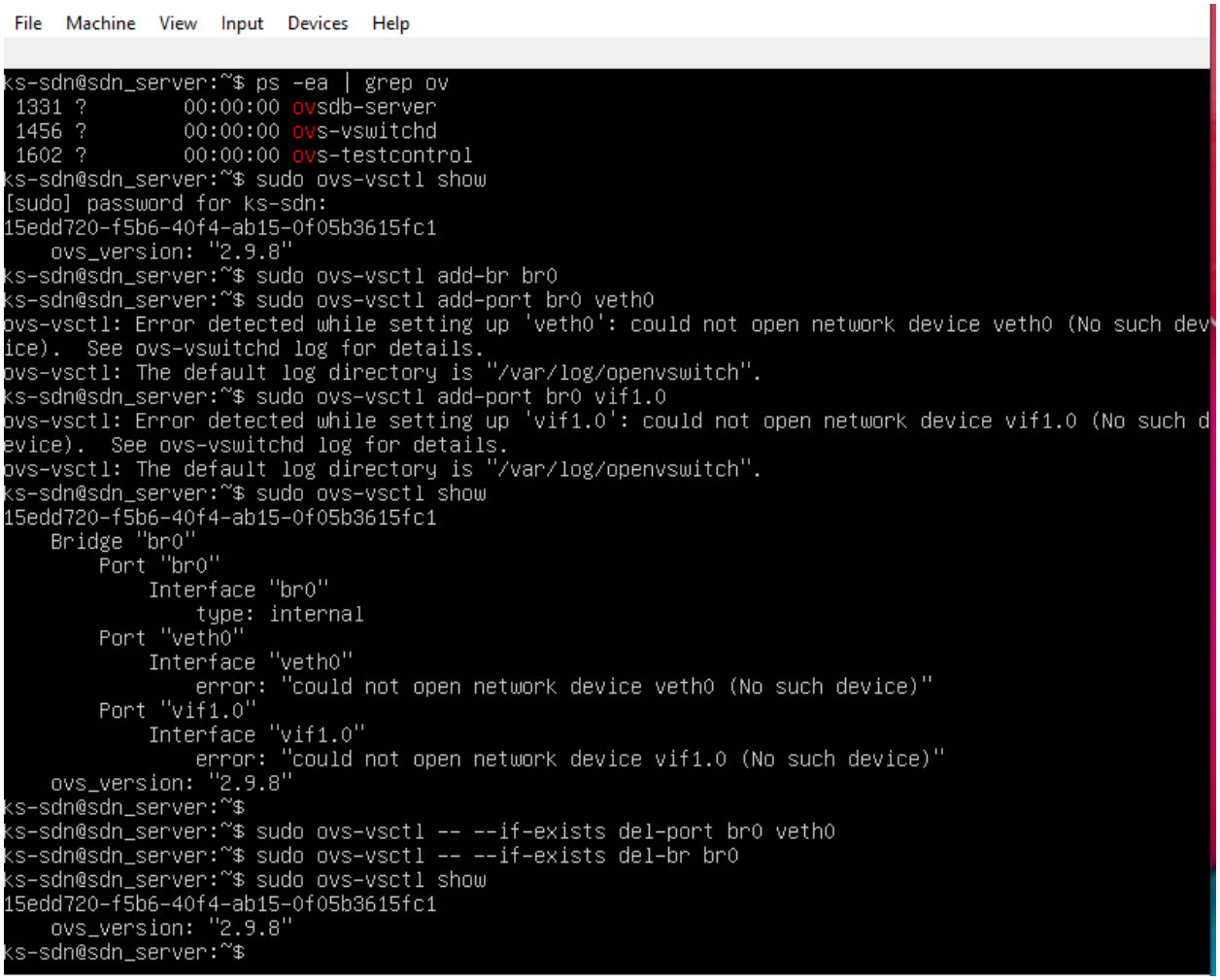
SDN DEMO PREP

Iteration 2

By

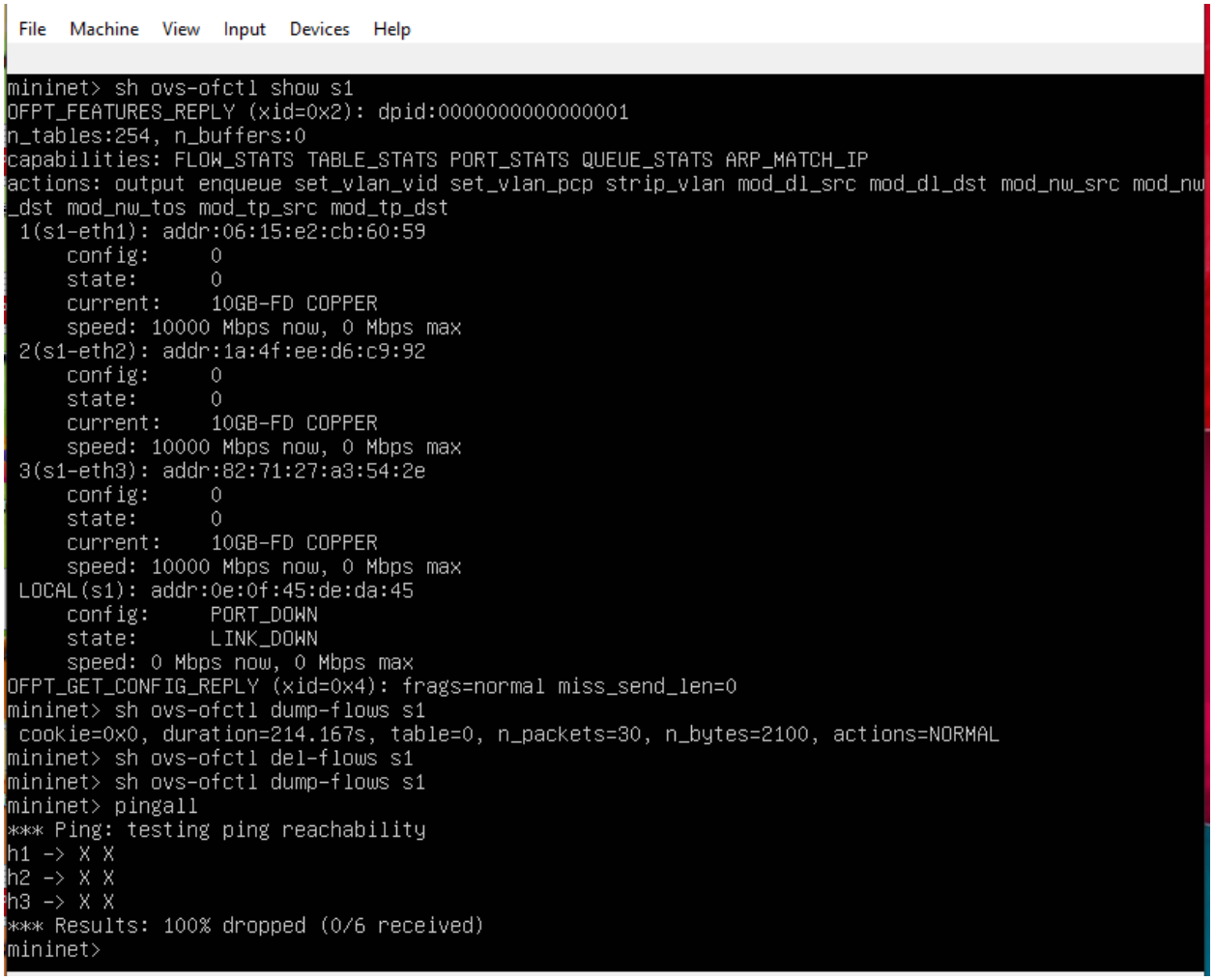
Kabiru Sanusi

Hands-on Training with Python Net Apps - Elective 1

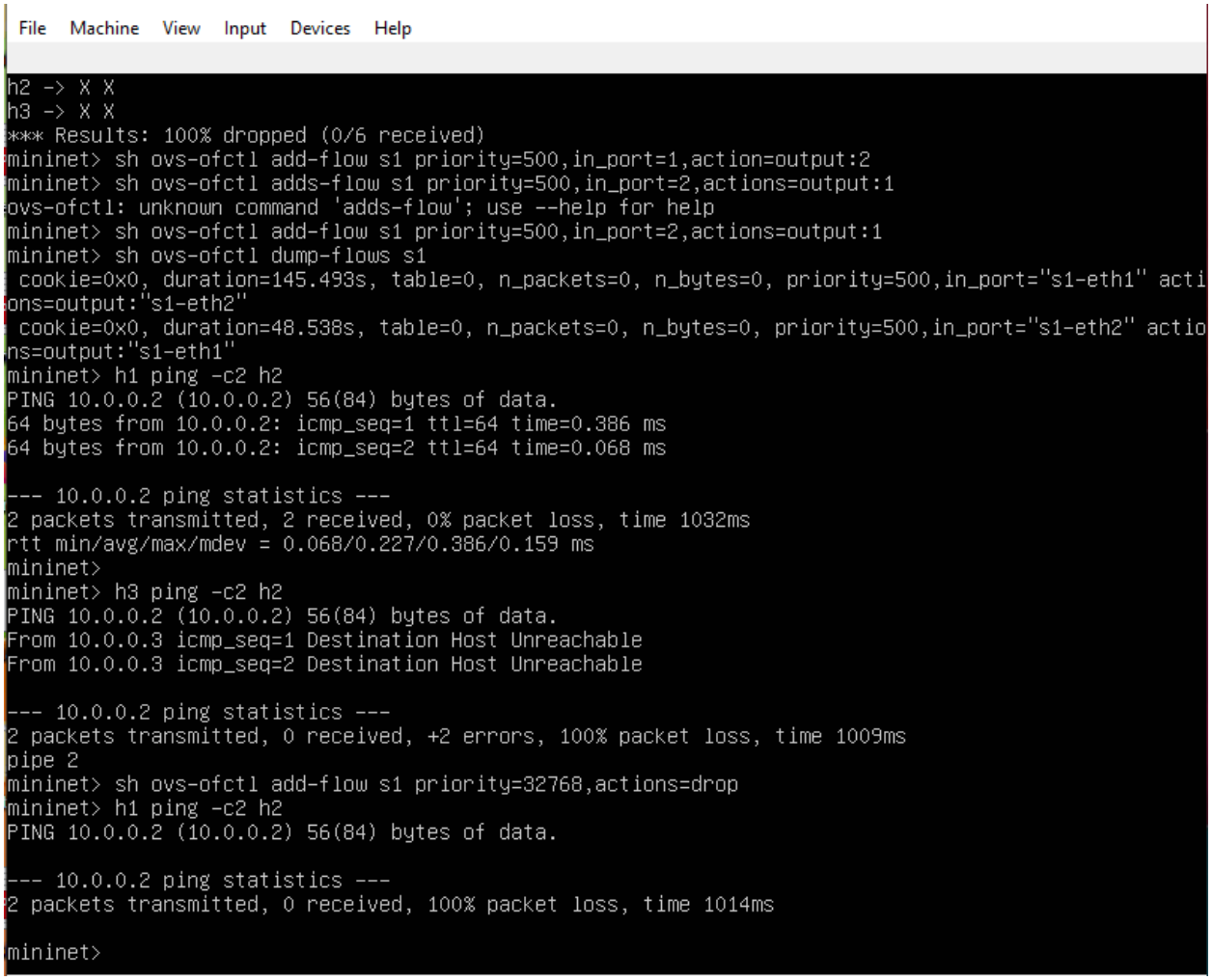


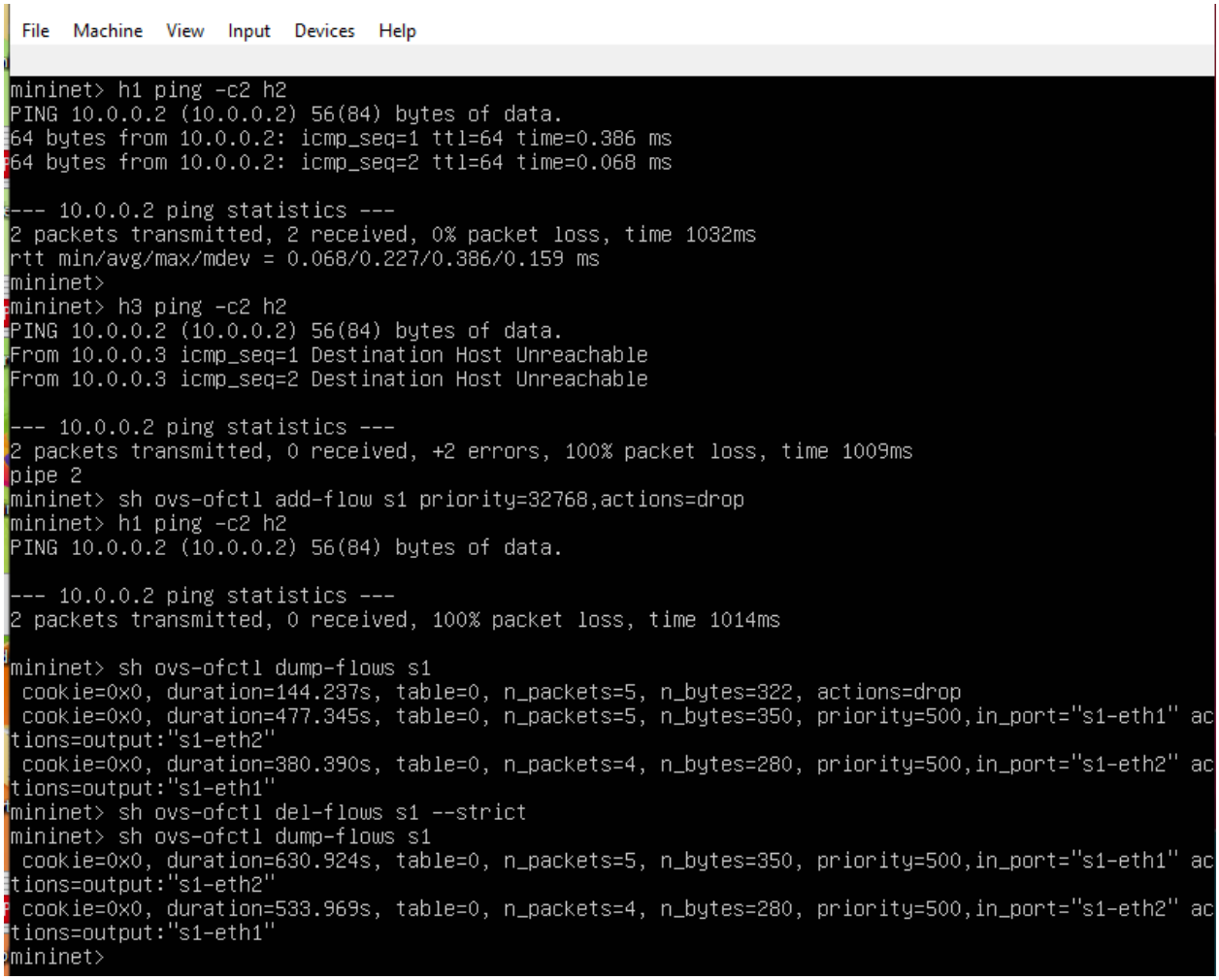
File Machine View Input Devices Help

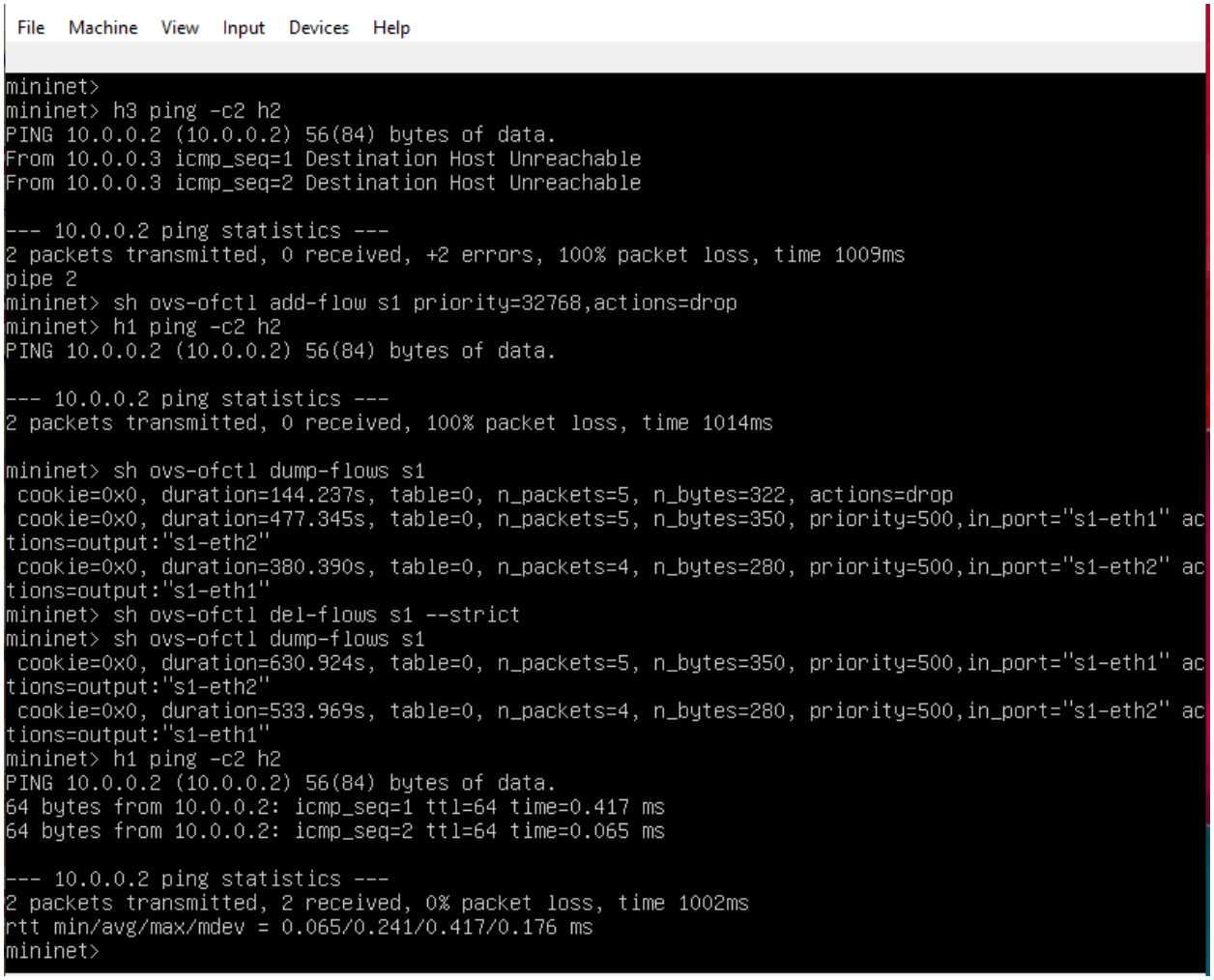
```
ks-sdn@sdn_server:~$ sudo mn --topo=single,3 --controller=none --mac
[sudo] password for ks-sdn:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=13374>
<Host h2: h2-eth0:10.0.0.2 pid=13376>
<Host h3: h3-eth0:10.0.0.3 pid=13378>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=13383>
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0
mininet> sh ovs-ofctl add-flow s1 action=normal
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```


File Machine View Input Devices Help
mininet> sh ovs-ofctl show s1
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst mod_nw_src mod_nw
_dst mod_nw_tos mod_tp_src mod_tp_dst
1(s1-eth1): addr:06:15:e2:cb:60:59
config: 0
state: 0
current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max
2(s1-eth2): addr:1a:4f:ee:d6:c9:92
config: 0
state: 0
current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max
3(s1-eth3): addr:82:71:27:a3:54:2e
config: 0
state: 0
current: 10GB-FD COPPER
speed: 10000 Mbps now, 0 Mbps max
LOCAL(s1): addr:0e:0f:45:de:da:45
config: PORT_DOWN
state: LINK_DOWN
speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
mininet> sh ovs-ofctl dump-flows s1
cookie=0x0, duration=214.167s, table=0, n_packets=30, n_bytes=2100, actions=NORMAL
mininet> sh ovs-ofctl del-flows s1
mininet> sh ovs-ofctl dump-flows s1
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet>

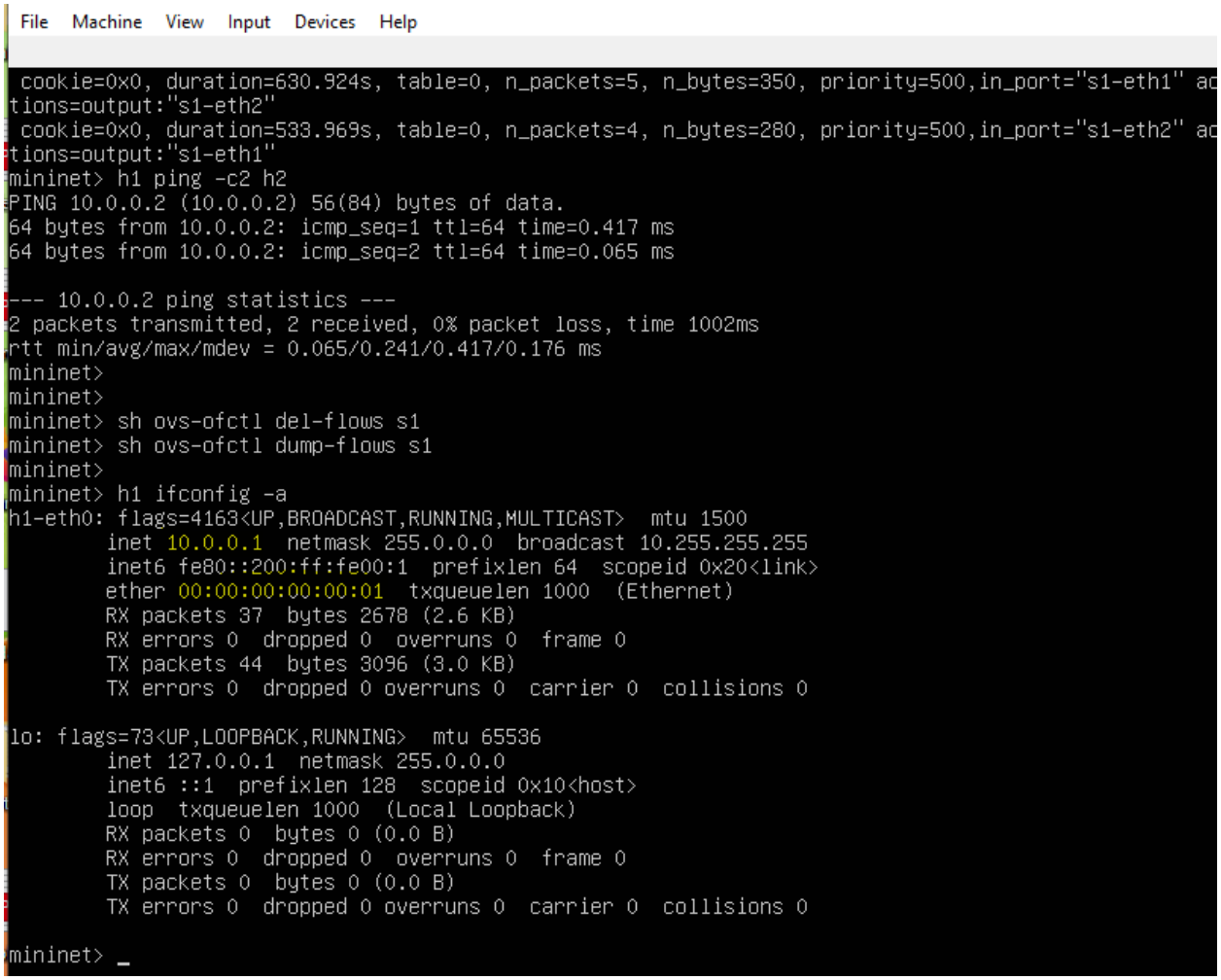
L1:

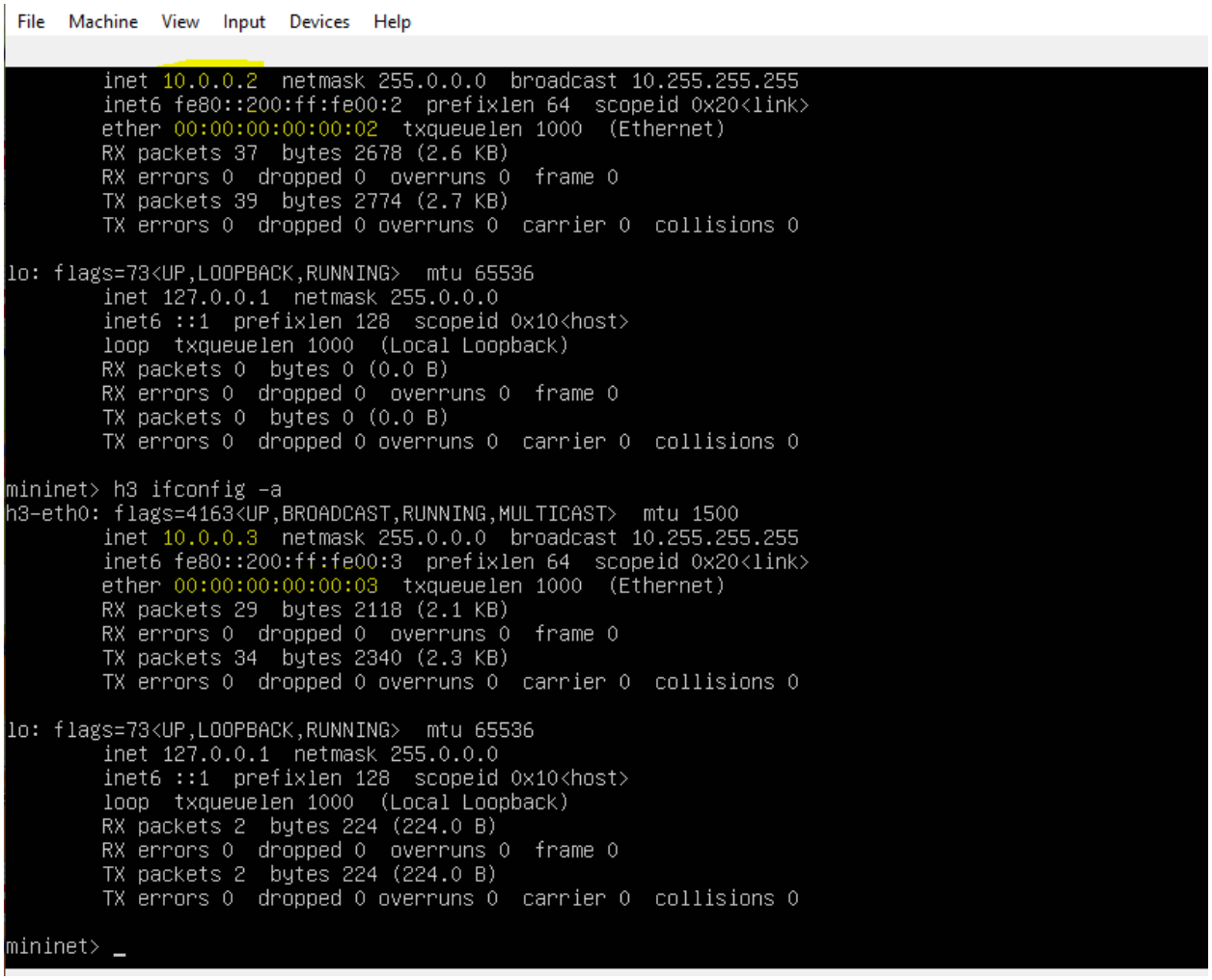

File Machine View Input Devices Help
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
mininet> sh ovs-ofctl add-flow s1 priority=500,in_port=1,action=output:2
mininet> sh ovs-ofctl adds-flow s1 priority=500,in_port=2,actions=output:1
ovs-ofctl: unknown command 'adds-flow'; use --help for help
mininet> sh ovs-ofctl add-flow s1 priority=500,in_port=2,actions=output:1
mininet> sh ovs-ofctl dump-flows s1
cookie=0x0, duration=145.493s, table=0, n_packets=0, n_bytes=0, priority=500,in_port="s1-eth1" acti
ons=output:"s1-eth2"
cookie=0x0, duration=48.538s, table=0, n_packets=0, n_bytes=0, priority=500,in_port="s1-eth2" actio
ns=output:"s1-eth1"
mininet> h1 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=0.386 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.068 ms
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1032ms
rtt min/avg/max/mdev = 0.068/0.227/0.386/0.159 ms
mininet>
mininet> h3 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.3 icmp_seq=1 Destination Host Unreachable
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1009ms
pipe 2
mininet> sh ovs-ofctl add-flow s1 priority=32768,actions=drop
mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1014ms
mininet>


File Machine View Input Devices Help
mininet> h1 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=0.386 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.068 ms
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1032ms
rtt min/avg/max/mdev = 0.068/0.227/0.386/0.159 ms
mininet>
mininet> h3 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.3 icmp_seq=1 Destination Host Unreachable
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1009ms
pipe 2
mininet> sh ovs-ofctl add-flow s1 priority=32768,actions=drop
mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1014ms
mininet> sh ovs-ofctl dump-flows s1
cookie=0x0, duration=144.237s, table=0, n_packets=5, n_bytes=322, actions=drop
