TÉCNICO LISBOA

AGISIT 20/21	**LAB ASSIGNMENT**	**Number:**	10
Software Defined Networking (SDN)		**Issue Date:**	07 Dec 2020
SDN Datacenter with MiniNet and OpendayLight		**Report Due:**	16 Dec 2020
Author: Prof. Rui Cruz		**Due Date:**	18 Dec 2020

1 Introduction

The objective of the experiments in this Lab assignment is to continue learning the deployment of IT Infrastructures, with a focus on network softwarization, i.e., the paradigms of **Software Defined Networking (SDN)** and Network Functions Virtualization (NFV)/Virtualized Network Functions (VNF).

For that purpose, we will make use of the network emulator Mininet [1] that will allow to create virtual network scenarios running in real-time. Additionally, we will use for SDN controller a version of the OpenDaylight (ODL) Controller [2], and also the Cisco OpenFlow Manager (OFM) [3] to play with Flows in the network.

The tools will be launched in a virtualized environment created with Vagrant [4]. The environement will be composed by three Virtual Machines, two of them equipped with analyser tools (e.g., tcpdump, Wireshark[5], etc.).

Preliminary Notes

One nice feature of the software stack we are going to use is that it is portable to many platforms including **YOUR OWN** personal computers, running the following Operating Systems:

- Microsoft Windows from version 10 up
- Apple macOS from versions 10.13 'High Sierra' up
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' up.

It is **not recommended** to apply this setup to a virtual machine (**nested virtualization**), although possible, as the configuration requires access to the Hypervisor environment (mandatory Virtualbox) in the host system.

Note: Avoid copying text strings from the examples or configurations illustrated in this document, as pasting them into your system or files may introduce/modify some characters, leading to errors or inadequate results.

[1] `http://mininet.org`
[2] `http://www.opendaylight.org/`
[3] `https://developer.cisco.com/codeexchange/github/repo/CiscoDevNet/OpenDaylight-Openflow-App/`
[4] `https://www.vagrantup.com`
[5] A web Manual of wireshark: `https://www.wireshark.org/docs/wsug_html_chunked`

AGISIT 20/21	**LAB ASSIGNMENT**	**Number:**	10
Software Defined Networking (SDN)		**Issue Date:**	07 Dec 2020
SDN Datacenter with MiniNet and OpendayLight		**Report Due:**	16 Dec 2020
Author: Prof. Rui Cruz		**Due Date:**	18 Dec 2020

2 Submitting the Results of the Experiments

The experiments that you will execute in this LAB will either produce results that you need to report or from which you will be asked questions about the execution of some procedures. In order to report the results you will achieve, proceed as follows:

2.1 General Procedure

On your system, it is advisable to **create a working folder for this lab** (not the Experiment folder!), that will contain the necessary files for reporting. These files may be screenshots, code snippets, text documents, system logs, etc.

The Experiment folder named **lab10-sdn** will be created into your **Projects folder**, **not located under the User Profile folder** (i.e., /Users/username/), by cloning the remote Repository in the **GIT** platform (`https://git.rnl.tecnico.ulisboa.pt`) by executing a command as follows (with the URL of your remote Repository):

```
$ git clone https://git.rnl.tecnico.ulisboa.pt/AGISIT-20-21/lab10-
  sdn.git
```

The procedure for submission is quite simple:

1. This Lab assignment MUST be reported up to the **Report Due** date;
2. In Moodle for the RC course (`https://moodle.dei.tecnico.ulisboa.pt/`), you will find an **Assignment** for reporting the results from this specific Lab experiment.
3. The Assignment Link opens a **TSUGI CLOUD Web Form** containing the Questions to be answered;
4. The Assignment **Due Date** corresponds to the date it MUST be considered completed;
5. When you are prepared with the requested materials (screen-shots, command line outputs, developed code, etc.) you will submit the items into the respective Exercise Form questions boxes of the Assignment;
6. In the same Exercise Form of the Assignment you may be asked to comment your submissions;
7. When finished answering the Exercise Form, click the button **Done** in the top left of the Form; You will be returned to the Assignment;
8. Please note that this process involves a **Peer Review** mechanism, in which you will be asked to provide **feedback** (anonymously) to the Reports of your classmates. So, after submitting your own assignment, get back to it to see the Reviews that the system have distributed automatically to you;

2.2 Specific Procedure

For this Lab Experiment you will need to submit the following items:

1. Submit the content of the topology script created for your Datacenter, to submit in the TSUGI Form where asked;
2. Submit the result of the command **sudo ovs-ofctl -O OpenFlow13 dump-ports s2** of your Datacenter, to submit in the TSUGI Form where asked;
3. Submit the result of the command **sudo ovs-ofctl -O OpenFlow13 dump-flows s2** of your Datacenter, to submit in the TSUGI Form where asked;
4. A screenshot of the Topology with a broken link (taken from MiniEdit), to submit in the TSUGI Form where asked;
5. A screenshot of the ODL controller Topology window, after creating the Datacenter infrastructure and playing with traffic, to submit in the TSUGI Form where asked;

WARNING. Submissions MUST BE MADE in the TSUGI CLOUD Web Form. No other type of submission will be considered.

3 Software Defined Networking (SDN)

Software Defined Networking (SDN) is a way to manage networks that separates the **control plane** from the forwarding plane (oi.e, the **data plane**). SDN offers a centralized view of the network, giving an SDN Controller the ability to act as the "brains" of the network. SDN involves an application interacting with a network (composed of domain-specific devices) for the purpose of simplifying operations or enabling a service. In our experiments, the application will be the Cisco OpenFlow Manager. In SDN, a (logical) Controller is positioned between the application and the network and interacts with network elements (e.g., switches) in the southbound direction using a variety of different protocols. In the northbound direction it presents an abstraction of the network using common Representational State Transfer (REST) APIs.

One of the most well-known protocols used by SDN Controllers to communicate with the switches/routers is **OpenFlow**[6], however, it is not the only SDN standard, as other protocols can also be used by an SDN Controller such as the YANG (Yet Another Next Generation) modelling language[7] and NETCONF (NETwork CONFiguration) protocol[8].

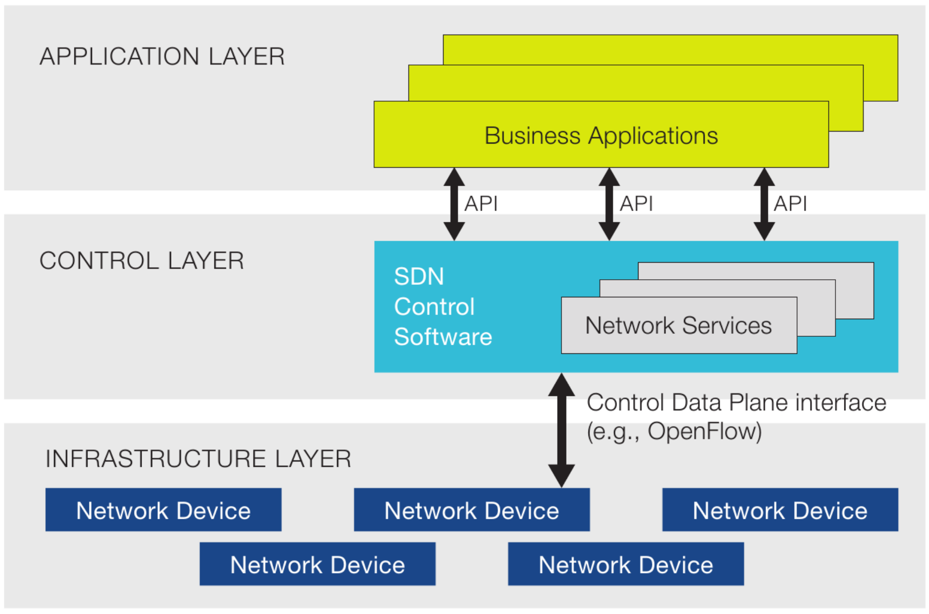


Figure 1: SDN Architecture

In the SDN architecture (Figure 1), the splitting of the control and data forwarding functions is referred to as "disaggregation", because these pieces can be sourced separately, rather than deployed as one integrated system. SDN is a complementary

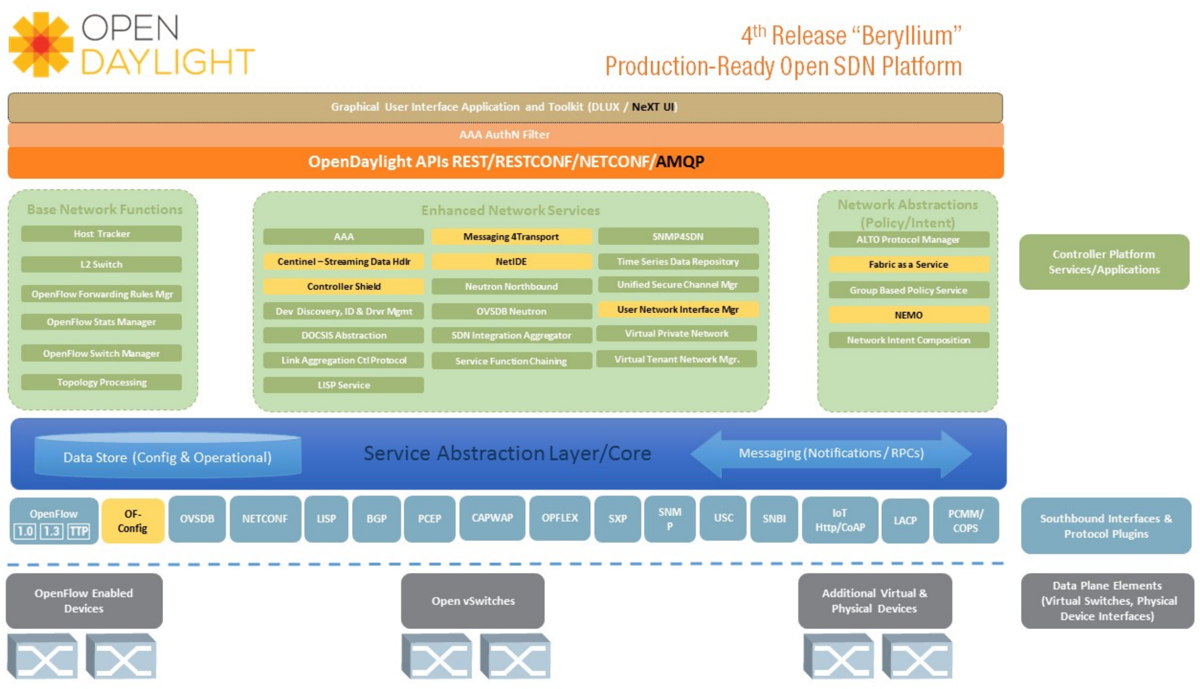
[6]Open Networking Foundation, OpenFlow Switch Specification 1.51 `https://goo.gl/A9DCvP`
[7]RFC 6020 `https://tools.ietf.org/html/rfc6020`
[8]RFC 6241 `https://tools.ietf.org/html/rfc6241`

approach to Network Functions Virtualization (NFV) for network management. While they both manage networks, both rely on different methods.

