INTERNATIONAL CONFERENCE
ON INFORMATION & COMMUNICATION TECHNOLOGIES

(ICICT-2017)

Software Defined Networking (SDN)

Ambrose AHURRA[1], Kenneth MAATHE KAMURALI[2],
Diarmuid Ó BRIAIN[3], Dorothy OKELLO[4]
netLabs!UG, Makerere University, Kampala, Uganda

[1]Email: ndahura12@gmail.com, [2]Email:maathek08@gmail.com,
[3]Email:diarmuid@obriain.com, [4]Email:dkokello@cedat.mak.ac.ug

Software Defined Networking (SDN)

Ambrose AHURRA[1], Kenneth MAATHE KAMURALI[2],
Diarmuid Ó BRIAIN[3] , Dorothy OKELLO[4]
netLabs!UG, Makerere University, Kampala, Uganda
[1]Email: ndahura12@gmail.com, [2]Email:maathek08@gmail.com,
[3]Email:diarmuid@obriain.com, [4]Email:dkokello@cedat.mak.ac.ug

Abstract: The legacy network has just about run its last mile, advancements in cloud computing, virtualisation and data centres have led to an exponential growth in data traffic that will soon over-power legacy networks.

Software Defined Networks (SDN) are a new networking paradigm in which there is separation of the forwarding and control planes. The control is migrated to a separate entity called the SDN Controller, Leaving the forwarding plane with bare-metal and virtual switches (switches devoid of control logic) to carry out packet forwarding. SDN allows external applications to program the network via an Application Programmable Interface (API). The most popular SDN protocol is OpenFlow. OpenvSwitch (OvS) is the most widely used OpenFlow switch.

This paper will, investigate the operation and use cases of SDN, demonstrate programmability using the API. Through the building of a physical testbed using Mikrotik RB750GL switches, Raspberry Pi single-board computers and GNU/Linux workstations.

Key Words: SDN, OvS, Mikrotik, ODL.

1 Introduction

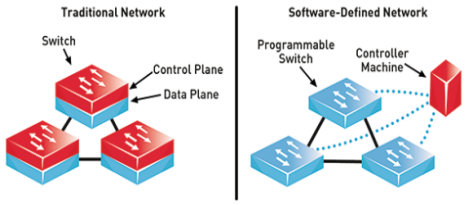

Illustration 1: Legacy Vs SDN network set-up

SDN seeks to enhance network orchestration and management through separation of the control and forwarding planes. SDN can be implemented using both virtual software switches and hardware switches running the OpenFlow protocol. The current networks are heavily reliant on header encapsulation that keeps eating into the Maximum Transfer Unit (MTU) size of packets. The continuing advancements in cloud computing, Internet of Things (IoT) and Virtualisation require a network that is highly flexible and scalable. Illustration 1 demonstrates the logical difference between legacy networks and SDN.

The SDN architecture is demonstrated in Illustration 2, it shows a three tiered architecture where the infrastructure layer is separated from the control layer. The control layer contains the network services in the form of a network policy that controls infrastructure layer devices using the OpenFlow protocol. The network policy can be manipulated by applications in the application layer and thereby they control the switching functionality in the infrastructure layer.

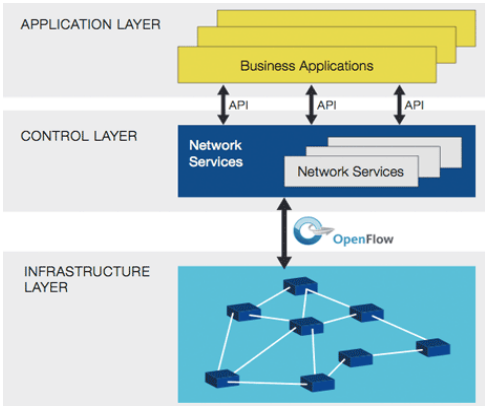

Illustration 2: Basic SDN architecture

2 Objectives

- To build a physical testbed and network to demonstrate how SDN operates.
- Compare the SDN model to legacy network models using performance parameters such as bandwidth, latency and reliability.
- Demonstrate programmability through pushing flows, retrieving network statics using the RESTful (REST) API.

3 Methodology

3.1 OvS on Raspberry Pi

As an experimental approach the OvS is implemented on the Raspberry Pi single board computer. Since the Raspberry Pi has only one on-board Ethernet interface additional interfaces are provided using USB to Ethernet adapters.

