International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 6 Issue 2 March 2016
International Manuscript ID : 2249054XV6I2032016-22

EFFECTIVE IMPLEMENTATION FOR DELIVERY OF REAL TIME SERVICES USING SOFTWARE DEFINED NETWORKING

Ranjna Kumari
Research Scholar
Department of Computer Science
Himachal Pradesh University
Shimla, Himachal Pradesh, India

Prof. Manu Sood
Research Supervisor
Department of Computer Science
Himachal Pradesh University
Shimla, Himachal Pradesh, India

Abstract

Virtualization is one of the key components during the performance evaluation of network enabled environment including distributed computing, cloud computing, grid computing or pervasive computing. In the classical aspects, it is difficult to perform the implementation at physical devices all the time in the research and development process. The network administrators and forensic teams are working on software defined networking (SDN) using which the network components can be controlled and managed using virtual infrastructure and suites. On the physical implementation viewpoint, the single error or oversight can damage the entire network integration. Now days, the advent of SDN products are being used in the research, development and corporate industry so that the effective control including routing, scheduling, security and related algorithms can be implemented on real networks. In this research paper, a unique and pragmatic implementation of the virtualization is done using mininet-openflow integration to evaluate the performance of network and data packets transmission. The mininet-openflow network is being used to get the effective and real-time results using virtual machines on different types of network topologies.

Keywords - MiniNet, OpenFlow, Software Defined Networking, Virtualization, Open Source SDN Suite

INTRODUCTION

Software-defined networking (SDN) is the unique and effective technology for the development and integration of agile as well as flexible network infrastructure using virtualization in the contemporary data centers.

Software-defined networking is the approach towards computer networking which allows the network and digital resources administrators in managing the network services using higher level abstraction and effective functionality. This process is implemented with the use of decoupling of system which makes the decisions where data traffic is to be sent with the use the control plane.

In the current scenario and implementations, OpenFlow [1] is deployed as Software Defined Networking (SDN) technology. In SDN technology there is the decoupling of control and data planes in network [3][4]. A software based controller is responsible for managing the forwarding information of one or more switches; the hardware only handles the forwarding of traffic according to the rules set by the controller [5]. OpenFlow is an SDN technology proposed to standardize the way that a controller communicates with network devices in SDN architecture. It was proposed to enable researchers to test new ideas in a production environment [6][7][8].

OpenFlow provides the migration layer for control logic from a switch into the controller. It presents a protocol for the effective communication between controller and network switches [9][10].

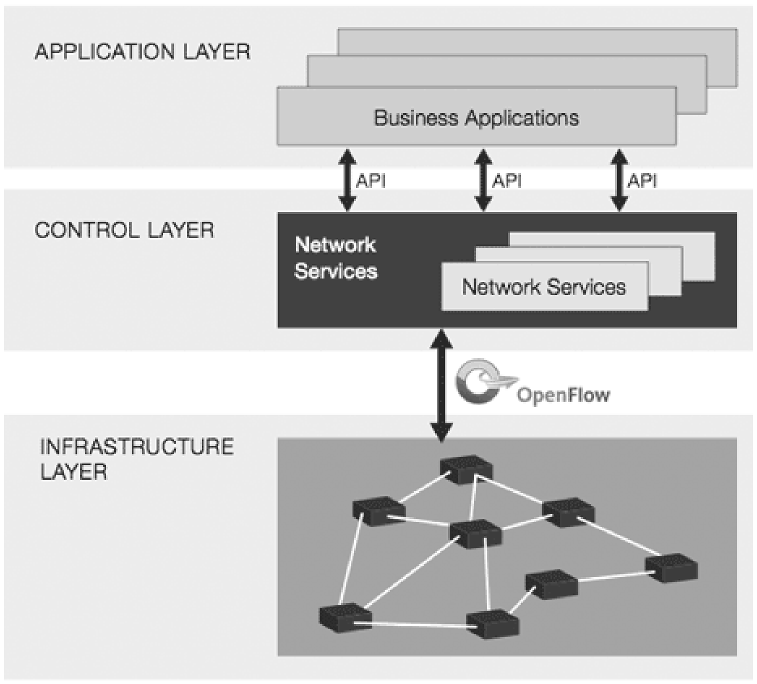


Figure 1 – Software Defined Networking [2]

PROBLEM FORMULATION AND RESEARCH OBJECTIVES

The existing implementations of the classical virtualization and network components management in a grid and distributed environment are very complex and less cost effective. As the cost is directly associated with the turnaround time and complexity, it is very important to keep track of these issues to overcome the flaws in network infrastructure management.

There is the need of software defined networking using which the complete set of network components can be managed with efficiency and overall cost factor can be reduced a lot.

Following are the research objectives in this work which are accomplished the simulation -

- To evaluate the performance of SDN suite and associated virtualization
- To perform the experiments on varying number of nodes and network devices
- To calculate and analyze the turnaround time and round trap time in each simulation iteration so that the effectiveness can be measured.

