Octoplus: Software-Defined Networking (SDN) for Undergraduate Education

Eric Alfano, Meaghan Callahan, Mira Yun, Magdy Ellabidy and Chen-Hsiang Yu

Department of Computer Science and Networking
Wentworth Institute of Technology
Boston, MA 02115, USA
{alfanoe, callahanm3, yunm, ellabidym, yuj6}@wit.edu

Abstract

The concept of Software-Defined Networking (SDN) was introduced in 1995, shortly after Sun Microsystems released Java. Although researchers spent a lot of time investigating this topic, it only gets the public's attention in recent years because of the obstacles in IPv6 deployment and the need to extend the Internet. However, this research topic belongs to graduate level and it is not easy to be taught or experimented in undergraduate education for Computer Networking students. In this paper, we propose *Octoplus*, a SDN based project with off-the-shelf, low cost hardware, to bring SDN research into undergraduate education, starting from design decisions, implementation to testing and evaluation. The system consists of different SDN-related hardware and software components, including Banana Pi, Open vSwitch, OpenDaylight, and OpenFlow Manager, etc. The *Octoplus* not only provides a perfect example of introducing advanced research into undergraduate education, but it also demonstrates that off-the-shelf and low cost hardware can be used to teach SDN topic to undergraduate level education. We believe this example can be easily modified and used in other schools' curriculum.

Keywords – SDN, Hands-on Project, Computer Networking Undergraduates.

I. INTRODUCTION

The concept of Software-Defined Networking (SDN) is not new and it was introduced after Sun Microsystems released Java in 1995. AT&T's GeoPlex can be considered as one of the first SDN projects [1]. Although researchers spent a lot of time investigating this topic, it only receives the public's attention in recent years because of the obstacles in IPv6 deployment, IP multicasting services, and the need to have Future Internet. However, since the concept of this topic is difficult and mainly for graduate level, it was not included into undergraduate curriculum of Computer Networking.

Wentworth Institute of Technology is an experiential learning university; along with the theoretical learning experience we give our students the opportunity to have hands-on learning with the most current and trending technologies, including Routing, Switching, Wireless Networks, Network Design, Internet of Things (IoT), Cybersecurity, SDN, etc. SDN is one of those emerging network architecture technologies that allows network administrators to dynacally and centrally manage, control and change network behavior. Basically, the SDN architecture separates the network control from the forwarding functions and allows the network control to be programmable [2].

SDN is a promising technology in terms of simplifying the deployment and operation of networks and lowering the total cost of managing the enterprise. However, this research topic mainly belongs to graduate level and it is not easy to be taught or experimented in undergraduate education for Computer Networking students. Because the content of SDN is theoretical and dry, we not yet bring this topic into regular teaching for undergraduate education, but instead we guide students to hands-on working in a customized SDN project to understand the theories.

In this paper, we propose *Octoplus*, a SDN based project, to bring SDN topic into undergraduate education. The *Octoplus* presents experiment design, design decisions, implementation, testing and evaluation. However, the technology of SDN currently is implemented on very costly switches that range in the tens of thousands of dollars. Another contribution of the *Octoplus* is that it is built with off-the-shelf, low cost hardware, including using a Banana Pi as a SDN switch, running Open vSwitch software (open source) on a virtual machine, etc. This gives students the opportunity to work with SDN technology without the high cost to educational institutions. In the following of this paper, we will describe the *Octoplus*: SDN design and implementation as well as show how we tested our implementation.

II. SDN AND DESIGN DECISIONS

SDN is an architecture that allows computer networking technologies to become more centralized and adaptive [2]-[4]. Network traffic can be shaped as "flows" from a single control plane without having to manipulate separate components. SDN's potential market impact extends from the private domain to networking industry. Benefits include traffic shaping, rapid adjustment of flows, backup of flow configuration, interoperability between vendor-neutral switches, centralized management through a common interface, etc [3]. SDN can also be used to adjust flows based on failure, providing redundancy with minimal latency and zero packet loss [4].

While being one of the most sought after new topics in the computer networking realm today, SDN's infrastructure costs are notorious. Firmware supporting SDN is included only on the newest switches, often costing tens of thousands of dollars. While this equipment is indeed industrial-grade, such quality is often not needed or desired, such as for Proof of Concept and educational investigations. For these reasons, this paper introduces how to build an inexpensive SDN infrastructure for computer networking undergraduates.

One alternative for learning about SDN is to create all the components in a virtual environment, including the switches. The software Mininet [5] can create a virtual network, including SDN switches and simulated end nodes. Another approach is to directly install SDN switching software, Open vSwitch [6], onto a virtual machine, and interact with it through external interfaces. While these are both inexpensive and simple ways for students to quickly explore SDN, there is no physical and dedicated SDN switch.

