OpenDaylight and OpenNebula Integration: Testing Traffic Management

Omayma Belkadi, Alexandru Vulpe, *Member, IEEE*, Yassin Laaziz, and Simona Halunga, *Member, IEEE*

***Abstract* — Software-Defined Networking (SDN) and Cloud Computing are now two of the most adopted technologies, on which many organizations are working to enhance every day. For instance, SDN is particularly emerging to solve networking complexity in cloud data centers, so we see many attempts to integrate Network and Cloud Managers. In this paper, we address an integration of these two technologies, particularly a yet undiscussed combination of two popular frameworks: OpenNebula and OpenDaylight. These open source solutions are widely used for cloud management and network management, yet there are no developed modules for communication between the two. Therefore, we propose a simple way for OpenDaylight to manage OpenNebula's compute nodes, using a common component they both support: OpenvSwitch. We compared OpenNebula with the popular OpenStack cloud manager, as it is attracting more attention in both academia and industry, by evaluating some relevant time metrics and discussing the differences of the proposed technologies. Then, we deployed a test topology to conduct some traffic management techniques in this integration. Our results show that OpenNebula's deployment time as well as clean-up time is significantly lower than OpenStack, but OpenStack takes less time to the running state, besides proving the simplicity of traffic management in OpenNebula using OpenDaylight.**

***Keywords* — Software-Defined Networking, Cloud Computing, OpenNebula, OpenDaylight, Traffic management.**

I. Introduction

THE Software-Defined Networking (SDN) concept has been recently growing to address the complexity of networks. Separating Data and Control planes allows a better management and innovation in this field and makes it finally follow the ongoing change in the IT and computing industry. Going back to the central model, especially brought by the appearance of Cloud Computing, SDN came to change the domination of standard protocols and the tied control/data planes in each device, all to allow customizing and automating networks.

Similar to SDN, Cloud Computing came to change hardware and software infrastructures, by providing several services as storage and computing powers, without being tied to a certain hardware location or configuration.

With the rise of cloud computing, the SDN concept finds its interest, because of the enormous size and complexity of networks and hardware involved. However, integrating these two concepts still suffers many challenges, particularly in the field of open source software, where it is often necessary to show a bit of ingeniousness to find the right solutions to make new integrations.

One of the most important and mature projects in SDN contributions is OpenDaylight (ODL), which is a modular open source platform. On the other hand, cloud frameworks such as OpenNebula (OPNBL) and OpenStack (OPSTK), are largely used to build different cloud environments and are of great utility since some virtualization solutions are not affordable, or they rely on cloud providers with strict policies.

In a previous paper presented at the TELFOR conference [1], we focused on the Integration of ODL and OPNBL. This integration was not obvious as there are no modules in ODL or OPNBL to ensure the connection between the two, unlike the OPSTK case. This issue was resolved by using an intermediate open source solution that they both support which is OpenvSwitch (OvS). To the best of the authors' knowledge, this implementation has not been reported before.

Here we present an extended version of this work, including further analysis and validation of this integration, in terms of testing traffic management possibilities using ODL features. Therefore, we propose a use case topology and a set of flow rules as a demonstration.

The rest of the paper is organized as follows: Section 2 will briefly present the used open source solutions, and then in Section 3 we list the related work to our study. Section 4 will describe the setup made, while Section 5 explains the use case topology setup. We discuss the results in Section 6. Finally, we conclude the paper in Section 7.

Paper received May 22, 2020; accepted September 22, 2020. Date of publication December 25, 2020. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Grozdan Petrović.

This paper is revised and expanded version of the paper presented at the 27th Telecommunications Forum TELFOR 2019 [1].

This work was partially funded by a grant of the Romanian National Authority for Scientific Research and Innovation, CCCDI – UEFISCDI, project number ERANET-ERAGAS-ICT-AGRI3-FarmSusteinaBl-1, within PNCDI III and by a grant of the Ministry of Research and Innovation, UEFISCDI Romania, project no. 8Sol/2018 within PNCDI III.

Omayma Belkadi is with the Abdelmalek Essaadi University, National School of Applied Sciences, LabTIC, Tangier, Morocco (e-mail: belkadi.omayma@gmail.com).

