A Web Application for Automatically Generating the Network Infrastructure for Software Defined Networks on a Cloud

Álvaro Jarrín, Eng.[1], Iván Bernal, Ph.D.[1], and David Mejía, M.Sc.[1]
[1] Escuela Politécnica Nacional, Ecuador, {alvaro.jarrin, ivan.bernal, david.mejia}@epn.edu.ec

***Abstract*– This paper presents the implementation of a cloud that is used for instantiating virtual machines with a set of tools typically used for configuring and simulating networks based on Software Defined Networking (SDN). In terms of infrastructure, a web application is developed and included in each instantiated virtual machine so that even a user with minimal knowledge about SDN can create simple networks to start experimenting with this new network architecture. Users interact with the web application through a web interface that allows to select a topology, numbers of hosts and switches and choose from a given set of SDN controllers (POX, RYU and Pyretic). Some results obtained with the cloud, web application and the created SDN are presented.**

***Keywords*-- cloud computing; OpenStack; Software Defined Networking; web application.**

I. Introduction

Nowadays, computer networks have become a basic need to establish and facilitate communication, this requires constant development in this field to meet and improve the offered services and thus meet the needs of users. In this process two concepts, cloud computing and SDN (Software Defined Networking), have become disruptive forces in the way that networks are implemented.

Cloud computing [1] proposes to centralize computing resources and offer them as a service through Internet; this concept according to NIST (National Institute of Standards and Technology) must have five essential characteristics [2]: 1) Self-service on demand which allows users to increase or decrease the contracted services but minimizing interactions with providers; 2) High bandwidth for taking advantage of offered services; 3) Adequate computational resources so the cloud service provider ensures a good performance of the offered services; 4) Quick scalability requires enough infrastructure to allow rapid increase in the offered resources to clients; 5) Measurement of services so that there exists mechanisms for pricing of services considering that they are used on demand

SDN [3] is a network architecture that separates the control plane from the forwarding plane, allowing the control plane to become fully programmable and can be run in a computer (controller). This idea allows to create an architecture where a controller specifies how different traffic flows are handled and networking devices (switches) perform switching fulfilling policies specified by the controller. Controllers and switches communicate by means of a protocol such as Openflow.

For introducing SDN concepts to students, a hands-on approach becomes remarkable useful so that they can experiment with these new concepts. A ready to use SDN environment which can be provided to students would be very welcome and even better if the resources can be provided for each student on an individual basis. We propose to fulfill these objectives by combining the ideas that both controllers and switches for SDN can be virtualized and that a cloud can provide infrastructure as a service (IaaS). A virtual machine (VM) with a set of tools for structuring totally virtualized SDN is provided as an image for creating as many instances as required only limited by the available cloud resources. Once the VM is up and running, a user can interact with a web application that we developed and is included within the VM, the application generates a Python script that is run and creates an SDN given the topology and controller platform chosen by the user through a web interface.

The remaining of the paper is organized as follows: Section II outlines some useful details about cloud computing. Section III presents some aspects regarding OpenStack, including its main services. Section IV and V outline some aspects related to the design and implementation of the cloud and the web application, respectively. Section VI presents some results obtained from running tests with the cloud and web application. Finally, conclusions are outlined in Section VII.

II. Cloud Computing

A. Cloud Computing Service Models

1) SaaS (Software as a Service service) model: seeks to eliminate the need to install and run a particular software on personal computers and proposes to guarantee access to the software through a connection to the network.

2) PaaS (Platform as a Service) model: is a group of services that abstracts operating systems, middleware and configuration details, and gives developers the ability to provision, develop, design, test and implement applications.

3) IaaS (Infrastructure as a Service) model: provides access to computing resources remotely through a connection to the network which is usually the Internet. In the case of IaaS, the offered computing resources consist of virtualized hardware. This model enables customers to create cost-effective and easy-to-extend computing solutions in which all

Digital Object Identifier (DOI): http://dx.doi.org/10.18687/LACCEI2018.1.1.235
ISBN: 978-0-9993443-1-6
ISSN: 2414-6390

the complexity and cost associated with hardware management are left to the service provider. If the scale or volume of business activity of the customer fluctuates, or if the company plans to grow, it can use the computing resources delivered through the network without having to acquire, install and integrate hardware on its own.