The first SDN architectures were essencially built by Google and Amazon as they were looking for new ways to build huge, "webscale" datacenters around the Globe made with commodity and open source parts, and traditional networking solutions, were not able to provide the flexibility and the global control that those organizations wanted.

4 OpenDaylight (ODL) Controller

SDN Controllers in SDN correspond to the "brains" of the network. The Controller is the application that acts as a strategic control point in the SDN network, used to manage the flow control to the switches/routers (via southbound APIs) and the applications and business logic (via northbound APIs) in order to deploy "intelligent" networks.



OPEN DAYLIGHT
4th Release "Beryllium"
Production-Ready Open SDN Platform
Graphical User Interface Application and Toolkit (DLUX / NeXT UI)
AAA AuthN Filter
OpenDaylight APIs REST/RESTCONF/NETCONF/AMQP
Base Network Functions
Host Tracker
L2 Switch
OpenFlow Forwarding Rules Mgr
OpenFlow Stats Manager
OpenFlow Switch Manager
Topology Processing
Enhanced Network Services
AAA
Messaging 4Transport
SNMP4SDN
Centinel – Streaming Data Hdlr
NetIDE
Time Series Data Repository
Controller Shield
Neutron Northbound
Unified Secure Channel Mgr
Dev Discovery, ID & Drvr Mgmt
OVSDB Neutron
User Network Interface Mgr
DOCSIS Abstraction
SDN Integration Aggregator
Virtual Private Network
Link Aggregation Ctl Protocol
Service Function Chaining
Virtual Tenant Network Mgr.
LISP Service
Network Abstractions (Policy/Intent)
ALTO Protocol Manager
Fabric as a Service
Group Based Policy Service
NEMO
Network Intent Composition
Controller Platform Services/Applications
Data Store (Config & Operational)
Service Abstraction Layer/Core
Messaging (Notifications / RPCs)
OpenFlow 1.0 1.3 TTP
OF-Config
OVSDB
NETCONF
LISP
BGP
PCEP
CAPWAP
OPFLEX
SXP
SNMP
USC
SNBI
IoT Http/CoAP
LACP
PCMM/COPS
Southbound Interfaces & Protocol Plugins
OpenFlow Enabled Devices
Open vSwitches
Additional Virtual & Physical Devices
Data Plane Elements (Virtual Switches, Physical Device Interfaces)

Figure 2: OpenDaylight Architecture

An SDN Controller platform typically contains a collection of "pluggable" modules that can perform different network tasks. Some of the basic tasks include inventorying what devices are within the network and the capabilities of each, gather network statistics, etc. Extensions can be inserted that support more advanced capabilities, such as running algorithms to perform analytics and orchestrating new rules throughout the network.

OpenDaylight (ODL), which is now a Project of the **Linux Foundation**, is a Java-based Controller supporting OpenFlow and other southbound APIs and includes critical features, such as high-availability and clustering (Figure 2).

An OpenDaylight Controller is implemented solely in software and is kept within its own Java Virtual Machine (JVM), but it can be deployed in a variety of production network environments.

5 OpenFlow Manager (OFM)

OpenFlow Manager (OFM) [9] is an application developed to run on top of OpenDaylight to visualize OpenFlow topologies, program OpenFlow paths and gather OpenFlow statistics. Figure 3 depicts the architecture of the OFM components. The application is not anymore being developed, but the functionality was integrated in Cisco commercial solution.

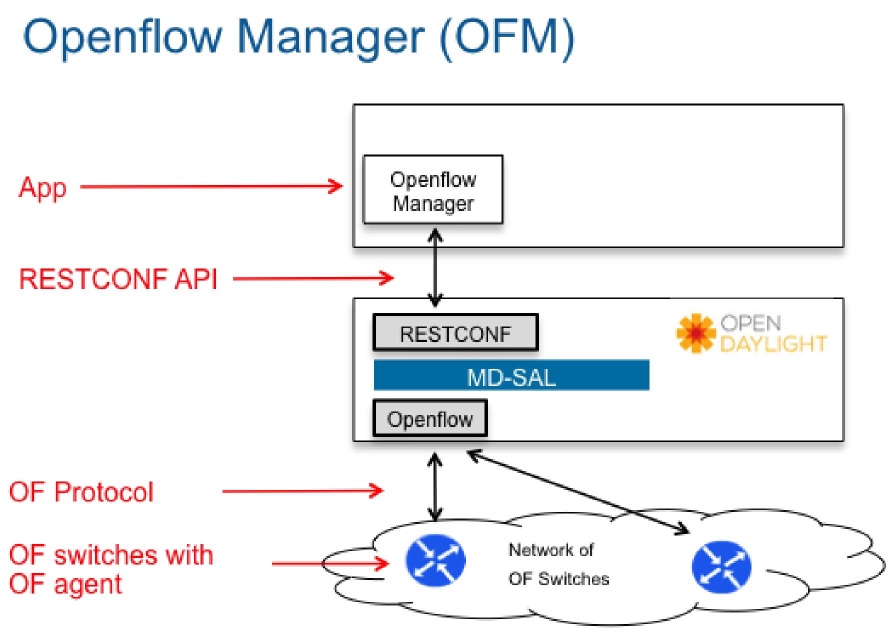


Figure 3: OpenFlow Manager architecture.

From the bottom-up of Figure 3 we have a network of OpenFlow switches employing a “Match/Action” forwarding paradigm to execute flow switching operations through the network.

[9] Cisco DevNet OFM `https://developer.cisco.com/codeexchange/github/repo/CiscoDevNet/OpenDaylight-Openflow-App/`

Instead of cumbersome per-device configuration and management, OFM provides a web browser user interface that operators use to manage OpenFlow networks and devices.

6 Mininet - SDN Network Emulator

Mininet [10] provides a virtual test bed and development environment for SDN. Mininet enables SDN development on any laptop or PC, and SDN designs can move seamlessly between Mininet, and the real hardware running at line rate in live deployments. Mininet enables:

- Rapid prototyping of software-defined networks
- Complex topology testing without the need to wire up a physical network
- to run network real code including standard Unix/Linux network applications as well as the real Linux kernel and network stack.

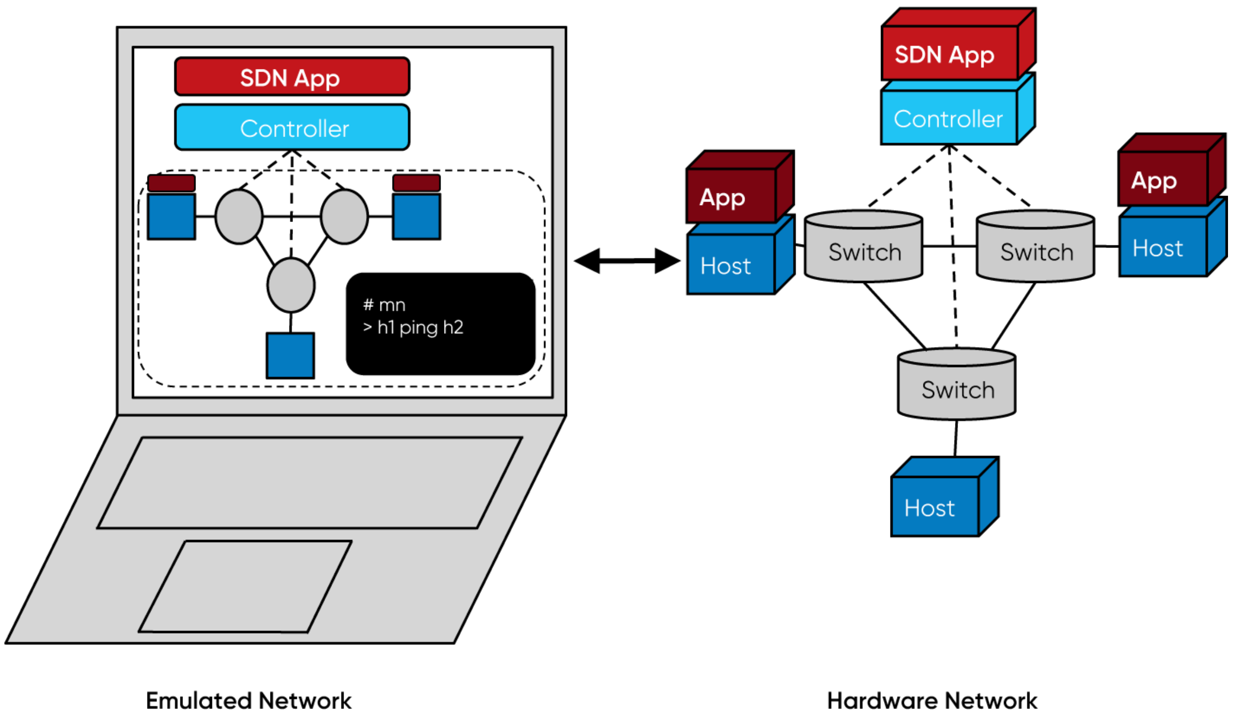


Figure 4: Mininet

Mininet provides an extensible Python API for network creation and experimentation. A Mininet network consists of:

[10]Mininet `http://mininet.org`

- **ISOLATED HOSTS:** A group of user-level processes moved into a network namespace that provide exclusive ownership of interfaces, ports and routing tables.
- **EMULATED LINKS:** Linux Traffic Control (tc) enforces the data rate of each link to shape traffic to a configured rate. Each emulated host has its own virtual Ethernet interface(s).
- **EMULATED SWITCHES:** The default Linux Bridge or the Open vSwitch running in kernel mode is used to switch packets across interfaces. Switches and routers can run in the kernel or in the user space.

