**CELL iteration 3 10/01/16: Task 3.3, Elective 2, Python NetApps; Dottie Kessler 10/2/16**

1. Start VM, login (mininet/mininet) and obtain ip (**sudo dhclient eth1**)
2. Run Xming: then xterm from within mininet
3. Putty to VM; run wireshark then filter of ; capture->interfaces->loopback (**sudo wireshare &**)
4. Putty to VM run ODL (**cd ~/opendaylight; sudo ./run.sh -virt ovsdb**)
5. Browser to host open IP of VM (admin/admin) [**http://192.168.56.101:8080**](http://192.168.56.101:8080)

**Task 3.3. Hand on Training with Python Net Apps elective 2 NORTHBOUND**

Select 5 examples for Python NetApps

**Python Net Apps (executing the scripts)**

**sudo ~/foldername/**[**filename.py**](http://filename.py/)

This is the default command assuming you have configured controller in the python file. If not you can specify command to connect to the remote controller in the mininet CLI itself.

Descriptions of the Examples:

<https://github.com/mininet/mininet/blob/master/examples/README.md>

1. controllers.py: Create a network where different switches are connected to different controllers by creating a custom Switch() subclass.

* **Explain the app at a high level**

Stores and starts three controllers, starts the cli, then stops the controllers, it creates a custom switch class and passes it to the Mininet() constructor

(2 controllers c0, c1) C0 port 6633; C1 port 6634; C2 remote localhost 127.0.0.4

3 switches s1 s2 s3

4 hosts h1 h2 h3 h4

* **Explain the output**

mininet@mininet-vm:~/mininet/examples$ **sudo ./controllers.py**

Unable to contact the remote controller at 127.0.0.1:6633

\*\*\* Creating network

\*\*\* Adding hosts:

h1 h2 h3 h4

\*\*\* Adding switches:

s1 s2 s3

\*\*\* Adding links:

(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)

\*\*\* Configuring hosts

h1 h2 h3 h4

\*\*\* Starting controller

c0 c1

\*\*\* Starting 3 switches

s1 s2 s3 ...

mininet> **net**

h1 h1-eth0:s2-eth1

h2 h2-eth0:s2-eth2

h3 h3-eth0:s3-eth1

h4 h4-eth0:s3-eth2

s1 lo: s1-eth1:s2-eth3 s1-eth2:s3-eth3

s2 lo: s2-eth1:h1-eth0 s2-eth2:h2-eth0 s2-eth3:s1-eth1

s3 lo: s3-eth1:h3-eth0 s3-eth2:h4-eth0 s3-eth3:s1-eth2

c0

c1

1. cpu.py: Test iperf bandwidth for varying cpu limits

* **Explain the app at a high level**

For a set of 5 cpu limit values [ .45, .4, .3, .2, .1 ], in a loop, create a custom host then start the mininet network. Pingall the hosts, test the network bandwidth. It runs an iperf TCP server on one hosts and an iperf client on another and parses the bandwidth achieved. The default topology is a single switch connected to 2 hosts. Testing the speed of the connection.

* **Explain the output**

mininet@mininet-vm**:~/mininet/examples$ sudo ./cpu.py**

\*\*\* Testing with rt bandwidth limiting

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

\*\*\* error: please enable RT\_GROUP\_SCHED in your kernel

\*\*\* Skipping host rt

\*\*\* Testing with cfs bandwidth limiting

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(s1, h1) (s1, h2)

\*\*\* Configuring hosts

h1 (cfs 45000/100000us) h2 (cfs 45000/100000us)

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Ping: testing ping reachability

h1 -> h2

h2 -> h1

\*\*\* Results: 0% dropped (2/2 received)

.\*\*\* h2 : ('kill %iperf',)

------------------------------------------------------------

Server listening on TCP port 5001

TCP window size: 85.3 KByte (default)

------------------------------------------------------------

[ 4] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 52847

[ ID] Interval Transfer Bandwidth

[ 4] 0.0- 0.0 sec 0.00 Bytes 0.00 bits/sec

[ 5] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 52848

[ 5] 0.0- 5.0 sec 10.5 GBytes 18.0 Gbits/sec

\*\*\* Stopping 1 controllers

c0

\*\*\* Stopping 2 links

..

