OPENFLOW TUTORIAL

BASIC CONCEPTS

HAVE FUN

Contents

Table of Figures

Summary

Software-Defined Networking (SDN) is an emerging network architecture that aims to be dynamic, practical, advantageous and extendible. This architecture therefore lends itself to being ideal for the dynamic nature of modern applications. Software-Defined Networking (SDN) is "*the physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices"* (Matteson, 2014) according to Open Network Foundation that defines its standard.

The following tutorial can be useful for who is looking for an innovative solution based on standard protocols and open-source platforms, to increase network capabilities and to model the network for specific purposes. The OpenFlow protocol seems, in fact, to fill the gap between the rapidly growing market requirements and inadequate traditional network capabilities. In this sense, Software Defined Networking (SDN) represents a disruptive innovation in the IT landscape.

The complexity of current networks makes it difficult for technicians to apply a broad set of access policy, security, QoS and other directives to satisfy the high demand of mobile users' expectations. The current report explains through examples the contributions of the *SDN* architecture as dynamic, easy to manage, economically advantageous and adaptable solution; In fact, using simple open-source tools like *Mininet* and *OpenDaylight,* it is possible to build complex topologies and investigate real case scenarios.

Mininet is a very powerful tool that well suited for tests purpose. Using Linux kernel command, it can simulate an entire network with light virtualisation, allowing, within the context of the execution of a single operating system, to initiate virtual interfaces. The importance of monitoring tools like *Wireshark* is also fundamental to debug controller and traffic, and improvements can be obtained both in memory waste and in implementation time. Working with the *OpenDaylight* controller makes it possible to centralise any type of analysis and functionality on packets in transit in the network while maintaining an overview of real-time networking.

What will you learn?

The tutorial will investigate SDN concepts/scenarios, including a modular and flexible platform that acts as controller. We want to highlight the function of providing centralised control and programmable physical and virtual devices to improve and speed up network performance. Besides, we will show the benefits of control devices through standard open-source protocols reducing operational complexity. OpenFlow Protocol has the scope to ensure a high level of abstraction of its functionality, to help developers and network engineers create new applications, customise configuration and network management, providing a transparent approach that promotes innovation and simplifies the network infrastructure management.

We will investigate the *OpenFlow* protocol implementation using *Mininet*, *Wireshark* and *SDN controller* demonstrating OpenFlow protocol in action to give an overall analysis of several aspects:

1. How the protocols work.
2. Traffic generated to perform OpenFlow.
3. Flow table actions and modifications.
4. Different topologies.
5. Interaction between components.

Introduction

The *SDN* concept started in 2005 when A. Greenberg and G. Hjalmtysson wrote an article in talking about the fundamental concepts of networking oriented to programmability and control.

What's SDN-fundamental architecture

The logical structure of *SDN* consists of three main functional layers. (Jarraya, 2014) In the first level (**Application layer**), we find a series of applications, through which developers must interface to use the network and its functionality. The second is the **control layer**, the computational capacity of the entire network. The check is implemented through a series of suitably developed software. The software can interact with device decision tables, to safely route the *data flow*, based on the model developed by the controller. At the last level (**Infrastructure layer**), there is the physical infrastructure made through dedicated hardware. The elements that make up this level can only perform basic packet functions, but their primary purpose is to securely forward packets by building connections in an extremely stable and fast way. (ManarJammal, 2014)

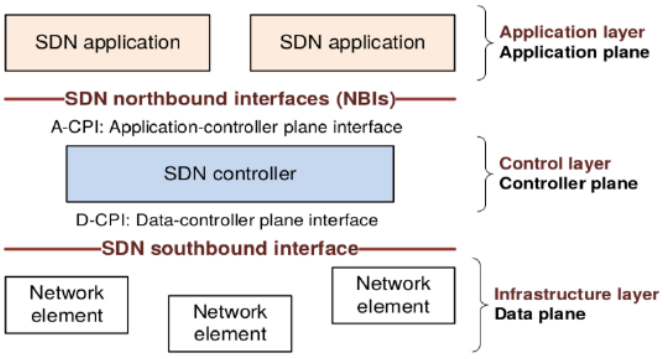


Figure 1 Logical SDN Structure

What's OpenFlow Protocol?

OpenFlow allows direct access and modification of a device's forwarding plan (switch or router), both physical and virtual, through the network. **SDN technologies based on OpenFlow allow IT to support broadband and the dynamic nature of modern applications, to adapt the network to the changing business needs, and to reduce in a way the complexity of management and operations is significant**.

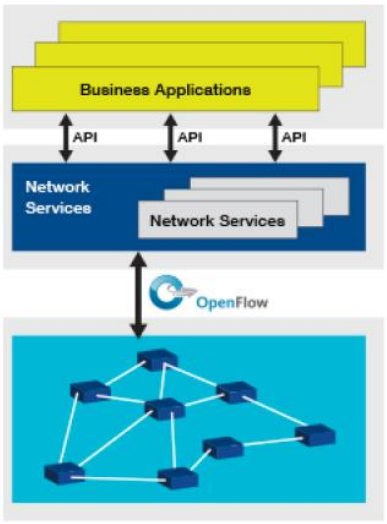


Figure 2 OpenFlow Logical Architecture (Bennett, 2015)

A logical switch that supports OpenFlow contains one or more flow tables and a group table, which performs the control and forwarding of packets.

OpenFlow Pipeline and Tables

The pipeline OpenFlow processes a packet received by an OpenFlow switch. A series of operations are performed on the packet comparing their fields and those in the flow tables, whose purpose is to find the set of instructions that will go next performed to forward the packet properly. The instructions collected during the processing, with every correspondence found, are saved in the Action-Set.

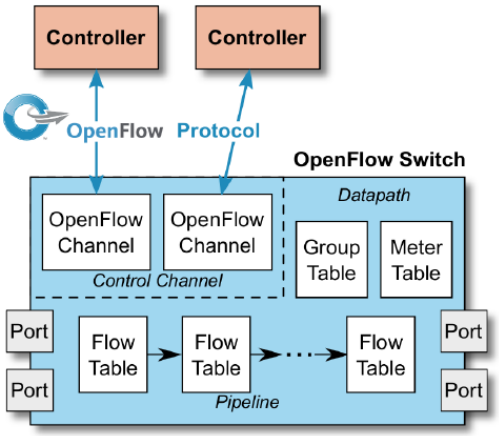


Figure 3 OpenFlow Switch Components (Curtis, 2018)

Flow Table

The main components of a single item in a table are:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

- **match fields**: packet values are compared to look for matches. They contain the front door, the packet header, and other optional fields;
- **priority**: precedence in case multiple matches;
- **counter**: increased every time for every packet that matches with a flow entry;
- **instructions**: instructions to add to the Action-Set or ways that to pipeline flow;

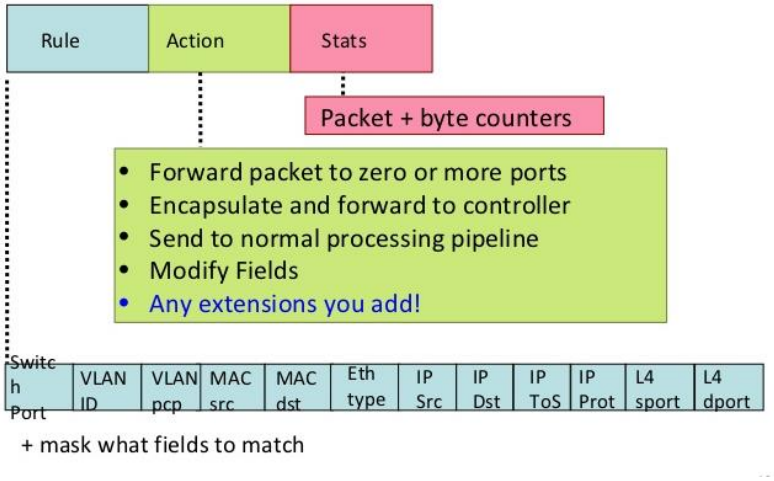


Figure 4 Flow Table Entry and Actions

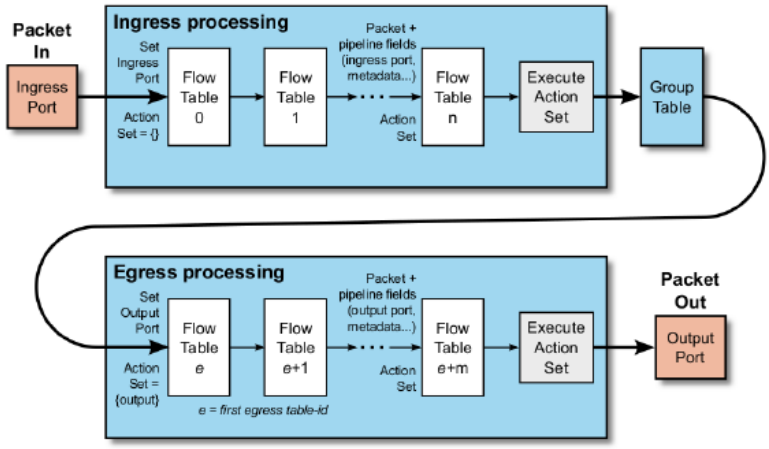


Figure 5 Packet flow in the pipeline (Foundation, 2016)

Technical Specifications-Mininet

The idea of OpenFlow is to separate the functions of the *data plane* (processing data packet forwarding and managed by the OpenFlow switch) and the control plane (controlling the data plane and undertaken by the OpenFlow controller). Performing simple tests, we can understand the interaction process between the OpenFlow switch and the OpenFlow controller. The environment consists of mainly three parts:

1. Virtual machine using Hyper-V, configured with the following characteristics:
    a. CPU: 1 Core
    b. RAM: 2 GB
    c. Storage: 10 GB
    d. Ubuntu 18.04 OS
    e. JVM: 1.7

running *Mininet*, simulating an *OpenFlow* switch and several hosts and *Wireshark* installed.

2. *OpenDaylight* controller, running in the virtual machine providing OpenFlow controller functionality.

3. *OpenDaylight-Openflow-App*, *OpenFlow Manager* software, which runs in the same virtual machine as the Open Daylight controller, provides more user-friendly functions.

Thanks to each of these components, we can set up an SDN network and understand its main concepts based on the labs done in class.

Mininet

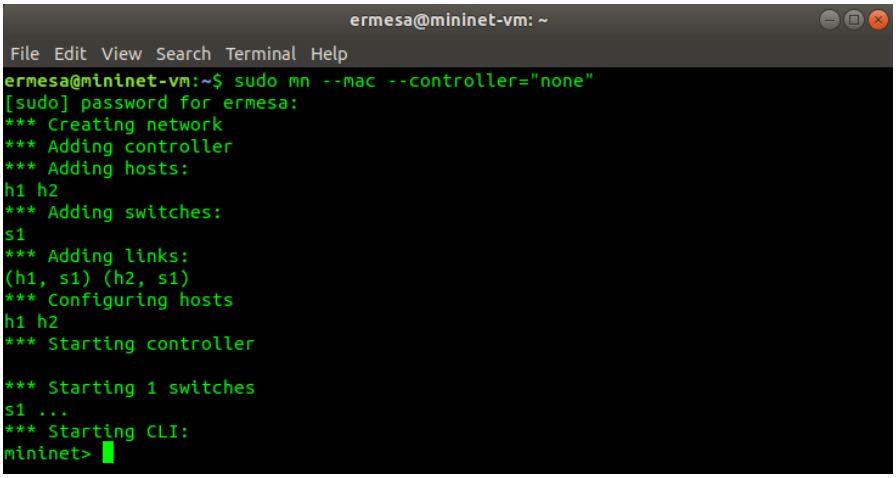
Mininet (Appendix) allows to create network topologies according to specific needs. This emulation software has been developed to be able to perform extensive tests, also using limited resources such as Notebook and PC-Desktop. Mininet is designed to automate the creation process network, using Linux kernel commands to simulate desired configurations. It is possible to create an arbitrary number of hosts, switches and controller, so the user can also test multiple network topologies. Mininet connects switches and hosts through virtual ethernet, and therefore it allows to test even complex networks on a single Linux environment without needing to achieve them in the physical world.

Mininet Test

The configuration of a network topology using *Mininet* is quite simple: through the *sudo mn* command, it is possible to create a network that has two hosts (h1 and h2), a switch (s1) and no controller. In a second test, a controller (c0) is added to verify the fundamental role of the controller in an SDN network:

Test controller="none"

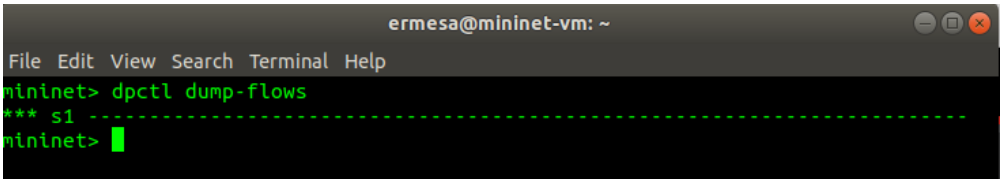
```
mininet@mininet-VM:~$ sudo mn --mac --controller="none"
```



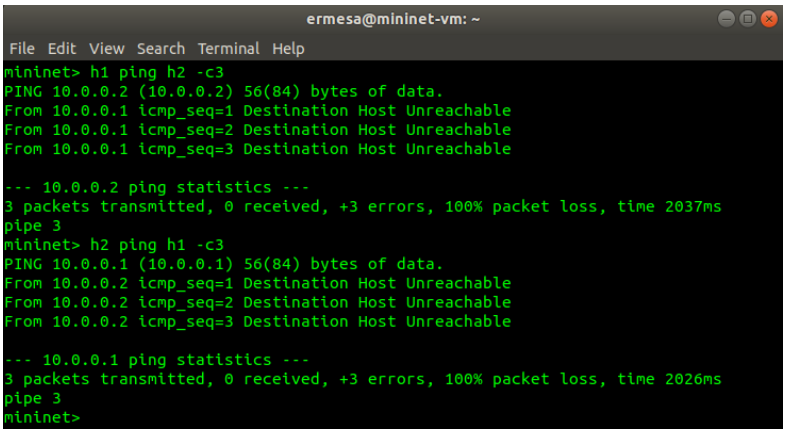
The command creates a network with:

- One switch (S1)
- Two hosts (h1 and h2)
- h1 eth0 connected to s1 eth0
- h2 eth0 connected to s1 eth1
- no OpenFlow controller

We can notice using the command *dpctl dump-flows* that the flow table is empty due to the absence of the controller, proving that in an SDN network, the controller is essential.