B. Implementation Models

These models differ basically in the location of the services, since these can be of public access through the Internet or private access which means that services can be accessed only from a private network. Other aspects to consider when selecting the cloud deployment model are the cost of investment, information confidentiality, and services to be deployed. Fig. 1 presents the different implementation models and a short explanation of each one follows

1) Public Cloud: A cloud is called public [4] when the infrastructure and logical resources are owned and managed by the service provider and are part of the environment that is available to the general public; users only hire the services that they need which can be accessed via the Internet. The model is usually "pay per use", this means that only consumed services are paid.

2) Private Cloud: in this model the infrastructure is managed only by an organization. Private clouds [5] are usually implemented by large companies because of the high degree of knowledge and commitment required to virtualize the entire business environment. This model has advantages like the simplification of the administration of the applications, access control to the applications and reduction of the costs of the required licenses.

3) Community Cloud: this model is aimed at organizations that employ similar computing resources, which can be shared for reducing costs in infrastructure implementation or for sharing and accessing information that belongs to each of the organizations and decide to work together [6].

4) Hybrid Cloud: this model is formed by the combination of public and private clouds; the hybrid cloud [7] has the advantage of keeping the two models available to the organization, so they can be used according to the needs and resources. With this model, organizations can handle different architectures for the area of information technology, can rely on servers in the private cloud and use applications in the public cloud or can hire physical servers in data centers and use services in public and private clouds.

III. OpenStack

OpenStack [8], [9] is software that controls large pools of computation, storage and networking resources via a set of APIs (Application Programming Interface) or web portals. The services that OpenStack handles are mentioned below. It is important to note that the services and components used to create a cloud depend on the requirements and services the cloud will provide. For this paper, the Juno version was used.

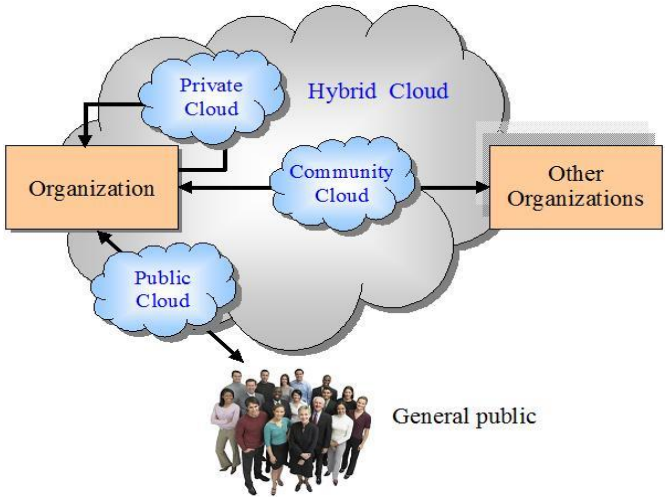

Fig. 1. Cloud deployment models [10]

A. Basic Services

The basic services consist of: compute (Nova), dashboard (Horizon) and networking (Neutron). Nova manages the life cycle and hosts the created instances in the OpenStack environment; this service is responsible for loading and allocating resources to each instance in the cloud. Horizon, is a web portal that allows to interact and to manage the OpenStack services through a graphical interface. Neutron allows the creation and administration of the network that interconnects the instances and the networking virtual devices.

B. Storage Services

There are two elements that conforms the storage services; object storage (Swift) and block storage (Cinder). Swift stores and retrieves unstructured objects through an HTTP API based on RESTful, while Cinder provides block storage for instances that are running and has a driver to facilitate handling and creation of block storage.

C. Shared Services

The shared services are conformed by: image service (Glance), identity service (Keystone) and telemetry (Ceilometer). Glance stores and manages the images of the disks of the virtual machines (VM). OpenStack Compute uses Glance when instantiating VMs. Keystone provides the authentication and authorization services for the other OpenStack services; besides, it has a catalogue of endpoints for all OpenStack services. This service manages the profiles of the services to guarantee that they have the required resources for correct functioning. Finally, Ceilometer monitors and manages OpenStack metrics for billing, scalability and service statistics.

