Mininet as a Container Based Emulator for Software Defined Networks

Manu Sood
Himachal Pradesh University
93 PUBLICATIONS **549** CITATIONS

Volume 4, Issue 12, December 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Mininet as a Container Based Emulator for Software Defined Networks

Kuldeep K. Sharma
Student M Tech, Computer Science,
Dept. of Computer Science,
Himachal Pradesh University, India

Manu Sood
Professor,
Dept. of Computer Science,
Himachal Pradesh University, India

Abstract*— Mininet is network emulation software that allows launching a virtual network with switches, hosts and an SDN controller all with a single command on a single Linux kernel. It is a great way to start learning about SDN and Open-Flow as well as test SDN controller and SDN applications. Mininet can be used to deploy large networks on a single computer or virtual machine provided with limited resources. It is freely available open source software that emulates Open-Flow device and SDN controllers.

***Keywords*— SDN, Mininet, Open-Flow, Python, Wireshark**

I. INTRODUCTION

Mininet is a network emulator which runs collection of end-hosts, switches, routers and links on a single Linux kernel. Mininet [1] emulate compete network on a single virtual machine, running the same kernel system and user code by using lightweight virtualization.

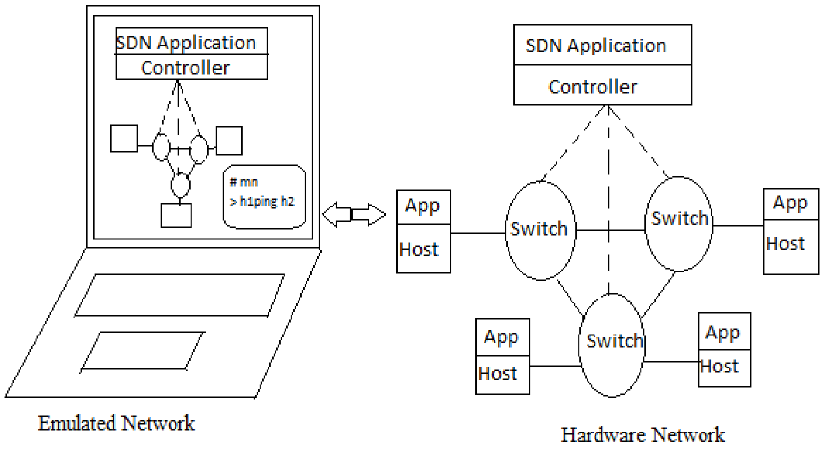
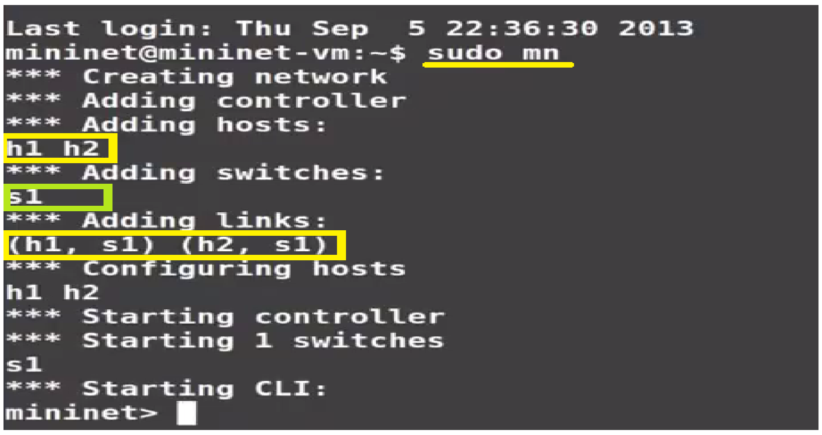


Fig 1 Emulating hardware network using Mininet

Mininet's virtual hosts, switches, links and controllers are the real things only difference is that they are just created using software rather than hardware (Fig 1). Mininet allows the user to easily create, customize, share and test SDN networks. Mininet can simulate SDN networks, can run a controller for experiments [2]. Mininet's VM include couple of SDN controllers to emulate real world scenarios. The default controllers such as POX [3], are good for implementing advance concepts.