We can check the connectivity between the two hosts by launching the command *h1 ping h2* showing that hosts are unreachable:



The reason for the Destination Host Unreachable result is easy to understand. Without having set any controller, the packets reached the OpenFlow switch which however did not know what to do with it at the moment that the controller was absent. In the logic of SDN networks, the packets that come to the network devices must be turned to the controller if in their flow table there are no rules relating to incoming packets. Very simply now we can try building a controller with the following command using the *Mininet* default controller: *mininet@mininet-vm:~$ sudo mn –mac*

```
ermesa@mininet-vm: ~
ermesa@mininet-vm:~$ sudo mn --mac
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Moreover, this time the ping command will not fail:

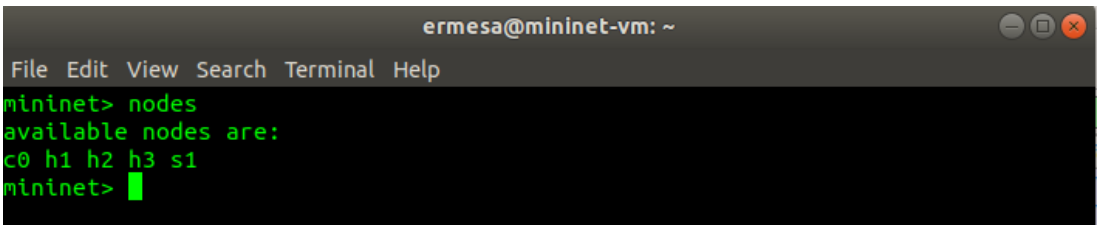
```
ermesa@mininet-vm: ~
mininet> h1 ping h2 -c4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.613 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.066 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.083 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3050ms
rtt min/avg/max/mdev = 0.047/0.202/0.613/0.237 ms
mininet> h2 ping h1 -c4
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=0.043 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.072 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.043 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3066ms
rtt min/avg/max/mdev = 0.042/0.050/0.072/0.012 ms
mininet>
```

We can also analyse the flow table that now is populated as the controller orchestrate the forwarding of the packets:

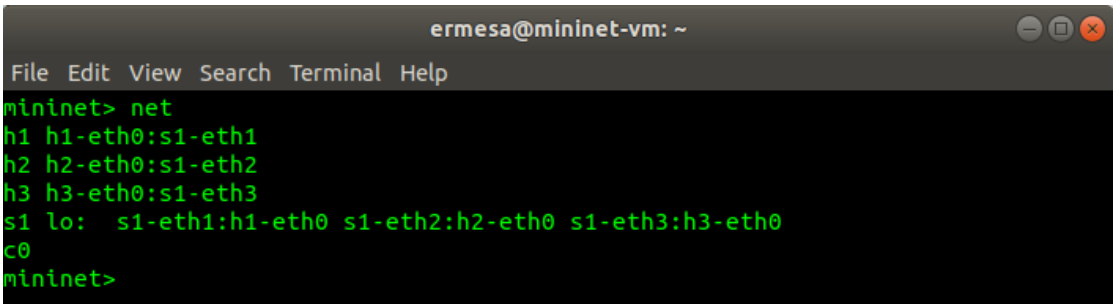
```
ermesa@mininet-vm: ~
File  Edit  View  Search  Terminal  Help
0, priority=1,icmp,in_port="s1-eth1",vlan_tci=0x0000/0x1fff,dl_src=00:00:00:00:0
0:01,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type
=8,icmp_code=0 actions=output:"s1-eth2"
 cookie=0x0, duration=33.794s, table=0, n_packets=3, n_bytes=294, idle_timeout=6
0, priority=1,icmp,in_port="s1-eth2",vlan_tci=0x0000/0x1fff,dl_src=00:00:00:00:0
0:02,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type
=0,icmp_code=0 actions=output:"s1-eth1"
 cookie=0x0, duration=24.143s, table=0, n_packets=3, n_bytes=294, idle_timeout=6
0, priority=1,icmp,in_port="s1-eth2",vlan_tci=0x0000/0x1fff,dl_src=00:00:00:00:0
0:02,dl_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type
=8,icmp_code=0 actions=output:"s1-eth1"
 cookie=0x0, duration=24.142s, table=0, n_packets=3, n_bytes=294, idle_timeout=6
0, priority=1,icmp,in_port="s1-eth1",vlan_tci=0x0000/0x1fff,dl_src=00:00:00:00:0
0:01,dl_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type
=0,icmp_code=0 actions=output:"s1-eth2"
 cookie=0x0, duration=61.925s, table=0, n_packets=25, n_bytes=1942, priority=0 a
ctions=CONTROLLER:128
mininet>
```

Once created the desired topology, it is possible to interact with Mininet through CLI (Command Line Interface) command lines:

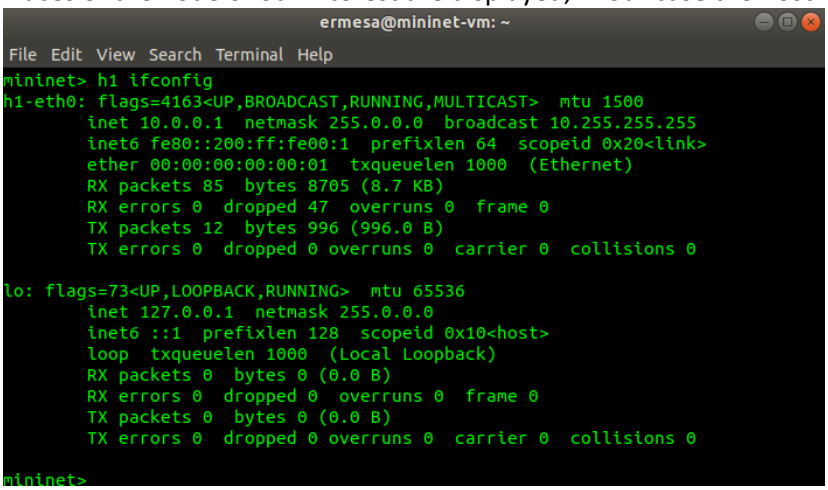
nodes: displays the nodes created; in our case, we have *three hosts, one switch and one controller.*



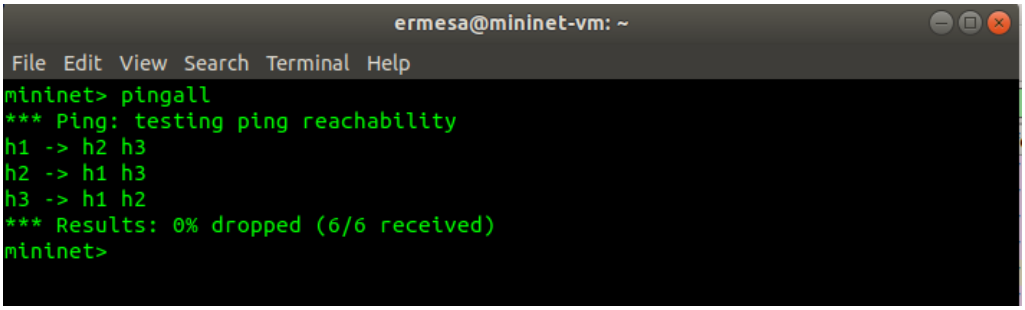
net: shows the nodes and connections associated with their ports.



h1 ifconfig: The interfaces of the node of our interest are displayed, in our case the host h1

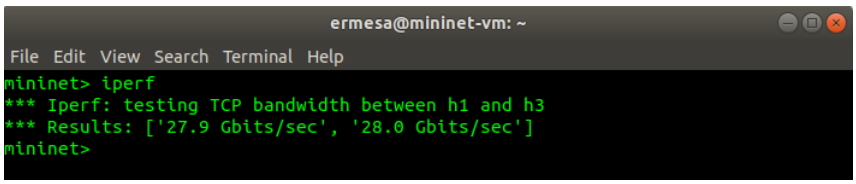


pingall: ping all hosts connected to the network:

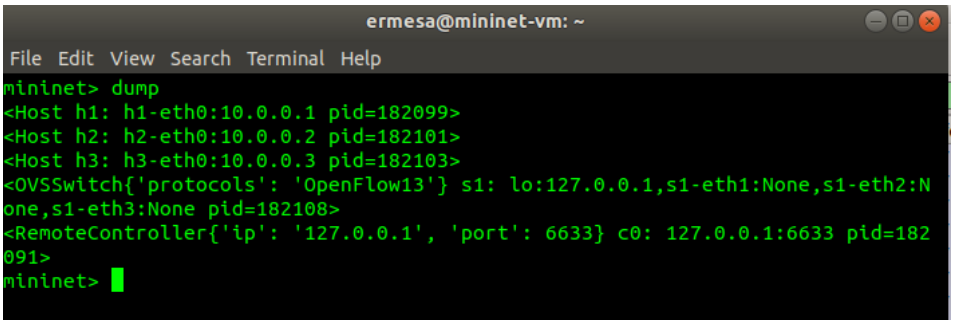


Through the *pingall* command, inserted from the Mininet console, if this does not give an error, we can verify the communication between the controller and the switch of the emulated virtual network.

iperf: perform a band test between 2 network hosts.



dump: print the information relating to all the nodes created on the screen.



It is also recommended to clean up ARP cache on the hosts. Otherwise, we might not see some ARP requests/replies as the cache will be used instead:

mininet> h1 ip -s -s neigh flush all

mininet> h2 ip -s -s neigh flush all

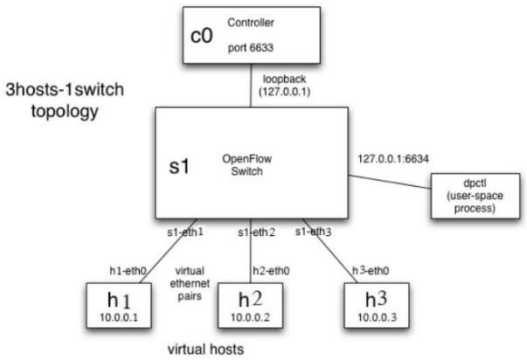
```
mininet> h2 ip -s -s neigh flush all
10.0.0.1 dev h2-eth0 lladdr 00:00:00:00:00:01 used 2049/2044/2021 probes 1 STALE
10.0.0.3 dev h2-eth0 lladdr 00:00:00:00:00:03 used 2049/2049/2025 probes 4 STALE

*** Round 1, deleting 2 entries ***
*** Flush is complete after 1 round ***
mininet> h1 ip -s -s neigh flush all
10.0.0.3 dev h1-eth0 lladdr 00:00:00:00:00:03 used 2055/2055/2025 probes 4 STALE
10.0.0.2 dev h1-eth0 lladdr 00:00:00:00:00:02 used 2055/2050/2019 probes 1 STALE

*** Round 1, deleting 2 entries ***
*** Flush is complete after 1 round ***
mininet> h3 ip -s -s neigh flush all
10.0.0.2 dev h3-eth0 lladdr 00:00:00:00:00:02 used 2063/2058/2023 probes 1 STALE
10.0.0.1 dev h3-eth0 lladdr 00:00:00:00:00:01 used 2063/2058/2023 probes 1 STALE

*** Round 1, deleting 2 entries ***
*** Flush is complete after 1 round ***
mininet>
```

Test with remote controller

We can create a network using the remote controller by typing it on the terminal example:

sudo mn --topo single,3 --mac --switch ovs --controller remote

```
ermesa@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovs --controller rem
ote
[sudo] password for ermesa:
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```



The line of code in details:

- **sudo** is the instruction that enables us the administrator permissions on Linux.
- **mn** is the instruction that is asked to Mininet to create the desired topology.
- **--topo single mouse, 3** create the network topology, a switch and three hosts connected to it.
- **--mac requests** that each interface be assigned a MAC address equivalent to the IP address
- --**switch** allows to choose which type of switch implementation should be used for emulation.
- --**controller** asks that the switches connect to the specified controller.
- **remote** without specifying an IP address, the switches will connect to the loopback address 127.0.0.1 (it is used in TCP / IP networks to identify the local machine on which the programs are running, called localhost) with TCP ports: 6633.

We can test the connectivity between hosts using *mininet> h1 ping h2 -c3* :

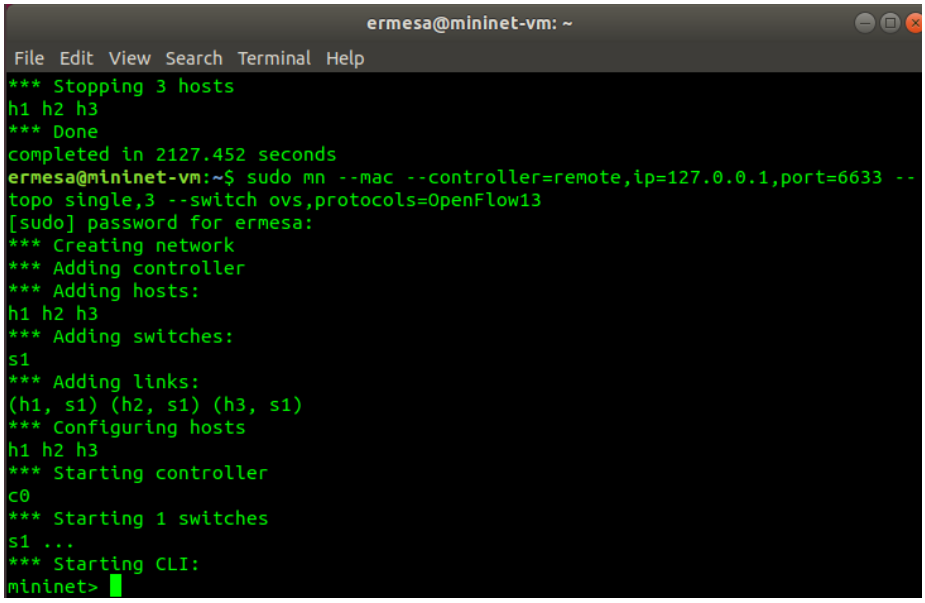
```
ermesa@mininet-vm: ~
File  Edit  View  Search  Terminal  Help
mininet> h1 ping h2 -c3
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.782 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.140 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.156 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2028ms
rtt min/avg/max/mdev = 0.140/0.359/0.782/0.299 ms
mininet>
```

By observing the output of this command, we can notice that the time of the first ping is higher than the others. Because the first waiting for the rules to manage this type of actions to be installed in the switch.

Interaction with Wireshark

Wireshark (Appendix), is a software for protocol analysis or packet sniffer, used for solving network problems, for protocol analysis and development. The software is released under the *OpenSource* license and runs on most Unix systems and Microsoft Windows systems. For simplicity, we create a minimal topology with Mininet, to analyse it with Wireshark:

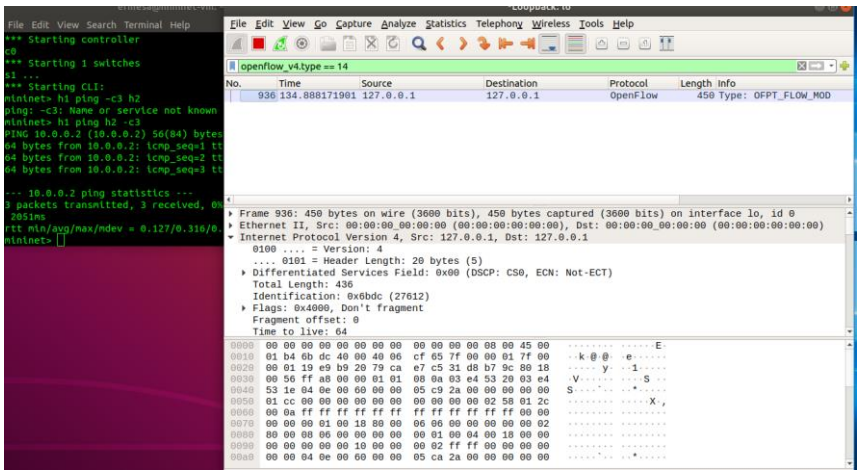
sudo mn --mac --controller=remote,ip=127.0.0.1,port=6633 --topo single,3 --switch ovs,protocols=OpenFlow13



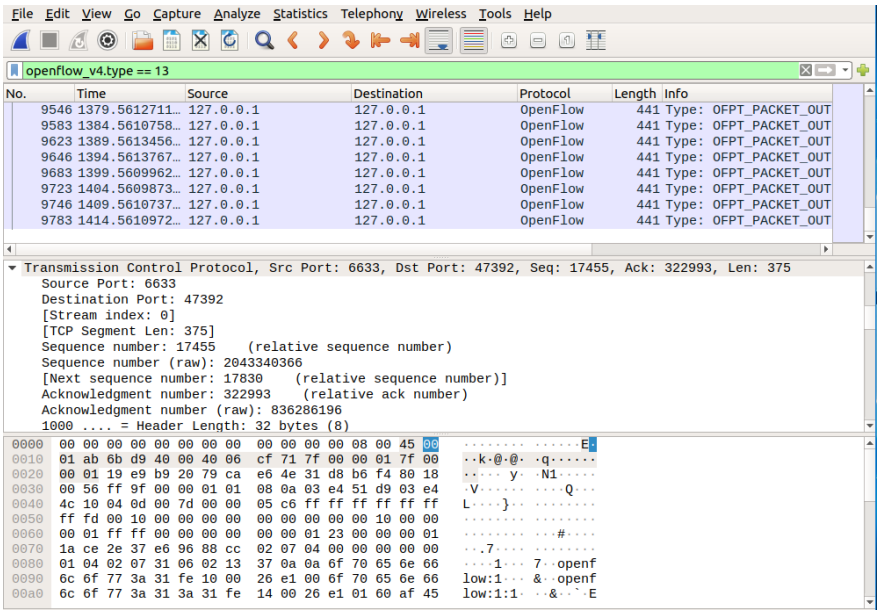
We start Wireshark in administrator mode, having traffic captured on the Mininet loopback interface. We can immediately note the exchange of *OpenFlow* messages, which takes place between the controller and the switch. In this way, it is also possible to debug the controller being able to identify any anomalies.

