Paper ID #23981

Hands-on Labs and Tools for Teaching Software Defined Network (SDN) to Undergraduates

Dr. Emil H. Salib, James Madison University

Professor in the Integrated Science & Technology Department at James Madison University. Current Teaching - Networking & Security and Cross Platform Mobile Application Development. Current Research - Private Cloud Computing, Mobile IPv6 and Design for Motivation Curriculum

Mr. John David Lester

Hands-on Labs and Tools for Teaching Software Defined Network (SDN) to undergraduates

Dr. Emil H. Salib,and John D. Lester
salibeh@jmu.edu, lesterjd@dukes.jmu,edu
Dept of Integrated Science and Technology (ISAT),
James Madison University (JMU), Harrisonburg, VA 22807

1 Introduction

The origins of Software-Defined Networking (SDN) arose from graduate research work out of Stanford. A graduate student, Martin Casado, observed that the current network operations are hard [1], [2], [3]. He and his advisor took on this as an opportunity and defined a possible solution [4] that is now known as SDN. The cause of network operations problems is a rigid architecture. The first problem is the contrast between the methods that determine the forwarding state: distributed forwarding state is determined by control plane algorithms and the manual configuration state is determined by network operators. Control plane algorithms are good at adapting to change in the network. Network operators are slower and more error prone at adapting to changes in the network, especially at scale. So Martin hypothesized that a program can solve the problem for network operators by determining the forwarding state that was previously manually configured. He then discovered that there were no well-defined Application Programming Interfaces (APIs) for datapath configurations that were focused on consistent state management. Further, there were no general, higher-level, distributed algorithms for determining the data plane state. Martin reduced the network operations problem to two main network architectural problems. The first problem is poor forwarding abstractions. Network switches are fixed function pipelines not designed to be programmed, the implementations of the chipsets were often exposed through their API, and the APIs had unclear state consistency semantics. The second problem he noticed is that the control plane is a distributed system that is coupled with the data plane's topology.

SDN is an approach to solving the network operations problem by (1) generalizing the datapath and (2) decoupling the control plane distribution model from the data plane physical topology. Generalizing the datapath means to move from a fixed-function datapath model to a flexible, programmable datapath model with a well-defined API for consistent state management. The API investigated in the hands-on portion of this paper is OpenFlow. The reasoning behind decoupling the control plane distribution model from the data plane physical topology is simply that tighter clusters are easier to program to and the control plane is implemented independently from the data plane's topology and hardware. The newly decoupled control plane can be designed as a sort of network operating system that provides abstractions to control applications such as topology, link state, and inventory [5].

Over the last few years, SDN has become a very active area of research and operational experimentation [6], [7], [8]. However, the introduction of the SDN foundational knowledge at the undergraduate curriculum level is slow and elementary [9], [10]. Few hands-on, lab-based teaching materials exist in this area both for the undergraduate faculty members and the students. Seeing the value of SDN through our recent study (Senior Capstone Project), we believe it is a great opportunity and a critical mission to identify and enhance the right tools and platforms that enable educators and students to teach, learn, and stay up-to-date on SDN. We also believe that it's imperative to demonstrate how these tools may be effectively utilized and applied through the development and delivery of fully tested hands-on labs and exercises to our undergraduate inter-networking classes.

The paper is organized as follows. In section 2, we provide a brief overview of the opportunity and solution definitions and the objectives of the work described in this paper. Section 3 focuses on the solution implementation along with a brief overview of SDN architecture and components. Section 4 provides the solution results in the form of hands-on labs for teaching SDN to undergraduate students. In section 5, we offer our conclusions and discusss potential future work.

2 Definition and Objectives

Opportunity Definition

Our opportunity is to enable undergraduate students and educators to acquire hands-on knowledge of SDN concepts.

Solution Definition

To address the lack of SDN knowledge among the undergraduate community, we must identify effective tools that enable educators to introduce and teach SDN to networking beginners and advanced students. We must deliver a bundle of technologies that can be used to demonstrate the main concepts of SDN. An understanding of the inner workings of SDN will be provided through a set of well-tested lab exercises that allow students and educators to practice and develop their skills in installing and running network applications.

Objectives

The main goal of this effort is to provide tools that assist students and teachers in acquiring hands-on knowledge of SDN architecture. The following objectives are required to realize our goal:

- Identify and apply an open source SDN controller to modify, enhance, and build network applications on,
- Identify and exercise an open source virtual switch that supports OpenFlow,
- Validate the controller platform by exercising and analyzing currently available network applications, and
- Deliver the aforementioned technologies as a single virtual machine for educators to use in their course of introducing SDN to their undergraduate students.

3 Solution Implementation

In this section, we will describe the two platforms (and their tools) we identified and adopted in the implementation of our solution to the introduction of hands-on labs and exercises in the college-level undergraduate networking courses. Each is packaged into a virtual machine that can be readily used for teaching the new concepts of SDN.

- The first is based on Mininet[11], a self-contained SDN environment that emulates the network (including Open vSwitch and hosts) and allows one to use local or external controllers
- The second is based on GNS3 [12] along with a number of Docker [13] containers (controllers, Open vSwitches, and hosts).

From a teaching perspective, we found the first platform to be ideal for beginners as it allows the students to focus on experimenting with and understanding the SDN concepts using the self contained Mininet environment. This relieves the students from the burden of integrating the SDN components when they are running on separate devices. However, it is limited in the integration of SDN components with non-simulating devices including physical and emulated (virtual) components such as Network Function Virtualization (NFV) elements and traditional routers and switches. This is where the GNS3 platform shines and enables one to develop and deliver more advanced exercises and labs for senior-level courses. Before we describe some in further detail, the next sub-section provides a brief overview of SDN architecture and components.

