Mininet as Software Defined Networking Testing Platform

Karamjeet Kaur
Panjab University
20 PUBLICATIONS **440** CITATIONS
SEE PROFILE

Japinder Singh
Shaheed Bhagat Singh State Technical Campus
12 PUBLICATIONS **536** CITATIONS
SEE PROFILE

Navtej Ghumman
Shaheed Bhagat Singh State University
27 PUBLICATIONS **561** CITATIONS
SEE PROFILE

Mininet as Software Defined Networking Testing Platform

Karamjeet Kaur[1], Japinder Singh[2] and Navtej Singh Ghumman[3]
[1,2,3]*Department of Computer Science and Engineering,*
Shaheed Bhagat Singh State Technical Campus, Ferozepur, India
E-mail: [1]bhullar1991@gmail.com, [2]japitaneja@gmail.com, [3]navtejghumman@yahoo.com

***Abstract*—Mininet is an emulator for deploying large networks on the limited resources of a simple single Computer or Virtual Machine. Mininet has been created for enabling research in Software Defined Networking (SDN) and OpenFlow. Mininet emulator allows running unmodified code interactively on virtual hardware on a simple PC. It provides convenience and realism at very low cost. The alternative to Mininet is hardware test beds which are fast, accurate but very expensive and shared. The other option is to use simulator which is very cheap but sometimes slow and require code modification. Mininet offers ease of use, performance accuracy and scalability.**

Keywords: Mininet, SDN, OpenFlow

I. INTRODUCTION

There is need to model hosts, switches, links and SDN/OpenFlow controllers. Mininet [1] allows creating topologies of very large scale size up to thousands of nodes and perform test on them very easily. It has very simple command line tools and API. Mininet allows the user to easily create, customize, share and test SDN networks. (Fig. 1).

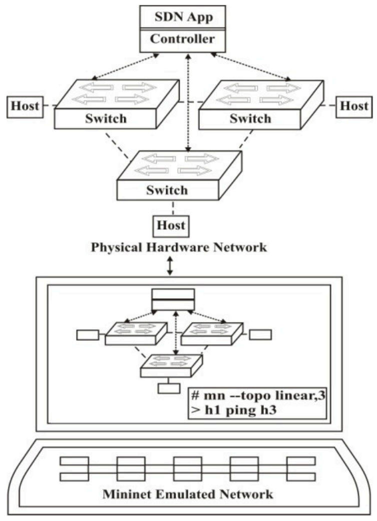


Fig. 1 Emulating Real Networks in Mininet

Mininet is freely available open source software that emulates OpenFlow devices and SDN controllers. Mininet can simulates SDN networks, can run a controller for experiments [2]. It allows emulating real world network scenarios Couple of SDN controllers are included with in Mininet VM. The default controllers are good but for implementing advance concepts, POX [3] controller is used.

II. SDN AND OPEN FLOW

In traditional networks [4], the data plane and the control plane are tightly coupled on the same device (Fig. 2). Therefore in traditional networks, development of new applications and modification in behavior of existing devices is very difficult task. Software Defined networking (SDN) [5] overcomes these problems by shifting the control logic from devices to the centralized place. The shifted control logic is called SDN Controller or Network Operating System (NOS) [6]. The controller has a global view of the entire network, Therefore by using SDN you can manage the functionality of network in a very efficient manner.

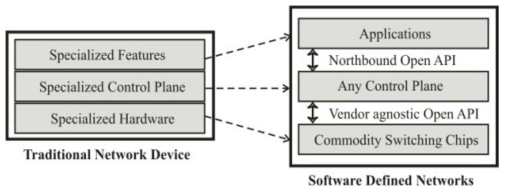


Fig. 2 Separate Control and Data Plane

Open Flow [7] is a standard protocol that is used to provide a communication between controller and dumb device. The controller and dumb devices are called control plane and data plane respectively. The Open Flow controller is responsible for deciding which action is to be performed by the switch. The decision approach is either Reactive or Proactive.

In the Reactive approach, when a packet arrives at a switch, switch does not know how to handle that packet. Therefore switch sends the packet to the controller [8]. Controller is responsible for inserting a flow entry into the flow table of a switch using the openflow protocol. The main disadvantage of this approach is that switch is totally dependent upon controller decision. So when a switch loses the connection with the controller, it cannot handle that packet.