7 Setting up the Experimental Environment

To start, you need to create a project directory for this Lab. On your system, be it Microfoft Windows, Linux or macOS, it is advisable to create a **Projects folder not located under the User Profile folder** (i.e., /Users/username/), that will then contain the **Lab folders** that will be managed by **Git** for versioning (i.e, will contain Repositories).

For that purpose you will clone into that **Projects folder** the remote Repository in the **GIT** platform (`https://git.rnl.tecnico.ulisboa.pt`) named **lab10-sdn** (where **XX** is you IST ID) by executing a command as follows (with the URL of your remote Repository):

```
$ git clone https://git.rnl.tecnico.ulisboa.pt/GISIT-20-21/lab10-
  sdn.git
```

The **lab10-sd** project folder should have a structure similar to the following:

```
.
|__ Vagrantfile
|__ odl-provision.sh
|__ ofm-provision.sh
|__ shared
    |__ start_mininet.sh
    |__ topos
        |__ att
        |__ chordal
        |__ fattree
        |__ geant
```

For Microsoft Windows hosts, you need to start “**VcXsrv**” before spin off the lab environment. The “**VcXsrv**” application will start a **X-Windows** server that will forward application GUIs running inside the VM to the host system.

Verify also that you have an environment Variable with name **DISPLAY** and with **Variable value** of `localhost:0.0`.

AGISIT 20/21	**LAB ASSIGNMENT**	**Number:**	10
Software Defined Networking (SDN)		**Issue Date:**	07 Dec 2020
SDN Datacenter with MiniNet and OpendayLight		**Report Due:**	16 Dec 2020
Author: Prof. Rui Cruz		**Due Date:**	18 Dec 2020

8 SDN Experimental Environment

Before starting the environment, it is recommended to inspect and interpret the **Vagrantfile** for the experiment, confirming that the addressing scheme to be used in the three VMs that will be launched does not collide with addresses alredy in use by your host system.

If everything seems adequate you need to ensure that you have already started a X11 server. In Windows run **VcXsrv**, in macOS run **XQuartz**, in Linux X11 is normally started with the system. After confirming that you have the X11 server listening, you can then boot the environment with the command:

```
:~$ vagrant up
```

Please note that the OpenDaylight Controler VM will take several minutes (perhaps more than 15 minutes, depending on the speed of downloading the necessary software) to be ready, and also to load the configurations and the Features of the application.

When ready, establish three separate Terminal sessions using the following commands:

```
:~$ vagrant ssh mininet
:~$ vagrant ssh odl
:~$ vagrant ssh ofm
```

Starting with the OpenDaylight Controller—and if you need to halt the VM, for example, to continue later the experiments—you should enter the following command immediately after entering the SSH session in order to ensure that the controller server is stopped.

```
vagrant@odl:~$ sudo ./distribution-karaf-0.6.4-Carbon/bin/stop
```

When the Bash prompt returns you can enter the following command to start the service in interactive mode:

```
vagrant@odl:~$ sudo ./distribution-karaf-0.6.4-Carbon/bin/karaf
```

That command will start the interactive controller interface, resulting in something similar to Figure 5.

Now go the OpenFlow Manager (OFM) Terminal session and enter the following commands:

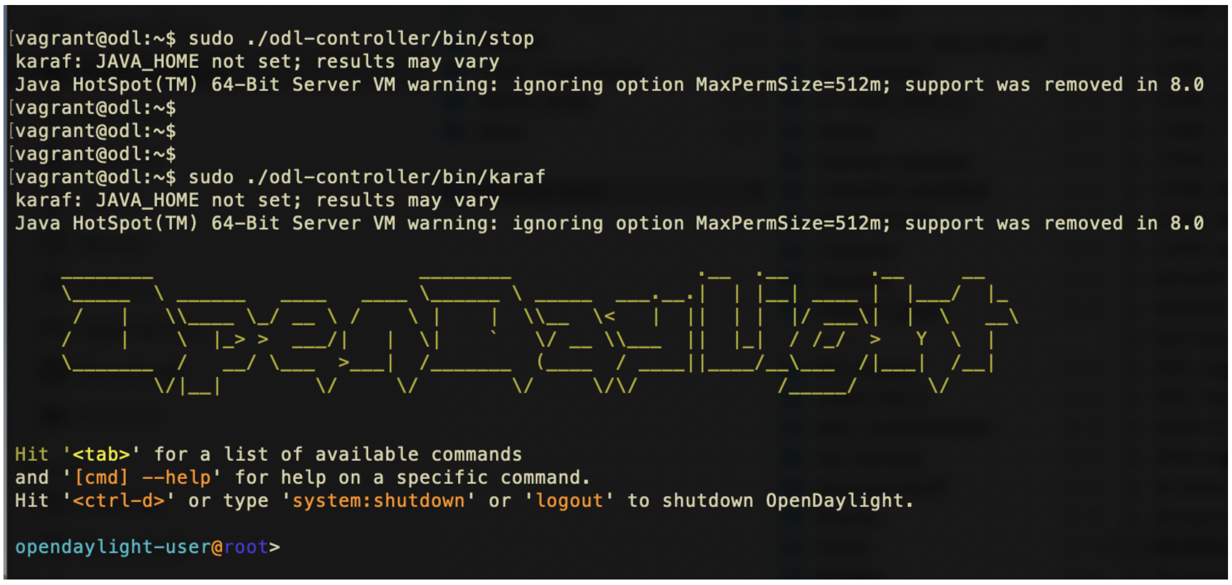
```
vagrant@ofm:~$ cd OpenDaylight-Openflow-App/

vagrant@ofm:~/OpenDaylight-Openflow-App$ grunt
```

Those commands will give as result a web server in a running state (waiting...):

With a Browser in your host system, open two windows (or two tabs), one with the URL `http://10.10.10.2:8181/index.html` for the ODL Controller, and another with the

AGISIT 20/21	**LAB ASSIGNMENT**	**Number:**	10
Software Defined Networking (SDN)		**Issue Date:**	07 Dec 2020
SDN Datacenter with MiniNet and OpendayLight		**Report Due:**	16 Dec 2020
Author: Prof. Rui Cruz		**Due Date:**	18 Dec 2020



```
[vagrant@odl:~$ sudo ./odl-controller/bin/stop
 karaf: JAVA_HOME not set; results may vary
 Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512m; support was removed in 8.0
[vagrant@odl:~$
[vagrant@odl:~$
[vagrant@odl:~$
[vagrant@odl:~$ sudo ./odl-controller/bin/karaf
 karaf: JAVA_HOME not set; results may vary
 Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512m; support was removed in 8.0

    ________                       ________                .__  .__       .__     __
    \_____  \ ______   ____   ____ \______ \ _____  ___.__.|  | |__| ____ |  |___/  |_
     /   |   \\____ \_/ __ \ /    \ |    |  \\__  \<   |  ||  | |  |/ ___\|  |  \   __\
    /    |    \  |_> >  ___/|   |  \|    `   \/ __ \\___  ||  |_|  / /_/  >   Y  \  |
    \_______  /   __/ \___  >___|  /_______  (____  / ____||____/__\___  /|___|  /__|
            \/|__|        \/     \/        \/     \/\/            /_____/      \/


 Hit '<tab>' for a list of available commands
 and '[cmd] --help' for help on a specific command.
 Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

 opendaylight-user@root>
```

Figure 5: The OpenDaylight Interactive interface.

URL `http://10.10.10.4:9000` for the OFM application.

The ODL Controller will present a Login window as in Figure 6. Use **admin/admin** as username and password.

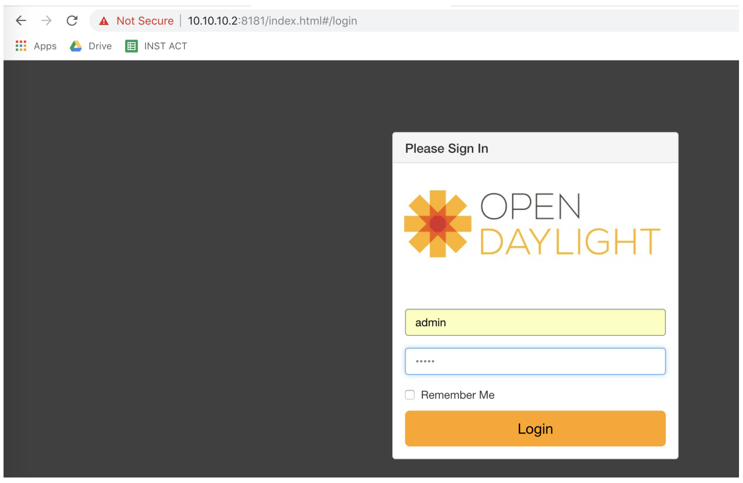


Figure 6: OpenDaylight Login window

8.1 Testing the environment

In order to test if all is running go to the Mininet Terminal session and enter the following command (this is a single command line!):

```
vagrant@mininet:~$ sudo mn --topo=linear,3 --controller=remote,ip
   =10.10.10.2,port=6653 --switch ovsk,protocols=OpenFlow13
```

What that command means is:

- `--topo=linear,3` : create a linear topology with 3 switches
- `--controller=remote,ip=10.10.10.2,port=6653` : the SDN controler is remote with specific IP address and port
- `--switch ovsk,protocols=OpenFlow13` : the switches are of type OpenVSwitch and should run OpenFlow protocol version 1.3

```
vagrant@mininet:~$ sudo mn --topo=linear,3 --controller=remote,ip
   =10.10.10.2,port=6653 --switch ovsk,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet>
```

At the Mininet prompt, enter a **pingall**) command. You will observe results similar to these, i.e., each host (h1, h2, h3) sent successfully pings to all the other hosts (6/6 received):

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Looking now at the ODL browser window, and refreshing the Topology, you should observe the network as illustrated in Figure 7, and similarly in OFM as illustrated in Figure 8.