\*\*\* Stopping 1 switches

s1

\*\*\* Stopping 2 hosts

h1 h2

\*\*\* Done

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(s1, h1) (s1, h2)

\*\*\* Configuring hosts

h1 (cfs 40000/100000us) h2 (cfs 40000/100000us)

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Ping: testing ping reachability

h1 -> h2

h2 -> h1

\*\*\* Results: 0% dropped (2/2 received)

\*\*\* h2 : ('kill %iperf',)

[ 4] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 52851

[ ID] Interval Transfer Bandwidth

[ 4] 0.0- 0.0 sec 0.00 Bytes 0.00 bits/sec

[ 5] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 52852

[ 5] 0.0- 5.0 sec 8.56 GBytes 14.6 Gbits/sec

\*\*\* Stopping 1 controllers

c0

\*\*\* Stopping 2 links

..

\*\*\* Stopping 1 switches

s1

\*\*\* Stopping 2 hosts

h1 h2

\*\*\* Done

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(s1, h1) (s1, h2)

\*\*\* Configuring hosts

h1 (cfs 30000/100000us) h2 (cfs 30000/100000us)

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Ping: testing ping reachability

h1 -> h2

h2 -> h1

\*\*\* Results: 0% dropped (2/2 received)

\*\*\* h2 : ('kill %iperf',)

[ 4] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 52855

[ ID] Interval Transfer Bandwidth

[ 4] 0.0- 0.0 sec 0.00 Bytes 0.00 bits/sec

[ 5] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 52856

[ 5] 0.0- 5.0 sec 4.79 GBytes 8.23 Gbits/sec

\*\*\* Stopping 1 controllers

c0

\*\*\* Stopping 2 links

..

\*\*\* Stopping 1 switches

s1

\*\*\* Stopping 2 hosts

h1 h2

\*\*\* Done

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(s1, h1) (s1, h2)

\*\*\* Configuring hosts

h1 (cfs 20000/100000us) h2 (cfs 20000/100000us)

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Ping: testing ping reachability

h1 -> h2

h2 -> h1

\*\*\* Results: 0% dropped (2/2 received)

\*\*\* h2 : ('kill %iperf',)

[ 4] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 52859

[ ID] Interval Transfer Bandwidth

[ 4] 0.0- 0.0 sec 0.00 Bytes 0.00 bits/sec

[ 5] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 52860

[ 5] 0.0- 5.1 sec 2.30 GBytes 3.90 Gbits/sec

\*\*\* Stopping 1 controllers

c0

\*\*\* Stopping 2 links

..

\*\*\* Stopping 1 switches

s1

\*\*\* Stopping 2 hosts

h1 h2

\*\*\* Done

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(s1, h1) (s1, h2)

\*\*\* Configuring hosts

h1 (cfs 10000/100000us) h2 (cfs 10000/100000us)

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Ping: testing ping reachability

h1 -> h2

h2 -> h1

\*\*\* Results: 0% dropped (2/2 received)

\*\*\* h2 : ('kill %iperf',)

[ 4] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 52863

[ ID] Interval Transfer Bandwidth

[ 4] 0.0- 0.0 sec 0.00 Bytes 0.00 bits/sec

[ 5] local 10.0.0.2 port 5001 connected with 10.0.0.1 port 52864

[ 5] 0.0- 5.0 sec 757 MBytes 1.26 Gbits/sec

\*\*\* Stopping 1 controllers

c0

\*\*\* Stopping 2 links

..

\*\*\* Stopping 1 switches

s1

\*\*\* Stopping 2 hosts

h1 h2

\*\*\* Done

sched cpu client MB/s

**cfs 45.00% 18060.587182**

**cfs 40.00% 14701.18643**

**cfs 30.00% 8235.269953**

**cfs 20.00% 3905.899649**

**cfs 10.00% 1259.25502**

1. **emptynet,py**: Shows how to create an empty Mininet object (without a topology object) and add nodes to it manually.