Wireshark configuration

At this point, we see in Wireshark, a group of messages. Because all the messages are sent via localhost, determining the sender is difficult when there are many emulated switches. To ignore the initial communication messages, we just type in the filter: *openflow_v4*. For example, to filter messages to match the *FLOW MOD* message, we can use *openflow_v4.type == 14*:



To filter part to match the *PACKET OUT* message, we can use *openflow_v4.type == 13*:



OpenFlow protocol Analysis

In this section, we use *Mininet* and *Wireshark* to analyse the structure of the messages and understand how the OpenFlow protocol works.

The ONF, the organisation that manages OpenFlow standard, defined it as the first communication interface between the control plane and the data plane of an SDN architecture. This kind of protocol is fundamental within the SDN networks, and it is for this reason that is also the most used protocol in this sector, **as in this way the control of the network is moved from proprietary network devices to open external control software**.

The OpenFlow protocol can be broken into four components: *message layer, state machine, system interface, and configuration*. (flowgrammable, 2020)

Component	Description
Message Layer	The message layer is the core of the protocol stack. It defines valid structure and semantics for all messages. A typical message layer supports the ability to construct, copy, compare, print, and manipulate messages.
State Machine	The state machine defines the core low-level behavior of the protocol. Typically, it is used to describe actions such as: negotiation, capability discover, flow control, delivery, etc.
System Interface	The system interface defines how the protocol interacts with the outside world. It typically identifies necessary and optional interfaces along with their intended use, such as TLS and TCP as transport channels.
Configuration	Almost all aspects of the protocol have configurations or initial values. Configuration can cover anything from default buffer sizes and reply intervals to X.509 certificates.
Data Model	Another way to consider the OpenFlow protocol is to understand its underlying data model. Each switch maintains a relational data model that contains the attributes for each OpenFlow abstraction. These attributes either describe an abstractions capability, its configuratio

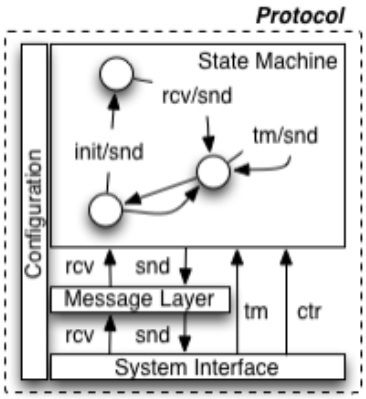
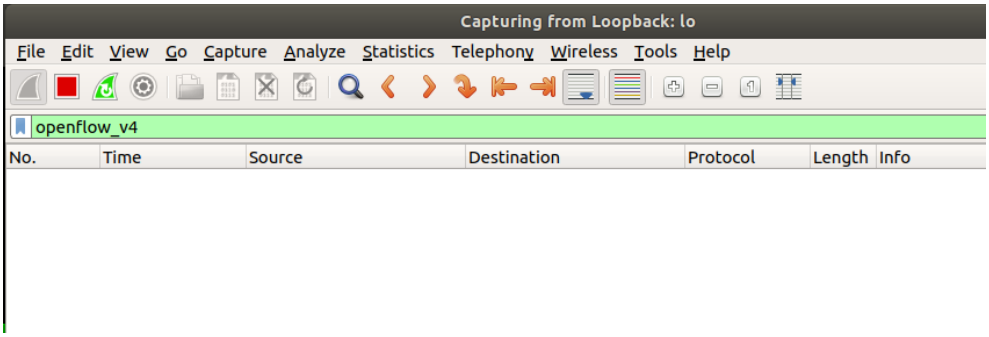


Figure 6 OpenFlow Components (flowgrammable, 2020)

OpenFlow is part of the interaction as soon as a switch is inserted within the SDN topology.

To better understand and prove the structure of the OpenFlow based message from the literature available on-line some tests are carried out to break its structure using a simple topology created with *Mininet* and analysed using *Wireshark*, two tools also presented before. First of all, we notice that before any setup no OpenFlow type messages are shown in Wireshark (lo interface):



With the following command, we create a single switch with three hosts attached to it. The hosts will be assigned static IP addresses and MAC addresses

sudo mn --mac --controller=remote,ip=127.0.0.1,port=6633 --topo single,3 --switch ovs,protocols=OpenFlow13

```
ermesa@mininet-vm:~$ sudo mn --mac --controller=remote,ip=127.0.0.1,port=6633
 --topo single,3 --switch ovs,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

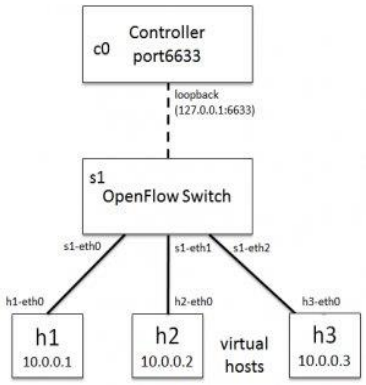
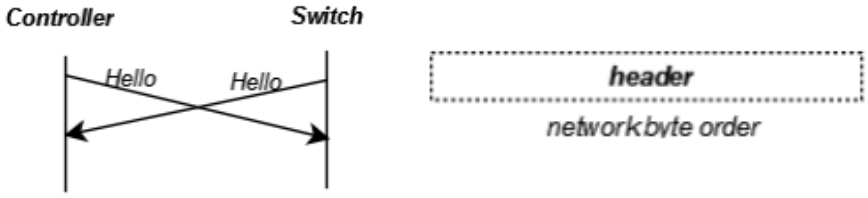
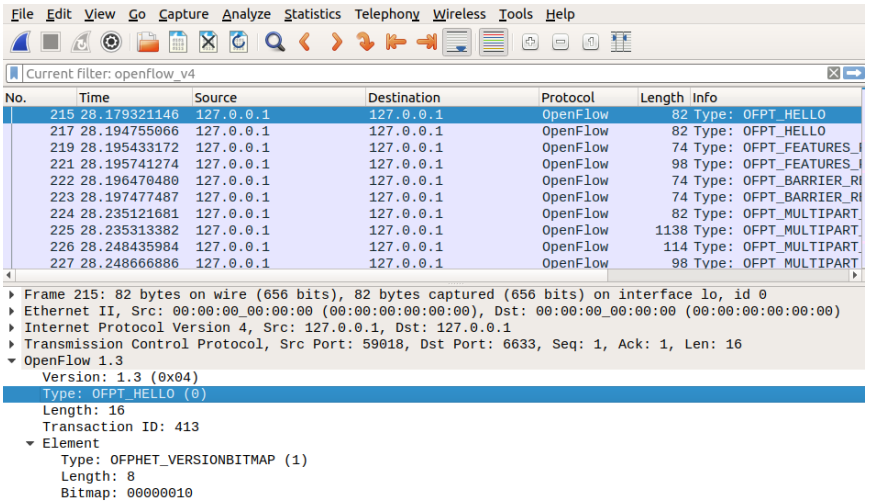


Figure 7 Single switch topology

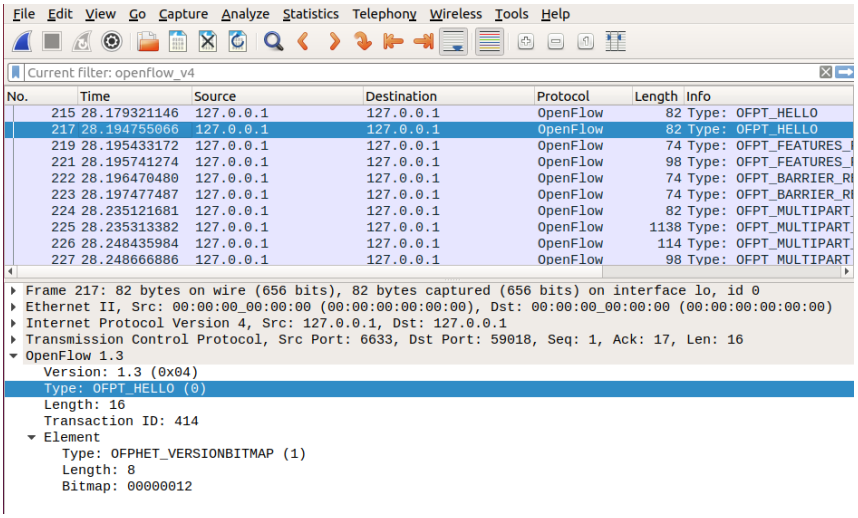
Once the topology is created, the first step is to create a channel of secure communication (TCP/TLS, System Interface Layer) between switch and controller. Once do that, the switch will initiate the conversation by sending a packet *OFTP_HELLO* to the controller, specifying the supported version (Version field).



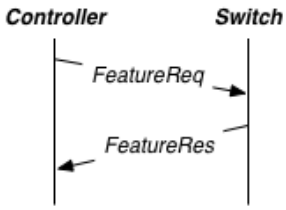
The *Hello* message is captured by *Wireshark,* as shown in the test carried out and has no body; it is comprised of only a header (flowgrammable, 2020):



Consequently, the controller responds in turn with an *OFTP_HELLO* packet in which it establishes the final version of the protocol to be used. If there is no common version, the switch will be notified by the controller through a packet of type *OFTP_HELLO_FAILED_ERROR_MSG*.



After this initial exchange of messages, the controller will send a packet of type *OFTP_FEATURES_REQUEST,* which will contain only one header. The switch will respond with an *OFTP_FEATURES_REPLY* packet within which it will specify a series of data relating to the network device such as *capabilities*, *buffer size* (number of packets the switch can queue) and a *datapath_id* that identifies it univocally.



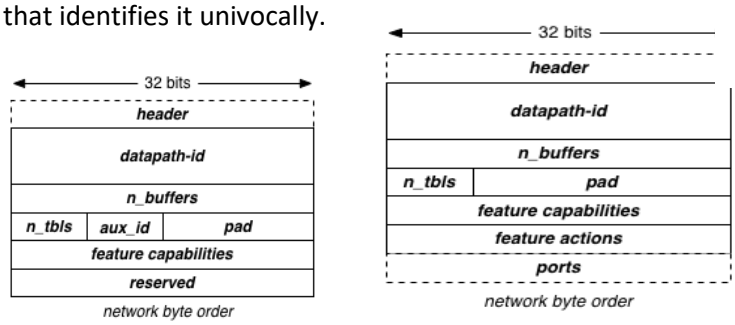


Figure 8 FeatureReq and FeatureRes Controller (flowgrammable, 2020)

In the figure below is shown the *Wireshark* capture of both *req* and *replay* messages :

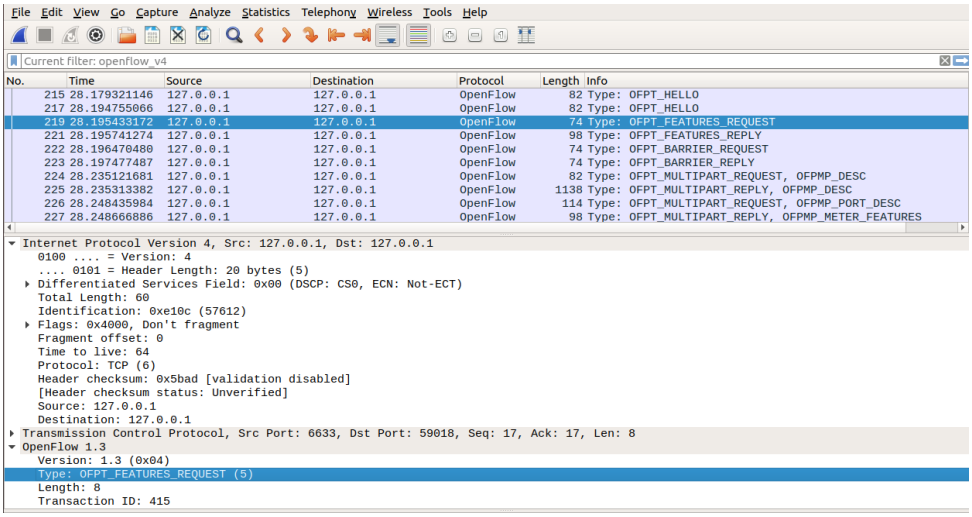


Figure 9 Feature Request captured in Wireshark

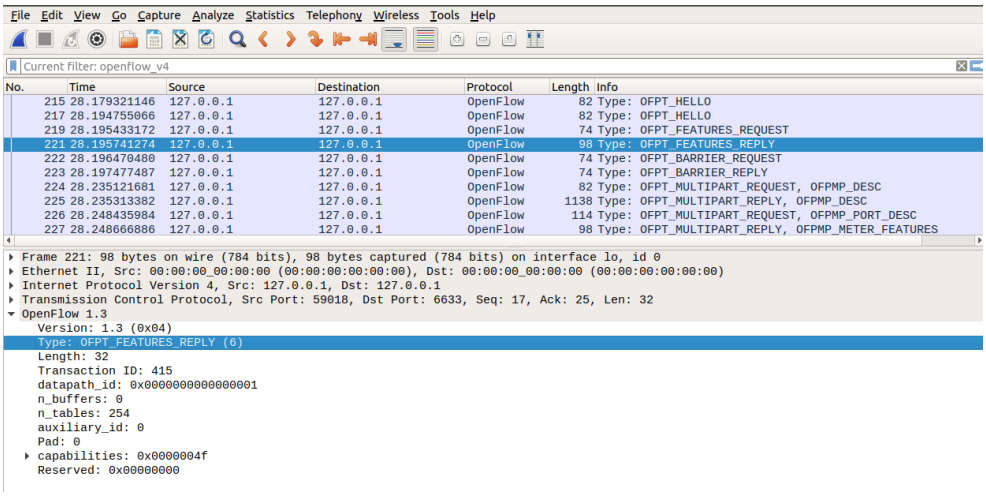
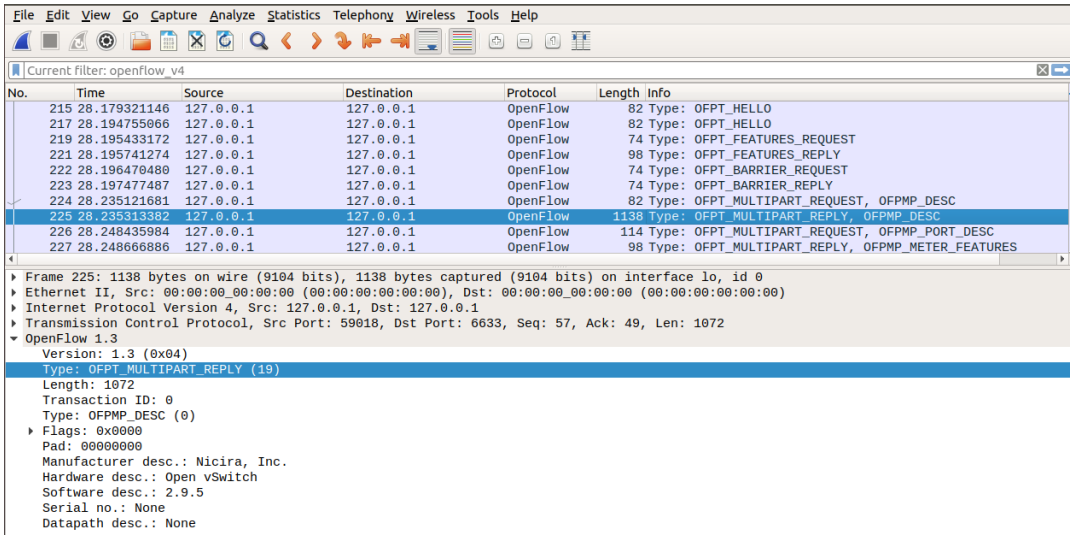


Figure 10 Feature Reply captured in Wireshark

OFTP_MULTIPART_REQUEST and are used *OFTP_MULTIPART_REPLY are used* to know some information as the status of the component, the number of ports, the flows of the device



After the initial configuration has completed correctly, the device network is ready to receive incoming packet streams. As soon as the switch receives a particular flow-in, it will immediately check its flow tables looking for a match with the packet arrived: if it finds no actions related to that packet, by default it will send the controller a packet of type *OFTP_PACKET_IN*. Within the *packet_in* there will be a *header*, a *buffer_id* which it will represent a univocal id of the packet arrived and other fields (i.e. *total_len*, *in_port*).

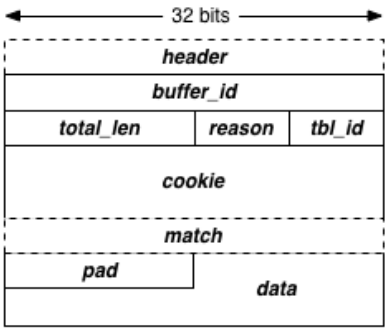
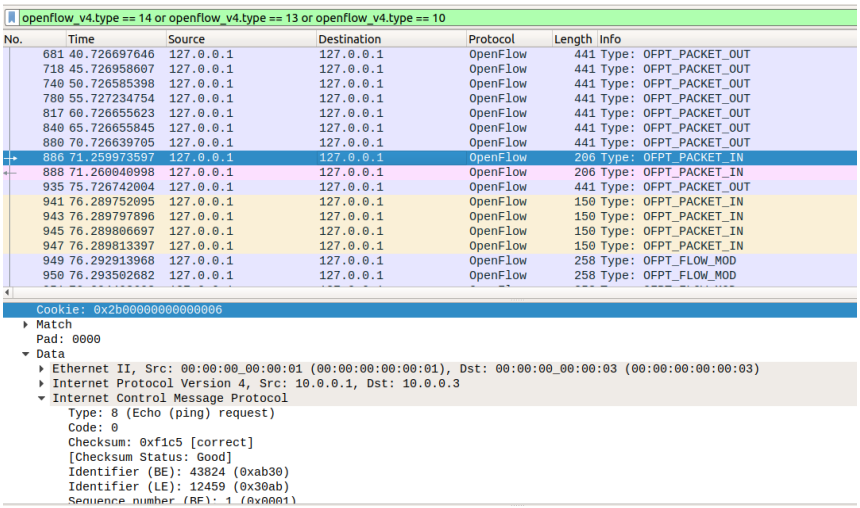


Figure 11 Packet_IN Structure OpenFlow 1.3

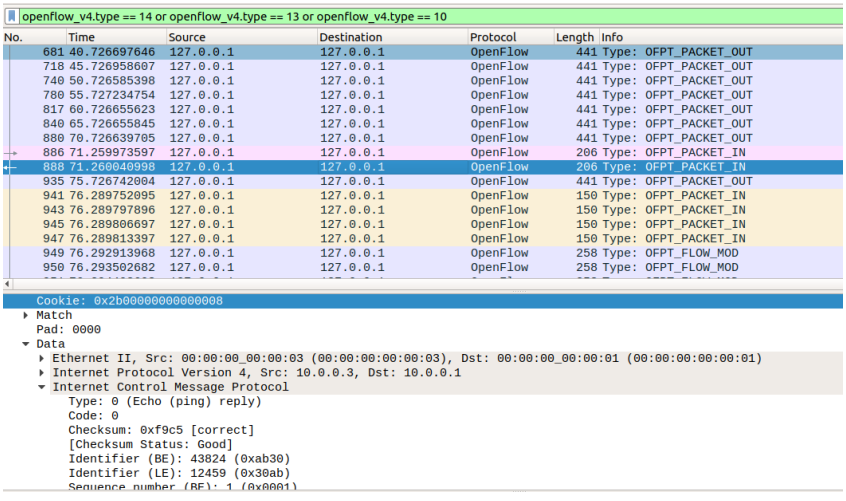
Once the controller has received the *packet_In*, it will determine what action record inside the switch and will communicate it through a packet of type *OFTP_FLOW_MOD*. In this way, the switch will be able to record in its flow table the type of action to be performed in the future and possibly it will be able to send the packet out via a message of type *OFTP_PACKET_OUT*. After between the sending host and the receiving host, the action is registered, it will no longer need to involve the controller.

We can now perform *ping* between hosts *h1* and *h2* using command *h1 ping h3 -c3*.

Echo ping request from h1 10.0.0.1 to h3 10.0.0.3:



Echo ping reply from h1 10.0.0.1 to h3 10.0.0.3:



The figure below shows the exchange of packets between the two virtual hosts h1 and h3 created within the Mininet network. The first packet reaches the destination taking longer than the others (0.555 ms against 0.425 ms on average of the following ones). The delay happens because, being **the first packet that is exchanged between the two hosts and that passes inside the switch, the latter will deliver it to the controller at first**.