In the Proactive approach [9], the controller pre populates the flow entries in the flow tables of each switch. This approach overcomes the limitation of reactive approach because even if the switch loses the connection with controller, it does not disrupt traffic.

The main advantages of SDN over traditional approach are that it allows you to quickly test and deploy new applications in real network, minimize capital and operating expenses and allows centralized management of each switch.

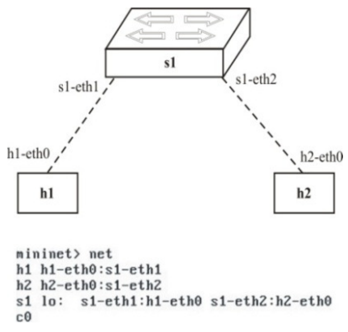
III. Mininet Topologies

Mininet contains number of default topologies such as minimal, single, reversed, linear and tree [10]. This section explains these topologies one by one. Understanding naming method for interfaces, hosts and switches is essential for prospering using Mininet. Switches are named from s1 to sN. Hosts are named h1 to hN. Host interfaces are named prefixed with host's name following by Ethernet name starting with 0. First interface of host 'h1' is called 'h1-eth0' and third interface of host 'h2' is called 'h2-eth2'. First port of switch 's1' is named 's1-eth1'. In switches, numbering begins with 1.

A. Minimal

Minimal is very simple topology that contains 1 OpenFlow switch and 2 hosts. It also creates links between switch and two hosts (Fig. 3).

# mn--topo minimal



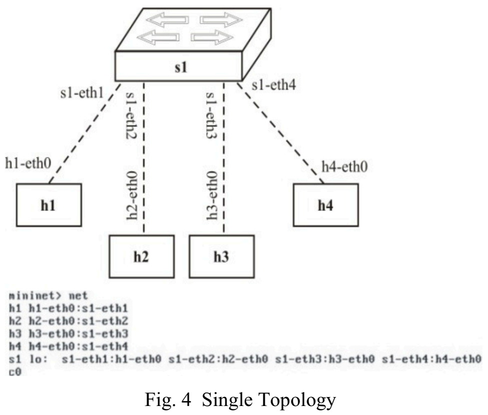
```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
c0
```

Fig. 3 Minimal Topology

B. *Single*

It is a simple topology with one openflow switch and k hosts. It also creates a link between switch and k hosts (Fig. 4).

# mn--topo single, 4

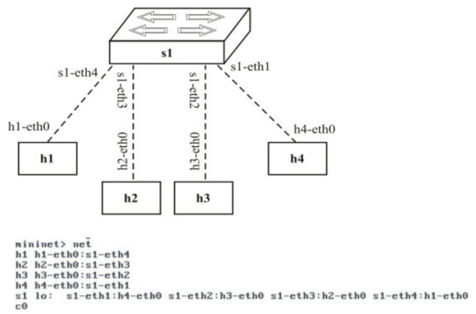


```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0
c0
```

Fig. 4 Single Topology

C. *Reversed*

It is similar to single topology but connection order is reversed (Fig. 5).



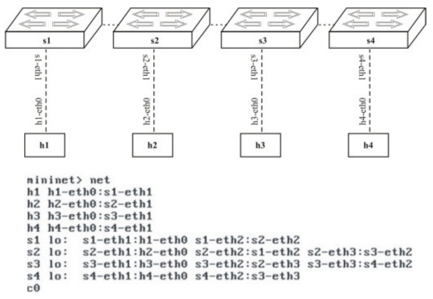
```
mininet> net
h1 h1-eth0:s1-eth4
h2 h2-eth0:s1-eth3
h3 h3-eth0:s1-eth2
h4 h4-eth0:s1-eth1
s1 lo:  s1-eth1:h4-eth0 s1-eth2:h3-eth0 s1-eth3:h2-eth0 s1-eth4:h1-eth0
c0
```

Fig. 5 Reversed Topology

# mn--topo reversed, 4

D. Linear

Linear topology contains k switches and k hosts. It also creates a link between each switch and each host and among the switches (Fig. 6).



