International Journal of Engineering Education Vol. 33, No. 2(B), pp. 877–886, 2017
Printed in Great Britain

0949-149X/91 $3.00+0.00

Linux Based Virtual Networking Laboratories for Software Defined Networking*

VLADIMIR DJURICA and MIROSLAV MINOVIĆ
University of Belgrade, Faculty of Organisational Sciences, Jove Ilića 154, 11 000 Belgrade, Republic of Serbia.
E-mail: {djuricav | miroslav.minovic}@fon.bg.ac.rs

With the fully mature and a vast number of available virtualization solutions, there is an uptake in creating the opportunity for remote and/or virtual laboratories to either supplement or fully replace physical networking laboratories. Our approach focuses on nodes, rather than environments hosting the nodes. The paper addresses setting up virtual laboratories made of widely available, general purpose, operating systems based on Linux that act as a network operating system. We discuss several Software Defined Networking solutions, and lay out the configuration setup for virtual laboratories. We evaluate them by the opportunity they provide in the context of learning and a potential experience. We conclude with an observation that with an increasing number of Linux based network operating systems, management of network forwarding devices becomes management of servers, which leads to the unification of the cloud fabric.

Keywords: remote laboratory; virtual classroom; education in computer networking; Linux-based network operating systems

1. Introduction

The role and significance of virtual laboratories, either remote or local, in computer networking education, and especially Software Defined Networking (SDN) is continuously growing [1]. There are ongoing considerations of its learning effectiveness [2], the ability to mimic real environments, as well as reflections on drawbacks and opportunities. Currently, virtual laboratories are either hosted on a private infrastructure or public clouds. They are running on a variety of proprietary or open source solutions, and can provide either a fully virtualized environment, or be a supplement to the existing physical infrastructure. In our opinion, some of the key features that the virtual networking laboratories provide are: (i) increased access to laboratories, (ii) decreased price of experimentation, (iii) increased flexibility and number of options, (iv) increased scalability, (v) ability to save state, (vi) ability to migrate device, (vii) ability to perform templating.

The goal of this paper is to survey and evaluate the setup of several computer networking virtual laboratories that can be used for education and experimentation. We further compare solutions by their primary use cases. We focus primarily on SDN, and Linux based network operating systems for bare-metal switches. We assess YANC, OpenSwitch, Cumulus VX and Mininet as virtual laboratory solutions, as well as review their technical capabilities (Tables 5 and 6).

Our hypothesis is that the Linux based virtual laboratories can be used to create inexpensive, both traditional and SDN computer networking laboratories, including virtual and non-virtual, as well as local and remote setups.

Computer networks are the major Internet building block and its evolution is essential to its growth. Every network architecture contains control and (packet) forwarding plane. In traditional architectures those two planes are tightly coupled together. For that reason, the traditional architectures have proved to be complex, slow to respond to changes and difficult to manage. As an answer to these challenges, there have been many attempts to make networks more programmable [3–7]. Software Defined Networking (SDN) [8] is an approach to computer networking architecture, which enables the innovation, programmability and simplified network management. It breaks the vertical integration by decoupling control and data planes, providing logically centralized control and network programmability.

In the last several years, there has been an uptake in the development of network operating systems based on Linux. These systems are portable across multiple hardware platforms, unlike many existing proprietary solutions. Furthermore, they can also be easily ported into any general purpose hypervisor, and used to setup a virtual teaching facility, or experimentation testbed. These virtual laboratories can be setup in one of the following ways: (1) local virtual laboratory, running on a host-based hypervisor (known as type 2 virtualization), where a researcher/student can set up the laboratory on its own computer, (2) remote virtual laboratory, running on a bare-metal hypervisor (known as type 1 virtualization), mainly used by institutions, and (3) an augmented laboratory based on the Linux oper-

* Accepted 13 December 2016.

ating system. The third option means that by installing the software along with the package dependencies to any Linux based machine, it is possible to create a virtual laboratory. Such laboratory can sit on a personal computer, as well as on a remote server, bare-metal or virtualized.

For the above mentioned cases of full stack virtualizations (options 1 and 2), we setup laboratories that use images which would normally be deployed to the physical hardware without any modification. We suggest use of vendor agnostic, industry recognized Open Virtualization Format for distributing full virtualization based laboratories. This way, the students are not limited by choice of a hypervisor tool and may experience a full production environment. Full virtualization is particularly valuable for the cases of writing and testing networking applications, especially for the advanced courses in computer networking. Also, compatibility and portability of such applications is not an issue.

