International Journal of Computer Applications (0975 - 8887)
Volume 138 - No.4, March 2016

Campus Network using Software Defined Networking

Dependra Dhakal
SMIT
CSE Department
Rangpo, Majitar, Sikkim

Bishal Pradhan
SMIT
CA Department
Rangpo, Majitar, Sikkim

Sunil Dhimal
SMIT
CSE Department
Rangpo, Majitar, Sikkim

ABSTRACT

Software Defined Networking (SDN) is a paradigm where a software based controller governs the overall network behavior. SDN promotes centralization of network by separating the networks control plane from its data plane. OpenFlow, one of the techniques of SDN technology, is a new approach to networking. This paper discusses application aware routing and traffic engineering in the context of Software Defined Networking (SDN). This paper demonstrates the use of an Open Flow controller to implement application processing logic. With the use of Open Flow switches it is possible to provision the network to treat the packet flows for video, audio and web differently based on user needs and requirements. The idea is to provide a better network efficiency, low bandwidth wastage.

Keywords

SDN, Firewall, OpenFlow

1. INTRODUCTION

SDN is an advent to networking in which the control plane is decoupled from the forwarding plane [7]. Thus, network switches get to be straight- forward sending devices and the control rationale is executed in a consistently incorporated controller disentangling approach authorization and system (re)configuration and advancement. SDN provides various advantages including load balancing, on-interest provisioning, bandwidth management and also, the capacity to scale network resources. The simplified architecture of SDN is shown in figure1.

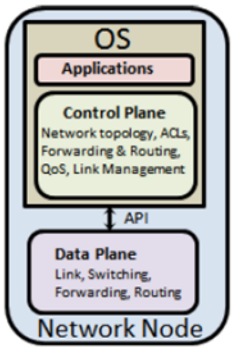

OS
Applications
Control Plane
Network topology, ACLs, Forwarding & Routing, QoS, Link Management
API
Data Plane
Link, Switching, Forwarding, Routing
Network Node

Fig. 1. sdn architecture[20]

The partition of the control plane and the sending plane can be known by method for a programming interface between the switches and the SDN controller. OpenFlow is the communication protocol used as an intermediary between the control and forwarding layer of the SDN construction modeling. OpenFlow allows the direct access of the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor based) [1][8]. OpenFlow is the standard protocols for moving the network control from switches to central controller [4]. POX [2] is a python based open source SDN controller which is used in tour work while Mininet [3] is a network emulator. It executes a collection of end-hosts, routers, switches and links on a Linux kernel. Traditional network architectures are gradually getting ill-suited to meet the requirements of today's enterprises, carriers, and end users .Campus networks are fast and difficult to manage which leads to flexibility and reliability issues. Network changes are liable to extensive provisioning times and setup mistakes. The system devices from few top sellers are firmly coupled and have exclusive CLIs and APIs for arrangement and administration of devices. Software Defined Networking (SDN) acquaints huge perceptibility and adaptability with networking utilizing Open flow controller. But one of the fundamental challenges in SDN is to build robust firewalls for protecting OpenFlow-based networks where network states and traffic are frequently changed. A traditional firewall deployed in a network examines all the incoming and outgoing packets to defend against attacks and unauthorised access [12][13]. However, all the insiders in a network are trusted and not monitored. Internal traffic is not seen and cannot be filtered by the firewall. Therefore any person having access to the network can launch attack on authenticated devices inside the network. Also in an OpenFlow network, network states change dynamically. Along these lines, it needs a firewall in which the tenets of the firewall are additionally redesigned continuously. The main aim of this work is to implement a firewall application by writing code for a SDN controller POX in python. The SDN firewall monitors all the internal traffic in the network and allows selective blocking of host-to-host communication based on network state and time of the day. When a connection between a switch and a controller is up, the application installs flow entries to disable the traffic between each pair of the MAC addresses in the list. The firewall rules can be installed on all switches in the network. and to optimize the available bandwidth in software defined network to meet the requirements of various applications running on top of it. An Open Flow controller is used to implement application processing logic. With the use of Open Flow switches it is possible to provision the network to treat the packet flows for video, audio and web differently based on user needs and requirements. The idea is to provide a better network efficiency, low bandwidth wastage.