D. Services at higher levels

At higher levels, we can find the following services: orchestration (Heat) and database service (Trove).

Orchestration mission is to create a human and machine-accessible service for managing the entire lifecycle of infrastructure and applications within clouds. Heat implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code, this service allows to use different cloud applications using HOT (Heat Orchestration Template) format or the AWS (Amazon Web Services) CloudFormation template formats. Heat provides both an OpenStack-native REST API and a CloudFormation-compatible Query API. Trove provides a database as a service, which is scalable and secure. It is designed for relational and non-relational database engines.

IV. Implementation of the Cloud

A. Multi-node cloud

OpenStack has a set of alternatives for its implementation, depending on users' needs. We selected to use a multi-node model which includes the controller, compute and network nodes. Each of the nodes fulfils a specific function and has a set of services and components that must interact with each other to obtain the desired result.

The controller node is the brain of this model considering that it hosts services such as: the message system server, the database, the identity service and the storage for images. These services have a direct and continuous relationship with the services of all the other nodes. The network node is responsible for managing the network services and that all OpenStack services and their respective components are connected to the network. The compute node hosts and manages the instances; it is necessary to install several components that allow the interaction between this node and the services of authentication, images, the image server, the graphic interface and the database. Table I lists the OpenStack services that are hosted by each of the nodes.

B. Environment

An HP ProLiant DL360e Gen8 server was used which includes an E5-2403v2 Intel Xeon processor (4 cores, 1.8 GHz, 10MB cache), 8GB of RAM and a 1TB hard disk. ESXi was installed directly on the hardware of the server as the hypervisor. On the virtualized environment three virtual machines are created for hosting the roles of the OpenStack nodes described above.

The available hardware resources were allocated as described in Table II. The controller node needs a good amount of RAM since it hosts several critic services so 3GB of the available 8GB are allocated; in terms of storage, the controller requires considerable disk space since it hosts the image service. Since the compute node hosts and manages the instances, 3GB of the remaining 5GB of RAM were allocated to this node. Finally, for the network node, the remaining 2GB are allocated to this node.

TABLE I
COMPONENTS INSTALLED ON EACH NODE

Node	Components
Controller	Identity Service Network Time Protocol (NTP) Image Service SQL database service Message Queue Dashboard Network Administration Computing Administration ML2 network plug-in
Compute	KVM Hypervisor Open vSwitch Computing Service Open vSwitch networking agent ML2 network plug-in
Network	Open vSwith DHCP Agent Networking metadata agent ML2 network plug-in Open vSwitch networking agent L3 networking agent

TABLE II
CHARACTERISTICS ON EACH NODE

Node	Type	Value
Controller	Operative System	Ubuntu 14.04 LTS
	RAM	3GB
	Hard disk	150GB
	Network interfaces	1 x 10/100 Mbps
Compute	Operative System	Ubuntu 14.04 LTS
	RAM	3GB
	Hard disk	150GB
	Network interfaces	2 x 10/100 Mbps
Network	Operative System	Ubuntu 14.04 LTS
	RAM	2GB
	Hard disk	150GB
	Network interfaces	3 x 10/100 Mbps

The number of network interfaces in each node is defined by the OpenStack multi-node architecture being used. In the controller node, a network interface is required to be connected to the management network; in the compute node two network interfaces are needed, one connected to the management network and the other to the network for tunnels; the network node has three network interfaces to have connection to the management network, tunnel network and external network. Every node employs Ubuntu 14.04 LTS as the operating system.

At first, the configuration of the network environment must be addressed since it will be used for installing OpenStack. The management network is created for the communication and interaction of the OpenStack services that are housed in the different nodes; the tunnel network is needed for the connection between the network and compute nodes so that the traffic that is generated by instances in the compute node can get to the outside world through the network node; the external network is connected to the network node so that

the OpenStack environment has an Internet connection. Fig. 2 shows the network diagram used.