There is a limited number of laboratories and tools used for SDN related experimentation. Mininet [9] has been often used for these purposes, while other tools that we mention in this paper are mostly not being used, or not at all. We also wanted to highlight that SDN is a coined term for the entire suite of heterogeneous solutions, and it is still evolving. Our goal is to setup labs for SDN category of bare-metal network switches based on open source and free software, as well as consider other experimentation, Linux based, platforms such as YANC. We present our experience with setting up laboratories based on images deployable to the bare-metal without modifications. For the solutions we build from ground, we assume Ubuntu distribution, while highlighting that any other Debian based distribution can be used. While this is not a full survey of all the tools that are available, it showcases solutions that have emerged in the last four years, and whose potential and the current adoption encourages us to think of them as solid tools both for students and industry.

We start with explaining approaches in computer networking: traditional computer networking (Section 2), Software Defined Networking (Section 3), focusing on bare-metal switches (Section 4). Section 2 contains the problem statement. Section 3 discusses SDN as an answer to the challenges posed in the traditional architecture design. Then, in Section 4, we discuss the network switch operating systems for bare-metal switches and laboratory solutions such as: YANC, OpenSwitch, Cumulus Linux and Mininet. This section contains an example on how to setup a laboratory with a basic topology, together with a general approach and guidelines. Finally, in the discussion section (Section 5), we perform the evaluation of each of the laboratories and conclude the paper in Section 6.

2. Traditional computer networking

In this and the following section, we discuss the concepts and main challenges in computer networking. We start with an introduction of the traditional networking concepts that lays out the setting and provides a better learning experience for the reader. Next, we make a connection between the traditional computer networking, SDN and bare-metal switches, as we provide the state of the art overview of the field for which we are setting up the virtual laboratories for.

Computer networks consist of network elements performing different functions such as switching, routing, firewalling, etc. [10]. Based on functionality, a network is composed of two planes: control plane and data, or forwarding plane. The control plane makes traffic decisions and is considered to be a network's brain [11]. The forwarding plane performs data forwarding functions based on control plane's decisions. In traditional computer networking, control and data planes are coupled together, typically in a proprietary vendor box [10], as depicted in Fig. 1. Network elements are independent, distributed and converge to a certain state based on the information exchange. They run large number of networking protocols that are, in many cases, proprietary and closed solutions [8]. The greatest challenges that the traditional networks face are: (i) significant number of users that need or use a fraction of features available within a network device, (ii) increased capital expenses of vertically integrated control and data planes approach to the network boxes, (iii) slow innovation due to the complexity of changes in network protocols, (iv) difficult or impossible experimentation with new protocols (v) slow adoption of new features due to the complexity of a specific vendor change request process (subject to acceptance, desired timeline, specific needs, quality, etc.), (vi) diverse configuration interfaces across vendors and products, and (vii) decentralized management of non-programmable networks (challenge in automation and orchestration that results in high operating expenses and are error-prone). Finally, there is (viii) a huge inertia in the evolution of the Internet due to the need for agreement among multiple organizations and large capital investments (i.e., Internet ossification [12]).

The first attempt to answer the above mentioned challenges resulted in the appearance of programmable networks [5]. Then, the SDN approach followed [10], driven by the desire to innovate at a software speed, and make the development of

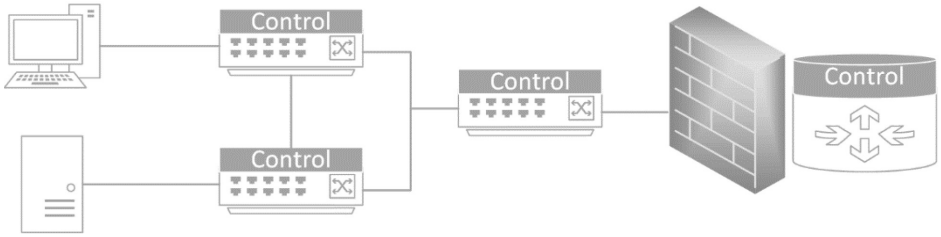

Fig. 1. Traditional, vertically integrated computer networks architecture.

computer networks similar to any other software solution.

Another significant aspect of computer networking, either traditional or SDN, is network virtualization [13]. The motivation for virtualization has been: (i) increased and more efficient utilization of hardware resources that can be shared, (ii) isolation, mainly for security, privacy, and management reasons, (iii) aggregation of physical resources to form large virtual resources, larger than physical, (iv) dynamic allocation of resources that can be done on demand and (v) simplified management through generalized programming abstractions.

Within programmable networks [5], predecessors of SDN, the concept of network virtualization was not new. In telecommunications, there was X.25 protocol that had a division of logical virtual circuits and physical substrate to the supporting virtual calls and permanent virtual circuits. Then, there was a concept of virtual local area networks (VLANs) that allowed full logical separation and isolation of different networks. Virtual Private Networks (VPNs) came as an option to keep the traffic private across public infrastructure. A virtual point-to-point connection establishes the VPN, either by using tunneling protocol or a traffic encryption. Later on, network virtualization flourished in several categories: virtualization of network interface cards (NICs) (vNIC and pNIC), virtualization of switches (vSwitch [14], VALE [15]), and virtualization of LAN's in clouds (such as VXLAN, NVGRE and STT).

