

# Performance Analysis of a Software Defined Network Using Mininet

Chaitra N. Shivayogimath and N.V. Uma Reddy

**Abstract** Network is growing day by day. New devices are getting added into the network making it very difficult for an IT administrator to configure the ACLs and the other network parameters in the devices. Flexibility and programmability are the key factors in the present day scenario. Software Defined Networks (SDN) is the evolving network technology which provides the two factors mentioned. The latency in the packet delivery is less compared to the legacy Hardware Defined Networks (HDN) and in turn the throughput is also high. The work done in this paper provides a Proof of Concept (POC) for the better throughput of SDN-based routing.

**Keywords** SDN · Mininet · Throughput · Latency · HDN

## 1 Introduction

An evolving networking technology coming into prominence is “Software Defined Networking” (SDN) [1]. SDN separates the control plane and data plane in networking devices such as routers and switches with the help of an API. All the routing decisions are done by a centralized device called controller and the forwarding done by a dumb device, such as a switch.

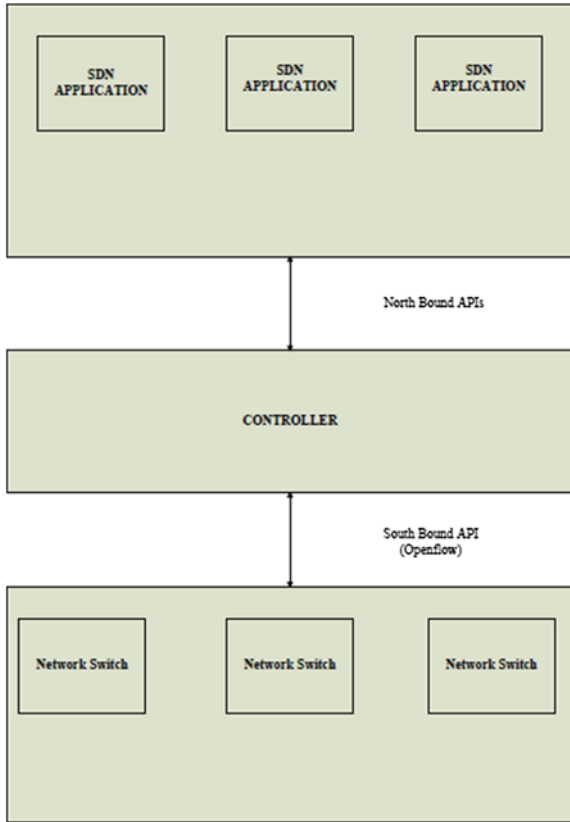
A controller talks to the underlying network device, i.e. the forwarding switch via the “Southbound API”, this is a protocol which talks from switch to controller and vice versa. One such popular protocol is, OpenFlow protocol by the Open Network Foundation (ONF) [2]. This is a flow-based protocol. The routing

---

C.N. Shivayogimath (✉) · N.V. Uma Reddy  
Department of E&C, AMC Engineering College, Bangalore, India  
e-mail: chaishivyogi@gmail.com

N.V. Uma Reddy  
e-mail: nvumareddy@gmail.com

**Fig. 1** SDN architecture



decisions from the controller are inserted into the forwarding switch, which is now called as OpenFlow switch as flow rules into the flow table of the switch. The applications are the “Northbound APIs” which sit on the controller for various applications as shown in Fig. 1. An analogy to a computer would better explain what SDN is. Similar to the way we install programmes on the OS of our computer (e.g. Browsers, text editors, softwares IDEs, etc.), APIs are written to the NOS (Network Operating System) of the controller.

There is a huge need for flexibility and programmability of network in the present day world. Due to virtualization, a single machine can be partitioned into many servers. A virtual instance of a server can be modified, cloned and can be used as another virtual server by starting it as another new instance. So it is easy for migration of server from one machine to another. But it creates a problem in the legacy Hardware Defined Networks (HDN). A major problem will be caused with VLANs whenever a VM moves, VLAN has to be reconfigured. In general terms, to match the flexibility of server virtualization, the network manager needs to be able to dynamically add, drop and change network resources and profiles. This process is difficult to do with legacy network switches, in which the control logic for each

switch is co-located with the switching logic. Another need is, rapidly increasing mobile devices such as smartphones, PDAs, tablets, notebooks accessing the network. Network managers must be able to respond to the changing QoS and security requirements [3].