2. LITERATURE REVIEW

[1] OpenFlow is an open standard which help researchers to run experimental protocols in the campus environment. Notwithstanding permitting analysts to assess their thoughts in true, the objective is to urge organizing merchants not to uncover the inner working of their products. So, it needs a firewall in which the principles of the firewall are additionally upgraded continuously. The future aspects can allow adding OpenFlow to their products and deploying them in campus network. Furthermore, allowing them to re-use the controllers and experiment on the works build by others. [1][15]Main concepts of software defined networking and how it differs from traditional networking is explained along with a broad range of existing solutions and future directions for the concept and idea of SDN. The architecture of SDN is the introduction of dynamic programmability in forwarding devices through open south- bound interfaces, the decoupling of the control and data plane, and the global view of the network by logical centralization of the network brain. Ongoing research, challenges, threat vectors of SDN architecture and countermeasures for the same in open flow networks is discussed. POX [2] is a Python based open source OpenFlow/Software Defined Networking Controller. POX is used for faster development and prototyping of new network applications. POX controller is pre-installed along with mininet. The POX controller can be passed different parameters according to real or experimental topologies, thus allowing running experiments on real hardware, testbeds or in mininet emulator. POX controller can be used to convert cheap, dumb merchant silicon devices into hub, switch, router or middle boxes such as firewall, load balancer. [3][14]A technique to streamline the data transfer capacity of the system by characterizing certain arrangement of rules in the POX controller. Amid the transmission of information packets the Ethernet switch takes suitable choice by tuning in. The experiment is conducted using Iperf tool and the results show that the utilization of bandwidth increases and the average Round-Trip Time (RTT) time decreases for the POX controller.

3. DESIGN AND IMPLEMENTATION

In Open Flow switch the ingress packet matched against the flow table and sent to the controller if no match is found. The controller chooses what to do with the packet and sends it back to the switch. The switch then executes the activity which the controller characterized. In the event that the packet in the flow table matches the table then the activity is execute as shown in the figure 2.
Two types of scenario are implemented that includes firewall management based on some criteria and bandwidth management which are discussed in the following sub sections.

3.1 SDN Firewall Implementation

It is a firewall application implemented in a software defined network, using mininet and python based on the custom topology as shown in the figure 3.
In figure 3, a small network is set up on the virtual machine, with mininet installed. This network contains a remote controller, six switches; each with a host. This topology is utilized to test different principles and strategies characterized in the firewall design.
Algorithm: firewall.py: The code actualizes a firewall application in a SDN to block particular host-to-host correspondence based on the time of the day. It recognises the ip addresses to be blocked

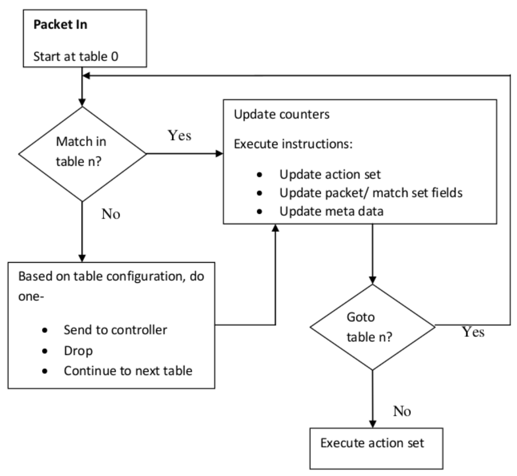


Fig. 2. open flow switch algorithm[9]

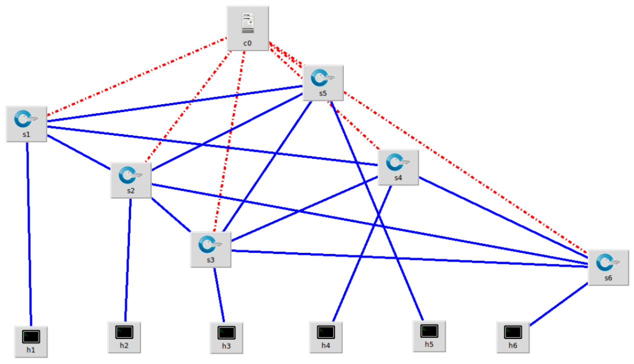


Fig. 3. Custom topology for firewall

from certain other ip addresses and also directs the controller to add specific flow-entries in the flow tables of the switches, so that packets from these blocked ip addresses are independently handled by the switches in future.
Steps:

(1) Get the current time.
(2) Define the firewall policies based on time of the day.
(3) Place the OpenFlow socket in a state in which it is listening for approaching associations.
(4) Define a function "sendRule()" to install flows for allowing and dropping packets between hosts.
(5) Define a function "addRule ()" that allows adding firewall rules into the firewall table.
(6) Define a function "DeleteRule ()" that allows deleting firewall rules from the firewall table.
(7) The function "launch()" launches the firewall module.