3.1 Brief Overview of SDN Architecture and Components

This sub-section will provide a brief overview of the basic architecture of SDN [14]. SDN's generally have three major components as shown in Figure 1 [15]:

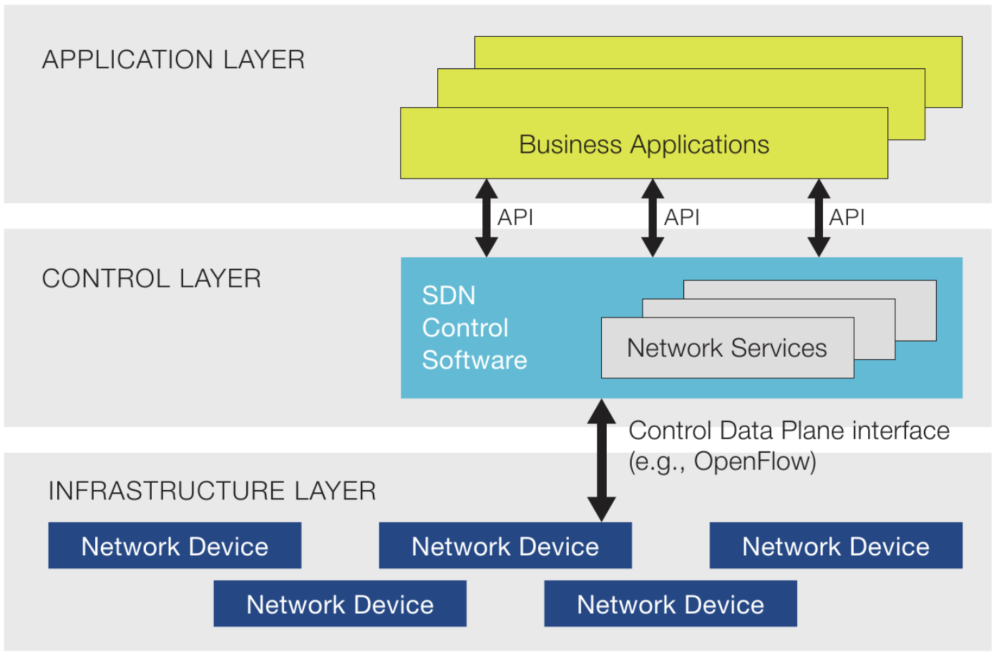


Figure 1: SDN Layered Architecture [15]

- SDN Applications: SDN Applications are programs that communicate behaviors, policies, algorithms and needed resources with the SDN controller via APIs. These applications could include networking routing, network management, or business policies used to run large data centers.
- SDN Controller: The SDN Controller is a logical entity that receives instructions or requirements from the SDN Application layer and relays them to the networking devices. The controller also extracts information about the network from the hardware devices and communicates back to the SDN Applications with an abstract view of the network, including statistics and events about what is happening.
- SDN Networking Devices: The SDN networking devices control the forwarding and data processing capabilities for the network. This includes forwarding and processing the datapath.

The SDN architecture APIs are often referred to as northbound and southbound interfaces, defining the communication between the applications, controllers, and networking components. A Northbound interface (NBI) is defined as the connection between the controller and applications, whereas the Southbound interface (SBI) is the connection between the controller and the physically networking hardware. Because SDN is a virtualized architecture, these elements do not have to be physically running on the same hardware.

In this paper, we will focus on the southbound interface and therefore the following is a brief description of the widely adopted OpenFlow protocol and the OpenFlow-based switch known as Open vSwitch. OpenFlow [16] is the widely accepted implementation of the southbound interface. Also, it is a defined standard where both the SDN controller and Network Devices must meet for successful inter-operability. Further details can be viewed in the OpenFlow specifications hosted at the Open Networking Foundation's (ONF) website [17]. Open vSwitch [18] is a software switch that implements the OpenFlow standard. The Open vSwitch Database Management Protocol (OVSDB) is a management protocol intended to allow programmatic access to the Open vSwitch database [19]. The Open vSwitch Database Schema is documented at [20]. Next, we provide a brief description of the two platforms mentioned earlier.

3.2 Introductory Undergraduate Networking Courses Platform

To introduce SDN exercises in introductory courses, we used of a very useful tool for orchestrating varieties of network configurations for rapid prototyping of SDN exercises: Mininet [11].

3.2.1 Mininet

Mininet uses container-based virtualization to make a single system act as a complete network. This allows the user to specify a network topology of hosts and switches and automates their creation. Mininet has support for Open vSwitch. One may use the Mininet CLI or API written in Python. Mininet's Python API can be used to develop more customizable topologies. Section 4 hands-on exercises will use Mininet's CLI to create simple, built-in topologies.

3.2.2 Controller for Beginners

Mininet has built-in controller classes to support different network controllers, such as, the Mininet reference controller, the Mininet ovs-controller and the less used NOX Classic. The open source SDN controller we chose is POX (inherited from NOX) as it is one of the most popular controllers for use with Mininet. POX, which is an open-source controller for developing SDN applications, provides an efficient way to implement the OpenFlow protocol. The POX controller allows running SDN applications like hub, switch, load balancer, and firewall.

POX is written in Python and offers an easy start for newcomers to SDN. POX is designed as a group of loosely coupled modules that can communicate with each other through events. POX includes many libraries for making it easier to create OpenFlow applications. POX provides an OpenFlow module that listens for connections. Modules can listen to the OpenFlow module for events such as when a connection to an OpenFlow switch is established or terminated. For example, the event handler for a new connection to an OpenFlow datapath (switch) provides information such as the datapath id, the port numbers, and port mac addresses. POX also provides a library for parsing and constructing packets. It should be noted that POX only implements OpenFlow 1.0. Due to this limitation, we chose OpenDayLight and Ryu controllers for the advanced networking courses platform. POx Wiki [21] is the best documentation we found for developing SDN applications with POX.

3.3 Advanced Undergraduate Networking Courses Platform

For advanced inter-networking courses, where integration of physical and emulated devices with SDN controllers and application is required, we chose GNS3 (see Figure 2) .