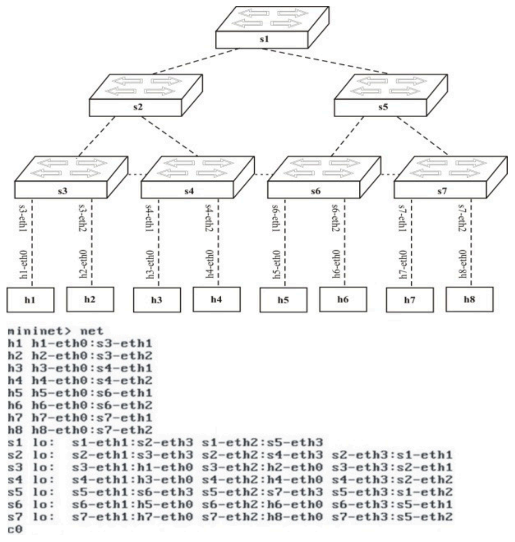
```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
s1 lo:  s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo:  s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:h3-eth0 s3-eth2:s2-eth3 s3-eth3:s4-eth2
s4 lo:  s4-eth1:h4-eth0 s4-eth2:s3-eth3
c0
```

Fig. 6 Linear Topology

# mn--topo linear, 4

E. Tree

Tree topology contains k levels and 2 hosts are attached to per switch (Fig. 7).



```
mininet> net
h1 h1-eth0:s3-eth1
h2 h2-eth0:s3-eth2
h3 h3-eth0:s4-eth1
h4 h4-eth0:s4-eth2
h5 h5-eth0:s6-eth1
h6 h6-eth0:s6-eth2
h7 h7-eth0:s7-eth1
h8 h8-eth0:s7-eth2
s1 lo:  s1-eth1:s2-eth3 s1-eth2:s5-eth3
s2 lo:  s2-eth1:s3-eth3 s2-eth2:s4-eth3 s2-eth3:s1-eth1
s3 lo:  s3-eth1:h1-eth0 s3-eth2:h2-eth0 s3-eth3:s2-eth1
s4 lo:  s4-eth1:h3-eth0 s4-eth2:h4-eth0 s4-eth3:s2-eth2
s5 lo:  s5-eth1:s6-eth3 s5-eth2:s7-eth3 s5-eth3:s1-eth2
s6 lo:  s6-eth1:h5-eth0 s6-eth2:h6-eth0 s6-eth3:s5-eth1
s7 lo:  s7-eth1:h7-eth0 s7-eth2:h8-eth0 s7-eth3:s5-eth2
c0
```

Fig. 7 Tree Topology

# mn--topo tree, 3

IV. Creating Custom Topologies

Using Mininet, you can easily create a custom topologies [11]. For example creating custom topology having 2 switches and 5 hosts (Fig. 8) needs just writing a few lines of Python [12] code. You can also easily create very complex flexible, robust. You can also configured that topology based on the parameters that are to be pass to it, and reuse that topology for multiple experiments.

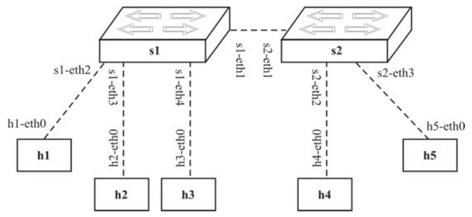
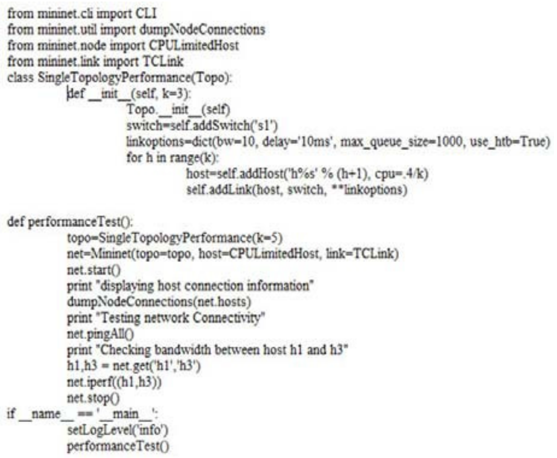


Fig. 8 Custom Topology

In the following Listing 1 contain Python code for creating custom topology having 2 switches and 5 hosts. This topology is run in Mininet by using following command.