cookie=0x0, duration=477.345s, table=0, n_packets=5, n_bytes=350, priority=500,in_port="s1-eth1" actions=output:"s1-eth2"
cookie=0x0, duration=380.390s, table=0, n_packets=4, n_bytes=280, priority=500,in_port="s1-eth2" actions=output:"s1-eth1"
mininet> sh ovs-ofctl del-flows s1 --strict
mininet> sh ovs-ofctl dump-flows s1
cookie=0x0, duration=630.924s, table=0, n_packets=5, n_bytes=350, priority=500,in_port="s1-eth1" actions=output:"s1-eth2"
cookie=0x0, duration=533.969s, table=0, n_packets=4, n_bytes=280, priority=500,in_port="s1-eth2" actions=output:"s1-eth1"
mininet>

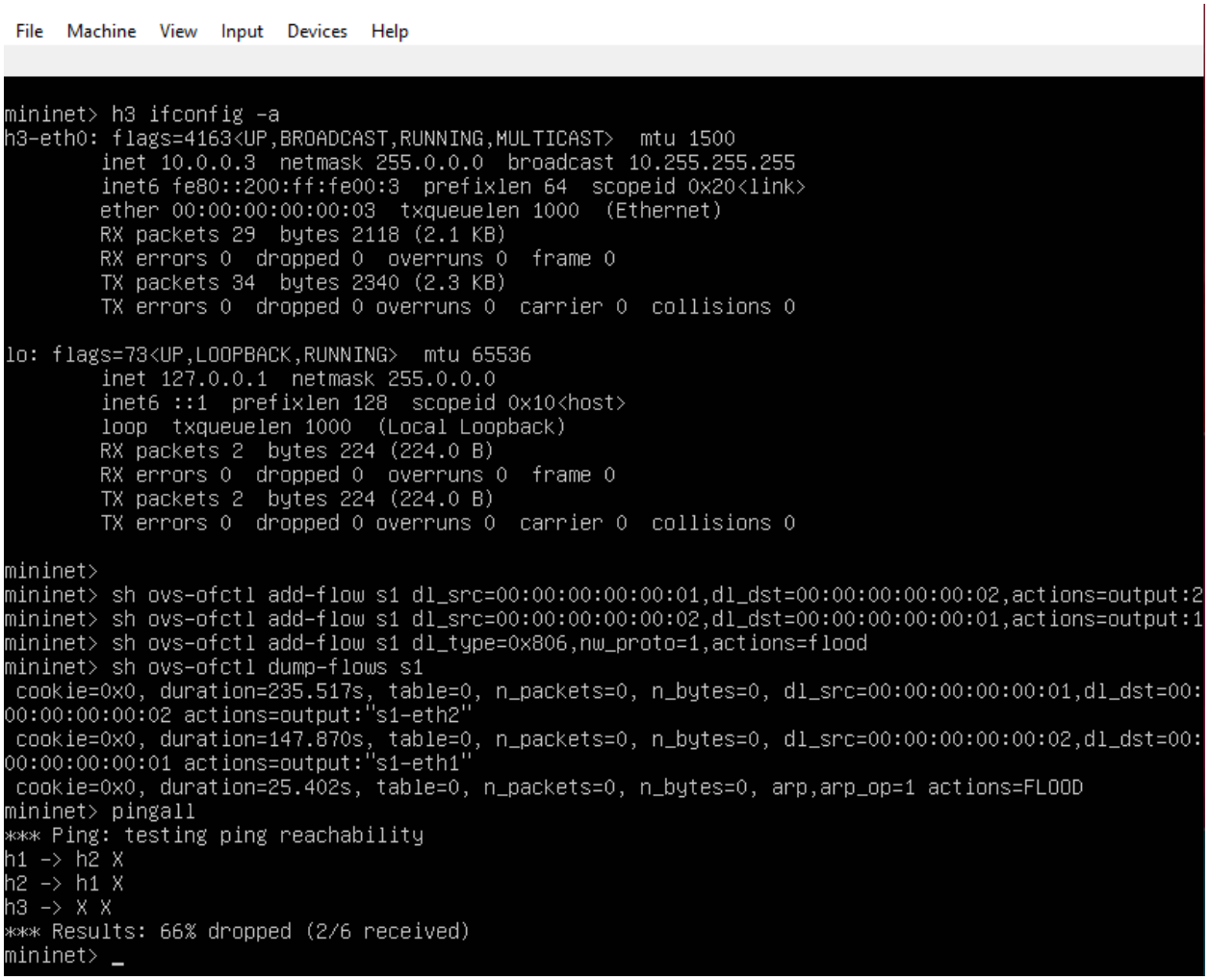

File Machine View Input Devices Help
mininet>
mininet> h3 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.3 icmp_seq=1 Destination Host Unreachable
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1009ms
pipe 2
mininet> sh ovs-ofctl add-flow s1 priority=32768,actions=drop
mininet> h1 ping -c2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1014ms
mininet> sh ovs-ofctl dump-flows s1
cookie=0x0, duration=144.237s, table=0, n_packets=5, n_bytes=322, actions=drop
cookie=0x0, duration=477.345s, table=0, n_packets=5, n_bytes=350, priority=500,in_port="s1-eth1" ac
tions=output:"s1-eth2"
cookie=0x0, duration=380.390s, table=0, n_packets=4, n_bytes=280, priority=500,in_port="s1-eth2" ac
tions=output:"s1-eth1"
mininet> sh ovs-ofctl del-flows s1 --strict
mininet> sh ovs-ofctl dump-flows s1
cookie=0x0, duration=630.924s, table=0, n_packets=5, n_bytes=350, priority=500,in_port="s1-eth1" ac
tions=output:"s1-eth2"
cookie=0x0, duration=533.969s, table=0, n_packets=4, n_bytes=280, priority=500,in_port="s1-eth2" ac
tions=output:"s1-eth1"
mininet> h1 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=0.417 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.065 ms
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.065/0.241/0.417/0.176 ms
mininet>

L2:

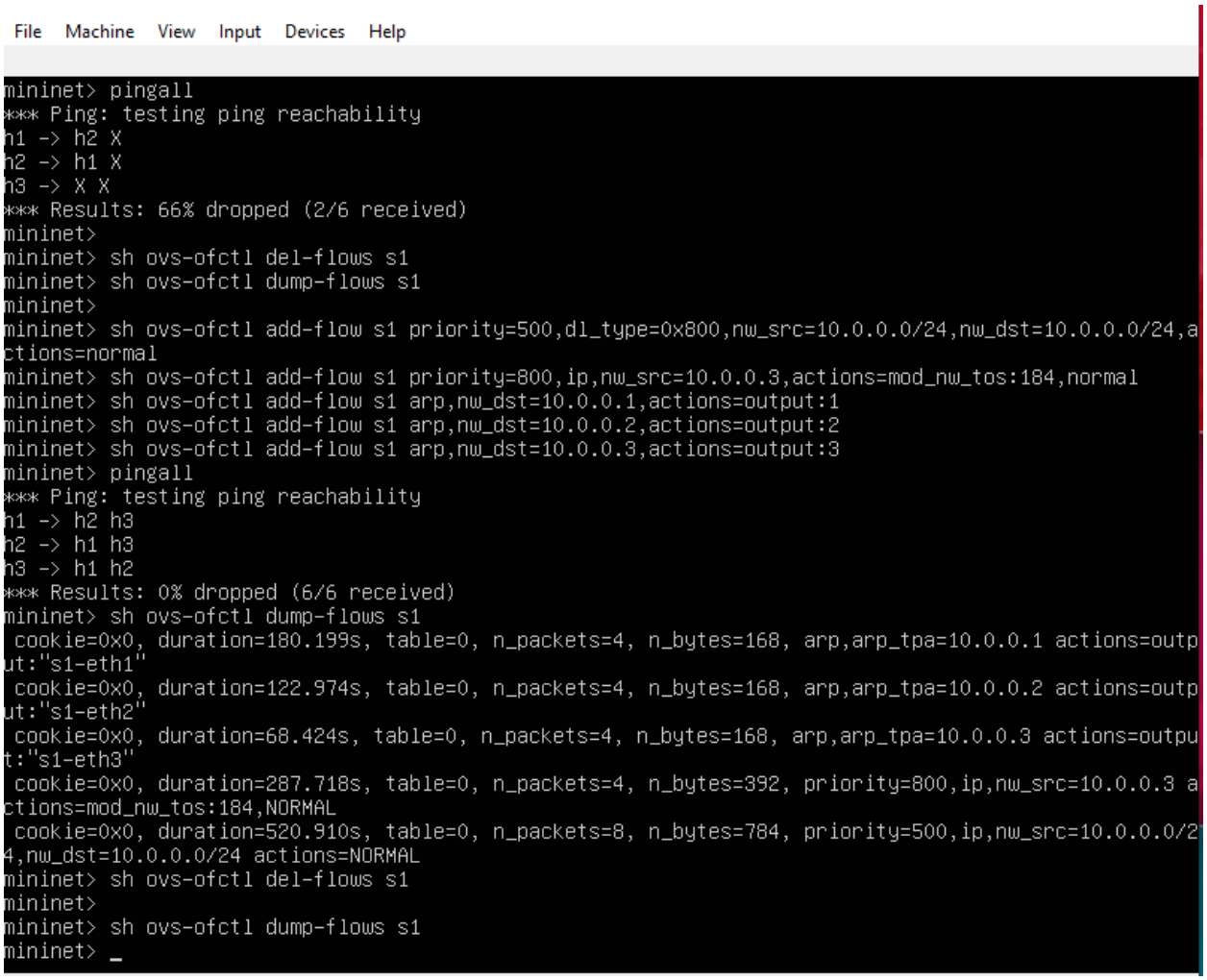

File Machine View Input Devices Help
cookie=0x0, duration=630.924s, table=0, n_packets=5, n_bytes=350, priority=500,in_port="s1-eth1" ac
tions=output:"s1-eth2"
cookie=0x0, duration=533.969s, table=0, n_packets=4, n_bytes=280, priority=500,in_port="s1-eth2" ac
tions=output:"s1-eth1"
mininet> h1 ping -c2 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=0.417 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.065 ms
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.065/0.241/0.417/0.176 ms
mininet>
mininet>
mininet> sh ovs-ofctl del-flows s1
mininet> sh ovs-ofctl dump-flows s1
mininet>
mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
inet6 fe80::200:ff:fe00:1 prefixlen 64 scopeid 0x20<link>
ether 00:00:00:00:00:01 txqueuelen 1000 (Ethernet)
RX packets 37 bytes 2678 (2.6 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 44 bytes 3096 (3.0 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
mininet> _


File Machine View Input Devices Help
inet 10.0.0.2 netmask 255.0.0.0 broadcast 10.255.255.255
inet6 fe80::200:ff:fe00:2 prefixlen 64 scopeid 0x20<link>
ether 00:00:00:00:00:02 txqueuelen 1000 (Ethernet)
RX packets 37 bytes 2678 (2.6 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 39 bytes 2774 (2.7 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
mininet> h3 ifconfig -a
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.0.3 netmask 255.0.0.0 broadcast 10.255.255.255
inet6 fe80::200:ff:fe00:3 prefixlen 64 scopeid 0x20<link>
ether 00:00:00:00:00:03 txqueuelen 1000 (Ethernet)
RX packets 29 bytes 2118 (2.1 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 34 bytes 2340 (2.3 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 2 bytes 224 (224.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 2 bytes 224 (224.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
