Assignment 7: Intro to Software Defined Networks (SDN) with OpenFlow in Linux

# author: Aitor Martin Rodriguez ([aitor.martinrodriguez@uis.no](mailto:aitor.martinrodriguez@uis.no))

Department of Electrical Engineering and Computer Science, University of Stavanger

# Objectives

In this assignment you will put into practice what you have learned about Software Defined Networks (SDN).

The objectives of this assignment are the following:

* Deploy a simple software-defined network using *mininet* and the POX SDN controller (<https://github.com/noxrepo/pox>).
* Learn the basic concepts of the OpenFlow protocol in various test scenarios.

# Background / Scenario

In traditional networks, networking hardware is built to perform specific functions. For example, switches are built to do VLAN management and layer 2 switching, and routers are built to do layer 3 routing, OSPF, BGP… Restructuring a traditional network often requires swapping devices, cabling and manually reconfiguring network elements.

In Software Defined Networks (SDNs) there is a separation between two planes: the **data plane** (also known as forwarding plane) and the **control plane.** The data plane takes care of handling the incoming traffic and performing actions over them, such as dropping, forwarding or queueing. The control plane implements the intelligence in the network, with decision-making for forwarding, traffic shaping, security… Unlike in traditional networks, where devices implement the functions of both planes, SDNs replace routers, switches and other network elements with **generic forwarding nodes that only implement the data plane**. The **control plane** is instead taken care of by a **centralized controller** (or series of controllers) which has a full overview of the network.

**OpenFlow** was the first protocol proposed for sharing control information between a controller and forwarding nodes, and it has been standardized and maintained by the Open Networking Foundation since 2009. It is now widely supported by network switches and open-source SDN controllers. OpenFlow is based on **flow-based forwarding**: traffic is interpreted as a series of flows rather than packets, and decision-making is performed over flows rather than individual packets.

In this lab, you will deploy a simple emulated network over Linux and attach an SDN controller to it. Using OpenFlow, this controller will implement the control plane of your network. You will run different applications implementing the control plane and observe OpenFlow traffic using Wireshark.

# Requirements

* A computer running any Linux distribution, either natively or on a virtual machine.
  + You can use the same VM used previously in DAT230 or DAT300.
  + If you need help to set up a Linux VM, check the Lab Setup Guide provided on CANVAS.

***Preparation***

* + - 1. Install the following packages using the **apt** package management tool: *git, wireshark, mininet, xterm, iperf3.* If you are using your Linux installation from DAT230, these should already be installed. Remember to use *sudo* to run commands as a superuser. When installing Wireshark, press <Yes> when the installation window appears to allow superusers to capture packets.

$ sudo apt install -y <package1> [package2] [package3]...

* + - 1. Clone the following GitHub repository in your home directory.

$ git clone <https://github.com/amartin320/DAT300-SDN>

* + - 1. Inside the DAT300-SDN folder, clone the POX networking software platform from GitHub.

$ cd DAT300-SDN

$ git clone <https://github.com/noxrepo/pox>

# Instructions

## Start the custom *mininet* topology and run sanity checks.

Before bringing in the SDN controller, you will test that mininet works as expected and no libraries are missing. You will use a custom topology provided within the assignment files.

* + - 1. Start a mininet using the custom topology named **‘mytopo’** provided in the file ‘**topology.py’**. This topology is an asymmetric dumbbell topology with 2 hosts on one side (PC1 and PC2) and 1 host on the other side (PC3), as shown in Figure 1.

NOTE: Beware of long dashes (double dash, --) in the commands!

$ sudo mn --custom <topology\_file.py> --topo <topology\_name>

**A diagram of a block diagram

Description automatically generated**

Figure 1. Custom topology for this assignment

* + - 1. Verify that all the nodes are reachable by running ‘**pingall’** from the mininet console. You should see a message which says “\*\*\* Results: 0% dropped (6/6 received)”.
      2. Run a ping from PC1 to PC3 and verify that the latency stabilizes slightly above 10 milliseconds. You can run the ping from an xterm window (by running **‘xterm PC1’**) or you can also directly run **‘PC1 ping PC3’** from the mininet console.
      3. Verify that the S1-S2 link is bandwidth-limited to 20 Mbps by running **‘iperf’** from the mininet console.
      4. Stop the mininet and clean all processes.

mininet > exit

$ sudo mn -c

## Start the external SDN controller using the ‘forwarding.hub’ model, and observe OpenFlow traffic

Now, we are going to introduce the SDN controller into the topology, as shown in Figure 2.

A diagram of a network function

Description automatically generated

Figure 2. Custom topology with incorporated POX SDN controller

* + - 1. Restart the mininet, specifying that switches will attempt to connect to a remote SDN controller.

NOTE: No blank spaces in “remote,ip=127.0.0.1,port=6633”!