AGISIT 20/21	**LAB ASSIGNMENT**	**Number:**	10
Software Defined Networking (SDN)		**Issue Date:**	07 Dec 2020
SDN Datacenter with MiniNet and OpendayLight		**Report Due:**	16 Dec 2020
Author: Prof. Rui Cruz		**Due Date:**	18 Dec 2020

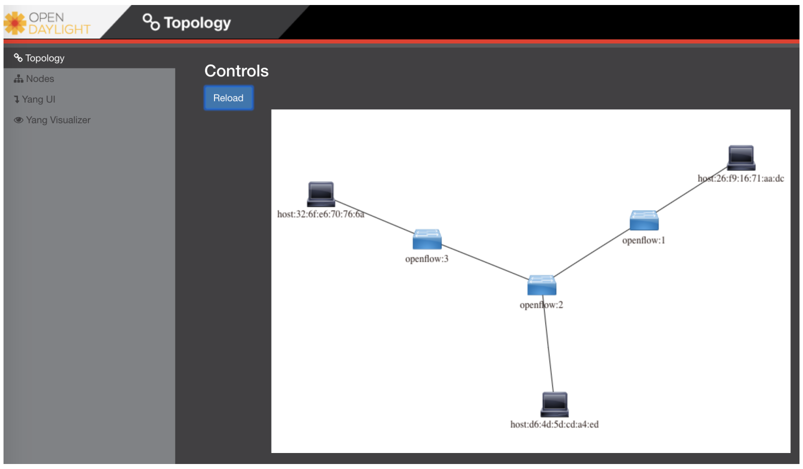


Figure 7: OpenDaylight Topology

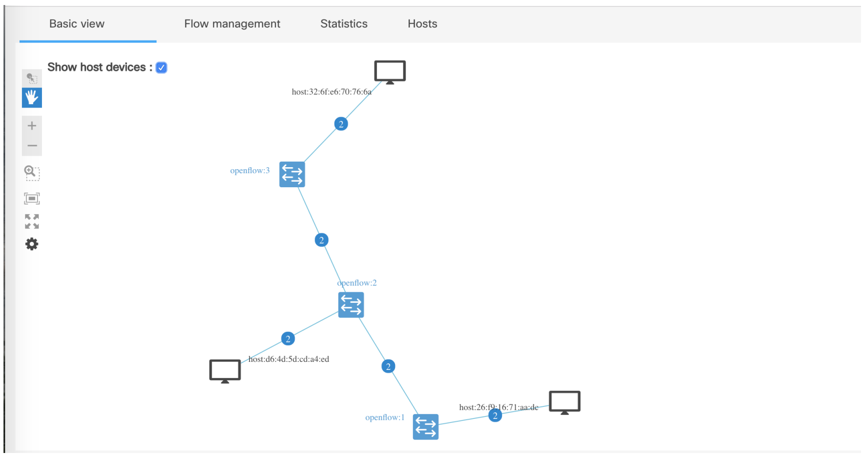
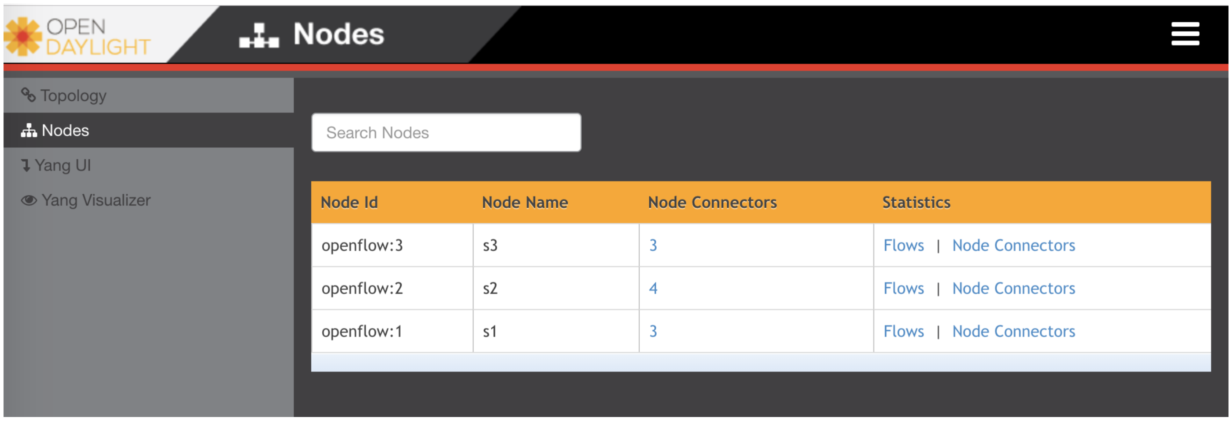


Figure 8: OpenFlow Manager Basic view

Open now a second Terminal session with Mininet, and enter the command **sudo ovs-vsctl show**, which will display that three Bridges (s1, s2 and s3) have been created, all talking to the ODL Controller at port 6653 (your output should be similar to the one that follows for the first two Bridges):

```
vagrant@mininet:~$ sudo ovs-vsctl show
c83f1e43-0dce-40d3-bbaa-d799bb716d98
    Bridge "s2"
        Controller "ptcp:6655"
        Controller "tcp:10.10.10.2:6653"
            is_connected: true
        fail_mode: secure
        Port "s2-eth2"
            Interface "s2-eth2"
        Port "s2-eth1"
            Interface "s2-eth1"
        Port "s2"
            Interface "s2"
                type: internal
        Port "s2-eth3"
            Interface "s2-eth3"
    Bridge "s1"
        Controller "ptcp:6654"
        Controller "tcp:10.10.10.2:6653"
            is_connected: true
        fail_mode: secure
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1"
            Interface "s1"
                type: internal
```

Bridges `s1`, `s3` have three “physical interfaces”, and Bridge `s2` has four interfaces, named for example in the first Bridge, `"s1-eth1"`, `"s1-eth2"`, and `"s1"` of type internal, which is a local management interface.

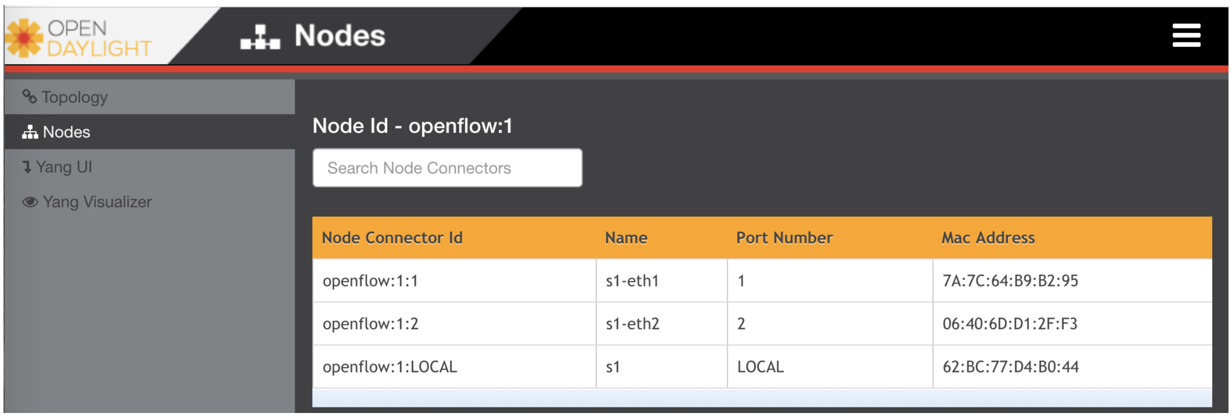


Node Id	Node Name	Node Connectors	Statistics
openflow:3	s3	3	Flows | Node Connectors
openflow:2	s2	4	Flows | Node Connectors
openflow:1	s1	3	Flows | Node Connectors

Figure 9: OpenDaylight Nodes

Looking now in the ODL window, selecting “Nodes” view (Figure 9), you can observe that Switch 1 has three “node connectors”, and if you click on the number of connectors for that Switch, you can observe in the table that opens (Figure 10) that one is the LOCAL management interface.

To observe the traffic that occurred in Switch 1, use the following two commands:



Node Id - openflow:1

Node Connector Id	Name	Port Number	Mac Address
openflow:1:1	s1-eth1	1	7A:7C:64:B9:B2:95
openflow:1:2	s1-eth2	2	06:40:6D:D1:2F:F3
openflow:1:LOCAL	s1	LOCAL	62:BC:77:D4:B0:44

Figure 10: OpenDaylight Node Connectors

```
sudo ovs-ofctl -O OpenFlow13 dump-ports s1

sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

The first command provides results similar to the following, showing the traffic in the ports of Switch 1:

```
vagrant@mininet:~$ sudo ovs-ofctl -O OpenFlow13 dump-ports s1
OFPST_PORT reply (OF1.3) (xid=0x2): 3 ports
  port  1: rx pkts=16, bytes=1112, drop=0, errs=0, frame=0, over
     =0, crc=0
           tx pkts=699, bytes=59115, drop=0, errs=0, coll=0
           duration=3394.211s
  port  2: rx pkts=705, bytes=59603, drop=0, errs=0, frame=0, over
     =0, crc=0
           tx pkts=688, bytes=58345, drop=0, errs=0, coll=0
           duration=3394.211s
  port LOCAL: rx pkts=8, bytes=648, drop=0, errs=0, frame=0, over
     =0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
           duration=3394.188s
```

The second command provides results similar to the following, showing the Flows installed in Switch 1:

```
vagrant@mininet:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x2b00000000000005, duration=4166.198s, table=0, n_packets
    =11, n_bytes=742, priority=2,in_port=1 actions=output:2,
    CONTROLLER:65535
```