mininet> _

Ethertype (decimal)	Ethertype (hex)	Exp. Ethernet (decimal)	Exp. Ethernet (octal)	Description
0000	0000-05DC	-	-	IEEE802.3 Length Field
0257	0101-01FF	-	-	Experimental
0512	0200	512	1000	XEROX PUP (see 0A00)
0513	0201	-	-	PUP Addr Trans (see 0A01)
	0400			Nixdorf
1536	0600	1536	3000	XEROX NS IDP
	0660			DLOG
	0661			DLOG
2048	0800	513	1001	Internet Protocol version 4 (IPv4)
2049	0801	-	-	X.75 Internet
2050	0802	-	-	NBS Internet
2051	0803	-	-	ECMA Internet
2052	0804	-	-	Chaosnet
2053	0805	-	-	X.25 Level 3
2054	0806	-	-	Address Resolution Protocol (ARP)
2055	0807	-	-	XNS Compatability
2056	0808	-	-	Frame Relay ARP
2076	081C	-	-	Symbolics Private
2184	0888-088A	-	-	Xyplex


File Machine View Input Devices Help
mininet> h3 ifconfig -a
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.0.3 netmask 255.0.0.0 broadcast 10.255.255.255
inet6 fe80::200:ff:fe00:3 prefixlen 64 scopeid 0x20<link>
ether 00:00:00:00:00:03 txqueuelen 1000 (Ethernet)
RX packets 29 bytes 2118 (2.1 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 34 bytes 2340 (2.3 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 2 bytes 224 (224.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 2 bytes 224 (224.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
mininet>
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,actions=output:2
mininet> sh ovs-ofctl add-flow s1 dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,actions=output:1
mininet> sh ovs-ofctl add-flow s1 dl_type=0x806,nw_proto=1,actions=flood
mininet> sh ovs-ofctl dump-flows s1
cookie=0x0, duration=235.517s, table=0, n_packets=0, n_bytes=0, dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=output:"s1-eth2"
cookie=0x0, duration=147.870s, table=0, n_packets=0, n_bytes=0, dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
cookie=0x0, duration=25.402s, table=0, n_packets=0, n_bytes=0, arp,arp_op=1 actions=FLOOD
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 -> X X
*** Results: 66% dropped (2/6 received)
mininet> _

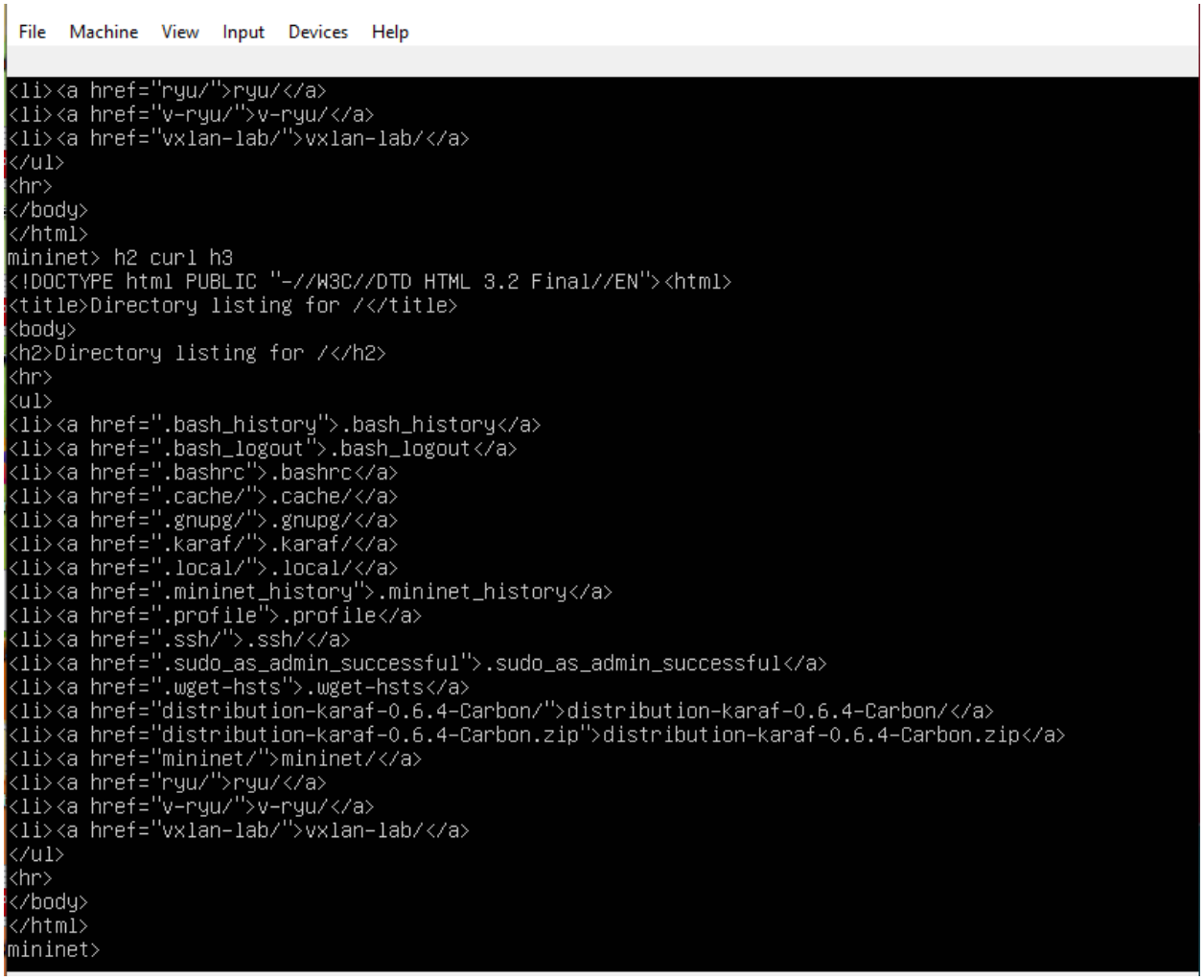
L3:


File Machine View Input Devices Help
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X
h2 -> h1 X
h3 -> X X
*** Results: 66% dropped (2/6 received)
mininet>
mininet> sh ovs-ofctl del-flows s1
mininet> sh ovs-ofctl dump-flows s1
mininet>
mininet> sh ovs-ofctl add-flow s1 priority=500,dl_type=0x800,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,a
ctions=normal
mininet> sh ovs-ofctl add-flow s1 priority=800,ip,nw_src=10.0.0.3,actions=mod_nw_tos:184,normal
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.1,actions=output:1
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.2,actions=output:2
mininet> sh ovs-ofctl add-flow s1 arp,nw_dst=10.0.0.3,actions=output:3
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet> sh ovs-ofctl dump-flows s1
cookie=0x0, duration=180.199s, table=0, n_packets=4, n_bytes=168, arp,arp_tpa=10.0.0.1 actions=outp
ut:"s1-eth1"
cookie=0x0, duration=122.974s, table=0, n_packets=4, n_bytes=168, arp,arp_tpa=10.0.0.2 actions=outp
ut:"s1-eth2"
cookie=0x0, duration=68.424s, table=0, n_packets=4, n_bytes=168, arp,arp_tpa=10.0.0.3 actions=outpu
t:"s1-eth3"
cookie=0x0, duration=287.718s, table=0, n_packets=4, n_bytes=392, priority=800,ip,nw_src=10.0.0.3 a
ctions=mod_nw_tos:184,NORMAL
cookie=0x0, duration=520.910s, table=0, n_packets=8, n_bytes=784, priority=500,ip,nw_src=10.0.0.0/2
4,nw_dst=10.0.0.0/24 actions=NORMAL
mininet> sh ovs-ofctl del-flows s1
mininet>
mininet> sh ovs-ofctl dump-flows s1
mininet> _

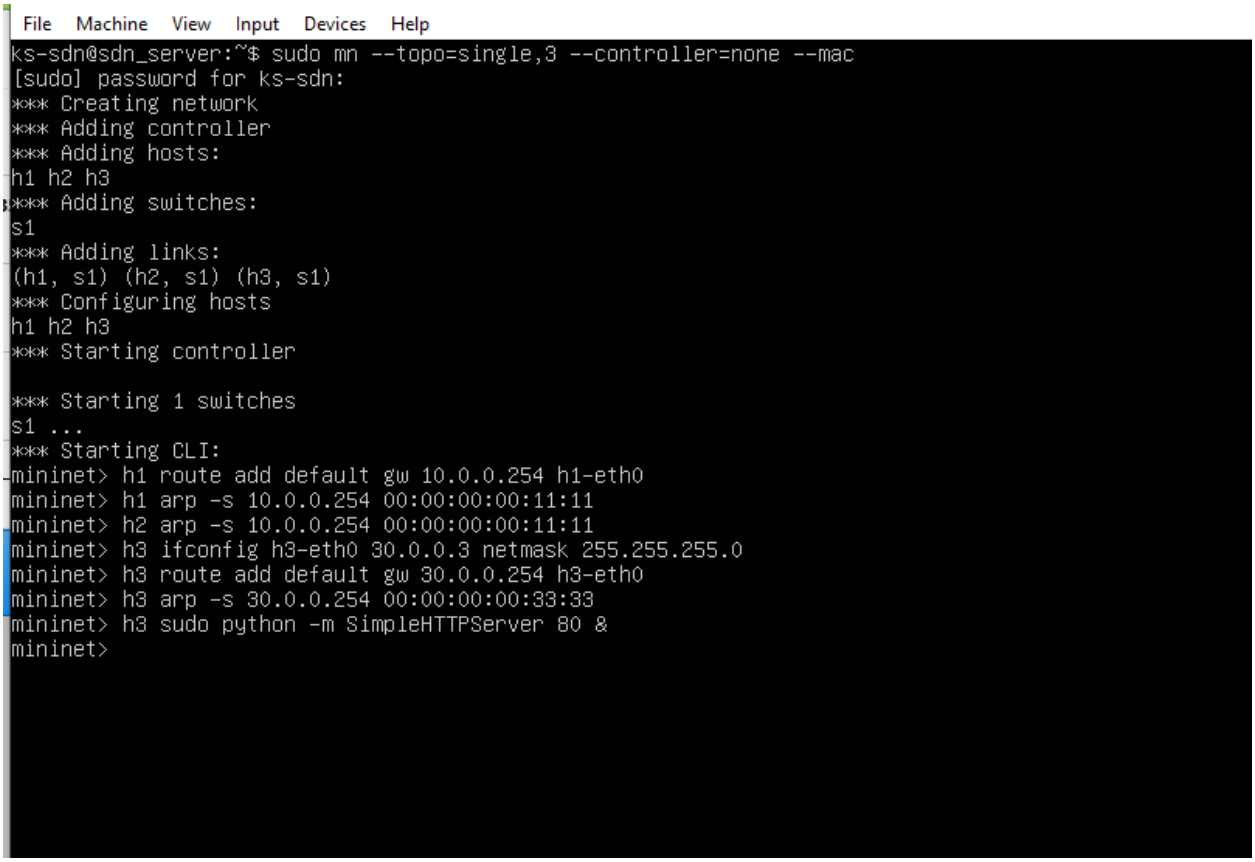
L4:

File Machine View Input Devices Help