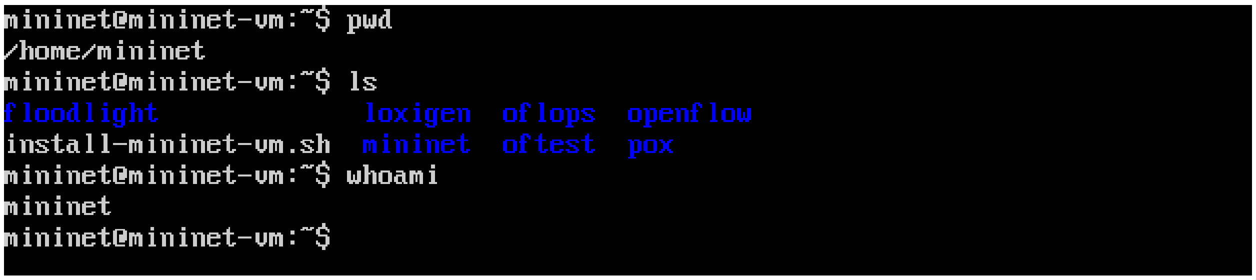
PROPOSED WORK AND IMPLEMENTATION

Following tools are used for implementation of the complete virtualization scenario -

1. Windows 7 (Host OS)
2. putty (SSH Client)
3. Ubuntu 14.04 (Guest OS)
4. Oracle VirtualBox (Installation of Guest OS)
5. MiniNet 2.2.1
6. OpenFlow
7. Floodlight Controller

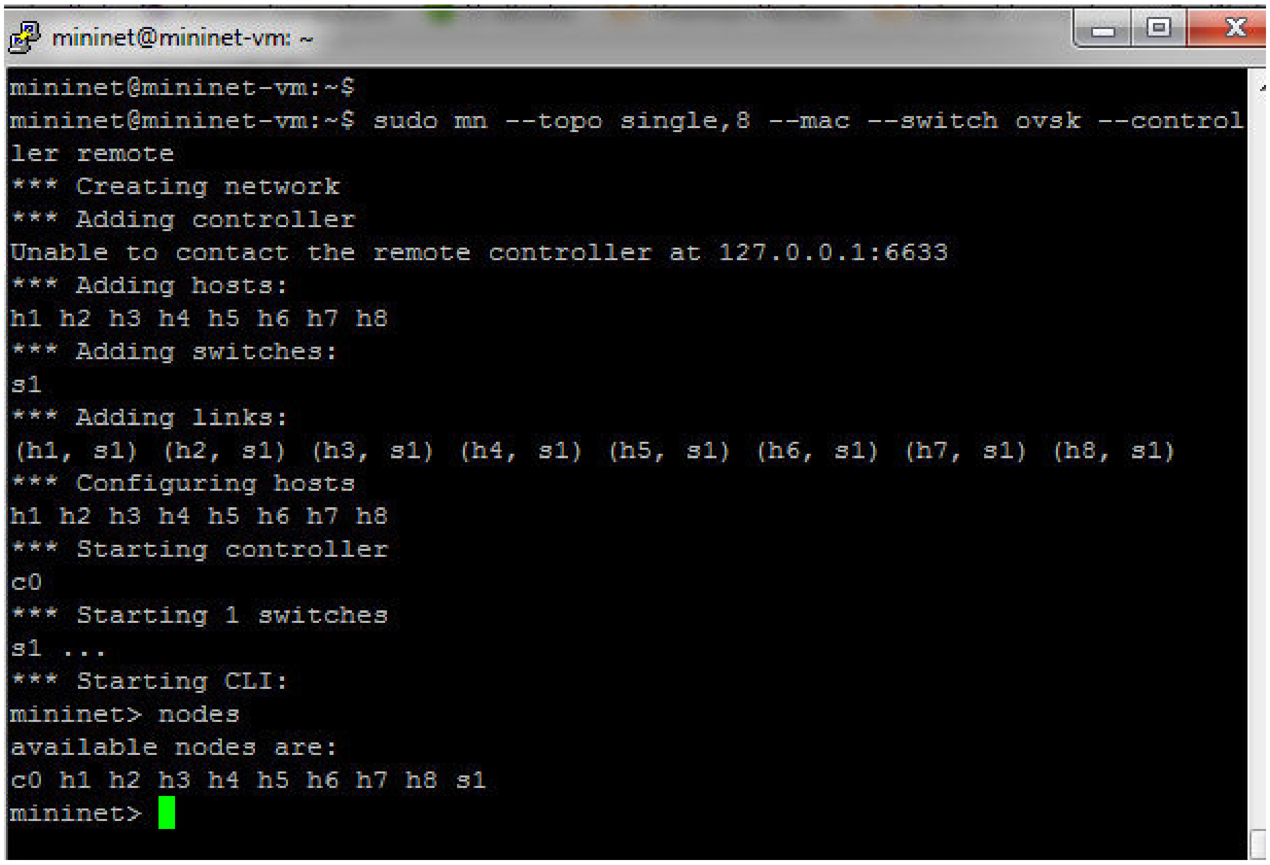
RESULTS AND DISCUSSION

To perform the implementation different scenarios and topologies of the network infrastructure are modeled and simulated. Using MiniNet and OpenFlow, the virtualization and remote connections are established. With the assorted topologies and simulations, following results are fetched and analyzed in tabular form which encapsulates the time as major parameter on multiple nodes topologies.



```
mininet@mininet-vm:~$ pwd
/home/mininet
mininet@mininet-vm:~$ ls
floodlight           loxigen  oflops  openflow
install-mininet-vm.sh  mininet  oftest  pox
mininet@mininet-vm:~$ whoami
mininet
mininet@mininet-vm:~$
```

