OPENFLOW VIRTUALIZATION: A DECLARACTIVE INFRASTRUCTURE OPTIMIZATION SCHEME FOR HIGH PERFORMANCE COMPUTING

Christiana. C. Okezie [1], Okafor Kennedy. C [2], Udeze Chidiebele. C [3]

[1] Electronics andComputer Engineering Department, Nnamdi Azikiwe University, Awka,
[2, 3] R&D Department, Electronics Development Institute, Awka.
NIGERIA.

[1] christanaobioma@yahoo.com, [2] arissyncline@yahoo.com, [3] udezechidi@yahoo.com

ABSTRACT

Contempriarily, networks have become a critical component of all infrastructures in High Performance Computing (HPC) environments. With the wide acceptance of cloud computing, 4G long term eveolution (LTE), Smart grid integrated iTs and other internetworking applications, a highly optimized scheme is recommended. This scheme will give network operators more control of their infrastructure, allowing customization and optimization, thereby reducing overall capital and operational costs. This paper presents Openflow virtualization in layer 2 switch which serves as a uniform vendor interface between control and data planes. The operating system logically constructs and presents a logical map of the entire network to services or control applications implemented on top of it, hence slicing and virtualizing the underlying network. With this, a Synthesis Service Differenciation (SSD) model was introduced by writing internetwork operating system (IOS) scripts that manipulates the logical map of the switch slice with IP address ranges. The result is an increased ability to quickly introduce new services and to adapt the network faster when service changes are required in HPCs.

Keywords: Openflow, HPC, 4G, Infrastructures, LTE, Virtualization, Vendors, SSD, IOS, HPCs

INTRODUCTION

According to the website in [1], OpenFlow is a communications protocol that gives access to the forwarding plane of a network switch or router over the network. Recent technical advances have moved a large proportion of locally hosted enterprise services from private, in-house machine rooms to shared datacenters maintained by third party providers. This shift was enabled by the introduction of cloud computing and infrastructure virtualization [2]. OpenFlow which is an instance of software-defined networking (SDN) gives access deep within the network forwarding plane while providing a common, simple, API for network-device control [3].

We believe that just as vendors design switches, routers and other products for specific markets, OpenFlow technology seeks to break the propriatory constraints attached to devices by the vendors, hence creating an open platform for third party integrations. OpenFlow provides network administrators with a set of elements that allows them to define flows and to define the path that they will follow without disturbing the existing traffic. It also provides methods to define policies to find automatically paths that accomplish certain characteristics, like having higher band width, suffering less latency, reducing the number of hops and reducing the required energy that needs traffic to reach its destination [4]. With OpenFlow virtualization, it becomes possible to create an open standard that can be implemented in ethernet switches, routers and wireless access points (AP) regardless of the vendor firmwares.

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 4 July 2013

Since OpenFlow is a relatively new technology currently under deployment, several research issues that should be solved among others are robustness, ease of management, performance and scalability [5]. It is practicable to delegate the management of network segment/s to experts, as if it was a completely independent network. OpenFlow switches are deployed into UNIX/Linux platforms and available in ethernet switches/routers from different vendors [5].

Following the complexity and resource requirements of today's data center networks, this paper presents OpenFlow Virtualization with Synthesis Service Differenciation (SSD) model to enable new services and integrations to adapt to any network faster when service changes in high computation envirnments.

A BRIEF OVERVIEW ON OPENFLOW HARDWARE ABSTRACTION

According to [6], the draft document describes the OpenFlow Hardware Abstraction API as a programming interface inside the OpenFlow switch software architecture. The interface provides isolation between vendor specific hardware details and OpenFlow switch implementations. It abstracts the physical hardware port to an OpenFlow port type and the hardware packet processing tables to show manipulation operations. Figure 1 illustrates how the OpenFlow Hardware Abstraction API fits into the software architecture of an OpenFlow switch.

The purpose of the OpenFlow HW Abstraction API is to allow the software components of an OpenFlow Switch to control hardware Datapath and hardware port interfaces. It provides an extensible mechanism to expose vendor specific hardware features to the OpenFlow controller when such exposure is necessary or useful. Some abstracted components include [6]:

I. Port Interface: A conduit through which packets may be sent or received. Typically associated to a physical port, but some abstractions may group physical ports (e.g. trunking) or associate multiple interfaces to a physical port (e.g., each with a different VLAN). An interface may have additional state such as \up/down" or spanning tree state. In addition, an interface may maintain statistics on its use.

II. Flow Table and Datapath: A Flow Table is a prioritized list of flow descriptions, each with associated actions and may be implemented in hardware or in software. There may be multiple Flow Tables in a switch (for instance, one in hardware, one in software). Each Flow Table is associated with a Datapath. The Datapath accepts packets to be processed according to the rules in its Flow Table. It updates the packets and forwards them according to the actions of the matching flow entry. Packets may arrive from a port interface on the switch or from the OpenFlow protocol stack.

III. OpenFlow Protocol Stack: Makes connection with a Controller and passes off commands as they arrive.

IV. OpenFlow Datapath: The main controlling component for OpenFlow. It usually maintains a software flow table and processes packets from the datapath. It coordinates the multiple flow tables (software and hardware) in the system. It handles packets arriving from the hardware to be processed by the software flow table or forwarded to the controller.

V. Port Manager: Implements the needed functionality to expose a port from the hardware (physical or otherwise) to OpenFlow. May provide some control or expose port state (enabled, link state, spanning tree state, etc.) Allows access to port statistics. Packet data flows from the hardware through the port manager to the OpenFlow datapath.

VI. Hardware Table Manager: Implements the needed functionality to expose hardware packet processing tables to OpenFlow. It Converts OpenFlow flow insertion messages into hardware specific table entries and allows access to flow table and per-flow statistics.