Configuring and installing OpenStack services require keeping certain order since several services are required by others to work properly. The first service that must be installed is the Keystone since it handles permissions for users and keeps record of installed services and their location; then Glance is installed followed by Nova, since Nova depends on Glance; Neutron is installed next and finally Horizon is installed.

In every service, several common configurations must be performed such as setting the connection string to the database, specify the location of the services to be used and register the service itself and its location with Keystone. Additionally, each service has specific configurations for its correct operation.

V. Web Application

A. Functionality of the web application

The web application is designed so that a beginner on SDN concepts can start working without a broad knowledge on this subject since dealing with SDN, in many cases, may require to interact with terminals and handle commands to generate the desired topology.

The graphical interface aims to allow the user to generate the python script for creating an SDN topology without having to manipulate commands and it also presents the result in a graphical representation.

The virtual machine that is hosting the web application employs Xubuntu as the operating system given that it offers good performance and low hardware requirements, it is based on Ubuntu.

The VM has as preinstalled tools those necessary for the implementation of the infrastructure for an SDN, such as: Mininet (network simulator), Open vSwitch (virtual switches with Openflow support) and SDN controllers (POX, RYU and Pyretic).

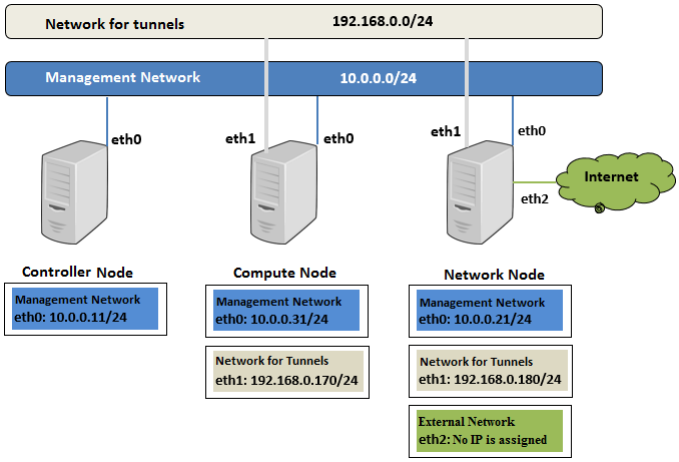

Fig. 2 Network diagram

The web application allows to define tree different parameters of the SDN: topology, number of hosts and SDN controller; the switch used on any case is Open vSwitch.

1) Supported topologies: The network topology is the physical and logical distribution of the devices in a network, the web application presents two alternatives: tree and simple. Tree topology has a "root" switch from which "branches" come out, each branch continues to expand generating new branches until reaching the hosts. Two parameters must be specified: depth, defines how many levels there should be from the main switch to the hosts; and fanout, defines how many devices are connected to each of the branches. The total number of hosts is defined by the depth and fanout parameters. Fig. 3a shows the representation of the tree topology with depth=2 and fanout=3 generated by the web application. Simple topology has only one level, the hosts are connected directly to the switch and the switch to the controller, so only the number of hosts must be specified. Fig. 3b shows a topology with 4 hosts.

2) Number of Hosts: this parameter specifies the number of hosts that will be connected to the switch in the simple topology but it is disabled when the tree topology is selected.

3) SDN Controller: It has a global view of the overall network and it establishes the policies to handle the traffic that enters and leaves every switch. Three frameworks (POX, Ryu and Pyretic) are presented as options in the web application and should be selected according to requirements such as programming language and compatibility with specific devices. POX offers functionalities and services to easy up the development of new modules for implementing controllers using Python. Ryu is a component-based framework with well-defined APIs that simplify the process of creating SDN controllers, it is also based on Python and supports several versions of Openflow. Pyretic is a member of the Frenetic family of SDN programming languages, and allows programmers to develop modular applications providing powerful high level abstractions.

B. Implementation of the web application

For the implementation of the web application, the Django framework is used. Django offers tools that facilitate the development of the web application. The development is defined in two stages; the first one corresponds to the frontend that is what the user sees and allows users to interact with the web application; the second one is the backend that is where all the logic of the web application is located.