Function: sendRule()
Steps:

(1) Begin.

(2) Extricate the IP location of the source attempting to impart by means of the ARP protocol.
(3) Get the destination address.
(4) Check if the source and the destination match any rule in the firewall table.
(5) On the off chance that there is a rule in the firewall table to hinder the communication, then drop the packet else insert a entry in the firewall table.
(6) Perform the appropriate action that is forward/drop the packet.

Function: addRule ()
Steps:

(1) Extract the source and destination address.
(2) Check if the standard for the pair is as of now present in the firewall table.
(3) Call sendRule()
(4) If rule not already present, then add the rule in the firewall table.

Function: DeleteRule()
Steps:

(1) Take the source and destination pair.
(2) Delete the rule from the firewall table if there exists any between the pair.

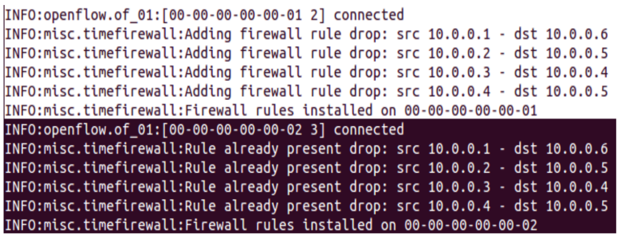
Function: launch()
Steps:

(1) Start the firewall module.
(2) End.

With the POX controller running up, the topology is created using using the following code:
sudo mn ?custom topology.py –topo mytopo –mac –controller=remote, ip=127.0.0.1, port=6633
The firewallpolicies.csv specifies which ip addresses have to be blocked. This file is being called and used by firewall.py code, which is another python code that recognises the ip addresses to be blocked from certain other IP addresses and also directs the controller to add specific flow-entries in the flow tables of the switches, so that packets from these blocked ip addresses are independently handled by the switches in future.
The learning algorithm runs along with the POX controller that forces the switches to behave as normal switches that have learning capabilities. When a new packet reaches a switch, the switch acts according to the OpenFlow protocol, in which it sends the packet to the controller, as the switch is unaware of the action, it needs to perform. The controller informs the switch the required action and hence the switch 'learns' the source address and its corresponding action. The specific flow entries are added in the flow tables of the switches. The following snapshot in figure 4 shows how firewall rules are installed.
After the firewall are installed in node 1, ping test is carried out between node 1 and node 6, the packets are dropped because the rules installed that is shown in the figure 5 below.

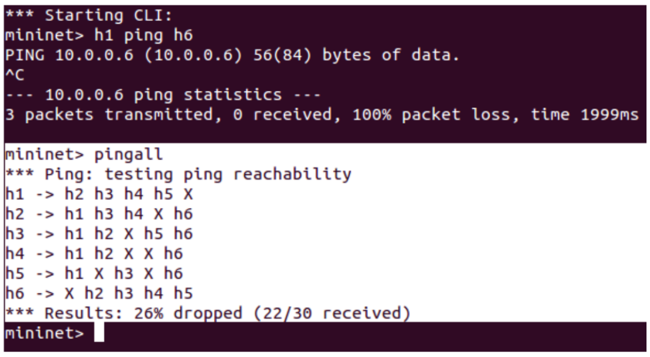
3.2 Bandwidth Management

Similarly Bandwidth Management technique were used based on the custom topology as shown in the figure 6 and is implemented on some bandwidth management policies. The flowchart for the bandwidth management is shown in the figure 7.



```
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:misc.timefirewall:Adding firewall rule drop: src 10.0.0.1 - dst 10.0.0.6
INFO:misc.timefirewall:Adding firewall rule drop: src 10.0.0.2 - dst 10.0.0.5
INFO:misc.timefirewall:Adding firewall rule drop: src 10.0.0.3 - dst 10.0.0.4
INFO:misc.timefirewall:Adding firewall rule drop: src 10.0.0.4 - dst 10.0.0.5
INFO:misc.timefirewall:Firewall rules installed on 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
INFO:misc.timefirewall:Rule already present drop: src 10.0.0.1 - dst 10.0.0.6
INFO:misc.timefirewall:Rule already present drop: src 10.0.0.2 - dst 10.0.0.5
INFO:misc.timefirewall:Rule already present drop: src 10.0.0.3 - dst 10.0.0.4
INFO:misc.timefirewall:Rule already present drop: src 10.0.0.4 - dst 10.0.0.5
INFO:misc.timefirewall:Firewall rules installed on 00-00-00-00-00-02
```

