibilities:

Check all that apply. (NA=3)

- [✓] We might program a router with it! ✓
- [✓] We might program a packet inspection application with it!
- [ ] We might program a MAC filtering application with it! ✓
- [✓] We might program an IP address filtering application with it! ✓

✖

Incorrect:
Exactly! This is often given as lab exercise.
No because once a flow is created, relevant packets are not managed by POX anymore.
Why not?
Of course!

Vous avez utilisé 3 essais sur 3