* **Explain the app at a high level**

Creates an empty network, adds controller c0, and hosts h1 h2. It adds a switch s3, and creates the links then starts the network. Then it runs the CLI and then stops the network

* **Explain the output**

mininet@mininet-vm:~/mininet/examples$ **sudo ./emptynet.py**

\*\*\* Adding controller

\*\*\* Adding hosts

\*\*\* Adding switch

\*\*\* Creating links

\*\*\* Starting network

\*\*\* Configuring hosts

h1 h2

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s3 ...

\*\*\* Running CLI

\*\*\* Starting CLI:

mininet> net

h1 h1-eth0:s3-eth1

h2 h2-eth0:s3-eth2

s3 lo: s3-eth1:h1-eth0 s3-eth2:h2-eth0

c0

mininet> dump

<Host h1: h1-eth0:10.0.0.1 pid=3765>

<Host h2: h2-eth0:10.0.0.2 pid=3768>

<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=3773>

<Controller c0: 127.0.0.1:6633 pid=3758>

mininet> pingall

\*\*\* Ping: testing ping reachability

h1 -> h2

h2 -> h1

\*\*\* Results: 0% dropped (2/2 received)

1. **miniedit.py**: A simple network editor for Mininet.

* **Explain the app at a high level**

GUI editor for designing topology for Mininet. We have the mininet VM started, connected to it via SSH and enabled X forwarding.