II. MININET AND SOFTWARE DEFINED NETWORKING

Initially, Mininet was created by a group of professors at Stanford University to be used as a tool to research and to teach network technologies. Now a day, Mininet is designed to easily create virtual software defined networks consisting of an Open-Flow controller [4], a flat Ethernet network of multiple Open-Flow enabled Ethernet, switches and multiple hosts connected to those switches. It has built-in functions that support using different types of controllers and switches. We can create also complex custom scenarios using the Mininet Python API [5].
SDN switches, hosts, controllers and links can be created by typing commands through Mininet's command line interface as shown in Fig 2. Sudo mn is a command which is used to create two hosts with 1 switch and a controller. SDN architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The Open-Flow [6] protocol is a foundational element for building SDN solutions.
Open-Flow is a standard protocol that is used to provide a communication between controller, control plane and data plane respectively.



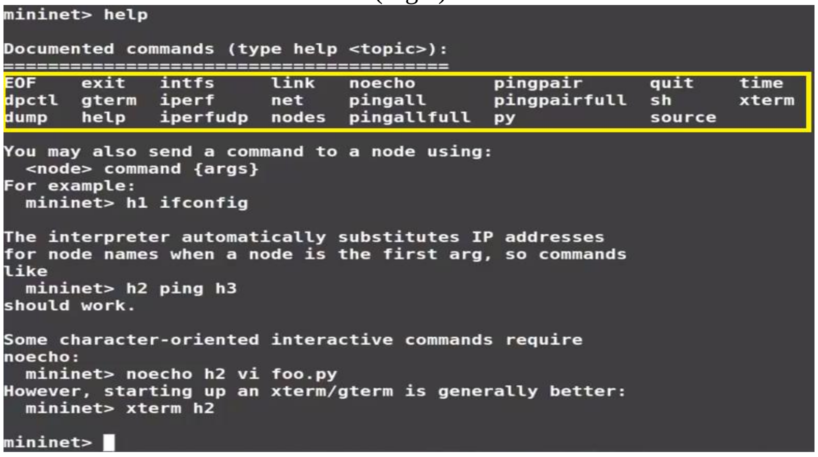
```
Last login: Thu Sep  5 22:36:30 2013
mininet@mininet-vm:~$ sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> 
```

Fig 2 Creating emulated hosts, switches, controllers and links using Mininet

III. MININET BASIC COMMANDS

Mininet command line interface is very easy to use. In order to inspect and control each network element from the command line, Mininet [7] provides a number of commands. Some useful Mininet commands are as listed below:

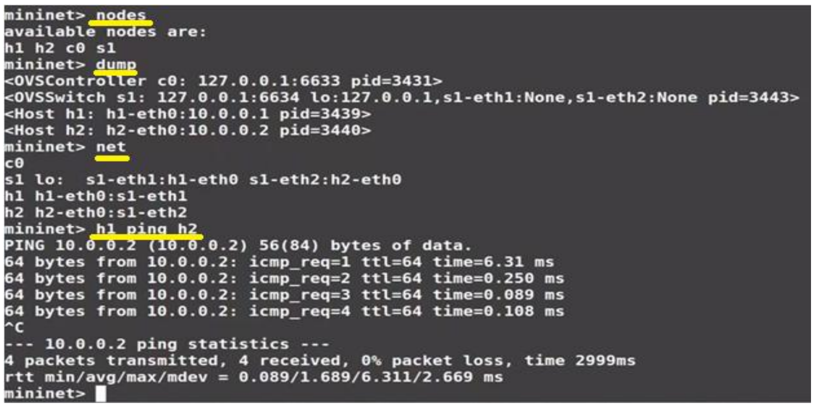
1) *help:* it is used to see a list of available commands (Fig 3).



```
mininet> help

Documented commands (type help <topic>):
========================================
EOF    exit   intfs     link   noecho       pingpair      quit    time
dpctl  gterm  iperf     net    pingall      pingpairfull  sh      xterm
dump   help   iperfudp  nodes  pingallfull  py            source

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
  mininet> xterm h2

mininet> 
```

