Implementation of Virtualization in Software Defined Networking (SDN) for Data Center Networks

Nader F. Mir, Jayashree N. Kotte, and Gokul A. Pokuri
nader.mir@sjsu.edu
Department of Electrical Engineering
San Jose State University
San Jose, CA, 95195

Abstract—**Software Defined Networking (SDN) is an emerging technology in IT industry. There is enormous pressure on network operators to change the conventional network architecture to accommodate the rising need for innovative services and fluctuating demands. The reasons behind the network architecture evolution are heavy traffic, scalability and enormous bandwidth shortage. With the implementation of SDN in a Data Center Network (DCN), providing centralized control can eliminate major issues of routing. In this paper, we develop a user application to scale the DCNs and analyze SDN's performance. We consider networks connecting 8 hosts to 512 hosts and measure performance metrics in terms of latency, packet drop rate and bandwidth. The tools being used are Mininet, vSwitch, OpenFlow controller and VMware workstation. Python language is used to bind the tools together. Customized topologies are created and implemented using OpenFlow controller to determine SDN's efficiency.**

Keywords-SDN; software defined networking; virtualization; data centers.

I. INTRODUCTION

During early 2004, networks had distributed configurations. After 2004, the centralization of network control came into existence with the emergence of Border Gateway Protocol (BGP) that run on Routing Control Platforms (RCP). RCP was generalized for decision planes, which was responsible for computing routes. Also, RCP was generalized for data plane that forwards packets [1]. In 2008, OpenFlow was presented, whose fundamental roots came from RCP. The idea behind OpenFlow was the decoupling of the control and data planes.

Virtualization in networks was another effective trend that allows network administrators to run applications on fewer physical servers [2][3]. No business can afford application downtime and virtualization provided a way to decrease application downtime. High availability and fault tolerance are built right into the platform. Virtualization provides bigger savings by letting a network administrator run many apps on fewer servers at the industry's lowest net virtualization cost.

In general, any routing device contains two planes of operations: control plane and data plane. The task of the data plane is to forward packets to destinations. Routers determine the path for routing packets based on their respective routing tables. The control plane computes these routing tables. The ideas of network virtualization and decoupling data and control planes resulted in the *Software Defined Networking* (SDN).

SDN enables application of virtualization principles on network infrastructure by abstracting network resources, pooling and automating them to outshine the limitations of the network architectures [4]. SDN changes the device-level configuration by centralizing the control plane [5][6].

In SDN, the control lies in the centralized controller, which runs on top of the data plane. This controller provides a complete picture of the network and is responsible for controlling all the routers in the network. This feature of the SDN controller gives network operators network-wide control. The separated control plane offers faster innovation as the orchestration of the network constituents is done on an open interface.

OpenFlow is an open interface standard that enables network operators to control the chips placed in a switch using software. This paved the way for better innovation and easier migration of resources with ambiguous demands from enterprises. OpenFlow brought the open vSwitch into existence [6]. Open vSwitch became the root of system architecture decoupling. It also enabled protocols under the experimenting state to co-exist with the legacy protocols in the network.

To implement SDN in any organization, the working protocol is typically OpenFlow. Communication between the centralized controller and switches is facilitated by the OpenFlow protocol [7][8]. The OpenFlow is a set of commands that the SDN controllers use to pass on the routing decisions to vSwitches for routing table updating.

A *data center* is either a physical or virtual storehouse of data. Its functions include collection, storage, management and propagation of data. A data center contains all the logs of the company varying in the level of significance. Data centers are also very similar to an operations center focused on networks. They hold many automated computers that monitor the Web activity, server access and performance of the networks.

The rest of this paper presents the components of the simulation implementation including the Mininet setup and the simulation architecture. The paper then proceeds with the implementation details, followed by some results of the simulation, including several charts presenting network latency and packet drops.

 ISBN: 978-1-61208-450-3

II. COMPONENTS OF IMPLEMENTATION

A. SDN Implementation

The infrastructure of the implementation for this article is made up of decoupled control and data planes, as mentioned earlier. The control plane is typically a software program. These software programs are usually written in high-level languages such as Python and C. The data plane is programmable hardware, which differs from traditional networks in which the data plane is not programmable.

The routing path computations made by SDN controller affect the routing table in a vSwitch. The computing decisions are propagated to the vSwitches using a set of control commands. Openflow is the set of control commands that are made use of by both data and control planes for communication.