Figure 2 – Setting up and Analyzing the mininet

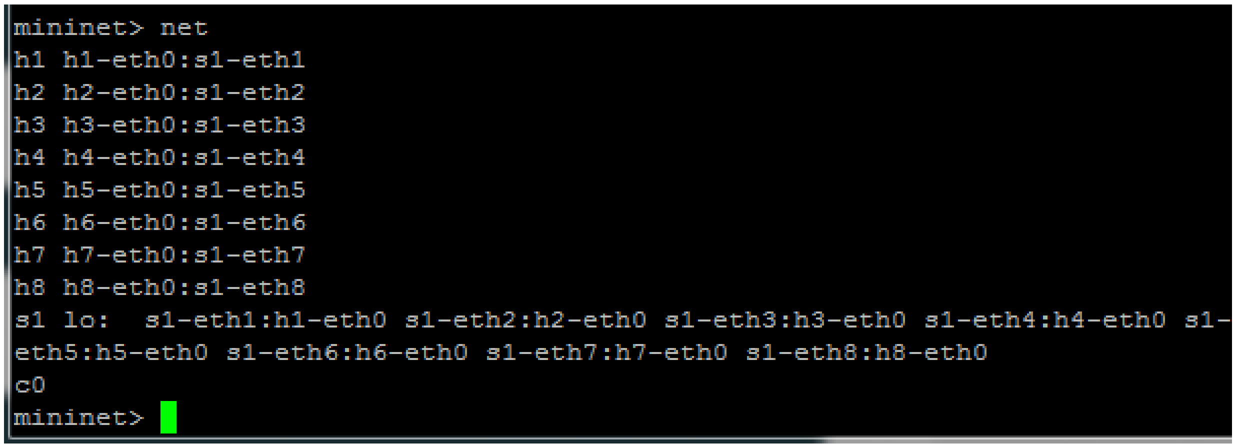


```
mininet@mininet-vm:~$
mininet@mininet-vm:~$ sudo mn --topo single,8 --mac --switch ovsk --control
ler remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1
*** Adding links:
 (h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s1) (h8, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 h6 h7 h8 s1
mininet>
```

Figure 3 – Creating the nodes and switches in mininet

With the implementation of figure above, the following operations are performed at SSH Client-

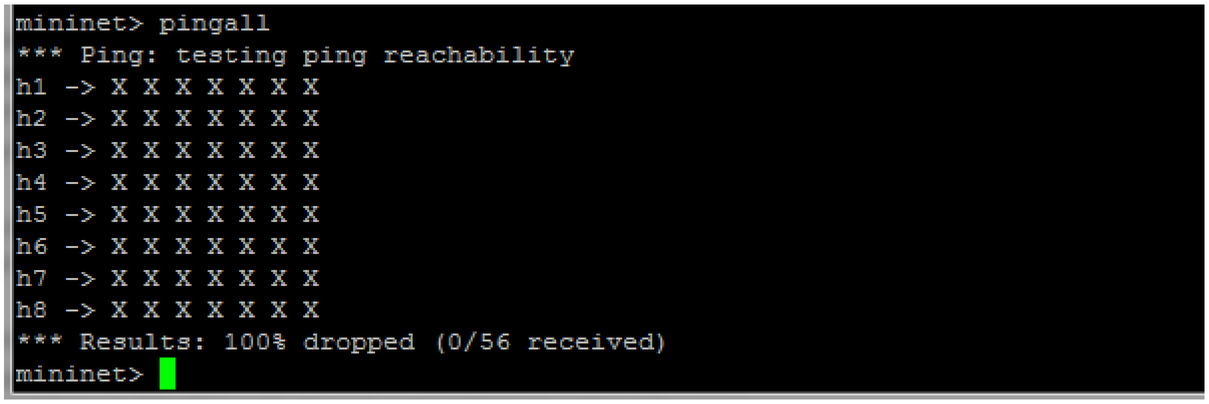
- Creation of 8 virtual hosts having separate IP addresses.
- Creation of single OpenFlow switch in kernel having 3 ports.
- Establishment of connection with every virtual host to switch using virtual ethernet cable.
- Configuration of MAC address of each host equal to its IP.
- Configuration of OpenFlow switch for connection to the remote controller.



```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
h5 h5-eth0:s1-eth5
h6 h6-eth0:s1-eth6
h7 h7-eth0:s1-eth7
h8 h8-eth0:s1-eth8
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0 s1-
eth5:h5-eth0 s1-eth6:h6-eth0 s1-eth7:h7-eth0 s1-eth8:h8-eth0
c0
mininet>
```