Alexandru Vulpe is with the University Politehnica of Bucharest, Telecommunications Department, Bucharest, Romania (e-mail: alex.vulpe@radio.pub.ro).

Yassin Laaziz is with the Abdelmalek Essaadi University, National School of Applied Sciences, LabTIC, Tangier, Morocco (e-mail: ylaaziz@uae.ma).

Simona Halunga is with the University Politehnica of Bucharest, Telecommunications Department, Bucharest, Romania (e-mail: simona.halunga@upb.ro).

II. SDN AND CLOUD PLATFORMS

A. Opendaylight

ODL [2] is an open-source SDN controller, developed in 2013 at first as a collaboration between IBM and Cisco, then hosted under the Linux Foundation. The main goal from this project was having a tool to manage and deploy SDN and even Network Function Virtualization (NFV) solutions. By now, with a very active community, ODL has reached its 10th release, Neon, in March 2019, providing more enhanced use cases besides improved stability and scalability.

Furthermore, ODL is a combination of many integrated projects. In each release, board and chair members sponsoring the project decide adding, enhancing or removing a set of features and components. In this work, we rely specifically on low-level interfaces such as OpenFlow and OVSDB (OvS DataBase management protocol) during our setup.

The motivation is that OpenFlow is the protocol allowing communication between OpenDaylight and its network, while OVSDB is the southbound protocol to manage and define the schema for the OvS database to grant the communication between the SDN controller and the virtual switch. In addition to that, we use in our work ODL User Experience (DLUX) application feature, which provides a web user interface to visualize the network, its topology and connected nodes.

B. OpenNebula

OPNBL [3] is a modular system implementing many Cloud architectures and providing several services. It is an open source, highly scalable and advanced cloud computing manager. The Distributed Systems Architecture (DSA) Research Group launched in 2008 the first release of OPNBL. Due to its modularity, the project has been significantly enhanced by the recent version, 5.8.1 that we are using in this work. Not only that, but OPNBL is also easily integrated with many other solutions, different from other cloud platforms. It guarantees full interoperability with all existing components, avoiding vendor restrictions.

C. OpenvSwitch

Virtual machines networking was usually configured using Linux bridges, but it got challenging as they were not designed originally for virtual networking, and with the apparition of OpenFlow protocol, more limitations appeared. Therefore, OvS [4] was developed to resolve these issues. It is an open source virtual switch working in two modes, the first, normal mode; it handles by itself the switching and forwarding functionalities. While the second one, flow mode, which is the one we used during our setup, uses the flow table to decide the forwarding rules of packets. This flow table is mainly managed by the SDN Controller, and by it, control flows could be installed or removed to meet network needs with high automation and abstraction.

D. OpenStack

OPSTK [5] is a Cloud solution, written in Python, giving large pools of compute, storage and networking. Developed by NASA, it represents a set of multiple open source components, with a very active community and partners, which make it the most used solution to build private Cloud. The project is dedicated to massive infrastructures, allowing to plugging needed components, making it very flexible for use. These components are divided into about ten categories, respectively: Compute, Storage, Networking, Data and analytics, Security and compliance, Deployment, Management, Applications, and Monitoring. In each category, OPSTK maintains officially the basic components, such as:

- Nova in Compute: allows users to create, deploy and manage virtual machines, and supports many hypervisors like KVM, Hyper-V, VMware ESXi, and Linux Containers such as LXC,
- Swift and Cinder in Storage
- Neutron in Networking: provides ‘networking as a service’ capabilities, and Software-Defined Networking technology.

III. BACKGROUND & RELATED WORK

The only paper found tackling ODL and OPNBL platforms' integration was [6]. The writers argued in their description that the project would be based on the use of ODL as the network manager, specifically developing the OpenDOVE API [7]. Their purpose was to develop a framework (i.e. BEACON) enabling a federated cloud network, using SDN to connect the overlay networks. The BEACON architecture was intended to be based on open source solutions, as a start mentioned using ODL with OPNBL and/or OPSTK. However, at the time of writing there is no evidence of the BEACON framework integrating ODL with OPNBL.