```
 cookie=0x0, duration=520.910s, table=0, n_packets=8, n_bytes=784, priority=500,ip,nw_src=10.0.0.0/2
4,nw_dst=10.0.0.0/24 actions=NORMAL
mininet> sh ovs-ofctl del-flows s1
mininet>
mininet> sh ovs-ofctl dump-flows s1
mininet> h3 python -m SimpleHTTPServer 80 &
mininet> sh ovs-ofctl add-flow s1 actions=normal
mininet> h1 curl h3
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a>
<li><a href=".bash_logout">.bash_logout</a>
<li><a href=".bashrc">.bashrc</a>
<li><a href=".cache/">.cache/</a>
<li><a href=".gnupg/">.gnupg/</a>
<li><a href=".karaf/">.karaf/</a>
<li><a href=".local/">.local/</a>
<li><a href=".mininet_history">.mininet_history</a>
<li><a href=".profile">.profile</a>
<li><a href=".ssh/">.ssh/</a>
<li><a href=".sudo_as_admin_successful">.sudo_as_admin_successful</a>
<li><a href=".wget-hsts">.wget-hsts</a>
<li><a href="distribution-karaf-0.6.4-Carbon/">distribution-karaf-0.6.4-Carbon/</a>
<li><a href="distribution-karaf-0.6.4-Carbon.zip">distribution-karaf-0.6.4-Carbon.zip</a>
<li><a href="mininet/">mininet/</a>
<li><a href="ryu/">ryu/</a>
<li><a href="v-ryu/">v-ryu/</a>
<li><a href="vxlan-lab/">vxlan-lab/</a>
</ul>
<hr>
</body>
</html>
mininet> _
```



Open vSwitch Multiple Flow Tables


File Machine View Input Devices Help
ks-sdn@sdn_server:~$ sudo mn --topo=single,3 --controller=none --mac
[sudo] password for ks-sdn:
*** Creating network
*** Adding controller
*** 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> h1 route add default gw 10.0.0.254 h1-eth0
mininet> h1 arp -s 10.0.0.254 00:00:00:00:11:11
mininet> h2 arp -s 10.0.0.254 00:00:00:00:11:11
mininet> h3 ifconfig h3-eth0 30.0.0.3 netmask 255.255.255.0
mininet> h3 route add default gw 30.0.0.254 h3-eth0
mininet> h3 arp -s 30.0.0.254 00:00:00:00:33:33
mininet> h3 sudo python -m SimpleHTTPServer 80 &
mininet>

File Machine View Input Devices Help

```
 GNU nano 2.9.3                        tables.txt                          Modified

#table 0 - Access Control
table=0,ip,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,actions=resubmit(,1)
table=0,arp,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,actions=resubmit(,1)
table=0,icmp,nw_src=10.0.0.1,nw_dst=30.0.0.3,actions=resubmit(,1)
table=0,tcp,nw_src=10.0.0.1,nw_dst=30.0.0.3,tp_dst=80,actions=resubmit(,1)
table=0,ip,nw_src=30.0.0.3,actions=resubmit(,1)
table=0,priority=0,actions=drop

#table 1 - NAT
table=1,ip,nw_src=10.0.0.1,nw_dst=30.0.0.3,actions=mod_nw_src=5.5.5.5,resubmit(,2)
table=1,ip,nw_src=30.0.0.3,nw_dst=5.5.5.5,actions=mod_nw_dst=10.0.0.1,resubmit(,2)
table=1,priority=0,actions=resubmit(,2)

#table 2 forward/route
table=2,ip,nw_dst=10.0.0.1,actions=mod_dl_dst=00:00:00:00:00:01,output:1
table=2,ip,nw_dst=10.0.0.2,actions=mod_dl_dst=00:00:00:00:00:02,output:2
table=2,ip,nw_dst=30.0.0.3,actions=mod_dl_dst=00:00:00:00:00:03,output:3
priority=0,table=2,arp,nw_dst=10.0.0.1,actions=output:1
priority=0,table=2,arp,nw_dst=10.0.0.2,actions=output:2
_

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos  M-U Undo
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Spell ^_ Go To Line M-E Redo
```

```
mininet>
mininet>
mininet> sh ovs-ofctl dump-flows s1 -O OpenFlow13 > exampleLog.txt
mininet> sh more tables.txt
#table 0 - Access Control
table=0,ip,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,actions=resubmit(,1)
table=0,arp,nw_src=10.0.0.0/24,nw_dst=10.0.0.0/24,actions=resubmit(,1)
table=0,icmp,nw_src=10.0.0.1,nw_dst=30.0.0.3,actions=resubmit(,1)
table=0,tcp,nw_src=10.0.0.1,nw_dst=30.0.0.3,tp_dst=80,actions=resubmit(,1)
table=0,ip,nw_src=30.0.0.3,actions=resubmit(,1)
table=0,priority=0,actions=drop

#table 1 - NAT
table=1,ip,nw_src=10.0.0.1,nw_dst=30.0.0.3,actions=mod_nw_src=5.5.5.5,resubmit(,2)
table=1,ip,nw_src=30.0.0.3,nw_dst=5.5.5.5,actions=mod_nw_dst=10.0.0.1,resubmit(,2)
table=1,priority=0,actions=resubmit(,2)

#table 2 forward/route
table=2,ip,nw_dst=10.0.0.1,actions=mod_dl_dst=00:00:00:00:00:01,output:1
table=2,ip,nw_dst=10.0.0.2,actions=mod_dl_dst=00:00:00:00:00:02,output:2
table=2,ip,nw_dst=30.0.0.3,actions=mod_dl_dst=00:00:00:00:00:03,output:3
priority=0,table=2,arp,nw_dst=10.0.0.1,actions=output:1
priority=0,table=2,arp,nw_dst=10.0.0.2,actions=output:2

mininet>
```

```
#table 2 forward/route
table=2,ip,nw_dst=10.0.0.1,actions=mod_dl_dst=00:00:00:00:00:01,output:1
table=2,ip,nw_dst=10.0.0.2,actions=mod_dl_dst=00:00:00:00:00:02,output:2
table=2,ip,nw_dst=30.0.0.3,actions=mod_dl_dst=00:00:00:00:00:03,output:3
priority=0,table=2,arp,nw_dst=10.0.0.1,actions=output:1
priority=0,table=2,arp,nw_dst=10.0.0.2,actions=output:2

mininet> sh ovs-ofctl add-flows s1 tables.txt
mininet> h1 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=0.495 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.069 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1031ms
rtt min/avg/max/mdev = 0.069/0.282/0.495/0.213 ms
mininet> h1 ping -c2 30.0.0.3
PING 30.0.0.3 (30.0.0.3) 56(84) bytes of data.
64 bytes from 30.0.0.3: icmp_seq=1 ttl=64 time=0.365 ms
64 bytes from 30.0.0.3: icmp_seq=2 ttl=64 time=0.056 ms

--- 30.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.056/0.210/0.365/0.155 ms
mininet> h2 ping -c2 30.0.0.3
connect: Network is unreachable
mininet>
```

```
File  Machine  View  Input  Devices  Help
--- 30.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.056/0.210/0.365/0.155 ms
mininet> h2 ping -c2 30.0.0.3
connect: Network is unreachable
mininet> h1 curl 30.0.0.3
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href=".bash_history">.bash_history</a>
<li><a href=".bash_logout">.bash_logout</a>
<li><a href=".bashrc">.bashrc</a>
<li><a href=".cache/">.cache/</a>
<li><a href=".gnupg/">.gnupg/</a>
<li><a href=".karaf/">.karaf/</a>
<li><a href=".local/">.local/</a>
<li><a href=".mininet_history">.mininet_history</a>
<li><a href=".profile">.profile</a>
<li><a href=".ssh/">.ssh/</a>
<li><a href=".sudo_as_admin_successful">.sudo_as_admin_successful</a>
<li><a href=".wget-hsts">.wget-hsts</a>
<li><a href="distribution-karaf-0.6.4-Carbon/">distribution-karaf-0.6.4-Carbon/</a>
<li><a href="distribution-karaf-0.6.4-Carbon.zip">distribution-karaf-0.6.4-Carbon.zip</a>
<li><a href="exampleLog.txt">exampleLog.txt</a>
<li><a href="mininet/">mininet/</a>
<li><a href="ryu/">ryu/</a>
<li><a href="tables.txt">tables.txt</a>
<li><a href="v-ryu/">v-ryu/</a>
<li><a href="vxlan-lab/">vxlan-lab/</a>
</ul>
<hr>
</body>
</html>
mininet>
```

Hands-on Training with Python Net Apps - Elective 2

APP - 1

```
ks-sdn@sdn_server:~/mininet/examples$ ls
baresshd.py       controllers2.py  limit.py           multitest.py      scratchnetuser.py
bind.py           controllers.py   linearbandwidth.py natnet.py         simpleperf.py
clustercli.py     controlnet.py    linuxrouter.py     nat.py            sshd.py
clusterdemo.py    cpu.py           miniedit.py        numberedports.py  test
clusterperf.py    emptynet.py      mobility.py        popenpoll.py      tree1024.py
cluster.py        hwintf.py        multilink.py       popen.py          treeping64.py
clusterSanity.py  __init__.py      multiping.py       README.md         vlanhost.py
consoles.py       intfoptions.py   multipoll.py       scratchnet.py
ks-sdn@sdn_server:~/mininet/examples$ cat controllers.py _
```

```
"""
Create a network where different switches are connected to
different controllers, by creating a custom Switch() subclass.
"""

from mininet.net import Mininet
from mininet.node import OVSSwitch, Controller, RemoteController
from mininet.topolib import TreeTopo
from mininet.log import setLogLevel
from mininet.cli import CLI

setLogLevel( 'info' )

# Two local and one "external" controller (which is actually c0)
# Ignore the warning message that the remote isn't (yet) running
c0 = Controller( 'c0', port=6633 )
c1 = Controller( 'c1', port=6634 )
c2 = RemoteController( 'c2', ip='127.0.0.1', port=6633 )

cmap = { 's1': c0, 's2': c1, 's3': c2 }

class MultiSwitch( OVSSwitch ):
    "Custom Switch() subclass that connects to different controllers"
    def start( self, controllers ):
        return OVSSwitch.start( self, [ cmap[ self.name ] ] )


