Lab 1: Introduction to SDN

General Instructions

In this series of labs, we will look at more advanced concepts in computer networks. We will focus on Software Defined Networks (SDN), a new paradigm in network management used nowadays in many large-scale deployments.

The successful completion of the labs requires your provision of a report for each lab. All exercises in the instructions must be answered unless stated otherwise. Your answers should be concise; you don't need more than a few lines to answer each question. Questions that needs to be answered are within boxes. Some exercises ask you to discuss with your lab partner. You do not need to provide written answers to those questions.

You should complete the labs in groups of two persons — use the group you've created in pingpong! You are of course encouraged to discuss with other groups, but all your submissions must be the results of your own work. Once finished, upload your solution as a PDF document to pingpong, and don't forget to identify both members of the group.

Additional documents, such as source code, are available on pingpong.

It is assumed that you run the labs in the windows environment at Chalmers. We use a virtual machine and VirtualBox to have access to a linux environment. You may use your own computers, however we might not be able to provide support in that case.

If you haven't done it yet, please watch the videos on SDN by David Mahler. A link is given on pingpong.

Software Defined Networks

In conventional networks, the control and management are done locally by dedicated equipment, such as routers and switches. Special algorithms, rules sets and specific hardware (ASICs) are used to perform operations like packet routing and flow forwarding. Configuration is device-centric, and topology changes, require human intervention or long transition phases to come back to normal performance. The lack of flexibility can especially hinder performances in high traffic networks such as Internet Service Providers backbones and within datacenters.

In SDN, the control and data planes are decoupled. The forwarding logic can now be *programmed* and is not integrated into the hardware as before. The control plane can therefore fully observe the system and sends commands to the elements of the data plane to configure them.

The control plane is composed of one or more *controllers*. A controller centrally defines the behavior of the network, by computing forwarding rules on request. Switches compose most of the elements of the data plane (along with routers) and are called *learning switches*.
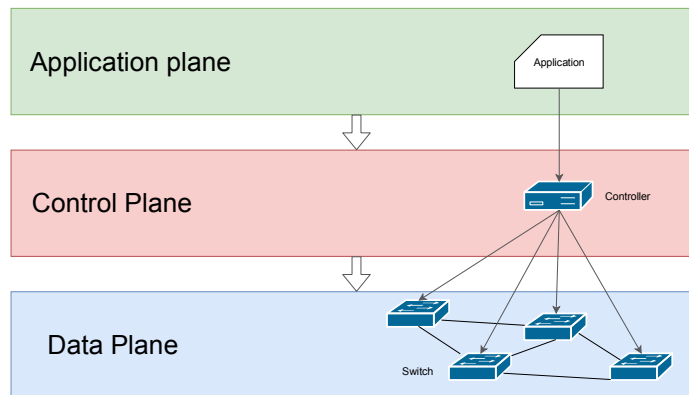
Figure 1: Architecture of Software Defined Networks (SDN)

Setup

We will experiment with Software Defined Networks using Mininet [1]. Mininet is a network emulator that can emulate switches, controllers and execute application code. It is available for many Linux distributions. If you are already using Linux, you can install it directly, although we recommend you to use a virtual machine (VM). Mininet makes updates to your network interfaces to work and might affect your system.

We will use VirtualBox [2] to run our virtual machine. VirtualBox is available for Windows, Mac OS, and Linux. You can use another hypervisor of your choice, but be aware that we will only support you for technical problems if you use VirtualBox.

A virtual machine has been prepared for you. Download it on Canvas.