The control planes consist of logic that controls the packet forwarding behavior of the routers and switches. It also contains configuration procedure for middle boxes, such as firewalls, and load balancers.

The data plane is responsible for forwarding packets in the network, so it contains routing tables and hardware pertaining to the network. To summarize the functionality of SDN, it is safe to say that computing the shortest paths are functions of the control plane and data plane functions handle packets and routing them from input port to output port. The separation of control and data planes facilitates the evolution of software and hardware independently. Also, it enables the controller to be programmed by high-level programs. This kind of control enforcement makes it easy to debug the network.

B. DCN Implementation

Any data center typically harbors numerous hosts. Data center networks are broadly classified into three types:

i. Server-routed networking
ii. Switch-routed networking
iii. Server-Routed Networking

A DCN, such as a server-routed networking DCN, is typically a three-tier topology. The first layer is usually an access layer that connects the server racks called top-of-rack (TOR) switches, as shown in Fig. 1. The next layer of devices consists of aggregate switches that connect the access layer and a core layer. Switching in a DCN basically involves routing a packet from a source host to a destination server. In reality, the structure of DCN has the core layer switches connected to the TOR switches, which is in turn connected a server rack. Packets within the DCN network find paths traversing through TOR and core switches but packets originating from outside the DCN are routed to the destination only if core switches provide access. Such packets from outside then find their destination server by traversing the TOR switches and server racks. The core layer switches in a data center network enforce network security and load balancing.

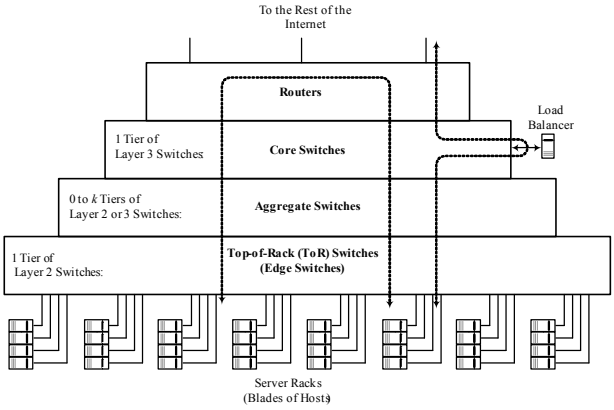


Figure 1. Data Center Architecture

C. Virtualization

Virtualization provides a layer on hardware that can be used to install an instance of an operating system. Normally, a *hypervisor* is installed on the hardware to carry out the functionality of virtualization. Hypervisors are classified into two types. The Type-1 hypervisor is installed directly on the hardware and the instances of different operating systems are installed on it. With the use of management software, it can move the instances of operating systems between physical servers based on the needs. A Type-2 hypervisor is also called hosted hypervisor. Hosted hypervisors are installed on the operating system.

D. MININET

Mininet [8] is a virtual network emulation software that is used to introduce or launch a network consisting of switches, hosts and a SDN controller. Mininet uses OpenFlow switches and routers. An example of Mininet topology used in data centers is the tree architecture, as shown in Figure 2. Network packets are routed through the SDN controller, and an action is taken for that particular packet. There is an extra latency on the first packet because of the communication of the controller. After the first ICMP request and reply, the flows are cached by the switch for a limited time. The following Mininet command creates a network with four hosts (servers) and three switches in a Tree topology: `$sudo mn --topo=tree,2,2`. We use this topology to simulate a data center network.

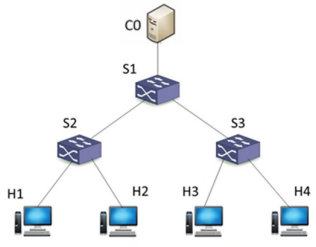


Figure 2. Tree topology, as a simple data center network

 ISBN: 978-1-61208-450-3

E. POX Controller

POX [9] is a strictly reactive cross platform controller built on Python platform with fairly simple source code. The actual modelling of POX is to dynamically respond to links and switches as their status changes to make sure those connections are always maintained. The event handler handles the status change events. It is very useful for research and experimentation. There are numerous ways to run POX and it has many components. POX controller is bundled along with Mininet.

F. OVS Switch

Open vSwitch is a virtual switch, which is used to connect hosts, in this case Virtual machines. It supports many traditional switch features like VLAN tagging, 8012.1q trunking, Spanning tree protocol, port mirroring, monitoring, tunneling (GRE, IPSec) and QoS control. Open vSwitch works in two types of modes.