Figure 4 – Viewing the network connections and configuration in mininet

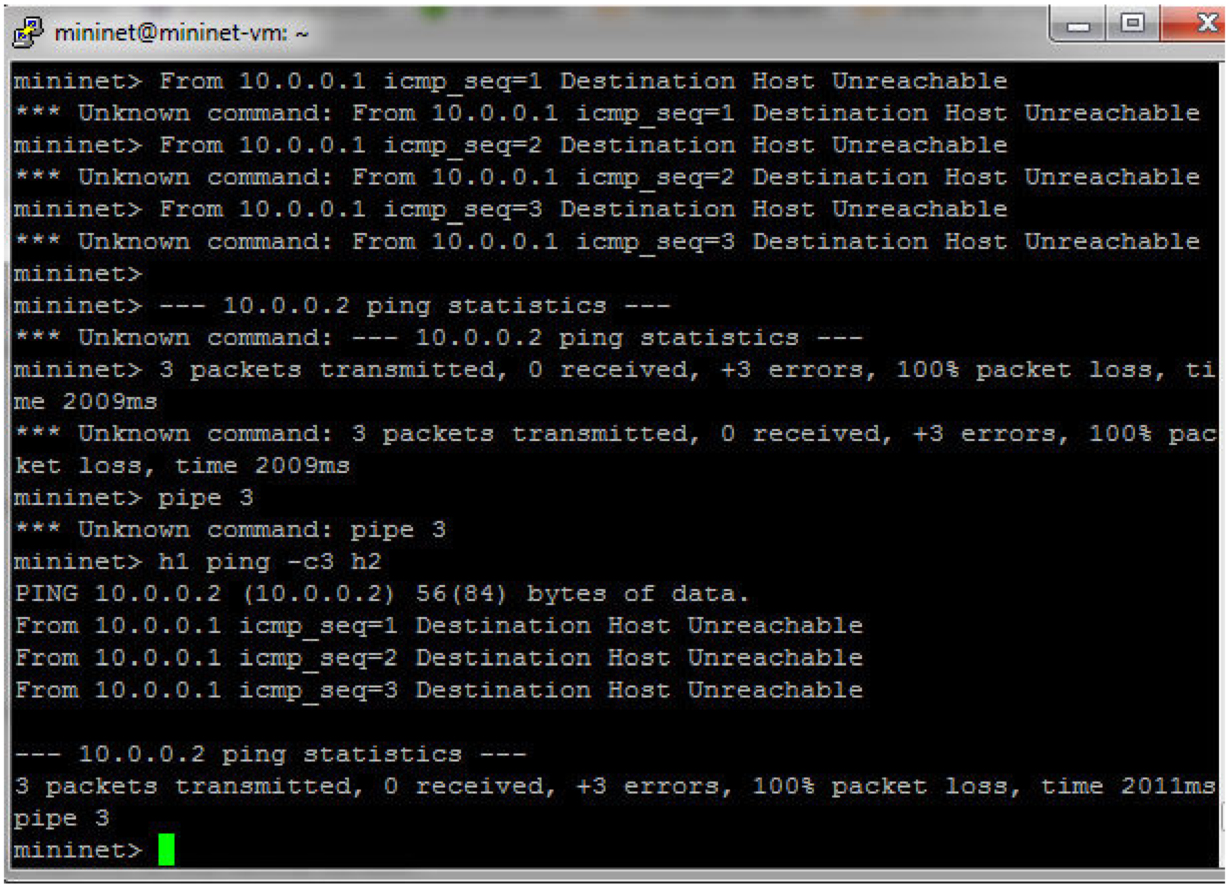
An easier way to run this test is to use the Mininet CLI built-in **pingall** command, which does an all-pairs **ping**:



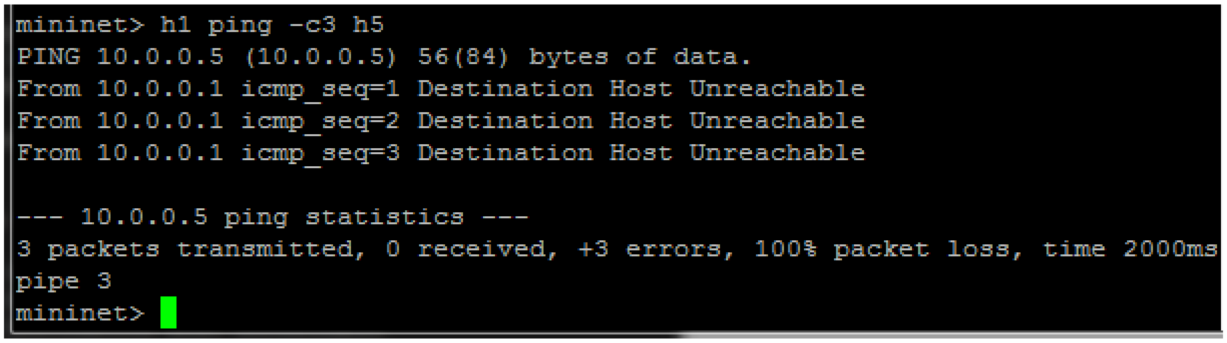
```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X X X X
h2 -> X X X X X X X
h3 -> X X X X X X X
h4 -> X X X X X X X
h5 -> X X X X X X X
h6 -> X X X X X X X
h7 -> X X X X X X X
h8 -> X X X X X X X
*** Results: 100% dropped (0/56 received)
mininet>
```

Figure 5 – Ping Reachability Analysis

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 6 Issue 2 March 2016
International Manuscript ID : 2249054XV6I2032016-22