Having a physical switch which students can touch provides a much more useful and practical way to learn about SDN. Besides being the most common way SDN is actually being implemented, it allows for other devices to be physically connected and influenced by this technology. By including an SDN switch into a lab network architecture, students will be able to learn in a practical sense and examine how their SDN configurations affect a real network.

In this paper, we present *Octoplus*, a SDN project with open source software and inexpensive hardware for undergraduates. The Banana Pi R1[7] was chosen as the switch hardware as it contains 1 Gigabit WAN port and a 4-port Gigabit switch running open source Linux. Due to the switch chip, it was necessary to underlie the SDN switching software with a virtual LAN (VLAN) configuration to properly address the switching ports. This allows the SDN software specific access to each port.

The SDN switching software loaded onto the Banana Pi is Open vSwitch [6]. This software enables the SDN switch functionality, dividing the networking into control, data, and management planes. Although far from industry-reliable standards, this custom switch has a capable processor for Open vSwitch and high-speed Gigabit ports which provide reasonable performance, sufficient for switch's intended use.

The management and control planes' functionalities are implemented via OpenDaylight [8], the largest open source SDN controller. The controller will run from a Linux virtual machine and communicate with the SDN switch using OpenFlow [9]. OpenFlow is the communication protocol that allows for the controller to determine paths (flows) for packets across switches on an SDN network. Flows will be set graphically from OpenFlow Manager (OFM), which is hosted on the same Ubuntu 16.04 virtual machine that OpenDaylight is running on.

OFM sits on top of the controller, as an optional addition to make setting flows more simple and easy for students to understand.

III. BUILDING *OCTOPLUS*

In this section, we present the system architecture of *Octoplus*, starting from design, implementation, testing and evaluation. All components are off-the-shelf, low cost hardware.

The system architecture *Octoplus* is illustrated as Figure 1. The Banana Pi, Open vSwitch, OpenDaylight, and OpenFlow Manager (OFM) are the main SDN-related hardware and software components for the *Octoplus*. Additionally, two Raspberry Pi's were connected to the SDN switch, configured as Apache web servers for testing.

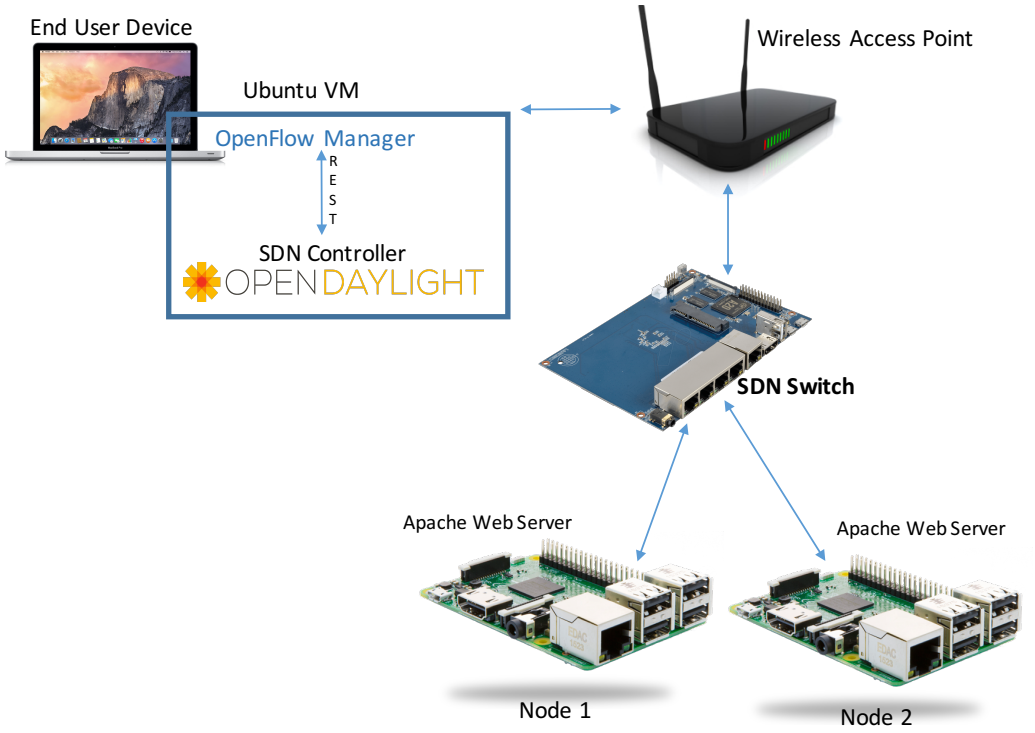

Figure 1. *Octoplus* System Architecture

3.1 SDN Switch Settings

The first step in the process of creating a SDN switch on the Banana Pi is to install and update the operating system, Linux, using Bananian OS 16.04 [10]. On Bananian, preliminary configurations included installing *sudo*, making a new non-root user, adding this user to *sudoers*, and changing the default shell to *bash*.