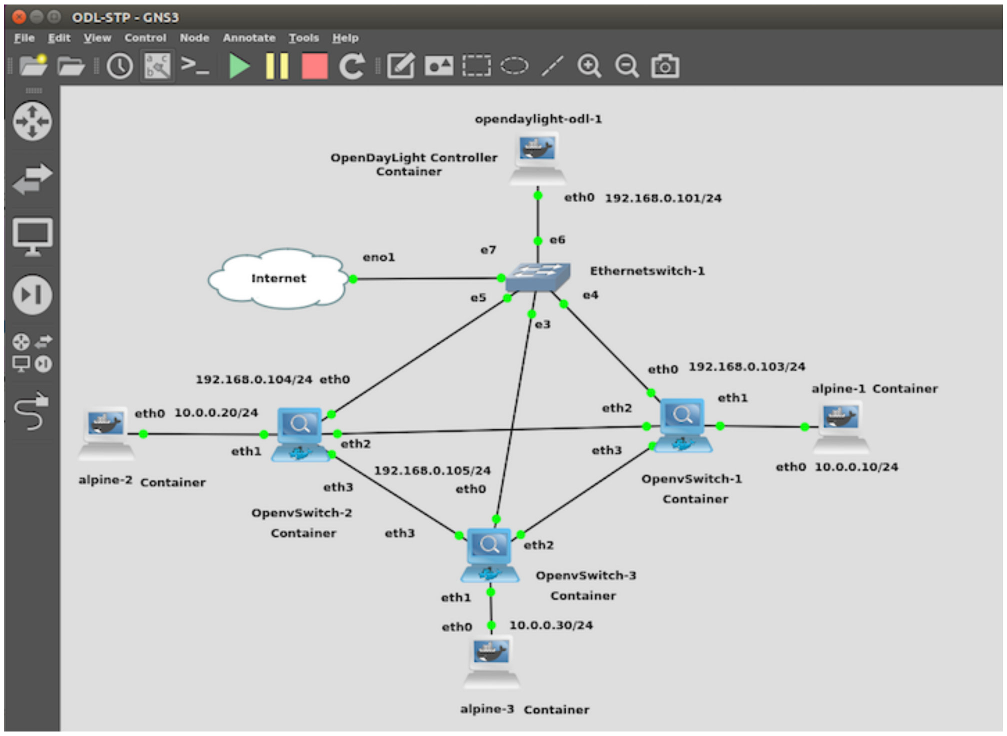


Figure 2: An example of a simple SDN-based switching network

3.3.1 GNS3

Graphical Network Simulator-3 (GNS3) is an open-source network software emulator written in Python. It allows the combination of virtual and real devices, used to simulate complex networks. It uses Dynamips emulation software to simulate the Cisco IOS. Most recently, GNS3 has expanded its integration portfolio to include Docker Containers, VMware virtual machines (VMs), VirtualBox VMs and KVM/QEMU VMs. GNS3 is used by many large companies including Exxon, Walmart, AT&T, and NASA, and is also popular for preparing network professional certification exams. At our university we use it exclusively for senior-level inter-networking classes. Figure 2 represents an example of an SDN-based network we used in one of the labs to be described in detail in Section 4.

3.3.2 Docker Containers

In the design and development of the SDN-based labs on this platform, we made extensive use of Docker containers. Docker [13] is a tool that can package an application and its dependencies in a virtual container that can run on any Linux server. This helps eenhance the flexibility and portability of where the application can run, whether on premises, public cloud, private cloud, or bare metal. For example, Docker also speeds up the development of new and more elaborate labels as the developer is relieved of the tedious tasks of downloading, installing, configuring and integrating numerous software packages.

For example, in a lab to be described in Section 4, we used four different Docker containers: (1) the Docker Hub:osrg/ryu (Ryu SDN Controller) [22], (2) opendaylight/odl (OpenDaylight SDN Controller) [23], (3) alpine (A minimal Docker image based on Alpine Linux) and (4) gns3/openvswitch (An openvswitch container for GNS3). The containers saved the authors significant time and resources. One major advantage of Docker containers is that they are customized in a lightweight size. For example, the alpine Linux is customized as a host with a complete package index and is only 5 MB in size. This is ideal for developimng and performing networking labs.

4 Solution Results - Labs & Exercises

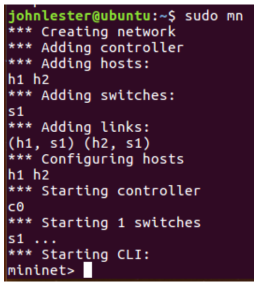
In this section, we present a number of hands-on lab exercises designed and developed for teaching SDN functionality to undergraduate telecommunications and networking students. Exercises 1 and 2 are designed for introductory courses in networking while Exercise 3 is for senior students majoring in telecommunications, networking and security.

4.1 Exercise 1: Open vSwitch Flows Using Mininet

The teaching approach here is to ease students into the concept: the students start Step 1 on Exercise 1 by configuring the Mininet environment to allow the ovs-controller to automatically create and populate the flows in the Open vSwitches. This action is triggered by a request from a host such as pingall. In Step 2, the students learn and practice how to design the flows and manually populate them into the Open vSwitches.

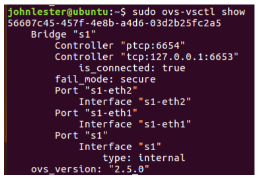
Step 1 - Create Flows in OpenvSwitch (OVS) automatically by the controller

The students start by creating a minimal topology of one OVS bridge connected to two hosts and a controller. Also, they print a summary of the Open vSwitch database contents.
# mn
# ovs-vsctl show



```
johnlester@ubuntu:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

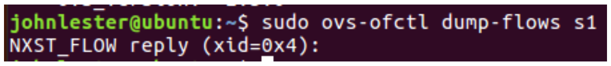
Figure 3: Mininet creates the network