1. Normal mode
2. Flow mode.

In normal mode, the switch acts like a traditional switch i.e., it acquires information about the network and computes the path. It learns the path by source MAC address and at the beginning it broadcasts and multicasts traffic until it learns all the paths. In flow mode the SDN controller configures the paths.

There are two criteria called match and action. If the condition matches the specification the respective action is executed. The match field can be layer 2, layer 3 or layer 4, therefore it can match the IP addresses, MAC addresses and transport protocols for the match condition. Open vSwitch follows a top to bottom matching approach.

Once the matching is done, the corresponding action is executed. Every flow has a specific priority assigned to it. Packet priorities are checked at the firewall level, if rejected by the switch they are rerouted to the SDN controller, which decides the entry of the packet into the network.

G. Integration of SDN in DCN

The main problem in managing a data center is supplying and migration of resources/services in response to the traffic load. SDN solves this issue by logically controlling the routing tables in the switches. If two VM's are communicating with each other, then the switches are aware of the routing paths to each of the VMs correctly. Apart from this, migration of VM becomes easy as the SDN controller regularly updates the routing table in a switch making use of a centralized database. Integration of SDN with DCN creates improved load balancing of traffic in a DCN, improving bandwidth utilization, and scaling the network with changing demands and applications. In this paper, we aim to address the scalability issue and show the efficiency of SDN in reducing the bandwidth utilization thereby improving the load balancing of traffic in a DCN. Careful study of performance parameters such as bandwidth, packet drop rate, latency performance statistics in terms of graphical representation were done.

Figure 3 shows the operation of software defined networking technology in DCN with separated control and data planes. The control plane contains the SDN controller, which is responsible for smooth flow of operations in the network. The data plane, on the other hand, merely acts as a packet forwarding backplane, which is composed of network components such as open vSwitches. Communication between the data plane and control plane takes place through the OpenFlow protocol [7]. OpenFlow protocol communicates all the routing decisions and routing table entries for routing packets to open vSwitches from SDN controller.

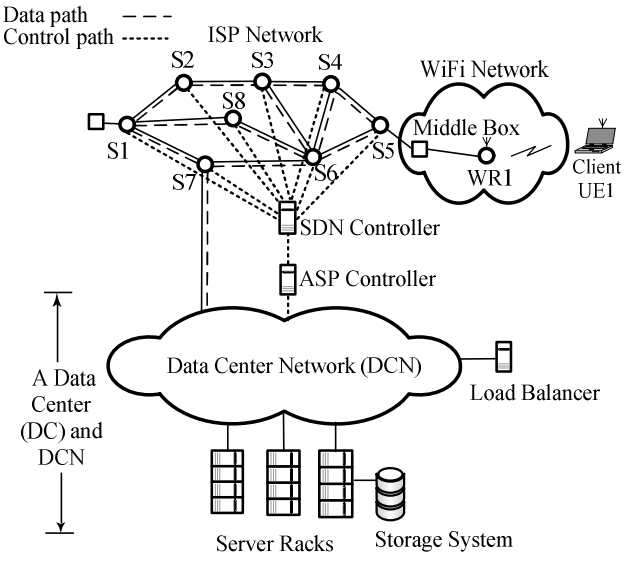


Figure 3. SDN in DCN

H. NFV

Network Function Virtualization (NFV) [4] has emerged as a result of exceeding demand of scaling of applications and services. NFV makes use of SDN to provide services in a wide spread environment.

Previously, network designers encountered a problem of last hop determination for L2 when a packet arrived. This problem was dealt with by creating the vSwitch. vSwitches addressed the problem by creating L2 VLANs. At later stages, with the increase of virtual machines, the need for L3-7 services between VMs arose. This issue gave birth to a horde of virtual versions of hardware products like vRouter, vFirewall, vNIC and virtual load balancers [4], which helped in controlling and managing the traffic in the network. Since a large number of virtual devices were being used, the need for automation/orchestration environment emerged. Due to fluctuating demands, the focus shifted to open environments for orchestration. All this complexity in management needed

 ISBN: 978-1-61208-450-3

a simplification. This was brought forth by SDN controllers such as OpenDaylight [4], which were used to simplify management of the network. To resolve the issue of scaling the server based networking across a range of servers, Network Function Virtualization was proposed.