<http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/>

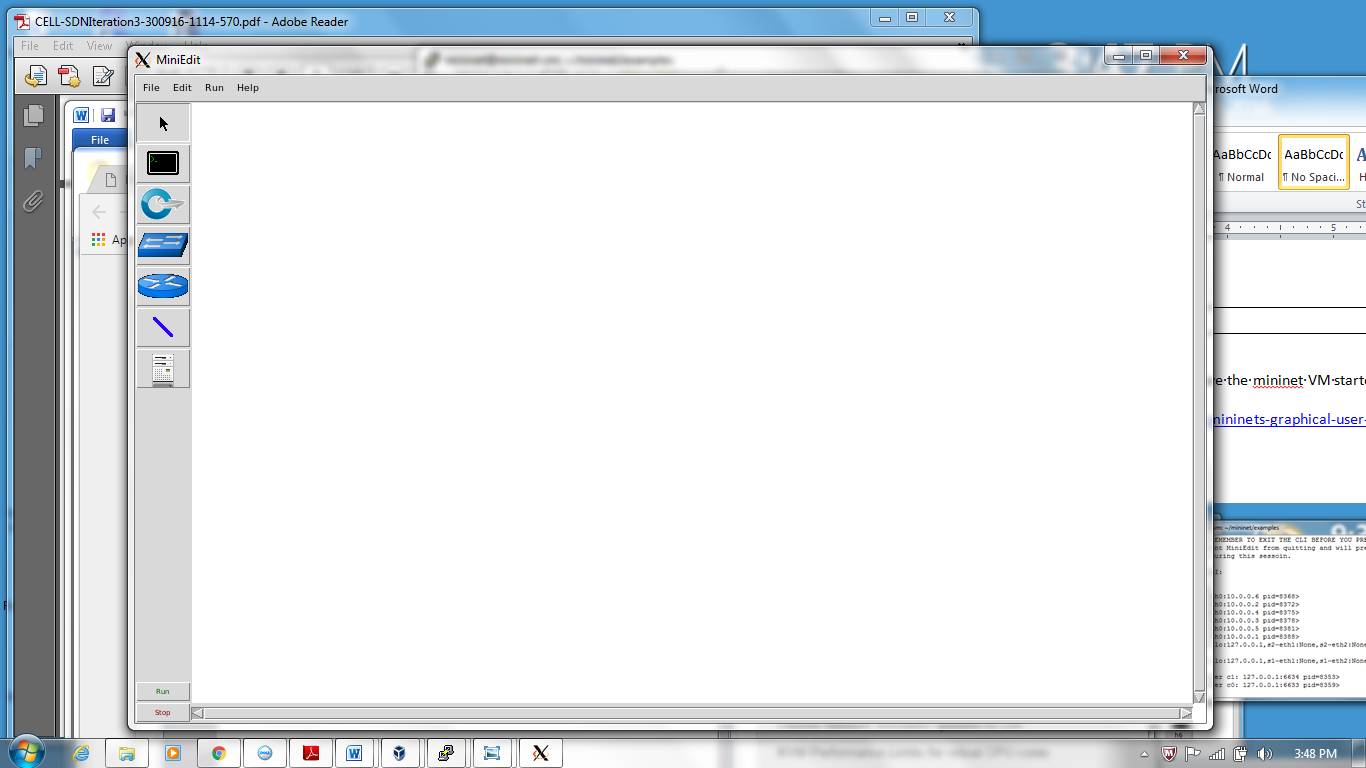
* **Explain the output**

mininet@mininet-vm:~/mininet/examples$ **sudo ./miniedit.py**

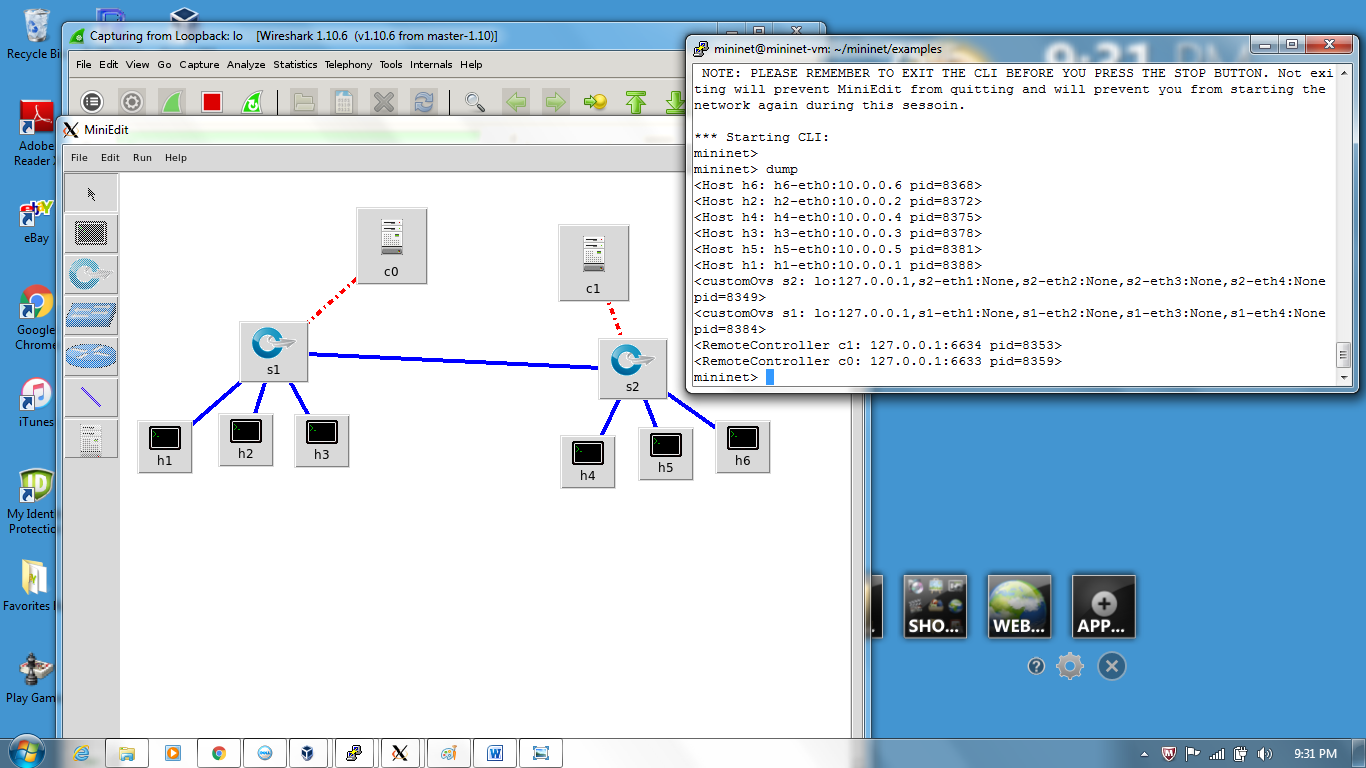
MiniEdit running against Mininet 2.2.1

topo=none

Runs as root, gives a blank canvas



You use the gui and menu options to create a topology, then configure the controllers and set the preferences, then start the cli. You can save the topology, a .mn file; and you can save the custom topology python .py file.