```
johnlester@ubuntu:~$ sudo ovs-vsctl show
56607c45-457f-4e8b-a4d6-03d2b25fc2a5
    Bridge "s1"
        Controller "ptcp:6654"
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        fail_mode: secure
        Port "s1-eth2"
            Interface "s1-eth2"
        Port "s1-eth1"
            Interface "s1-eth1"
        Port "s1"
            Interface "s1"
                type: internal
    ovs_version: "2.5.0"
```

Figure 4: Overview of Open vSwitch database

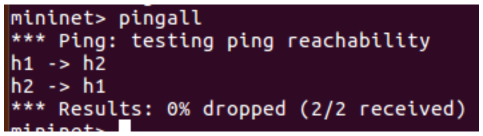
To begin, the flow table of the s1 bridge is empty. S1 is the Open vSwitch created by Mininet when the mn command is launched.
# ovs-ofctl dump-flows s1



```
johnlester@ubuntu:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
```

Figure 5: Empty flow table on s1

The students initiate a pingall so that they can begin to understand how the controller and flow tables work.
mininet>pingall

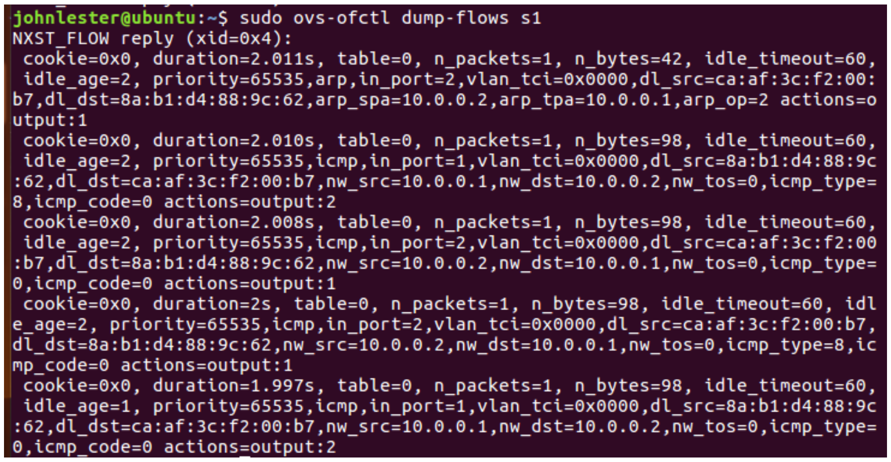


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Figure 6: Verifying connectivity between the hosts

Next, the students view the flow tables. There should be a number of flow installed by the controller in response to the traffic they generated on the hosts and switches.
# ovs-ofctl dump-flows s1

The flow table in Figure 7 does not look like a typical switch built-in forwarding table. For example, a flow in an SDN flows table specifies the input and output ports. As you can see, the first flow states if a packet is received at port 2 (in_port=2), forward to port 1 (actions= output:1).



```
johnlester@ubuntu:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2.011s, table=0, n_packets=1, n_bytes=42, idle_timeout=60,
 idle_age=2, priority=65535,arp,in_port=2,vlan_tci=0x0000,dl_src=ca:af:3c:f2:00:
b7,dl_dst=8a:b1:d4:88:9c:62,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=2 actions=o
utput:1
 cookie=0x0, duration=2.010s, table=0, n_packets=1, n_bytes=98, idle_timeout=60,
 idle_age=2, priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=8a:b1:d4:88:9c
:62,dl_dst=ca:af:3c:f2:00:b7,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=
8,icmp_code=0 actions=output:2
 cookie=0x0, duration=2.008s, table=0, n_packets=1, n_bytes=98, idle_timeout=60,
 idle_age=2, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=ca:af:3c:f2:00
:b7,dl_dst=8a:b1:d4:88:9c:62,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type=
0,icmp_code=0 actions=output:1
 cookie=0x0, duration=2s, table=0, n_packets=1, n_bytes=98, idle_timeout=60, idl
e_age=2, priority=65535,icmp,in_port=2,vlan_tci=0x0000,dl_src=ca:af:3c:f2:00:b7,
dl_dst=8a:b1:d4:88:9c:62,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type=8,ic
mp_code=0 actions=output:1
 cookie=0x0, duration=1.997s, table=0, n_packets=1, n_bytes=98, idle_timeout=60,
 idle_age=1, priority=65535,icmp,in_port=1,vlan_tci=0x0000,dl_src=8a:b1:d4:88:9c
:62,dl_dst=ca:af:3c:f2:00:b7,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=
0,icmp_code=0 actions=output:2
```

Figure 7: Flow table populated by the controller

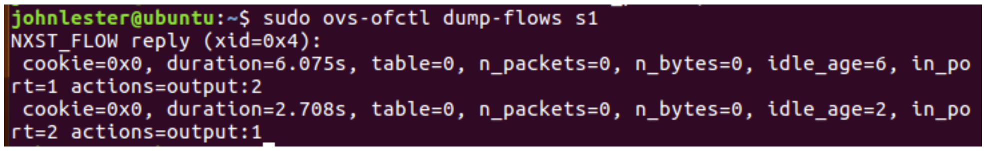
Step 2 - Create Flows in OpenvSwitch (OVS) manually

In this step, the students add the flows rather than have the ovs-controller add the flows to better understand the OpenFlow protocol. To do this, the students start Mininet with no controller for the bridges to connect to.
# mn - -controller=none - -mac

After verifying the flow table is empty and no connectivity exists between the hosts, the students add a simple configuration to enable connectivity between the hosts.

# ovs-ofctl add-flow s1 in_port=1,action=output:2
# ovs-ofctl add-flow s1 in_port=2,action=output:1

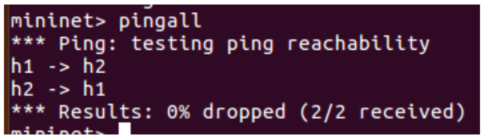
The flow table should now show these two new flow entries.
# ovs-ofctl dump-flows s1



```
johnlester@ubuntu:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6.075s, table=0, n_packets=0, n_bytes=0, idle_age=6, in_po
rt=1 actions=output:2
 cookie=0x0, duration=2.708s, table=0, n_packets=0, n_bytes=0, idle_age=2, in_po
rt=2 actions=output:1
```

Figure 8: Verifying the flows were added

The hosts should now be able to communicate.
mininet>pingall



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
```

Figure 9: Verifying connectivity between the hosts

By the end of Exercise 1, the students would have a clear understanding of the flow structure: both the ones that are created automatically by the SDN controller and manually by a network administrator in the field.