The vendors have been demanding the service providers to change the architecture as they faced issues of the service being too slow, expensive and not being able to bring up new services. Vendors demanded very flexible software, which could be attained with NFV aided by SDN.

III. IMPLEMENTATION

VMware workstation, such as VMware Fusion or Virtualbox, was installed. Then Mininet, which is a network emulator, was installed and run on the VMware fusion. A complex virtual DCN was programmed using Python scripts in Mininet. A complex tree topology of server-routed networking scheme was used in this paper. Tests were performed on the DCN that allowed us to find the differences of traditional/conventional routing and routing using SDN.

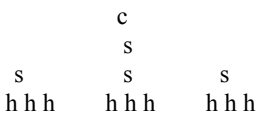
Utilizing Mininet and its Python API, an application script called Tree_1024.py was created to test the scalability of Tree_1024.py. This user application was created to reach the aim of this paper i.e., model how SDN scales in terms of latency and bandwidth as the numbers of hosts on the network increase.

Utilizing the Python API provided by Mininet makes it possible to automate the capturing of the various measurements required to reach the goals described in this paper.

Command: Function Run_mininet runs the following functions in loop for each n number of hosts (i.e., 8, 16, 32, 64, 128, 256, 512) Contains the following Mininet API function calls, TreeNet(depth=i, fanout=2, switch=OVSKernelSwitch),

Command: Treenet(depth, fanout, switch) indicates the number of levels in the network and fan-out indicates the number of leaves per switch in the network.

Eg Treenet(depth = 2, fanout=3, switch=OVSKernelSwitch) would create a network that would look like:



There are two types of switches that are used in Mininet: 1) UserSwitch – this virtual switch is created and used from user space and has no connection to the kernel; and 2) OVSwitchKernel – based on Open vSwitch but running in the Linux kernel. It handles traffic between different virtual machines on the same physical device and network, which the device is connected to.

The main program used in this paper is Tree_1024.py. This program was aimed at comparing the efficiency of SDN with increasing number of hosts. In other words, we addressed the scalability of a SDN run data center network and analyzed it through important performance metrics.

Command: Network.iperf(fmt='g')

To calculate the bandwidth between hosts, the network iperf command is utilized. This command returns the bandwidth between two hosts as a list. Here, iperf is run between the first host and the last host (e.g. in a network with 64 hosts iperf is run between host1 and host64). Using the argument fmt='g' the results are returned in Gbps (Giga bits per second).

Internally the command run as:

Local_host$iperf –p 5001 –t 5 –f g –c dest_host_ip

The iperf is run against the dest_host_ip from the local_host machine.

In the above command

-p 5001: indicates the port on which the iperf command is supposed to run.

-t 5: Indicates that iperf has to run for 5 seconds

-f g: Indicates that the bandwidth values should be returned in Gbps.

-c dest_host_ip: Indicates the destination IP address the iperf should run against.

Proc_ping

The result of the network.pingAllFull() is a Python list of lists. For each ping command run, the following list is created.

pingresultslist = [Node, dest, [sent, rec, rttmin, rttavg, rttmax, rttstd]]

Node – the IP address of the host sending the ping command.

Dest - IP address of the destination of the ping command.

Sent - Number of ICMP ping packets sent.

Rec - Number of ICMP ping packets received.

rttmin – minimum round trip time of all pings for that instance of ping command.

rttavg - Average round trip time of all pings for that instance of ping command.

rttmax – maximum round trip time of all pings for that instance of ping command.

rttstd - standard deviation of all pings for that instance of ping command.

These values are saved in variables and ping result values are stored in a csv file.

Proc_iperf

The result of network.iperf() is a list containing two values of the bandwidth for each path (Local host to remote host and vice versa).The values are cleaned (removes the Gbps in each result), averaged and stored in a variable. The values are stored in a csv file at the end.

Once the run_mininet command is run, the mini_plot function is run to create plots for the Bandwidth and Latency for the different numbers of hosts tested (8, 16, 32, 64, 128, 256, 512). All the rttavg values for each 'n' number of hosts are averaged and saved in a variable. Matplotlib that is python plotting library is used to plot the values.

The above program runs a loop to generate tree architectures by varying the depth. Depending on the depth

 ISBN: 978-1-61208-450-3

value given, it will generate tree architecture of 16, 32, 64, 256, 512 and 1024 hosts. The program also generates authentic graphs in Mininet by comparing the latency and bandwidth with the number of hosts. These graphs show the performance of SDN with increasing number of hosts.

