

IMT Atlantique

Département Informatique

Technopôle de Brest-Iroise - CS 83818

29238 Brest Cedex 3

URL: www.imt-atlantique.fr



UE PRIP

2023

Lab 4: SDN

Version: 1.4

Report filled-in by:



IMT Atlantique

Bretagne-Pays de la Loire

École Mines-Télécom

1. Objectives

- Understand the concept of SDN and figure out its differences comparing to the traditional architecture.
- Gain insight into the operation of OpenFlow and observe the operations.
- Program different basic SDN controllers based on the POX controller.

2. Pre-Lab

- Read the introduction of this document
- Familiarize with command `ovs-ofctl`, you can consult its man page
- Familiarize with POX controller, read the POX's API available in [3]. In particular, study the following sections: 'Working with packets: pox.lib.packet', 'OpenFlow Events: Responding to Switches/Packet in', 'OpenFlow Messages/ ofp_packet_out - Sending packets from the switch, ofp_flow_mod - Flow table modification' and 'Match Structure'.

3. Introduction

3.1. Software Defined Networks

Software Defined Networking (SDN) is an architectural approach that optimizes and simplifies network operations and gives hope to change some limitations of current network infrastructures. First, it separates the network's control logic (the control plane) from the underlying routers and switches that forward the traffic (the data plane). Second, it allows a logically centralized network control which is often realized as an SDN controller, and where network switches become simple forwarding devices. See Fig. 1.

The four key characteristics to remember about SDN are [6]:

1. The control and data planes are disjointed
2. Forwarding decisions are flow-based instead of destination-based
3. Control logic is moved to an external entity, called SDN controller or NOS (Network Operating System)
4. Network is programmable through applications running on top of the controller

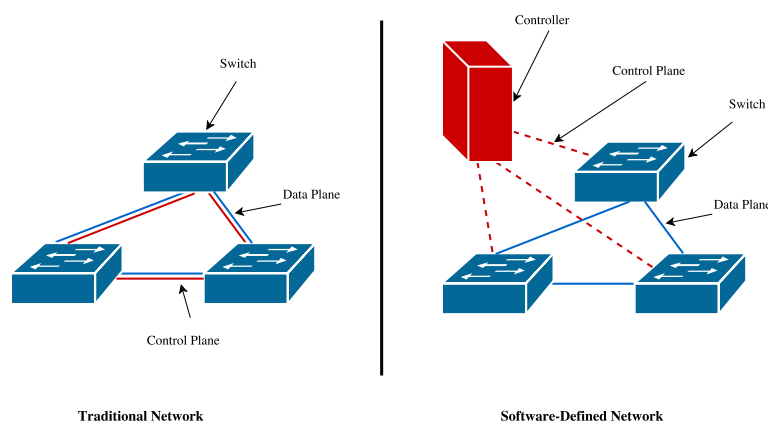


Figure 1: Difference between traditional architecture and SDN

SDN has gained significant attention among major industrial players including Cisco, Broadcom, Google, IBM, and Intel, and has been deployed in wide area networks, campus networks, and data centers.

In this practical class, you will learn the fundamentals of SDN and gain hands-on experience. In order to put all the theory into practice, we will use Mininet to emulate the entire network of switches (that will be just simple forwarding devices, without any “brain” of their own), virtual hosts (running standard Linux software), the links between them and an SDN controller.

3.2. OpenFlow

OpenFlow is a communication protocol that gives access to the forwarding plane of a network switch or router over the network. It enables network controllers to program the path of network packets across a network of switches. This separation of the control from the forwarding allows for more sophisticated traffic management than is feasible using classical routing protocols. Also, OpenFlow allows switches from different vendors -often each with their own proprietary interfaces and scripting languages- to be managed remotely using a single, open protocol [7]. In other words, OpenFlow is an API that is standardized between control plane and data plane. OpenFlow is an enabling technology for SDN.

How does OpenFlow work?

