International Journal of Advances in Applied Sciences (IJAAS)
Vol. 9, No. 1, March 2020, pp. 70~76
ISSN: 2252-8814, DOI: 10.11591/ijaas.v9.i1.pp70-76

Software defined network emulation with OpenFlow protocol

Tsehay Admassu Assegie
Department of Computing Technology, Aksum University, Ethiopia

Article Info

Article history:

Received Nov 9, 2019
Revised Jan 10, 2020
Accepted Feb 3, 2020

Keywords:

Mininet
OpenFlow
POX controller
SDN emulation
Software defined network

ABSTRACT

In software defined network the network infrastructure layer where the entire network devices, like switches and routers reside is connected with the separate controller layer with the help of standard called OpenFlow. The open flow standard enables different vendor devices like juniper, cisco and Huawei switch to connect to the controller or a software program. The software program controls and manages the network devices. Therefore, software defined network architecture makes the network flexible, cost effective and manageable, enables dynamic provisioning of bandwidth, dynamic scale out and dynamic scale in compared to the traditional network. In this study, the architectures and principles of software defined network is explored by emulating the software defined network employing a mininet.

Corresponding Author:

Tsehay Admassu Assegie,
Department of Computing Technology,
College of Engineering and Technology,
Aksum University,
1010 Aksum Universit, Aksum, Ethiopia.
Email: tsehayadmassu2006@gmail.com

1. INTRODUCTION

The number of devices connecting to the network increases day by day but, capability of the routing table is limited, a traditional internet protocol (IP) network become increasingly difficult to manage the devices' in the network and configuration errors have become common problems in addition to the network management problems. The administrator has to go to every device in the network and issue usually a vendor specific commands to set policies, routing information and many network parameters required for the networking device to function and operate smoothly.

The complexity of device management had therefore brought the researchers to divert their attention to a new networking paradigm, the software defined network principle. In the traditional network, the software is bundled with the hardware, this bundled technique is called integrated approach and this is costly and even takes time to converge if any failure occurs in the network. The interfaces and the command used in device management are vendor-specific. The bundling of software with the hardware has made it difficult to manage the device and as a solution to these problems a new approach has emerged which is called software defined network. In this approach, the software which is used in device functionality management is separated from the hardware.

Software defined network is a networking architecture where the control logic and forwarding functionalities are separated into different layers, the controllable program and the physical infrastructure layers. The key principles of software defined network are: Data plane- is the layer responsible for end to end delivery of data in the network, control plane-deals with IP routing and forwarding decisions, programmable network centrally managed- the management functionalities are centralized, open interfaces between the device in the control plane and the data plane.

2. RELATED WORKS

In this section, the research works carried by different researchers related to a software defined network based on the open flow standard, the architectures of software defined networks, the principles of software defined networks and practical implementation issues of software defined network are reviewed [1-25]. Many researches have been carried out by different scholars on the implementation, challenges and future directions but still the software defined network is in its infancy and further researches are required on this emerging virtualized network infrastructure.

A study on emulation of software defined networks indicated that mininet is a powerful software defined network emulation tool [1-10]. Although it is been shown in the study that the mininet is a best tool to emulate software defined networks, the authors used a limited number of hosts in their experiment. This shows that another study to be carried out to test how the emulator behaves in a large-scale network.

The software defined network is an emerging trend in the field of networking. With this new emerging trend, the traditional integrated network is assumed to be decoupled into the control plane where the management and functionalities of devices in the data plane, like forwarding packets, policy making in the network is controlled at this layer. And the routing and data forwarding plane is separated [2-11] form the control plane.

A software defined network is programmable network where the control logic, which is responsible for the configuration and device management in traditional network, is centralized [12-15] into separate plane called control plane. In traditional networks, the administrator or network operators have to move to each device location and configure the devices according to predefined policy of software integrated into the device. In software defined networks all the control logic is separated from the networking devices like switches and routers and is placed in a layered called the control panel.

In software defined network, the open flow is the standard used to forward packets towards their destinations [14-20]. The open flow switches make forwarding decision based on the flow tables. The open flow is the network abstraction layer which defines the standard protocol for communication in the network. It allows the network infrastructure layer to be connected with the control layer. This acts as a bridge between the networking devices and the software defined network controller program. The migration of control logic, which is used to be strongly integrated in the networking devices (for example, Ethernet switches and routers) into accessible and logically centralized controllers, enables the underlying networking infrastructure to be abstracted from the application's point of view. This separation provides a more flexible, programmable, vendor-agnostic, cost effective and innovative network architecture.