3. Software defined networking

SDN is an approach to computer networking that assumes: (i) decoupling of control and data planes, (ii) definition of interactions between the planes through the well defined abstractions (API's), (iii) logically centralized management, (iv) programmability, (v) and the open standard concept.

Data plane is a collection of simple forwarding devices, typically a hardware, that performs forwarding actions based on control plane decisions such as forward, drop, forward to controller or header rewrite. Control plane performs traffic decisions and instructs forwarding plane, through a southbound API, most commonly OpenFlow [16]. Control plane has full topology overview over its domain and a complete control over multiple data plane elements and its states. This control logic lives inside SDN controller and is also called network operating system. Controller provides services to the SDN applications via a north bound API. Depending on the architecture design, there can be single or multiple controllers. In the instance of multiple controllers, peer-to-peer information exchange can be established through an east-west API. To support the multi controller environment, a network hypervisor is used and provides the ability to create a new topology as an overlay to the existing hardware topology.

Next, to clarify some other key concepts, we define NFV, NV and discuss their relations to SDN. SDN separates control from forwarding, while providing logically centralized management, orchestration and automation of network resources. NFV optimizes network services, while decoupling network functions from proprietary hardware appliances, running them in software with the goal to do a better and faster service innovation and provisioning. NV is a technology enabler for multi-tenancy by providing the support for coexistence of various network architectures. It is also an enabler for coexistence of multiple SDN controllers.

Another almost fundamental thing for SDN context is an OpenFlow protocol. Conceived at Stanford and initially used on campus networks [16], OpenFlow has been considered as the first standard communication interface between control and data planes. It is an open protocol for flow programming in network devices, supporting production from experimental traffic isolation. This feature becomes the key testbed for conducting network experiments. It became possible to try new, non-IP based network protocols. Key building blocks of an OpenFlow switch are: (i) flow table, having a matched action (forward, drop, flood, header rewrite, send to controller) for any flow, (ii) secure channel, for communication between a switch and a controller and (iii) OpenFlow protocol

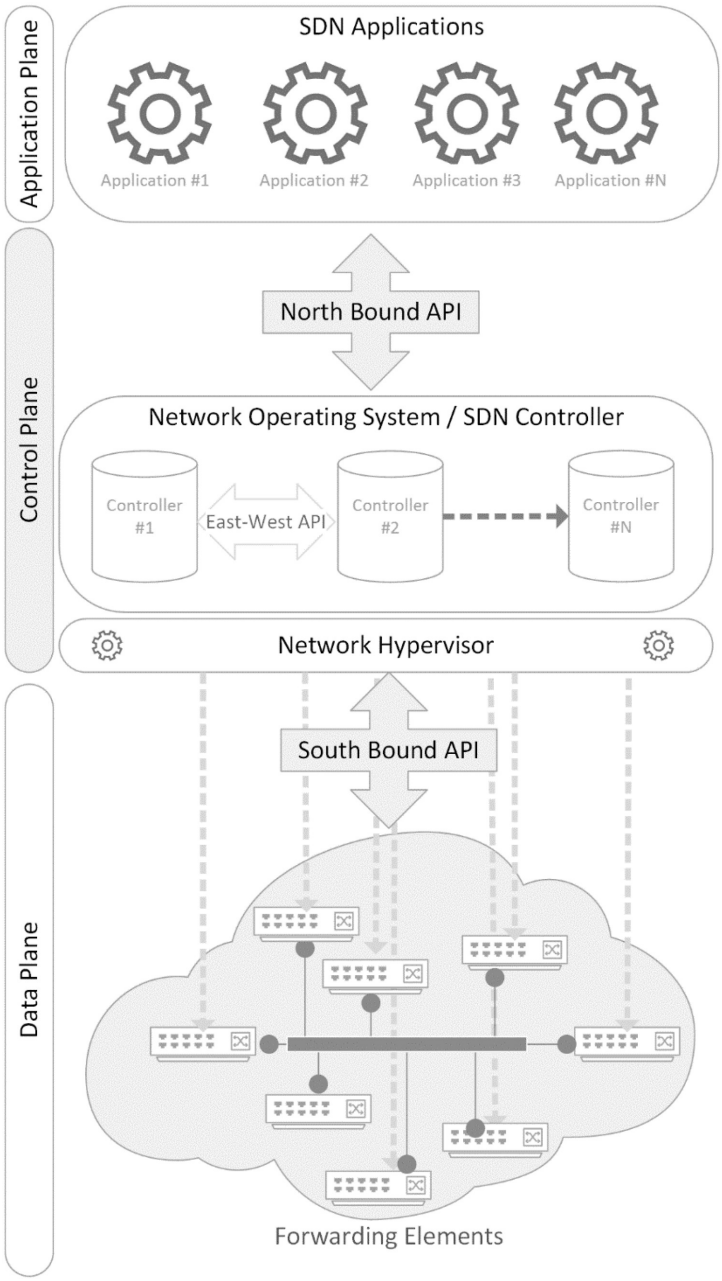

Fig. 2. Possible representation of SDN architecture.

that provides a communication standard between a switch and a controller.