topo = TreeTopo( depth=2, fanout=2 )
net = Mininet( topo=topo, switch=MultiSwitch, build=False, waitConnected=True )
for c in [ c0, c1 ]:
    net.addController(c)
net.build()
net.start()
CLI( net )
net.stop()
ks-sdn@sdn_server:~/mininet/examples$ _
```

Explanation of Code App:

- The first statement is a string literal, this is the module documentation string, explaining what the app does, with the help of docstring tools, user can interactively browse through the code.

- The five import statements are used to import **Mininet, OVSSwitch, Controller, RemoteController, TreeTopo, setLogLevel,** and **CLI modules** from their parent modules respectively.
- **setLogLevel()** function was called by passing an argument to setup log level.
- **#** is used to write comments for the code
- Two instances of class **Controller, c0, c1** were instantiated with respective instance variables, to run executable files (OpenFlow Controllers)
- one instance of class **RemoteController** were also instatntiated with its instance variables.

- A dictionary of the instances was assigned to cmap in the form of key:value pairs enclosed with {} to form a data structure were the unique keys are s1, s2, and s3 and values are c0, c1, c2.

- A MultiSwitch class was created with a valid attribute (function) to start up an OVS OpenFlow Switches on controllers.
- Instance of TreeTopo class with instance variables (depth=2, fanout=2) instantiated
- Instance of Mininet class with instance variables instatntiated
- The **for** statement is used to iterate a list and then add controller, this will add two controllers.
- **net.build()** build mininet network.
- **net.start()** start controllers and switches.
- **CLI(net)** runs a batch or interactive mode for the network.
- The **net.stop(**) stop controllers, switches, and hosts.

RESULT OF CODE:

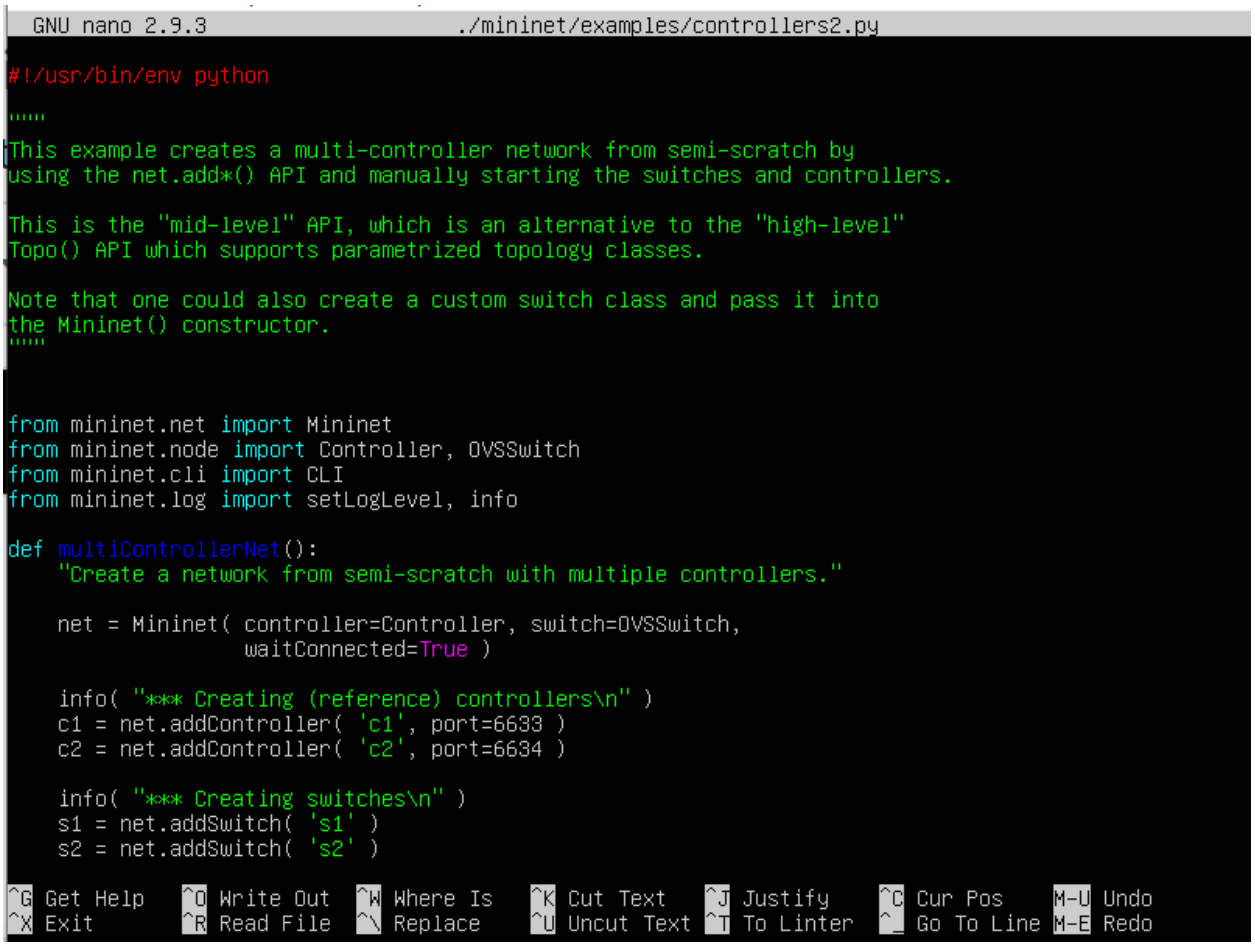
```
ks-sdn@sdn_server:~/mininet/examples$ ls
baresshd.py       controllers2.py  limit.py           multitest.py      scratchnetuser.py
bind.py           controllers.py   linearbandwidth.py  natnet.py         simpleperf.py
clustercli.py     controlnet.py    linuxrouter.py     nat.py            sshd.py
clusterdemo.py    cpu.py           miniedit.py        numberedports.py  test
clusterperf.py    emptynet.py      mobility.py        popenpoll.py      tree1024.py
cluster.py        hwintf.py        multilink.py       popen.py          treeping64.py
clusterSanity.py  __init__.py      multiping.py       README.md         vlanhost.py
consoles.py       intfoptions.py   multipoll.py       scratchnet.py
ks-sdn@sdn_server:~/mininet/examples$ cd ../..
ks-sdn@sdn_server:~$ sudo mn -c
[sudo] password for ks-sdn:
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-te
stcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs
-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
ks-sdn@sdn_server:~$
```

```
ks-sdn@sdn_server:~$ sudo python ./mininet/examples/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 ...
*** Waiting for switches to connect
s1 s2 s3
*** Starting CLI:
mininet> _
```

```
*** Waiting for switches to connect
s1 s2 s3
*** Starting CLI:
mininet> help

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

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> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> _
```

```
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=2920>
<Host h2: h2-eth0:10.0.0.2 pid=2922>
<Host h3: h3-eth0:10.0.0.3 pid=2924>
<Host h4: h4-eth0:10.0.0.4 pid=2926>
<MultiSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=2931>
<MultiSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=2934>
<MultiSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=2937>
<Controller c0: 127.0.0.1:6633 pid=2903>
<Controller c1: 127.0.0.1:6634 pid=2908>
mininet> nodes
available nodes are:
c0 c1 h1 h2 h3 h4 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
mininet> exit
*** Stopping 2 controllers
c0 c1
*** Stopping 6 links
......
*** Stopping 3 switches
s1 s2 s3
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
ks-sdn@sdn_server:~$
```

APP - 2


GNU nano 2.9.3 ./mininet/examples/controllers2.py
#!/usr/bin/env python
"""
This example creates a multi-controller network from semi-scratch by
using the net.add*() API and manually starting the switches and controllers.
This is the "mid-level" API, which is an alternative to the "high-level"
Topo() API which supports parametrized topology classes.
Note that one could also create a custom switch class and pass it into
the Mininet() constructor.
"""
from mininet.net import Mininet
from mininet.node import Controller, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
def multiControllerNet():
"Create a network from semi-scratch with multiple controllers."
net = Mininet(controller=Controller, switch=OVSSwitch,
waitConnected=True)
info("*** Creating (reference) controllers\n")
c1 = net.addController('c1', port=6633)
c2 = net.addController('c2', port=6634)
info("*** Creating switches\n")
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line M-E Redo

```
  GNU nano 2.9.3                    ./mininet/examples/controllers2.py

    s2 = net.addSwitch( 's2' )

    info( "*** Creating hosts\n" )
    hosts1 = [ net.addHost( 'h%d' % n ) for n in ( 3, 4 ) ]
    hosts2 = [ net.addHost( 'h%d' % n ) for n in ( 5, 6 ) ]

    info( "*** Creating links\n" )
    for h in hosts1:
        net.addLink( s1, h )
    for h in hosts2:
        net.addLink( s2, h )
    net.addLink( s1, s2 )

    info( "*** Starting network\n" )
    net.build()
    c1.start()
    c2.start()
    s1.start( [ c1 ] )
    s2.start( [ c2 ] )

    info( "*** Testing network\n" )
    net.pingAll()

    info( "*** Running CLI\n" )
    CLI( net )

    info( "*** Stopping network\n" )
    net.stop()


if __name__ == '__main__':
    setLogLevel( 'info' )  # for CLI output

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Linter  ^_ Go To Line M-E Redo
```

```
    info( "*** Stopping network\n" )
    net.stop()


if __name__ == '__main__':
    setLogLevel( 'info' )  # for CLI output
    multiControllerNet()