$ sudo mn --custom <topology\_file.py> --topo <topology\_name> --controller remote,ip=127.0.0.1,port=6633

**Q: According to the command above, where is the “remote” controller node located?**

* + - 1. Start a ping between two arbitrary hosts in the topology.

**Q: Was the ping successful? Why?**

*Hint:**You can run tcpdump on various interfaces to understand what is going on, e.g., to see if ICMP packets are leaving or reaching a node.*

* + - 1. Open Wireshark and start capturing traffic on the loopback interface (lo). You might have to run wireshark as sudo from a new terminal window to be able to capture traffic.

$ sudo wireshark

**Q: The capture will show a series of TCP segments in grey and red. What type of TCP segments are these? Which device is sending them and who is the destination? Interpret what is going on.**

* + - 1. Start the Wireshark capture again. In a new terminal, enter the “pox” directory and run the SDN controller (POX) using the “forwarding.hub” model.

$ python3 pox.py --verbose forwarding.hub

Verify that the SDN controller is attached to the forwarding nodes in the mininet topology, looking at the POX console output.

Verify that you can observe OpenFlow traffic in Wireshark, and stop the capture after a few seconds. If they are missing, add two columns to see the source and destination ports respectively (you can do this easily by selecting the port fields in any packet and pressing CTRL+SHIFT+I). Then save the capture in a PCAP file.

Answer the following questions:

**Q: Looking at the OpenFlow traffic on Wireshark, which transport protocol does OpenFlow use?**

**Q: How many OFPT\_HELLO messages can you see? Who is the sender and who are the receivers of each message? Note down the port numbers for each node participating in OpenFlow traffic.**

|  |  |
| --- | --- |
| **Node name** | **Transport Layer Port** |
|  |  |
|  |  |
|  |  |

**Q: Observe the OFPT\_FEATURES\_REQUEST and OFPT\_FEATURES\_REPLY messages. What kind of information is exchanged in these messages? Observe the “reply” messages and write down the information you recognize.**

NB! OFPT\_FEATURES\_REQUEST messages might be included together with OFPT\_STATS\_REQUEST messages in the same OpenFlow packet.

**Q: Why is this information needed by the SDN controller?**

**Q: Observe the OFPT\_FLOW\_MOD messages, which are used to modify the flow tables on the forwarding nodes. Which modifications is the controller applying to each node? Do these rules have a timeout – i.e., do they ever expire?** **Fill in the table with the match fields.**

Verify your answer to your last question by looking at the flow table in s1. You can do this by running the command “sudo ovs-ofctl dump-flows S1” from a new terminal.

**Q: What does the flow table in S1 show?**

**Q: According to the entry in the flow table, which network device is the OpenFlow switch emulating with this rule?**

**After the flow tables are configured, the SDN controller and the forwarding nodes exchange some messages periodically.**

**Q: What are these messages and what is their purpose? How often are they exchanged? Verify what happens if you shut down the SDN controller and try to explain it.**

* + - 1. Verify that the switches are behaving as expected by running a ping between two hosts and capturing traffic in various nodes to see where the traffic is going.

## Start the external SDN controller using the ‘forwarding.l2\_learning’ model, and observe OpenFlow traffic

* + - 1. Stop the SDN controller with CTRL+C. Start another Wireshark capture and run the controller again using the “l2\_learning” application.

**$ python3 pox.py --verbose forwarding.l2\_learning**

### Q: In this case, how many flows are added to the flow table initially? Why?

* + - 1. Start a ping between two arbitrary nodes, and observe how the flow table is updated by running “sudo ovs-ofctl dump-flows S1” from a new terminal. Keep the ping running.

**Q: Observe the OFPT\_PACKET\_IN messages. What do they carry inside? Looking at the header information, find out why they are being sent (i.e., the “reason” field).**

**Q: How many entries are there initially in the flow table? Do any of these disappear?**

**Q: What is the difference between the “idle” and the “hard” timeout values in the table?**

* + - 1. You can now close all the windows and terminals. Clean the mininet processes.

$ sudo mn -c

## Study the l2\_learning.py source code

In this part, you will study the code used in Part 3 for the “l2\_learning” model. This will be useful as preparation for Assignment 8.

* + - 1. Open the l2\_learning.py file located in pox/pox/forwarding with a text editor of your choice and study the code.
      2. Answer the following questions:

**Q: Which function of the LearningSwitch class is called when a new data packet (i.e., OFPT\_PACKET\_IN) is received in the controller?**

**Q: What is the name of the variable keeping the Layer 2 MAC address table in the controller? Which line fills the MAC address table with the information from the received packet?**

**Q: Which line configures an action to flood the packet if the destination MAC does not match any entries in the MAC address table?**

**Q: Which lines of code configure how long each installed flow will last in the flow table? Which are the hard-coded values?**