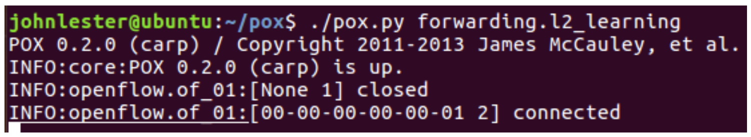
4.2 Exercise 2: OpenFlow Protocols Using Wireshark

So far, the students have seen what goes on at the data-plane level, that is, the packet forwarding plane. Next, the students will see what goes on at the control-plane level, which is the domain of the SDN controller. We will replace the built-in controller (know as ovs-controller) with another that is more flexible and powerful. The control framework the students are asked to use is POX [24] which happens to be bundled with Mininet as well.

Step 1 - POX Forwarding L2 Application

From the POX directory, the students will execute the POX script with the L2 learning module.

$ ./pox.py forwarding.l2_learning



```
johnlester@ubuntu:~/pox$ ./pox.py forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
```

Figure 10: Running the POX l2 learning application

The students are asked then to launch Wireshark and capture on the loopback interface to see the OpenFlow packets between the switch and POX controller.

# wireshark

Note that the dissector for OpenFlow 1.0 does not support all the fields, unlike OpenFlow 1.3, which is fully supported. Wireshark does not support the action structure for OpenFlow 1.0 or the match field for Ethernet type. The information can be deciphered easily by looking at the raw hex data. The students may also have to reference the OpenFlow 1.0 specification for certain fields.

Next, the students will create the Open vSwitch bridge. An OpenFlow 1.0 connection will be established between the Open vSwitch datapath and the POX controller listening at 127.0.0.1:6633.

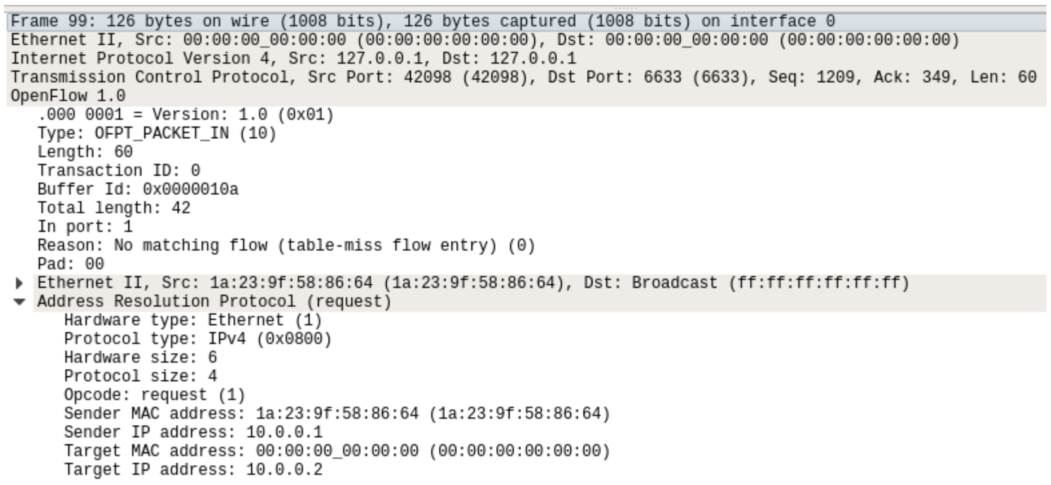
# mn - -controller remote,ip=127.0.0.1 - -switch ovsk,protocols=OpenFlow10

Step 2 - OpenFlow message exchange analysis using Wireshark

The students are to analyze OpenFlow packets using Wireshark. The analysis of OpenFlow packets should allow them to understand what the l2_learning module in POX does. The students then execute the pingall on Mininet.

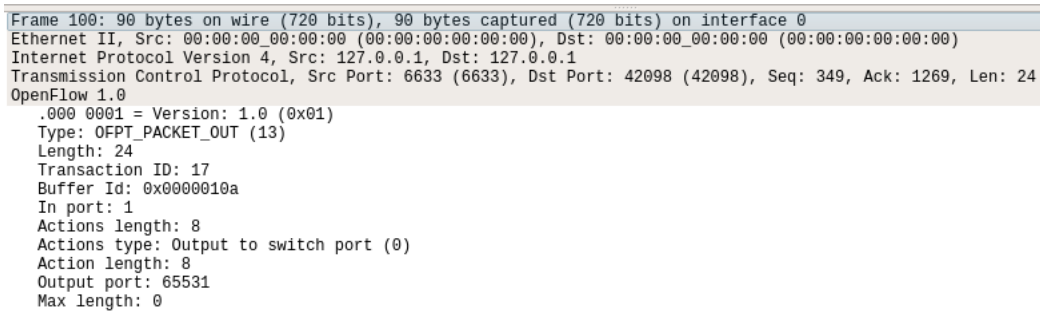
mininet>pingall

The students can view the series of messages (in the form packets) exchanged between the controller and the switch with Wireshark. Examples of the packet in, packet out, and flow mod OpenFlow messages can be seen in Figures 11, 12, and 13. Also, they will learn first-hand encapsulation of packets such as the arp packets within the OpenFlow application layer as seen in Figure 11.