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Linter  ^_ Go To Line M-E Redo
```

Code Explanation:

- The first statement is a string literal, this is the module documentation string, explaining what the app does, with the help of docstring tools, user can interactively browse through the code.

- The four import statements are used to import **Mininet, OVSSwitch, Controller, setLogLevel, Info, and CLI modules** from their parent modules respectively.

- **def** is used to define **multiControllerNe**t function that create multiple controllers by first instantiating Mininet class, called **info()** method, added two controllers to local variables **c1, c2**, added two switches to local variables **s1, s2**, added two hosts to local variables **host1, and host2**, used the **for** loop to add links to hosts and switches.
    - **net.build()** build mininet network
    - **c1.start()** start first controller
    - **c2.start()** start second controller
    - **s1.start([c1])** start first switch on first controller
    - **s2.start([c2])** start second switch on second controller
    - **net.pingAll()** pings all hosts
    - **CLI(net)** runs a batch or interactive mode for the network.
    - The **net.stop()** stop controllers, switches, and hosts.

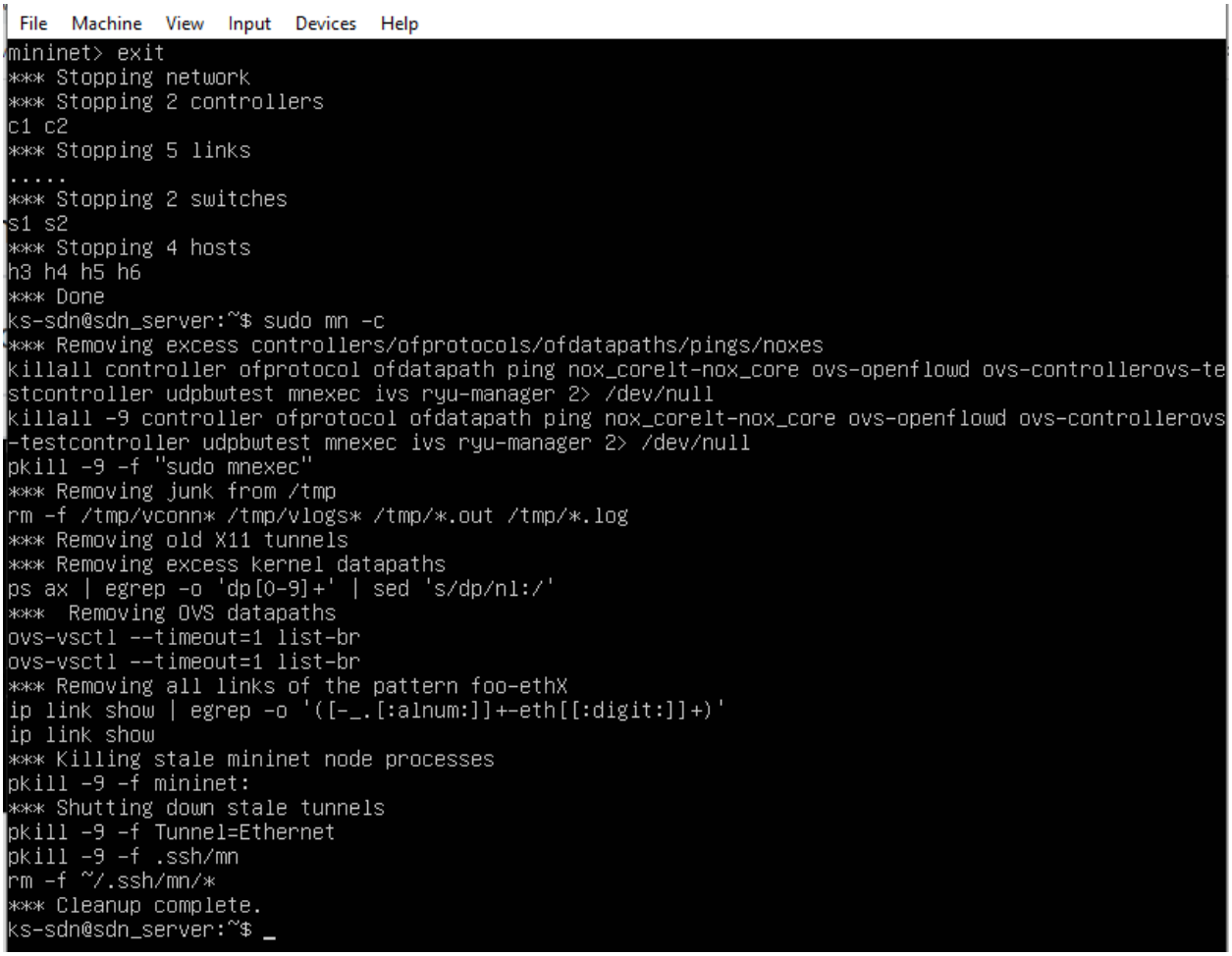
- The **if __name__ = “ __main__”:** statement makes the file usable as a script and importable module.

This app should create mininet network with two controllers, two switches, and two hosts on each switch, and then ping all nodes.

RESULT:

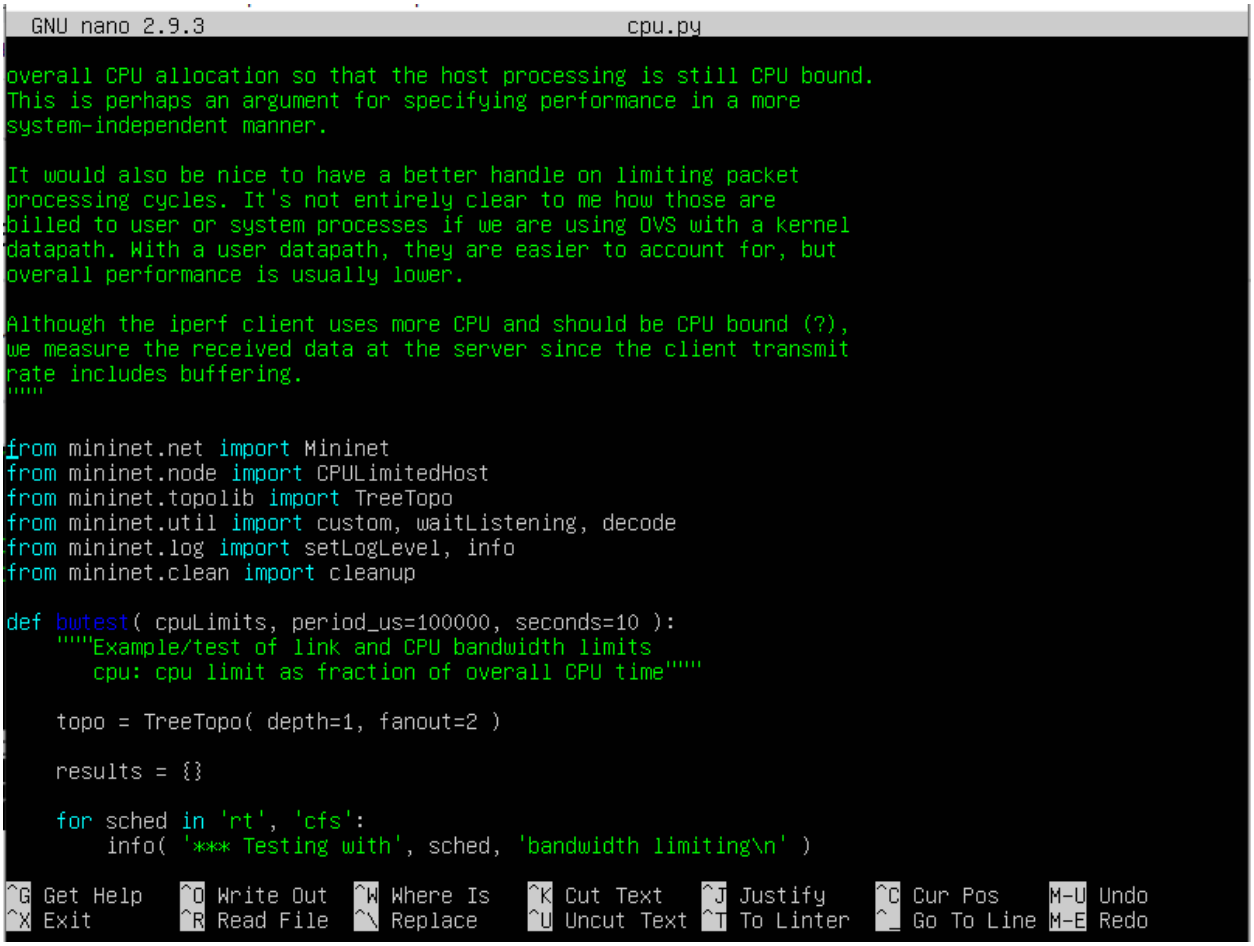
File Machine View Input Devices Help

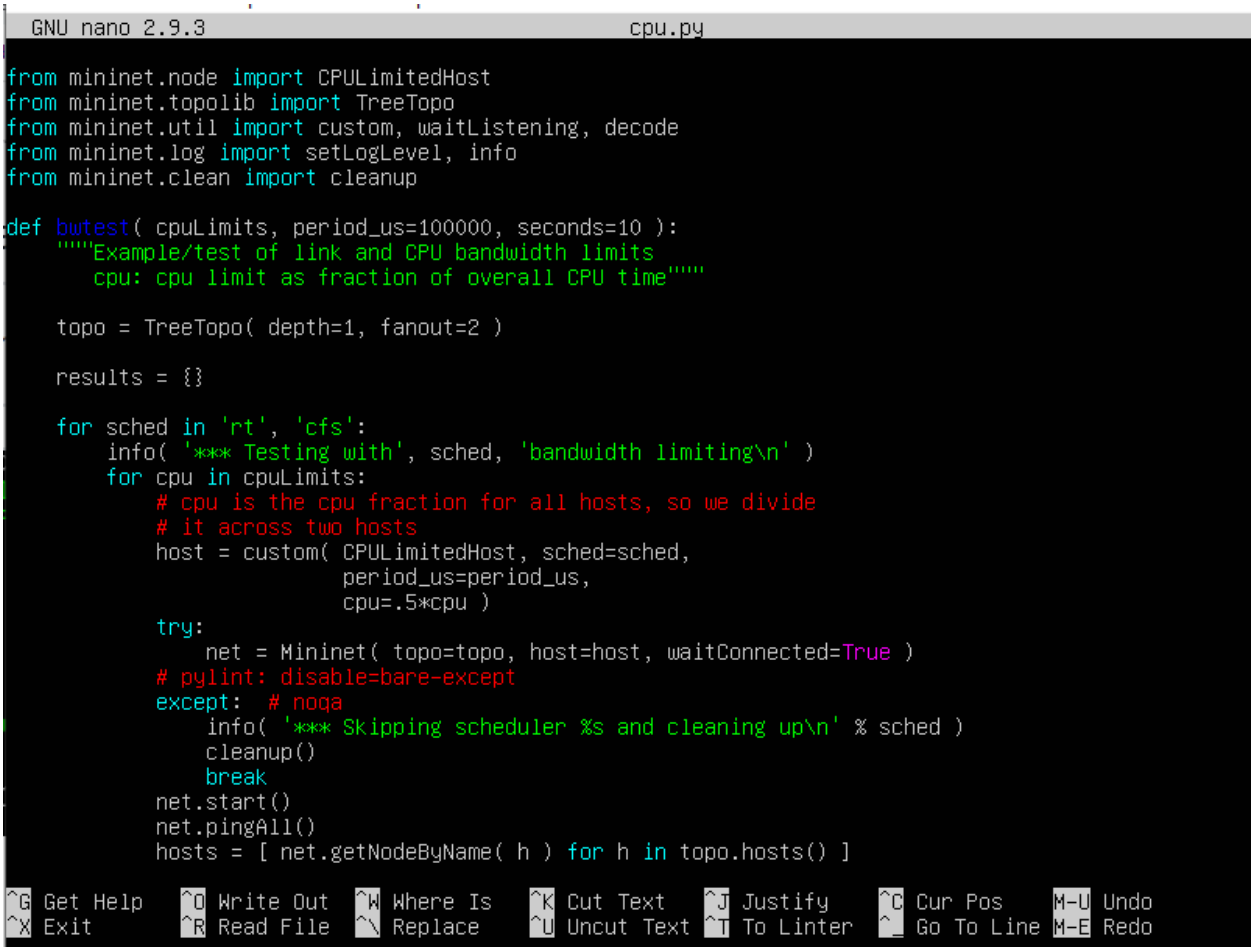
```
ks-sdn@sdn_server:~$ sudo python ./mininet/examples/controllers2.py
[sudo] password for ks-sdn:
*** Creating (reference) controllers
*** Creating switches
*** Creating hosts
*** Creating links
*** Starting network
*** Configuring hosts
h3 h4 h5 h6
*** Testing network
*** Ping: testing ping reachability
h3 -> h4 h5 h6
h4 -> h3 h5 h6
h5 -> h3 h4 h6
h6 -> h3 h4 h5
*** Results: 0% dropped (12/12 received)
*** Running CLI
*** Starting CLI:
mininet> dump
<Host h3: h3-eth0:10.0.0.1 pid=3375>
<Host h4: h4-eth0:10.0.0.2 pid=3377>
<Host h5: h5-eth0:10.0.0.3 pid=3379>
<Host h6: h6-eth0:10.0.0.4 pid=3381>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=3367>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=3370>
<Controller c1: 127.0.0.1:6633 pid=3354>
<Controller c2: 127.0.0.1:6634 pid=3359>
mininet> net
h3 h3-eth0:s1-eth1
h4 h4-eth0:s1-eth2
h5 h5-eth0:s2-eth1
h6 h6-eth0:s2-eth2
s1 lo:  s1-eth1:h3-eth0 s1-eth2:h4-eth0 s1-eth3:s2-eth3
s2 lo:  s2-eth1:h5-eth0 s2-eth2:h6-eth0 s2-eth3:s1-eth3
c1
c2
mininet> _
```

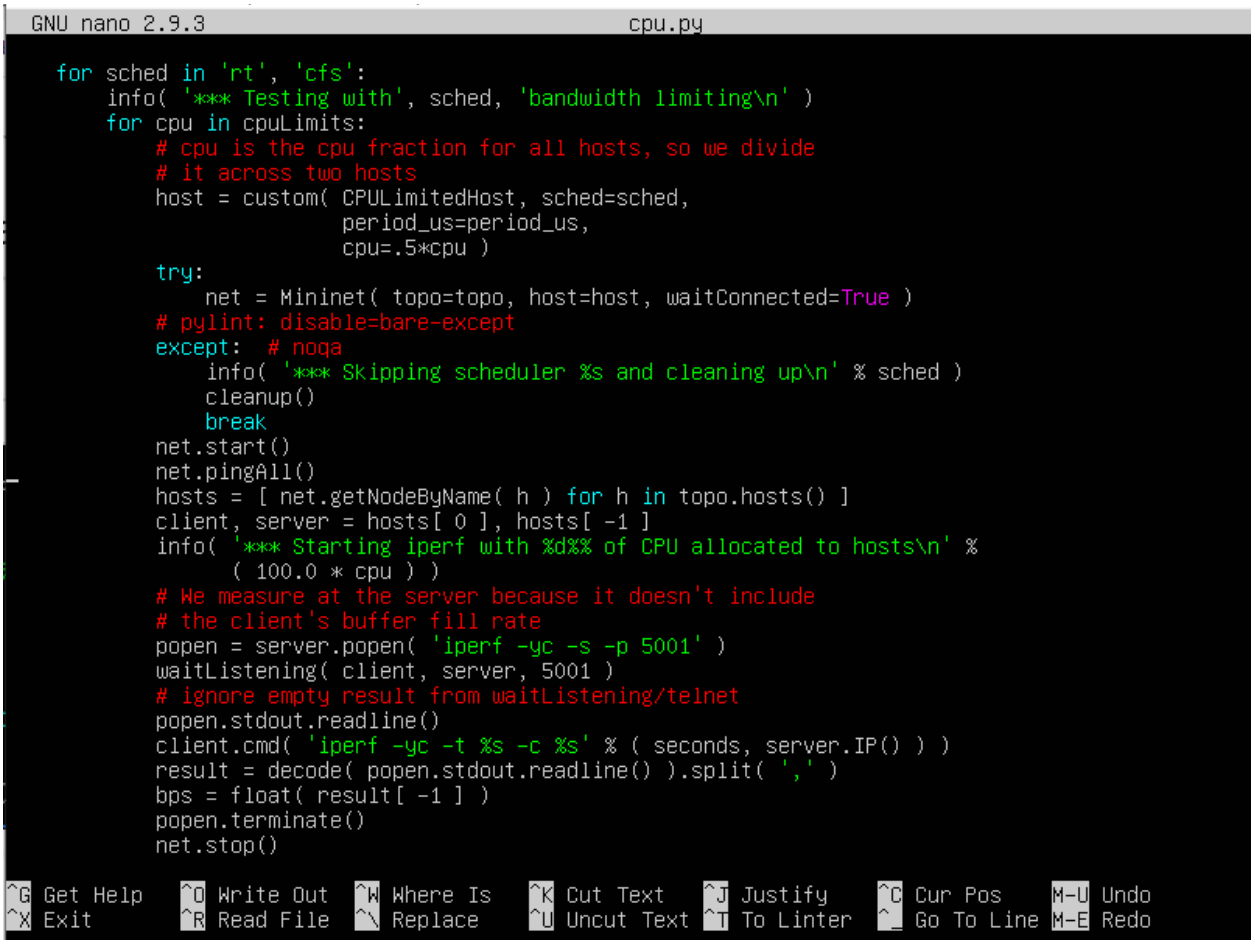

File Machine View Input Devices Help
mininet> exit
*** Stopping network
*** Stopping 2 controllers
c1 c2
*** Stopping 5 links
.....
*** Stopping 2 switches
s1 s2
*** Stopping 4 hosts
h3 h4 h5 h6
*** Done
ks-sdn@sdn_server:~$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-te
stcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs
-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
ks-sdn@sdn_server:~$ _

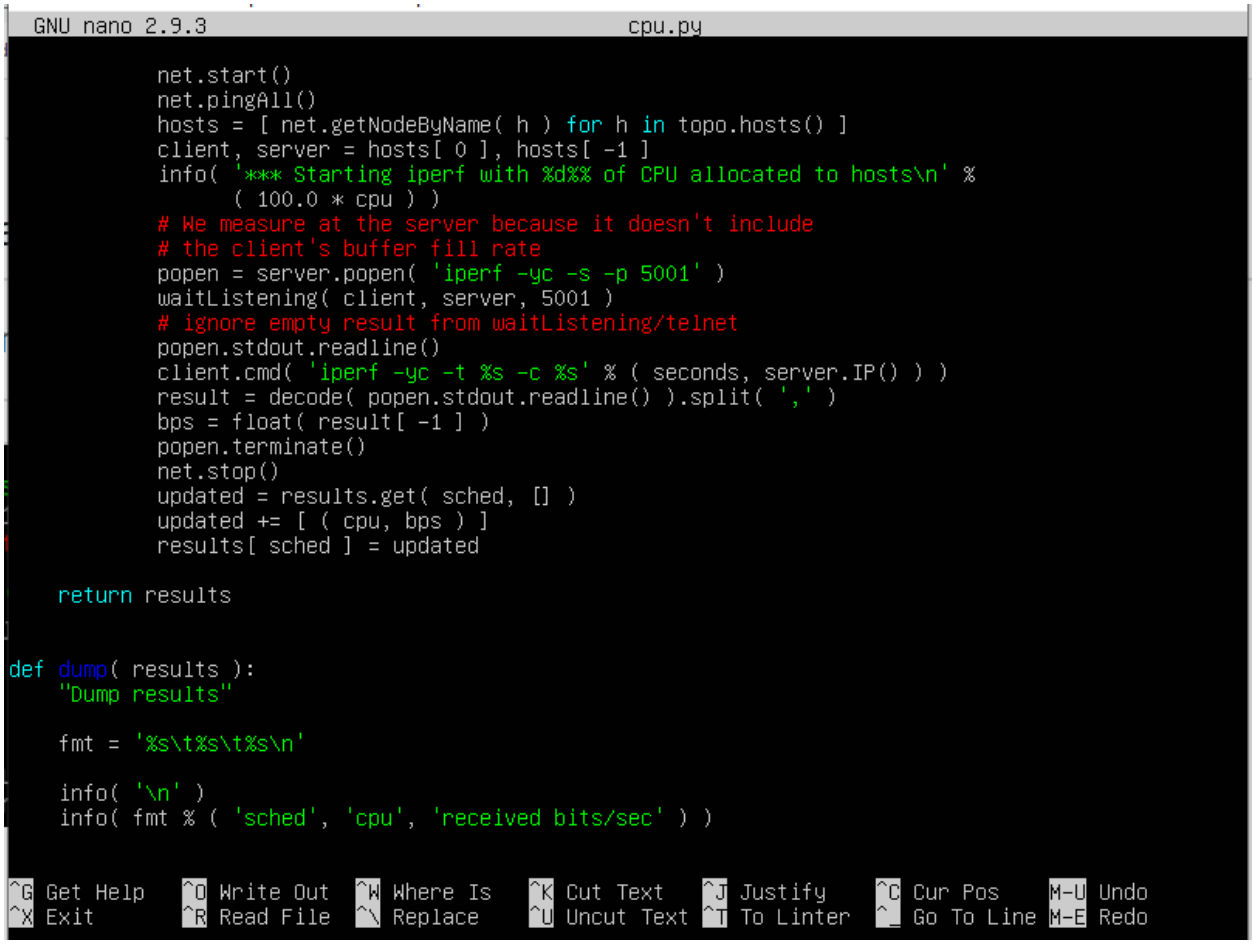
APP - 3

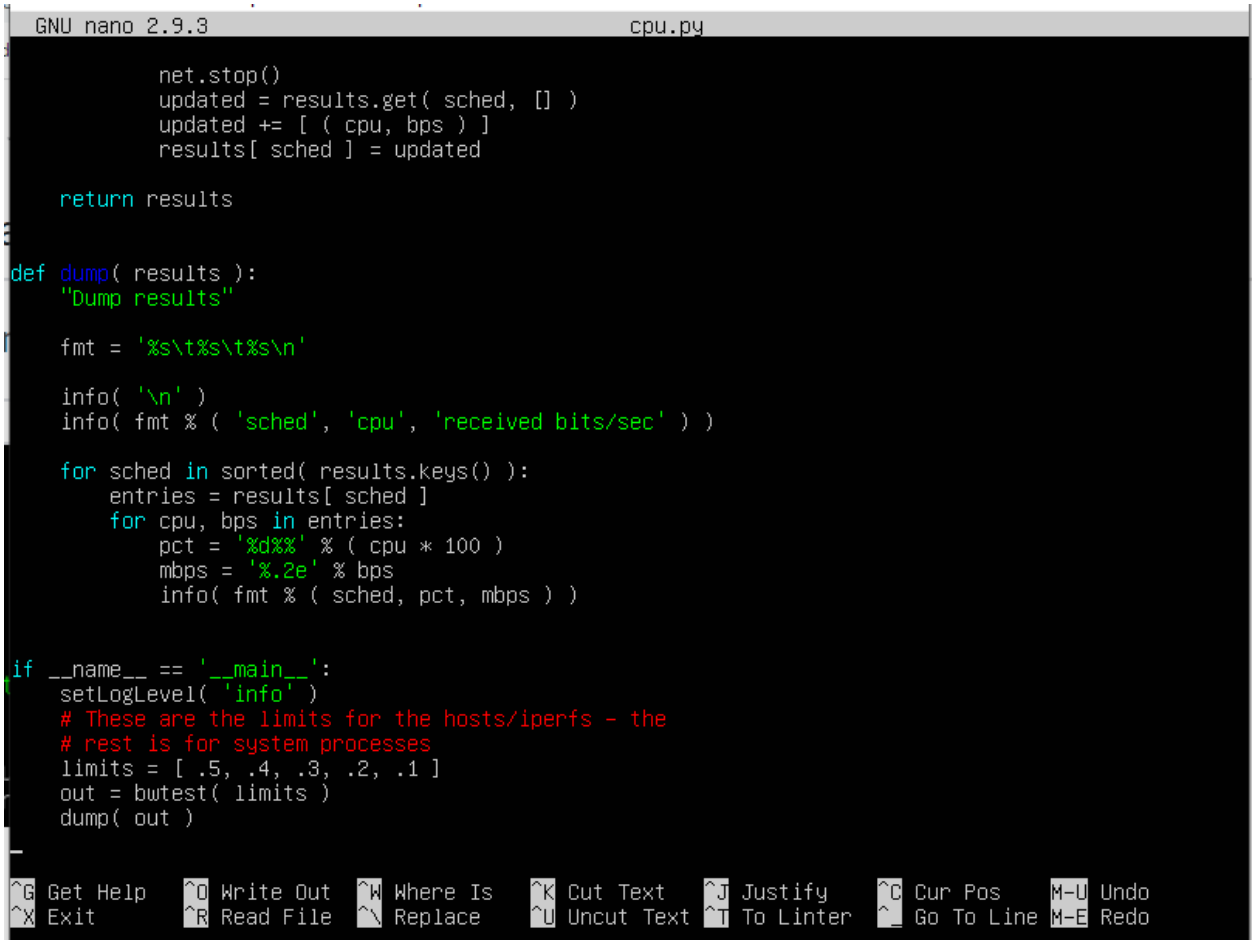

GNU nano 2.9.3
cpu.py
#!/usr/bin/env python
"""
cpu.py: test iperf bandwidth for varying cpu limits
Since we are limiting the hosts (only), we should expect the iperf
processes to be affected, as well as any system processing which is
billed to the hosts.
We reserve >50% of cycles for system processing; we assume that
this is enough for it not to affect results. Hosts are limited to
40% of total cycles, which we assume is enough to make them CPU
bound.
As CPU performance increases over time, we may have to reduce the
overall CPU allocation so that the host processing is still CPU bound.
This is perhaps an argument for specifying performance in a more
system-independent manner.
It would also be nice to have a better handle on limiting packet
processing cycles. It's not entirely clear to me how those are
billed to user or system processes if we are using OVS with a kernel
datapath. With a user datapath, they are easier to account for, but
overall performance is usually lower.
Although the iperf client uses more CPU and should be CPU bound (?),