The driver structure in the OS kernel supports communication from one module to another during the initialization phase

The OpenFlow HW Abstraction API has the following goals:

I. Protect Proprietary Code: Vendor proprietary code may be written to support this API and released as a binary to protect the intellectual property of the vendor.

II. Insulate OpenFlow Switch Code: Allow the OpenFlow Switch software components to be written independent of the underlying hardware. This will facilitate porting the OpenFlow Switch code to new hardware.

III. Insulate Vendor Code: When a new version of OpenFlow is released, this API should permit the reuse of existing HW drivers which support this API. This is particularly important if the HW driver is released as a binary object.

IV. Flexible Deployments: In environments such as Linux with a user/kernel space distinction, the API should function on either side of the division or across it. More generally, the API should not impede the creation of library to support the functionality.

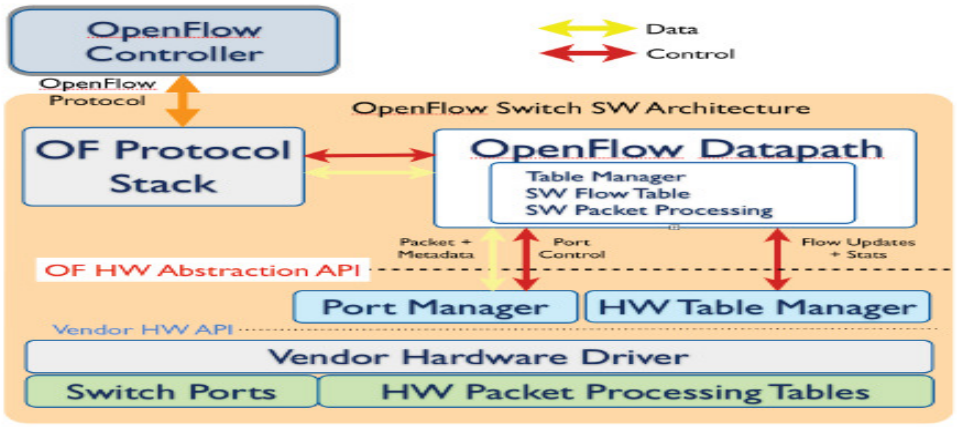

Figure 1. High Level OpenFlow Switch Software Architecture [6]

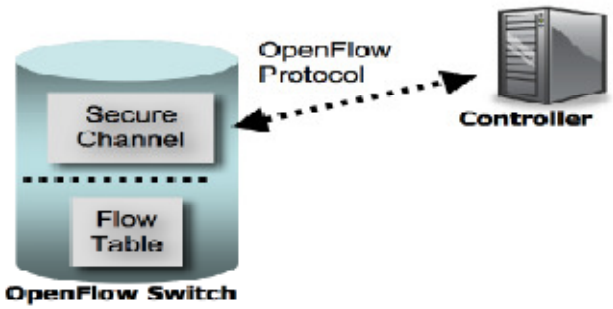

Figure 2. An OpenFlow switch communication with a controller over a secure connection using the OpenFlow protocol [7].

An OpenFlow Switch consists of a flow table, which performs packet lookup and forwarding, and a secure channel to an external controller (Figure 2). The controller manages the switch

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 4 July 2013

over the secure channel using the OpenFlow. The Flow table comprises of the header fields, actions and counters. Our proposed OpenFlow Virtualization with Synthesis Service Differentiation (SSD) model will leverage on the Openflow hardware abstraction to implement SSD. Figure 3i & 3ii shows a closed and openflow Proprietary switch models.