Because Banana Pi has 5 Ethernet Interfaces, we need to set them up as sub-interfaces which will allow for routing between the interfaces. The Wi-Fi interface needs to be configured with an IP address, to be used as a northbound connection to the controller as shown in Figure 2 (Left).

Next, we need to set up VLAN configurations for each port (port 0-4) on Banana pi by editing *swconfig* file as shown in Figure 2 (Right).

In order for the ports to initialize when the board boots, it is necessary to add commands in the *rc.local* file (*/etc/rc.local*) bringing up each sub-interface, e.g. `ifconfig eth0.10x up`

```
source-directory /etc/network/interfaces.d
auto lo
iface lo inet loopback

auto eth0.101
iface eth0.101 inet manual
auto eth0.102
iface eth0.102 inet manual
auto eth0.103
iface eth0.103 inet manual
auto eth0.104
iface eth0.104 inet manual
auto eth0.105
iface eth0.105 inet manual

allow-hotplug wlan0
auto wlan0

iface wlan0 inet dhcp
       wpa-ssid "SSID"
       wpa-psk "WIFIPASSWORD"
```

```
#!/bin/sh
ifconfig eth0 up
swconfig dev eth0 set reset 1
swconfig dev eth0 set enable_vlan 1
swconfig dev eth0 vlan 101 set ports '3 8t'
swconfig dev eth0 vlan 102 set ports '4 8t'
swconfig dev eth0 vlan 103 set ports '0 8t'
swconfig dev eth0 vlan 104 set ports '1 8t'
swconfig dev eth0 vlan 105 set ports '2 8t'
swconfig dev eth0 set apply 1
```

Figure 2. *Network Interfaces* and *Swconfi*g Configurations: (Left) Configure switch ports and connect switch to Wireless Access Point (Right) Separate ports via VLANs

After the OS and ports are configured properly, Open vSwitch can be installed via *apt-get* as shown in Figure 3 (Top). *Apt* will install the version which is supported by Bananian OS and its v3.4 kernel. Figure 3 (Bottom) command show a virtual bridge, `br0`, being created in the switching software. In order to register the switch with a controller, an IP address must be provided, even if the controller does not yet exist. It also becomes necessary to add the Banana Pi's Ethernet interfaces to this bridge, so that the controller can detect and interact with each one.

```
$ sudo apt-get update
$ sudo apt-get install -y openvswitch-switch openvswitch-common
```

```
$ sudo ovs-vsctl add-br br0
$ sudo ovs-vsctl set-controller br0 tcp:CONTROLLER_IP:6633

$ sudo ovs-vsctl add-port br0 eth0.101
$ sudo ovs-vsctl add-port br0 eth0.102
$ sudo ovs-vsctl add-port br0 eth0.103
$ sudo ovs-vsctl add-port br0 eth0.104
$ sudo ovs-vsctl add-port br0 eth0.105
```

Figure 3. Open vSwitch: (Top) Install Open vSwitch (Bottom) Configure Open vSwitch

3.2 SDN Controller and OFM Settings

Next we create a virtual machine to act as the controller and host the graphical flow manager. Using Ubuntu 16.04 server, we installed OpenDaylight Beryllium and enabled required features. As shown in Figure 4 (Bottom), OpenDaylight features are necessary to enable more capabilities on the controller, adding layer 2 and 3 capabilities, data modeling, a graphical core, and more.

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get install -y oracle-java8-installer

$ wget https://nexus.opendaylight.org/content/repositories/opendaylight.re
lease/org/opendaylight/integration/distribution-karaf/0.4.4-Beryllium-SR4/distribution-
karaf-0.4.4-Beryllium-SR4.tar.gz