Software defined network platform provides an effective solution to cost-effective high-speed network services when compared with the traditional network [15-25]. The pox controller is used to manage the device in network infrastructure and they are connected with the help of standard protocol called the open flow. Devices for different vendors may be used in a software defined network using this open flow standard. The packet flow process in software defined network is demonstrated in Figure 1.

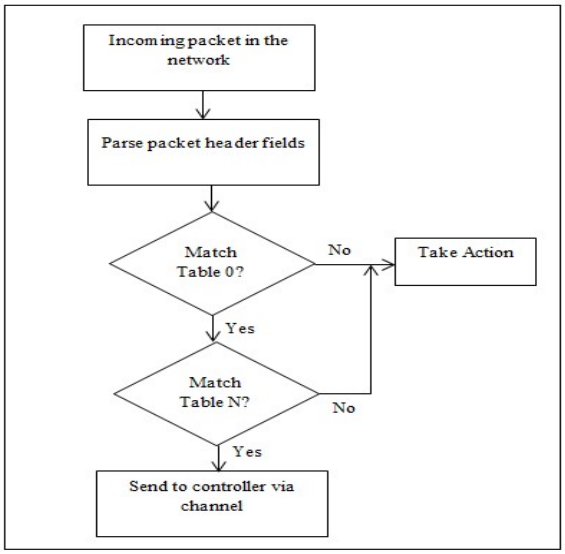

Figure 1. Open flow packet flow chart

3. RESEARCH METHOD

To create a software defined networks, we have used mininet, emulation software defined for modeling and testing software defined network.

3.1. Creating SDN network

To create model of a software defined network, we used a mininet, which is a very powerful LINUX based software defined network emulation tool used by many researchers. It is customizable tool and allows setting certain parameters like bandwidth, delay, packet loss and queue size to different links in OpenV Switches used in software defined network. To model a software defined network in an OpenFlow using mininet emulator we have followed the following steps:

a. Creating software defined networks using command:

tt@ubuntu:~$ sudo mn –topo single,4 –controller remote

The command sudo mn –top=single, 1, 4 –controller=remote creates a software defined network with a controller an OpenFLow switch and four hosts.

b. Listing the nodes available in the software defined network using command nodes:

mininet> nodes

The result of the command shows:

available nodes are:
c0 h1 h2 h3 h4 s1

The hosts h1, h2, h3 and h4 are created and a controller is added into the software defined network.

In Figure 2, all the virtual hosts are connected to the open flow switches, and the open flow switches are connected to the POX controller. The POX controller is a platform implemented in python to emulate the control plane in software defined network. The OpenFlow switches are used to connect the hosts with the POX controller. In this topology the controller is configured with port 6633 and loopback address of 127.0.0.1.

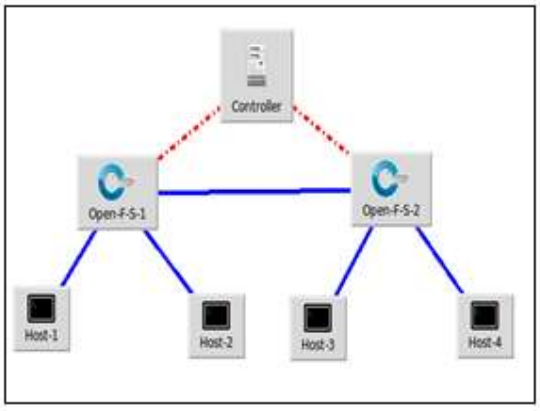

Figure 2. A software defined network, hosts and OpenFlow switches connected in a linear topology

3.2. Managing OpenFlow switch

We have started the POX controller with a Simple_Switch application. The switch application keeps tracks of where the host with MAC address is located and accordingly sends packets towards the destination and does not flood it out through all ports. The SDN switch (for instance, an OpenFlow switch), the SDN controller, and the interfaces present on the controller for communication with forwarding devices,

generally southbound interface (OpenFlow) and network applications interface (northbound interface) are the fundamental building blocks of an SDN deployment. Switches in an SDN are often represented as basic forwarding hardware accessible via an open interface, as the control logic and algorithms are offloaded to a controller. OpenFlow switches come in two varieties: pure (OpenFlow-only) and hybrid (OpenFlow-enabled).