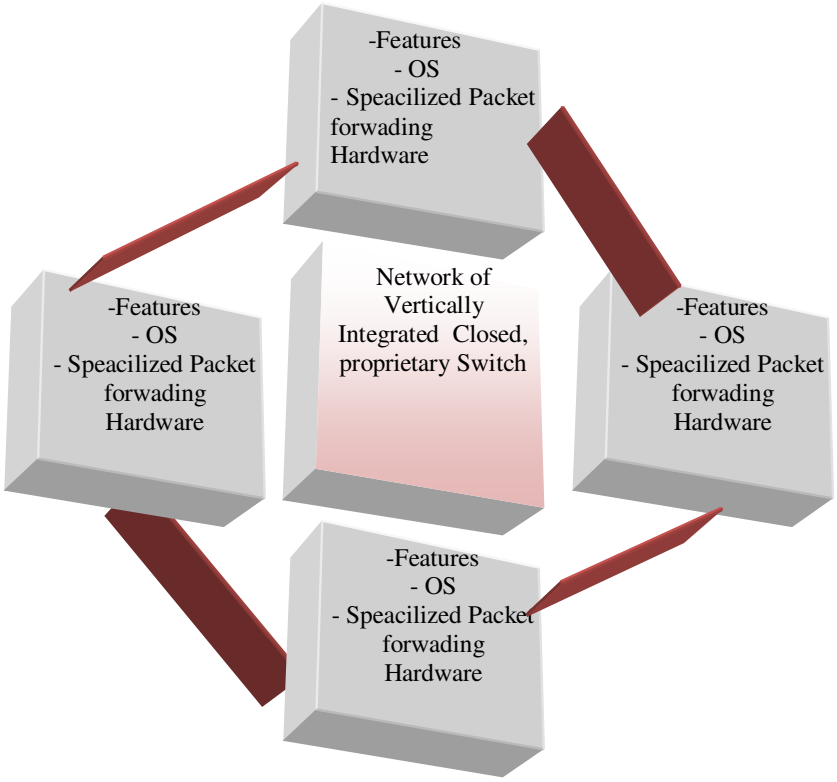

Figure 3i. A closed/ Proprietary switch model

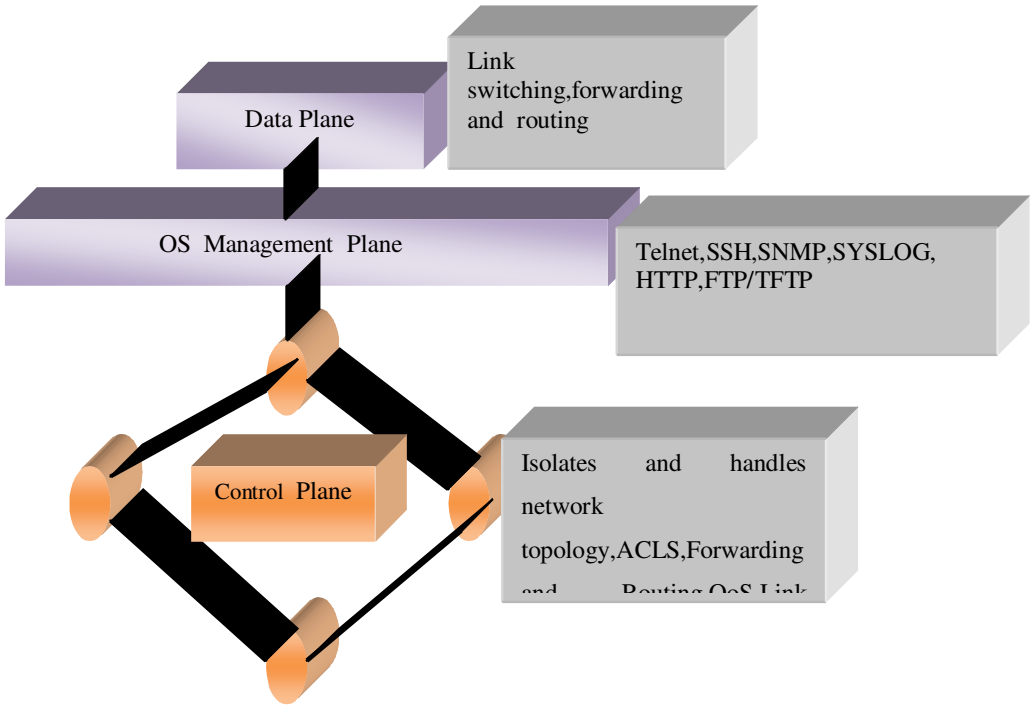

Figure 3ii. OpenFlow/SDN switch model

In figure 3ii, the model separates between data plane and control planes, opens interface between control and data plane while the network control and management is handled by the OS kernel.

RELATED WORKS

Though OpenFlow technlogy is relatively new, various works were reviewed to allow this research make contribution to knowledge. In this work, OpenFlow Virtualization draws inspiration from the works in [5], [15], [16], and [17]. The works advocates controlling network switches from a separate, logically centralized system. The paper in [5] is by far the most related research work. The work outlined the shortcomings of current START architecture [18] viz: Access control is too coarse-grained, hosts cannot be dynamically remapped to different portions of the network, and monitoring is not continuous. The authors in [5] proposed an architecture that has the following salient features as address issues in [18]: a policy specification framework, distributed network monitoring, and the ability to take specific actions using programmable switches. The works in [2], [13], [14] discused OpenFlow as a cost reduction agent for computing infrastrucuture while the paper in [4] presented OpenFlow Switching Performance in the context of testbed deployment, comparision of performance with layer-2 Ethernet Switching technology and with layer-3 IP Routing technology. The work submits that OpenFlow can do the same functions as switching and routing with the same and, in some situations, better performance. OpenFlow devices identify traffic flows following forwarding rules established by network managers and it also provides a flow-based network virtualization framework to avoid interferences among different flows [5].

However, this paper discusses Synthesis Service Differentiation (SSD) approach for priority tagging to devices based on thier priority index in HPCs and carries out an evalutaion on Openflow virtualization experiments. Figure 5 shows our SSD DCN Architecture.