Start VirtualBox and use File → Import Appliance. Import the VM we gave you. It might take a few minutes. Once it is done, you can start the VM. The credentials are given in the following table:

User	Password
sdn	sdn

Table 1: Virtual Machine Credentials

Please refer to the appendix if you want to create your own virtual machine.
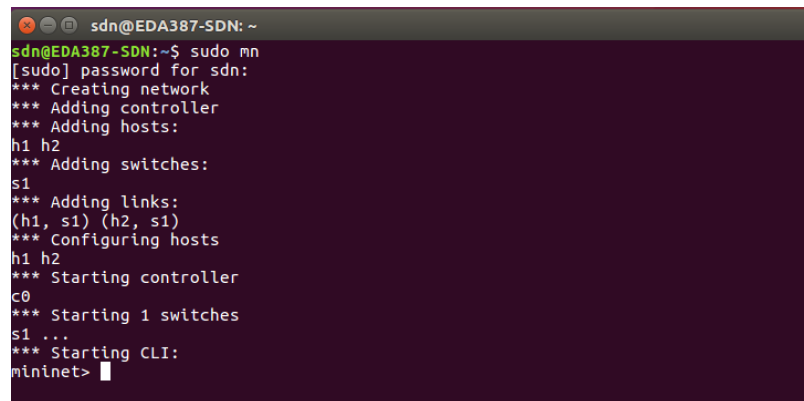
Testing your environment

You should now have a working environment. You can try to start a very simple network with the command:

---

[1] http://mininet.org/
[2] https://www.virtualbox.org/wiki/Downloads

$ sudo mn

If everything is working, you should see something like this:

Figure 2: Starting Mininet

You can now interact with the newly created network using the Command Line Interface (CLI). By default, Mininet creates a network composed of two host machines,*h1* and *h2*, and a switch in between, *s1*. A native controller *c0* is also present. You will now try the connectivity between the two hosts. Try the command *pingall*. You should see that no packets are dropped. You can also try *iperf* to see the available bandwidth. Now, ping from one specific host by using *h1 ping h2*. Finally, you can execute any bash commands on a specific host by using *xterm h1*, where you replace *h1* by the name of the host. Once the new terminal is created, check the network interfaces with *ifconfig*. Once you are done, use the command *exit* in the mininet CLI to stop the network.

Important!
Later on, Mininet might crash if you have errors in your scripts. Mininet won't be able to start anymore if that's the case. To correct this behavior, use:
$ sudo mn -c
The -c option will clean Mininet's internal files.

Part 1 - Network Discovery

Your environment is now ready and you should know how to interact with mininet. In this first part, we will take a closer look at how OpenFlow creates the forwarding rule.

From a terminal, go to the floodlight directory (in the VM we gave you, floodlight is located in ~/labs/lab1/) and starts your controller with the command:

$ java −jar target/ floodlight . jar

It takes a few seconds to initialize the controller. Once it is ready (you should see *Sending LLDP packets out of all the enabled ports* printed regularly), open a second terminal and

start the custom network topology we gave you, with the command:

$ sudo python topo.lab1.py

If everything is working correctly, you should see that the controller registered a switch.

Question 1.   In a conventional network, how can a switch know where to forward an incoming packet? Please explain briefly the mechanism.

Question 2. Ping from one host to another. You should see that at the very beginning, the RTT is significantly longer.   Why? Stop pinging and wait for about 30 seconds, and try again. Is the RTT now stable? Why?

Question 3.   Plot the Round Trip Time (RTT) evolution over time.   You need to produce a plot with the runtime as the x-axis and the RTT time as the y-axis.   You need two curves, the RTT from h1 to h2, and the RTT from h2 to h1.One measurement is not enough! You will need to restart both the network and the controller many times and average over multiple measures.

Remember, if Mininet crashes or has any problem, use the command **$ sudo mn -c**

Part 2 - Understanding OpenFlow

We saw just before that a switch must learn what action it should execute if it receives a new packet. We will now take a closer look at how OpenFlow behave in the presence of new flows.

Open a new terminal.Start Wireshark in superuser mode with the command sudo wireshark (at startup, a message error should appear, this is normal.Start capturing incoming packets on the loopback interface (lo). To only display the important packets, use *openflow.v4* as the display filter (don't forget to press enter to activate the filter).   In the other terminals, start the controller and the mininet topology as before.You should see new traffic appearing in Wireshark.