IV. RESULTS AND OBSERVATIONS

The performance metrics that we have generated in Mininet using the automated Python script are latency and bandwidth. The values of latency and bandwidth for hosts starting from 8, 16, 32, 64, 128, 256 and 512 have been calibrated and plotted.

Fig. 4 shows bandwidth utilization in a network with SDN control when the number of hosts increases. Fig. 5 shows the latency in the same network.

In Fig. 6, the comparison of dropped packets in different network sizes using OVS and POX controller is illustrated. With the increase in network size, the number of packets dropped increases exponentially. For the networks that are created, ping time varies with respect to the controllers.

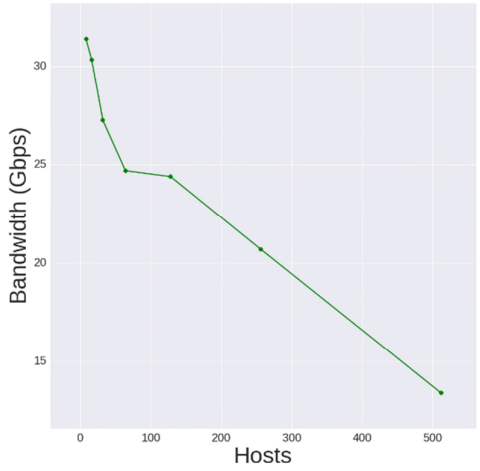


Figure 4. Bandwidth v number of hosts

Fig. 7 shows a comparison of ping time between H1 and H2 for different networks using OVS and POX controller

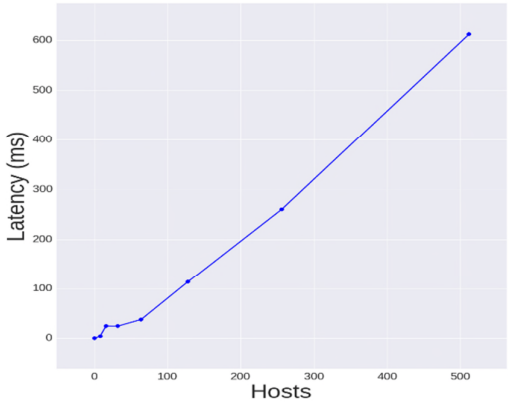


Figure 5. Latency v number of hosts

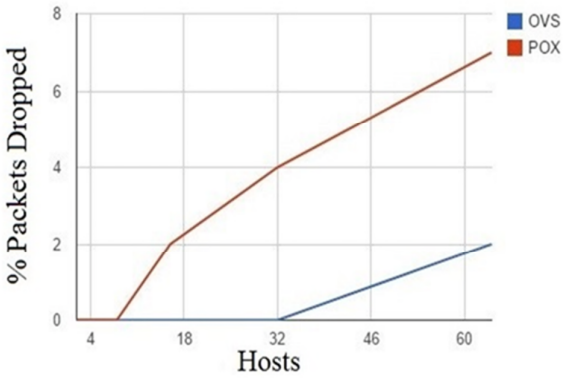


Figure 6. Percentage of packet drop vs hosts

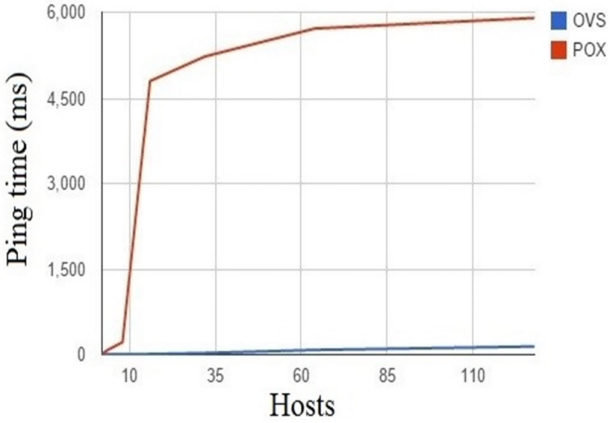


Figure 7. Ping time v/s hosts

Fig. 8 shows a comparison of ping time between a first host and a last host for different networks using OVS and POX controller. The ping results are examined between the first host and the second host of the network. With respect to POX controller design, the effectiveness to connect the source and destination host decreases.