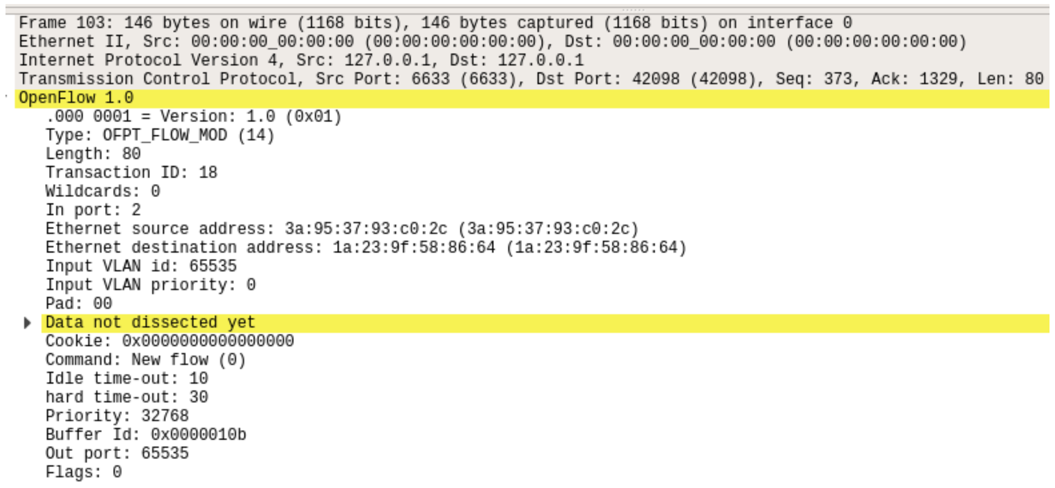
```
Frame 99: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 42098 (42098), Dst Port: 6633 (6633), Seq: 1209, Ack: 349, Len: 60
OpenFlow 1.0
   .000 0001 = Version: 1.0 (0x01)
   Type: OFPT_PACKET_IN (10)
   Length: 60
   Transaction ID: 0
   Buffer Id: 0x0000010a
   Total length: 42
   In port: 1
   Reason: No matching flow (table-miss flow entry) (0)
   Pad: 00
 ▸ Ethernet II, Src: 1a:23:9f:58:86:64 (1a:23:9f:58:86:64), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 ▾ Address Resolution Protocol (request)
      Hardware type: Ethernet (1)
      Protocol type: IPv4 (0x0800)
      Hardware size: 6
      Protocol size: 4
      Opcode: request (1)
      Sender MAC address: 1a:23:9f:58:86:64 (1a:23:9f:58:86:64)
      Sender IP address: 10.0.0.1
      Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
      Target IP address: 10.0.0.2
```

Figure 11: OpenFlow Packet In message encapsulating ARP request



```
Frame 100: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 42098 (42098), Seq: 349, Ack: 1269, Len: 24
OpenFlow 1.0
   .000 0001 = Version: 1.0 (0x01)
   Type: OFPT_PACKET_OUT (13)
   Length: 24
   Transaction ID: 17
   Buffer Id: 0x0000010a
   In port: 1
   Actions length: 8
   Actions type: Output to switch port (0)
   Action length: 8
   Output port: 65531
   Max length: 0
```

Figure 12: OpenFlow Packet Out message encapsulating Flood ARP request



```
Frame 103: 146 bytes on wire (1168 bits), 146 bytes captured (1168 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 42098 (42098), Seq: 373, Ack: 1329, Len: 80
OpenFlow 1.0
    .000 0001 = Version: 1.0 (0x01)
    Type: OFPT_FLOW_MOD (14)
    Length: 80
    Transaction ID: 18
    Wildcards: 0
    In port: 2
    Ethernet source address: 3a:95:37:93:c0:2c (3a:95:37:93:c0:2c)
    Ethernet destination address: 1a:23:9f:58:86:64 (1a:23:9f:58:86:64)
    Input VLAN id: 65535
    Input VLAN priority: 0
    Pad: 00
  ▸ Data not dissected yet
    Cookie: 0x0000000000000000
    Command: New flow (0)
    Idle time-out: 10
    hard time-out: 30
    Priority: 32768
    Buffer Id: 0x0000010b
    Out port: 65535
    Flags: 0
```

Figure 13: OpenFlow Flow Mod message

4.3 Exercise 3: Spanning Tree Protocol (STP) using OpenDayLight (ODL) & RYU

In this exercise, the students will make use of the advanced platform, that is, GNS3 and Docker Containers briefly described earlier in Section 3.3. They will start with a simple network arrangement that consists of three Open vSwitch bridges connected in a loop as shown in Figure 2. In a traditional extended LAN network, the STP is implemented in the bridges and switches. STP allows the switches to elect a root bridge and ensure that each LAN is served by only one bridge. In an SDN-based network, the STP implementation is an application that can be exercised through an SDN controller. An SDN controller translates the application logic into OpenFlow protocol messages to OpenFlow-based devices such as Open vSwitches.

Step 1 - STP SDN application of ODL controller

This step should help the students understand how to use the L2 Switch application through the OpenDayLight (ODL) controller. First, the students launch the ODL controller with karaf distribution using

$ ./bin/karaf

and start ODL web application using

feature:install odl-dluxapps-application

Next, they exercise the L2 switch application, which includes the loop removal capabilities, by executing the following:

feature:install odl-l2switch-switch-ui

Figure 14 shows ODL representation of the network constructed on GNS3 and shown in Figure 2.