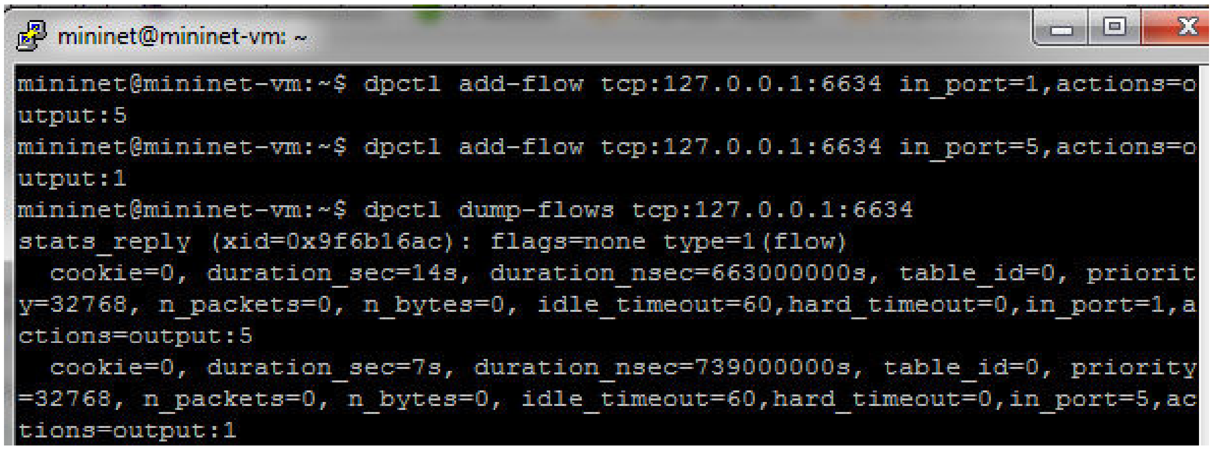
```
mininet> From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
*** Unknown command: From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
mininet> From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
*** Unknown command: From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
mininet> From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
*** Unknown command: From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
mininet>
mininet> --- 10.0.0.2 ping statistics ---
*** Unknown command: --- 10.0.0.2 ping statistics ---
mininet> 3 packets transmitted, 0 received, +3 errors, 100% packet loss, ti
me 2009ms
*** Unknown command: 3 packets transmitted, 0 received, +3 errors, 100% pac
ket loss, time 2009ms
mininet> pipe 3
*** Unknown command: pipe 3
mininet> h1 ping -c3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2011ms
pipe 3
mininet>
```



```
mininet> h1 ping -c3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2000ms
pipe 3
mininet>
```

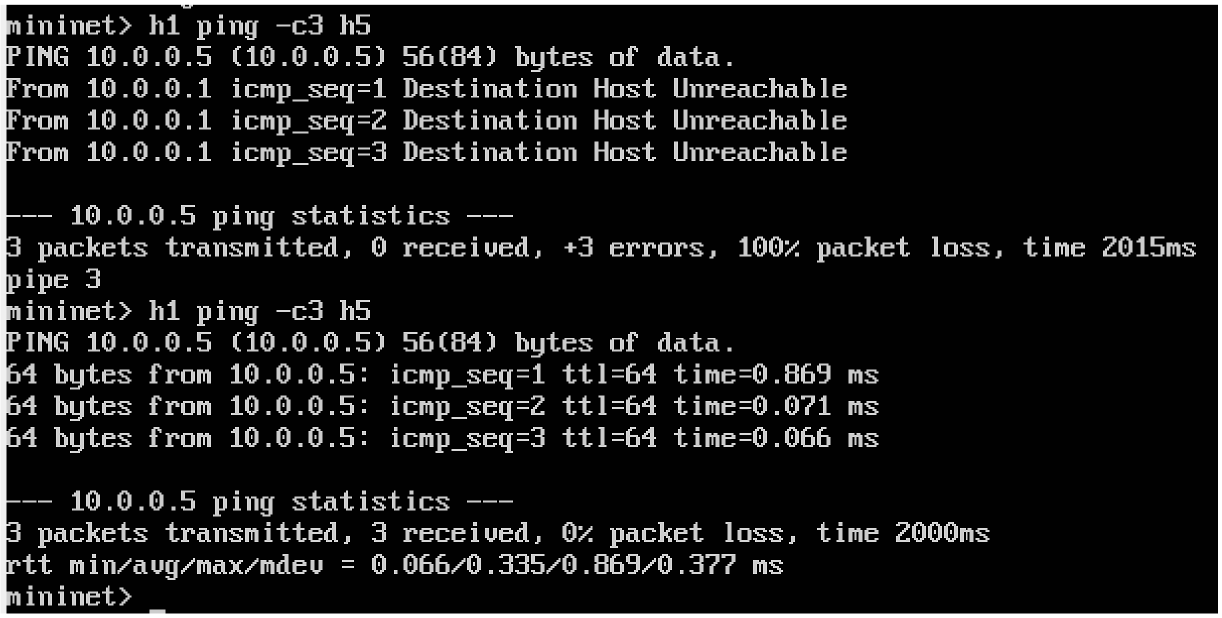
Figure 6 – Initial ping request failed with 100% packets loss