Requirements

Raspberry Pi 2B, GNU/Linux Minibian jessie 2015-11-12 release Kernel 4.1.7-v7+, USB to Ethernet adapters, OvSh 2.5.0, ODL Beryllium SR4.

On a fresh install of the Minibian GNU/Linux OS download the OvS.

```
root@ovs2:~# wget http://openvswitch.org/ releases/openvswitch-2.5.0.tar.gz
root@ovs2:~# tar -xzvf openvswitch\\ 2.5.0.tar.gz
root@ovs2:~# cd openvswitch-2.5.0
root@ovs2:~# apt-get update
```

Install build dependencies.

```
root@ovs2:~# apt-get install python-simplejson python-qt4 libssl-dev python-
twisted-conch automake autoconf gcc uml-utilities libtool build-essential pkg-
config
```

Install the GNU/Linux headers to be used used for building OvS kernel module.

```
root@ovs2:~# apt-get install linux-\\ headers-3.16.0-4-rpi
root@ovs2:~# ./boot.sh
root@ovs2:~# ./configure –with linux=/lib/modules/3.16.0-4-rpi/build
root@ovs2:~# make
root@ovs2:~# make install
```

After the install, the kernel module is loaded and OvS is started, a start script is passed to the '*rc.local*' file to always load the the kernel module and start OvS every time the Raspberry Pi boots. After this is complete a check with the '*ovs-vsctl show*' command returned the data path of the switch. '*modinfo openvswitch' returns* the details of the kernel module.

```
root@ovs2:~# ovs-vsctl show e65a62f4-53a6-481d-a395-4f2419d80ce9
root@ovs2:~# modinfo openvswitch
filename: /lib/modules/4.1.7 v7+/kernel/net/openvswitch/openvswitch.ko
license:      GPL
description:  Open vSwitch switching datapath
srcversion:   F83021F5CFFAB96ADDA1C75
depends:
intree:       Y
vermagic:     4.1.7-v7+ SMP preempt mod_unload modversions ARMv7
```

3.2 OpenvSwitch on Mikrotik RB75GL

For a more realistic examination of SDN and OpenFlow, industry standard switching hardware was chosen. The inbuilt router OS of the RB750GL was replaced by the open source router firmware called OpenWrt as demonstrated in Illustration 4, Using the '*Chaos calmer*' release and with the OvS kernel module loaded, the RB750GL was converted into an OpenFlow enabled switch.

Brief description of the RB750GL

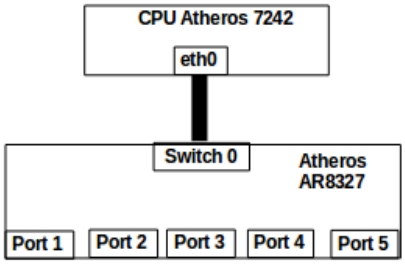

Illustration 3: RB750GL Layout

As can be seen in Error: Reference source not found the RB750GL is single switch Small Office – Home Office (SOHO) device with an Atheros switch and Central

Processing Unit (CPU). It has five Gigabit Ethernet ports.

Requirements

RouterBoard 750GL, OpenWrt Chaos Calmer release, OpenvSwitch opkg package.

OpenWrt is booted onto the router using a Trivial File Transfer Protocol (TFTP) server and router Nand flashed using the wget2nand utility as demonstrated in Illustration 4.

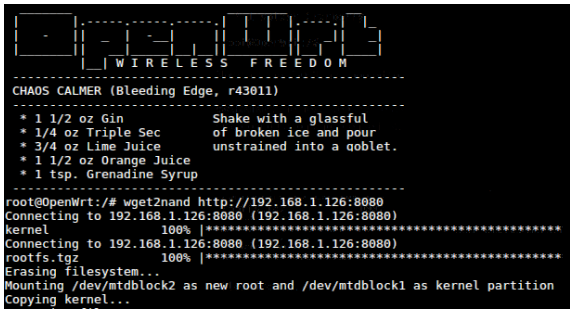

Illustration 4: OpenWrt

The OpenFlow module is then installed.

```
root@OpenWrt:~# opkg update
root@OpenWrt:~# opkg install openvswitch-common openvswitch-ipsec /openvswitch-
switch
```

Confirming the OvS is running properly, this can be seen with the return of the OvS Datapath ID.

```
root@OpenWrt:~# ovs-vsctl show
90039771-aa22-42ed-926d-3b773d7d30ab
root@OpenWrt:~#
```