Fig. 4. Firewall rules installed



```
*** Starting CLI:
mininet> h1 ping h6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
^C
--- 10.0.0.6 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 1999ms

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 X
h2 -> h1 h3 h4 X h6
h3 -> h1 h2 X h5 h6
h4 -> h1 h2 X X h6
h5 -> h1 X h3 X h6
h6 -> X h2 h3 h4 h5
*** Results: 26% dropped (22/30 received)
mininet>
```

Fig. 5. Ping statistics

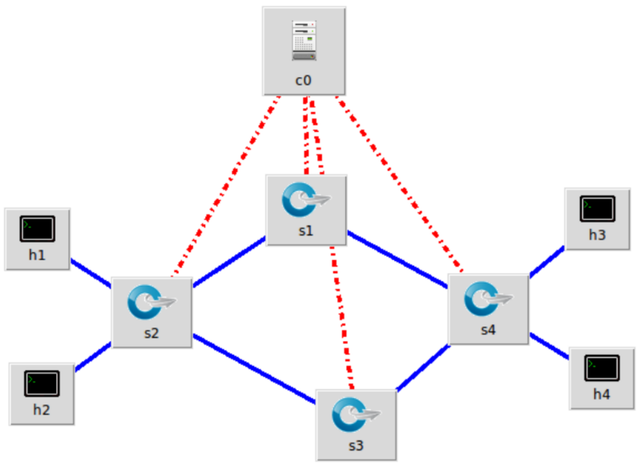


Fig. 6. Custom topology for Bandwidth Management

The POX application programming interface has been used to tailor traffic dynamically based on requirements and access to bandwidth on demand. The path of the packet flow in the circuit is decided in real time based on the application/service requirement. For instance, a VOIP call would require the packets to pass through a low latency path. Another example is of video traffic in which low jitter is more important than low latency. So it will be beneficial if the traffic is engineered to take a path which is efficient for different applications/services running in the network.
The applications have been prioritized for the available bandwidth use. For instance, if a video streaming session of high priority is on

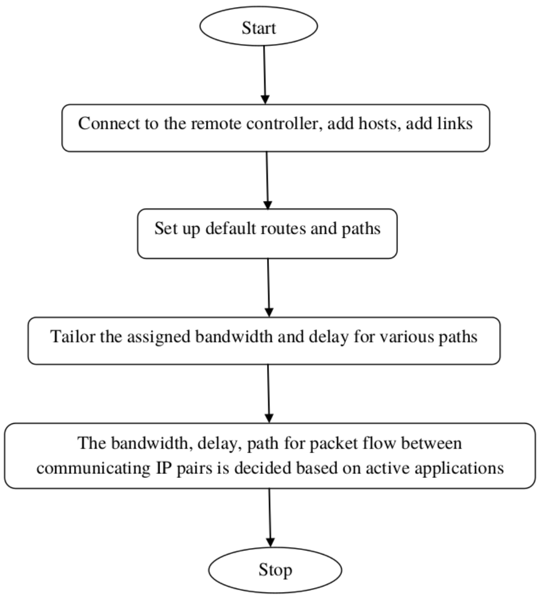
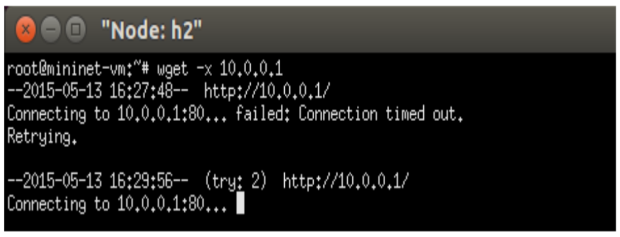

Start
Connect to the remote controller, add hosts, add links
Set up default routes and paths
Tailor the assigned bandwidth and delay for various paths
The bandwidth, delay, path for packet flow between communicating IP pairs is decided based on active applications
Stop

Fig. 7. Bandwidth Management

and only UDP packets are required to flow through the network to achieve low jitter, the TCP communication is blocked entirely.
sudo mn –custom /path/topology.py –topo mytopo –mac –switch ovsk –controller remote
Custom TCP packets are generated using the hping3 command. It was observe that only the ip pairs which match a firewall rule in the configuration file are allowed to pass TCP packets between them. TCP communications between all the other hosts are blocked, while the ICMP and ARP packets can flow. Its analysed by using ping command.
Bandwidth Optimization
Steps:

(1) Start
(2) Connect the remote controller.
(3) Add-hoc hosts.
(4) Add links.
(5) Set up default routes and paths.
(6) Tailor the assigned bandwidth and delay for various paths.
(7) The bandwidth delay for packet flow between communicating ip pairs is decided based on active applications.
(8) End

Firewall
When a switch connects to the controller, the code initializes the connection to the switch as well as adding low-priority flow entries to allow certain types of packets to pass through (i.e. ICMP, IP, ARP), but block all TCP packets that are not specified by a rule in the firewall configuration file. It pushes medium priority flow entries for rules from the configuration file. When it receives a packet, it checks the configuration rules to ensure that there is a match, then pushes flow entries from the packet specifics. If there is not a match, it pushes a flow entry with null action, so the switch will drop packets from that flow.
Steps:
1. Launch the firewall module.
2. The controller reads the configuration rules specified on the command line into memory and listens for a switch to connect on the default port 6633.
3. Once a switch has connected, the controller loads rules onto the switch to allow ICMP and ARP packets to pass through. It also installs rules that instruct the switch to forward packets matching the configuration file rules to the controller. These rules should be of the form:
(ip) [/ (netmask)] (port) (ip) [/ (netmask)] (port)
Where, any of the IP or port fields may be replaced with the term ’any’.
4. The final rule installed to the switch on the initial phase instructs the switch to drop all TCP packets that do not match the previous rules.
5. After all the rules has been sent to the switch, the switch waits for incoming connections that match its rule set and forwards those matching packets to the controller as appropriate.
6. When the controller receives encapsulated packets from the switch, these packets are compared against the rule set. These packets should be allowed flows since they have already been allowed to flow via the switch. Packets are matched with a configuration rule and a flow entry is added to the flow tables to allow TCP traffic between the originating source host and its intended destination host. If the encapsulated packet does not match the rule set, it is dropped. Snapshot for port blocking is shown in fig 8 below.



```
root@mininet-vm:~# wget -x 10.0.0.1
--2015-05-13 16:27:48--  http://10.0.0.1/
Connecting to 10.0.0.1:80... failed: Connection timed out.
Retrying.

--2015-05-13 16:29:56--  (try: 2)  http://10.0.0.1/
Connecting to 10.0.0.1:80...
```

Fig. 8. Connection failed as HTTP ports are blocked

4. RESULTS AND DISCUSSION

The following graphs Figure 9 , Figure 10 and Figure 11 are obtained by initiating a communication between various hosts using Iperf tool. Here, a TCP server runs on host h4 and host h1 and h2 are the clients. The graphs show the comparison based on different parameters such as bandwidth, jitter and Round Trip Time (RTT). As seen in the results, the controller decides the flow priority and path based on the communicating IP pairs. Here the communication between server h4 and client h1 is given priority by the controller when both connections are established simultaneously.The communication between h4 and h1 is routed via higher bandwidth path as the communicating IP pair has the highest priority.

5. CONCLUSION

The practical virtual systems was created, running genuine portion, switch and application code, on a solitary machine utilizing Mininet.Firewall Policies were defined and rules added to the firewall table which changes progressively taking into account necessity and time of the day.Selective obstructing of packets in view of source and destination IP address was carried out. Application processing logic implemented using the pox OpenFlow Controller. Network was tailored to treat the packet flows for video, audio and