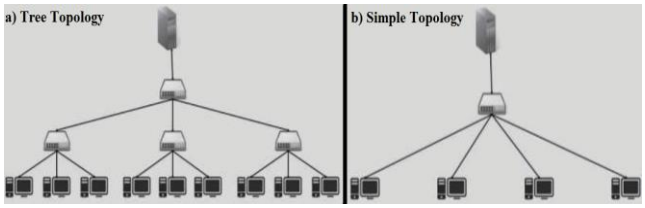

Fig. 3 a) Tree topology with fanout=3 and depth=2 b) Simple topology

1) Frontend: It is developed using three languages: HTML (HyperText Markup Language), which allows defining the content of the web application; CSS (Cascading Style Sheets), which allows styling the content; and, JavaScript defines the functionality of the web application. For example, when the user selects an option from an element of the graphical interface, the JavaScript code determines which value the user has selected using an if statement and picks the image to be displayed to represent the corresponding topology (See Fig. 4).

2) Backend: It is developed using Python for specifying the processes necessary for automating the creation of the SDN infrastructure. Fig. 5 and Fig. 6 show two sections of the generated script. Fig. 5 shows the Python code that performs the automatic initialization of the selected controller, while Fig. 6 shows the code that creates the SDN infrastructure in Mininet, performs a connectivity test on all the hosts and destroys the topology.

VI. Results

Functional tests verify that the entire created environment is operating properly. The tests were divided into three stages.

A. Testing the operation of the cloud

Before verifying the operation of the cloud, it must be confirmed that OpenStack services are all enabled and working and then it is checked which images are available and which instances have been created in the cloud.

Finally, the state of resources of the cloud is checked. An instance of the VM running in the cloud is shown in Fig.7 which sums up that all the aforementioned tests passed and the instance is working.

```
if (valor == "simple") {

    switch(document.getElementById('select_2').value) {
    case "1":
        $('#btn_detener_Topo').show();
        $('#img_controller').show();
        $('#img_switch').show();
        $('#img_host_1').show();
        $('#img_host_1').css('margin-left',"46%");

        //Se dibuja las lineas entre dispositivos
        jsPlumb.ready(function() {

                dibujarLinea("img_controller",'img_switch");
                dibujarLinea("img_switch","img_host_1")

        });
        jsPlumb.repaintEverything();

        break;
```

Fig. 4 Segment of JavaScript code

```
def crear_pox():
    os.chdir('/home/ubuntu/pox')
    subprocess.call('python pox.py log.level --DEBUG forwarding.tutorial_l2_hub &',shell=True)
```

Fig. 5 POX controller initialization

```
def topologia_simple(host):

    net = Mininet( topo=SingleSwitchTopo( host), controller=partial(RemoteController, ip='127.0.0.1', port=6633) )
    net.start()
    resultado=net.pingAll()
    net.stop()
    subprocess.call('sudo mn -c',shell=True)
    subprocess.call('sudo fuser -k 6633/tcp',shell=True)
    return resultado
```

Fig. 6 Code in Mininet

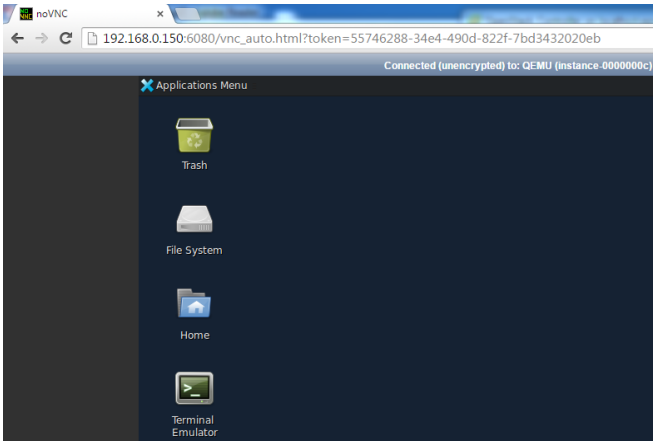

Fig. 7 Instance of the VM running in the cloud

B. Testing the creation of the SDN infrastructure

In the VM instance, it must be verified that every tool required for setting up the SDN infrastructure is working properly. This is accomplished by creating different topologies with every available controller. The result of one of the tests is presented in Fig.8, a simple topology with 4 switches and a remote POX controller is created in Mininet.