Fig 3 Displaying help command using Mininet

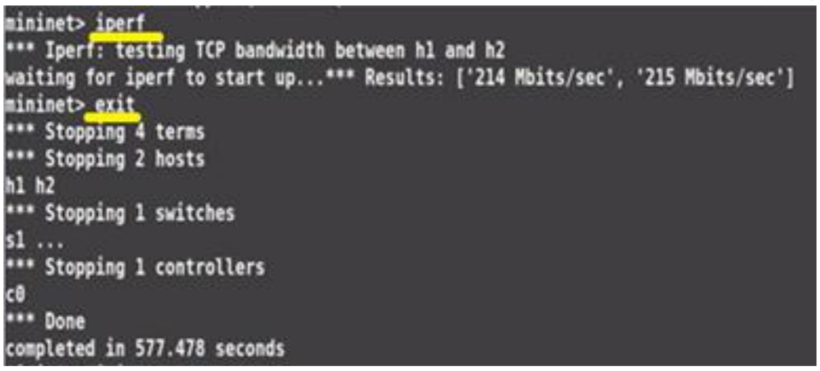
2) *nodes:* it is used to see list of nodes available (Fig 4).
3) *dump:* it lists information about the nodes, switches and controllers in the simulated network (Fig 4).
4) *link:* it tells us which host and switches are connected to each other .
5) *net:* it is similar to link, it shows how network elements are connected to each other in the simulated network (Fig 4).



```
mininet> nodes
available nodes are:
h1 h2 c0 s1
mininet> dump
<OVSController c0: 127.0.0.1:6633 pid=3431>
<OVSSwitch s1: 127.0.0.1:6634 lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=3443>
<Host h1: h1-eth0:10.0.0.1 pid=3439>
<Host h2: h2-eth0:10.0.0.2 pid=3440>
mininet> net
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
mininet> h1 ping 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=6.31 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.250 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.089 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.108 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.089/1.689/6.311/2.669 ms
mininet> 
```

Fig 4 Showing nodes, dump, net and ping commands using Mininet

6) *ping:* it shows the connectivity between particular hosts (Fig 4).
7) *pingall:* it displays the connectivity between all hosts and tells us which hosts are connected to each other.
8) *Pingallfull:* it gives more detail about how the hosts are connected. It will tell us minimum, average and maximum time between two hosts in ms.
9) *iperf:* it is generally used for tcp connection, it is used to test bandwidth between hosts (Fig 5).
10) *eof:* it is used for exiting the current topology.
11) *exit:* it is similar to eof , it is also used to exit from topology (Fig 5).



```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h2
waiting for iperf to start up...*** Results: ['214 Mbits/sec', '215 Mbits/sec']
mininet> exit
*** Stopping 4 terms
*** Stopping 2 hosts
h1 h2
*** Stopping 1 switches
s1 ...
*** Stopping 1 controllers
c0
*** Done
completed in 577.478 seconds
```

Fig 5 showing iperf and exit commands using Mininet

12) *iperfudp*: it is similar to iperf, but used for UDP connections instead of TCP connections.
13) *ifconfig:* it is used to check the IP address of a virtual host.
14) *xterm:* it is used to open a new window connected to a node.
15) *intfs:* it displays the interfaces which are connected.
16) *dpctl:* it is used to view the flows in switch table.

IV. MININET TOPOLOGIES

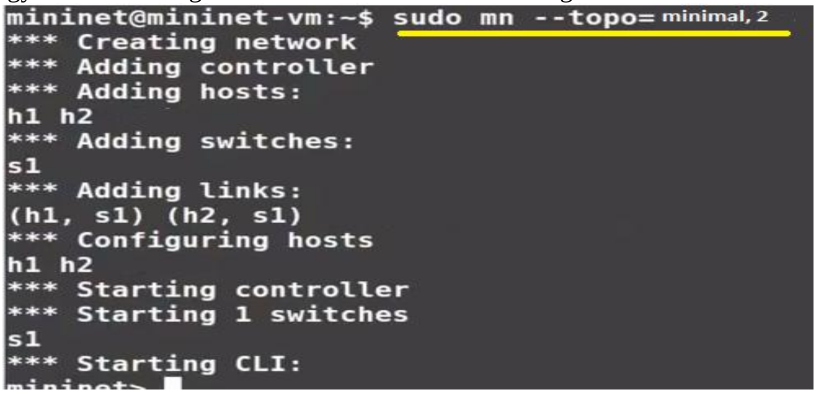
Mininet topologies are basically classified into two types

A. *Default Topologies*

Mininet contains five default topologies [8] such as minimal, single, reversed, linear and tree. Network controller is denoted by C0, switches are denoted by S_1 to S_n and hosts are denoted by H_1 to H_n in the figures used to represent these topologies.