OpenFlow switch is a basic forwarding element, which is accessible via OpenFlow protocol and interface. Although at first glance this setup would appear to simplify the switching hardware, flow-based SDN architectures such as OpenFlow may require additional forwarding table entries, buffer space, and statistical counters that are not very easy to implement in traditional switches with application specific ICs (ASICs). In an OpenFlow network, switches come in two flavors, hybrid (OpenFlow enabled) and pure (OpenFlow only). Hybrid switches support OpenFlow in addition to traditional operation and protocols (L2/L3 switching). An OpenFlow switch consists of a flow table, which performs packet lookup and forwarding. Each flow table in the switch holds a set of flow entries that consists of:

- Header fields or match fields, with information found in packet header, ingress port, and metadata, used to match incoming packets.
- Counters, used to collect statistics for the particular flow, such as number of received packets, number of bytes, and duration of the flow.
- A set of instructions or actions to be applied after a match that dictates how to handle matching packets. For instance, the action might be to forward a packet out to a specified port.

OpenFlow Switch Packet processing logic:

Create MAC table
if (packet into switch)
{ parse packet to reveal src and dst MAC addr
store in dictionary mapping between MAC and portl
lookup dst MAC into port dictionary of switch to find next hop
if(next hop is found) {crate flow mode
send }
else
flood all ports=in_port

Starting mininet with three hosts and one OpenFlow switch s1

```
tt@ubuntu:~$ sudo mn --topo=tree,1,3 --mac --controller=remote --switch
 ovsk,protocols=OpenFlow13
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

Dump flow on switch 1

```
mininet> dpctl dump-flows -O OpenFLow13
*** s1 ----------------------------------
--------
OFPST_FLOW reply (OF1.3) (xid=0x2):
```

4. RESULT AND ANALYSIS

In the experiment, we have used the topology of Mininet given in Figure 2. This topology includes two OpenFlow switch connected to four hosts a one OpenFlow reference controller. Upon execution of Mininet emulation environment with this topology, the OpenFlow controller and switch initiate

the OpenFlow protocol, which can be captured and viewed in the Wireshark capturing window which is shown in Figure 3. On top of this emulated SDN platform, POX is used as the SDN controller. The following screenshot shows the captured traffic, which shows the Hello message, feature request/reply. This confirms that the OpenFlow switch in this setup is connected to the OpenFlow controller. Now we can check the connectivity of each host by a simple ping command: mininet> h1 ping –c 3 h2 and the result of the ping command request is demonstrated in Figure 4 and Figure 3 shows openflow captured packet.

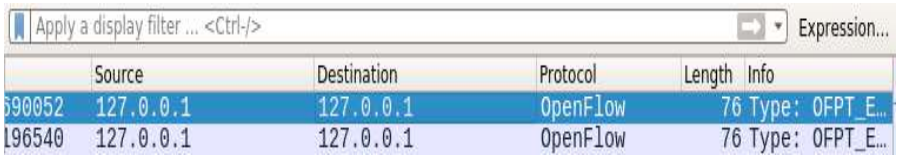

Figure 3. OpenFlow traffic, captured in wireshark

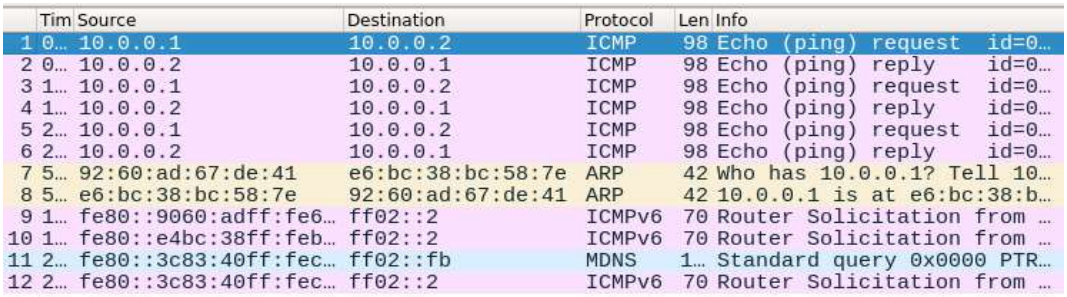

Figure 4. The captured traffic after issuing an h1 ping –c 3 h2 command in mininet

The ping sends three ping request packets as shown in Figure 6. A flow entry covering ICMP ping traffic was previously installed in the switch, so no control traffic was generated, and the packets immediately pass through the switch. An easier way to run this test is to use the Mininet CLI built-in pingall command, which does an all-pairs ping. Another useful test is a self-contained regression test. The following command created a minimal topology, started up the OpenFlow reference controller, runs an all-pairs-ping test, and tore down both the topology and the controller. The data rate of each emulated Ethernet link in Mininet is enforced by Linux Traffic Control (tc), which has a number of packet schedulers to shape traffic to a configured rate. Mininet allows you to set link parameters, and these can even be set automatically from the command line:

sudo mn –link tc, bw=10, delay=10ms

Check the traffic bandwidth using command and verify result:

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.50 Mbits/sec', '11.9 Mbits/sec']
```