C. Testing the web application

First it is verified that the web server hosted in the instance of the VM is up and running and it allows to access the web application. Fig. 9 shows that the server is running and the first request is processed in the server side, while Fig. 10 presents the web page sent by the server to the initial request made by a web client.

As part of the tests, a simple topology with three hosts and Pyretic as the controller is created. Fig. 11 shows the graphical representation of the topology that the user configured.

```
ubuntu@sdnhubvm:~[17:57]$ sudo mn --topo single,4 --mac --controller remote --switch ovsk
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Fig. 8 Creating an SDN infrastructure with POX in Mininet

```
Django version 1.7.6, using settings 'interfaz_1.settings'
Starting development server at http://192.168.1.7:1234/
Quit the server with CONTROL-C.
[15/May/2015 07:52:52] "GET / HTTP/1.1" 200 10484
[15/May/2015 07:52:52] "GET /static/css/style.css HTTP/1.1" 304 0
[15/May/2015 07:52:53] "GET /static/css/bootstrap.min.css HTTP/1.1" 304 0
[15/May/2015 07:52:53] "GET /static/js/jquery-1.11.2.js HTTP/1.1" 304 0
[15/May/2015 07:52:53] "GET /static/js/bootstrap.min.js HTTP/1.1" 304 0
[15/May/2015 07:52:54] "GET /static/images/epn-logo2.png HTTP/1.1" 304 0
[15/May/2015 07:52:54] "GET /static/js/script.js HTTP/1.1" 304 0
[15/May/2015 07:52:54] "GET /static/js/jquery.jsPlumb-1.7.5-min.js HTTP/1.1" 304
 0
```

Fig. 9 Result of the initial request on the server

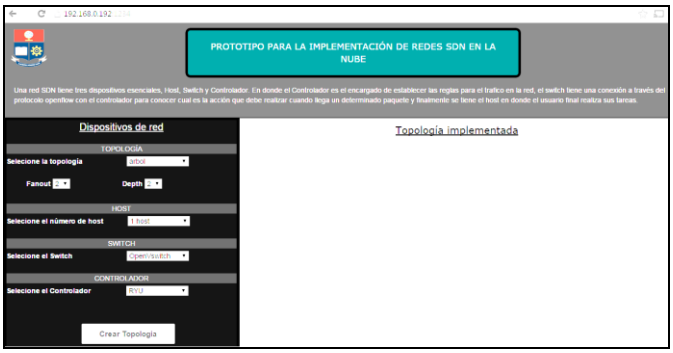

Fig 10. Initial view of the web application in the client side

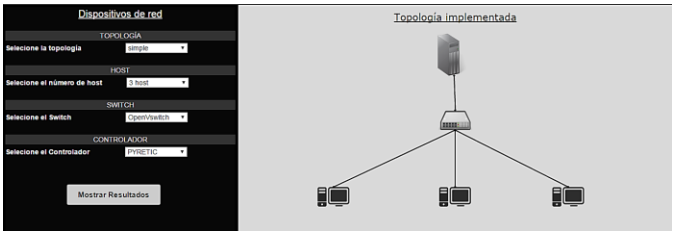

Fig. 11 Results of a simple topology with Pyretic

Fig. 12 shows the result presented by the web application after a user selected a tree topology with fannout=3 and depth=3. Fig. 13 shows the displayed result in the command line when the generated script is run; the switches, hosts and the controller are created and then the connectivity test is performed. The results of the connectivity tests will be displayed on the client side on a pop-up window.

VII. Conclusions

A web application is available that allows the creation and usage of SDN infrastructures in a VM instantiated on a cloud that offers the IaaS Model. For accomplishing this, a cloud was implemented using OpenStack with a multi-node model; an image was created and uploaded to the cloud holding the necessary tools for the creation of SDN infrastructure including a set of popular controllers (POX, RYU and Pyretic).

The hardware resources that were available for our implementation were limited for the multi-node environment so degradation in the performance of the VM instance was evident. From the point of view of the end user, there is a considerable waiting time when creating the controller, virtual switches, and hosts, interconnecting the devices to comply with the selected topology and finally simulating the network in Mininet. Currently, several servers a higher number of processors, RAM and storage are being bought and for sure this will enhance the performance of our system allowing to also have more instances.

It has been shown that it is feasible to implement whole networks based on the principles of SDN with all their elements virtualized (hosts, switches and controllers) on a cloud.

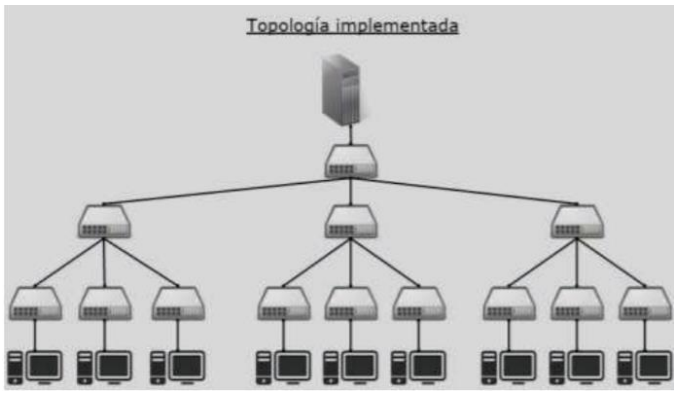

Fig 12. Results of a tree topolgy