Finally the network is set up by configuring all the five interfaces. Each port is configured as a single port as follows;

```
# Ethernet/Poe
config switch_vlan
        option device 'switch0'
        option vlan '1'
        option ports '0t 1'
# Ethernet 2
config switch_vlan
        option device 'switch0'
        option vlan '2'
        option ports '0t 2'
```

The rest of the ports are configured in a similar fashion. The resulting interfaces are named following the convention *eth0.vlan_id*. The OvS is configured as follows;

```
# Create bridge
root@OpenWrt:~# ovs-vsctl add-br bro
# Adding interfaces to the bridge
root@OpenWrt:~# ovs-vsctl add-port bro eth0.2
root@OpenWrt:~# ovs-vsctl add-port bro eth0.3
root@OpenWrt:~# ovs-vsctl add-port bro eth0.4
root@OpenWrt:~# ovs-vsctl add-port bro eth0.5
```

Checking the OvS configuration results in the following output:

```
root@OpenWrt:~# ovs-vsctl show 03d9485f-a5de-4d04-81eb-ce642031a96b
    Bridge bro
        Controller "tcp:192.168.5.10"
        Port "eth0.2"
            Interface "eth0.2"
        Port "eth0.3"
            Interface "eth0.3"
        Port bro
            Interface bro
                type: internal
        Port "eth0.4"
            Interface "eth0.4"
        Port "eth0.5"
            Interface "eth0.5"
```

Check the kernel module. (Ttsubo, n.d)

```
root@OpenWrt:~# modinfo openvswitch
module:     /lib/modules/3.10.49/openvswitch.ko
license:    GPL
depends:    libcrc32c, gre
```

4 Technology description

OpenFlow is the protocol that enables the SDN controller to talk to both virtualised and hardware switches. The communication path is via a Secure Sockets Layer (SSL) channel on the Transmission Control Protocol (TCP) port 6633 as demonstrated in Illustration 5.

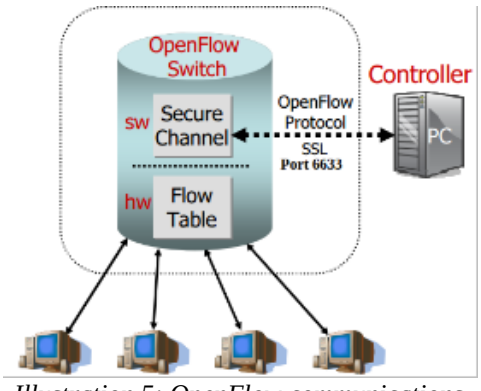

Illustration 5: OpenFlow communications

The SDN controller pushes flows into the flow table to enable communication among the hosts. (Software Defined Networking (SDN), n.d.)

5 Developments

The OpenFlow protocol is now on version 1.4 while the OvS is on version 2.7.0. The ODL has had six releases currently on Carbon. Universities like Stanford University have already deployed SDN. The list of vendors producing OpenFlow enabled Hardware for vitualisation and Data Centre applications has grown including among others HPE, IBM, Extreme Networks.

6 Results

An initial laboratory network was established first using the Raspberry Pi OvS and then the Mikrotik OpenFlow switch and several hosts. ODL displays the network with the hosts as in Illustration 6.

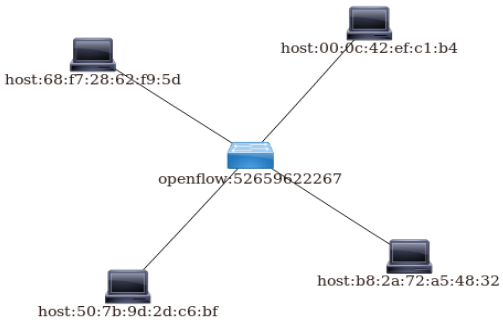

Illustration 6: ODL network topology

6.1 Programming the Switch

The most basic way of pushing flows to the Switches in the Forwarding plane is using the curl utility in GNU/Linux.

Here are the details of the fields for cURL command.

--user <user-name>:<password>:

Specifies the *username* and *password* to use for ODL authentication.

-H Accept: <response-content-type>:

Specifies the *content type* that is expected in the response body for the request. Usually *Accept: application/xml*.

-H Content-type: <request-content-type>:

Specifies the content of the request body. Usually *Content-type: <application/xml>*.

-X <request-type>:

Specifies the *type* of request you want to send to ODL. For example: PUT, GET or DELETE a flow.