This will set the bandwidth of the links to 10 Mbps and a delay of 10 ms. Given this delay value, the round trip time (RTT) should be about 40 ms, since the ICMP request traverses two links (one to the switch, one to the destination) and the ICMP reply traverses two links coming back. The round-trip time for packet is illustrated in Table 1. As shown in Table 1, the round trip time is higher in linear topology compared to the tee and hybrid topology. In Figure 5 the round trip time of each captured packet is shown and Figure 6 shows that, the tree topology in OpenFlow based software defined network performs better than the linear and hybrid topologies. The round-trip time for tree topology is much lower than all types of topologies used in OpenFlow based software defined networks. The hybrid approach performed moderately compared to the tree and linear topology whereas the linear topology is the lower in performance than the tree and hybrid topologies.

```
mininet> h1 ping -c 10 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=46.4 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=47.8 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=42.9 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=41.8 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=42.8 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=42.8 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=42.1 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=42.5 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=42.1 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=42.9 ms
```

Figure 5. Round-trip delays between nodes for basic OpenFlow topologies

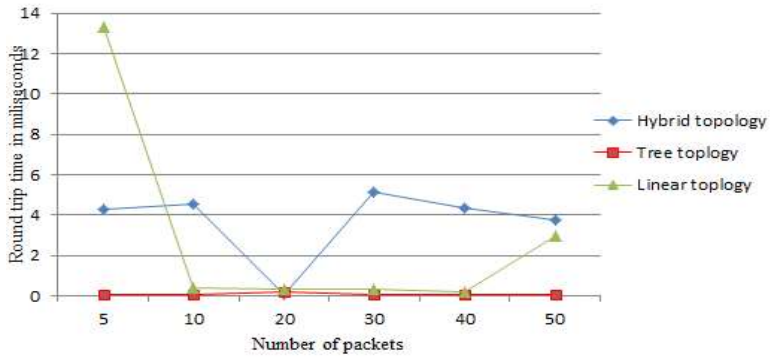

Figure 6. Round trip time between nodes, hybrid, linear and tree OpenFlow topologies

Table 1. Round-trip delay between nodes for basic OpenFlow topologies

# of packets	Hybrid topology (Round Trip Time in milliseconds)		Tree topology (Round Trip Time in milliseconds)		Linear topology (Round Trip Time in milliseconds)	
	Maximum	minimum	maximum	minimum	maximum	minimum
5	4.3	0.088	13.3	0.09	30.3	0.085
10	4.53	0.052	0.375	0.079	0.315	0.087
20	0.059	0.206	0.317	0.063	0.287	0.048
30	5.15	0.087	0.305	0.037	0.27	0.046
40	4.36	0.067	0.188	0.056	0.309	0.053
50	3.78	0.047	2.95	0.079	0.251	0.048

5. CONCLUSION

Software defined network (SDN), which is often denoted as a revolutionary emergining idea in computer networking, promises to dramatically simplify network control, management, and enable innovation through network programmability. Mininet facilitates the creation and manipulation of software defined network components. The mininet is helpful to explore OpenFlow, which is an open interface for controlling the network elements through their forwarding tables. A network element can be converted into a switch or a router via low level primitives defined in the OpenFlow.

In this study, we have emulated a software defined network using mininet and POX, a software platform developed by Python which we have used as a controller with OpenFlow switch and we have discussed the characteristics of software defined network. By issuing commands in the mininet environment we have showed that the network infrastructure can be virtualized for example Ethernet link bandwidth can be dynamically provisioned from a controller, like POX programmatically. Finally, the pefromance of three topologies namely, hybrid, tree and linear toplogy is evaluated in the emulated test bed environment and results shows that the hybrid toplogy is better in performance compared to the linear and tree topologies.

REFERENCES

[1] Faris Keti and Shavan Askar. "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments," *6th International Conference on Intelligent Systems, Modeling and Simulation*, 2015.

[2] Danda B. Rawat, "Software Defined Networking Architecture, Security and Energy Efficiency: A Survey," *IEEE Communication Surveys & Tutorials*, Vol. 19, No. 1, 2017.