## 2 Analysis of Latency of Packets in SDN

The traditional way of measuring a network's performance is the packet latency and the throughput. The later is dependent on the foster. An SDN-based network has less latency per packet and hence an increased throughput compared to the legacy HDN. Since hardware testbeds are time-consuming and costlier, simulators are used. The emulation for SDN is done using Mininet [4], an SDN emulator and for HDN, the emulator used is GNS3.

### 2.1 Hardware Defined Network

A network emulated in GNS3 is shown in Fig. 1.

GNS3 is a network emulating software used to design networks in lab environments for testing and study purpose [5]. A sample network as shown in Fig. 2 is created. Once the network is up and running, the latency of the packets can be determined by conducting a ping test between the hosts. Figure 3 shows the time taken for a ping packet to reach from host H1 to Host H3.

Figure 4 shows the ping test conducted from H3 to H1. On observing the time taken by a packet, it can be analysed that the latency is consistently more and same for every packet.

#### Analysis

- On observing Figs. 3 and 4, the latency for every packet from H3 to H1 is either same or more.
- When the first packet from the source arrives at switch, the switch registers in its CAM table (or MAC table) the MAC and IP address of the source host corresponding to the port at which the packet arrives.
- The switch checks its CAM table for the destination address in the CAM table. Since this is the first packet and there is no entry for the address, the switch forwards the packet to the router for the routing decisions.
- The router makes the routing decisions and sends the routing decision entry to switch. The switch forwards the packet accordingly and flushes the entry.
- This process repeats for every packet that arrives consecutively to the switch leading to high latency of every packet.

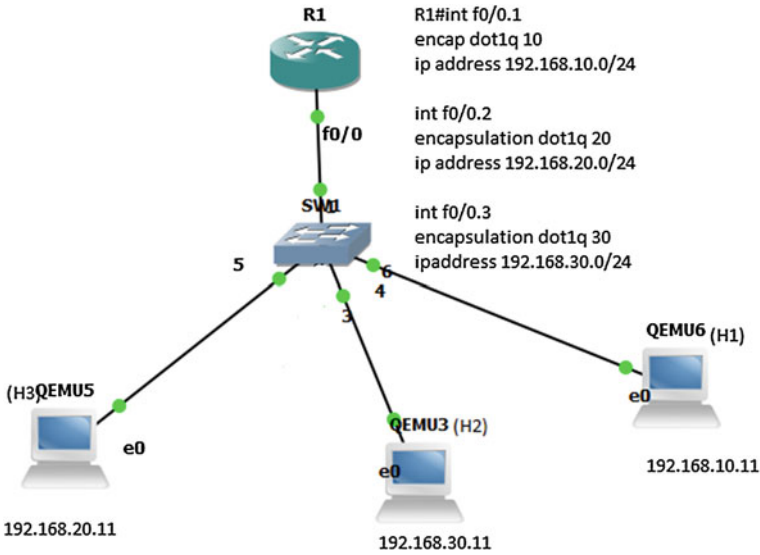
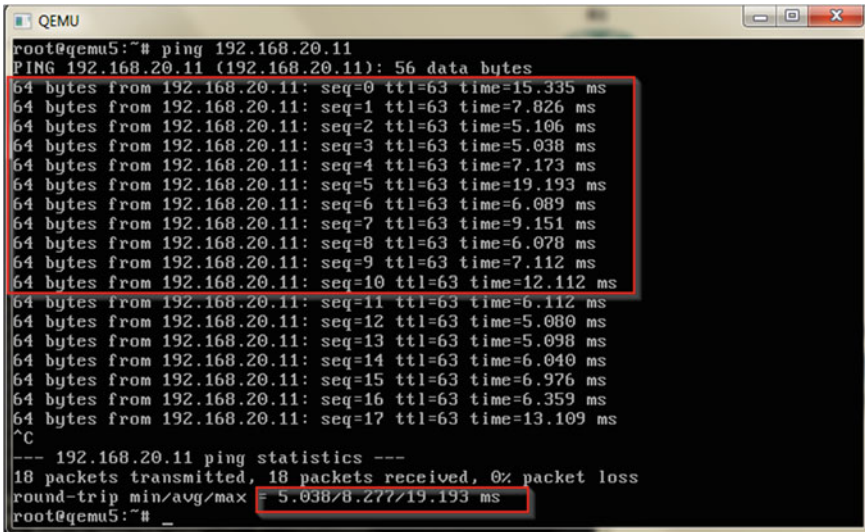