```
cookie=0x2b00000000000006, duration=4166.197s, table=0, n_packets
    =21, n_bytes=1470, priority=2,in_port=2 actions=output:1
cookie=0x2b00000000000000, duration=4172.033s, table=0, n_packets
    =835, n_bytes=70975, priority=100,dl_type=0x88cc actions=
    CONTROLLER:65535
cookie=0x2b00000000000007, duration=4172.033s, table=0, n_packets
    =14, n_bytes=1112, priority=0 actions=drop
```

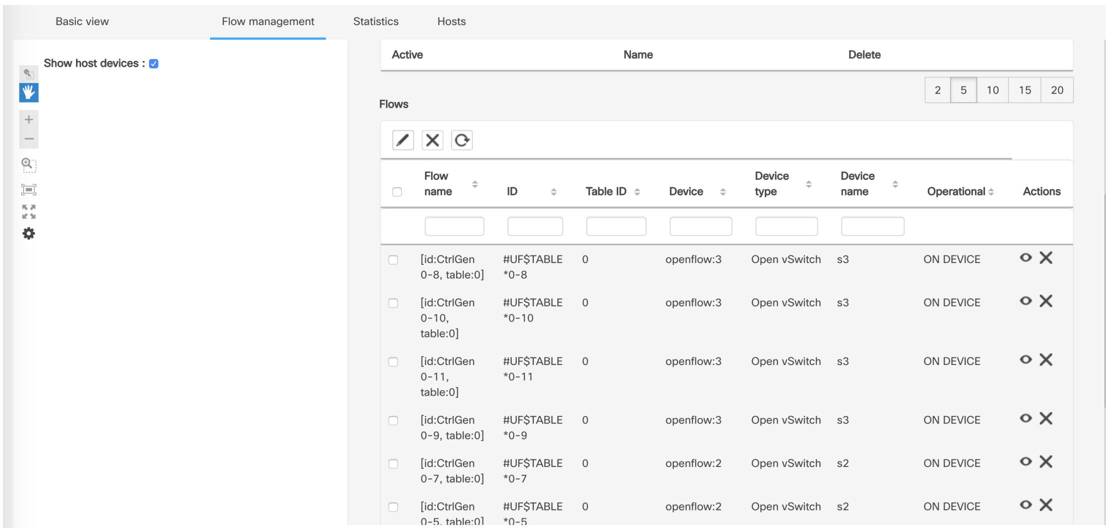
You can observe that the controller has installed four Flow entries in **flow table 0**.

- The first entry says that traffic received in **port 1** is forwarded (actions) to **port 2** and also to the controller.
- The second entry says that any traffic received in **port 2** is sent to **port 1**.
- The third entry says that if the traffic type is `0x88cc` it is sent to the controller. But notice that the priority for this flow entry is the highest (100), meaning that it is the first flow entry to be processed. The Ethernet type `0x88cc` correspond to the Link Layer Discovery Protocol (LLDP), meaning that this is the method for the controller to discover how switches are connected to each other, by injecting LLDP messages to the network in order to discover the topology.
- The fourth entry, with the lowest priority (0), says that any other traffic not matching a flow entry is dropped.

In order to inspect and create flows in real-time, we will use the OFM application. First, in the Mininet prompt, start a ping from **host 3** (in switch 3) to **host 1** (in switch 1) with the command:

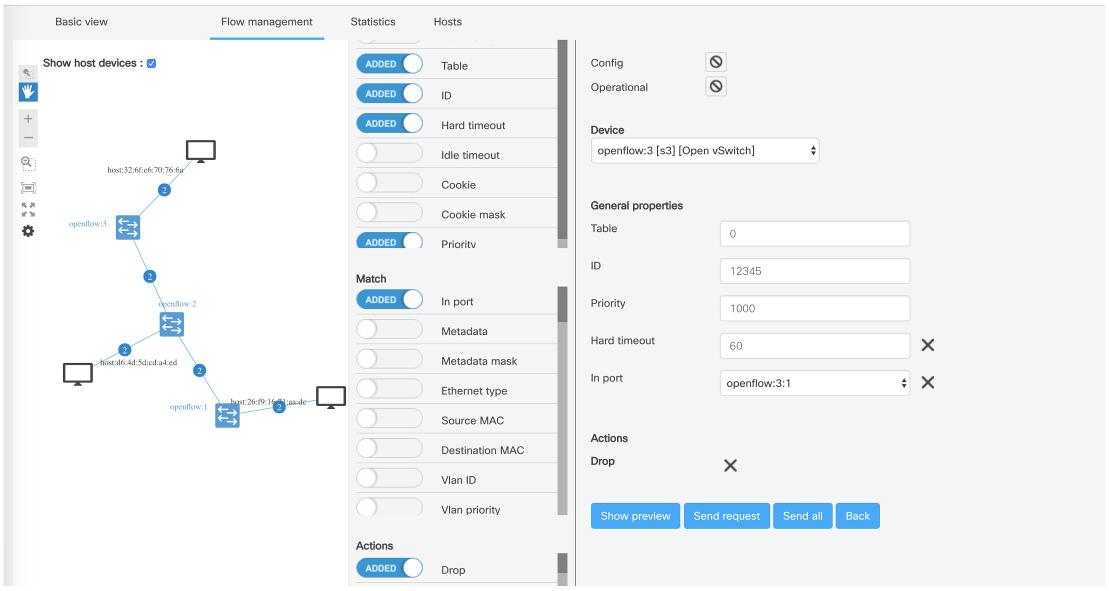
```
mininet> h3 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.213 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.274 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.080 ms
.....
```

While the ping is generating traffic, let's in OFM inspect the installed Flows (Figure 11) and create a new flow, as illustrated in Figure 12 (select the "pencil" icon).



	Flow name	ID	Table ID	Device	Device type	Device name	Operational	Actions
	[id:CtrlGen 0-8, table:0]	#UF$TABLE *0-8	0	openflow:3	Open vSwitch	s3	ON DEVICE	
	[id:CtrlGen 0-10, table:0]	#UF$TABLE *0-10	0	openflow:3	Open vSwitch	s3	ON DEVICE	
	[id:CtrlGen 0-11, table:0]	#UF$TABLE *0-11	0	openflow:3	Open vSwitch	s3	ON DEVICE	
	[id:CtrlGen 0-9, table:0]	#UF$TABLE *0-9	0	openflow:3	Open vSwitch	s3	ON DEVICE	
	[id:CtrlGen 0-7, table:0]	#UF$TABLE *0-7	0	openflow:2	Open vSwitch	s2	ON DEVICE	
	[id:CtrlGen 0-5, table:0]	#UF$TABLE *0-5	0	openflow:2	Open vSwitch	s2	ON DEVICE	

Figure 11: OFM Flow Management



ADDED Table
ADDED ID
ADDED Hard timeout
Idle timeout
Cookie
Cookie mask
ADDED Priority

Match
ADDED In port
Metadata
Metadata mask
Ethernet type
Source MAC
Destination MAC
Vlan ID
Vlan priority

Actions
ADDED Drop

Config
Operational

Device
openflow:3 [s3] [Open vSwitch]

General properties
Table: 0
ID: 12345
Priority: 1000
Hard timeout: 60
In port: openflow:3:1

Actions
Drop

Figure 12: OFM Create and install Flow

In the **Flow create** window (Figure 12) on the right pane, select the **Device** for value openflow:3 [s3], insert for **Table** the value **0**, insert flow **ID** of 12345, insert **Priority** of 1000 and a **Hard timeout** of 60 (seconds), then, on the left pane select **Match** of **In port** and on the right pane selecting openflow:3:1, and then similarly activate **Action** on the left and select on the right the **Drop**.

Before installing the flow, observe that the ping is still succeeding, but as soon as the flow is installed, by clicking on button **Send request**, the ping stops, but the increment of the sequence starts displaying “Destination Host Unreachable”.

Observe the Flow table entry in OFM Flow management window (press refresh). When the hard timeout is reached the Flow entry is automatically removed from the Flow Table of the switch, and as you may then observe, the ping resumes.

8.2 Preparing the environment for Lab experiments

In order to ensure a clean and stable environment to start new experiments, proceed as follows:

1. In the Terminal session of Mininet, type **exit** to shutdown the emulated test network;
2. In that Terminal session use the command **sudo mn -c** in order to clean any remaining cached date from the previous emulated test network.

9 A Data Center Network

In this experiment we will use a GUI for Mininet, called **MiniEdit** (written in Python). The MiniEdit script is located in Mininet's *examples* folder. Mininet needs to run with root privileges so we start Miniedit using the sudo command:

```
$ sudo ~/mininet/examples/miniedit.py
```

If your environment is configured properly, the Miniedit Window will show up to your host. Miniedit has a simple user interface that presents a canvas with a vertical toolbar with icons on the left side of the window, and a menu bar along the top of the window as shown in Figure 13.

The icons represent the following tools (top to bottom):

- The ***Select*** tool (pointer) is used to move nodes around on the canvas.
- The ***Host*** tool creates nodes on the canvas that will perform the function of host computers.
- The ***Switch*** tool creates OpenFlow-enabled switches on the canvas. These switches are expected to be connected to a controller.
- The ***Legacy Switch*** tool creates a learning Ethernet switch with default settings. The switch will operate independently, without a controller. The legacy switch cannot be configured and is set up with Spanning Tree disabled, so do not connect legacy switches in a loop.
- The ***Legacy Router*** tool creates a basic router that will operate independently, without a controller. It is basically just a host with IP Forwarding enabled. The legacy router cannot be configured from the MiniEdit GUI.

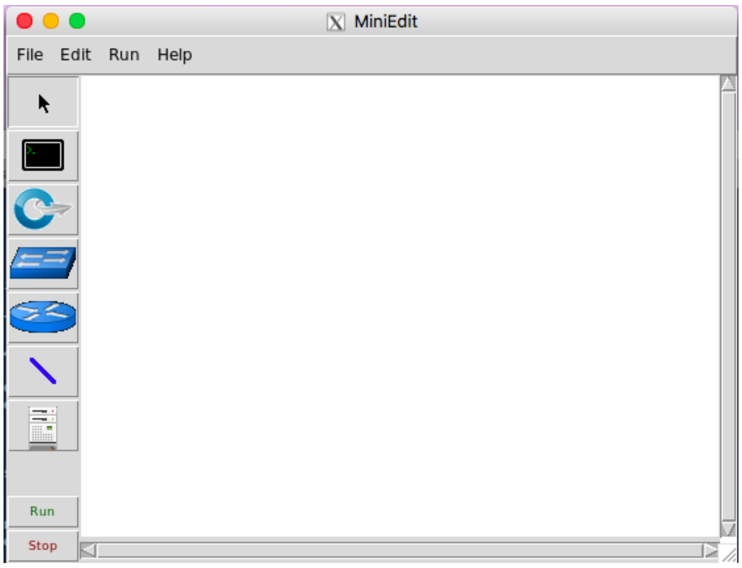


Figure 13: Miniedit graphic interface.

- The ***NetLink*** tool creates links between nodes on the canvas.
- The ***Controller*** tool creates a controller. By default, the MiniEdit creates a mininet OpenFlow reference controller, which implements the behavior of a learning switch.
- The ***Run*** starts Mininet simulation scenario currently displayed in the MiniEdit canvas. The ***Stop*** button stops it. When MininEdit simulation is in the "Run" state, right-clicking on network elements reveals operational functions such as opening a terminal window, viewing switch configuration, or setting the status of a link to "up" or "down".

9.1 Create a simple Data Center Network

Now that you got familiarized with SDN, you will try a more complex infrastructure, such as the network of a small Data Center. To create a simple Data Center network, you will add eight (8) hosts to the network scenario. Click on the ***Host*** icon, then move the pointer to the location on the Miniedit canvas where you want the host to appear, then click again. A host icon will appear on the canvas. As long as the ***Host*** tool is active, you can add more hosts. Keep clicking at each spot on the canvas where you want a host to appear. Add six (6) OpenFlow-enabled ***Switches*** and one (1) ***Controllers*** using the same method, as illustrated in Figure 14. **Pay attention to the position and name of the devices.**

Next, add links between the nodes as exemplified in Figure 14. Click on the ***NetLink*** tool, then click on a node and drag the link over to another node. Connect every host to

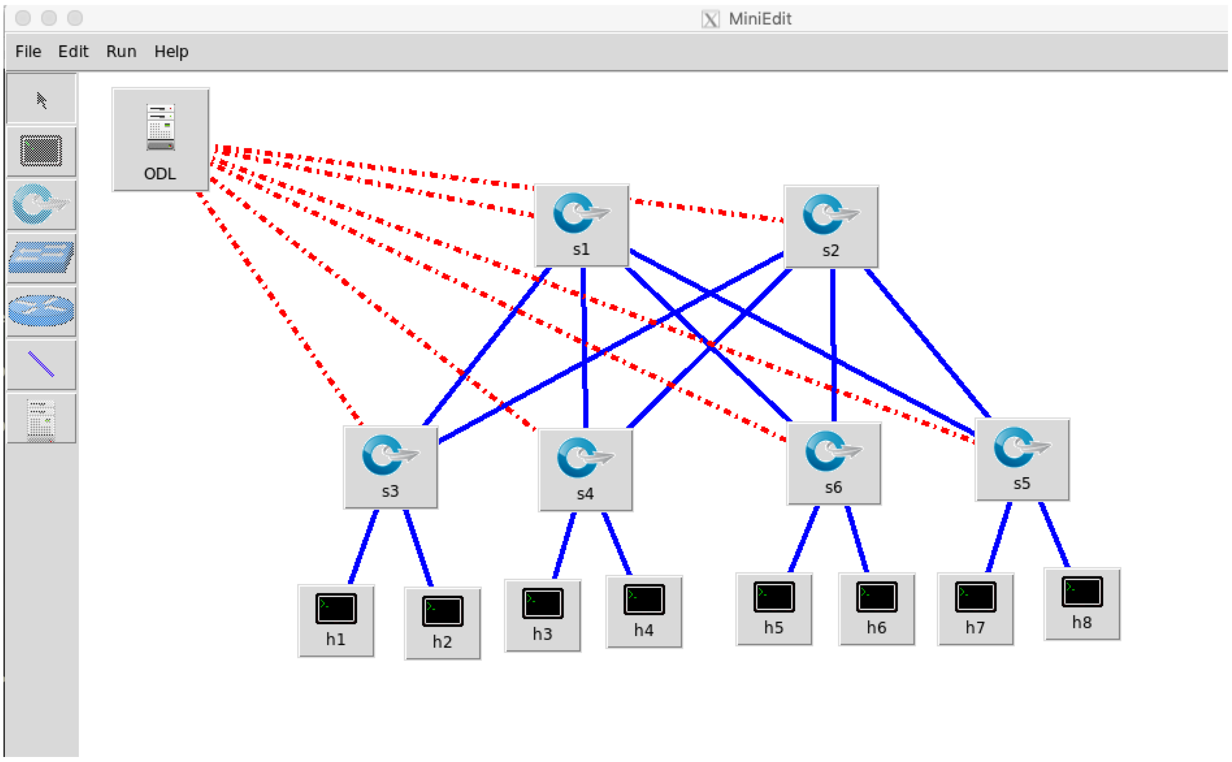


Figure 14: Example of network topology - adding devices.

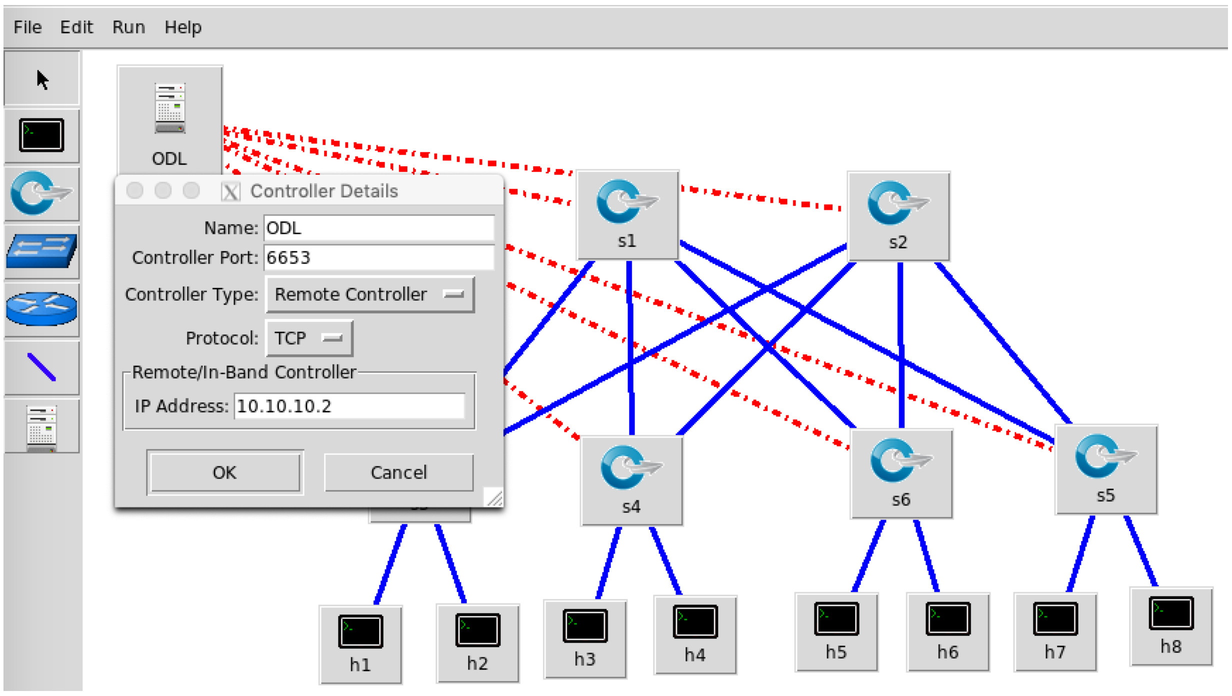
at least one switch. Connect the switches together to create a network. Then, connect each switch to the controller.

9.1.1 Configure Controller and Preferences

Right-click on the controller icon and select Properties from the menu that appears (Figure 15). The default port number for the ODL controller is 6653 and the IP address is 10.10.10.2.

To set Miniedit preferences (Figure 16), use the Miniedit menu command, *Edit* → *Preferences*. By default, the Miniedit console window does not give the user access to the Mininet Command Line Interface (CLI). To be able to use the Mininet CLI when a simulation is running, check the **Start CLI** box. Also check the Open vSwitch OpenFlow version to 1.3.

Now you should have a software-defined network (SDN) scenario allowing each host to communicate with any other host in the network. Save the Mininet Topology (*.mn) file as **datacenter-topology.mn** (in **shared** folder, in order to be available at the shared folder in the host system) so you can load this scenario into Miniedit in the future. Also, export a Mininet Custom Topology (*.py) Python script file as **dacenter-topology-script.py** so that you can use to run the scenario in a terminal window (without Miniedit).



Controller Details

Name: ODL
Controller Port: 6653
Controller Type: Remote Controller
Protocol: TCP
Remote/In-Band Controller
IP Address: 10.10.10.2

Figure 15: Configuring the controller in Miniedit.

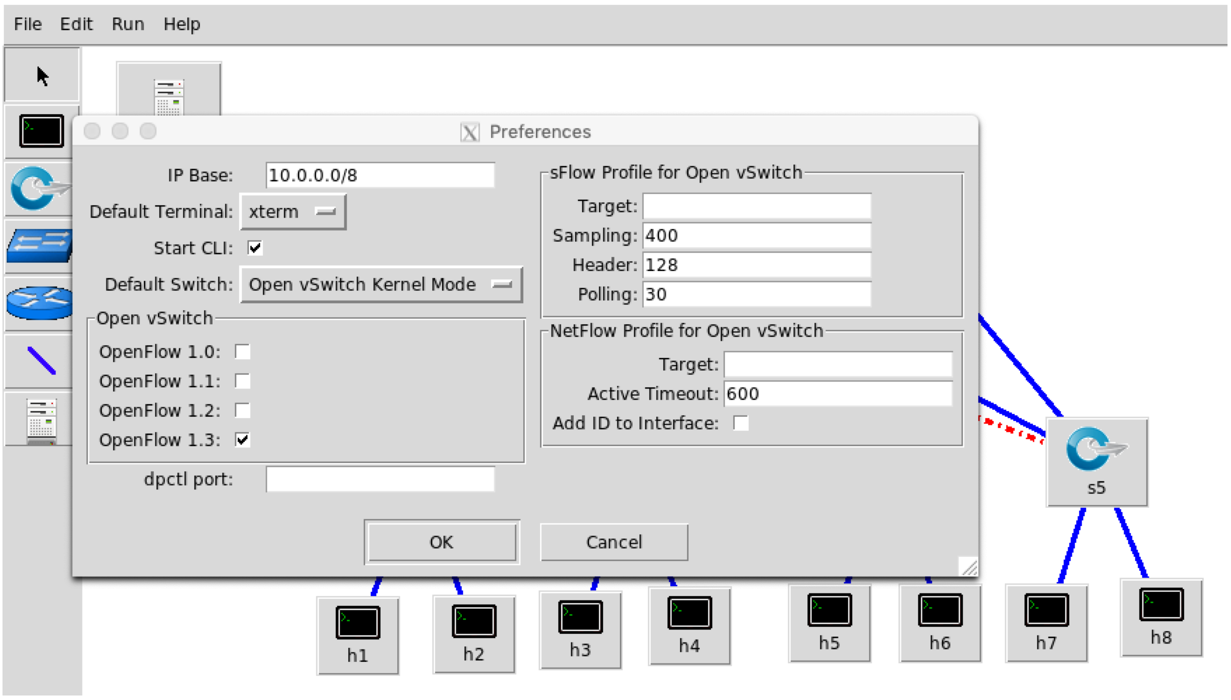
9.1.2 Run the Data Center network scenario

To start the simulation scenario, click the Run button on the Miniedit GUI. In the terminal window from which you started Miniedit, you will see some messages showing the progress of the simulation startup and then the Miniedit CLI prompt (because you checked the ***Start CLI*** box in the Miniedit preferences window). Pay attention to the **warning message** that is displayed after the network has been started. Ensure you quit from the CLI by typing ***exit*** at the Mininet prompt before clicking the emulation **Stop** button.

9.1.3 Some Experiments with the network

After starting the simulation scenario, you will view the status of different elements in the network, and will be capable of opening terminal session in the hosts, run network traffic, run programs on simulated hosts, and simulate network failures.

First, check the switch configurations in the network to verify that everything is set up correctly. You can run the Miniedit menu command *Run* → *Show OVS Summary* to see a listing of switch configurations. In this case, you can verify if each switch is listening to the correct controller on the correct port. An example of the output can be seen in Figure 17.



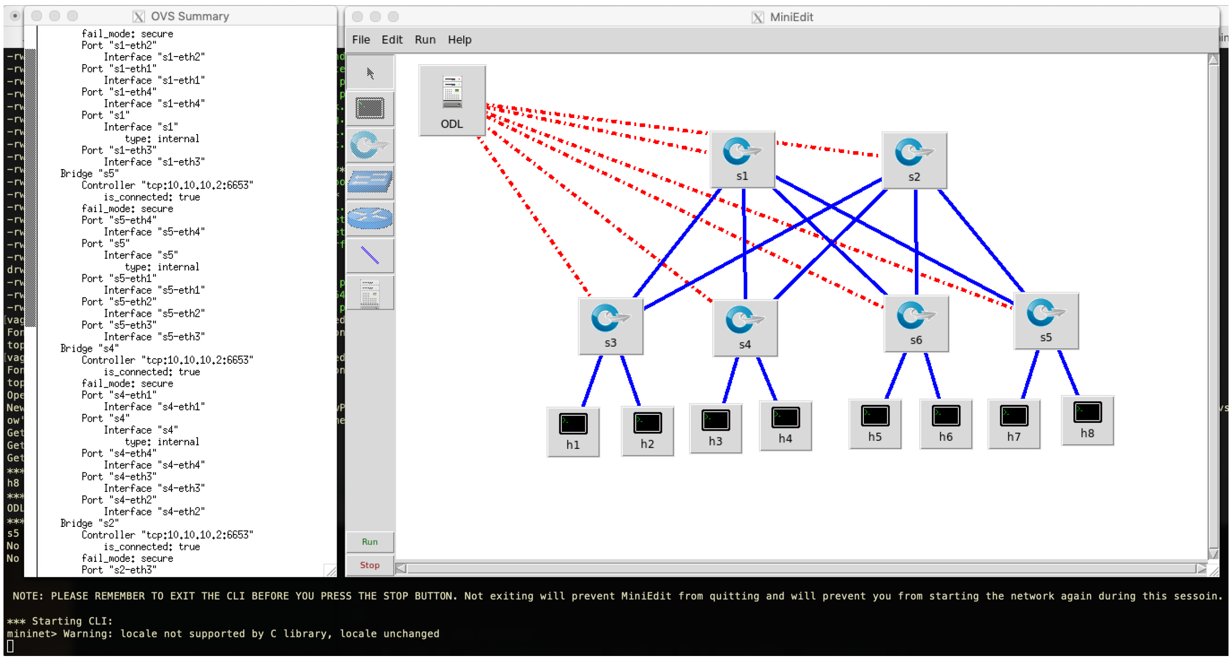
IP Base: 10.0.0.0/8
Default Terminal: xterm
Start CLI: ✓
Default Switch: Open vSwitch Kernel Mode

Open vSwitch
OpenFlow 1.0: ☐
OpenFlow 1.1: ☐
OpenFlow 1.2: ☐
OpenFlow 1.3: ✓
dpctl port:

sFlow Profile for Open vSwitch
Target:
Sampling: 400
Header: 128
Polling: 30

NetFlow Profile for Open vSwitch
Target:
Active Timeout: 600
Add ID to Interface: ☐

Figure 16: Miniedit preferences.