The screenshots shown in Fig. 9 and 10 give the estimation of the ping time analysis between hosts. The first ping time takes more time as the route computation takes place whereas the remaining pings are faster as the routes have already been computed and cached.

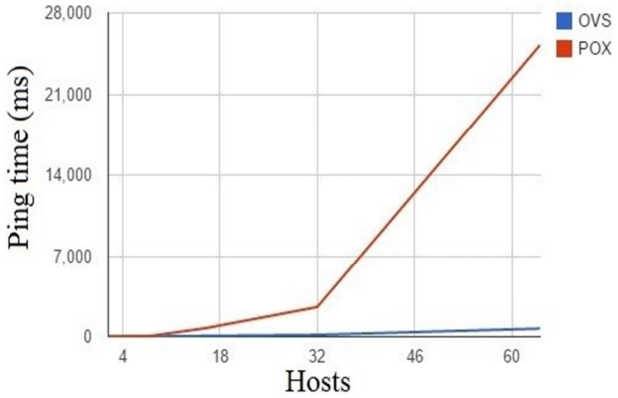


Figure 8. Ping time v/s hosts

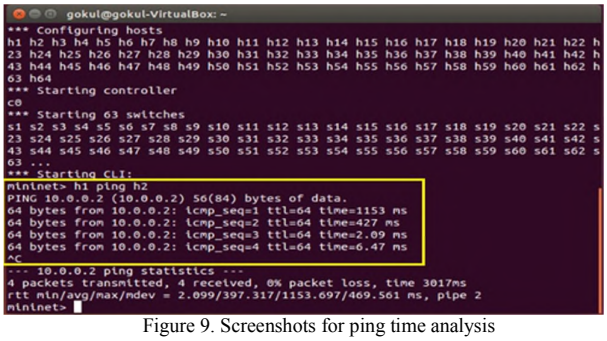


Figure 9. Screenshots for ping time analysis

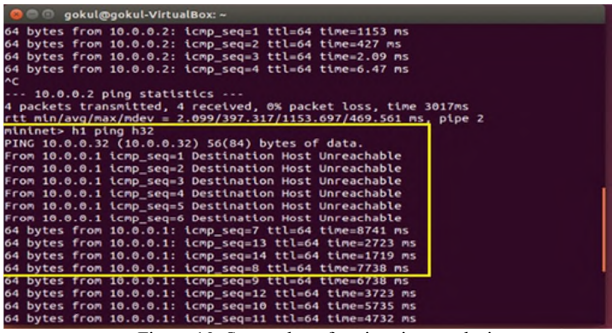


Figure 10. Screenshots for ping time analysis

V. CONCLUSION

We have implemented SDN in a data center network making use of virtualization principles. We have looked at how the network scales from 8 hosts to 512 hosts in terms of latency, packet drop rate and bandwidth. From the results, it can be safely concluded that, as the number of hosts increases, the bandwidth available decreases, and the latency between the host and packet drop rate increases accordingly. These results are in accordance with expected results at the beginning of the paper and show the efficiency of SDN over traditional network infrastructure.

VI. REFERENCES

[1] Udacity.com, 'Computer Networking Basics Training Course Online', 2015. [Online]. Available: https://www.udacity.com/course/computer-networking--ud436. [Accessed: 10- May- 2015].

[2] Vmware.com, 'Virtualization Basics, What is Virtualization: VMware | United States', 2015. [Online]. Available: http://www.vmware.com/virtualization/virtualization-basics/what-is-virtualization. [Accessed: 10- May- 2015].

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, 'OpenFLow: Enabling Innovation in Campus Networks', 2015.

[4] N. Mir, *Computer and communication networks*, 2nd ed. Upper Saddle River, NJ: Pearson Hall, 2015.

[5] T. Nadeau and K. Gray, *SDN*. Sebastopol, CA: O'Reilly Media, 2013.

[6] Sdnhub.org, 'OpenFlow version 1.3 tutorial | SDN Hub', 2015. [Online]. Available: http://sdnhub.org/tutorials/openflow-1-3/. [Accessed: 10- May- 2015].

[7] V. Tiwari, 'SDN and OpenFlow for Beginners', 2013.

[8] Minimiet, http://mininet.org. [Accessed: 10- May- 2015].

[9] POX Controller, https://openflow.stanford.edu/display/ONL/POX [Accessed: 10- May- 2015].

 ISBN: 978-1-61208-450-3