These topologies are briefly explained below:

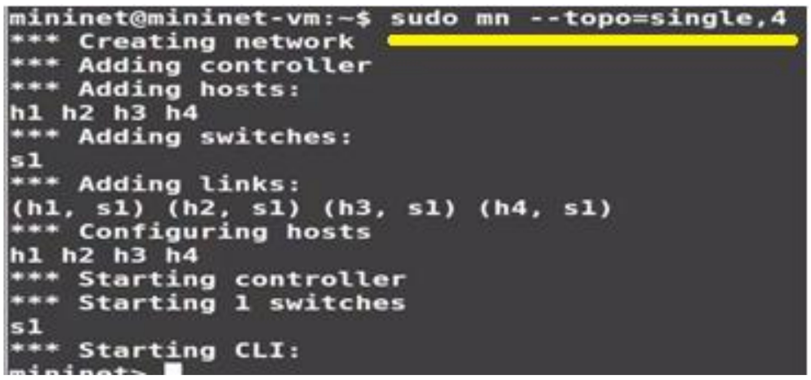
1) *Minimal:* It contains 1 controller, 1 Open-Flow switch and two hosts by creating link between switch and hosts (Fig 10). Topology creation through command line is as shown in Fig 6.



```
mininet@mininet-vm:~$ sudo mn --topo= minimal, 2
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Fig 6 Creating minimal topology using Mininet

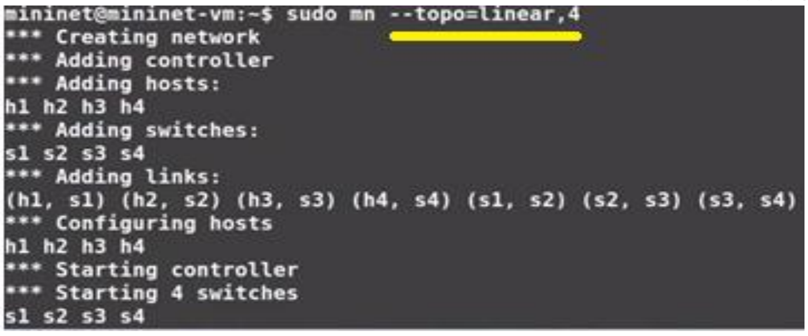
2) *Single:* It contains 1 switch and having n number of hosts (Fig 11). Topology creation through command line is as shown in Fig 7.



```
mininet@mininet-vm:~$ sudo mn --topo=single,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

Fig 7 Creating single topology using Mininet

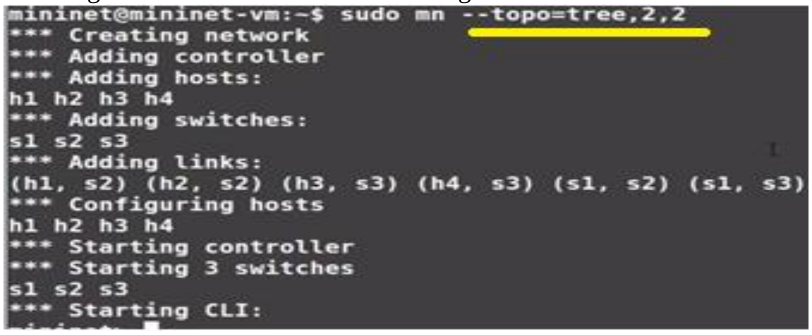
3) *Reversed:* It is similar to single topology but having connection in reverse order.
4) *Linear:* It contains n-number of switches having n-number of hosts linked to switches in linear order as shown in Fig 12. Topology creation through command line is as shown in Fig 8.



```
mininet@mininet-vm:~$ sudo mn --topo=linear,4
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (s1, s2) (s2, s3) (s3, s4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 4 switches
s1 s2 s3 s4
```