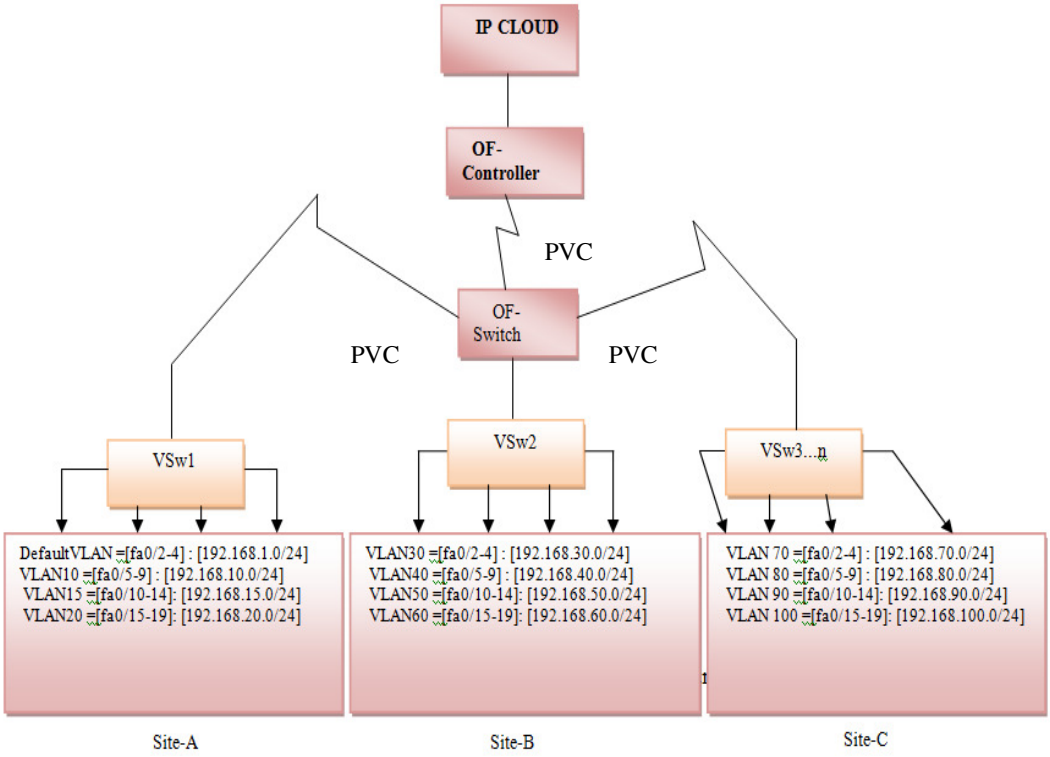

Figure 5. SSD DCN Architecture

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 4 July 2013

METHODOLOGY

This work adopts two approaches to analyze Openflow virtualization and performance characteristics of Synthesis Service Differenciation (SSD) in HPCs. Firstly, our SSD model and computation was done to assign priority tags to end devices using comon IP VLAN design mapping and in the second approach, we focused on openflow switch data path and analyze the OpenFlow implementation in Linux based PCs. Forwarding throughput and packet latency is analysed and compared with layer-2 Ethernet switching under loaded, underloaded and overloaded conditions with different traffic patterns. In this work, the first approach will only be addressed while future work will address the later.

Experimental Setup

At Kswitch Labs, we have several devices and softwares in which we can test the experiments. For the completion of the work, we are going to use one desktop PCs, one laptop and one traffic generator. The following components were deployed for OpenFlow virtualization analysis:

Desktop PC

- CPU: Intel Core2 Duo E6750 2.66 Ghz
- RAM: 4 GB 2066MHz
- HARD DISK:80 GB
- NIC: 2 x Intel PRO/1000 PT dual port 1 Gbps PCIx-Express
- Operating System: Linux Ubuntu 10.10 64 bit. Kernel 2.6.27

Mobile Laptop

- CPU: Intel Core2 Duo P8400 2,6 Ghz
- RAM: 8 GB 800MHz
- HARD DISk: 500 GB
- NIC: Realtek RTL8168 Gigabit Ethernet
- Operating System: Linux Ubuntu 10.10 64 bit. Kernel 2.6.27

Agilent N2X traffic generator

- Modular chassis: up to 4 modules
- Used modules: E7919A 1000Base-X GBIC-RJ45 and E7919A 1000Base-
- N2X Packets 6.4 System Release traffic analyzer software

Packet Tracer 5.3 for IP VLAN mapping (Priority tagging)

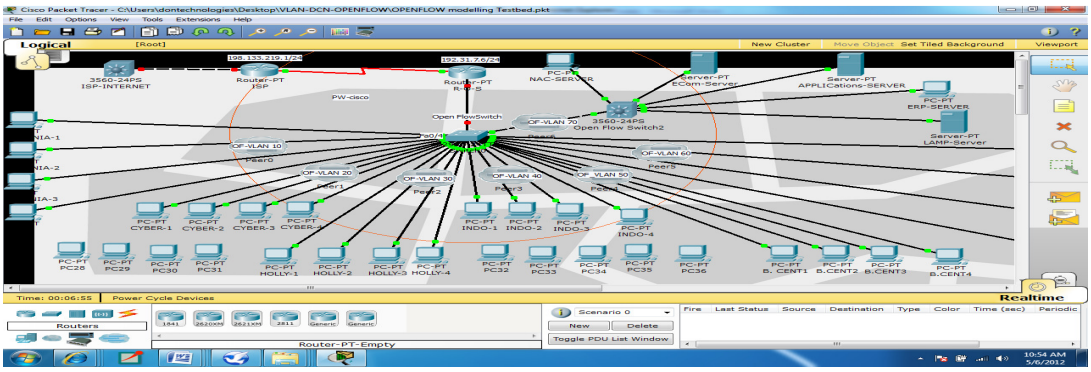

Figure 4i. Testbed environment for SSD with Packet tracer 5.3. (Senario-1)

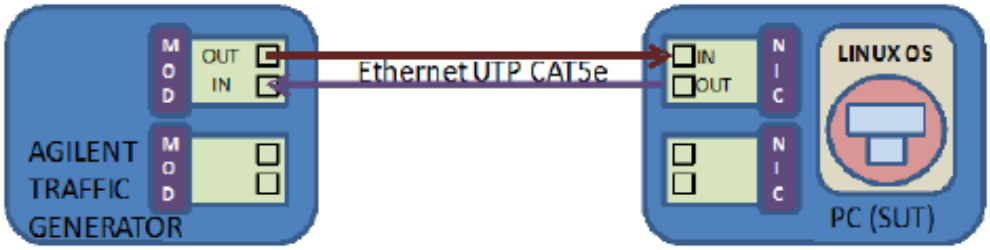

Figure 4ii. Testbed environment for Ageilent N2X router (Senario-2)

The experimental setup for our second approach is shown in Figure 4ii. The test machine (TM) is a standard PC mounted where OpenFlow switching application was installed for initial contact. The configuration of the PC is an Intel Core2 Duo E6750 2.66 Ghz with RAM: 4 GB 2066MHz, 80 GB HARD DISK. Network connectivity is provided by two Intel PRO/1000 PT dual port1 Gbit/s NICs plugged into PCI-Express-x bus running Operating System-. Linux Ubuntu 10.10 64 bit Kernel 2.6.27. The test is to be based on synthetic traffic generated using an Agilent N2X router tester [8] equipped with Gigabit Ethernet modules. The use of the router tester ensures both bottleneckfree traffic generation and high measurement precision not achievable with standard PCs and software traffic generators. All links use UTP cat 5e cabling. Switching tests use directly the Linux kernel implementation while bridge-utils software suite is used to create a virtual switch and to connect the test interfaces to it. Routing is performed using the IP forwarding engine available in the Linux kernel. We present the results and analysis for this senario in our future work.