```
mininet> h1 ping h3 -c1
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.555 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.555/0.555/0.555/0.000 ms
mininet> h1 ping h3 -c1
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.425 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.425/0.425/0.425/0.000 ms
mininet> h1 ping h3 -c1
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.411 ms
```

Figure 12 Compare time to reach the destination

Remote Controller

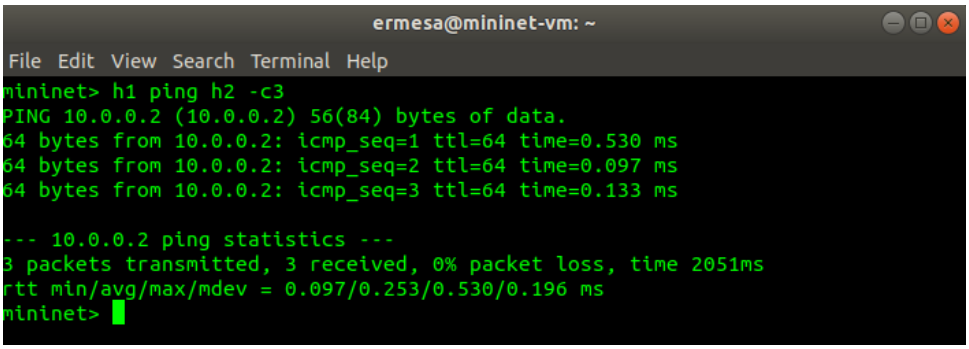
In a Mininet network, switches can be connected to a remote controller, which could be in the VM, outside the VM or anywhere in the world.

The OpenFlow tutorial uses *controller --remote* for starting up a simple learning switch that we can create using a controller framework like OpenDaylight, ONOS, POX, NOX, Beacon or Floodlight. **This setup may be convenient if we already have a custom version of a controller framework or we want to test a controller running on a different physical machine** (maybe even in the cloud). We can try the command filling the host IP and listening port:

```
$ sudo mn --controller=remote,ip=[controller IP],port=[controller listening port]
```

```
$ sudo mn --controller=remote,ip=127.0.0.1,port=6633
```

If we generate some traffic (e.g. *h1 ping h2*), we should be able to observe some output in the flow table showing that the switch has connected and that some flow table entries have been installed. The messages should be read from highest priority to lowest priority.