Fig 8 Creating linear topology using Mininet

5) *Tree:* It contains n-level of switches and hosts are attached to lower level switches as shown in the Fig 13. Topology creation through command line is as shown in Fig 9.



```
mininet@mininet-vm:~$ sudo mn --topo=tree,2,2
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s2) (h2, s2) (h3, s3) (h4, s3) (s1, s2) (s1, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 3 switches
s1 s2 s3
*** Starting CLI:
```

Fig 9 Creating tree topology using Mininet

B. *Custom Topologies*

Custom topologies [1] in Mininet can be created by using Python code. These topologies are run in Mininet by using commands. Python API [5] use its own classes, methods, functions and variables to create these topologies.

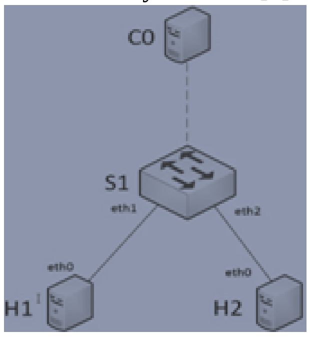


Fig 10. Minimal Topology

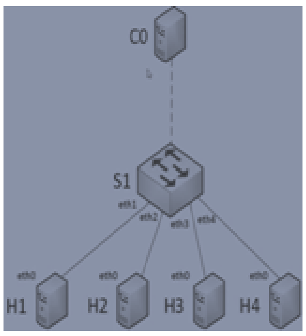


Fig 11. Single Topology

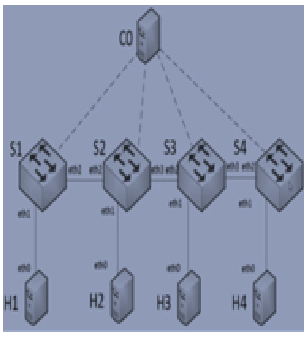


Fig 12. Linear Topology

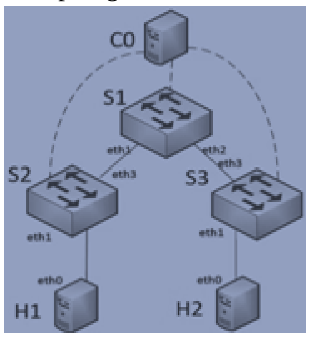
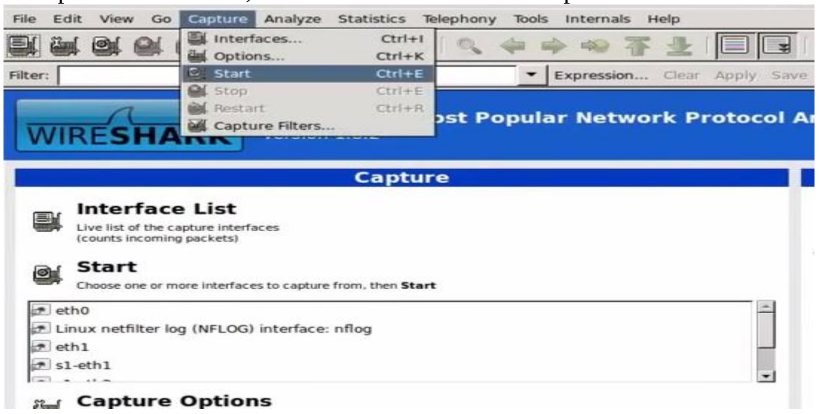


Fig 13. Tree Topology

V. CAPTURING MININET TRAFFIC USING WIRESHARK

To monitor the communication between the controller and any switches in the simulated network, we start Wireshark and capture traffic on Mininet VM's loopback interface. Wireshark must be started with superuser privileges so on the Mininet VM terminal.