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 6 Issue 2 March 2016
International Manuscript ID : 2249054XV6I2032016-22



```
mininet@mininet-vm: ~
mininet@mininet-vm:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=o
utput:5
mininet@mininet-vm:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=5,actions=o
utput:1
mininet@mininet-vm:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x9f6b16ac): flags=none type=1(flow)
  cookie=0, duration_sec=14s, duration_nsec=663000000s, table_id=0, priorit
y=32768, n_packets=0, n_bytes=0, idle_timeout=60,hard_timeout=0,in_port=1,a
ctions=output:5
  cookie=0, duration_sec=7s, duration_nsec=739000000s, table_id=0, priority
=32768, n_packets=0, n_bytes=0, idle_timeout=60,hard_timeout=0,in_port=5,ac
tions=output:1
```

Figure 7 – Creating bidirectional connections for ping

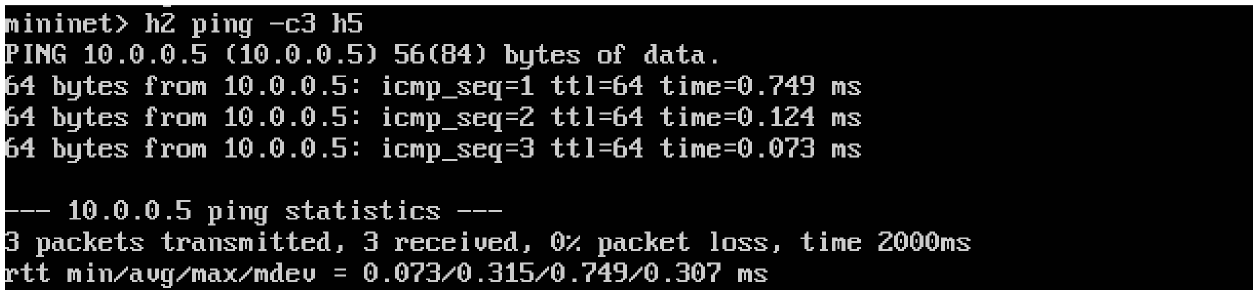


```
mininet> h1 ping -c3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2015ms
pipe 3
mininet> h1 ping -c3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=0.869 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.071 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.066 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.066/0.335/0.869/0.377 ms
mininet> _
```

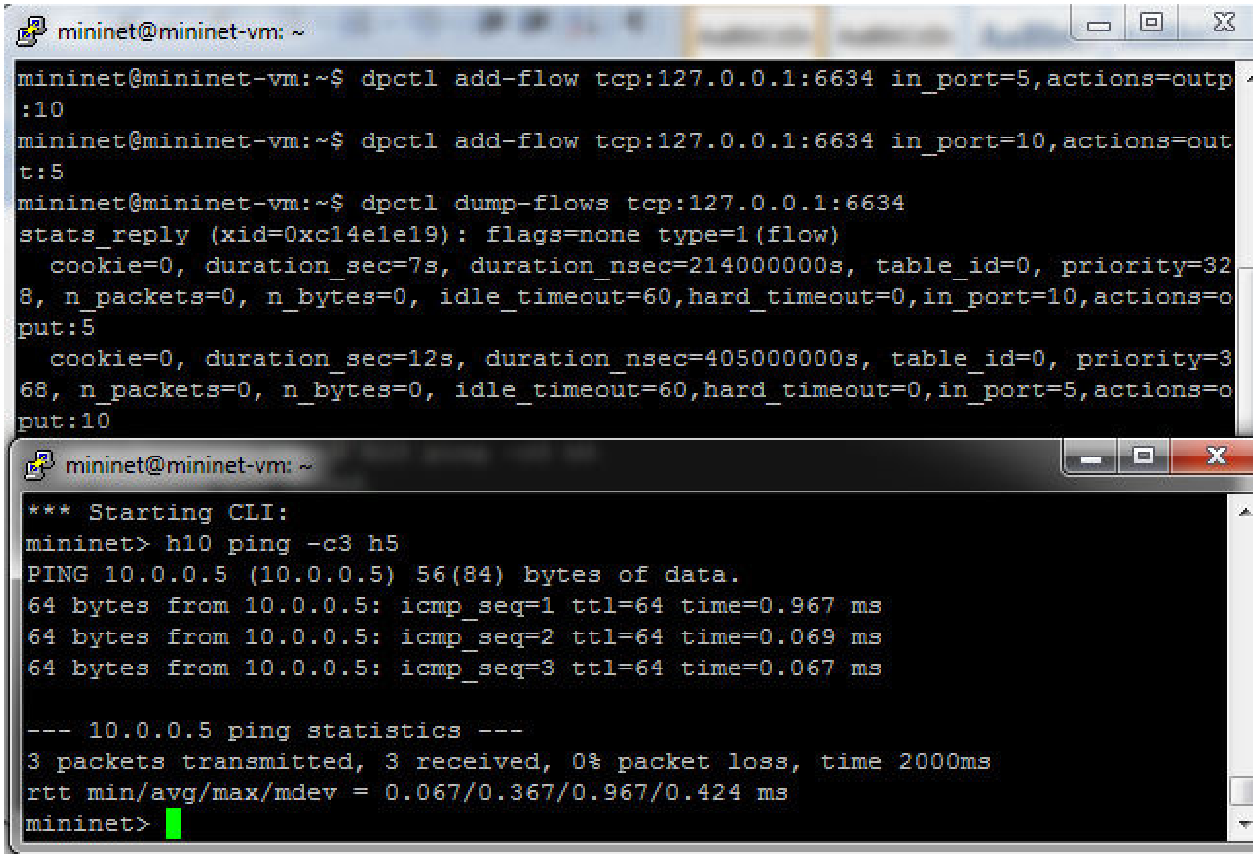
Figure 8 – Measurement of time from ping request after connections

In the similar implementation methodology, the data packet transfer route is set from the nodes 2-5 and 5-10.