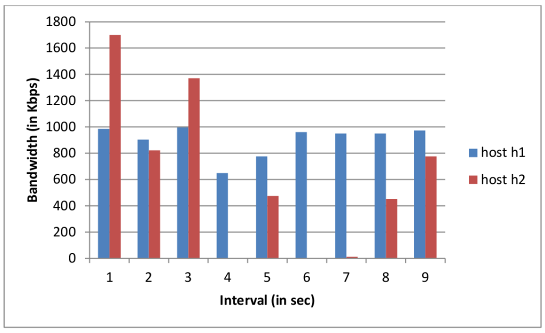


Fig. 9. Bandwidth result of client h1 and h2

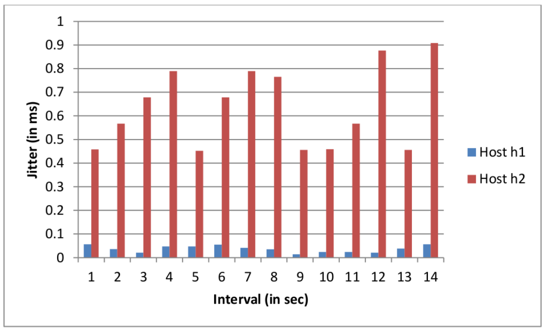


Fig. 10. Jitter result of client h1 and h2

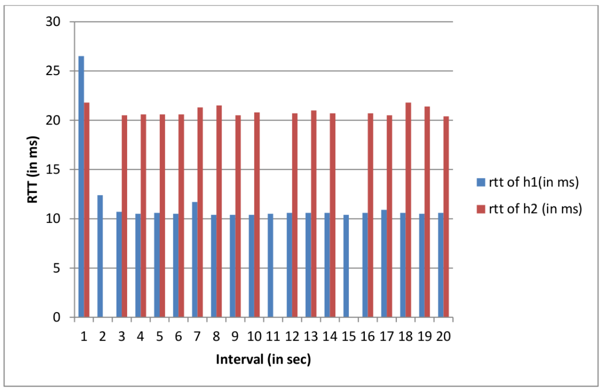


Fig. 11. RTT result of client h1 and h2

web differently based on user needs and requirements.In future, it can be implemented in real time network. The implementation done in the emulated environment can be deployed in campus and home networks but more evaluation based on performance and scalability is required. The functionalities which were deployed in a single controller can be distributed across multiple OpenFlow controllers for better efficiency, link state and controller based load balancing. Real applications can be tested if OpenvSwitches are used. Various applications can be deployed on the network to test the performance of each.

6. REFERENCES

[1] Abhishek Bagewadi, Dr. K N Rama Mohan Babu, ?Towards an Ethernet Learning Switch and Bandwidth Optimization using POX Controller, ?International Journal of Advanced Research in Computer and Communication Engineering, vol. 3, Issue 7, July 2014.

[2] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy (2014)Network Innovation using OpenFlow:A Survey?, IEEE Communications Surveys and Tutorials, VOL. 16, No. 1, pp 493-512.

[3] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, ?A survey of software-defined networking: Past, present, and future of programmable networks,? Communications Surveys Tutorials, IEEE, vol. 16, no. 3, pp. 1617?1634, Third 2014.

[4] Kreutz, D.; Ramos, F.M.V.; Esteves Verissimo, P.; Esteve Rothenberg, C.Azodolmolky, S.; Uhlig, S., "Software-Defined Networking: A Comprehensive Survey," Proceedings of the IEEE, vol.103, no.1, pp.14,76, Jan. 2015.

[5] Kuldeep K. Sharma, Manu Sood ,?Mininet as a Container Based Emulator for Software Defined Networks,? International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, issue 12, December 2014.

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, ?NOX: towards an operating system for networks,? Comp. Comm. Rev., 2008.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, ?OpenFlow: enabling innovation in campus networks,? SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69?74, Mar. 2008.

[8] ONF, ?Open networking foundation,? 2014. [Online]. Available: https://www.opennetworking.org

[9] Open Networking Foundation, "OpenFlow Switch Specification Version 1.3.0," June 25, 2012

[10] OpenDaylight, ?OpenDaylight: A Linux Foundation Collaborative Project,? 2013. [Online]. Available: http://www.opendaylight.org

[11] Saro Velrajan, ? Application Aware Routing in Software defined Networks? , March 12, 2015.

[12] Sean Wilkins, ?A Guide to choosing a next- generation Firewall?, December 2014.

[13] Software-Defined Networking: The New Norm for Networks, Open Networking Foundation, White Paper [Online]. Available: https://www.opennetworking.org.

[14] Sukhveer Kaur, Japinder Singh and Navtej Singh Ghumman-Network Programmability Using POX Controller, International Conference on Communication, Computing and systems (ICCCS-2014), Department of Computer Science and Engineering, SBS State Technical Campus, Ferozepur, India.

[15] Thomas A. Limoncelli, "OpenFlow: A Radical New Idea in Networking," Communications of the ACM, August 2012.

[16] Varun S. Moruse1, Miss. A. A. Manjrekar - ?Software Defined Network Based Firewall Technique?, International Journal of Computer Engineering and Technology Volume 4, Issue 2, pp. 598-606, March ? April (2013).

[17] William Stallings, ?Software-Defined Networks and OpenFlow,? The Internet Protocol Journal, vol. 16, no.1, March 2013.

[18] Retrieved: http://mininet.org/overview/