-d <request-body>:

Specifies the request body (like Flow, Group, Meter, and so on). This is required for a PUT or POST request only. (“cURL utility,” n.d.)

```
$ cURL –user <username>:<password> -H <header1> -H <header2> -X <request-type>
<url> -d '<request-body>'
```

Tests carried out on the network yielded the following results. Bandwidth testing on the Raspberry Pi OvS.

From Table 1, it is evident that the Raspberry Pi is not robust enough to be used as a yardstick for judging SDN. To this end, Table 2, shows results from the more capable Mikrotik OpenFlow switch.

Table 1: Raspberry Pi OpenvSwitch Performance

Transmission Speeds in Mbits/sec					
TCP			UDP		
Window size	Transfer	Speed	Window size	Transfer	Speed
85.0KB	5.38MB	3.8	208KB	1.25 MB	1.05
128KB	4.75MB	3.79	256KB	1.25 MB	1.05
256KB	4.88MB	3.79	512KB	1.25 MB	1.05

Table 2: Mikrotik OpenFlow Switch Performance

Transmission Speeds in Mbits/sec					
TCP			UDP		
Window Size	Transfer	Speed	Window Size	Transfer	Speed
85.3KB	519 MB	434	208KB	1.25 MB	1.05
128KB	615 MB	516	256KB	1.25 MB	1.05
256KB	607 MB	509	512KB	1.25 MB	1.05

It is prudent to compare the bandwidth performance of the Mikrotik OpenFlow switch and a legacy Mikrotik switch.

Table 3, shows the results from the legacy switch.

Table 3: Legacy Mikrotik Switch Performance

Transmission Speeds in Mbits/sec					
TCP			UDP		
Window Size	Transfer	Speed	Window Size	Transfer	Speed
85.3KB	112 MB	94.1	208KB	1.25 MB	1.05
128KB	112 MB	94.1	256KB	1.25 MB	1.05
256KB	112 MB	94.2	512KB	1.25 MB	1.05

7 Business Benefits

Some of the use cases for SDN include;

Network Access Control (NAC), SDN offers granular level control to set varying privileges for different users and devices in campus networks and or enterprises. (“Six Campus Networks SDN Use Cases That You Need to Know About,” n.d.)

Network virtualisation, creation of abstracted virtual networks on top of physical hardware in the cloud or in large enterprises reducing reducing the deployment time.

Application aware routing is vital in scenarios where data from different applications needs to be separated, treated differently especially delay sensitive applications.

Mobile Network Virtualisation is the Ability to virtualise a mobile network thus allowing multiple operators to share common hardware/infrastructure for multiple networks and allowing different operators to control their own slice of the network. Some deployments will not necessarily employ complete end-to-end virtualisation but might instead choose to virtualise parts of the network. (“Most Common SDN & NFV Use Cases Defined,” n.d.)

8 Conclusion

SDN is a concept that is still taking at the pioneering stage in the networking field. From our investigation it is clear that OpenFlow networks can more than hold their own in terms of hardware performance. However, the real value of SDN lies in its ability to simplify both physical and virtual network orchestration and therefore support the development of elastic network to complement elastic compute and elastic storage as the basis for the Cloud Integrated Network of the future (Weldon, M. K, 2016).

9 Bibliography

cURL utility. (n.d.). [reference]. Retrieved April 4, 2017, from http://www.brocade.com/content/html/en/user-guide/bvc-14-user-guide/GUID-647D854E-84AD-42C0-983D-71464A0DA918.html.

Most Common SDN & NFV Use Cases Defined (n.d.). Retrieved April 3, 2017, from https://www.sdxcentral.com/sdn-nfv-use-cases/

Opendaylight. (n.d.). Retrieved from https://media.readthedocs.org/pdf/opendayli ht/stable-beryllium/opendaylight.pdf

Six Campus Networks SDN Use Cases That You Need to Know About. (n.d.). Retrieved April 3, 2017, from https://www.sdxcentral.com/articles/contributed/sdn-use-cases-campus-networks/2013/07/

Software Defined Networking (SDN). (n.d.). Retrieved April 4, 2017, from http://vtmedia.eu/index-2.html

Ttsubo. (n.d.). RouterBOARD (RB750GL)の OpenFlowOpenvSwitch. Retrieved March 1, 2017, from http://ttsubo.hatenablog.com/entry/2014/11/02/221447

Weldon, M. K. (2016). *The future X network: a Bell Labs perspective*. Crc Press.