An OpenFlow Switch presents one or a set of flow tables (according on the OpenFlow version); each flow table entry contains a set of packet fields to match, and an action, as shown in Fig. 2. When an OpenFlow Switch receives a packet it has never seen before, for which it has no matching flow entries, it sends this packet to the controller. The controller then makes a decision on how to handle this packet. It can drop the packet, or it can add a flow entry directly in the switch on how to forward similar packets in the future. More specifically, the process of a packet can be described as follows [4]:

- If a packet matches an entry in the flow table, perform the actions according to the flow table.
- If a packet does not match any entry in the flow table, send it to the Openflow controller:
 - The controller will figure out what to do with such packet.
 - The controller will then respond to the switch, informing how to handle such a packet so that the switch would know how to deal with such packets next time.
 - For each flow, ideally the controller will be queried once.

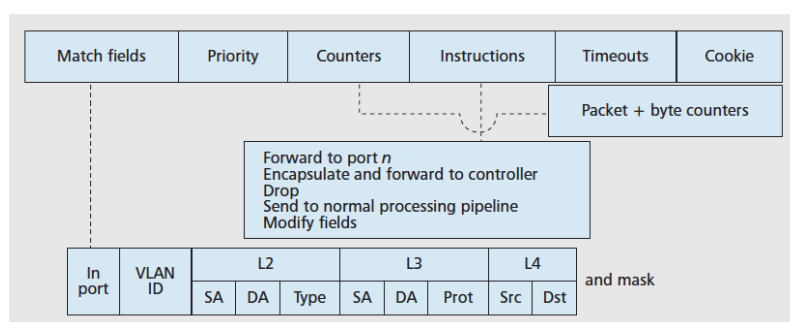


Figure 2: A flow table entry.

In the newest versions of OpenFlow, a switch has an Openflow pipeline which contains multiple flow tables, each flow table containing multiple flow entries. Packets are matched against multiple tables in the pipeline. Matching starts at the first flow table and may continue to additional flow tables.

In our lab exercise, we will be working with just one OpenFlow table for each switch.

3.3. Open vSwitch

Open vSwitch is an open source software switch designed to be used as a virtual switch in virtualized server environments. A virtual switch forwards traffic between different virtual machines (VMs) on the same physical host and also forwards traffic between VMs and the physical network. With virtual switches we can have several switches supported by one same infrastructure. Open vSwitch is open to programmatic extension and control using OpenFlow and the OVSDb (Open vSwitch Database) management protocol [8].

OVSDb allows to manage (i.e. configure) and control the virtual switch, while OpenFlow allows to control the switch (i.e. add forwarding rules). Both protocols are complementary and virtual switch can work with either both at a time or just one of them. In this lab, we are going to interact first with an OVS through OVSDb, and secondly through a python OpenFlow API (the POX controller). At any time, you can use the OVSDb commands, for instance, to dump the flow table of the switch.

Open vSwitch (OVS) is not the only existing virtual switch. There are also, for instance, VMware virtual switch (standard & distributed) and Cisco Nexus 1000V. OVS differs from the commercial offerings from VMware and Cisco. One point worth noting about OVS is that there is not a native SDN Controller or manager, like the Virtual Supervisor Manager (VSM) in the Cisco 1000V or vCenter in the case of VMware's distributed switch. Open vSwitch is also meant to be used with third-party controllers and managers.

4. Hands On

4.1. Understanding flows: OVS configuration using OVSDb management protocol

The network you'll use in this exercise includes a single virtual switch and three hosts connected as shown in Fig. 3. We are not using an SDN controller but provide configuration information through the OVSDb management protocol.