There are several papers dealing with integration of ODL with OPSTK as well as comparison papers between OPNBL and OPSTK. In comparison works of these two platforms, run around 2013 [8] then later in 2016 [9], various aspects were taken into consideration, such as performance and flexibility. Yet, the two platforms by now have added and enhanced many services to resolve their former limitations. Authors in [10] implement their own cloud architecture on top of OPSTK and OPNBL and compare their performances. However, they use different hardware features, which makes their results unsound. Nevertheless, they find that OPSTK and OPNBL perform on par, one being better than the other in different scenarios. Other researchers analyse different SDN controllers and their integration with OPNSTK. Authors in [11] analysed 4 different SDN controllers and came to the conclusion that ODL performs the worst when integrated with OPSTK in terms of average latency but better in terms of TCP/UDP packet loss. A cloud testbed is proposed in [12] and authors aim to demonstrate it by taking into account a video application and measuring VM migration time and VM downtime, service continuity level and QoE of the VM-to-VM traffic. However, no results are being shown in the paper. Finally, traffic evaluation in an OPSTK-ODL integration [13] has been considered. Here, authors conclude that periodic synchronous plane traffic scales with the number of VMs and their respective network interfaces and VM-related events induce asynchronous traffic, with

the bandwidth also scaling according to the number of VMs and interfaces. The even distribution leads to the highest overall traffic volume.

IV. OpenDaylight and OpenNebula Integration

During this prior setup, we manage, using OvS, the interfaces created by OPNBL, and use SDN in its very known Controller ODL. Table 1 presents the versions of open source solutions we used. We intended to have the latest stable releases of each platform. Hence, as an exception for ODL, we used an anterior version during our work, i.e. Oxygen. This version is the latest release of ODL containing the DLUX Applications feature.

Table 1: Platforms versions

	Name	*Version*	*Release*
CLOUD	OPNBL	5.10.1	Dec 2019
SDN	ODL	Oxygen	Mar 2018
CONNECTOR	OvS	2.9.5	Apr 2019

DLUX provides a Graphical User Interface (GUI) that helps the ODL user to visually verify its network topology and connected devices, seeing the whole network components. Therefore, the figures in the paper are captured from it as a verification of a successful setup. In addition, we had to install a set of features in our ODL Controller, such as the OVSDB and OpenFlow plugins, which are mandatory to enable the communication between these solutions.

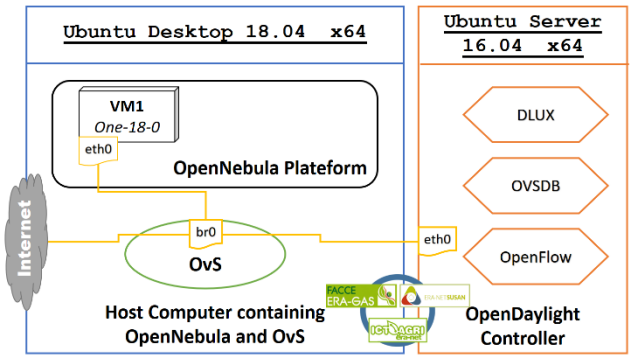

Fig. 1. Integration setup schema.

As Fig. 1 shows, our setup consists of two Linux machines. The first is an Ubuntu Desktop containing OPNBL platform and OvS, located in the lab network, and the second one is an Ubuntu Server hosting ODL Controller located in an exterior network. Therefore, the communication between these components is enabled throughout the OvS switch.

Hence, an OvS bridge, br0, was created in the lab network segment with the following commands:

```
#ovs-vsctl add-br br0
#ovs-vsctl add-port br0 eth0
```

This latter was configured as the network interface for OPNBL's compute nodes. Therefore, br0, is a bridge between the lab's and OPNBL's machines. Moreover, we created an Alpine [14] virtual machine in our cloud platform for testing, as can be seen in Fig. 2.

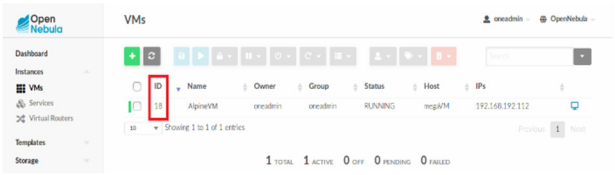

Fig. 2. OPNBL's User Interface.