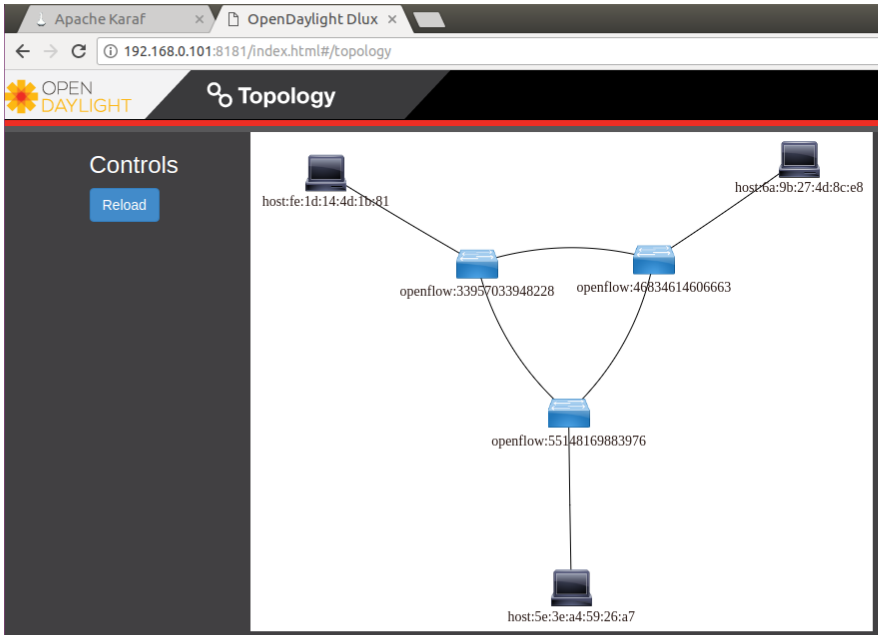
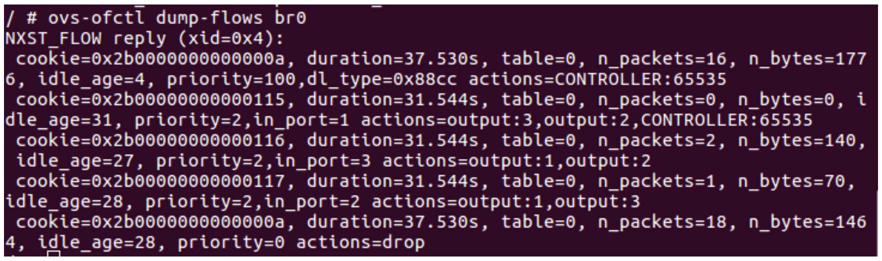


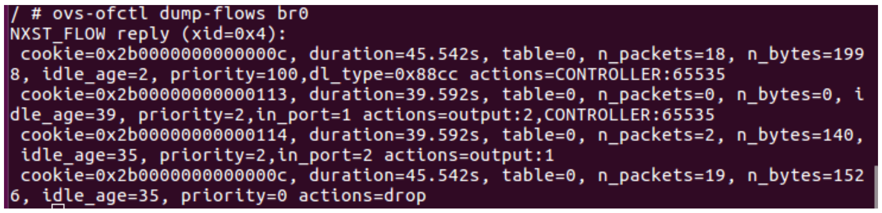
Figure 14: ODL representation of three switches in a loop

To verify that the ODL L2 switch has discovered and prevented the loop, the students examine the flow tables of the three switches, which are shown in Figures 15-17. They should observe (about the flow table of OpenvSwitch-2, in Figure 16) that there is not a flow involving port 3 (eth3). The same is true for port 3 (eth3) of OpenvSwitch-3.



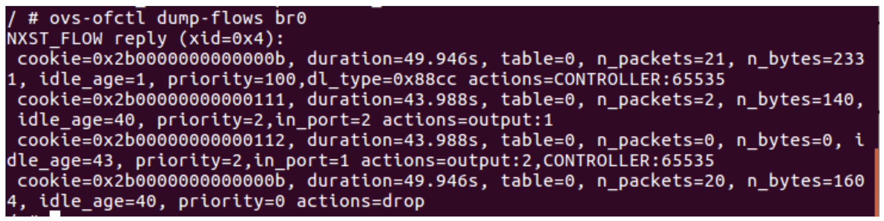
```
/ # ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x2b0000000000000a, duration=37.530s, table=0, n_packets=16, n_bytes=177
6, idle_age=4, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000115, duration=31.544s, table=0, n_packets=0, n_bytes=0, i
dle_age=31, priority=2,in_port=1 actions=output:3,output:2,CONTROLLER:65535
 cookie=0x2b00000000000116, duration=31.544s, table=0, n_packets=2, n_bytes=140,
 idle_age=27, priority=2,in_port=3 actions=output:1,output:2
 cookie=0x2b00000000000117, duration=31.544s, table=0, n_packets=1, n_bytes=70,
idle_age=28, priority=2,in_port=2 actions=output:1,output:3
 cookie=0x2b0000000000000a, duration=37.530s, table=0, n_packets=18, n_bytes=146
4, idle_age=28, priority=0 actions=drop
```

Figure 15: Flow table of OpenvSwitch-1



```
/ # ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x2b0000000000000c, duration=45.542s, table=0, n_packets=18, n_bytes=199
8, idle_age=2, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000113, duration=39.592s, table=0, n_packets=0, n_bytes=0, i
dle_age=39, priority=2,in_port=1 actions=output:2,CONTROLLER:65535
 cookie=0x2b00000000000114, duration=39.592s, table=0, n_packets=2, n_bytes=140,
 idle_age=35, priority=2,in_port=2 actions=output:1
 cookie=0x2b0000000000000c, duration=45.542s, table=0, n_packets=19, n_bytes=152
6, idle_age=35, priority=0 actions=drop
```

Figure 16: Flow table of OpenvSwitch-2



```
/ # ovs-ofctl dump-flows br0
NXST_FLOW reply (xid=0x4):
 cookie=0x2b0000000000000b, duration=49.946s, table=0, n_packets=21, n_bytes=233
1, idle_age=1, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000111, duration=43.988s, table=0, n_packets=2, n_bytes=140,
 idle_age=40, priority=2,in_port=2 actions=output:1
 cookie=0x2b00000000000112, duration=43.988s, table=0, n_packets=0, n_bytes=0, i
dle_age=43, priority=2,in_port=1 actions=output:2,CONTROLLER:65535
 cookie=0x2b0000000000000b, duration=49.946s, table=0, n_packets=20, n_bytes=160
4, idle_age=40, priority=0 actions=drop
```

Figure 17: Flow table of OpenvSwitch-3

Throughout this step of this exercise, the students should gain a strong grasp of how the SDN applications work and how to go about analyzing the openvswitch flow records.