```
        fail_mode: secure
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1-eth4"
            Interface "s1-eth4"
        Port "s1"
            Interface "s1"
                type: internal
        Port "s1-eth3"
            Interface "s1-eth3"
    Bridge "s5"
        Controller "tcp:10.10.10.2:6653"
            is_connected: true
        fail_mode: secure
        Port "s5-eth4"
            Interface "s5-eth4"
        Port "s5"
            Interface "s5"
                type: internal
        Port "s5-eth1"
            Interface "s5-eth1"
        Port "s5-eth2"
            Interface "s5-eth2"
        Port "s5-eth3"
            Interface "s5-eth3"
    Bridge "s4"
        Controller "tcp:10.10.10.2:6653"
            is_connected: true
        fail_mode: secure
        Port "s4-eth1"
            Interface "s4-eth1"
        Port "s4"
            Interface "s4"
                type: internal
        Port "s4-eth4"
            Interface "s4-eth4"
        Port "s4-eth3"
            Interface "s4-eth3"
        Port "s4-eth2"
            Interface "s4-eth2"
    Bridge "s2"
        Controller "tcp:10.10.10.2:6653"
            is_connected: true
        fail_mode: secure
        Port "s2-eth3"
```

```
NOTE: PLEASE REMEMBER TO EXIT THE CLI BEFORE YOU PRESS THE STOP BUTTON. Not exiting will prevent MiniEdit from quitting and will prevent you from starting the network again during this sessoin.

*** Starting CLI:
mininet> Warning: locale not supported by C library, locale unchanged
```

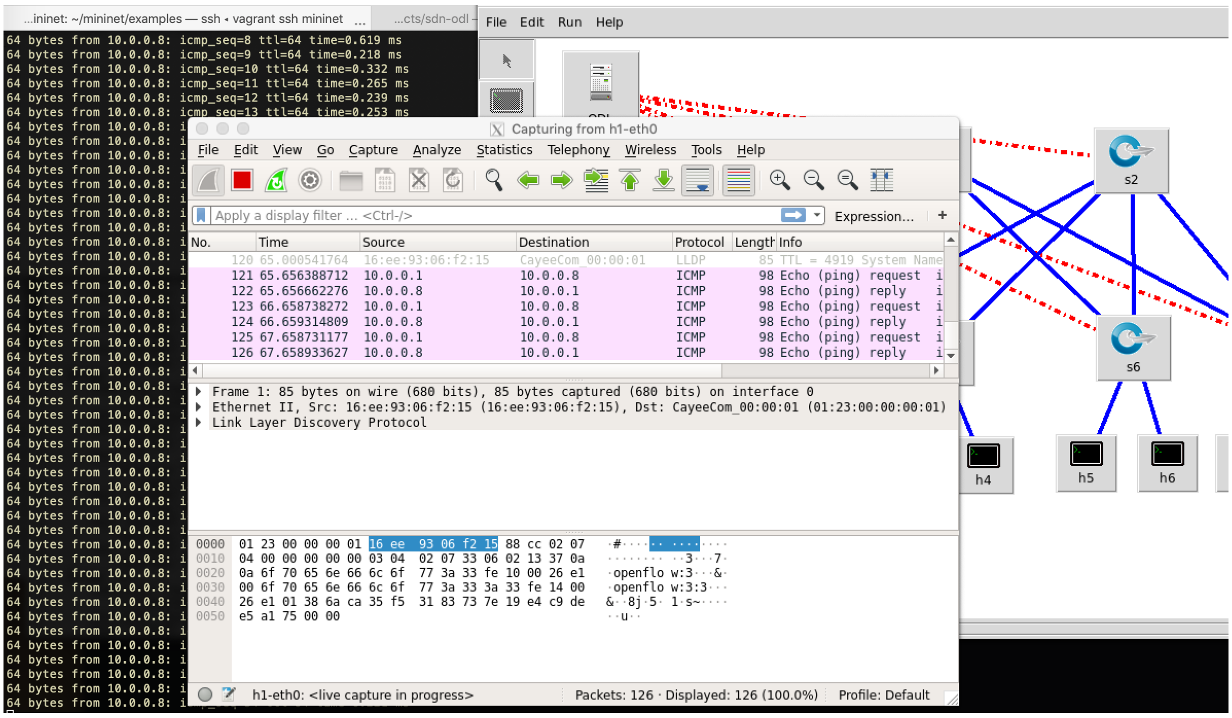
Figure 17: Show OVS Summary.

You can open a console terminal with ***xterm*** of any host of the emulated network, by clicking the host with the right mouse button and selecting Terminal, or from the Mininet prompt with the command **xterm h1**, for example to open the console of host **h1**. From the Miniedit menu command, you can also open a root Terminal with *Run* → *Root Terminal*.

9.1.4 Generate and monitor traffic in the network

Considering that you replicated a network with the topology suggested in Figure 14, open a *xterm* window on hosts **h1** (connected to switch **s3**) and **h8** (connected to switch **s5**), i.e., the hosts in opposite sides of the network. In the **h1** *xterm* window, start Wireshark with the command `wireshark &`. Next, open the menu *Capture* → *Interfaces*. Mark the **h1-eth0** network interface checkbox and then click on the **start** capture button.

In the Wireshark window you will see ICMP packets successfully sent and responses received similar to Figure 18.



...ininet: ~/mininet/examples — ssh ◂ vagrant ssh mininet ...
...cts/sdn-odl
File Edit Run Help
Capturing from h1-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
s2
s6
h4
h5
h6