We assign this whole network management to our SDN controller by the following command:

```
#ovs-vsctl set-controller br0 tcp:<server_ip>:6633
```

Then we add the next command line to enable all versions of OpenFlow in OvS, to be used with ODL;

```
#ovs-ofctl -O OpenFlow13 dump-flows br0
```

With this, ODL is now able to manage the network of the given Cloud through OvS.

To verify our setup, we verified if the created VM is listed in ODL nodes and topology files. In Fig. 3, we can see that ODL detects the Alpine machine as one of its network nodes (one-18-0 is the name given by OPNBL to its nodes in the network, where 18 represents the machine's creation ID, circled in Fig. 2).

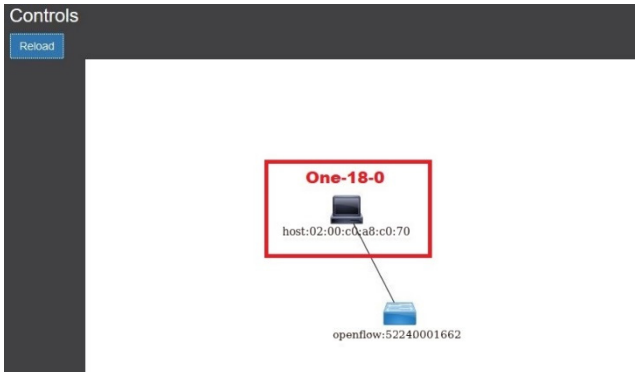

Fig. 3. OPNBL's node in ODL network.

Furthermore, we can see in Fig. 4 how the Alpine VM is appearing in the network topology, managed by the OvS switch (OpenFlow switch), besides the other running machines in the lab. Which means ODL has successfully detected this given environment, consequently it can manage this network, which is basically the Cloud's nodes, whether by traffic filtering, efficient load balancing, control data transmission delays, and other unavailable services that the Cloud platform doesn't provide by itself.

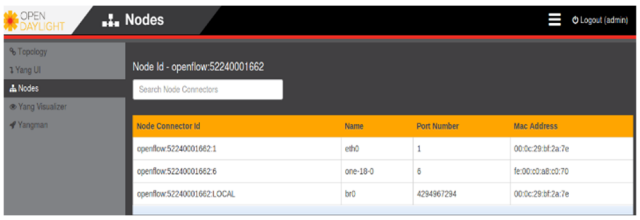

Fig. 4. ODL's nodes.

V. Traffic Management in OpenNebula using OpenDaylight

In this section, we demonstrate traffic management in OPNBL using ODL features. A proposed testbed has been set up to proceed this evaluation. It is composed of three Alpine machines, deployed in the OPNBL Cloud, this way we have a clean environment to manage the network flows using ODL.

A. System Model

For this setup, we use two servers containing ODL and OPNBL respectively, with the hardware specification mentioned in Table 2 below. Then followed the same steps explained in Section IV to configure their integration.

Table 2: Hardware Specification

	ODL	*OPNBL*
OS	Ubuntu Server	Ubuntu Desktop
Version	18.04	18.04
RAM	8G	8G
CPU	4 cores	4 cores
Storage	30G	100G

As mentioned before, ODL has different built-in plugins used to communicate with networking devices, like OVSDB, OpenFlow and NETCONF, to enable the management switching devices. While the DLUX GUI allows us to use ODL capabilities.

The ODL controller comes with many features; in this testbed we make use of the following:

- GUI and visualizing the topology:

```
feature:install odl-dlux-core odl-dluxapps-nodes odl-dluxapps-topology odl-dluxapps-yangui odl-dluxapps-yangvisualizer odl-dluxapps-yangman
```

- To enable REST interface requests

```
feature:install odl-restconf-all
```

- For OpenFlow and OvS plugins

```
feature:install odl-l2switch-switch-ui odl-ovsdb-hwvtepsouthbound-ui odl-ovsdb-southbound-impl-ui odl-openflowplugin-flow-services-ui
```

Furthermore, ODL has two methods for traffic applications, whether by using the provided REST APIs in the Controller, or programming their own application through MD-SAL internal service modules implementation. In this testbed, we selected the first one.

MD-SAL uses YANG models used by ODL YANG tools to generate Java-based APIs, which allows simplifying the development of traffic applications.