Following results are fetched from the simulation of ping from the nodes in network



```
mininet> h2 ping -c3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=0.749 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.124 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.073 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.073/0.315/0.749/0.307 ms
```



```
mininet@mininet-vm: ~
mininet@mininet-vm:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=5,actions=outp
:10
mininet@mininet-vm:~$ dpctl add-flow tcp:127.0.0.1:6634 in_port=10,actions=out
t:5
mininet@mininet-vm:~$ dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0xc14e1e19): flags=none type=1(flow)
  cookie=0, duration_sec=7s, duration_nsec=214000000s, table_id=0, priority=32
8, n_packets=0, n_bytes=0, idle_timeout=60,hard_timeout=0,in_port=10,actions=o
put:5
  cookie=0, duration_sec=12s, duration_nsec=405000000s, table_id=0, priority=3
68, n_packets=0, n_bytes=0, idle_timeout=60,hard_timeout=0,in_port=5,actions=o
put:10
```

```
mininet@mininet-vm: ~
*** Starting CLI:
mininet> h10 ping -c3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=0.967 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.069 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.067 ms

--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.067/0.367/0.967/0.424 ms
mininet>
```

Figure 9 – Measurement of time from ping request after connections for multiple nodes

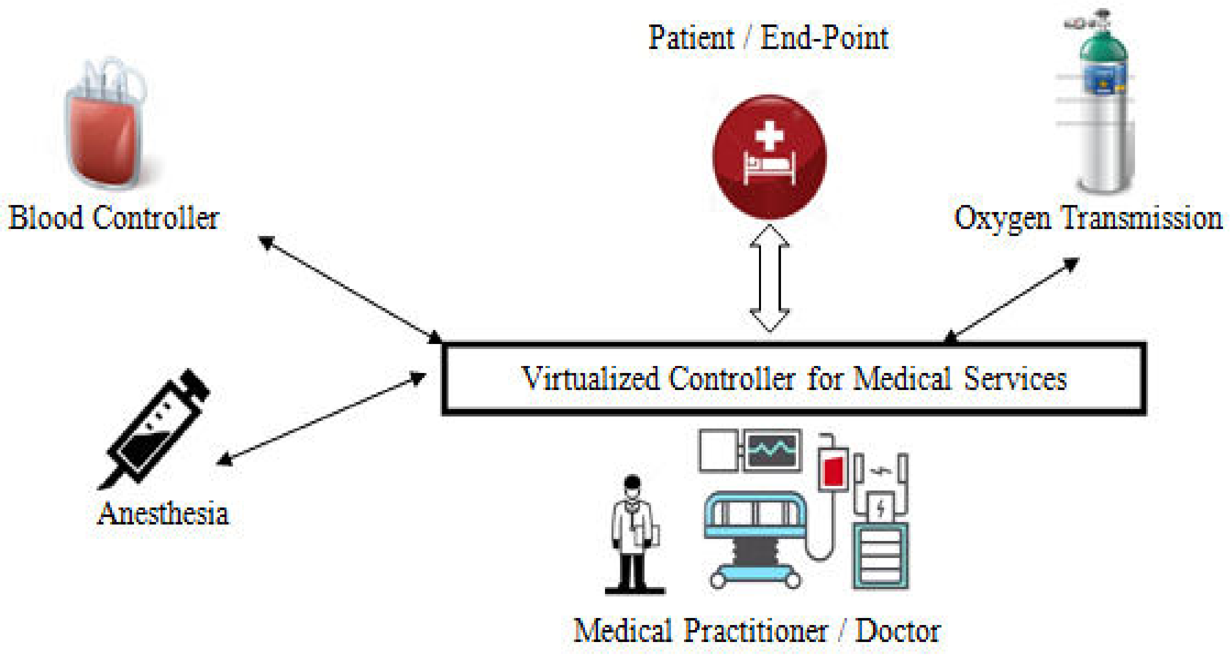

Patient / End-Point
Blood Controller
Oxygen Transmission
Virtualized Controller for Medical Services
Anesthesia
Medical Practitioner / Doctor

Figure 10 – Proposed Architecture and Implementation Scenario

Table 1 - Comparison of the Results in different network ping Scenarios

Number of Nodes	**Average Round Trap Time (ms)**
3 (ping Forward)	0.329
3 (ping Reverse)	0.195
8 (ping Forward)	0.335
8 (ping Reverse)	0.216
10 (ping Forward)	0.367
10 (ping Reverse)	0.219
20 (ping Forward)	0.366