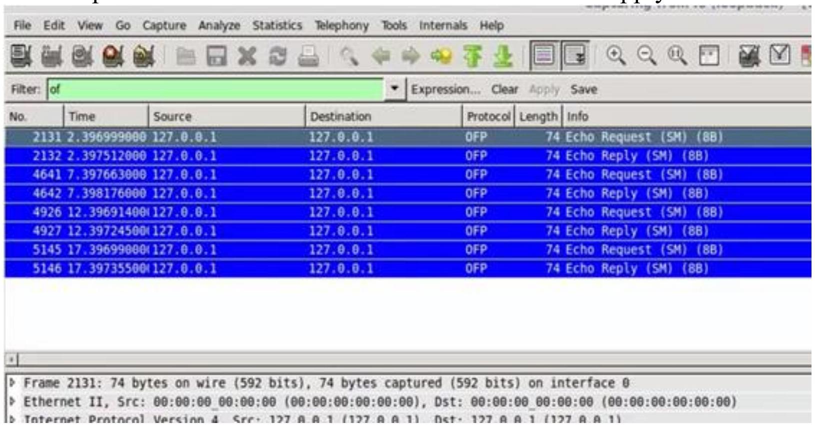
We set Wireshark [9] to capture packet on the Mininet VM's loopback interface. In Wireshark program, click on the menu command and then capture → interface, then select the interface and press the start button as shown in Fig 14.



File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Interfaces... Ctrl+I
Options... Ctrl+K
Start Ctrl+E
Stop Ctrl+E
Restart Ctrl+R
Capture Filters...
Filter:
Expression... Clear Apply Save
WIRESHARK
ost Popular Network Protocol A
Capture
Interface List
Live list of the capture interfaces (counts incoming packets)
Start
Choose one or more interfaces to capture from, then Start
eth0
Linux netfilter log (NFLOG) interface: nflog
eth1
s1-eth1
Capture Options

Fig 14 Capturing interface using Wireshark

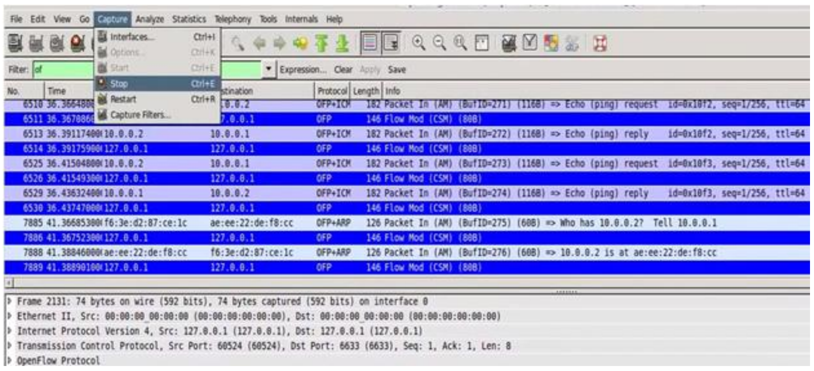
Since many processes are passing information through the loopback interface, we can apply filter on the Open-Flow message by entering the 'of' expression into the Wireshark filter box and click Apply as shown in Fig 15.


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: of
Expression... Clear Apply Save

```
No.   Time        Source     Destination  Protocol Length Info
 2131 2.396999000 127.0.0.1  127.0.0.1    OFP          74 Echo Request (SM) (8B)
 2132 2.397512000 127.0.0.1  127.0.0.1    OFP          74 Echo Reply (SM) (8B)
 4641 7.397663000 127.0.0.1  127.0.0.1    OFP          74 Echo Request (SM) (8B)
 4642 7.398176000 127.0.0.1  127.0.0.1    OFP          74 Echo Reply (SM) (8B)
 4926 12.39691400 127.0.0.1  127.0.0.1    OFP          74 Echo Request (SM) (8B)
 4927 12.39724500 127.0.0.1  127.0.0.1    OFP          74 Echo Reply (SM) (8B)
 5145 17.39699000 127.0.0.1  127.0.0.1    OFP          74 Echo Request (SM) (8B)
 5146 17.39735500 127.0.0.1  127.0.0.1    OFP          74 Echo Reply (SM) (8B)

▷ Frame 2131: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▷ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▷ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
```

Fig 15 Applying filter on Open-Flow messages

Now we see the Wireshark window open and monitoring for Open-Flow messages between the Controller and the Switch (Fig 16).