In addition to that, REST queries are the most used operations to fetch ODL managed networks, for this, YangUI, provided with the DLUX GUI features is used as the graphical REST client to build and send requests to ODL during our tests. We use it for network configurations, with the following main operations:

- GET: to get data from ODL
- POST: to send data to be saved in ODL
- DELETE: to send data to be deleted in ODL

B. Use case Topology

The proposed topology, presented in Fig. 6, includes all the connected elements, where we deployed three Virtual Machines as OPNBL's compute nodes (Fig. 5) running on Alpine Linux 3.11 operating system, each has 2 vCPUs, 1GB RAM and 5GB of storage, connected to the OvS network 10.0.2.0/24.

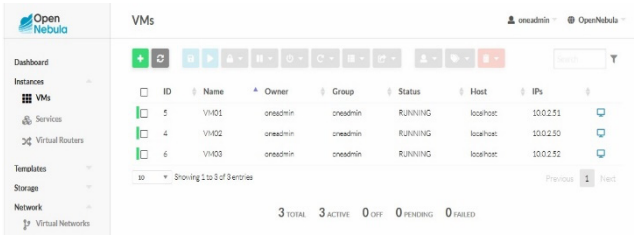

Fig. 5. OPNBL's testbed nodes.

ODL consists of five components as shown in Fig. 6, AAA (Authentication, Authorization and Accounting) enabling automatic identification, MD-SAL (Model-Driven Service Abstraction Layer) to unify data structures used by services, besides northbound/southbound APIs, finally southbound plugins (NETCONF, OVSDB, and OpenFlow).

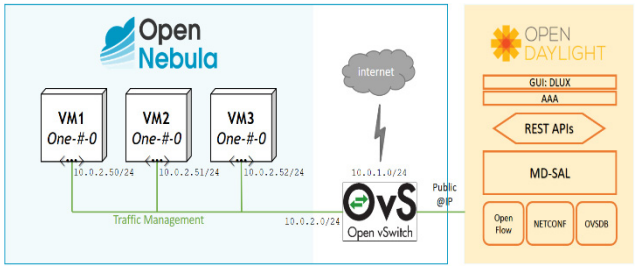

Fig. 6. Testbed topology schema.

Once the ODL controller is set on the OvS instance, OpenFlow messages of the connected VMs will be sent through it to ODL. Which we can view in detail on DLUX gui (Fig. 7).

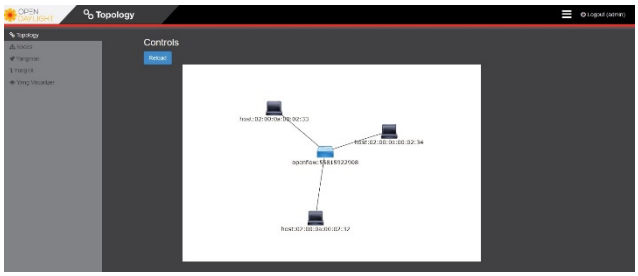

Fig. 7. Testbed topology in ODL.

Once the communication is established; ODL adds flows so the switches connected to it act like a learning switch, and the first flow table is called table 0.

VI. Results and Discussion

A. OPNBL Comparison with OPSTK

Table 3 below summarizes the differences between the two Cloud platforms collected from their documentation websites.

TABLE 3: OPNBL AND OPSTK PROPERTIES

	OPNBL	***OPSTK***
License	Apache 2.0	Apache 2.0
Min Hardware Requirements	CPU 2 cores RAM 2GB HD 100GB	CPU 4 cores RAM 4GB HD 20GB/node
Internal Organization	single integrated management	different sub-projects
Roadmap definition	managed by one organization	managed by vendors
Contributors	platform users	vendors' products
Cloud Model	Private, Public (Amazon EC2), Hybrid, Federated	Private, Public (APIs to Amazon EC2 and S3)
Access	Web UI and Console	Web UI and Console
Supported Virtualization	Xen, KVM, ESXi,VMWare ESX	Xen, KVM, ESXi,VMWare ESXXCP, QEMU, UML
Auto-scaling support	Available besides isolated clusters	Not available
select Storage Resources	Available	Not available

In addition, we ran an end-user comparison of these two platforms to evaluate their performance in terms of timing, which can be crucial in network automation. The process followed is mainly based on a regular usage such as creating a Virtual machine, studying how its deployment goes in terms of time and possible complications.

