# Hands-on Lab Description

# CS-NET-00008 – SDN Controller (POX) Lab

**Category:**

```
CS-NET: Computer Networking
```

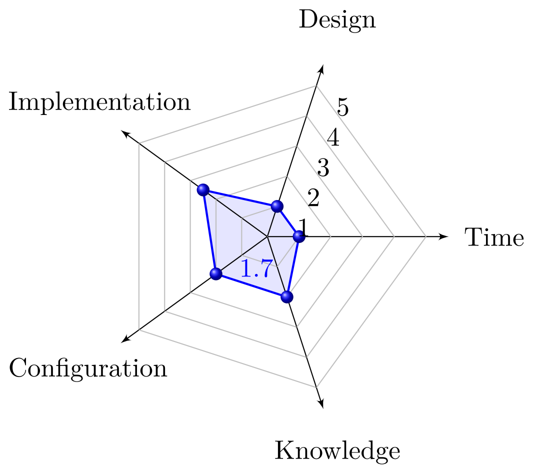
**Objectives:**

```
Install Open Virtual Switch (OVS)
Basic configuration of OVS on linux virtual machine
```

**Estimated Lab Duration:**

```
Expert: 40 minutes
Novice: 120 minutes
```

**Difficulty Diagram:**



**Difficulty Table.**

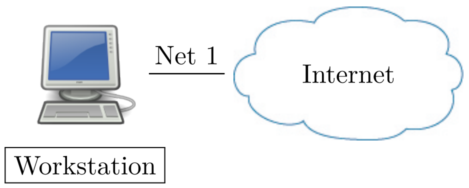
| Measurements | Values (0-5) |
|---|---|
| Time | 1 |
| Design | 1 |
| Implementation | 2.5 |
| Configuration | 2 |
| Knowledge | 2 |
| Score (Average) | 1.7 |

**Required OS:**

```
Linux: Ubuntu 18.04 LTS (Xenial Xerus)
```

**Lab Running Environment:**

```
VirtualBox https://www.virtualbox.org/ (Reference Labs: CS-SYS-00101)
```



```
Workstations: Linux (Ubuntu 18.04 LTS)
Internet: Accessible
```

**Lab Preparations:**

```
Open Virtual Switch (OVS) (Reference Lab: CS-NET-00006)
Mininet (Reference Lab: CS-NET-00007)
Install the POX controller by issuing command: git clone http://github.com/noxrepo/
    pox - Note: the VM that ThoTh Lab may provide POX pre-installed. Moreover,
    installing mininet or containernet will also install POX.
```

## Task 1.0 Environment Setup and Connectivity Testing

In this task, you will start to use the an SDN controller called POX controller which is an open source development platform for Python-based software-defined networking (SDN) control applications, such as the OpenFlow SDN controller.

1. Starting POX controller to makes OpenFlow switches to act as traditional Ethernet hubs, broadcasting (or flooding) every packet out all interfaces except the arrival interface.

```
$      cd pox
$      ./pox.py -verbose forwarding.hub
```

2. On a new terminal window, creating a single Mininet topology with an OpenFlow switch, remote controller and three hosts.

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

```
    DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
    INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
    INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
```

   Note: if you can see the similar message that red box showing above, it meaning the pox has successfully connected to you Mininet topology

3. The xterm command is a useful command for running interactive commands on each host.

```
$      xterm h1 h2 h3 # on the terminal of Mininet topology
```

   Note: The xterm can show the individual terminal for all the hosts we created on the current Mininet topology

4. On xterm terminal of host 2

```
    host2>  tcpdump -n -i h2-eth0
```

5. On xterm terminal of host 3

```
    host3>  tcpdump -n -i h3-eth0
```

   Note: tcpdump is a data-network packet analyzer computer program that runs under a command-line interface. It allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached.

6. On the xterm terminal of h1, try to send 10 packets from host 1 to host 2 to test the functionality between tcpdump and controller.

```
    host1>  ping -c 10 10.0.0.2 # 10.0.0.2 is the IP address of host 2
```

   **Next, try to implement the command shows above and see whether h2 and h3 can catch the packet sent by h1**

7. Useful Commands:
   **To terminate POX:** *Ctrl-C*
   **To terminate Mninet Topology:** *exit*

## Task 2.0 Invoking POX with Different build-in Components

In this task, you will get to invoke the POX with your Mininet by utilizing different functionalities of the POX controller.

1. Making your OpenFlow switches act as a type of l2 learning switch. However, this one is probably just about the simplest possible way to do is correctly. Unlike *l2_learning,l2_pairs* installs rules based purely on MAC addresses.

```
$       cd pox
$       ./pox.py samples.pretty_log forwarding.l2_learning
$       sudo mn --topo single,3 --mac --arp --switch ovsk --controller=remote # run
          this on a new terminal
```

```
      [core ] POX 0.5.0 (eel) is up.
      [openflow.of_01 ] [00-00-00-00-00-01 2] connected
```

As we can see above, this made switch in Mininet topology is acting as a type of L2 learning switch.

Note: we need to specify the controller to remote when we're creating the Mininet topology so that the POX controller can detect and connect to it.