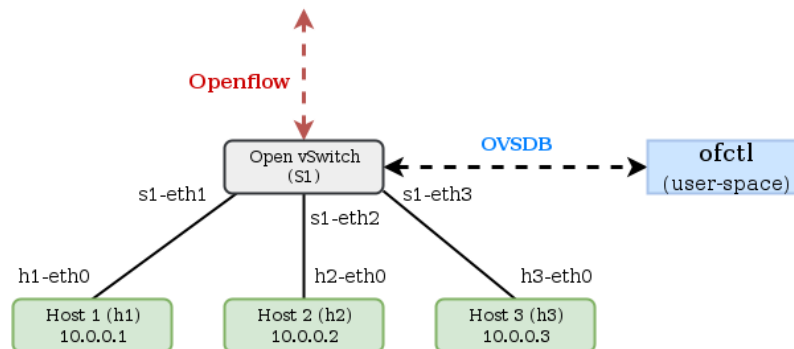


Figure 3: Topology without controller

1. Open the course's VM
2. Open a terminal inside the VM, and go to the `~/net_labs/sdn` directory. If the `~/net_labs/` directory is not present, you can obtain a local copy by cloning it:
`git clone https://redmine.telecom-bretagne.eu/git/net_labs`. If it is present, update its content: `git fetch origin`, `git merge master`.
3. Create the topology shown in Fig. 3, by launching Mininet with the following command:

```
sudo mn --topo single,3 --mac --switch ovsk --controller=none
```

| Port name | Port number | Hosts connected to this port |
|-----------|-------------|------------------------------|
| s1-eth1 | | |
| s1-eth2 | | |
| s1-eth3 | | |

Table 1: Port numbers and connected hosts

This tells Mininet to create a topology with a switch and 3 hosts attached to it. The hosts will be assigned static IP addresses and MAC addresses. In the above command, there are some important keywords worth paying attention to:

- **—mac**: makes the mac address of Mininet hosts the same as their node number, this just simplifies the analysis of captures
- **—switch ovsk**: uses Open vSwitch in kernel mode for each of the switches
- **—controller none**: we are not using any controller in this first exercise

4. In order to double-check that everything started correctly, use the following Mininet commands, in Mininet's CLI:

- **nodes** - to list all virtual devices in the topology
- **net** - to list the links between them
- **dump** - to see more info about the hosts

Before continuing you should make sure you know the working topology, the mapping between switch's port numbers and names, and the port through which each host is connected.

5. Use the following commands to complete Table 1.

- **sudo ovs-ofctl show s1** on a (new) terminal of the VM¹
- In Mininet's console command **net**

OVSDb protocol includes also a client tool, we are in particular going to use the **ovs-ofctl** command, which allows to access the flow tables of a switch. There are different sub-commands:

- **show < switch-name >**: shows some informations about the switch, including the ports
- **dump-flows < switch-name >**: dump all the flows of a switch
- **add-flow < switch-name > < flow >**: add a flow in the switch, check the flow syntax
- **mod-flows < switch-name > < flow >**: modify all the flows that include the same match pattern
- **del-flows < switch-name > < flow >**: delete all the flows that include the same match pattern

You can find the complete specification in the man page of the **ovs-ofctl** command (type **man ovs-ofctl** in a terminal) or see [2].

¹You can also type any terminal command in Mininet, preceding the command by **sh**

Question 4.1.

Execute command `sudo ovs-ofctl dump-flows s1` on a terminal. You should obtain an empty answer. What is that command for?

6. Run the `pingall` command on Mininet's CLI

Question 4.2.

The ping shouldn't work. Explain why.

We will now fill in s1's flow table using `sudo ovs-ofctl add-flow` command.

7. Run the following command lines from a terminal:

- `sudo ovs-ofctl add-flow s1 priority=1000,in_port=1,actions=output:2`
- `sudo ovs-ofctl add-flow s1 priority=1000,in_port=2,actions=output:1`

By these lines, we have added two flows to s1. We have indicated the priority, the in_port and the out_port in the rules. The rule with higher value of priority overrules any other rule with lower value.

8. Type `sudo ovs-ofctl dump-flows s1`. This allows you to check the flows you just added before.
9. Check the connectivity with `pingall`.

Question 4.3.

Between which hosts is there connectivity? Explain.

10. Manually configure the connection between h1 and h3 using the following command lines

- `sudo ovs-ofctl add-flow s1 priority=1000,in_port=1,actions=output:3`
- `sudo ovs-ofctl add-flow s1 priority=1000,in_port=3,actions=output:1`

Question 4.4.

Verify again the connectivity with Mininet's pingall command. Which is the result? In particular, explain why you have no longer connectivity between h1 and h2. Why would you want to have connectivity simultaneously between all hosts?