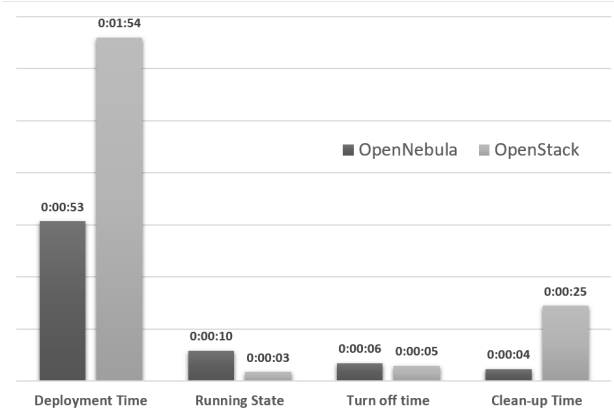

Fig. 8. OPNBL and OPSTK time metrics.

As Fig. 8 shows, we compared the deployment time, the time it takes a node to be in a running state, the time to be halted and finally the cleaning up time of the instance. For this aim, we used a freshly installed Linux machine, ready to use OS, with the same specifications on both sides, and in every step, we recorded the time metrics mentioned.

Possibly, due to its rich Marketplace allowing to clone ready-to-use images, OPNBL's deployment time took about 50 seconds, which is very significant compared to OPSTK that takes a longer time in the same conditions. However, in case one wants to upload a custom image and create the whole template manually from scratch instead of uploading a ready-to-use version from the marketplace, we have recorded that the whole process plus the deployment time will take about 3 minutes. On the other hand, for the machine to be in a running state, we can observe that OPSTK took less time, which can be explained by the fact that some platform services were executed during the Deployment phase, as OPSTK has a relevant number of components. We also do not see a significant difference when it comes to the time it takes to turn off the machines. Nevertheless, the cleaning up process of instances takes relatively more time for OPSTK. Taking into consideration these time measurements, we come to the conclusion that OPNBL is more rapid, user-friendly and less complicated to work with.

B. Implemented testbed

We perform traffic tests on a real OPNBL cloud deployment and evaluate the exchanged messages in the control plan. With this evaluation, we aim to demonstrate how traffic control and management could be done in the cloud using ODL or at least providing guidelines for having a centralized network controller in a Cloud environment.

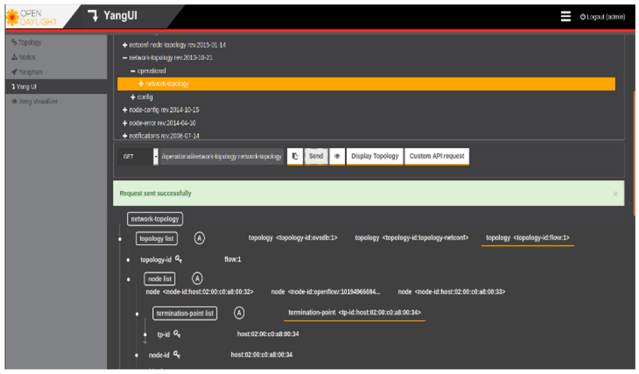

Fig. 9. Network topology API.

YangUI provides all available APIs in ODL, but only the ones installed that will work. The first API we tested is network topology, it helps to GET all the network information, such as connected nodes, ports, flow table statistics, mac and ip addresses, etc. as seen in Fig. 9.

To add or modify a flow through REST API the controller registers with the MD-SAL for configuration and data notification, then using RPC implementation, OpenFlow plugin adds the new flow, this request is sent using the YangUI as a REST call, with all parameters. Then MS-DAL generates a notification of the changing data to its flow programmer service to add the flow in the appropriate switch.

Fig. 10 shows how we added the flow to our OvS using YangUI.