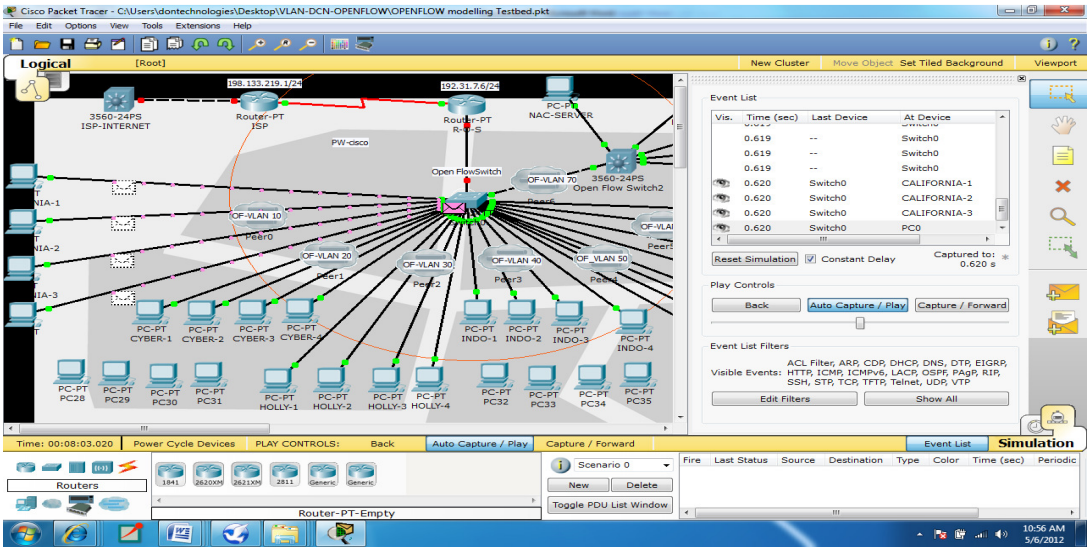

Figure 5. A VLAN IP Mapping simulation for SSD with Packet tracer 5.3. (Senario-1)

SIMULATION AND PROCESS MODEL RESULTS (Senario-1)

Openflow Ssd Vlan Ip Mapping

Basically, twelve VLAN mappings (for site a, site B and site C) were used for the Synthesis service differentiation. VLAN pririority tagging for different services and traffic was realised via IOS scripting on our OpenFlow switch. By using variable subnet mask concept, (VLSM) [9], the corresponding valid SSD real IP ranges were computed with the SSD priority tags (1, 10,15, 20, 30, 40, 50, 60, 70, 80, 90,100) matched with the port interfaces in the OF-switch. Figure 5 shows flow table ingress in OpenFlow switch model as discussed in [10]

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 4 July 2013

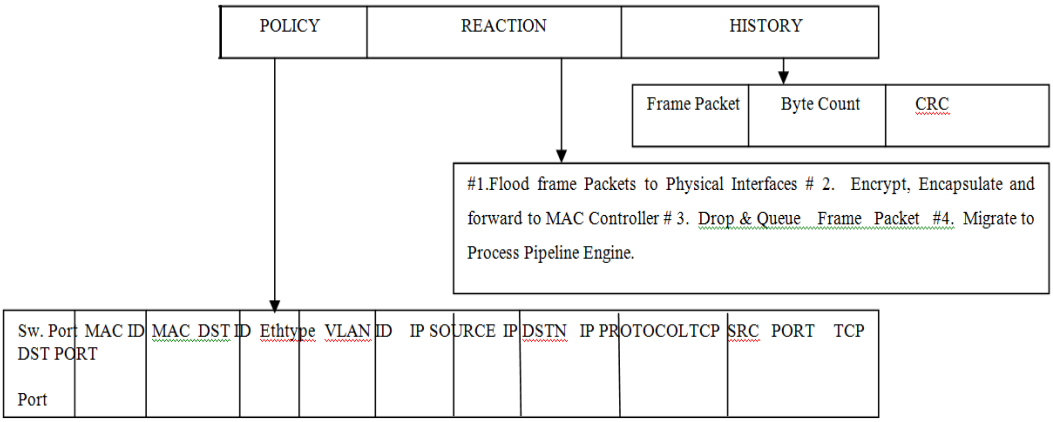

Figure 5. Flow table Ingress in OpenFlow Switch Model [10]

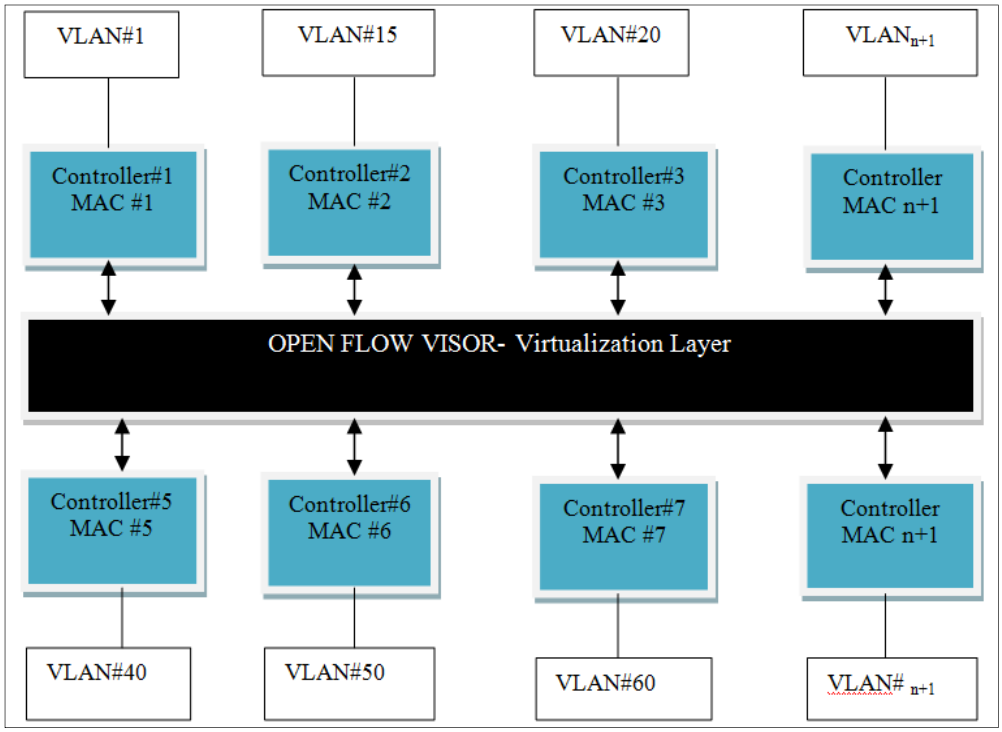

Figure 6. An OpenFlow Virtualized Switch model [10]