Step 2 - STP SDN application of RYU controller

In this step, the students are to try out a different controller, known as RYU controller on the same network configuration given in Figure 2. However, they are to find out a few interesting facts in the way each of the two controllers implement the STP functionality. In the case of ODL, the odl-l2switch-switch-ui feature blocked eth3 port on both openvswitch-2 and openvswitch-3 rendering the LAN between them unusable. However, in the case of the Ryu controller, the students should find that the RYU team implemented the STP functionality in a way that is almost identical to the STP traditional implementation in currently deployed switches and bridges. In addition, the students are to be presented with a flaw (a bug) in the STP Python code that is responsible for determining the root bridge. In appears that in the Ryu STP application, the value of the bridge ID for the openvswitch is computed in the reverse order. The students are assigned the task to modify the STP Python code of a Ryu SDN application (simple_switch_stp.py) to rectify this issue. This is a great opportunity for the students to get experience first-hand of the paradigm shift in the responsibility of future network analysts and implementers. Today, the network analysts place request for change to the vendor/supplier of the switches or bridges. In the SDN world, they will modify and enhance the application algorithms themselves.

5 Conclusions & Next Steps

We believe that SDN is here to stay and it's just a matter of time before the adoption rate will take off. Taking on the challenge of exploring and developing undergraduate-level hands-on lab exercises has been an extremely rewarding and fulfilling experience. We start the students on a self-contained environment such as Mininet. Once they are comfortable with that environment, they are introduced to more advanced tools that will enable them to explore and experiment with network arrangements where the integration of the physical and emulated devices are required. With the exercises as a starting point, we will rapidly be able to exercise more sophisticated applications for a wide range of algorithms, such as OSPF and BGP. More importantly, we will be able to enhance the undergraduate curriculum to include programming assignments for the students to develop their skills further in modifying and creating new applications and algorithms that would have been nearly unattainable in the traditional networking environment.

It should be noted that more research needs to be done on the overall effectiveness of hands-on learning. The effectiveness of other methods of teaching versus the framework laid out in this paper is to be evaluated in the future.

It is exciting to share our experience with SDN, the tools we adopted and the hands-on lab exercises we developed with those who are interested in getting a head start on this new paradigm in networking functionality and the innovative means of network management and operations. Below is a list of SDN-related topics that we see ourselves pursuing in hands-on research projects and teaching lab exercises in the near future.

- **On-demand, elastic network services utilizing NFV/SDN concept**: open-source architecture - DevStack, Open vSwitch, and VxLAN+Network Services Headers (NSH) encapsulation.
- **Testbed of the Open vSwitch project's Open Virtual Network (OVN)**: Open Virtual Network adds virtual network abstractions to the Open vSwitch bridge.
- **Testbed of the programmable data plane Programming Protocol**:Independent Packet Processors (P4) is another SDN protocol that looks promising by allowing the progammability of data plane protocols in the field.
- **Further research into inter-operability of SDN and traditional networks as well as the differences**: Most operational networks today are brownfield legacy networks. It is important to know if SDN inter-operates with legacy networks. If so, in what ways is it useful and in what ways is it not? Another question is how quickly do SDN networks respond to fail-over compared to legacy. Are they as reliable? Can SDN-based networks provide more insight into the network to predict and mitigate problems in the network?
- **Learning Networking by Reproducing Research Results**: One important use of SDN for undergraduate students is learning networking by reproducing research results. Students in the course described in [25] have used Mininet to replicate published research results.

Acknowledgment

The authors would like to thank the Integrated Science and Technology (ISAT) Department and the College of Science and Engineering (CISE) at James Madison University (JMU) for their support throughout the ISAT Senior Capstone Project process and the development of this paper. Special thanks are to Josie Salcedo, one of the authors senior capstone project partner and Paul Henriksen for his diligence and effort in reviewing and editing this paper.

References

[1] M. C. et al., "Network virtualization in multi-tenant datacenters," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14).*, 2014.

[2] M. Casado, "Origins and evolution of openflow/sdn," in *Open Networking Summit*, 2011.

[3] M. Casado, "Keynote: Make sdn real," in *Open Networking Summit*, 2017.

[4] N. Mckeown, "How sdn will shape networking," in *Open Networking Summit*, 2011.

[5] S. Shenker, "The future of networking, and the past of protocols," in *Open Networking Summit*, 2011.

[6] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 25487–25526, 2017.

[7] H. Farhady, H. Lee, and A. Nakao, "Software-defined networking: A survey," *Computer Networks*, vol. 81, pp. 79 – 95, 2015.

[8] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[9] S. Cosgrove, "Teaching software defined networking: It's not just coding," in *2016 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, pp. 139–144, Dec 2016.

[10] D. C. Sher-DeCusatis, C.J., "Developing a software defined networking curriculum through industry partnerships," in *Conference of the American Society for Engineering Education (ASEE Zone 1)*, 2014.

[11] "Mininet Documentation." https://github.com/mininet/mininet/wiki/Documentation, 2016.

[12] "The software that empowers network professionals.." https://www.gns3.com/, Retrieved, Feb, 2018.

[13] "What is docker?." https://www.docker.com/what-docker, Retrieved, Feb, 2018.

[14] "SDN Architecture Overview." https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf, Retrieved: Feb 2018.

[15] "Understanding the SDN Architecture." https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture, Retrieved Feb, 2018.

[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 Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[17] "Openflow specifications," Retrieved March 18, 2018.

[18] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," *NSDI'15 Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, pp. 117–130, May 04 - 06, 2015.

[19] "The Open vSwitch Database Management Protocol- RFC7047." https://tools.ietf.org/html/rfc7047, 2012.

[20] "Open vSwitch Database Schema." http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf, 2017.

[21] "POX Wiki." https://openflow.stanford.edu/display/ONL/POX+Wiki, 2015.

[22] "Welcome to RYU the Network Operating System(NOS)." http://ryu.readthedocs.io/en/latest/index.html, Retrieved Feb, 2018.

[23] "What is an OpenDaylight Controller? AKA: OpenDaylight Platform." https://www.sdxcentral.com/sdn/definitions/sdn-controllers/opendaylight-controller/, Retrieved Feb, 2018.

[24] "Mininet VM Images." https://github.com/mininet/mininet/wiki/Mininet-VM-Images, 2017.

[25] L. Yan and N. McKeown, "Learning networking by reproducing research results," *SIGCOMM Comput. Commun. Rev.*, vol. 47, pp. 19–26, May 2017.