20 (ping Reverse)	0.206

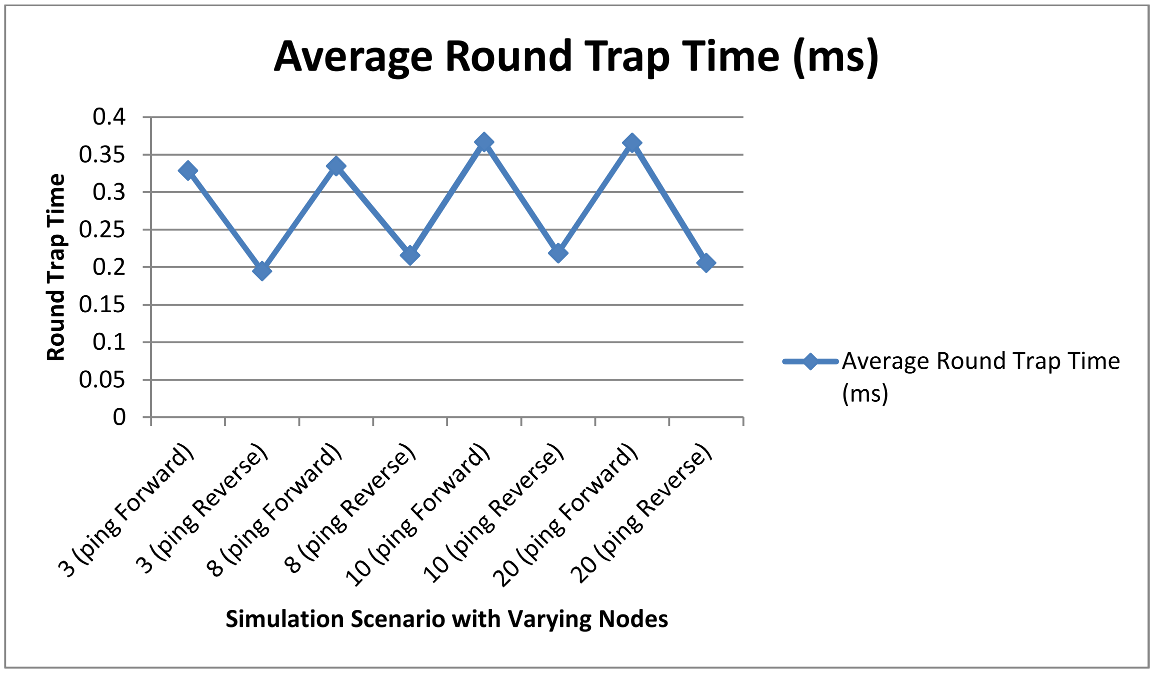


Figure 11 – Evaluation of round trap time after ping in multiple network topologies

CONCLUSION

With the use of software defined networking, the actual implementation and management of the network infrastructure is made easy. The base technologies behind the scenario are the virtualization and remote clients. In this manuscript, the implementation is done based on multiple simulation scenarios having different number of nodes, switches and controllers to analyze the performance. In the results and simulation, it is found that the reverse direction ping in secured shell client is taking less time rather than the streamline connection. If the bidirectional connection is set and established between x and y, in this case the forward connection ping is taking more time than the reverse connection ping.

REFERENCES

[1] 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, 2008.

[2] Tourrilhes, J., Sharma, P., Banerjee, S., & Pettit, J. (2014). SDN and OpenFlow Evolution: A Standards Perspective. *Computer*, (11), 22-29.

[3] Azodolmolky, S. (2013). *Software Defined Networking with OpenFlow*. Packt Publishing Ltd.

[4] Lantz, B., Heller, B., & McKeown, N. (2010, October). A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (p. 19). ACM.

[5] Msahli, M., Pujolle, G., Serhrouchni, A., Fadlallah, A., & Guenane, F. (2012, November). Openflow and on demand Networks. In *Network of the Future (NOF), 2012 Third International Conference on the* (pp. 1-5). IEEE.

[6] Salsano, S., Ventre, P. L., Prete, L., Siracusano, G., Gerola, M., & Salvadori, E. (2014, September). OSHI-Open Source Hybrid IP/SDN networking (and its emulation on Mininet and on distributed SDN testbeds). In *Software Defined Networks (EWSDN), 2014 Third European Workshop on* (pp. 13-18). IEEE.

[7] Lara, A., Kolasani, A., & Ramamurthy, B. (2014). Network innovation using openflow: A survey. *Communications Surveys & Tutorials, IEEE*, *16*(1), 493-512.

[8] Boughzala, B., Ben Ali, R., Lemay, M., Lemieux, Y., & Cherkaoui, O. (2011, May). OpenFlow supporting inter-domain virtual machine migration. In *Wireless and Optical Communications Networks (WOCN), 2011 Eighth International Conference on* (pp. 1-7). IEEE.

[9] Wang, S. Y. (2014, June). Comparison of SDN OpenFlow network simulator and emulators: EstiNet vs. Mininet. In *Computers and Communication (ISCC), 2014 IEEE Symposium on* (pp. 1-6). IEEE.

[10] Kim, H., Kim, J., & Ko, Y. B. (2014, February). Developing a cost-effective OpenFlow testbed for small-scale Software Defined Networking. In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*(pp. 758-761). IEEE.