Fig. 2 A legacy hardware defined network emulated in GNS3

```

QEMU
22 packets transmitted, 21 packets received, 4% packet loss
round-trip min/avg/max = 5.087/8.974/19.094 ms
root@qemu6:~# ping 192.168.10.11
PING 192.168.10.11 (192.168.10.11): 56 data bytes
64 bytes from 192.168.10.11: seq=0 ttl=63 time=10.408 ms
64 bytes from 192.168.10.11: seq=1 ttl=63 time=8.098 ms
64 bytes from 192.168.10.11: seq=2 ttl=63 time=8.264 ms
64 bytes from 192.168.10.11: seq=3 ttl=63 time=7.043 ms
64 bytes from 192.168.10.11: seq=4 ttl=63 time=8.108 ms
64 bytes from 192.168.10.11: seq=5 ttl=63 time=5.122 ms
64 bytes from 192.168.10.11: seq=6 ttl=63 time=7.989 ms
64 bytes from 192.168.10.11: seq=7 ttl=63 time=6.944 ms
64 bytes from 192.168.10.11: seq=8 ttl=63 time=6.156 ms
64 bytes from 192.168.10.11: seq=9 ttl=63 time=5.004 ms
64 bytes from 192.168.10.11: seq=10 ttl=63 time=8.117 ms
64 bytes from 192.168.10.11: seq=11 ttl=63 time=5.052 ms
64 bytes from 192.168.10.11: seq=12 ttl=63 time=6.058 ms
64 bytes from 192.168.10.11: seq=13 ttl=63 time=4.949 ms
64 bytes from 192.168.10.11: seq=14 ttl=63 time=8.054 ms
64 bytes from 192.168.10.11: seq=15 ttl=63 time=8.151 ms
^C
--- 192.168.10.11 ping statistics ---
16 packets transmitted, 16 packets received, 0% packet loss
round-trip min/avg/max = 4.949/7.094/10.408 ms
root@qemu6:~#
    
```

Fig. 3 Ping test conducted from H1 to H3



```
root@qemu5:~# ping 192.168.20.11
PING 192.168.20.11 (192.168.20.11): 56 data bytes
64 bytes from 192.168.20.11: seq=0 ttl=63 time=15.335 ms
64 bytes from 192.168.20.11: seq=1 ttl=63 time=7.826 ms
64 bytes from 192.168.20.11: seq=2 ttl=63 time=5.106 ms
64 bytes from 192.168.20.11: seq=3 ttl=63 time=5.038 ms
64 bytes from 192.168.20.11: seq=4 ttl=63 time=7.173 ms
64 bytes from 192.168.20.11: seq=5 ttl=63 time=19.193 ms
64 bytes from 192.168.20.11: seq=6 ttl=63 time=6.089 ms
64 bytes from 192.168.20.11: seq=7 ttl=63 time=9.151 ms
64 bytes from 192.168.20.11: seq=8 ttl=63 time=6.078 ms
64 bytes from 192.168.20.11: seq=9 ttl=63 time=7.112 ms
64 bytes from 192.168.20.11: seq=10 ttl=63 time=12.112 ms
64 bytes from 192.168.20.11: seq=11 ttl=63 time=6.112 ms
64 bytes from 192.168.20.11: seq=12 ttl=63 time=5.080 ms
64 bytes from 192.168.20.11: seq=13 ttl=63 time=5.098 ms
64 bytes from 192.168.20.11: seq=14 ttl=63 time=6.040 ms
64 bytes from 192.168.20.11: seq=15 ttl=63 time=6.976 ms
64 bytes from 192.168.20.11: seq=16 ttl=63 time=6.359 ms
64 bytes from 192.168.20.11: seq=17 ttl=63 time=13.109 ms
^C
--- 192.168.20.11 ping statistics ---
18 packets transmitted, 18 packets received, 0% packet loss
round-trip min/avg/max = 5.038/8.277/19.193 ms
root@qemu5:~#
```

