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.

Fig. 1. sdn architecture[20]
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 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 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.
and only UDP packets are required to flow through the network to achieve low jitter, the TCP communication is blocked entirely. 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
the web differently based on user needs and requirements. In future, it can be implemented in real-time networks. 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