Analysis of Synthesis Service Differentiation (Ssd)

A model of an OpenFlow Virtualized switch model is shown in figure 6. OpenFlow Virtualization and SSD offer high bandwidth optimization with low latency and consequent high through put in a HPC link. In the context of senario-1, this paper used a three source star topology Virtual Logical Aggreation Network (VLAN) in the model to define SSD for various traffic flows. The three source topolgy was mapped into twelve VLANs to be assigned to end system/workgroups locations as shown in figure 5. The simulation deals with twelve different cases for three different sites. the first case is devoted to site A (VLAN 1, VLAN 15, and VLAN 20), the second case is devoted to site B in which nodes are mapped to VLAN 30,VLAN 40,VLAN 50, and VLAN 60. The third case is devoted to site C (VLAN 70, VLAN 80, VLAN 90, and VLAN 100). In all cases, the SSD which defines VLAN mapping is done by port and MAC in the Openflow switch (OF-switch) as in figure 5.

Starting from VLAN 1 to VLAN 100 for site A, site B and site C, the priority tags were assigned for resource allocation and traffic management with due consideration to the OpenFlow virtualization on the OpenFlow switch. Also, only SVLAN 70 was assigned to the DCN servers. In the SSD VLAN modelling, Matlab Simevent [11] was used to create a process model for real QoS analysis while Packet tracer [12] was used to realise the priority tagging for the end systems via VLAN setup as shown in figure 5. The figures in the next section are plots from the simulation process model. Considering the OpenFlow switch capacities (buffer sizes), offered load intensity, throughput, queuing delay, latency and resource utilization, the following input parameters were utilized considering a workload applications for our OpenFlow switch. Table 5 shows our MATLAB Simevent process model parameters. The SSD VLAN backbone in this work considered the following network design issues: Loops traffic, Convergence, Broadcasts, Subnetworking, Security, media dependence.

Table 5. MATLAB Simevent process model parameters

Attributes	*Values*
Traffic Distribution	Exponential
Service Time	1sec.
Number of VLAN	12
Configuration Register	0x2142
Cable length between each nodes	100m with GB ethernet
Maximum packet size (MTU)	1500 bytes
Packet generation rate for each node packet/s	100
Entity Type	Standard
Generation Event Priority	300 bytes
Number of Entity Output Port	12
Number of Entity Input Port	12

RESULTS AND DISCUSSION

The results obtained from the VLAN MATLAB process model were presented and the various plots shown in Figures 6 to 6.3. The simulation was run for emulated OpenFlowVLAN core switch for a flow table size 25 to 1000 for different packet generation rates (arrival rates). Our benchmark setup uses the Simevent application, from the Mathworks platform. The emulated OpenFlow switch generates packet-in messages and we computed the performance characteristics of the controller in processing requests. It provides us with entities, attributes, subsytem block and FIFO blocks modelling the operation in abstract contexts. We then measured system behaviour in terms of packet-in requests processed per second. Throughput, OpenFlow table size against delay, resource utilization SSD latency plots were obtained for packet-in message flooded on the FIFO switch which now forms outgoing packet-in messages with frame payload. Figure 6 shows measures the throughput of the controller for service requests fairly assigned. The plot shows an acceptable response for the differentiated services

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 4 July 2013

on the VLANs. Regardless of the channel limitations, we conclude that OpenFlow switching is a better alternative to software Ethernet Switching or IP Routing because it does the same layer-2 and layer-3 functions with a high performance and scalability as shown in Figure 6.

Figure 6.1 depicts the plot of OpenFlow Size table against Queung delay. It is shown that the flow size is not affected by queuing delay, as such in normal mode, we expect a system that synthesizes and handles requests logically without any prior delays, hence improving the performance index of the system.