Fig. 4 Ping test conducted from H3 to H1

## 2.2 Software Defined Network

Mininet is an emulator for rapid prototyping of SDN with limited resource. Mininet creates virtual networks as shown in Fig. 5. A Controller (Control Plane), for making routing decisions, a switch for forwarding the packets based on the controller's routing decisions. The switch buffers the flow entry inserted by the controller into the flow table of the switch. This eliminates the necessity of contacting the controller for routing decision for every packet, thus reducing the latency of the consecutive packets after the first packet.

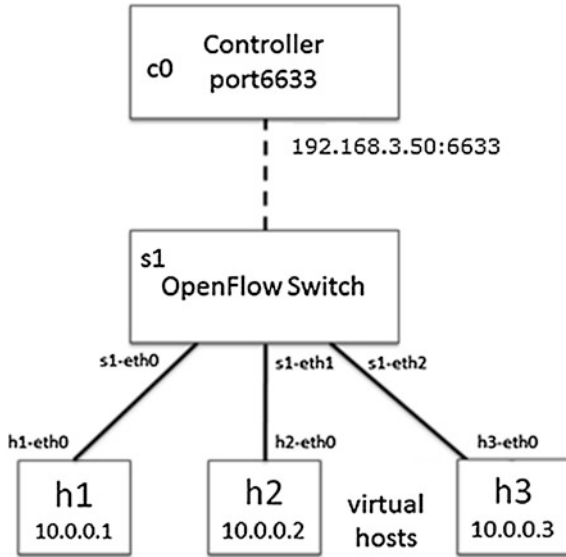
The topology shown in Fig. 5 is created in the Mininet network emulator by issuing the following command:

```
sudo mn --topo=single,3 --mac --switch=ovsk--  
controller=remote,ip=192.168.3.50,port=6633
```

The controller chosen is POX controller [6]. Once the network is created, the controller has to be instantiated. This controller is a remote controller on the IP address 192.168.3.50 over port 6633. Figure 6 shows the information of the OpenFlow switch connected to the controller.

The controller is up and now connected to the OpenFlow switch, which is on IP address 192.168.3.32. The virtual hosts H1, H2 and H3 that are created are connected to the OpenFlow switch via virtual Ethernet links. Conducting a ping all the tests determine the connectivity of all the hosts to the network. To determine the latency of packets in SDN, a ping test from H1 to H2 is conducted as shown in Fig. 7.