1. **sshd.py**: Create a network and start sshd on each host.

* **Explain the app at a high level**

Run an SSH daemon on every host, so from another terminal you can ssh into any host to run interactive commands. This connects the mininet data network to an interface in the root namespace.

* **Explain the output**

mininet@mininet-vm:~/mininet/examples$ **sudo ./sshd.py**

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2 h3 h4

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(s1, h1) (s1, h2) (s1, h3) (s1, h4)

\*\*\* Configuring hosts

h1 h2 h3 h4

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Waiting for ssh daemons to start

.

\*\*\* Hosts are running sshd at the following addresses:

h1 10.0.0.1

h2 10.0.0.2

h3 10.0.0.3

h4 10.0.0.4

\*\*\* Type 'exit' or control-D to shut down network

\*\*\* Starting CLI:

mininet>

mininet@mininet-vm:~/mininet/examples$ **ssh 10.0.0.1**

mininet@10.0.0.1's password:

Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86\_64)

\* Documentation: https://help.ubuntu.com/

New release '16.04.1 LTS' available.

Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Oct 3 13:02:48 2016 from 10.123.123.1

mininet@mininet-vm:~$ **ifconfig**

h1-eth0 Link encap:Ethernet HWaddr 06:79:6b:d1:df:e1

inet addr:10.0.0.1 Bcast:10.255.255.255 Mask:255.0.0.0

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:174 errors:0 dropped:0 overruns:0 frame:0

TX packets:134 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:18974 (18.9 KB) TX bytes:19107 (19.1 KB)

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

UP LOOPBACK RUNNING MTU:65536 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

mininet@mininet-vm:~$ **ping 10.0.0.3**

PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

64 bytes from 10.0.0.3: icmp\_seq=1 ttl=64 time=7.25 ms

64 bytes from 10.0.0.3: icmp\_seq=2 ttl=64 time=0.555 ms

64 bytes from 10.0.0.3: icmp\_seq=3 ttl=64 time=0.415 ms

^C

--- 10.0.0.3 ping statistics ---

3 packets transmitted, 3 received, 0% packet loss, time 2000ms

rtt min/avg/max/mdev = 0.415/2.741/7.255/3.192 ms

1. **tree1024.py**: Create a 1024-host network, and run the CLI on it.

* **Explain the app at a high level**

Depth 2, fanout 32 (1024 hosts) Creates a tree network. This did not run into scalability limits on my VM.

* **Explain the output**

mininet@mininet-vm:~/mininet/examples$ **sudo ./tree1024.py**

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64 h65 h66 h67 h68 h69 h70 h71 h72 h73 h74 h75 h76 h77 h78 h79 h80 h81 h82 h83 h84 h85 h86 h87 h88 h89 h90 h91 h92 h93 h94 h95 h96 h97 h98 h99 h100 (truncated)

h999 h1000 h1001 h1002 h1003 h1004 h1005 h1006 h1007 h1008 h1009 h1010 h1011 h1012 h1013 h1014 h1015 h1016 h1017 h1018 h1019 h1020 h1021 h1022 h1023 h1024

\*\*\* Adding switches:

s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19 s20 s21 s22 s23 s24 s25 s26 s27 s28 s29 s30 s31 s32 s33

\*\*\* Adding links:

(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s1, s10) (s1, s11) (s1, s12) (s1, s13) (s1, s14) (s1, s15) (s1, s16) (s1, s17) (s1, s1

(truncated)

mininet> **h951 ping -c2 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=76.4 ms

64 bytes from 10.0.0.2: icmp\_seq=2 ttl=64 time=2.20 ms

--- 10.0.0.2 ping statistics ---

2 packets transmitted, 2 received, 0% packet loss, time 1001ms

rtt min/avg/max/mdev = 2.202/39.332/76.462/37.130 ms

mininet> **h888 ifconfig**

h888-eth0 Link encap:Ethernet HWaddr 82:04:35:65:31:58

inet addr:10.0.3.120 Bcast:10.255.255.255 Mask:255.0.0.0

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:1 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:42 (42.0 B) TX bytes:0 (0.0 B)

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

UP LOOPBACK RUNNING MTU:65536 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

mininet> **h47 date**

Mon Oct 3 13:35:51 PDT 2016

1. **treeping64.py**: Create a 64-node tree network, and test connectivity using ping..