```
mininet> dpctl dump-flows --protocols=OpenFlow13
*** s1 ------------------------------------------------------------------------
--
 cookie=0x2b00000000000003, duration=60807.467s, table=0, n_packets=0, n_byte
s=0, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2a0000000000002a, duration=77.612s, table=0, n_packets=1, n_bytes=4
2, idle_timeout=600, hard_timeout=300, priority=10,dl_src=00:00:00:00:00:01,d
l_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
 cookie=0x2a0000000000002b, duration=77.612s, table=0, n_packets=1, n_bytes=4
2, idle_timeout=600, hard_timeout=300, priority=10,dl_src=00:00:00:00:00:02,d
l_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x2b00000000000009, duration=60805.461s, table=0, n_packets=33, n_byt
es=2358, priority=2,in_port="s1-eth1" actions=output:"s1-eth2",output:"s1-eth
3",CONTROLLER:65535
 cookie=0x2b0000000000000a, duration=60805.461s, table=0, n_packets=31, n_byt
es=2226, priority=2,in_port="s1-eth2" actions=output:"s1-eth1",output:"s1-eth
3",CONTROLLER:65535
 cookie=0x2b0000000000000b, duration=60805.461s, table=0, n_packets=31, n_byt
es=2226, priority=2,in_port="s1-eth3" actions=output:"s1-eth1",output:"s1-eth
2",CONTROLLER:65535
 cookie=0x2b00000000000003, duration=60807.459s, table=0, n_packets=10, n_byt
es=856, priority=0 actions=drop
mininet>
```

OpenDaylight

The remote controller used for test purpose is *OpenDaylight* (Appendix), the real engine of the SDN network. In this section, we explore in more details the characteristics of *OpenDaylight*, considered one of the most popular among SDN controllers. Furthermore, its structure is described and some of its main features analysed.

The ODL controller is an open-source, multi-protocol, scalable, extendable, modular infrastructure used for the implementation of a network based on the SDN concepts.

The open-source nature of the controller allows users to reduce operational complexity, extend the lifecycle of existing hardware devices at the physical level and to enable new services and available features.

The controller infrastructure provides centralised and programmable devices control and a high abstraction level of its functionality, to simplify the installation and management of the SDN network, allowing an improvement in its performance.

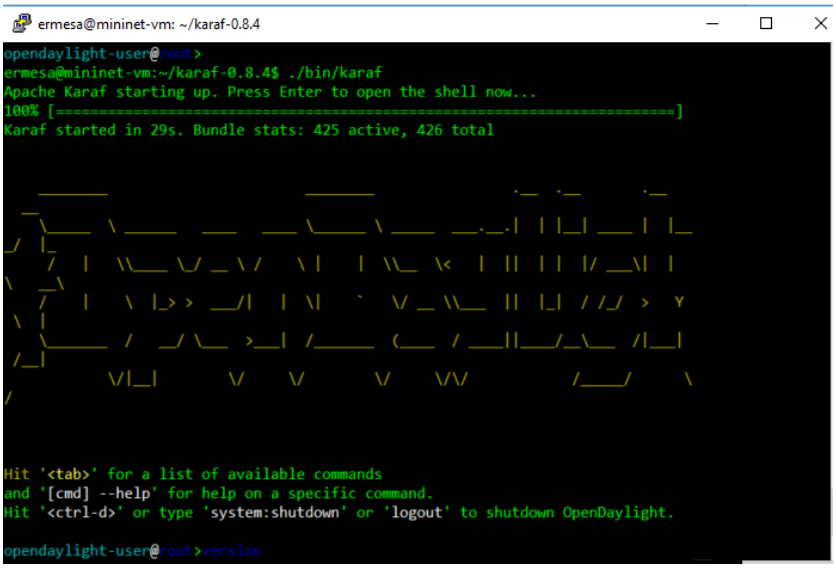


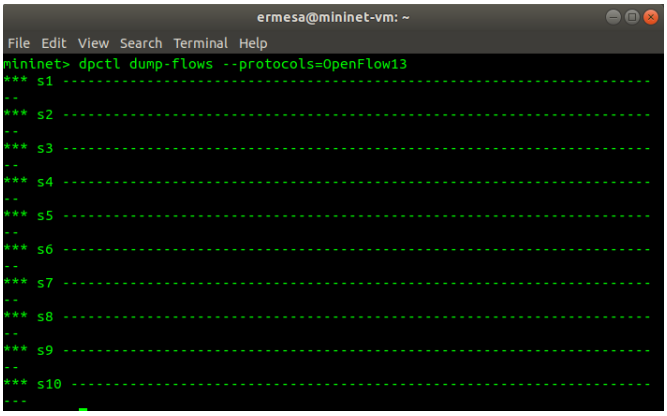
Figure 13 SSH session on managing ODL through Karaf console

OpenDayLight Tests

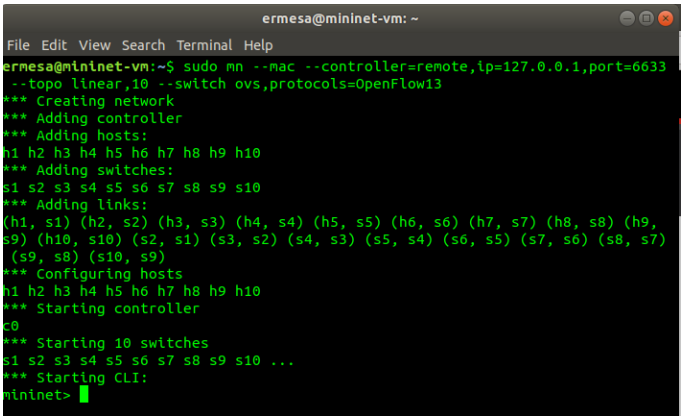
In this section, tests are performed to understand how to manage and monitor a centralised SDN network with OpenDaylight acting as a controller.

In previous sections, we used basic topologies to understand better how the protocols work and interact with each other. In this example, we want to create, through the Mininet network emulator, a linear network consisting of ten switches, each of which is connected to a terminal device.

With no packet transition, the flow table is empty as predictable from the behaviour analysis of the OpenFlow protocol communication using *Mininet* and *Wireshark* in the previous section:



sudo mn - linear type, 10 –mac --controller = remote, ip = <controllerIP>, port = 6633 --switch ovs, protocols = OpenFlow13



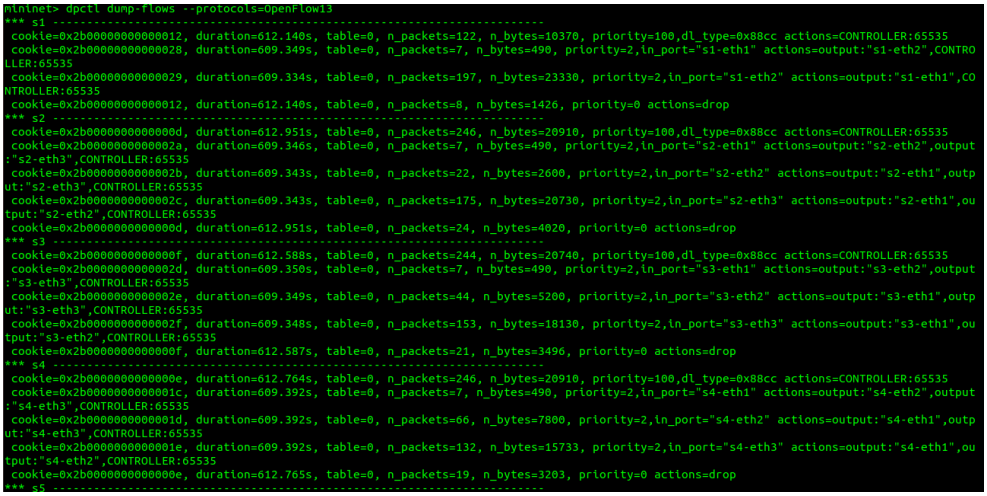


Figure 14 Flow table showing massages OpenFlow protocol communication

Through the *pingall* command, inserted from the Mininet console, if this does not give an error, we can verify the communication between the controller and the switches of the emulated virtual network:

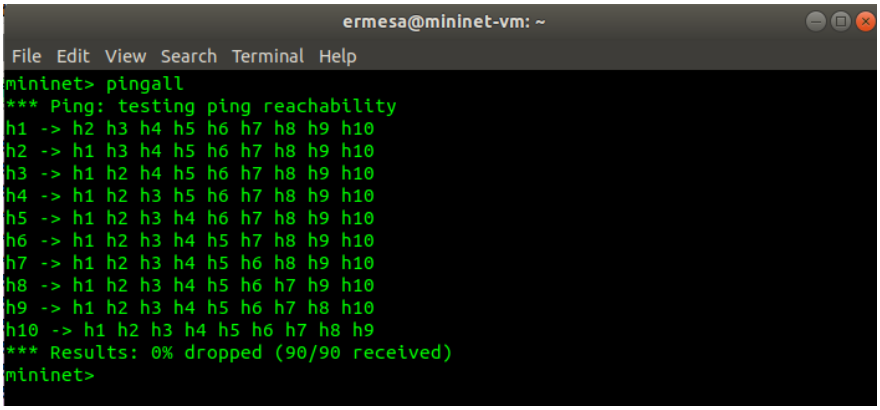
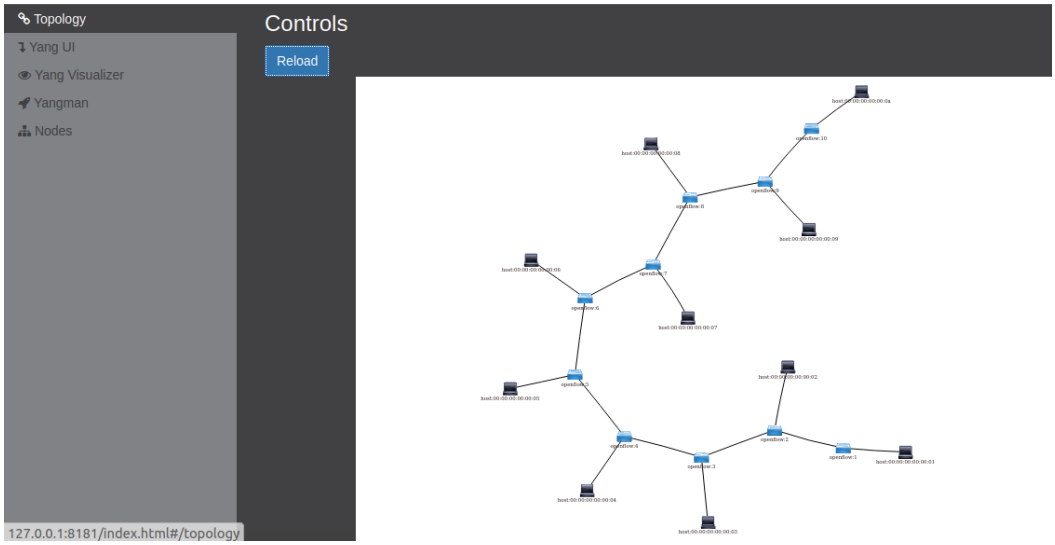


Figure 15 Pingall virtual hosts

Viewing the network in DLUX

Through the DLUX graphic interface, we can view the graphic representation of the virtual network generated in the previous paragraph after running the *pingall* command. *URL: http: // <IP controller>: 8181 / index.html*



Test with TSDR

TSDR provides a framework for the correct collection and storage of temporal data in a data store. TSDR offers features for three different datastores: HSQLDB, HBase and Cassandra. For the test, we use the HSQLDB datastore. **This type of data gives the user a real-time view of the phenomenon and predicts its future performance.**

feature: install odl-tsdr-hsqldb

Archiving HSQLDB files are automatically stored in the *<karaf folder>/tsdr/directory*. Once the *HSQLDB* datastore has been initialised, it is possible to test its operation through the ODL controller *karaf* console.

OpenFlow Protocol TSDR Test

The operation of the OpenFlow protocol can be tested through the TSDR framework. First, enable the TSDR feature, which allows to collect the packets of this protocol. (opendaylight-tsdr, 2018)

feature: install odl-tsdr-openflow-statistics-collector

Three different queries are available to view information about OpenFlow packets, which can be executed from the karaf console: *FLOWSTATS, PORTSTATS, FLOWTABLESTATS.*

FLOWSTATS. This query allows to view the list of characteristics of the individual flows installed, through the OpenFlow protocol, in the flow tables of the switches and specifies their behaviour and timespan. An example is shown below:

opendaylight-user@root>tsdr:list FLOWSTATS

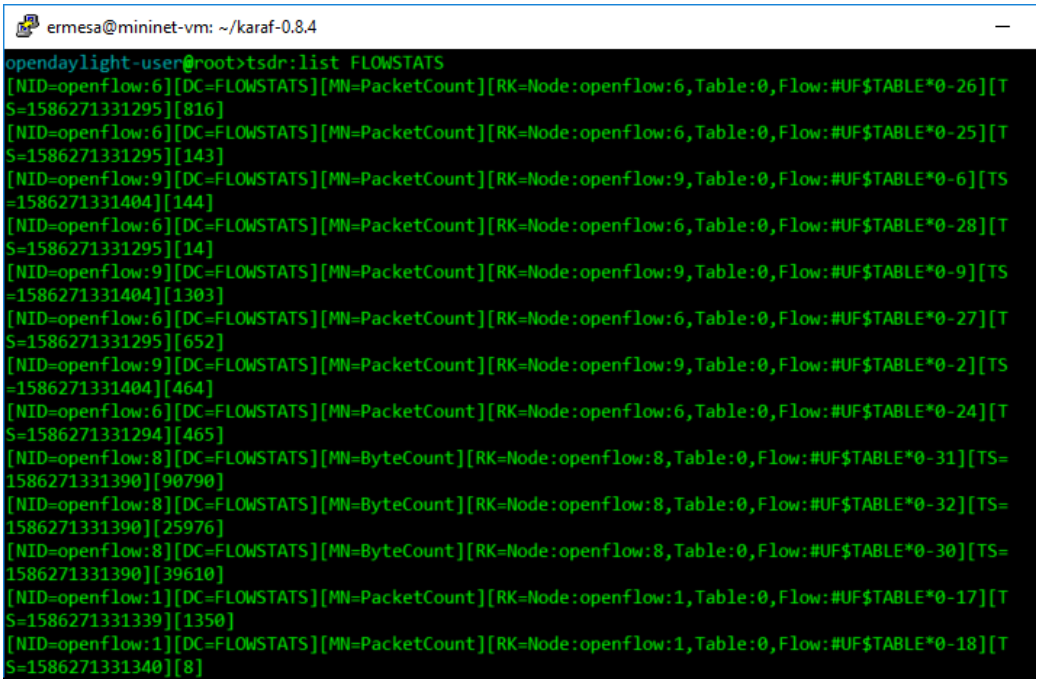
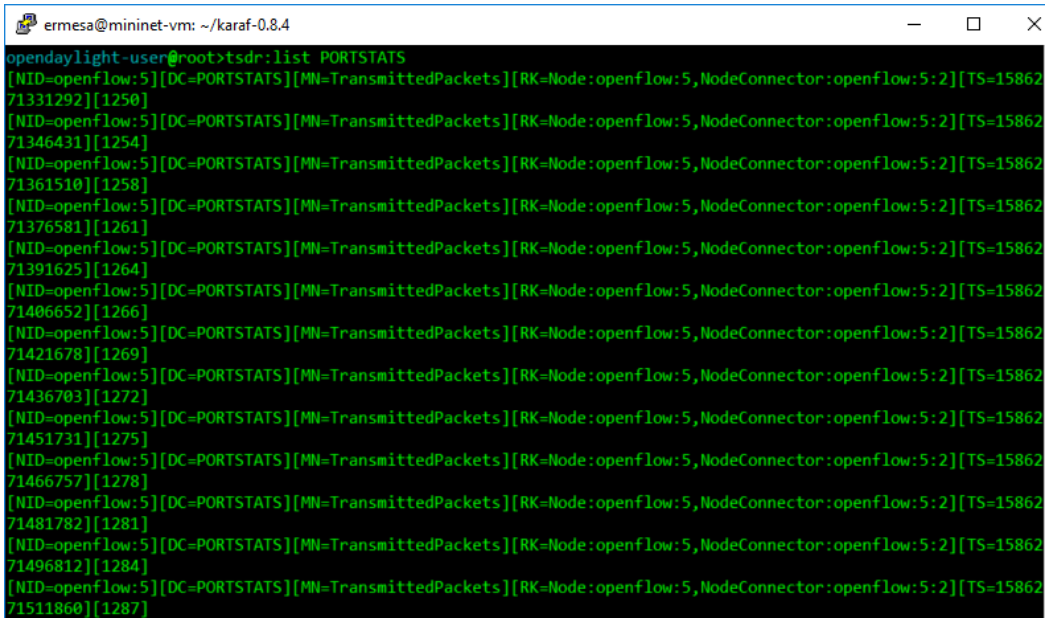
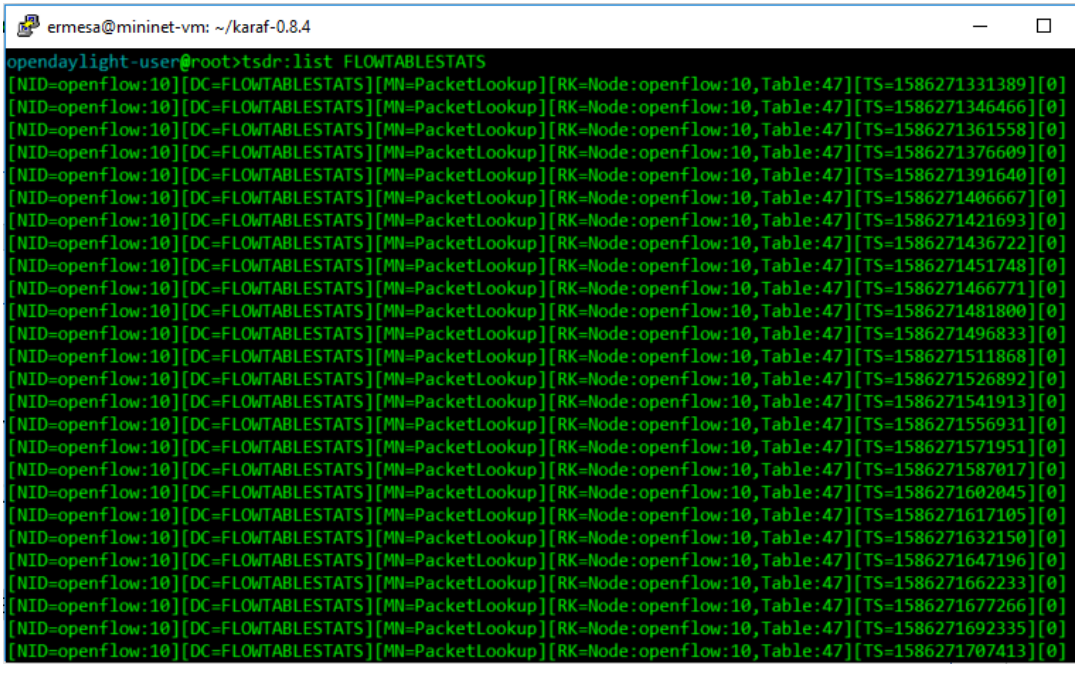


Figure 16 Statistics related to FLOWSTATS category

PORTSTATS. The following query is used to display the list of information relating to the behaviour of a packet passing through a specific port of a node belonging to the virtual network, as shown below: *opendaylight-user@root>tsdr:list PORTSTATS*



FLOWTABLESTATS. This last query is used to receive information related to the individual routing tables of the nodes present in the network: *opendaylight-user@root>tsdr:list FLOWTABLESTATS*



With the functions provided by TSDR, network administrators can use applications integrated in this framework, i.e. Grafana, that acquire the information obtained, process it and generate performance analyses of the automated network for a convergent administration and monitoring.

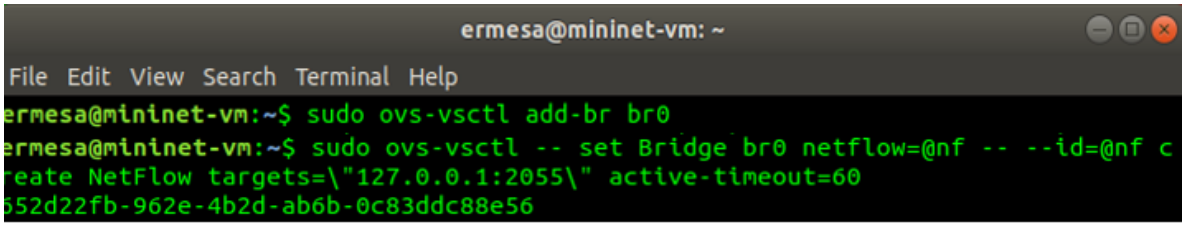
Learning switch process L2Switch

The functioning of the learning switch process can be verified through the TSDR framework, which collects information related to the NetFlow protocol. First of all, it is necessary to enable the analysis

protocol in the switches that make up the network. Besides, it is necessary to create a bridge through which NetFlow protocol records are sent from the OVS switches to the ODL controller.

ermesa@mininet-vm:~$ sudo ovs-vsctl add-br br0

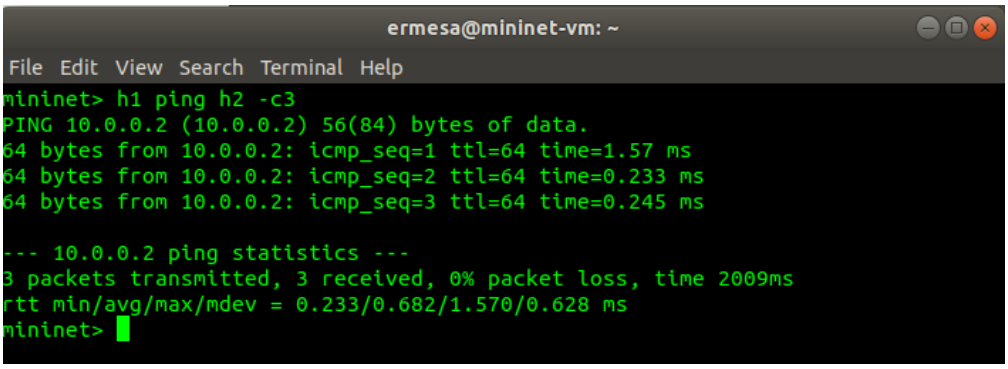
ermesa@mininet-vm:~$ sudo ovs-vsctl -- set Bridge br0 netflow=@nf -- --id=@nf create NetFlow targets=\"127.0.0.1:2055\" active-timeout=60



After that, it is necessary to install, in the karaf console of the ODL controller, the feature that has the function of collecting the information inherent to this specific protocol.

feature: install odl-tsdr-netflow-statistics-collector

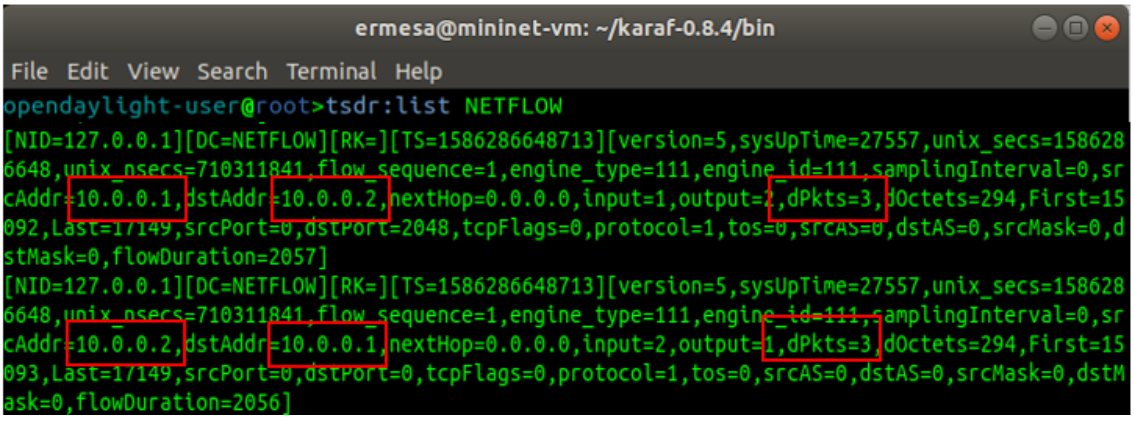
To verify the correct learning switch behaviour is necessary to create network traffic. To accomplish this, as we can see in the figure below, *ping between* specific hosts *h1 and h2*:



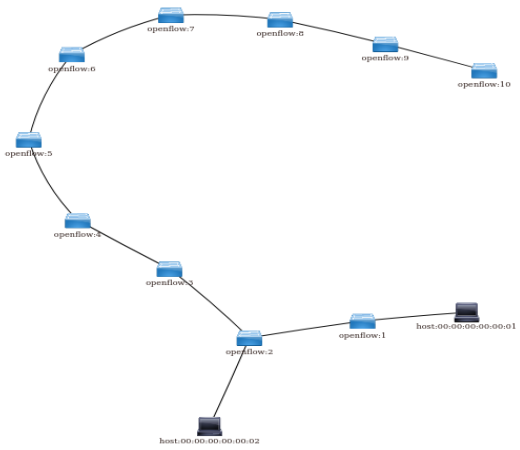
To view the information collected by TSDR, we type in the console:

tsdr: NETFLOW list

Through this query, it is possible to view the statistics relating to the NetFlow protocol. **The figure below highlights the direct exchange of packets between the two hosts and the speed with which it is carried out:**



Below is shown the representation of the topology after the ping in the ODL GUI:



OpenFlow Manager

The OpenFlow Manager (OFM) is an application that leverages the innovation to manage the OpenFlow network. (CiscoDevNet, 2014)

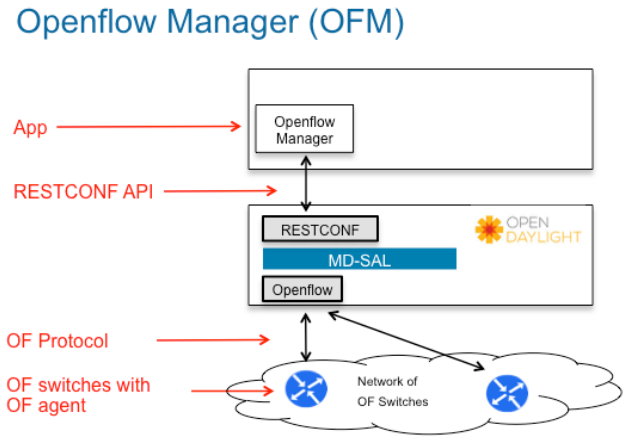
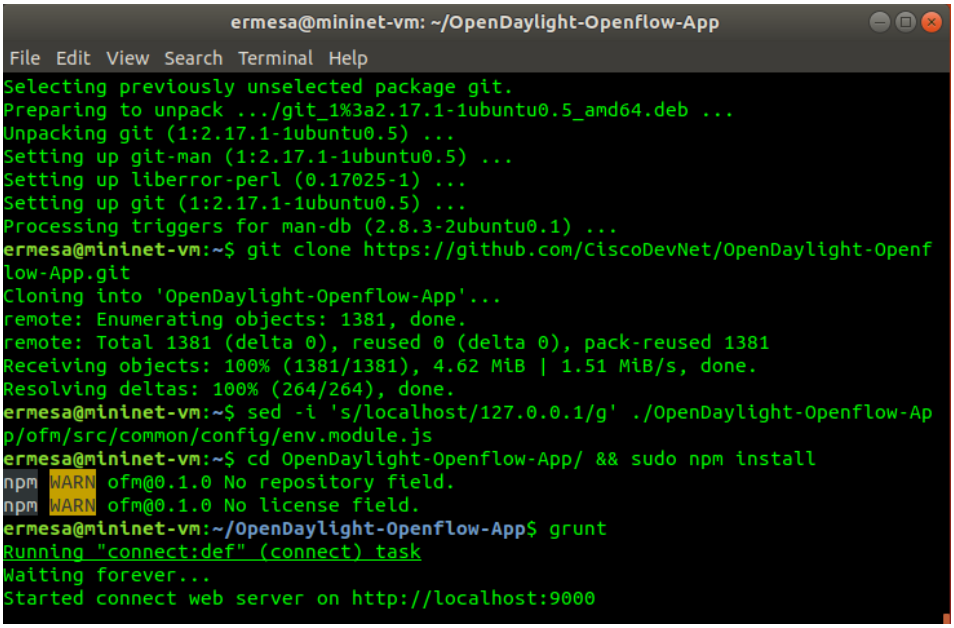


Figure 17 Architecture of the OFM components (CiscoDevNet, 2014)

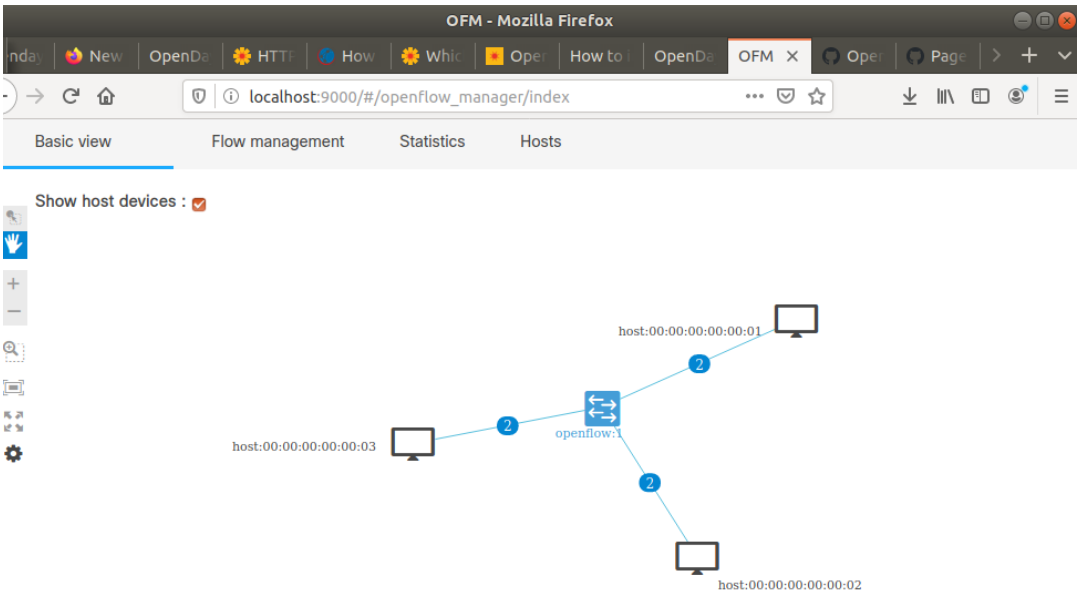
To install OFM and to start the webserver we can use:

```
$ git clone https://github.com/CiscoDevNet/OpenDaylight-Openflow-App.git
$ sed -i 's/localhost/10.10.10.2/g' ./OpenDaylight-Openflow-App/ofm/src/common/config/env.module.js
$ cd OpenDaylight-Openflow-App/ && sudo npm install
$ grunt
```



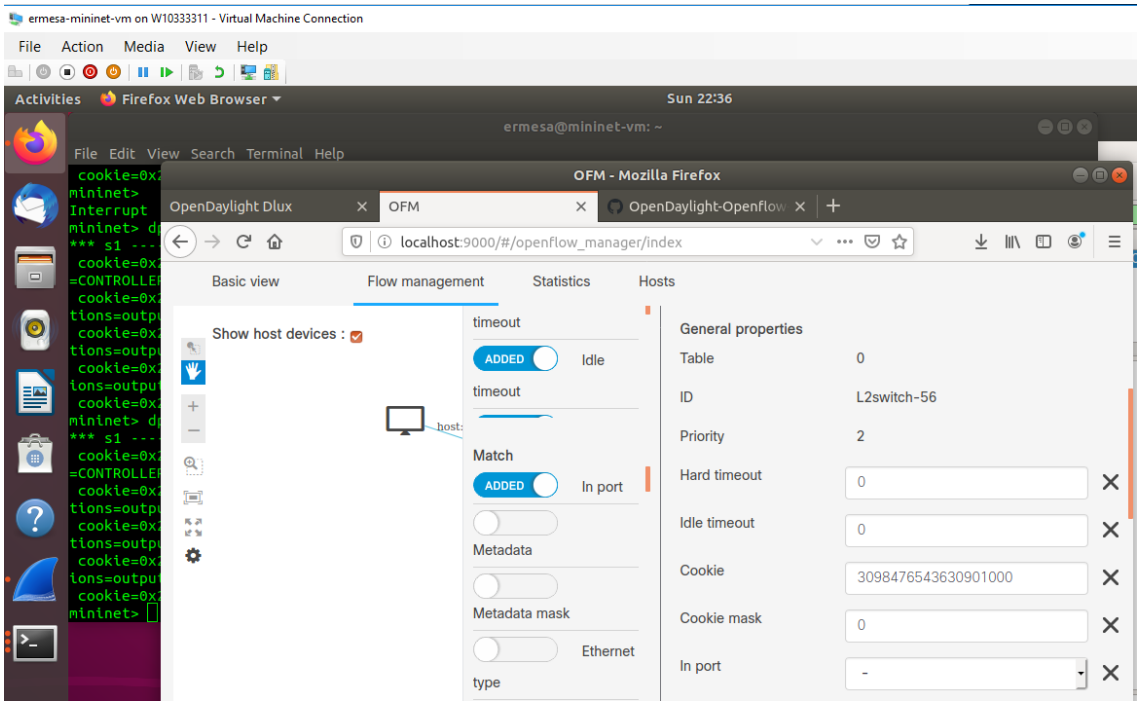
The basic subfunctions of OFM consist of:

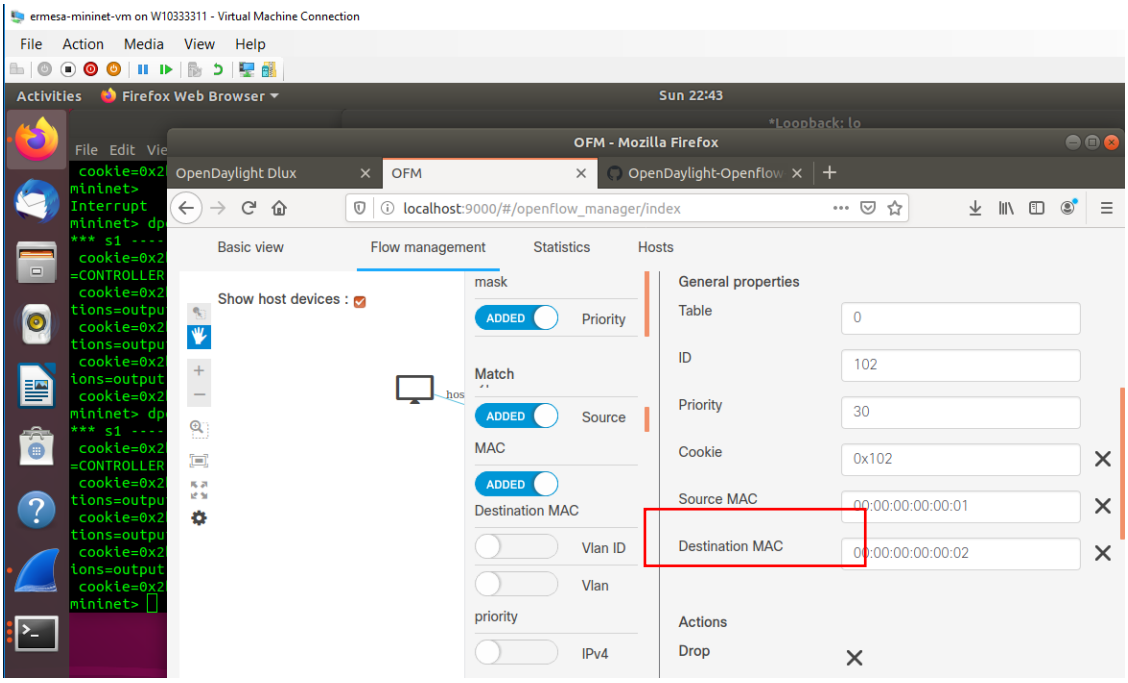
- *Basic View Tab*, the default tab, displays topology mapping the OpenFlow-enabled network devices and the hosts that are connected to them.
- *Flow Management Tab* can be used for determining the number of flows for each OpenFlow-enabled network device, viewing a listing of all currently configured flows, adding, modifying and deleting the flows.
- *Statistics tab* provides statistics for both the flows configured in the network and the corresponding device ports.
- *Hosts Tab* provides summary information for the OpenFlow-enabled host devices that OFM manages. (CiscoDevNet, 2014)



Flow entry Test

We can create a flow entry using OFM to manage the flow from h1 to h2. In this test, we drop the flow with source h1 and destination h2. To do this, we create a new flow in the flow table.

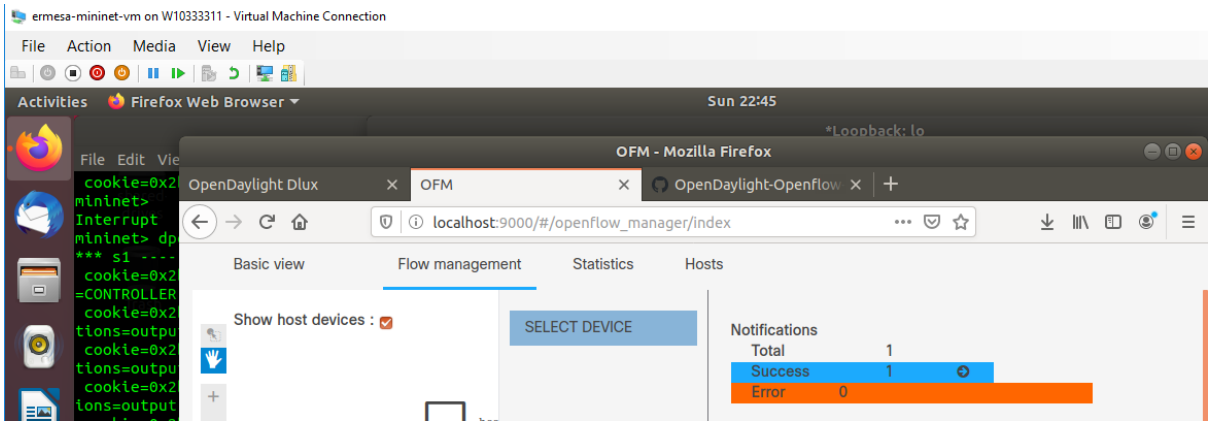


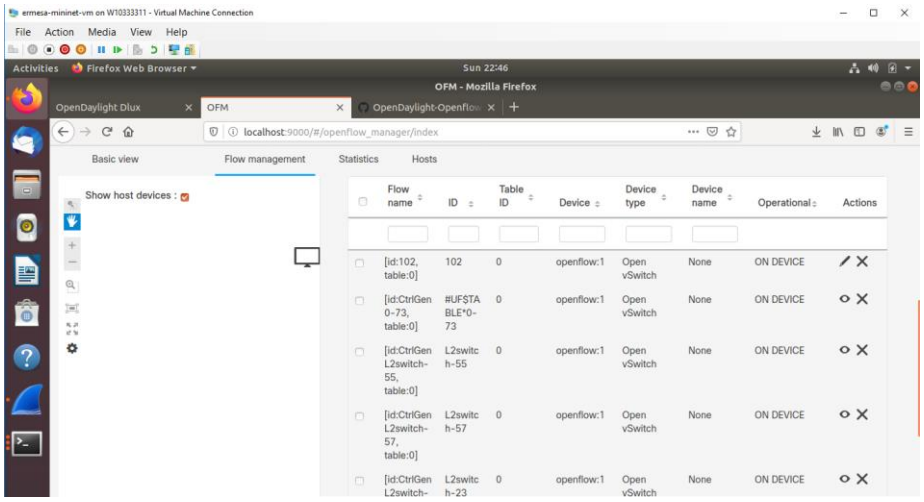


We can click *Show preview* to view the HTML text:

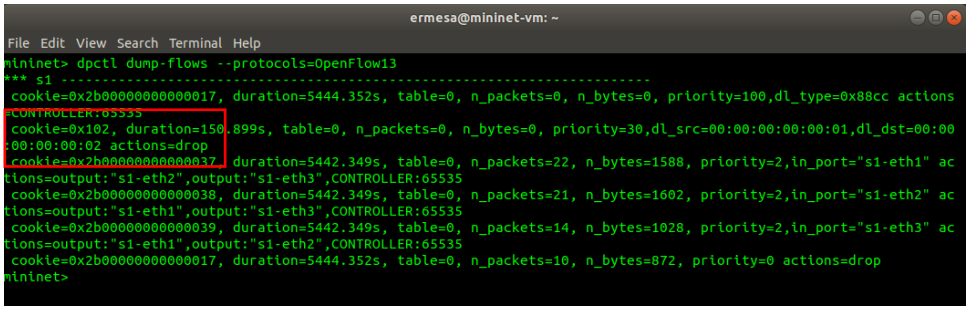
```
{
  "flow": [
    {
      "table_id": "0",
      "id": "102",
      "priority": "30",
      "cookie": "0x102",
      "match": {
        "ethernet-match": {
          "ethernet-source": {
            "address": "00:00:00:00:00:01"
          },
          "ethernet-destination": {
            "address": "00:00:00:00:00:02"
          }
        }
      },
      "instructions": {
        "instruction": [
          {
            "order": 0,
            "apply-actions": {
              "action": [
                {
                  "order": 0,
                  "drop-action": {}
                }
              ]
            }
          }
        ]
      }
    }
  ]
}
```


After performing a *Send Request,* we can notice that there is one *Success* action in the flow rule management section:





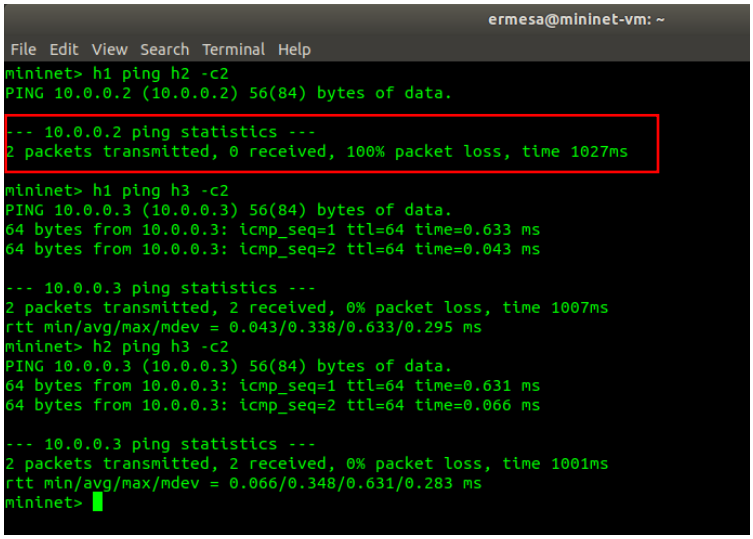
Now we can check the flow table in *Mininet* looking for the flow with cookie 0x102 which was defined in the previous steps:



As expected, **flows from h1 to h2 are being dropped**:

- h1 ping h2 failed
- h1 ping h3 succeeded
- h2 ping h3 succeeded

As we created a flow entry to manage the flow from h1 to h2 and we purposely dropped the flow with source h1 and destination h2; as a result, the ping fails.



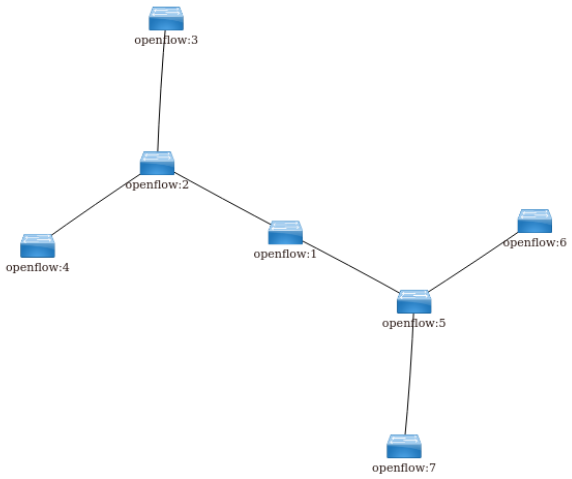
We can also create more elaborated topologies to investigate and manage them using ODL and OFM. For example:

To create a tree topology:

sudo mn --mac --topo=tree,3 --controller=remote,ip=127.0.0.1,port=6633 --switch ovs,protocols=OpenFlow13