The motivation for OpenFlow came as a compromise between the legacy technology with large deployment base and the new SDN concept. Since this protocol brought the standardized API to control data plane, networks were ready to become more programmable, more software defined. What OpenFlow offered was to leverage the existing hardware by performing firmware upgrade within network equipment, thus making it OpenFlow enabled. To this end, we recognize two types of OpenFlow switches: (i) dedicated, representing a simple data path forwarding element that does not support l2/l3 processing, and (ii) OpenFlow enabled switches, representing a vendor switch that performs a firmware upgrade to support the OpenFlow protocol.

Adding a new protocol to the OpenFlow enabled network environment is achieved by the implementation that takes place inside the controller. The steps include: (i) defining a new flow, (ii) tying it at the desired ports, (iii) setting the forwarding of all packets to the controller, and, finally, (iv) the new protocol sets the flow entries in all other OpenFlow switches within the experimental network slice.

The key benefit of the SDN approach in the computer networking education is the ability to

easily access and modify components and functionalities of the systems that are traditionally either difficult or impossible to modify [8, 10]. Furthermore, it can be done in a programmatic fashion, taking the learning and experimentation to a new level for both lecturers and students.

4. Network switch operating systems/for bare-metal switches

More recently, an emerging form of the forwarding element in SDN architectures, is a network white box or a bare-metal switch. Unlike vendor proprietary switches, these network elements can accept variety of switch operating systems, commonly Linux based. These are hardware agnostic switch operating systems, as long as the driver support exists. Combined with bare-metal switches, the approach allows flexibility of choice and the environment optimization, while preventing vendor or single solution locking.

There is a growing number of switch operating systems based on Linux, such as OpenSwitch [17], Cumulus Linux [18], Switch Light OS [19], PicOS [20], that can be ported on to the bare-metal network switches. In order for these operating systems to support different hardware, only drivers are needed. For this reason, we anticipate the future uptake in use of these systems as well as the growth in their diversity.

In the context of education, the experimentation becomes possible within any hypervisor or Linux based operating system, depending on a particular solution, which unlocks and simplifies new ways of setting up the laboratories and experiments. Further, we present individual solutions.

4.1 Yet Another Network Controller (YANC)

YANC SDN controller [21] uses Linux operating system and extends it to become a network operating system. The key challenge in SDN controller solutions that YANC is trying to tackle, is a monolithic SDN application design that is typically tied to a programming language supported by the controller. This is gating the pace and diversity of innovation of SDN applications. Furthermore, most of controllers have issues with platform portability across various hardware architectures. Linux, being a general and highly represented operating system, can be deployed across varieties of architectures. It can run on a commodity and legacy hardware, while securing the support for the future hardware architectures without a need to vet across the industry.

Within the YANC architecture, network application design is not limited by the controller. The abstraction of control plane protocol, such as the OpenFlow, is represented by the driver in YANC architecture. Supporting a new protocol assumes writing a new driver, that can easily be added, upgraded, replaced or removed. Administration of applications' behavior, rights and other interactions is performed by a system administrator, and manipulations of the network state can be achieved using standard Linux file system commands.

4.1.1 YANC Lab

With a slight modification, this can be used on any Linux. We will run it on Ubuntu, a Debian based distribution, and create the topology in Fig. 3, as per instructions in Table 1.

4.2 OpenSwitch

OpenSwitch was originally outsourced by Hewlett Packard (Enterprise), and, a year later, changed the project custodian to Linux Foundation. OpenSwitch is an open source, Linux based operating system. Its command line interface and commands are similar to those used for Cisco equipment. For this reason, industry networking professionals are expected to have a very steep learning curve and easily leverage their existing knowledge.

Native Linux applications can be run on OpenSwitch, and it can be integrated with various existing solutions, such as Quagga, and provides a restful API for application level control. The appliance itself provides an ability to control not only through the command line interface and API, but also through the web user interface, available through the standard web port 80.

4.2.1 OpenSwitch lab

We assume VirtualBox [22] for this lab. The case study considers two hosts trying to ping each other. The same lab base will be used for Cumulus VX below. Table 2 contains the setup instructions.