For the moment we have used one type of flow action: "output" which indicates to send the incoming matched packet through the indicated output port (or interface). A flow action can also be "flood", "drop", among others. Remember you can find more details about action types in the man page of the `ovs-ofctl` command by typing `man ovs-ofctl` in a terminal or referring to [2].

The following is a useful extract of the man page on Flow Syntax:

Some ovs-ofctl commands accept an argument that describes a flow or flows. Such flow descriptions comprise a series of field=value assignments, separated by commas or white space. (Embedding spaces into a flow description normally requires quoting to prevent the shell from breaking the description into multiple arguments.)

Flow descriptions should be in normal form. This means that a flow may only specify a value for an L3 field if it also specifies a particular L2 protocol, and that a flow may only specify an L4 field if it also specifies particular L2 and L3 protocol types. For example, if the L2 protocol type `dl_type` is wildcarded, then L3 fields `nw_src`, `nw_dst`, and `nw_proto` must also be wildcarded. Similarly, if `dl_type` or `nw_proto` (the L3 protocol type) is wildcarded, so must be the L4 fields `tcp_dst` and `tcp_src`. `ovs-ofctl` will warn about flows not in normal form.

ovs-fields describes the supported fields and how to match them. In addition to match fields, commands that operate on flows accept a few additional key-value pairs:

- *table=number:* For flow dump commands, limits the flows dumped to those in the table with the given number between 0 and 254. If not specified (or if 255 is specified as number), then flows in all tables are dumped. (N.B. we will not use this key-value since we are working with just one table per switch)
- *actions=[action][,action...]:* Specifies a comma-separated list of actions to take on a packet when the flow entry matches. If no action is specified, then packets matching the flow are dropped. The following forms of action are supported (among others):
 - `output:port`
 - `normal`
 - `flood`
 - `in_port`

For example, if you run the following command on your switch, you can get a switch acting like a hub:

- `sudo ovs-ofctl add-flow s1 priority=1000,actions=FLOOD`

You can also specify the `in_port` of the packet you want to be forwarded by the switch acting as a hub, i.e. forwarded to all the ports except the one where it arrived.

11. We will try new actions on the switch and new matching rules. Delete the previous flows added to s1, using command `sudo ovs-ofctl del-flows s1`

12. Inside the Mininet CLI, verify the ARP cache of every host (`hi arp -n` for $i = 1$ to 3). Wait till they are empty, or use `hi arp -d <address>` to remove each entry.

On the other hand, to send the packets to a specific destination, another way to match a packet is to use the destination **IP address**. To do so, you have to add a **matching rule on the protocol type field of the Ethernet header too (Ethernet type or dl.type)** (e.g. 0x0800: matches IPv4 source/destination IP address). For example, you can run this command to forward the packets going to h1:

- `sudo ovs-ofctl add-flow s1 priority=1000,dl_type=0x0800,nw_dst=10.0.0.1,actions=output:1`

13. Following previous examples, write a new configuration in order to ensure complete connectivity between all hosts. Do not forget that you need also to take into consideration ARP messages. It can be handy to remember that the Ethernet type code of ARP is 0x0806.
14. Verify connectivity

Question 4.5.

What flows did you added? Copy the exact syntax. What was the result of your connectivity check?

15. Once the connectivity is established between all hosts, open wireshark at `s1` (From the Mininet CLI, run `s1 wireshark &`).
16. Start a new capture at all s1's three interfaces (s1-eth1, s1-eth2, s1-eth3)
17. Run a ping from h1 to h2.

Question 4.6.

Which protocol messages do you see? To which application is this protocol related?

18. Look at the ICMP messages and verify the source and destination IP addresses of these messages

Question 4.7.

Why don't we get any ICMP message at h3?

4.2. OpenFlow, using POX controller

In this section we are going to use an SDN controller, in this case POX. POX is a python-based OpenFlow controller, which provides a python API with which one can develop its own applications. POX is a suitable controller for learning purposes, since it is simple and allows to understand the SDN concepts and manipulate OpenFlow switches. Please note that this controller might not be suitable for production environments, where controllers such as OpenDaylight [1] are preferred.