we measure the received data at the server since the client transmit
rate includes buffering.
"""
from mininet.net import Mininet
from mininet.node import CPULimitedHost
[Read 107 lines]
^G Get Help
^O Write Out
^W Where Is
^K Cut Text
^J Justify
^C Cur Pos
M-U Undo
^X Exit
^R Read File
^\ Replace
^U Uncut Text
^T To Linter
^_ Go To Line
M-E Redo


GNU nano 2.9.3
cpu.py
overall CPU allocation so that the host processing is still CPU bound.
This is perhaps an argument for specifying performance in a more
system-independent manner.
It would also be nice to have a better handle on limiting packet
processing cycles. It's not entirely clear to me how those are
billed to user or system processes if we are using OVS with a kernel
datapath. With a user datapath, they are easier to account for, but
overall performance is usually lower.
Although the iperf client uses more CPU and should be CPU bound (?),
we measure the received data at the server since the client transmit
rate includes buffering.
"""
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.topolib import TreeTopo
from mininet.util import custom, waitListening, decode
from mininet.log import setLogLevel, info
from mininet.clean import cleanup
def bwtest(cpuLimits, period_us=100000, seconds=10):
"""Example/test of link and CPU bandwidth limits
cpu: cpu limit as fraction of overall CPU time"""
topo = TreeTopo(depth=1, fanout=2)
results = {}
for sched in 'rt', 'cfs':
info('*** Testing with', sched, 'bandwidth limiting\n')
^G Get Help
^O Write Out
^W Where Is
^K Cut Text
^J Justify
^C Cur Pos
M-U Undo
^X Exit
^R Read File
^\ Replace
^U Uncut Text
^T To Linter
^_ Go To Line
M-E Redo


GNU nano 2.9.3 cpu.py
from mininet.node import CPULimitedHost
from mininet.topolib import TreeTopo
from mininet.util import custom, waitListening, decode
from mininet.log import setLogLevel, info
from mininet.clean import cleanup
def bwtest(cpuLimits, period_us=100000, seconds=10):
"""Example/test of link and CPU bandwidth limits
cpu: cpu limit as fraction of overall CPU time"""
topo = TreeTopo(depth=1, fanout=2)
results = {}
for sched in 'rt', 'cfs':
info('*** Testing with', sched, 'bandwidth limiting\n')
for cpu in cpuLimits:
# cpu is the cpu fraction for all hosts, so we divide
# it across two hosts
host = custom(CPULimitedHost, sched=sched,
period_us=period_us,
cpu=.5*cpu)
try:
net = Mininet(topo=topo, host=host, waitConnected=True)
# pylint: disable=bare-except
except: # noqa
info('*** Skipping scheduler %s and cleaning up\n' % sched)
cleanup()
break
net.start()
net.pingAll()
hosts = [net.getNodeByName(h) for h in topo.hosts()]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line M-E Redo


GNU nano 2.9.3
cpu.py
for sched in 'rt', 'cfs':
info('*** Testing with', sched, 'bandwidth limiting\n')
for cpu in cpuLimits:
# cpu is the cpu fraction for all hosts, so we divide
# it across two hosts
host = custom(CPULimitedHost, sched=sched,
period_us=period_us,
cpu=.5*cpu)
try:
net = Mininet(topo=topo, host=host, waitConnected=True)
# pylint: disable=bare-except
except: # noqa
info('*** Skipping scheduler %s and cleaning up\n' % sched)
cleanup()
break
net.start()
net.pingAll()
hosts = [net.getNodeByName(h) for h in topo.hosts()]
client, server = hosts[0], hosts[-1]
info('*** Starting iperf with %d%% of CPU allocated to hosts\n' %
(100.0 * cpu))
# We measure at the server because it doesn't include
# the client's buffer fill rate
popen = server.popen('iperf -yc -s -p 5001')
waitListening(client, server, 5001)
# ignore empty result from waitListening/telnet
popen.stdout.readline()
client.cmd('iperf -yc -t %s -c %s' % (seconds, server.IP()))
result = decode(popen.stdout.readline()).split(',')
bps = float(result[-1])
popen.terminate()
net.stop()
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line M-E Redo


GNU nano 2.9.3
cpu.py
net.start()
net.pingAll()
hosts = [net.getNodeByName(h) for h in topo.hosts()]
client, server = hosts[0], hosts[-1]
info('*** Starting iperf with %d%% of CPU allocated to hosts\n' %
(100.0 * cpu))
# We measure at the server because it doesn't include
# the client's buffer fill rate
popen = server.popen('iperf -yc -s -p 5001')
waitListening(client, server, 5001)
# ignore empty result from waitListening/telnet
popen.stdout.readline()
client.cmd('iperf -yc -t %s -c %s' % (seconds, server.IP()))
result = decode(popen.stdout.readline()).split(',')