**Fig. 5** SDN single topology with three hosts, simulated in Mininet



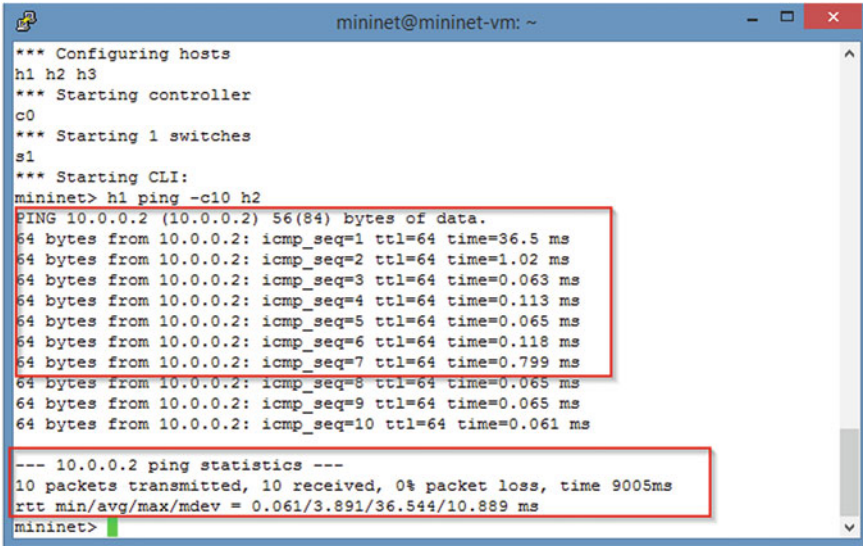
**Analysis**

- On observing from Fig. 7, the first packet takes 21.8 ms, i.e. more time compared to the consecutive packets. All the consecutive packets take very less time compared to the first packet.
- The reason for first packet to take longer is, the routing decision happens only for first packet. Once the controller inserts the flow rule for the first packet, the switch buffers the flow rule in its flow table for 30 s.

```

    Command Prompt - python pox.py log.level --DEBUG forwarding.l2_learning
    C:\Users\Chaitu>cd ..
    C:\Users>cd ..
    C:\>cd Python27
    C:\Python27>cd pox
    C:\Python27\pox>python pox.py log.level --DEBUG forwarding.l2_learning
    POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
    DEBUG:core:POX 0.2.0 (carp) going up...
    DEBUG:core:Running on CPython (2.7.7) Dec 10 2014 12:28:03
    DEBUG:core:Platform is Windows-8-6.2.9200
    INFO:core:POX 0.2.0 (carp) is up.
    DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
    
```

**Fig. 6** POX controller on the IP address 192.168.3.50



```

mininet@mininet-vm: ~
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1
*** Starting CLI:
mininet> h1 ping -c10 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=36.5 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=1.02 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.063 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.113 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.799 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.065 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.061 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9005ms
rtt min/avg/max/mdev = 0.061/3.891/36.544/10.889 ms
mininet>
  
```

Fig. 7 Ping test from H1 to H2

- The consecutive packets are forwarded by the switch without contacting the controller for the routing decision.
- After 30 s, the buffer is timed out and the flow table is cleared. Again the same procedure repeats.

Table 1 Comparison of the packet flow in HDN and SDN

Packet no.	HDN	SDN
1	Packet originating from H1 arrives to switch port	Packet originating from H1 arrives to switch port
2	The switch sends the packet to the router for routing decision	Switch checks for a flow rule corresponding to the packet in its flow table. If there is no entry, the packet is forwarded to controller
3	The routing decision from router is received and the switch just forwards the packet correspondingly	The flow rule is inserted in the openflow switch by the controller. The switch buffers this entry for further communication
4	The consecutive packet is again sent to the router for routing decision	All the consecutive packets are forwarded based on the flow table entry until the buffer time expires
5	Packet sent to router again	
6	Packet sent to router again	
:	:	:
<i>n</i>	Packet sent to router again	The buffer time is out and the packet will be sent to controller

### 3 Conclusion

The work in this paper proves that the latency of the packets in SDN is very much less than that for the legacy networks. Thus the throughput is high comparatively. Many other performance parameters like CPU usage and bandwidth can also be measured using command like *iperf* in Mininet. This command gives the bandwidth usage of the link.

Thus SDN is the future of the networking world with better performance than the legacy HDNs.

Table 1 summarizes the HDN and SDN packet flow.

**Acknowledgments** The authors would like to thank Mr. Santhosh Sundarasamy, Sr. Architect and Engineering Manager, Cloud Managed Security, Paladion Networks Pvt. Ltd. for his complete support and help in carrying out the experiment at Paladion Networks Pvt. Ltd. as part of the Internship Programme of the first author.

### References

1. Lantz B, Heller B, McKeown N. A network in a laptop: rapid prototyping for software-defined networks. In: 9th ACM workshop on hot topics in networks, Monterey, CA, 20–21 October 2010.
2. Openflow White paper by Open Network Foundation, 13 April 2012. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>.
3. Stallings W. Software defined networks and openflow. The Internet Protocol Journal 2013. <http://williamstallings.com/Papers/>.
4. Mininet VM image. <https://github.com/mininet/mininet/wiki/Mininet-VM-Images>.
5. GNS3, Network emulator. <http://www.gns3.com/>.
6. POX controller guide. <https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-InvokingPOX>.