We will keep the same topology already studied while adding a controller connected to s1 as shown in Fig. 4.

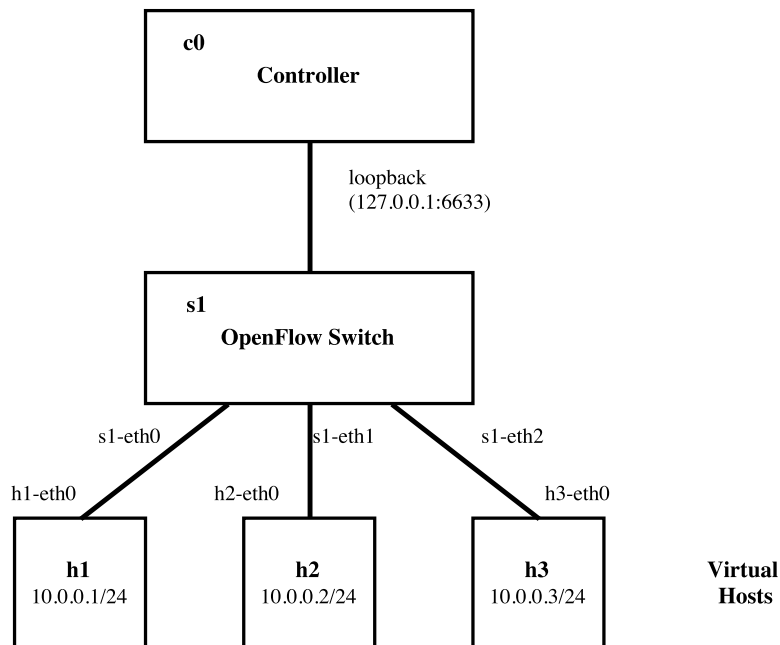


Figure 4: Topology with controller

4.2.1. Exercise 1 - A first simple controller

In this exercise, we are going to move the commands you have issued manually to a high-level-language script (python in this case), which will use POX's API to send OpenFlow messages to the switch, in order to instantiate the flows.

1. Exit the previous emulation by typing `exit` Mininet's CLI and clean it up using `sudo mn -c` on a terminal.
2. Go to the `net_labs` directory: `cd ~/net_labs/`
3. Copy file `~/net_labs/sdn/static_switch.py` into `/opt/pox/ext/` directory.
`sudo cp ~/net_labs/sdn/static_switch.py /opt/pox/ext/`
4. Run the controller using these commands on a new terminal:

```
cd /opt/pox/
./pox.py static_switch
```

5. Open a new terminal and launch the topology by running this command:

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

Notice that we have specified the `--controller remote` option. This tells Mininet that there will be a controller running *outside* Mininet. Indeed, this is the controller you have run in step 4.

6. Test connectivity between hosts using the `pingall` command. All hosts must be reachable.

Question 4.8.

Get an in-depth look at the first matching rule in the controller file (lines from 32 to 45 of file `static_switch.py`) What is the function of the action provided by these lines?

7. Launch wireshark at s1.
8. Start a new capture in s1-eth0, s1-eth1, s1-eth2 and the loopback interface “lo”
9. In wireshark, use the visualization filter `openflow_v1`.

Question 4.9.

What protocol messages are captured now? What is the new protocol appearing in this exercise by comparison to the last capture? Which is the protocol stack being used by this protocol?

4.2.2. Exercise 2 - L2 Learning switch

In this exercise, we are going to code a layer 2 learning switch using, as in the previous exercise, a high level programming language (python) and the POX controller. The code that you have seen in the previous exercise can help you understand how POX API works.

Question 4.10.

Which are the main differences between a Hub and a L2 Learning switch?

Implementation of the L2 learning switch

1. Copy file `~/net_labs/sdn/learning_switch.py` into `/opt/pox/ext/` directory, for that you can type the following on a terminal:

```
sudo cp ~/net_labs/sdn/learning_switch.py /opt/pox/ext/
```
2. Take a look to this file and in particular to the method `act_like_hub(...)`. What is the action used to ensure the hub behavior?