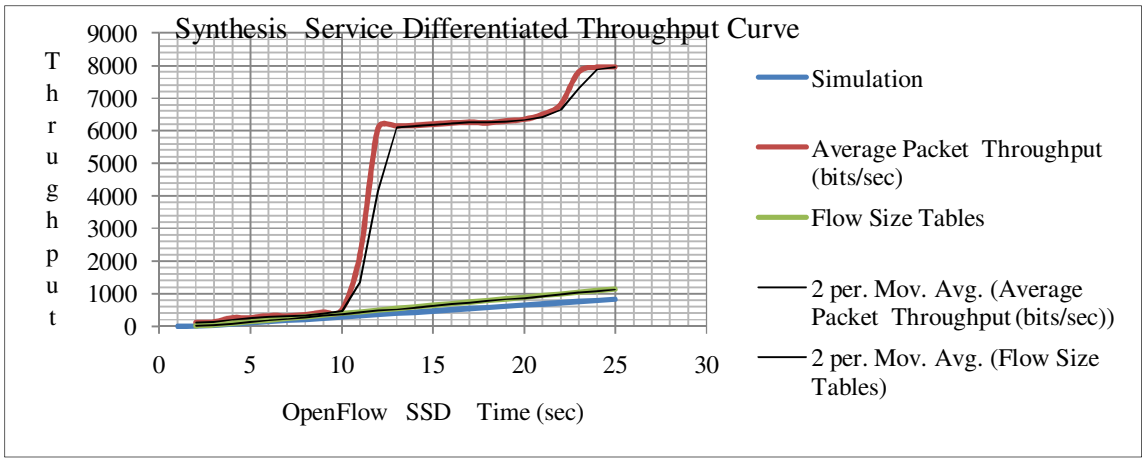

Figure 6. A SDD Throughput Curve

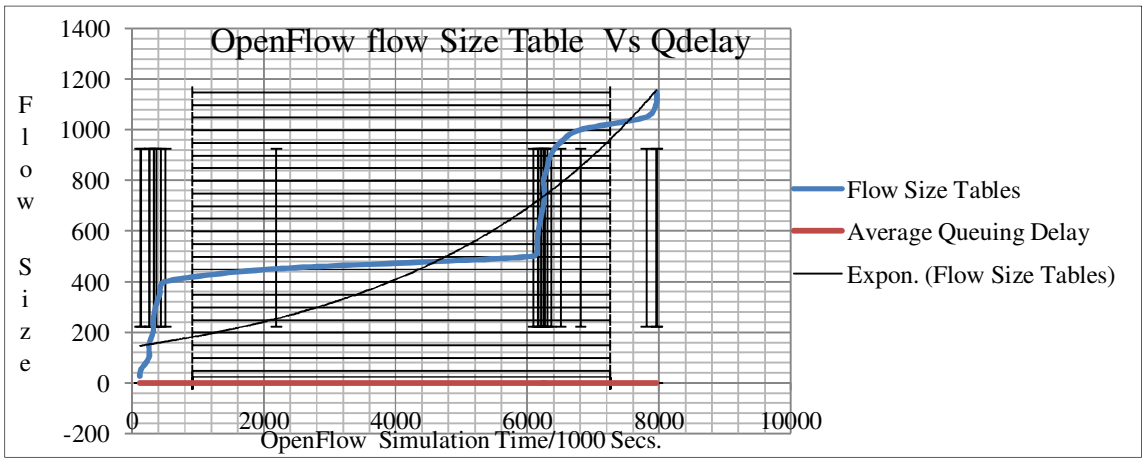

Figure 6.1. A plot of OpenFlow Flow Size Table against Queung delay

In figure 6.2, our control plane switch is connected to the controller, serving 100 distinct MAC addresses. In life testbed, the experiment is be run on a 32-core intel server running on a linux debian platform with 64GB of RAM and each controller configured to use a single thread of execution. However, owing to our future work, we restrict our analysis to a single-thread only (not supporting multi-threading functionality). For the simulation run between the switch and the controller, we run and measure the per-second rate of successful interactions. The resource utilization plot was shown in figure 6.2. In the VLAN data environment (HPC), latency, throughput, utilization, bandwidth is vital resources considered in traffic management. With the traffic load sources in the twelve star topology used in this paper, a final gradient was gradually established after a 0% to 40% and then a 100% throughput hit at the port 11 handling the node at that instance. Afterwards, resource allocations were fairly distributed. With a connection request, feasible regions of resource allocation are first established. From the test bed, enterprise servers are dumped in site C (VLAN 70, VLAN 80, VLAN 90 and VLAN 100). This

corresponds with the load sources 8, 9 and 10 from the plot. Resource utilization in these regions is quite high, but these are the regions with high priority tags and as such the computational power of the DCN servers are high. Thus, in DCNs, the regions of high priority tags have the highest resource utilization cycles.

Figure 6.3 shows the SSD latency plot. Basically, there are three sources of latency in conventional ethernet switches viz: wireline latency, switch fabric latency, and store and forward latency. It is known that OpenFlow allows the path of network packets through the network of switches to be determined by software running on cascaded devices. This separation of the control, allows for more sophisticated traffic management and as such gives room for improved latency response. We observe that at zero load time giving the load sources from the plot, the connection requests has been established achieveing a near 100% throughput in the model with a latency of less than 0.5µsec. This gives a better improvement over the conventional models. This literally depicts a high performance switched model since low latency networks have negative or zero initial load times.