Question 4.   Explain the roles of the different packets you see. You can use http://flowgrammable.org/sdn/openflow/message-layer/ to understand them.   We are using Openflow 1.3.Do not describe all the packets, but only the most important ones! Typically at the very beginning,   once the switch contacts the controller, and when a new packet is received.

We will now directly look into the memory of the switch.   Open vSwitch, the software used to run virtual switches, offers a command to look into the details of OpenFlow, *ovs-ofctl.*As with Wireshark, you can see the communications between your switch and your controller. Run

$ sudo ovs−ofctl snoop s1 −O OpenFlow13

You can also just check the switch's state with

$ sudo ovs−ofctl show s1 −O OpenFlow13

Start pinging between the two hosts again. Now execute the command

$ sudo ovs−ofctl dump−flows s1 −O OpenFlow13

You should see three flows.

> **Question 5.** Explain what each flow represents. Briefly explain what each field is used for.
>
> **Question 6.** Imagine now that a web server is running on h1. h2 starts an HTTP request to h1. What will happen? Describe briefly the messages exchanged between the switch and the controller, and the flows that would appear in the dump-flows result. Note: you can try to emulate the behavior by opening a terminal on h2 and by using the *wget* command and *python -m SimpleHTTPServer*.

Part 3 - Floodlight's User Interface

We have seen how SDN work from the user perspective, and from the switch perspective. We will now investigate how the controller views the network.

Floodlight has two interfaces we can use: an API, and a user interface. While both the network and the controller are running, connect to http://localhost:8080/ui/index. html.

You should see a dashboard appearing, with the switches and the hosts *discovered* by the controller. Look at the topology and discuss with your partner the different elements. Go then to the Switches tab, and select the switch. Start pinging again. In the Flows table, do you see the same flows as before? Discuss with your partner.

Stop the network and the controller. We will now try a different topology. Start again the controller, and start a new network with the command:

$ sudo mn −−topo=tree,depth=3 −−controller=remote,ip=0.0.0.0,port=6653

> **Question 7.** How many switches are present? What kind of topology is it? How many switches must be programmed to ping from h1 to h8?

We try again with a different topology, linear this time.

$ sudo mn −−topo=linear,n=1,k=7 −−controller=remote,ip=0.0.0.0,port=6653

> **Question 8.** How does the first RTT evolve with the number of switches to cross?

Conclusion

In this lab, we looked at SDN as a new paradigm for network management. We introduced the concepts of controllers and learning switches, and how OpenFlow allows us to *program*

our switches for flow forwarding. In the next lab, we will introduce you to the concept of self-stabilizing control plane, and how we can tolerate failures in SDN.

> For the next lab,   please read *Renaissance:Self-Stabilizing Distributed SDN Control Plane* (the paper is present on pingpong). We do not require you to fully understand all the details of the paper or to be able to do the proofs.   But pay close attention to algorithm 1, you will work with an implementation of that system during the next session.

Appendix

Creating your own VM

On your favorite OS, use the command:

```
$ sudo apt−get install   git  ant wireshark
```

Git is a version control system, we will use it to download and install a specific version of Mininet. Ant is used to build Java applications, and Wireshark [3] is a packet analyzer that listens and records all communications in the network.

Once this is done, we will clone the Mininet repository from Github.   Use the command:

```
$ git   clone https://github.com/mininet/mininet.git mininet
```

This will create a directory and copy all the files.   You then need to install Mininet.   From the same location, execute the following command:

```
$ ./mininet/utils/ install .sh −rmf −V 2.5.5 −3
```

By doing so, you are installing mininet and OpenFlow version 1.3.

Finally, we will use Floodlight[4] as our OpenFlow controller. It is open-source and written in Java. Go to http://www.projectfloodlight.org/download/ and download the version **1.2**. To build the controller, just use

```
$ ant build
```

from within the floodlight repository.

Authors

This series of labs were created by Valentin Poirot <poirotv@chalmers.se> and Emelie Ekenstedt <emeeke@student.chalmers.se>.

---

[3] https://www.wireshark.org/
[4] http://www.projectfloodlight.org/floodlight/