We will now keep the same topology as in Exercise 1 but will change the controller. The objective is to develop a layer 2 learning switch application. When a packet arrives to the switch, if the switch has no rule matching that packet, it will send the packet to the controller. The controller will process the packet. If the destination MAC of the packet is already associated with some port of the controlled switch, the controller will add a flow entry to the switch, in order to get the packet sent to the given output port, otherwise the controller will tell the switch to flood the packet on all ports, except the ingress port. At the same time, when receiving a packet, the controller will learn from the packet's "source mac address" and its "input port".

Note: if you need to print debugging information, you can call pox with the `----verbose` option:

```
./pox.py --verbose learning_switch
```

Following there is a naive algorithm for simple learning switch application:

```
if (source mac address is new)
    record the source mac and input port mapping
if (destination mac address is known)
    forward the packet to the destination
    install a flow table rule
else
    FLOOD the packet
```

3. Complete the method `act_like_switch()` in file `/opt/pox/ext/learning_switch.py` and modify that file in order to have that method executed whenever a packet arrives to the controller (instead of having method `act_like_hub(...)` being executed). Follow the comments provided in the code.
4. Test the proper functioning of your controller. For that stop the running controller if any, and run the new controller. Use, for instance, `pingall` command from Mininet's CLI
5. You can change the number of hosts in the topology, by running again the emulation with the number of hosts you want, for instance, for 10 hosts run

```
sudo mn --topo single,10 --mac --switch ovsk --controller remote
```

Your learning switch should work perfectly also in this topology.

Performance evaluation of the L2 learning switch and the Hub

Now your Learning Switch controller is working properly, let's test its performance. We will measure RTT (round trip time) using ping command and throughput using a command line tool called iperf. For both cases, we are going to run two tests, between two different hosts each, at the same time. We are going to run the tests both for the switch implementation and for the hub implementation. How to perform these tests is explained as follows.

1. Stop the emulation and run it again with the following command

```
sudo mn --topo single,4 --link tc,bw=1000 --mac --switch ovsk --controller remote
```

The new option we have added, `--link_tc,bw=1000`, tells Mininet to set all links' capacities to 1Gbps.

2. Open a xterm at hi, i=1 to 4

3. Run a 10 times ping to measure the average RTT between h1 and h2, and almost simultaneously run another ping from h3 to h4:

From h2: `ping -c 10 10.0.0.1`

From h4: `ping -c 10 10.0.0.3`

Observe the results.

4. Use the command line tool `iperf` to test the throughput. At the already opened terminals, start the following tests as simultaneously as possible:

At h1: `iperf -s`

At h2: `iperf -c 10.0.0.1`

At h3: `iperf -s`

At h4: `iperf -c 10.0.0.3`

Observe the results.

Let some time for the throughput test to run. Once the test finished, stop the servers (ctrl+c).

5. In the same way as before, test the performance of the hub implementation, in terms of RTT and throughput. For that you need to modify your controller to use the `act.like_hub()` method, and run it again. You do not need to restart the emulation.

Question 4.11.

What do you observe? What are the results obtained for RTT and bandwidth for the learning switch and for the hub. Explain the differences found and the causes for that differences.

Include your file `learning_switch.py` in the lab's report. (see Appendix where room is provided for this purpose.)

OpenFlow messages (Optional)

We will now take a look at the exchanged OpenFlow messages between switch and controller

1. Let's start the controller again, to see all messages. Stop your running controller
2. Start a new wireshark capture at s1's loopback interface
3. Run the controller again

Question 4.12.

What OpenFlow messages were captured after the controller coming up?

4. Perform a one time ping from h1 to h2: `h1 -c1 h2`

Question 4.13.

What OpenFlow messages were captured after the first time h1 tries to communicate with h2? Explain the purpose of the different messages.

5. Perform again a one time ping from h1 to h2

Question 4.14.

Do you see any OpenFlow message related to this second ping? Explain.