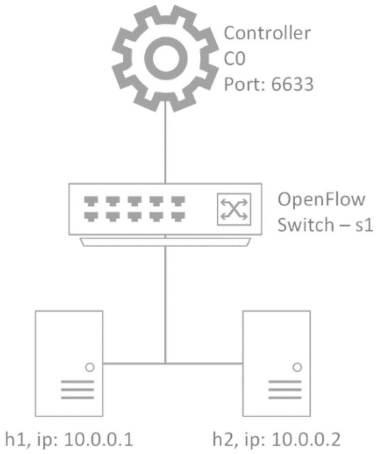

Fig. 3. Yanc-of-adapter SDN controller.

Table 1

Command	Description
sudo apt install git	• Install Git versioning system needed for cloning yanc repository from GitHub.
git clone https://github.com/ngn-colorado/yanc.git	• Cloning yanc repository to local Ubuntu machine
sudo apt install util-linux - install util-linux dependency sudo apt-get install bison sudo apt-get install flex sudo ./configure sudo make j8 sudo make install	• Change to 'yanc' directory • Download https://github.com/libfuse/libfuse and unpack fuse-2.9.6.tar.gz • installing FUSE dependency • installing util-linux dependency • installing bison tool dependency • installing flex tool dependency • run configuration • create build for FUSE dependency
sudo make	• go back to yanc folder and run make for yanc
sudo mkdir /net - create folder sudo chown <user> : <group> /net - make user owner of the folder sudo ./yanc -f /net	• create /net folder • make user owner of the folder • run yanc • this should started the yanc file system with /net as its mount point.
/<yanc-path>/apps/of-adapter/ - change folder to git submodule init git submodule update sudo apt install dbus export PKG_CONFIG_PATH=~/yanc/apps/of-adapter/lib/om-lib/om/ipc/dbus:$PKG_CONFIG_PATH sudo apt-get install libdbus-glib-1-dev	• For this step, we will need a separate terminal windows. • Next, we are starting yanc-of-adapter, utility that allows yanc file system connection to OpenFlow based switches. We will be using Mininet for this operation. • We will use git submodule commands to allow om-lib submodule. • Check: echo $PKG_CONFIG_PATH, if it's empty add path where 'dbus' folder is located. • to avoid reporting missing file dbus-1.pc install libdbus-glib-1-dev
cd /<yanc-path>/apps/of-adapter/ sudo make sudo ./yanc-of-adapter -h	• change directory • make build • This should create an executable named yanc-of-adapter. • To check if this step has been successful we run yanc-of-adapter -h
sudo apt-get install mininet sudo mn mininet> pingall	• Next, OpenFlow switches should allow yanc-of-adapter connection to yanc file system. • We will use Mininet to create virtual OpenFlow switches and connect them to yanc-of-adapter acting as SDN controller. First we install Mininet and then test for connectivity.
sudo ./yanc -f /net	• Next we are going to run yanc file system, yanc-of-adapter and Mininet. This will mount yanc under /net directory
<yanc-path>/apps/of-adapter sudo ./yanc-of-adapter /net unix:path=/var/run/dbus/system_bus_socket -vvv	• Open new terminal • go to the folder 'of-adapter' • start yanc-of-adapter (requires root privileges)
mn --controller=remote,ip=127.0.0.1,port=6633	• Open a new terminal and go to <mininet-folder>/bin • create topology

4.3 Cumulus linux

Cumulus Linux is a network operating system based on Debian Linux. It is a proprietary solution, created by Cumulus Networks [18]. The goal is to have Linux run on a commodity hardware spanning across generalized hardware architecture. The motivation is to overcome the following challenges: (i) limited scalability in traditional networking environments, (ii) complexity in configuration changes and its propagation across network elements, (iii) high cost of the equipment and its operation.

Many existing Linux tools, can potentially be run natively on this operating system. Bash (or other) scripting environment(s) can be fully utilized for automation tasks. Furthermore, a nowadays de facto industry standard orchestration, automation and provisioning tools are supported such as Chef, Puppet, CFEngine. Any industry standard programming language that Debian supports can be used, too. Switch driver is part of the Linux kernel, and writing and adding a new driver is required to provide support for a new hardware device. Routing, automation, orchestration, monitoring, applications, etc., are all in the user space and, therefore, part of the hardware abstraction layer. Finally, cost can be reduced through the utilization of a common (software) platform for an existing hardware, and by providing the ability to port system across a diversified number of vendors and multiple CPU architectures (x86, PowerPC and ARM).