```
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=0.619 ms
64 bytes from 10.0.0.8: icmp_seq=9 ttl=64 time=0.218 ms
64 bytes from 10.0.0.8: icmp_seq=10 ttl=64 time=0.332 ms
64 bytes from 10.0.0.8: icmp_seq=11 ttl=64 time=0.265 ms
64 bytes from 10.0.0.8: icmp_seq=12 ttl=64 time=0.239 ms
64 bytes from 10.0.0.8: icmp_seq=13 ttl=64 time=0.253 ms
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
64 bytes from 10.0.0.8: i
```

No.	Time	Source	Destination	Protocol	Length	Info
120	65.000541764	16:ee:93:06:f2:15	CayeeCom_00:00:01	LLDP	85	TTL = 4919 System Name
121	65.656388712	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request i
122	65.656662276	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply i
123	66.658738272	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request i
124	66.659314809	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply i
125	67.658731177	10.0.0.1	10.0.0.8	ICMP	98	Echo (ping) request i
126	67.658933627	10.0.0.8	10.0.0.1	ICMP	98	Echo (ping) reply i

```
▸ Frame 1: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface 0
▸ Ethernet II, Src: 16:ee:93:06:f2:15 (16:ee:93:06:f2:15), Dst: CayeeCom_00:00:01 (01:23:00:00:00:01)
▸ Link Layer Discovery Protocol
```

```
0000  01 23 00 00 00 01 16 ee  93 06 f2 15 88 cc 02 07   ·#······ ········
0010  04 00 00 00 00 00 03 04  02 07 33 06 02 13 37 0a   ········ ··3···7·
0020  0a 6f 70 65 6e 66 6c 6f  77 3a 33 fe 10 00 26 e1   ·openflo w:3···&·
0030  00 6f 70 65 6e 66 6c 6f  77 3a 33 3a 33 fe 14 00   ·openflo w:3:3···
0040  26 e1 01 38 6a ca 35 f5  31 83 73 7e 19 e4 c9 de   &··8j·5· 1·s~····
0050  e5 a1 75 00 00                                     ··u··
```

h1-eth0: <live capture in progress> Packets: 126 · Displayed: 126 (100.0%) Profile: Default

Figure 18: Wireshark with MiniEdit.

Next, in the **h8** *xterm* window start a ping to **h1** (which has address 10.0.0.1):

```
root@mininet:~# ping 10.0.0.1
```

Repeat the procedure for creating traffic, between **h7** (connected to switch **s5**) and **h4** (connected to switch **s4**) and start a ping from **h7** to **h4**:

AGISIT 20/21	**LAB ASSIGNMENT**	**Number:**	10
Software Defined Networking (SDN)		**Issue Date:**	07 Dec 2020
SDN Datacenter with MiniNet and OpendayLight		**Report Due:**	16 Dec 2020
Author: Prof. Rui Cruz		**Due Date:**	18 Dec 2020

```
root@mininet:~# ping 10.0.0.4
```

You can observe that the topology is **not of a Tree**, but a **Mesh**, meaning that there are possible **LOOPS** between the switches, for example **s1–s3–s2–s4**, which, in a traditional local network would not work if the **Spanning Tree Protocol** (STP) would not be active (which by default is in every switch). STP builds a loop-free logical topology, preventing bridge loops by disabling those links that are not part of the spanning tree, leaving a single active path between any two network nodes.

But in a network composed by SDN compatible switches, there is no STP in the switches, as they are not build with a Control Plane.

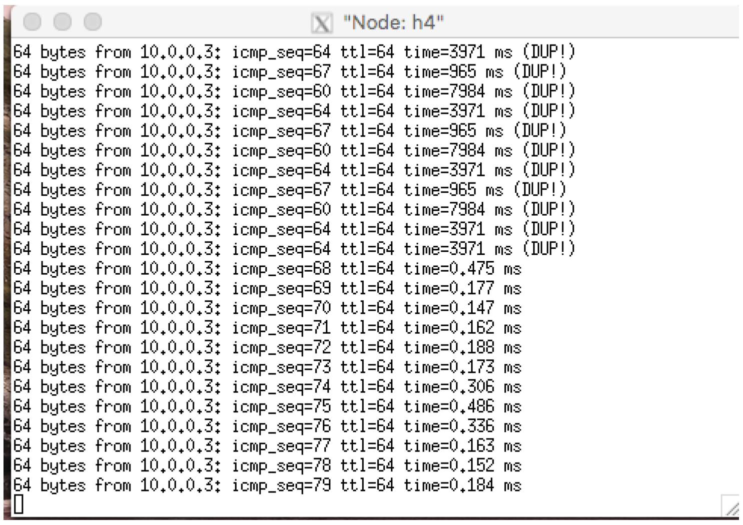
9.1.5 Simulate a broken link

To simulate a broken link in the network, move the mouse pointer over one of the blue links in the network (for example, between Switches **s5** and **s2**) and right-click. Choose ***Link Down*** from the menu that appears . That link will turn into a dashed blue line, indicating that it is down.

You may observe that pings from **h8** to **h1** momentarily halt, as no more traffic is received at host **h1**, but after a couple of seconds ping flow again to **h1**.

Simillarly, it may happen the same with pings between **h7** and **h4**.

Now, restore the link operation by right-clicking on the dashed line and choosing ***Link Up*** from the menu. The link will again appear as a solid blue line, and you will observe duplicate echo replies (DUP), as illustrated in Figure 19, which are packets with the same sequence number, meaning that the same packet was sent via both links of switch **s5** until the correct flows were re-installed in the switches by the controller.



```
64 bytes from 10.0.0.3: icmp_seq=64 ttl=64 time=3971 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=67 ttl=64 time=965 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=60 ttl=64 time=7984 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=64 ttl=64 time=3971 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=67 ttl=64 time=965 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=60 ttl=64 time=7984 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=64 ttl=64 time=3971 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=67 ttl=64 time=965 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=60 ttl=64 time=7984 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=64 ttl=64 time=3971 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=64 ttl=64 time=3971 ms (DUP!)
64 bytes from 10.0.0.3: icmp_seq=68 ttl=64 time=0.475 ms
64 bytes from 10.0.0.3: icmp_seq=69 ttl=64 time=0.177 ms
64 bytes from 10.0.0.3: icmp_seq=70 ttl=64 time=0.147 ms
64 bytes from 10.0.0.3: icmp_seq=71 ttl=64 time=0.162 ms
64 bytes from 10.0.0.3: icmp_seq=72 ttl=64 time=0.188 ms
64 bytes from 10.0.0.3: icmp_seq=73 ttl=64 time=0.173 ms
64 bytes from 10.0.0.3: icmp_seq=74 ttl=64 time=0.306 ms
64 bytes from 10.0.0.3: icmp_seq=75 ttl=64 time=0.486 ms
64 bytes from 10.0.0.3: icmp_seq=76 ttl=64 time=0.336 ms
64 bytes from 10.0.0.3: icmp_seq=77 ttl=64 time=0.163 ms
64 bytes from 10.0.0.3: icmp_seq=78 ttl=64 time=0.152 ms
64 bytes from 10.0.0.3: icmp_seq=79 ttl=64 time=0.184 ms
```

Figure 19: Duplicate echo replies after re-establishing the broken link.

When you stop the ping, you may then observe in the statistics that there were 0% PACKET LOSS, but many Duplicate packets.

Check the flow table on switch **s1** again both in the OFM Application and in mininet terminal window, enter the command `ovs-ofctl dump-flows s1` again.

10 Finishing your Experiments

The recommended sequence to stop and cleanup the experiment is as follows.

10.1 Stop the network emulation and cleanup

Quit Wireshark that you had initiated on hosts. Stop any ***ping*** command by pressing **Ctrl-C** on the keyboard. Then, quit the Mininet CLI by typing ***exit*** at the mininet prompt.

It is also advisable to "clean" any phantom Mininet processes in the system with the command `sudo mn -c`:

```
vagrant@mininet:~$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/
   noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core
    ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-
   nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /
   dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
......
*** Cleanup complete.
vagrant@mininet:~$
```

We can now exit the session:

```
vagrant@mininet:~$  exit
logout
Connection to 127.0.0.1 closed.
```

10.2 Stop the ODL Controller and the OFM Application

You can now stop the ODL controller by executing "logout" at the ODL prompt (it may take a bit of time . . .):

```
opendaylight-user@root>logout
```

```
vagrant@odl:~$
```

For the OFM, just press **Ctrl-C** on the keyboard at the VM session in order to stop the web server process:

```
vagrant@ofm:~/OpenDaylight-Openflow-App$ grunt
Running "connect:def" (connect) task
Waiting forever...
Started connect web server on http://localhost:9000
^C
vagrant@ofm:~/OpenDaylight-Openflow-App$
```

10.3 Halt the VMs

In order to stop the Virtual Machines and to verify the global state of all active Vagrant environments on the system, we can issue the following commands:

```
:~$ vagrant halt
==> odl: Attempting graceful shutdown of VM...
==> ofm: Attempting graceful shutdown of VM...
==> mininet: Attempting graceful shutdown of VM...
:~$ vagrant global-status
```

Confirm that the statuses of the VMs is 'powered off'. In order to prevent those machines to use resources, and if not using again, you may destroy them, as they can now be recreated with that simple command `vagrant up`. Confirm that there are no VMs listed.