$ tar -xvf distribution-karaf-0.4.4-Beryllium-SR4.tar.gz

$ ./distribution-karaf-0.4.4-Beryllium-SR4/bin/karaf
```

```
feature:install odl-restconf-all odl-l2switch-all odl-mdsal-all odl-yangtools-common
odl-dlux-core odl-dlux-node odl-dlux-yangui odl-dlux-yangvisualizer odl-openflowplugin-
all
```

Figure 4. OpenDaylight: (Top) Install Java and OpenDaylight (Bottom) Install features from the OpenDaylight CLI

With the controller running and connected to the SDN switch, we added a graphical component OFM to overlay the controller, making setting flows easy. Not only does OFM allow users to avoid complex OpenDaylight syntax, it also presents all of the options for setting flows in one unified interface. We installed dependencies and pulled the code for OFM, while leaving OpenDaylight running in the background as shown in Figure 5 (Top). *Grunt* is needed to present the OFM JavaScript components. It is necessary to configure the OpenDaylight VM's IP in a grunt configuration file for this to work as shown in Figure 5 (Bottom). Then start the Grunt web server form the OpenDaylight-OpenFlow-App directory, e.g. `grunt.`

```
$ sudo apt-get install -y npm nodejs-legacy git
$ sudo npm install -g grunt-cli

$ git clone http://github.com/CiscoDevNet/OpenDayLight-OpenFlow-App.git
$ cd OpenDayLight-OpenFlow-App/
```