```
ermesa@mininet-vm:~$ sudo mn --mac --topo=tree,3 --controller=remote,ip=127.0.0.1,port=6633 --switch ovs,protocols=O
penFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6) (s5, s7) (s6, h5) (s6, h6) (s7, h7)
 (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet>
```

Before simulating any traffic:

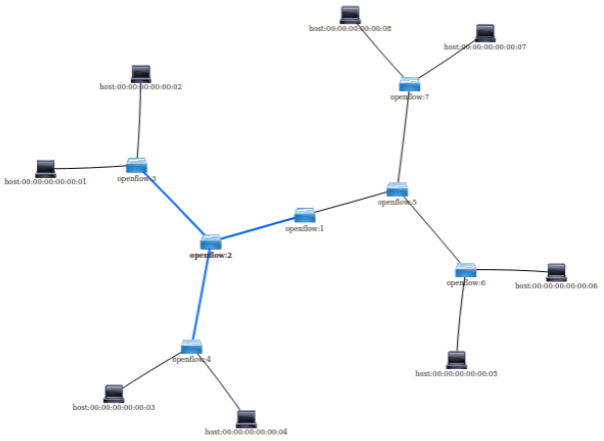


Now we can check the flow table; it shows only the switches communication with the controller to keep alive the communication:

```
ermesa@mininet-vm: ~
File  Edit  View  Search  Terminal  Help
*** s1 ------------------------------------------------------------------------
 cookie=0x2b00000000000018, duration=489.936s, table=0, n_packets=198, n_bytes=16830, priority=100,dl_type=0x88cc ac
tions=CONTROLLER:65535
 cookie=0x102, duration=489.951s, table=0, n_packets=0, n_bytes=0, priority=30,dl_src=00:00:00:00:00:01,dl_dst=00:00
:00:00:00:02 actions=drop
 cookie=0x2b00000000000046, duration=485.913s, table=0, n_packets=65, n_bytes=7597, priority=2,in_port="s1-eth1" act
ions=output:"s1-eth2",CONTROLLER:65535
 cookie=0x2b00000000000047, duration=485.913s, table=0, n_packets=66, n_bytes=7534, priority=2,in_port="s1-eth2" act
ions=output:"s1-eth1",CONTROLLER:65535
 cookie=0x2b00000000000018, duration=489.934s, table=0, n_packets=25, n_bytes=3977, priority=0 actions=drop
*** s2 ------------------------------------------------------------------------
 cookie=0x2b0000000000001c, duration=489.724s, table=0, n_packets=294, n_bytes=24990, priority=100,dl_type=0x88cc ac
tions=CONTROLLER:65535
 cookie=0x2b00000000000048, duration=485.917s, table=0, n_packets=26, n_bytes=2880, priority=2,in_port="s2-eth1" act
ions=output:"s2-eth2",output:"s2-eth3",CONTROLLER:65535
 cookie=0x2b00000000000049, duration=485.917s, table=0, n_packets=25, n_bytes=2677, priority=2,in_port="s2-eth2" act
ions=output:"s2-eth1",output:"s2-eth3",CONTROLLER:65535
 cookie=0x2b0000000000004a, duration=485.917s, table=0, n_packets=80, n_bytes=9574, priority=2,in_port="s2-eth3" act
ions=output:"s2-eth1",output:"s2-eth2",CONTROLLER:65535
 cookie=0x2b0000000000001c, duration=489.722s, table=0, n_packets=37, n_bytes=5864, priority=0 actions=drop
*** s3 ------------------------------------------------------------------------
 cookie=0x2b00000000000019, duration=489.914s, table=0, n_packets=99, n_bytes=8415, priority=100,dl_type=0x88cc acti
ons=CONTROLLER:65535
 cookie=0x2b0000000000004b, duration=485.921s, table=0, n_packets=6, n_bytes=420, priority=2,in_port="s3-eth1" actio
ns=output:"s3-eth2",output:"s3-eth3",CONTROLLER:65535
 cookie=0x2b0000000000004c, duration=485.920s, table=0, n_packets=6, n_bytes=420, priority=2,in_port="s3-eth2" actio
```

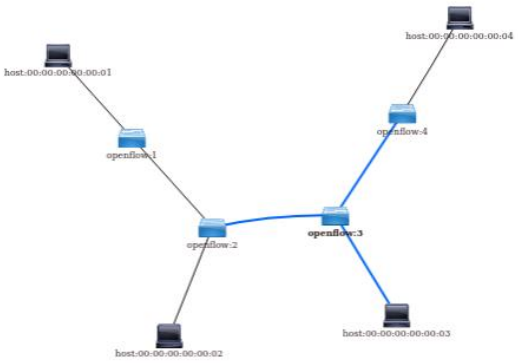
Pingall result in *Mininet* console and *ODL GUI*:

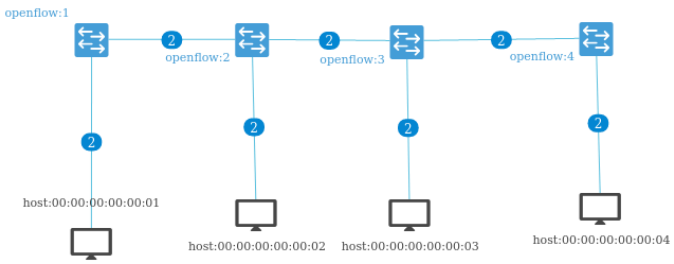
```
File  Edit  View  Search  Terminal  Help
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet>
```



To create a **Linear topology**:

```
ermesa@mininet-vm:~$ sudo mn --mac --topo=linear,4 --controller=remote,ip=127.0.0.1,port=6633 --switch ovs,protocols
=OpenFlow13
[sudo] password for ermesa:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s2, s1) (s3, s2) (s4, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```