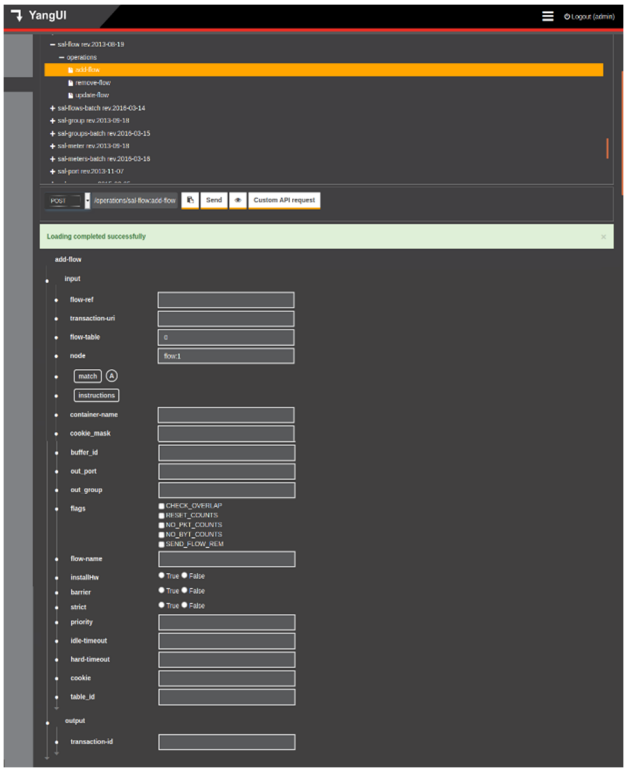

Fig. 10. YangUI interface to Add/delete/update Flows.

When a flow is deleted, the OpenFlow switch sends a notification to the MD-SAL, which registers it to the flow programmer service. This latter uses the OpenFlow plugin to get the received data. In YangUI, the DELETE call follows this logic to delete a network flow.

We verify these changes using the line command:

```
#ovs-ofctl -O OpenFlow13 dump-flows ovsbr
```

which helps to show the flow tables in our OvS as seen in Fig. 11 below:

```
root@OPNBL:/home/omayma# ovs-ofctl -O OpenFlow13 dump-flows ovsbr
 cookie=0x2b00000000000000, duration=495.707s, table=0, n_packets=0, n_bytes=0, priority=100,dl_type=0x88cc
 actions=CONTROLLER:65535
 cookie=0x2a00000000000000, duration=117.524s, table=0, n_packets=12, n_bytes=1008, idle_timeout=600, hard_
timeout=300, priority=10,dl_src=02:00:c0:a8:00:32,dl_dst=02:00:c0:a8:00:34 actions=output:"one-10-0"
 cookie=0x2a00000000000001, duration=117.522s, table=0, n_packets=12, n_bytes=1008, idle_timeout=600, hard_
timeout=300, priority=10,dl_src=02:00:c0:a8:00:34,dl_dst=02:00:c0:a8:00:32 actions=output:"one-8-0"
 cookie=0x2a00000000000004, duration=117.522s, table=0, n_packets=4, n_bytes=280, idle_timeout=600, hard_ti
meout=300, priority=10,dl_src=02:00:c0:a8:00:33,dl_dst=02:00:c0:a8:00:32 actions=output:"one-8-0"
 cookie=0x2a00000000000005, duration=117.522s, table=0, n_packets=4, n_bytes=280, idle_timeout=600, hard_ti
meout=300, priority=10,dl_src=02:00:c0:a8:00:32,dl_dst=02:00:c0:a8:00:33 actions=output:"one-9-0"
 cookie=0x2a00000000000006, duration=78.108s, table=0, n_packets=8, n_bytes=728, idle_timeout=600, hard_tim
eout=300, priority=10,dl_src=02:00:c0:a8:00:33,dl_dst=02:00:c0:a8:00:34 actions=output:"one-10-0"
 cookie=0x2a00000000000007, duration=78.108s, table=0, n_packets=8, n_bytes=728, idle_timeout=600, hard_tim
eout=300, priority=10,dl_src=02:00:c0:a8:00:34,dl_dst=02:00:c0:a8:00:33 actions=output:"one-9-0"
 cookie=0x2b00000000000000, duration=491.815s, table=0, n_packets=9, n_bytes=686, priority=2,in_port="one-8
-0" actions=output:"one-9-0",output:enp0s25,output:"one-10-0",CONTROLLER:65535
 cookie=0x2b00000000000001, duration=491.815s, table=0, n_packets=9, n_bytes=686, priority=2,in_port="one-9
-0" actions=output:"one-8-0",output:enp0s25,output:"one-10-0",CONTROLLER:65535
 cookie=0x2b00000000000002, duration=491.815s, table=0, n_packets=318, n_bytes=91229, priority=2,in_port=en
p0s25 actions=output:"one-8-0",output:"one-9-0",output:"one-10-0",CONTROLLER:65535
 cookie=0x2b00000000000003, duration=491.815s, table=0, n_packets=4, n_bytes=280, priority=2,in_port="one-1
0-0" actions=output:"one-8-0",output:"one-9-0",output:enp0s25,CONTROLLER:65535
 cookie=0xa, duration=495.707s, table=0, n_packets=2, n_bytes=386, send_flow_rem priority=0 actions=CONTROL
LER:65535
```