Table 2

Command	Description
Download appliance	• https://archive.openswitch.net/artifacts/periodic/master/latest/appliance/
Virtual Host: Switch #1, Switch #2 (same settings) Add network adapter Attached to: Internal Network Name: swp1 Promiscuous mode: Allow All Cable Connected	• import appliances • setup *cable connection* between hosts
user: root (no password) type: vtysh configure terminal interface eth1 ip address 192.168.0.1/30 no shutdown	• configure Switch #1
user: root (no password) type: vtysh configure terminal interface eth1 ip address 192.168.0.2/30 no shutdown	• configure Switch #2
ping 192.168.0.1	• test connectivity (e.g. from host #1 ping host #2)

4.3.1 Cumulus VX lab

Cumulus Linux VX comes in open virtualization (OVF) format, nonspecific to any hypervisor. It can be easily imported into a variety of hypervisors. For the purpose of this, and all labs, we assume the VirtualBox, the open source software. We will do a simple lab of installing two switches, establishing the connection and testing connectivity. Each of them will be running in a separate VM. In Cumulus, switch is called a leaf. For this lab (Table 3), we will use swp1 (swpX stands for switch port, X for port number). We consider the following topology:

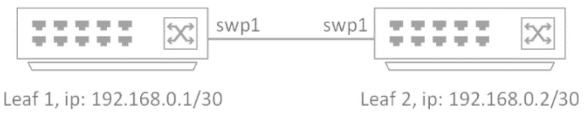

Fig. 4. Cumulus Linux VX lab topology.

Table 3

Command	Description
Download Cumulus VX 3.0.0 appliance	• https://cumulusnetworks.com/cumulus-vx/download/
Virtual Host: Leaf1, Leaf2 (same settings) Add network adapter Attached to: Internal Network Name: swp1 Promiscuous mode: Allow All Cable Connected	• import appliance Leaf1 • setup *cable connection* between hosts • repeat these steps for Leaf2
user: cumulus password: CumulusLinux!	• Login to the system
sudo nano /etc/network/interfaces # add the following lines for swp1: auto swp1 iface swp1 address 192.168.0.1/30 # save the /etc/network/interfaces file	• Define swp1 and swp2 interfaces on leaf1
sudo ifup swp1	• bring up the interface leaf1, swp1
sudo nano /etc/network/interfaces # add the following lines for swp1: auto swp1 iface swp1 address 192.168.0.2/30 # save the /etc/network/interfaces file	• repeat steps for leaf2
sudo ifup swp1	• bring up the interface, leaf2, swp1
ip link show dev swp1	• check link state on leaf1, leaf2
ping -c 4 192.168.0.2	• from leaf1 ping leaf2 to verify connectivity

4.4 Mininet

Mininet is an early SDN tool [9]. It's particularly significant use case is ability to create a large number of network resources and run the experiment on the top of that topology. Mininet is capable of emulating network elements such as hosts, switches (l2/l3), routers and links, while running on a single instance of Linux operating systems. It is installable on most distributions. It leverages the OpenFlow [16] as a switch control protocol. There are two enhancements of Mininet: Mininet Cluster Edition (CE), Mininet 2.0 (HiFi). The first one is the extension for large scale simulation at the order of thousands of nodes, and the second one is a newer version with the functionality extensions such as container based virtualization support, bandwidth limitation, etc.

4.4.1 Mininet Lab.

The easiest is to download and import Mininet's full-stack virtual machine from: https://github.com/mininet/mininet/wiki/Mininet-VM-Images and import into the hypervisor such as VirtualBox. Alternatively, a package-based install for, say, Ubuntu, is shown in Table 4.

5. Discussion

Most of the research efforts today focus on building the environment for setting up network topologies and running the experiments, such as Open Virtual Lab (OVL) [23] and Virtual Network User Mode Linux (VNUML) [24]. We consider the above solutions from the network node standpoint, potentially running in one such environment.

First, in Table 5, we evaluate setting up the laboratories across several criteria. We treat them as a laboratory solution, rather than just a product. The following are the evaluation criteria:

- *Solution*: Solution Name
- *HW Resources*: Minimum hardware resources that are required to setup a lab.
- *Portability*: Is it possible to configure a lab, and then move it to another place, e.g. other hypervisor / machine?
- *Cloudification*: Is it possible to setup a solution in the cloud environment, either private or public?
- *Setup*: How challenging is a full setup of the laboratory?
- *Learning curve*: For a particular SDN solution.