* **Explain the app at a high level**

This example creates a 64-host tree network, and attempts to check full connectivity using ping, for different switch/datapath types.

* **Explain the output**

mininet@mininet-vm:~/mininet/examples$ **sudo ./treeping64.py**

\*\*\* Testing reference user datapath

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64

\*\*\* Adding switches:

s1 s2 s3 s4 s5 s6 s7 s8 s9

\*\*\* Adding links:

(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s2, h1) (s2, h2) (s2, h3) (s2, h4) (s2, h5) (s2, h6) (s2, h7) (s2, h8) (s3, h9) (s3, h10) (s3, h11) (s3, h12) (s3, h13) (s3, h14) (s3, h15) (s3, h16) (s4, h17) (s4, h18) (s4, h19) (s4, h20) (s4, h21) (s4, h22) (s4, h23) (s4, h24) (s5, h25) (s5, h26) (s5, h27) (s5, h28) (s5, h29) (s5, h30) (s5, h31) (s5, h32) (s6, h33) (s6, h34) (s6, h35) (s6, h36) (s6, h37) (s6, h38) (s6, h39) (s6, h40) (s7, h41) (s7, h42) (s7, h43) (s7, h44) (s7, h45) (s7, h46) (s7, h47) (s7, h48) (s8, h49) (s8, h50) (s8, h51) (s8, h52) (s8, h53) (s8, h54) (s8, h55) (s8, h56) (s9, h57) (s9, h58) (s9, h59) (s9, h60) (s9, h61) (s9, h62) (s9, h63) (s9, h64)

\*\*\* Configuring hosts

h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64

\*\*\* Starting controller

c0

\*\*\* Starting 9 switches

s1 s2 s3 s4 s5 s6 s7 s8 s9

\*\*\* Running test

(truncated)

h64 -> h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63

\*\*\* Results: 0% dropped (4032/4032 received)

\*\*\* Stopping 1 controllers

c0

\*\*\* Stopping 72 links

........................................................................

\*\*\* Stopping 9 switches

s1 s2 s3 s4 s5 s6 s7 s8 s9

\*\*\* Stopping 64 hosts

h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64

\*\*\* Done

\*\*\* Tree network ping results:

reference user: 0% packet loss

Open vSwitch kernel: 0% packet loss

1. **limit.py**: Example of using link and CPU limits.

* **Explain the app at a high level**

Uses links and CPU limits, tests the link limit and the CPU bandwidth limits

* **Explain the output**

mininet@mininet-vm:~/mininet/examples$ **sudo ./limit.py**

\*\*\* Testing with rt bandwidth limiting

\*\*\* RT Scheduler is not enabled in your kernel. Skipping this test

\*\*\* Testing with cfs bandwidth limiting

\*\*\* Creating network

\*\*\* Adding controller

\*\*\* Adding hosts:

h1 h2

\*\*\* Adding switches:

s1

\*\*\* Adding links:

(s1, h1) (s1, h2)

\*\*\* Configuring hosts

h1 (cfs 10000/100000us) h2 (cfs 10000/100000us)

\*\*\* Starting controller

c0

\*\*\* Starting 1 switches

s1 ...

\*\*\* Testing network 10.00 Mbps bandwidth limit

\*\*\* Iperf: testing TCP bandwidth between h1 and h2

.\*\*\* Results: ['1.35 Gbits/sec', '1.37 Gbits/sec']

\*\*\* Testing CPU 10% bandwidth limit

\*\*\* Results: [9.6807715, 10.324229, 10.2828921, 9.9326819, 9.928390199999999, 10.3440456, 10.2807039, 9.8478066, 10.1424592, 9.8174881]

\*\*\* Stopping 1 controllers

c0

\*\*\* Stopping 2 links

..

\*\*\* Stopping 1 switches

s1

\*\*\* Stopping 2 hosts

h1 h2

\*\*\* Done