```
baseURL: "ODL_IP_ADDR:",
```

Figure 5. OpenFlow Manager (OFM): (Top) Prepare dependencies and download OFM(Bottom) OFM Configurations - *./ofm/src/common/config/env.module.js*

The OpenDaylight controller and OFM can be be accessed by their specific application ports (OpenDaylight: *http://ODL_IP_ADDR:8181/index.html*, OFM: *http://OFM_IP_ADDR:900*), along with the default credentials. It is important to note that webpage for OFM will not come up properly unless the SDN switch is on and communication has been established with OpenDaylight over OpenFlow.

3.3 Testing and Evaluating the *Octoplus*

To testing the *Octoplus*, we set up two Raspberry Pi's connected to the SDN switch for testing. These nodes were configured with static IP addresses and Apache web servers, each serving a text file designating which host the file resides on. A laptop was also attached to the switch for setting flows and interacting with the nodes.

In order to demonstrate the infrastructure and the capabilities of SDN to manipulate packets in a SDN, we used OFM to set up some simple flows, based on the architecture in Figure 1. The first flow allowed for ICMP ping packets to be redirected from Node 2 to Node 1 without latency or packet loss. The ping shown in in Figure X shows the direction of outbound packets changing from Node 2 (192.168.0 .12) to Node 1 (192.168.0.11).

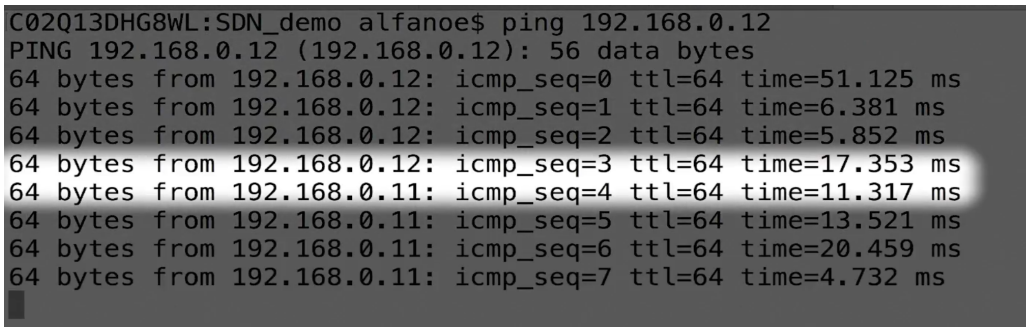

Figure 6. ICMP Ping Packets [Video Demo: https://www.youtube.com/watch?v=2N4YnfKP50w]

The criteria in the tables below depict some of the basic logic of an SDN switch. Packets are matched to certain rules, and the resulting action can modify packets' Ethernet and Network headers before sending the packets on.

Table 1. ICMP Redirection Flow

MATCH: Ethertype: IPv4 Dest. Address: MAC Dest.: TCP Port Dest.: IP Protocol:	 0x800 Node 2 IP - 192.168.0.12/32 Node 2 MAC 80 6
ACTION/MODIFY: IPv4 Dest. Address: MAC Dest.: Out port:	 Node 1 IP - 192.168.0.11/32 Node 1 MAC Node 1 Port

The second flow was similar in that it redirected packets for Node 2 to Node 1. This flow allowed an end device, the laptop, to send a web request to Node 2 but receive a reply from Node 1. Two text files were created on each node, stating the IP address and the node name (Node 1, Node 2) which the file originated from. The flow needed to match destination MAC and IP address, along with TCP port 80 within the packet, and modify MAC and IP addresses to match Node 1.

As shown in Table 2, the flow enables a change in packets going to Node 2 only. By specifying IPv4 address and its type, along with the MAC and HTTP application port 80, this ensures that only web requests destined for Node 2 are affected by the flow.

Table 2. HTTP Flow 1

MATCH: Ethertype: IPv4 Dest. Address: MAC Dest.: TCP Port Dest.: IP Protocol:	 0x800 Node 2 IP - 192.168.0.12/32 Node 2 MAC 80 6
ACTION/MODIFY: IPv4 Dest. Address: MAC Dest.: Out port:	 Node 1 IP - 192.168.0.11/32 Node 1 MAC Node 1 Port

With only HTTP flow 1 set, however, the redirection of the packet will fail due to the TCP handshake. This connection establishment will never complete properly because the first flow has changed the destination to be Node 1, but the end user's device is still expecting the response to be from Node 2. All packet manipulation is done on the switch and is unknown to the end devices. In order to remedy this issue, an additional flow is created matching MAC and IPv4 source address from Node 1, as well as TCP destination port 80. The switch modifies packets which matches the previously mentioned criteria, showing a source IP and MAC address from Node 2. The end user's device which originally requested the file will be unaware that the packet has been manipulated to retrieve the file from Node 1. Students are able to see the TCP handshake failure and subsequent retransmission by monitoring the packets on Wireshark as shown in Figure 7, before adding the second flow to fix this.

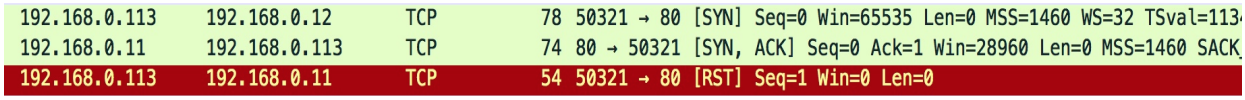

Figure 7. Wireshark: TCP Retransmission

By adding the second flow shown in Table 3, the SDN switch allows Node 2 to respond to web requests that were intended for Node 1, without the requesting PC being aware of the change.

Table 3. HTTP Flow 2

MATCH: Ethertype: IPv4 Source Address: MAC Source: TCP Source Port: IP Protocol:	 0x800 Node 1 IP - 192.168.0.11/32 Node 1 MAC 80 6
ACTION/MODIFY: IPv4 Source Address: MAC Source: Out port:	 Node 2 IP - 192.168.0.12/32 Node 2 MAC Wireless Access Point/Gateway Port

IV. CONCLUSION AND FUTURE WORK

SDN is a promising technology. It not only simplifies the deployment and operation of networks, but it can also lower the total cost of management for the enterprise. However, this topic was mainly for graduate research and was not included in regular curriculum for undergraduate education. Wentworth Institute of Technology is an experiential learning university that offers hands-on learning with the most current and trending technologies. We are working on integrating this topic into our regular teaching. This paper demonstrates one example of our design for SDN topic.

In this paper, we propose *Octoplus*, a SDN based project, to bring SDN topic into undergraduate education. The *Octoplus* starts from concepts and principles, experiment design, decisions for design, implementation, testing and evaluation. To address the cost issue, the system was built with off-the-shelf, low cost hardware, including using a Banana Pi as a SDN switch, running Open vSwitch software (open source) on a virtual machine, etc. The *Octoplus* is designed for undergraduate education of Computer Science and Networking and we believe this example can be easily modified and used in other schools' curriculum. In the future, we plan to extend this architecture into different modules and introduces each module with examples into existing networking courses.

REFERENCES

[1] George Vanecek, "GeoPlex: Universal Service Platform for IP Network-based Services ", CERIAS, October 1997.

[2] T. Nadeau and K. Gray, "SDN: Software Defined Net- works, " O'Reilly Media, Sept. 2013.

[3] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia and W. Kellerer, "Interfaces, attributes, and use cases: A compass for SDN," in IEEE Communications Magazine, vol. 52, no. 6, pp. 210-217, June 2014.

[4] S. Sezer et al., "Are we ready for SDN? Implementation challenges for software-defined networks," in IEEE Communications Magazine, vol. 51, no. 7, pp. 36-43, July 2013.

[5] Mininet, http://mininet.org/

[6] Open vSwitch, http://openvswitch.org/

[7] Banana Pi, http://www.banana-pi.org/

[8] OpenDaylight, https://www.opendaylight.org/

[9] Nick McKeown, et al., "OpenFlow: enabling innovation in campus networks. " SIGCOMM Computer Communication Review, vol. 38, no. 2, pp.69-74, April 2008.

[10] Bananian Linux, https://www.bananian.org/