[3] Diego Kreutz, "Software-Defined Networking: A Comprehensive Survey," *IEEE*, Vol. 103, No. 1, 2014.
[4] Xuan-Nam Nguyen, Damien Saucez, Chadi Barakat, and Thierry Turletti, "Rules Placement Problem in OpenFlow Networks: A Survey," *IEEE Communication Survey and turotials,* 2015.
[5] Daniel King, Charalampos Rotsos, Alejandro Aguado, Nektarios Georgalas, and Victor Lopez, "The Software Defined Transport Network: Fundamentals, Findings and Futures," *IEEE*, 2016.
[6] Sakir Sezer, Sandra Scott-Hayward, and Pushpinder Chouhan, "Implementation Challenges for Software-Defined Networks," *IEEE Communications Magazine*, July 2013.
[7] Aashish Dugar and M Madiajagan, "Study on SDN using mininet," *Indian Journal of Computer Science and Engineering*, 2016.
[8] Rajiv Ranjan, Sumit Thakur, and Pankaj Rai, "Implementing the concept of software defined network in cloud computing," *International Journal of Computer Engineering and Applications*, Vol. 11, No. 9, 2016.
[9] Akhilesh Thyagaturu, Anu Mercian, Michael P. McGarry, Martin Reisslein, and Wolfgang Kellerer, "Software Defined Optical Networks (SDONs): A Comprehensive Survey," *IEEE*, 2016.
[10] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (SDN): A survey," *Secur. Commun. Netw.*, Vol. 9, No. 18, 2017.
[11] Yogita Hande and M. Akkalakshmi, "A Study on Software Defined Networking," *International Journal of Innovative Research in Computer and Communication Engineering*, Vol. 3, No. 11, 2015.
[12] Wolfgang Braun and Michael Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices," *Future Internet*, Vol. 6, No.2, 2014.
[13] Zahra Al-Abri, Ahmed Al Maashri, Dawood Al-Abri, and Fahad Bait Shiginah, "Using SDN as a Technology Enabler for Distance Learning Applications," *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, Vol. 6, No. 2, pp. 225-234, 2018.
[14] Manar Jammala, Taranpreet Singha, Abdallah Shamia, RasoolAsalb, and Yiming Li, "Software-Defined Networking: State of the Art and Research Challenges," *Journal of Computer Networks*, 2014.
[15] R. Sahay, W. Meng, and Christian D. Jensen, "The application of Software Defined Networking on securing computer networks: A survey," *Journal of Network and Computer Applications*, Vol. 131, pp. 89–108, 2019.
[16] Sandhya, Yash Sinha, and K. Haribabu, "A survey: Hybrid SDN," *Journal of Network and Computer Applications*, Vol. 100, pp. 35–55, 2017.
[17] Murat Karakus and Arjan Durresi, "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)," *Computer Networks*, Vol. 112, pp. 279–293, 2017.
[18] Tianzhu Zhang, Paolo Giaccone, Andrea Bianco, and Samuele De Domenico, "The role of the inter-controller consensus in the placement of distributed SDN controllers," *Computer Communications*, Vol. 113, pp. 1–13, 2017.
[19] Sibylle Schallera and Dave Hoodb, "Software defined networking architecture standardization," *Computer Standards & Interfaces*, Vol. 54, pp. 197–202, 2017.
[20] Franciscus Wibowo, Mark A. Gregory, Khandakar Ahmed, and Karina M. Gomez, "Multi-domain Software Defined Networking: Research status and challenges," *Computer Standards & Interfaces*, Vol. 54, pp. 197–202, 2017.
[21] D. Singh, Bryan Ng, Yuan-Cheng Lai, Ying-Dar Lin, and Winston Seah, "Modeling Software-Defined Networking: Software and hardware switches," *Journal of Network and Computer Applications*, Vol. 122, pp. 24–36, 2018.
[22] Celyn Birkinshaw, Elpida Rouka, and Vassilios G. Vassilakis, "Implementing intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks," *Journal of Network and Computer Applications*, March 2019.
[23] Saleh Asadollahi, Bhargavi Goswami, and Atul M Gonsai, "Software Defined Network, Controller Comparison," *International Journal of Innovative Research in Computer and Communication Engineering*, Vol. 5, Special Issue 2, April 2017.
[24] Idris Zoher Bholebawa and Upena D. Dalal, "Design and Performance Analysis of OpenFlow-Enabled Network Topologies Using Mininet," *International Journal of Computer and Communication Engineering*, 2016.
[25] Tsehay Admassu Assegie and Pramod Sekharan Nair, "The performance of Gauss Markov's mobility model in emulated software defined wireless mesh network," *Indonesian Journal of Electrical Engineering and Computer Science*, Vol. 18, No. 1, April 2020.