2. Running POX itself doesn't do much – POX functionalities is provided by components (POX comes with a handful of components, but POX's target audience is really people who want to be developing their own). Components are specified on the commandline following any of the POX options above. An example of a POX component is forwarding.l2_learning. This component makes OpenFlow switches operate kind of like L2 learning switches. To run this component, you simply name it on the command line following any POX options

| **option** | **meaning** |
|---|---|
| –verbose | Display extra information (especially useful for debugging startup problems) |
| –no-cli | Do not start an interactive shell (No longer applies as of betta) |
| –no-openflow | Do not automatically start listening for OpenFlow connections |

3. You can run POX's web server component along with l2_learning

```
$       ./pox.py -verbose forwarding.l2_learning web.webcore
```

4. l2_learning has a "transparent" mode where switches will even forward packets, that are usually dropped (such as LLDP messages), and the web server's port number can be changed from the default (8000) to an arbitrary port

```
$       ./pox.py -verbose forwarding.l2_learning --transparent web.webcore
          --port=8888
```

Note, after your started your web server along with you controller and connected to your topology, you can view the details of your controller at *localhost:8888* or *127.0.0.1:8888* where the *8888* is the assigned port number when your starting the server

5. **Next, start to implement the component mentioned with your own Mininet topology above and see whether you can setup correctly. There are other useful components provided in POX controller**

*forwarding.hub*: The hub example just installs wildcarded flood rules on every switch, essentially turning them all into $10 ethernet hubs.

*forwarding.l2_learni*: This component makes OpenFlow switches act as a type of L2 learning switch. This one operates much like NOX's "pyswitch" example, although the implementation is quite different. While this component learns L2 addresses, the flows it installs are exact-matches on as many fields as possible. For example, different TCP connections will result in different flows being installed.

*forwarding.l2_pairs*: Like l2_learning, this component also makes OpenFlow switches act like a type of L2 learning switch. However, this one is probably just about the simplest possible way to do it correctly. Unlike l2_learning, l2_pairs installs rules based purely on MAC addresses.

*forwarding.l3_learning*: This component is not quite a router, but it's also definitely not an L2 switch. It's an L3-learning-switchy-thing. Perhaps the most useful aspect of it is that it serves as a pretty good example of using POX's packet library to examine and construct ARP requests and replies.
l3_learning does not really care about conventional IP stuff like subnets – it just learns where IP addresses are. Unfortunately, hosts usually do care about that stuff. Specifically, if a host has a gateway set for some subnet, it really wants to communicate with that subnet through that gateway. To handle this, you can specify "fake gateways" in the commandline to l3_learning, which will make hosts happy. For example, if you have some machines which think they're on 10.x.x.x and others that think they're on 10.0.2.x and they think there are gateways at the ".1" addresses:

```
$       ./pox.py forwarding.l3_learning --fakeways=10.0.0.1,10.0.2.1
```

*forwarding.l2_multi*: This component can still be seen as a learning switch, but it has a twist compared to the others. The other learning switches "learn" on a switch-by-switch basis, making decisions as if each switch only had local information. l2_multi uses openflow.discovery to learn the topology of the entire network: as soon as one switch learns where a MAC address is, they all do. Note that this means you must include openflow.discovery on the commandline.

......

6. Please figure out the meaning of forwarding.l2_pairs and forwarding.l3_learning and what are the differences between them (Please visit `https://openflow.stanford.edu/display/ONL/POX+Wiki.html` for more information)

## Lab Assessments (100 points)

Lab assessment for accomplishing Task 1 and Task 2 depends on the following facts: (Note: partial tasks require mannual grading)

1. (50 points) The POX controller can

   - connect to a mininet topology

```
$         ./pox.py -verbose forwarding.hub
$
$         sudo mn --topo single,3 --mac --arp --switch ovsk
            --controller=remote
```

```
        INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
        INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
        *** Creating network
        *** Adding controller
        Unable to contact the remote controller at 127.0.0.1:6653
        Connecting to remote controller at 127.0.0.1:6633
        *** Adding hosts:
        h1 h2 h3
        *** Adding switches:
        s1
        *** Adding links:
        (h1, s1) (h2, s1) (h3, s1)
        *** Configuring hosts
        h1 h2 h3
        *** Starting controller
        c0
        *** Starting 1 switches
        s1 ...
        *** Starting CLI:
```

- start the *forwarding.l2_pairs* component correctly

```
$         ./pox.py -verbose forwarding.l2_pairs
$
$         sudo mn --topo single,3 --mac --arp --switch ovsk
            --controller=remote
```

```
        DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
        INFO:openflow.of_01:[00-00-00-00-00-01 2] connected

        *** Creating network
        *** Adding controller
        Unable to contact the remote controller at 127.0.0.1:6653
        Connecting to remote controller at 127.0.0.1:6633
        *** Adding hosts:
        h1 h2 h3
        *** Adding switches:
        s1
        *** Adding links:
        (h1, s1) (h2, s1) (h3, s1)
        *** Configuring hosts
        h1 h2 h3
        *** Starting controller
        c0
        *** Starting 1 switches
        s1 ...
        *** Starting CLI:
```

- start the *forwarding.l3_learning* component correctly

```
$         ./pox.py forwarding.l3_learning --fakeways=10.0.0.1,10.0.2.1
$
$         sudo mn --topo single,3 --mac --arp --switch ovsk
            --controller=remote
```

```
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected

*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

2. (50 points) While connect to the controller, the mininet topology can (Please take snapshots for each task)
   - open the xterm terminal correctly
   - enable the ping command between two different host in two different xterm terminal (two different hosts)

The challenging task evaluation is optional and the instructor can decide on how to evaluate students' performance.