4.2.3. Exercise 3 - Extensions to traffic filtering (optional)

In this exercise, we will add some basic filtering feature to enforce a primitive security policy. For simplicity, let us suppose we want to filter out any HTTP traffic. To do so, we must properly determine the matching rule. Give a close look to POX documentation (in particular to the sections mentioned in the Pre-Lab section). For example, a matching rule in normal form must specify the data link layer payload type, (e.g. 0x0800 for IPv4) in this case, the protocol carried by the network layer packet, which is a decimal number with the value between 0-255 and is predefined for each traffic (ICMP is 1, TCP is 6, UDP is 17 [5]), and the destination or source transport port designed by `tp_dst` and `tp_src`, respectively.

First of all, we will test the results of an HTTP request between h1 (where a web server will run) and h2 which will act as client. We will try this before adding the filtering features. Everything should work fine. To do so:

1. Start a wireshark capture at h1-eth0 and at h2-eth0
2. Run a web server at h1, for that, from Mininet's CLI type: `h1 python3 -m SimpleHTTPServer 80 &`
3. Verify that your server is running well by accessing its index web page from h2, type at Mininet's CLI: `h2 wget -O - h1`
4. Look at both captures to see the packets exchanged. In particular observe the transport protocol used by HTTP and the port at which the web server listens. You will need this information for your matching rule.

Question 4.15.

Provide a pseudo-code for the new block of code you will need to add to your controller in order to filter out HTTP traffic. Do not proceed to the following steps if you don't have a clear answer to this question!

5. Extend your learning switch application to drop HTTP traffic.

Here are some tips: Remember to specify your matching rule in normal form (see introduction to this exercise), you might also need to take into account flows' priorities. For troubleshooting, at any moment you can use the command seen in Section 4.1 to query the switch flow table (`sudo ovs-ofctl dump-flows s1`). To go from your pseudo-code to the python code, you will need to take a close look to POX API [3].

6. Test your implementation as follows

- (a) Run a wireshark capture on h1 and another one in h2
- (b) Try again to access the index web page at h1 (`h2 wget -O - h1`)
- (c) Observe the output and the captured traffic. If everything is working well, you shouldn't be able to recover the index web page as you did in step 3

Question 4.16.

Is your implementation working properly? What is the result of the HTTP request? Do you see any HTTP packets in the captures? In which one? Explain your answer. Explain the new flow you have added in order to obtain this behavior.

Update the solution you have provided in the Appendix.

5. Conclusion

Question 5.1.

What are the main components of an SDN architecture?

Question 5.2.

What is OpenFlow? What functionalities of OpenFlow have you used in this lab? Explain with your own words.

Question 5.3.

According to what you've seen in previous labs and in this lab. Summarize with your own words the main difference between the classical networking paradigm and the SDN paradigm.

6. Acknowledgements

This lab was partly prepared based on:

- SDN Bootcamp provided by professors in University of Wisconsin [10]
- OpenFlow tutorial [9]

References

- [1] The OpenDaylight controller. [online] <https://www.opendaylight.org/>.
- [2] OpenvSwitch documentation. [online] <http://openvswitch.org/support/dist-docs/ovs-ofctl.8.txt>.
- [3] POX documentation. [online] <https://noxrepo.github.io/pox-doc/html/>.
- [4] Open Network Foundation. Openflow switch specification v1.4, 2013.
- [5] Assigned Internet Protocol Numbers IANA. [online] <https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>.
- [6] Diego Kreutz, Fernando M. V. Ramos, Paulo Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *CoRR*, abs/1406.0440, 2014.
- [7] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2), 2008.
- [8] Ben Pfaff and Bruce Davie. The open vswitch database management protocol. RFC 7047, 2013.
- [9] OpenFlow Tutorial. [online] <https://github.com/mininet/openflow-tutorial/wiki/Create-a-Learning-Switch>.
- [10] SDN Bootcamp Spring 2013 Sections University of Wisconsin. [online] <http://sdn.cs.wisc.edu/bootcamp/>.

7. Appendix: Your Exercise 3's code

You can include your code `learning_switch.py` in the box below.