Fig. 11. Show OvS flows.

VII. Conclusion

Network infrastructures affect highly the cloud networking, as it controls its VMs traffic. With the emerging concept of SDN, it is becoming possible to program these actions and reduce the traditional complexity. There are many Cloud Orchestrators in the market but none is as modular and easy to integrate as OPNBL, therefore we were motivated to use this platform to integrate with our chosen controller: ODL.

During this integration process, we ended up to simply use ODL to manage OvS through OpenFlow, having it already connected to OPNBL 's nodes. However, this is just a start of an unexplored idea. To test it more, we evaluated traffic management in our proposed topology. This experiment could be extended to other topologies and applications as a perspective. Possible challenges include the impact of SDN controller's management of only the VM networks or also the host networks and the horizontal or vertical scalability.

Acknowledgment

This main author is grateful for the support of Agence Universitaire de la Francophonie (AUF) via its Eugene Ionesco scholarship programme.

References

[1] O. Belkadi, Y. Laaziz, A. Vulpe and S. Halunga, "An Integration of OpenDaylight and OpenNebula for Cloud Management Improvement using SDN," *2019 27th Telecommunications Forum (TELFOR)*, Belgrade, Serbia, 2019, pp. 1-4,

[2] OpenDaylight Controller. [online] Available at: https://www.opendaylight.org [Accessed 12 May. 2020].

[3] OpenNebula. [online] Available at: https://opennebula.org/ [Accessed 25 Jun. 2019].

[4] OpenvSwitch. [online] Available at: https://www.openvswitch.org/ [Accessed 15 Mar. 2020].

[5] OpenStack Documentation [online] Available at https://docs.openstack.org [Accessed 25 Apr. 2020].

[6] R. Moreno-Vozmediano, E. Huedo, I. Llorente et al., "BEACON: A cloud network federation framework," *European Conference on Service-Orientedand Cloud Computing*, Springer, Cham, 2015. pp. 325-337.

[7] OpenDOVE. [online] Available at: https://wiki.opendaylight.org/view/Open\DOVE: Main [Accessed 25 Jun. 2019].

[8] X. Wen, G. Gu, Q. Li et al. "Comparison of open-source cloud management platforms: OpenStack and OpenNebula." *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, IEEE, 2012, pp. 2457-2461.

[9] A. Vogel, D. Griebler, C. A. F. Maron, C. Schepke, and L. G. Fernandes, "Private IaaS Clouds: A Comparative Analysis of OpenNebula, CloudStack and OpenStack," in *24th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, Greece: IEEE, Febuary 2016, pp. 672–679.

[10] E. Caron, L Toch, and J. Rouzaud-Cornabas, "Comparison on openstack and opennebula performance to improve multi-cloud architecture on cosmological simulation use case", *Research Report RR-8421. INRIA*, pp. 23, 2013.\

[11] O. Tkachova, M. J. Salim and A. R. Yahya, "An analysis of SDN-OpenStack integration," *2015 Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*, Kharkiv, 2015, pp. 60-62.

[12] C. H. Benet, R. Nasim, K. A. Noghani and A. Kassler, "OpenStackEmu — A cloud testbed combining network emulation with OpenStack and SDN," *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, 2017, pp. 566-568.

[13] M. He, A. M. Alba, E. Mansour and W. Kellerer, "Evaluating the Control and Management Traffic in OpenStack Cloud with SDN," *2019 IEEE 20th International Conference on High Performance Switching and Routing (HPSR)*, Xi'An, China, 2019, pp. 1-6.

[14] Alpine Linux. [online] Available at: https://alpinelinux.org [Accessed 15 Jun. 2019].