Table 4

Command	Description
sudo apt-get install mininet	• install mininet package
sudo service openvswitch-controller stop sudo update-rc.d openvswitch-controller disable	• deactivate open-vswitch controller
sudo mn –test pingall	• test Mininet

Table 5 Evaluation of laboratory solutions

Solution	HW Resources	Portability	Cloudification	Setup Difficulty	Learning curve
YANC	Minimum Linux	Yes	Yes	Complex	• requires advanced Linux knowledge • new concepts (state management via file system) • slow learning curve
OpenSwitch	1 x vCPU 512 MB RAM	Yes	Yes	Easy	• mimics a typical large vendors command line interface • provides the familiar traditional networking experience • assumes prior industry knowledge, steep learning curve • with no prior knowledge, medium learning curve
Cumulus VX	1 x vCPU 256 MB RAM	Yes	Yes	Easy	• requires intermediate Linux knowledge • provides familiar Linux experience • with prior Linux or networking knowledge, medium learning curve
Mininet	Minimum Linux	Yes	Yes	Easy	• new concepts by representation (different commands but familiar concepts)

Table 6. Technical capabilities of a solution

Increased Access	Lower Price	Increased Flexibility	Increased Scalability	State Save	Migration capability	Perform Templating
YANC	Yes	Yes	Yes	No	No**	Conditional***
OpenSwitch	Yes	Yes	Yes	Yes	Yes	Conditional***
Cumulus VX	Yes*	Yes	Yes	Yes	Yes	Conditional***
Mininet	Yes	Yes	Yes	No	No**	Conditional***

* Limited to the VX version. If advanced functionalities are need, the full version might be required.
** While the feature is not supported by default, it can be attained by installing the KVM or running it in the hypervisor.
*** Templating is the feature of the hypervisor. Since all of the tools can be installed and ran in the hypervisor of the choice, virtual laboratory can be set to have this feature for each of the tools considered.

In all our tests, we used the latest Ubuntu with its minimum requirements. Further resource savings can be made with different distributions, as well as installation choices and configurations. For that reason, wherever a different Linux distribution can be used, we refer to Minimum Linux. For the reference, Ubuntu system requirements [25] are:

- 1 x vCPU (Single core 700 MHz processor)
- 512 MB RAM (System Memory)

Below is the evaluation of the laboratory solutions based on a survey with three groups with 30 students each at the University of Belgrade, Faculty of Organisational Sciences. The above instructions were provided to the students upfront. Students were asked to rate the setup difficulty and evaluate the learning curve for the given solution.

Cumulus VX is a community version so its limitation is that it is not a production-ready system. It is good for educational purposes, but the scope of experimentation with VX (community edition) is limited. Mininet is an emulator and shows some challenges in accurately representing the behavior of a real system [26]. Mininet is also a very powerful tool for the proof of concept SDN applications using general purpose operating systems, and existing programming languages, and can be easily translated into an production environment. YANC laboratory also relies on Mininet. Table 6 provides another set of technical capabilities that is important for considering solution.

YANC and OpenSwitch can potentially be used equally in laboratories and the production. YANC provides a new bold experimentation which might be successful in the future. Students would benefit from learning different concepts, operating systems more, and would face the SDN controller. As for the OpenSwitch, its use case is leaned towards the traditional environment. Therefore, if a course curriculum is targeted towards exposing students to the vendor equipment for the purpose of doing real world experiment using the existing technologies, OpenSwitch would be a viable alternative.

6. Conclusions

Computer Networking is changing from the traditional towards the SDN driven approaches that consider the separation between control and forwarding planes. The potential education tools are growing in the number and diversity. With the virtualization solutions available, there is a gap, but also a significant opportunity to create remote and/or virtual laboratories to either complement or replace physical laboratories. Together, general purpose network operating systems and Software Defined Networking (SDN) solutions have made such laboratories easily achievable, and contain a rich ensemble of learning and experimentation testbeds in computer networking. We explored the opportunity to set up virtual education laboratories, while focusing on using Linux based network operating systems. Several designs were proposed and the steps for building each of them were outlined. We discussed the solutions and evaluated them based on the provided opportunities for learning and gaining experience, as well as showcased the best use case for each of the laboratories.

Our future research will explore opportunities for building a laboratory that consists of nodes with heterogeneous SDN technologies, and their evaluation using the similar criteria as before. Further, we plan to explore the ability to orchestrate such nodes in a programmatic fashion, in order to provide a learning experience for the higher level networking functions and applications. We believe that the networking laboratory built on the top of general operating systems converges towards managing servers only, and suggest these systems to be managed using the existing orchestration frameworks, such as Chef and Puppet, scripting like Bash or general purpose programming languages, e.g. Python. To this end, we plan to continue exploring the ability to express and teach high level network policies in virtual classroom environments.