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Interfaces... Ctrl+I
Options... Ctrl+K
Start Ctrl+E
Stop Ctrl+E
Restart Ctrl+R
Capture Filters...
Filter: of
Expression... Clear Apply Save

```
No.  Time        Source             Destination        Protocol Length Info
6510 36.3664800                     0.0.2              OFP+ICM  182 Packet In (AM) (BufID=271) (116B) => Echo (ping) request  id=0x10f2, seq=1/256, ttl=64
6511 36.3678860                     7.0.0.1            OFP      146 Flow Mod (CSM) (80B)
6513 36.39117400 10.0.0.2           10.0.0.1           OFP+ICM  182 Packet In (AM) (BufID=272) (116B) => Echo (ping) reply    id=0x10f2, seq=1/256, ttl=64
6514 36.39175900 127.0.0.1          127.0.0.1          OFP      146 Flow Mod (CSM) (80B)
6525 36.41504800 10.0.0.2           10.0.0.1           OFP+ICM  182 Packet In (AM) (BufID=273) (116B) => Echo (ping) request  id=0x10f3, seq=1/256, ttl=64
6526 36.41549300 127.0.0.1          127.0.0.1          OFP      146 Flow Mod (CSM) (80B)
6529 36.43632400 10.0.0.1           10.0.0.2           OFP+ICM  182 Packet In (AM) (BufID=274) (116B) => Echo (ping) reply    id=0x10f3, seq=1/256, ttl=64
6530 36.43747000 127.0.0.1          127.0.0.1          OFP      146 Flow Mod (CSM) (80B)
7885 41.36685300 f6:3e:d2:87:ce:1c  ae:ee:22:de:f8:cc  OFP+ARP  126 Packet In (AM) (BufID=275) (60B) => Who has 10.0.0.2?  Tell 10.0.0.1
7886 41.36752300 127.0.0.1          127.0.0.1          OFP      146 Flow Mod (CSM) (80B)
7888 41.38846000 ae:ee:22:de:f8:cc  f6:3e:d2:87:ce:1c  OFP+ARP  126 Packet In (AM) (BufID=276) (60B) => 10.0.0.2 is at ae:ee:22:de:f8:cc
7889 41.38890100 127.0.0.1          127.0.0.1          OFP      146 Flow Mod (CSM) (80B)

▷ Frame 2131: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▷ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▷ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▷ Transmission Control Protocol, Src Port: 60524 (60524), Dst Port: 6633 (6633), Seq: 1, Ack: 1, Len: 8
▷ OpenFlow Protocol
```

Fig 16 Wireshark window monitoring for Open-Flow messages between the Controller and the Switch

VI. CONCLUSION

Mininet is open source Network emulation software whose switches are programmed using the Open-Flow protocol. Software Defined Network design that run on Mininet can easily be transferred to hardware Open-Flow Switches for line rate packet forwarding. The program that run on Mininet can send packet with a given link speed and delay through an interface which look like a real Ethernet interface. Packets get processed by an interface which looks similar to actual Ethernet switch, router or middlebox. Mininet's virtual hosts, switches, links and controllers are programmable items which look like real hardware items. Major limitation of Mininet is that it can emulate networks using slower links and it is not efficient for high speed links, it is because packets are forwarded by a collection of software switches that share CPU and memory resources and usually have lower performance than dedicated switching hardware. Using Mininet, it is also difficult to emulate the networks that are spread over a large scale.

REFERENCES

[1] Introduction to Mininet at https://github.com/ mininet/mininet.wiki.git/

[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.12.

[4] A Practical Experience in Designing an OpenFlow Controller by: R.Bifulco, R.Canonico, M. Brunner, P.Hasselmeyer, F. Mir in Software Defined Networking (EWSDN), 2012 European Workshop on (October 2012), pp. 61 – 66.

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

[6] OpenFlow: Enabling innovation in Campus Networks – McKeown, Anderson, et al. -2008.

[7] Mininet Commands at http://mininet.org/

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

[9] Wireshark at https://www.wireshark.org/