# Python CustomTopologyPerformance.py



```
from mininet.cli import CLI
from mininet.util import dumpNodeConnections
from mininet.node import CPULimitedHost
from mininet.link import TCLink
class SingleTopologyPerformance(Topo):
        def __init__(self, k=3):
                Topo.__init__(self)
                switch=self.addSwitch('s1')
                linkoptions=dict(bw=10, delay='10ms', max_queue_size=1000, use_htb=True)
                for h in range(k):
                        host=self.addHost('h%s' % (h+1), cpu=.4/k)
                        self.addLink(host, switch, **linkoptions)

def performanceTest():
        topo=SingleTopologyPerformance(k=5)
        net=Mininet(topo=topo, host=CPULimitedHost, link=TCLink)
        net.start()
        print "displaying host connection information"
        dumpNodeConnections(net.hosts)
        print "Testing network Connectivity"
        net.pingAll()
        print "Checking bandwidth between host h1 and h3"
        h1,h3 = net.get('h1','h3')
        net.iperf((h1,h3))
        net.stop()
if __name__ == '__main__':
        setLogLevel('info')
        performanceTest()
```

Listing 1 Custom Topology Code

There are number of classes, functions, methods and variables in the Listing 1

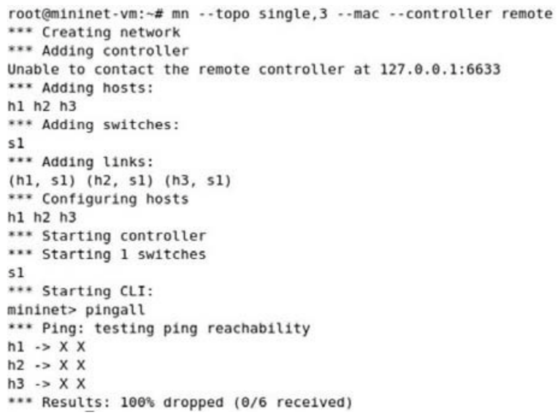
1. Topo: It is a base class for Mininet topologies.
2. Add Host (name, cpu = f): It is used for adding a host to the topology which contains two parameters. First parameter specifies the name of host and second specifies the fraction of overall system CPU resources that is to be allocated to the virtual host.
3. Add Switch(): It is used for adding a switch to the topology and returns the switch name, for example s1.
4. Add Link (node1, node2, **link options): It is used for adding a bidirectional link which contains three parameters. The first, second parameter specify the host and switch name respectively and third parameter specify the dictionary that contain number of options such as bandwidth, delay and loss characteristics, with a maximum queue size.
5. start(): It is used for starting your network.
6. stop(): It is used for stopping your network.
7. Mininet: It is used as a main class to create and manage a network.
8. net. hosts: It is used to show all the hosts in network.
9. ping All (): This is used to check connectivity between all nodes.
10. Set Log Level: There are number of Log Level such as info, debug, and output. Info is recommended as it provides useful information.
11. Dump Node Connections (): Dumps connections to/from a set of nodes.

There are basically two classes available such as CPU Limited Host and TC Link that can be used for performance limiting and isolation.

There are number of ways that these classes may be used, but simple way is to specify them as the default host and link classes to Mininet(), and then to apply the appropriate parameters in the topology.

V. Controlling Default Topology with DPCTL

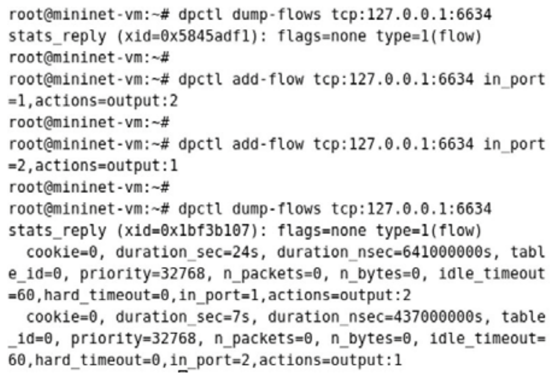
This topology (Fig. 9) creates 1 switch and 3 hosts. The option 'mac' set the mac address of each host according to node number and option 'remote' is to be used to connect switch to remote controller. As shown, hosts can not ping with each other because remote controller is not running, therefore flow table of switch does not contain any flow entry.



```
root@mininet-vm:~# mn --topo single,3 --mac --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
```

Fig. 9 Single Switch, 3 Hosts Topology

There are two methods to add flow entries into flow table of switch, remote controller and 'dpctl' [13] utility that works on port 6634 (Fig. 10). dpctl is a data path controller that comes with OpenFlow reference distribution and is used to manage the flow table of switch by using 'dpctl' commands. After adding flow entries by using ‘dpctl’ command, the host h1 and h2 can ping with each other (Fig. 11).