References

1. R. Heradio, L. de la Torre, D. Galan, F. J. Cabrerizo, E. Herrera-Viedma and S. Dormido, Virtual and Remote Labs in Education: a Bibliometric Analysis, *Comput. Educ.*, **98**, 2016, pp. 14–38.
2. X. Chen, G. Song and Y. Zhang, Virtual and Remote Laboratory Development: A Review, *Earth Sp. 2010 Eng. Sci. Oper. Challenging Environ.*, 2010, pp. 3843–3852.
3. J. T. Moore and S. M. Nettles, Towards practical programmable packets, in *Proceedings of the 20th Conference on Computer Communications (INFOCOM). Citeseer*, 2001.
4. K. Calvert, Reflections on network architecture: an active networking perspective, *ACM SIGCOMM Comput. Commun. Rev.*, **36**(2), 2006, pp. 27–30.
5. A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente and D. Villela, A survey of programmable networks, *ACM SIGCOMM Comput. Commun. Rev.*, **29**(2), 1999, pp. 7–23.
6. A. Lazar, K.-S. Lim, F. Marconcini and others, Realizing a foundation for programmability of atm networks with the binding architecture, *Sel. Areas Commun. IEEE J.*, **14**(7), 1996, pp. 1214–1227.
7. A. T. Campbell, I. Katzela, K. Miki and J. Vicente, Open signaling for ATM, internet and mobile networks (OPENSIG'98),' *ACM SIGCOMM Comput. Commun. Rev.*, **29**(1), 1999, pp. 97–108.
8. D. Kreutz, F. M. V Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky and S. Uhlig, Software-defined networking: A comprehensive survey, *Proc. IEEE*, **103**(1), 2015, pp. 14–76.
9. R. L. S. De Oliveira, C. M. Schweitzer, A. A. Shinoda and L. R. Prete, Using Mininet for emulation and prototyping Software-Defined Networks, in *2014 IEEE Colombian Conference on Communications and Computing, COLCOM 2014—Conference Proceedings*, 2014.
10. B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka and T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, *IEEE Commun. Surv. Tutorials*, **16**(3), 2014, pp. 1617–1634.
11. H. Farhady, H. Lee, and A. Nakao, Software-Defined Networking: A survey, *Comput. Networks*, **81**, 2015, pp. 79–95.
12. D. Taylor and J. Turner, Towards a diversified internet, *White Pap. Novemb.*, 2004.
13. T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram and others, Network virtualization in multi-tenant datacenters, in *USENIX NSDI*, 2014.
14. B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, A. Networks and M. Casado, The Design and Implementation of Open vSwitch, *12th USENIX Symp. Networked Syst. Des. Implement.*, 2015, pp. 117–130.
15. L. Rizzo and G. Lettieri, VALE, a switched ethernet for virtual machines, *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol.—Conex. '12*, no. June, p. 61, 2012.
16. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner,OpenFlow: enabling innovation in campus networks, *ACM SIGCOMM Comput. Commun. Rev.*, **38**(2), 2008, pp. 69–74.
17. "OpenSwitch."
18. C. Networks, "Cumulus Linux", 2014.
19. B. S. Networks, "Switch Light".
20. Pica8, "Pica8 PicOS".
21. M. Monaco, O. Michel and E. Keller, Applying operating system principles to SDN controller design, in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, 2013, p. 2.
22. W. M. Fuertes and J. E. L. Vergara, A quantitative comparison of virtual network environments based on performance measurements, *14th Work. HP Softw. Univ. Assoc.*, 2007.
23. M. Anisetti, V. Bellandi, A. Colombo, M. Cremonini, E. Damiani, F. Frati, J. T. Hounsou and D. Rebeccani, Learning computer networking on open paravirtual laboratories, *IEEE Trans. Educ.*, **50**(4), pp. 302–311, 2007.
24. F. Galan, D. Fernandez, J. Ruiz, O. Waliti and T. de Miguel, Use of virtualization tools in computer network laboratories, in *Information Technology Based Proceedings of the Fifth International Conference on Higher Education and Training, 2004, ITHET 2004.*, 2004, pp. 211–216.
25. "Ubuntu System Requirements", 2016.
26. F. Keti and S. Askar, Emulation of Software Defined Networks Using Mininet in Different Simulation Environments, in *Proceedings—International Conference on Intelligent Systems, Modelling and Simulation, ISMS*, 2015, vol. 2015-Octob, pp. 205–210.

Vladimir Djurica is a PhD candidate at University of Belgrade, Faculty of Organizational Sciences. He has an extensive experience in working as an IT Manager across Europe. Vladimir has worked as a consultant at Google and is currently working at Brocade Communications Systems. His research interests are in cloud technologies, virtualization and computer networks.

Miroslav Minović is an associate professor in the Department of Information Technology and a senior researcher at the Laboratory for Multimedia Communications. He obtained his PhD in Information Technology. He published a lot of journal and conference papers on HCI, Multimedia and biometric technologies. Miroslav is a member of IEEE society and acts as a National Contact Point for Serbia within the European Association for Biometrics. He serves as a guest editor for the International Journal of Engineering Education.

Prof. Minovic has been engaged in several commercial projects utilizing the actual information technologies and has developed a broad range of informational systems and solutions.