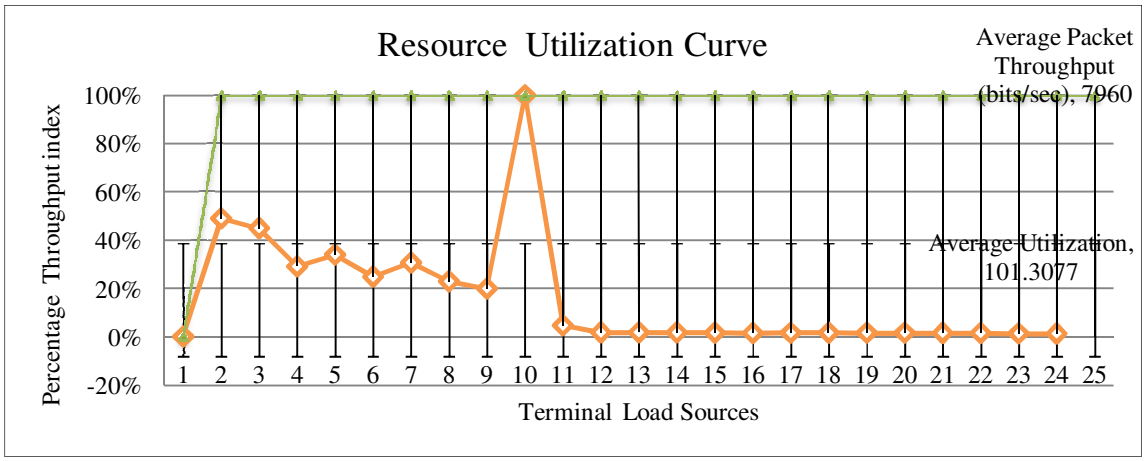

Figure 6.2. Resource Utilization Plot

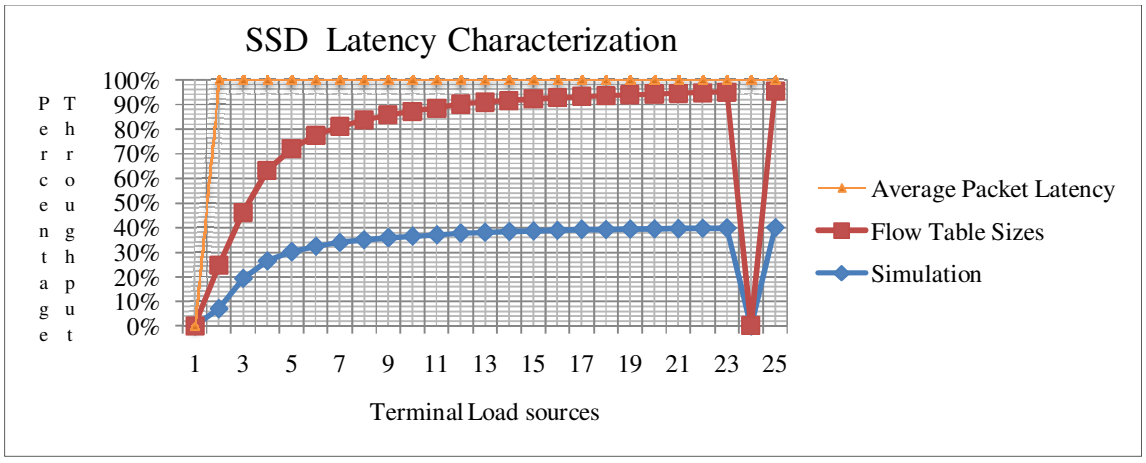

Figure 6.3. SSD latency character

CONCLUSION

This paper has a Synthesis Service Differenciation (SSD) model in the context of OpenFlow virtualization. For HPC market segemnts, an implementation of OpenFlow algorithms in layer 2 and layer 3 devices will offer excellent performance as our metrics of evaluation gives a better indication. The SSD DCN architecture is presented with its process model developed

ISSN-L: 2223-9553, ISSN: 2223-9944
Vol. 4 No. 4 July 2013

with MATLAB simevent for our data collection via the workspace command prompt. Motivated by existing literature on HPC datacenter networks and the challenges poised the current monolithic design of the underlying network; we implemented the SSD model as a way of assigning priority protocol to end devices for better management of the OpenFlow switch framework. With OpenFlow virtualization, we believe that this will provide an efficient network programming API which will integrate with base network control plane and data plane logics. This allows application integration to exercise flow level control with OpenFlow protocol thereby allowing the underlying infrastructure to achieve an efficient network performance.

Finally with OpenFlow virtualization, deployment flexibility, cost effectiveness and improved computational requirements will be achieved.

We are currently working on Openflow switch data path as well as OpenFlow implementation in Linux based setup. This will form part of our future work.

REFERENCES

[1]. OpenFlow, www.wikipedia.org/wiki/openflow

[2]. Charalampos Rotsos, Richard Mortier, Anil Madhavapeddy, Balraj Singh, Andrew W. Moore, "Cost*, Performance & Flexibility in OpenFlow: Pick Three",(unpublished).*

[3]. Charalampos Rotsos, Nadi Sarrar, Steve Uhlig , Rob Sherwood and Andrew W. Moore, *"OFLOPS: An Open Framework for OpenFlow Switch Evaluation"(unpublished)*

[4]. Torino, P. D. (2009). *OpenFlow Switching Performance*, Masters Thesis.

[5]. Bianco, A., Birke, R., Giraudo, L. & Palacin, M. (2010). *OpenFlow Switching: Data Plane Performance*, In the IEEE ICC proceedings, pp.1-5.

[6]. Erickson, D., Hara, M., Heller, B., Lantz, B., Pettit, J., Pfaff, B. & Talayc, D. (2009). *OpenFlow Hardware Abstraction API Specification*," DRAFT Version 0.4 based on OpenFlow 0.9.0, September 16.

[7]. Whitepaper: (http://OpenFlowSwitch.org)

[8]. Agilent N2X router tester. Web site: http://advanced.comms.agilent.com/n2x/

[9]. Lammle, T. (2007). *Cisco Certified Network Associate study Guide, Sixth Edition*, ISBN: 978-0-470-11008-9.

[10]. Udeze Chidiebele. C., Okafor Kennedy. C., Prof. Inyiama, H. C. & Okezie, Dr C. C. (2012). Effective Security Architecture for Virtualized Data Center Networks, (IJACSA) *International Journal of Advanced Computer Science and Applications,* 3, (1), pp 196-200,

[11]. Works, M. (2005). SimEVents user's Guide. The MathWorks, Inc.

[12]. Cisco Systems, "Packet tracer" http://www.cisco.com/academy_2007.

[14]. Google's next OpenFlow challenge. http://www.gigaom.com/.../googles-next-openflow-challenge-taking-sdns-to-the- ...

[15]. Next-Gen Network Drumbeats: Going With the OpenFlow,Available, www.wired.com/cloudline/2012/04/openflow/

[16]. Ankur Nayak, Alex Reimers, Nick Feamster, Russ Clark, "Resonance: Dynamic Access Control for Enterprise Networks" (unpublished).

[17]. Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N. & Ethane, S. S. (2007). *Taking control of the enterprise*. In SIGCOMM '07, [2] M. Casado, T.

[18]. Okafor, K. C. (2010). *SVLAN Approach to Congestion Management in Data Center Networks,* M.Eng Thesis, University of Nigeria, Nsukka.

[19]. Scanning Technology for Automated Registration, Repair and Response Tasks. https://start.gatech.edu/.