```
controller.py
pyretic_topo
simple
3
argumw
Unable to contact the remote controller at 127.0.0.1:6633
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
Connected to pyretic frontend.
INFO:core:POX 0.5.0 (eel) is up.
*** Ping: testing ping reachability
h1 -> INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
X X
h2 -> h1 h3
h3 -> h1 h2
*** Results: 33% dropped (4/6 received)
INFO:openflow.of_01:[00-00-00-00-00-01 1] closed
6633/tcp:            6186
```

Fig. 13 Initialization of the controller and connectivity test on the server

References

[1] S. Patidar, D. Rane, P. Jain, "A Survey Paper on Cloud Computing", *Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on*, 2012.

[2] P. Mell, and T. Grance, "The NIST Definition of Cloud Computing", 2011, [Online] Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[3] D. Kreutz, et al., "Software-Defined Networking: A Comprehensive Survey", *Proceedings of the IEEE 103*, 14-76, 2014.

[4] A. Li, M. Liang, L. O'Brien, H. Zhang, "The Cloud's Cloudy Moment: A Systematic Survey of Public Cloud Service Outage", *International Journal of Cloud Computing and Services Science*, vol. 2, no. 5, 2013.

[5] I. Milošević and V. Šimović, "Comparison of private cloud infrastructure implementation models", *Information Technology Interfaces (ITI), Proceedings of the ITI 2012 34th International Conference on*, 2012.

[6] A. Marinos, G. Briscoe, "Community Cloud Computing". *Cloud Computing. CloudCom 2009. Lecture Notes in Computer Science*, vol 5931. Springer, 2009.

[7] S. Yan, et al., "Infrastructure management of hybrid cloud for enterprise users", *Systems and Virtualization Management (SVM), 2011 5th International DMTF Academic Alliance Workshop on*, 2011.

[8] M. T. Jones, "Cloud computing and storage with OpenStack: Discover the benefits of using the open source OpenStack IaaS cloud platform", *Developer Works*, 2012.

[9] 451 Research, "OpenStack in Support of Public Cloud", 2016.

[10] ONTSI, Cloud Computing Retos y Oportunidades, 2012. [Online] Available: http://www.ontsi.red.es/ontsi/sites/default/files/1_estudio_cloud_computing_retos_y_oportunidades_vdef.pdf.

[11] C. Ghribi. M. Mechtri, D. Zeghlache, "OpenStack Installation Guide for Ubuntu 14.04 Juno", [Online] Available: https://github.com/ChaimaGhribi/OpenStack-Juno-Installation/blob/master/OpenStack-Juno-Installation.rst