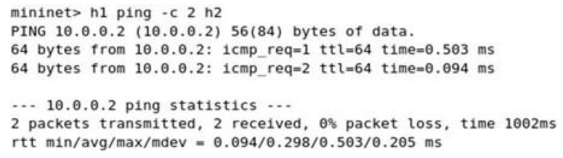


```
root@mininet-vm:~# dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x5845adf1): flags=none type=1(flow)
root@mininet-vm:~#
root@mininet-vm:~# dpctl add-flow tcp:127.0.0.1:6634 in_port
=1,actions=output:2
root@mininet-vm:~#
root@mininet-vm:~# dpctl add-flow tcp:127.0.0.1:6634 in_port
=2,actions=output:1
root@mininet-vm:~#
root@mininet-vm:~# dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0x1bf3b107): flags=none type=1(flow)
  cookie=0, duration_sec=24s, duration_nsec=641000000s, tabl
e_id=0, priority=32768, n_packets=0, n_bytes=0, idle_timeout
=60,hard_timeout=0,in_port=1,actions=output:2
  cookie=0, duration_sec=7s, duration_nsec=437000000s, table
_id=0, priority=32768, n_packets=0, n_bytes=0, idle_timeout=
60,hard_timeout=0,in_port=2,actions=output:1
```

Fig. 10 Flow Management Using DPCTL

Default idle timeout of each flow entry is 60s. Therefore after 60 seconds, flow entries will expire and needs to be added again and again. So installs a flow entry with longer timeout using following command:

dpctl add-flow tcp:127.0.0.1:6634 in_port = 1, idle_timeout = 180, actions = output:2



```
mininet> h1 ping -c 2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.503 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.094 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.094/0.298/0.503/0.205 ms
```

Fig. 11 Connectivity Test

VI. Conclusion

Mininet is a platform for rapid network prototyping. It can run unmodified network application code on small networks as well as very large networks. It is an alternative to run SDN experiments on emulated networks. Real systems are very painful to reconfigure. Virtual machines allow easier topology changes but suffer from scalability issues. Simulators are a good alternative but same source code cannot be deployed on real hardware. There are performance issues on Mininet. The challenges before Mininet are to model networks of very large scale with practical performance.

Acknowledgment

The authors would like to thank Mr. Vipin Gupta from U-Net Solutions, Moga, India for his valuable help.

References

[1] Handigol, Nikhil, Brandon Heller, Vimal kumar, Jeya kumar, Bob Lantz, and Nick McKeown. "Reproducible network experiments using container-based emulation." In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pp. 253–264. ACM, 2012.

[2] Lantz, Bob, Brandon Heller, and Nick McKeown. "A network in a laptop: rapid prototyping for software-defined networks." In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pp. 19. ACM, 2010.

[3] POX at https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-forwarding.l2_learnining.

[4] Feamster, Nick, Jennifer Rexford, and Ellen Zegura. "The road to SDN: an intellectual history of programmable networks." *ACM SIGCOMM Computer Communication Review* 44, No. 2 (2014): 87–98.

[5] Nunes, B.; Mendonca, M.; Nguyen, X.; Obraczka, K.; Turletti, T., "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *Communications Surveys & Tutorials, IEEE,* Vol. 1, No.99, pp. 18.

[6] Shenker, Scott, M. Casado, T. Koponen, and N. McKeown. "The future of networking, and the past of protocols." *Open Networking Summit* (2011).

[7] McKeown, Nick, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. "OpenFlow: enabling innovation in campus networks." *ACM SIGCOMM Computer Communication Review* 38, No. 2 (2008): 69–74.

[8] Fernandez, Marcial P. "Comparing openflow controller paradigms scalability: Reactive and proactive." In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pp. 1009–1016. IEEE, 2013.

[9] Fernandez, Marcial. "Evaluating OpenFlow Controller Paradigms." In *ICN 2013, The Twelfth International Conference on Networks*, pp. 151–157. 2013.

[10] Mininet Topologies at http://www.routereflector.com/2013/11/mini net-as-an-sdn-test-platform

[11] Mininet at https://github.com/mininet/mininet/wiki/Introduction-to-mininet.

[12] Python at https://www.python.org/.

[13] dpctl at http://archive.openflow.org/wk/index.php/HOTITutorial2010.