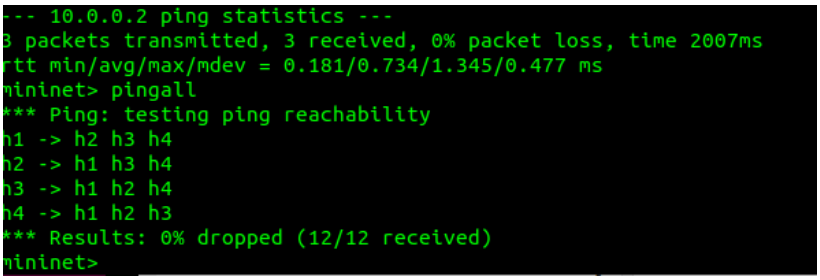



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 h4
h2 -> X h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 16% dropped (10/12 received)
mininet>
```

As we can see **the ping between h1 and h2 still fail because the entry flow created previously**:

	[id:102, table:0]	102	0	openflow:1	Open vSwitch	None	ON DEVICE

Once removed the entry flow the ping is successful:



This basic test demonstrates how the simple introduction of dynamic rules can modify the flow and facilitate network management to improve performance within a network environment significantly. The *ODL* controller is programmed to adapt to the dynamism of the requests that reach the network, simplifying and centralising the management of the same, allowing at the same time a rapid evolution thus guaranteeing a development to speed of the software. Furthermore, the examples in the report shown how the use of the web interface allows to interact with the ODL controller simply and rapidly and how the introduction of the REST API, available to network applications, have made it possible for new services and new functions to be implemented without the need to intervene on each network device manually.

Conclusions

The choice and use of a structure based on the SDN paradigm compatible with the OpenFlow protocol makes network, in economic terms, a competitive advantage. SDN technology offers a substantial improvement in bandwidth performance, a dynamic adaptation to the configuration of applications and commercial needs, all reducing the complexity of maintenance operations.

The tests have shown how it is possible to efficiently use the new SDN architecture and the Open-Flow protocol dedicated to it. Everything is made possible thanks to the *Mininet* simulation software and the *OpenDaylight* controller, which allow to emulate a network prototype that works quickly and with a simple PC available. The evaluation of the network was made on reachability and performance tests which, although quite simple, open the possibility to much more sophisticated tests, which also simulate real applications, obtaining immediately usable results.

All this confirm SDN the best current technology to evolve modern networks to a higher level by lowering management time and costs and being business companies always looking to develop their services in order to make them competitive, this technology finds its applications in many scenario cases: multi-tenant data centre, Multiplayer Online Role-Playing Games, Internet of Things, Content Distribution Networks.

References

Bennett, R., 2015. *sdn-3layers.* [Online]
Available at: https://hightechforum.org/networks-on-demand-the-promise-of-software-defined-networking/sdn-3layers/
[Accessed 2020].

CiscoDevNet, 2014. *github.* [Online]
Available at: https://github.com/CiscoDevNet/OpenDaylight-Openflow-App
[Accessed 2020].

Curtis, J., 2018. *Basic network flows; OpenFlow as a datapath programming standard.* [Online]
Available at: https://slideplayer.com/slide/12205467/
[Accessed 2020].

flowgrammable, 2020. *OpenFlow.* [Online]
Available at: http://flowgrammable.org/sdn/openflow/
[Accessed 2020].

Foundation, O. N., 2016. *ONF SDN Evolution,* s.l.: Open Networking Foundation.

Jarraya, Y., 2014. A Survey and a Layered Taxonomy of Software-Defined Networking. *IEEE COMMUNICATION SURVEYS & TUTORIALS,* 6(4).

ManarJammal, 2014. Software defined networking: State of the art and research challenges. *Elsevier.*

Matteson, S., 2014. *Software Defined Networking: The Cisco approach.* [Online]
Available at: https://www.techrepublic.com/article/software-defined-networking-the-cisco-approach/
[Accessed 2020].

Mutai, J., 2019. *How to install Wireshark on Ubuntu 18.04 / Ubuntu 16.04 Desktop.* [Online]
Available at: https://computingforgeeks.com/how-to-install-wireshark-on-ubuntu-18-04-ubuntu-16-04-desktop/
[Accessed 2020].

opendaylight, 2018. *Installing OpenDaylight.* [Online]
Available at: https://docs.opendaylight.org/en/stable-magnesium/getting-started-guide/installing_opendaylight.html
[Accessed 2020].

opendaylight-tsdr, 2018. *TSDR HSQLDB Data Store Installation Guide.* [Online]
Available at: https://opendaylight-tsdr.readthedocs.io/en/latest/hsqldb-install.html
[Accessed 2020].

Team, M., 2018. *mininet.* [Online]
Available at: http://mininet.org/download/
[Accessed 2020].

Appendix

Mininet Appendix

To install Mininet, mn and python API, directly on a computer with Linux operating system Ubuntu 18.04 we need to run the following code from the terminal:

sudo apt-get install mininet

After this completes, we need to deactivate *openvswitch-controller* if it is installed and/or running: *sudo service openvswitch-controller stop*

sudo update-rc.d openvswitch-controller disable

Then we can test Mininet:

sudo mn --test pingall

If Mininet complains that Open vSwitch is not working, we may need to rebuild its kernel module:

sudo dpkg-reconfigure openvswitch-datapath-dkms

sudo service openflow-switch restart

From the test performed, we have resumed the main advantages of using such technology:

- Speed: starting a simple network takes a few seconds.
- Create customised topologies: single or multi-switch.
- Packet forwarding: the functionalities tested on them can also be transferred to real switches.
- Mininet portability: can be run on Linux machines or a Virtual Machine.
- Ease of use: create simple or more complex experiments in Python conscript.
- Continuous growth of Mininet

Installation Wireshark

To install *Wireshark* stable release on Ubuntu 18.04, we add PPA repository and after we install Wireshark using (Mutai, 2019):

sudo add-apt-repository ppa:wireshark-dev/stable

sudo apt update

sudo apt -y install Wireshark

OpenDaylight

The ODL controller is a Java Virtual Machine (JVM) platform, so it can be installed and run on any hardware platform and operating system if it supports Java. We can interact with the controller through the *karaf* console, which can be started by executing the following instructions (opendaylight, 2018):

cd karaf-0.8.4

./bin/karaf

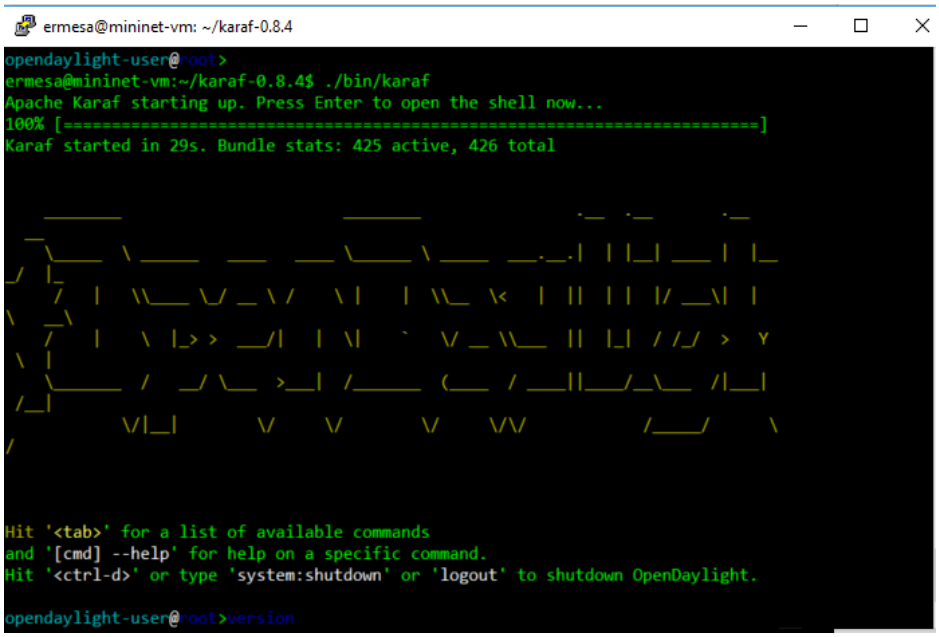


Figure 18 ODL Console via SSH

ODL Features are services and/or protocols that can be included in the controller software platform. Once the *karaf* console is started, we can view the complete list of features in alphabetical order by typing: feature: *list feature: list –installed*.

The DLUX feature provides a web-based graphical interface made available by ODL to simplify the overall vision and network management via the controller. The function can be installed in the ODL controller by typing the following instructions from the *karaf* console:

- *odl-dlux-core*
- *odl-dluxapps-nodes*
- *odl-dluxapps-topology*
- *odl-dluxapps-yangui*
- *odl-dluxapps-yangvisualizer*
- *odl-dluxapps-yangman*

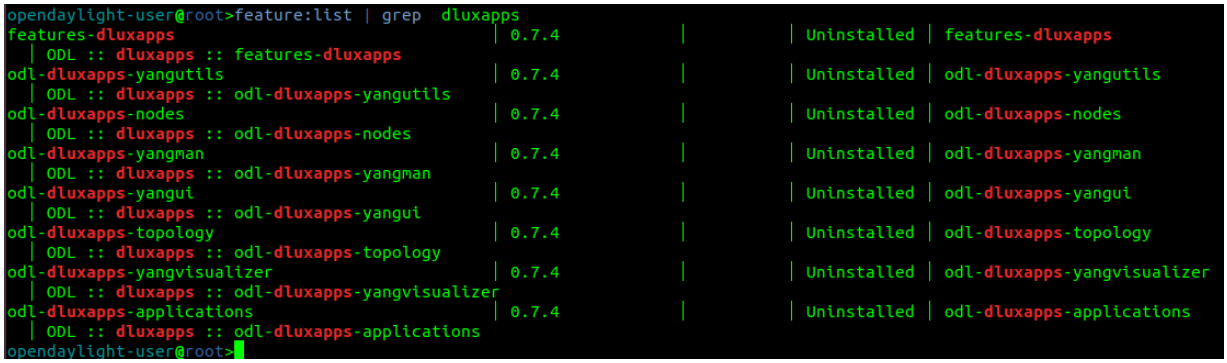
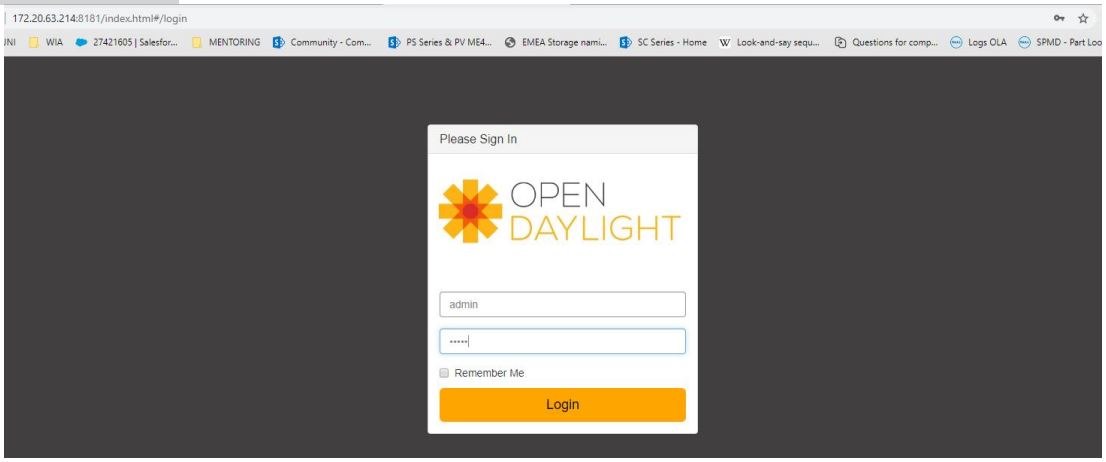


Figure 19 Features installed

The DLUX graphical interface collects, from the ODL database, information regarding the network topology, flows statistics and nodes characteristics and then displays them in a clear, simple and intuitive way. To access the DLUX interface, open a browser and enter the URL and login credentials:

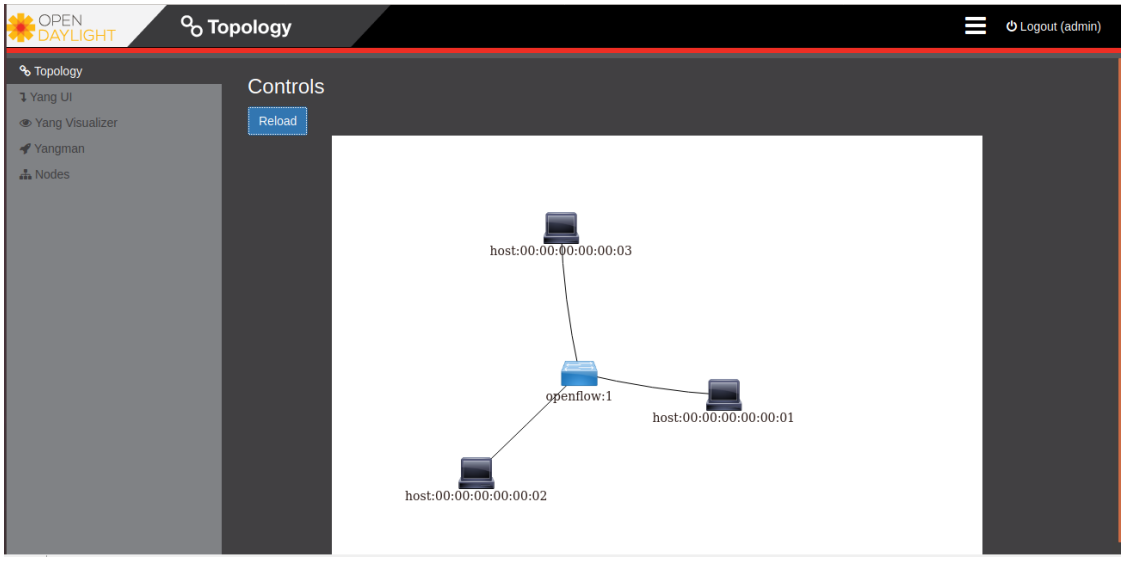
URL: http://<controllerIP>:8181/index.html
USER: admin

PASSWORD: admin



Topology

In the section, it is possible to display a graphic representation of the network on which we are working. Besides, by positioning the mouse on a host, a link or a switch, it is possible to view the source and destination ports details of that particular element. The benefits of a remote controller is also that through the DLUX graphical interface we can view the representation graph of the virtual network generated in the previous section and analysed through Wireshark: http://localhost:8181/index.html#/topology to investigate the SDN created infrastructure easily.



Nodes

The section allows to view information about the nodes and ports of the switches in the network. Besides, this section also allows to view the related statistics

Node Id	Node Name	Node Connectors	Statistics
openflow:1	None	4	Flows | Node Connectors

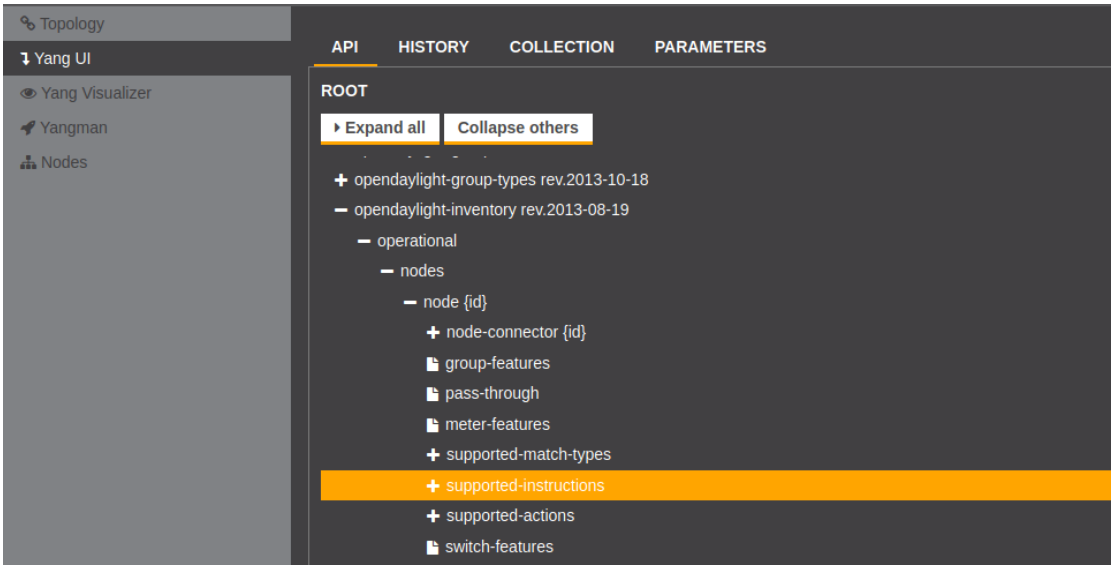
Figure 20 List switches belonging to the network

Node Connector Id	Name	Port Number	Mac Address
openflow:1:1	s1-eth1	1	b6:92:42:fc:fa:f5
openflow:1:2	s1-eth2	2	e2:eb:a4:f3:c3:4b
openflow:1:LOCAL	s1	4294967294	a2:a1:70:ce:90:4a
openflow:1:3	s1-eth3	3	4a:a9:97:20:e8:7f

Figure 21 Switch interfaces information

Yangui

In this section, it is possible to display a graphical interface through which it is possible to easily interact with the controller, through the APIs made available by ODL, to obtain, modify and delete resources.



L2Switch

The L2Switch feature enables the learning switch function in the network devices monitored by the ODL controller. The learning switch function is the process by which the switch manages the routing of the packet by associating the MAC address with the port of origin.

To install the L2switch function, we enter the following command in the *karaf* console:

feature: install odl-l2switch-switch

The operation is simple: when the switch receives a packet of which it does not know the destination, or it finds no association between the destination and a record contained in the flow tables, it sends the packet to the controller through the OpenFlow protocol. The controller receives the packet from the switch that has no feedback in the flow table, processes it and decides whether to add the relative flow entry or discard the packet. **Through this function, network traffic is optimised and speeded up**.

Time Series Data Repository

Time Series Data Repository (TSDR) is a framework for the collection, archiving, maintenance and interrogation of Time Series Data in ODL. TSDR has the task of collecting various temporal data and storing the *information* obtained in a TSDR datastore.

A series of data arranged in chronological order (timestamps) that express the dynamics of a certain phenomenon over time. **This type of data gives the user a real-time view of the phenomenon and predicts its future performance**. To enable the TSDR framework function in the ODL controller, type the following statement:

feature: install odl-tsdr-core

TSDR also offers the possibility of collecting information relating to the OpenFlow protocol. through the following feature *feature:install odl-tsdr-openflow-statistics-collector*

Finally, to obtain information, we can query the TSDR database through simple queries made up as follows: *tsdr: list <category>*

Where <category> corresponds to one of the following:

- *FLOWSTATS*: collects information relating to specific flows.
- *PORTSTATS*: collects data related to each switch interface.
- *FLOWTABLESTATS*: collects data relating to certain flow tables.
- *NETFLOW*: collects information related to the NetFlow protocol.