bps = float(result[-1])
popen.terminate()
net.stop()
updated = results.get(sched, [])
updated += [(cpu, bps)]
results[sched] = updated
return results
def dump(results):
"Dump results"
fmt = '%s\t%s\t%s\n'
info('\n')
info(fmt % ('sched', 'cpu', 'received bits/sec'))
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line M-E Redo



CODE EXPLANATION:

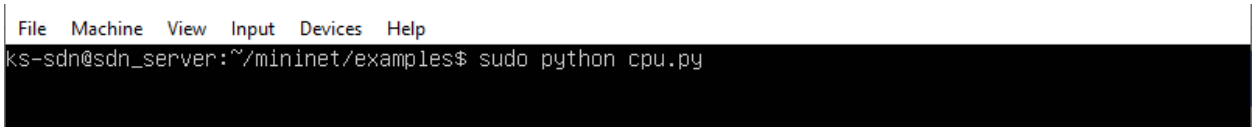
- The first statement is a string literal, this is the module documentation string, explaining what the app does, with the help of docstring tools, user can interactively browse through the code.
- The six import statements are used to import **Mininet, CPULimitedHost, TreeTopo, custom, waitListening, decode, setLogLevel, info,** and **cleanup** modules from their parent modules respectively.
- **def** statement is used to define bwtest function that instantiates TreeTopo class with local variables depth = 1, and fanout = 2, assigned an empty dictionary to variable results
    - Used nested **for** loop to iterates two strings ‘rt’, and ‘cfs’
        - Used **try** statement to handle exception within the inner for loop that iterates cpuLimits, called a custom functions with for arguments where cpu is divided by 2 (cpu=.5cpu)
        - The try clause is executed:
        - If no exception occurs, the except clause is skipped, and the code continues

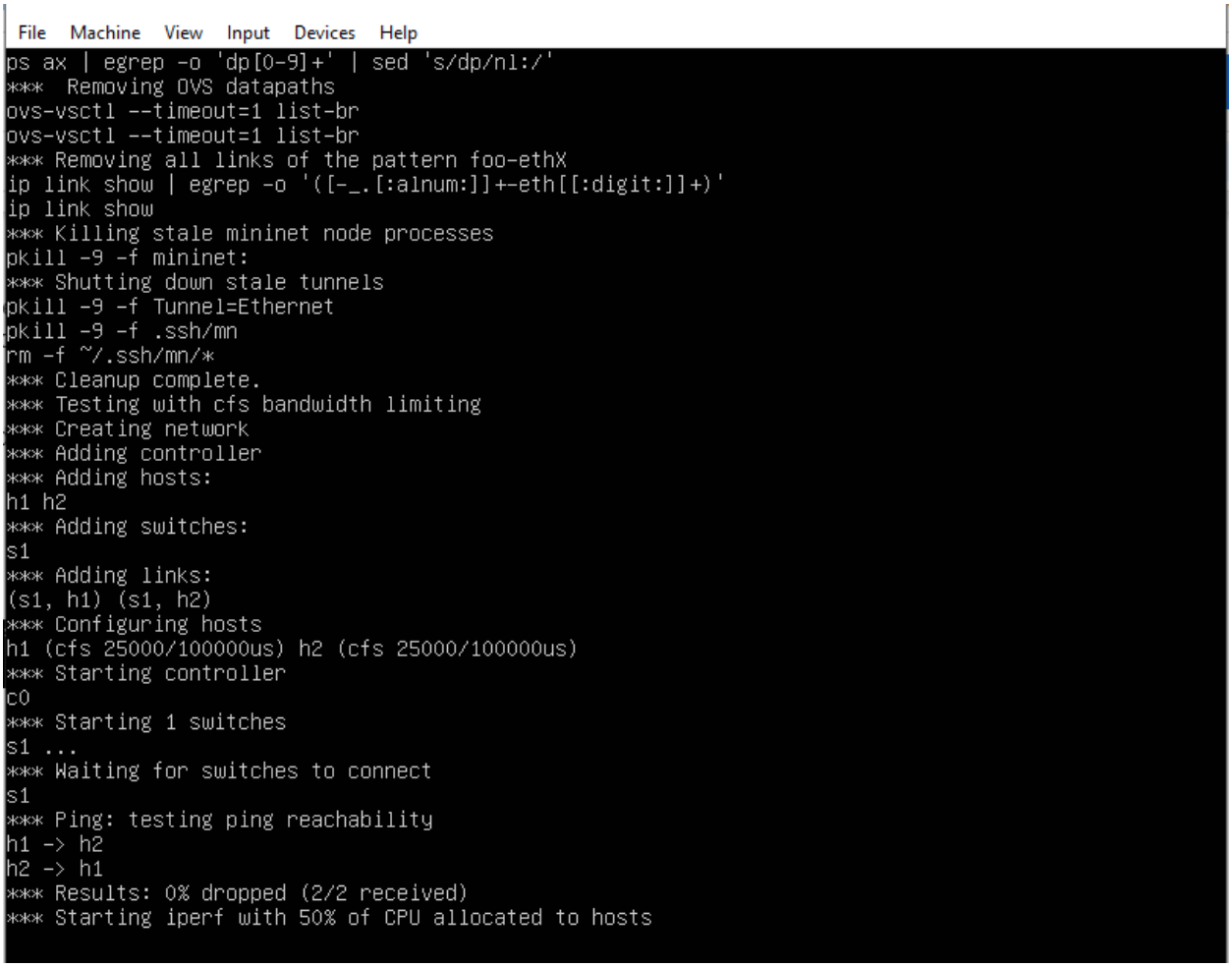
    - If exception occurs, the except clause is executed, and finished
    - **net.start()** start controller(s), switch(es)
    - **net.pingAll()** ping all hosts
    - A list of host names is assigned to hosts
    - **hosts[0]** is assigned to client, and **hosts[-1]** is assigned server
    - Used **iperf** to measure at the server and assigned to popen
    - Called client **cmd()** method with **iperf** command as argument
    - Formated the **popen** standard output
    - Returned **results**

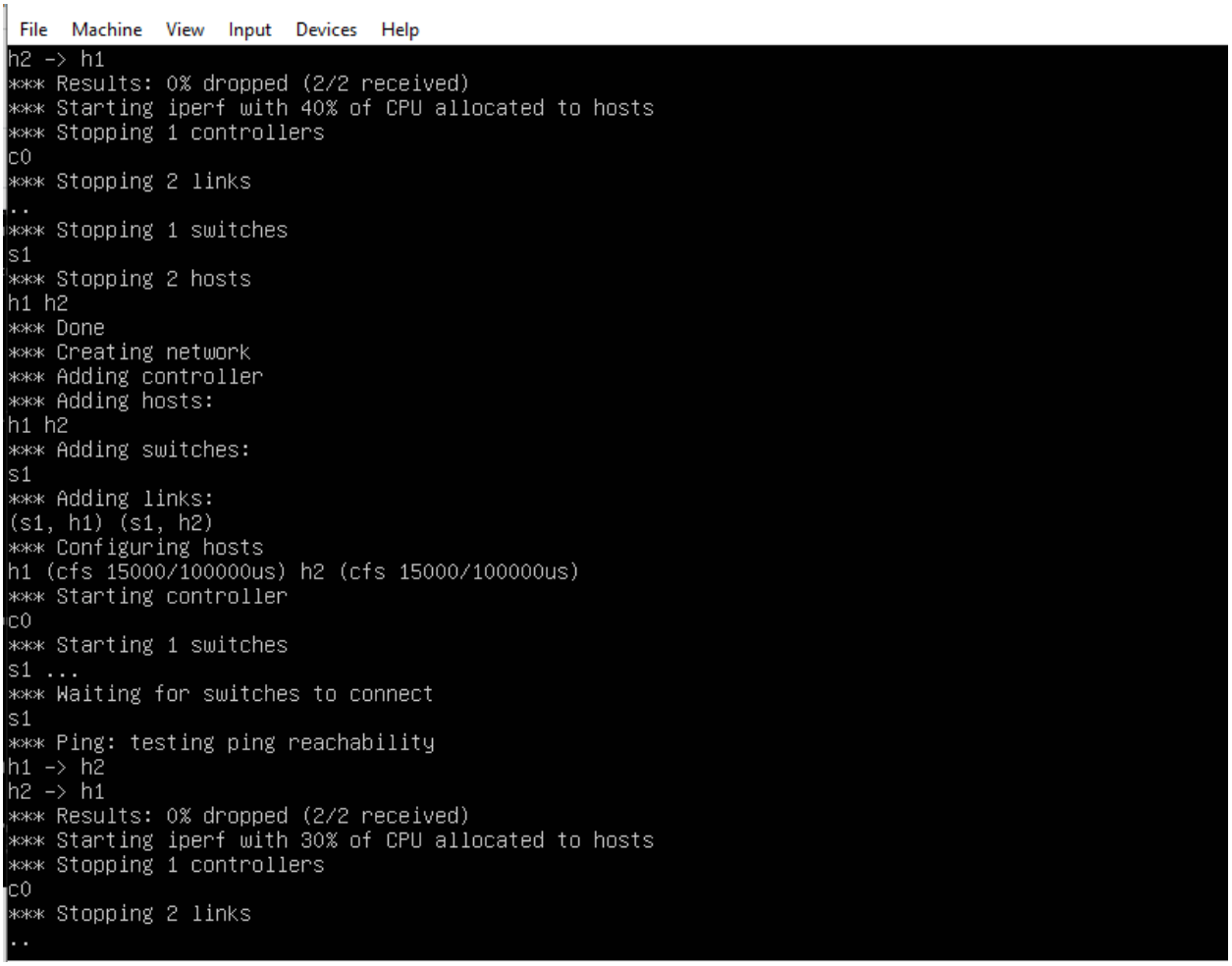
- **def** statement is used to define a dump function that is used to dump the results return in the bwtest function.

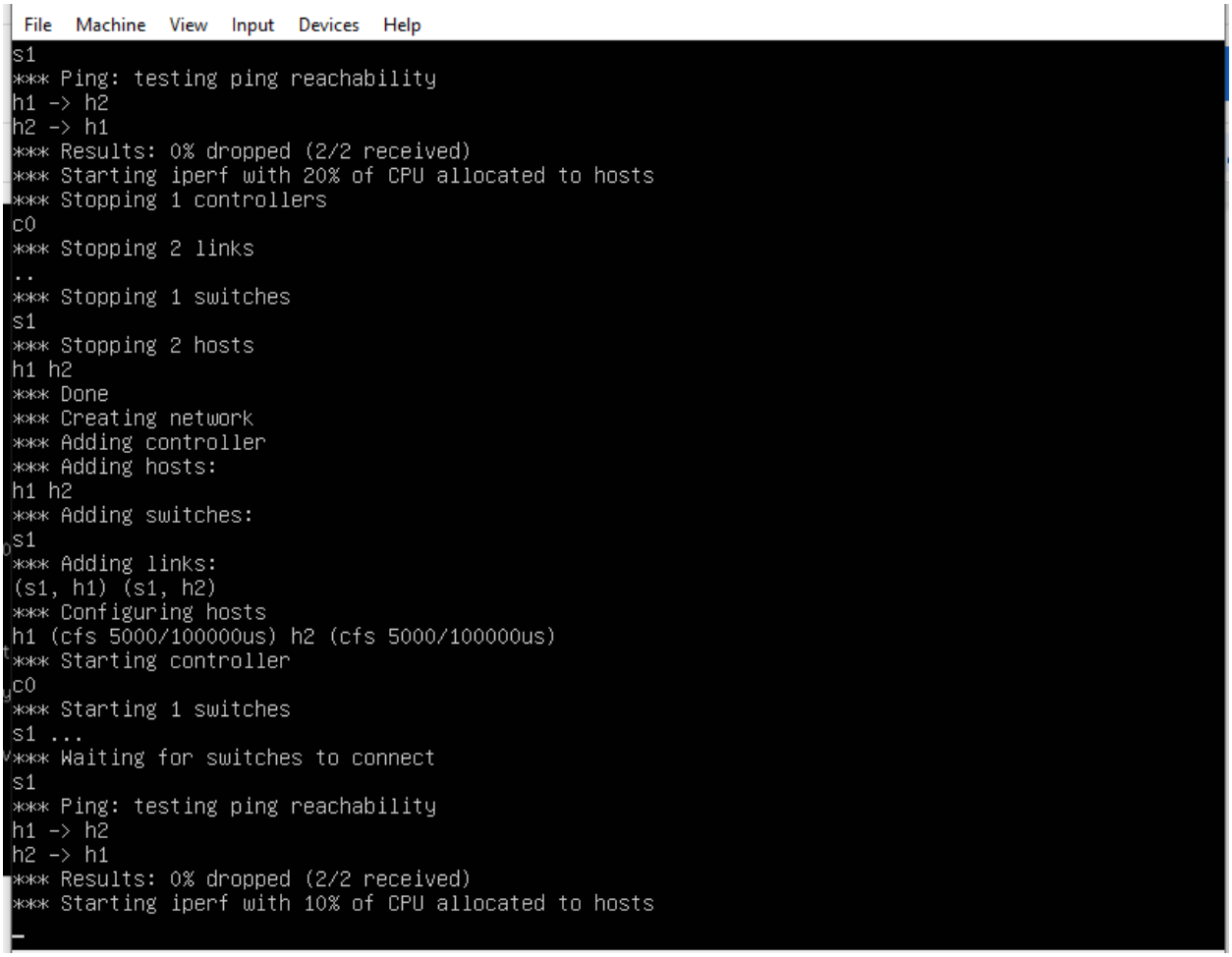
- The **if __name__ = “ __main__”**: statement makes the file usable as a script and importable module.

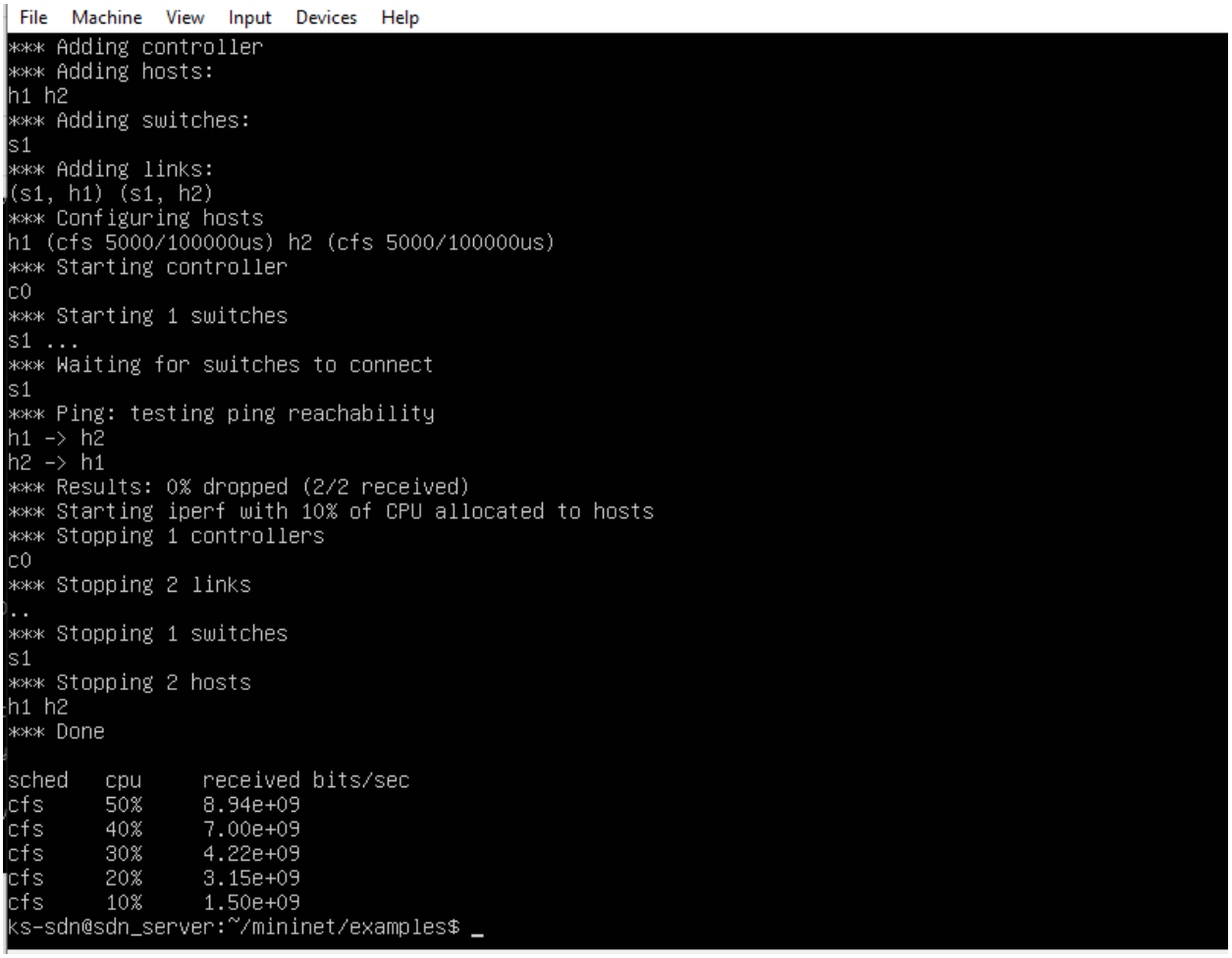
RESULT:




File Machine View Input Devices Help
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
*** 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 25000/100000us) h2 (cfs 25000/100000us)
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Starting iperf with 50% of CPU allocated to hosts

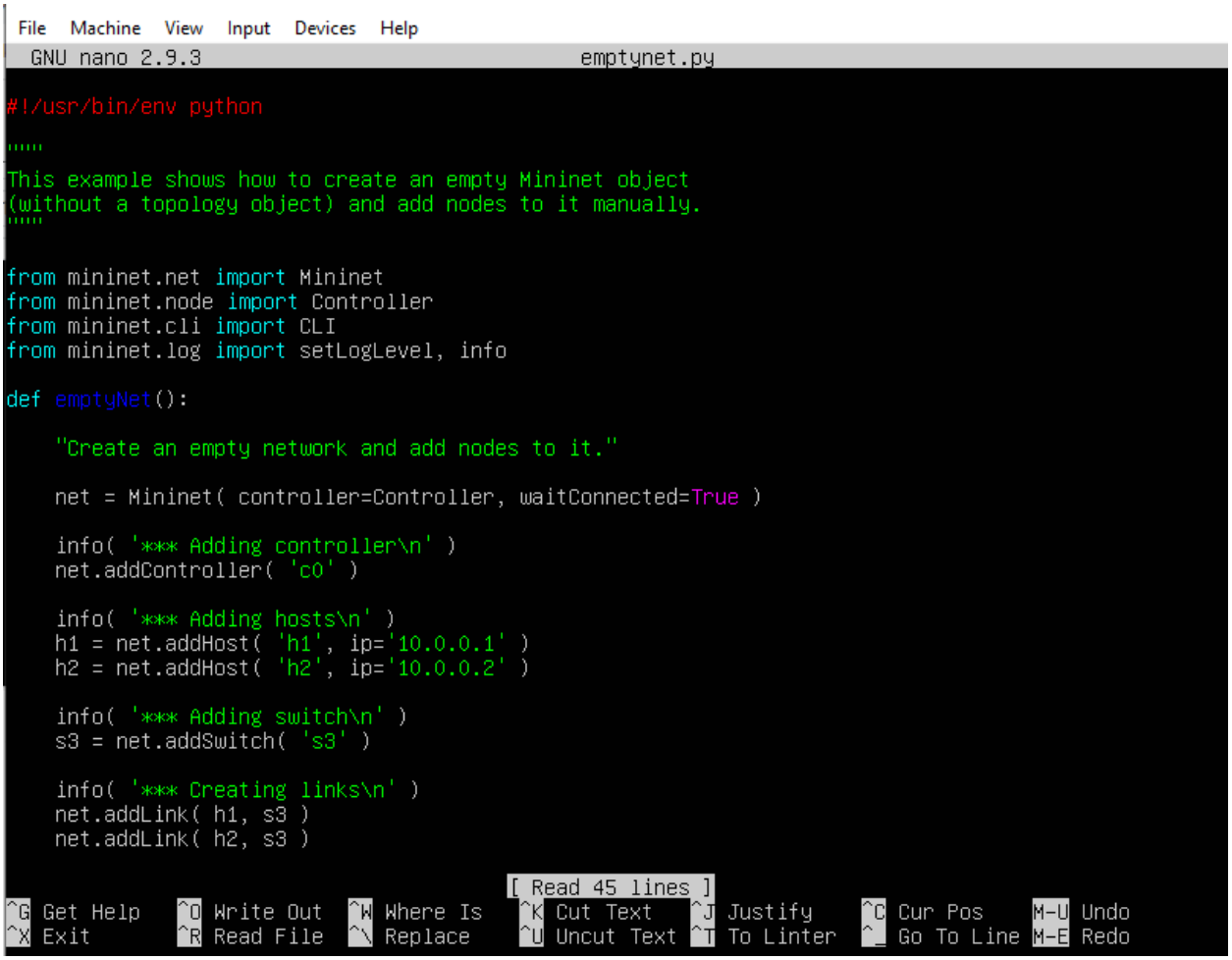

File Machine View Input Devices Help
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Starting iperf with 40% of CPU allocated to hosts
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2)
*** Configuring hosts
h1 (cfs 15000/100000us) h2 (cfs 15000/100000us)
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Starting iperf with 30% of CPU allocated to hosts
*** Stopping 1 controllers
c0
*** Stopping 2 links
..

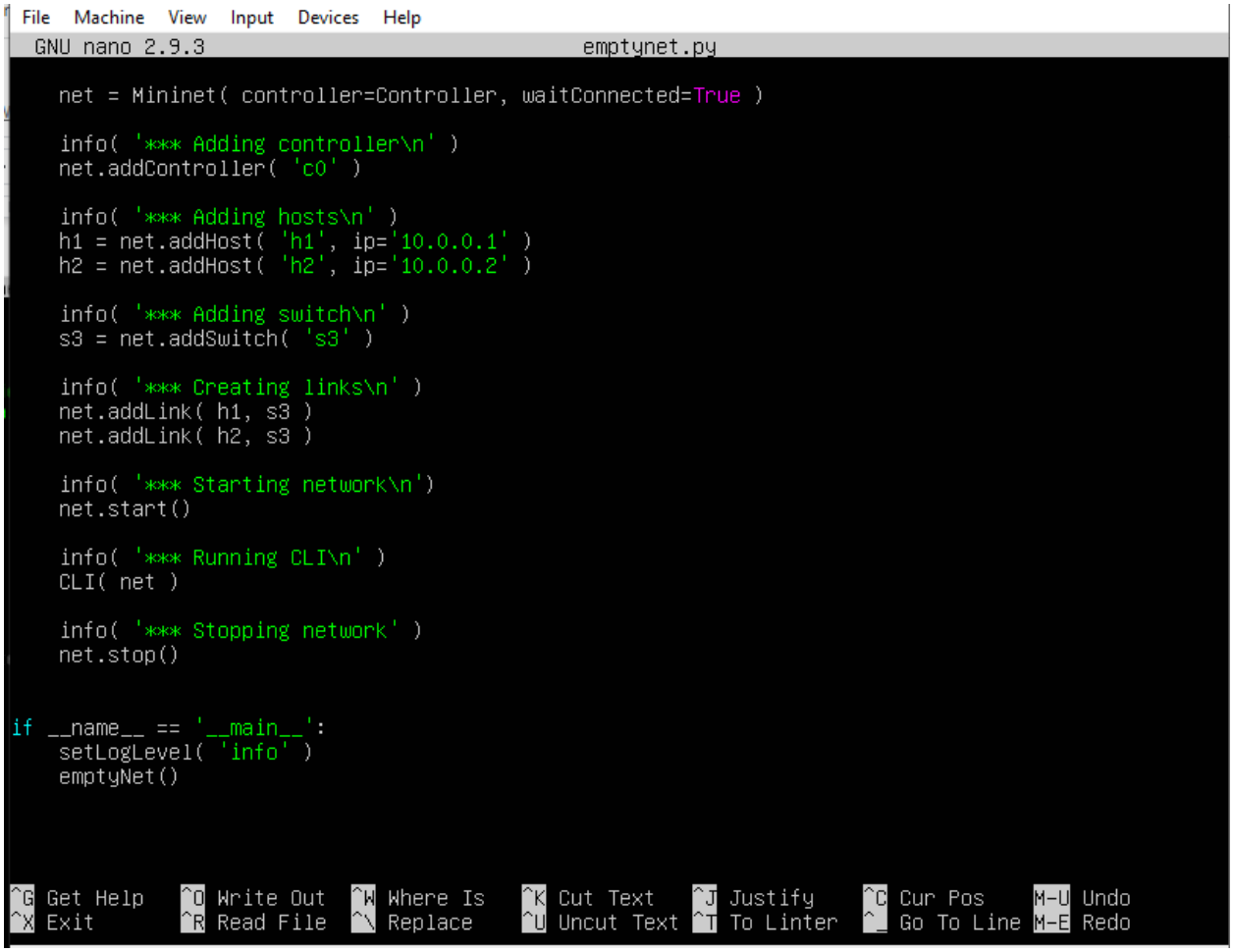

File Machine View Input Devices Help
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Starting iperf with 20% of CPU allocated to hosts
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2)
*** Configuring hosts
h1 (cfs 5000/100000us) h2 (cfs 5000/100000us)
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Starting iperf with 10% of CPU allocated to hosts



Summary: The API display the cpu usage and the bandwidth.

API - 4


File Machine View Input Devices Help
GNU nano 2.9.3 emptynet.py
#!/usr/bin/env python
"""
This example shows how to create an empty Mininet object
(without a topology object) and add nodes to it manually.
"""
from mininet.net import Mininet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.log import setLogLevel, info
def emptyNet():
"Create an empty network and add nodes to it."
net = Mininet(controller=Controller, waitConnected=True)
info('*** Adding controller\n')
net.addController('c0')
info('*** Adding hosts\n')
h1 = net.addHost('h1', ip='10.0.0.1')
h2 = net.addHost('h2', ip='10.0.0.2')
info('*** Adding switch\n')
s3 = net.addSwitch('s3')
info('*** Creating links\n')
net.addLink(h1, s3)
net.addLink(h2, s3)
[Read 45 lines]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line M-E Redo



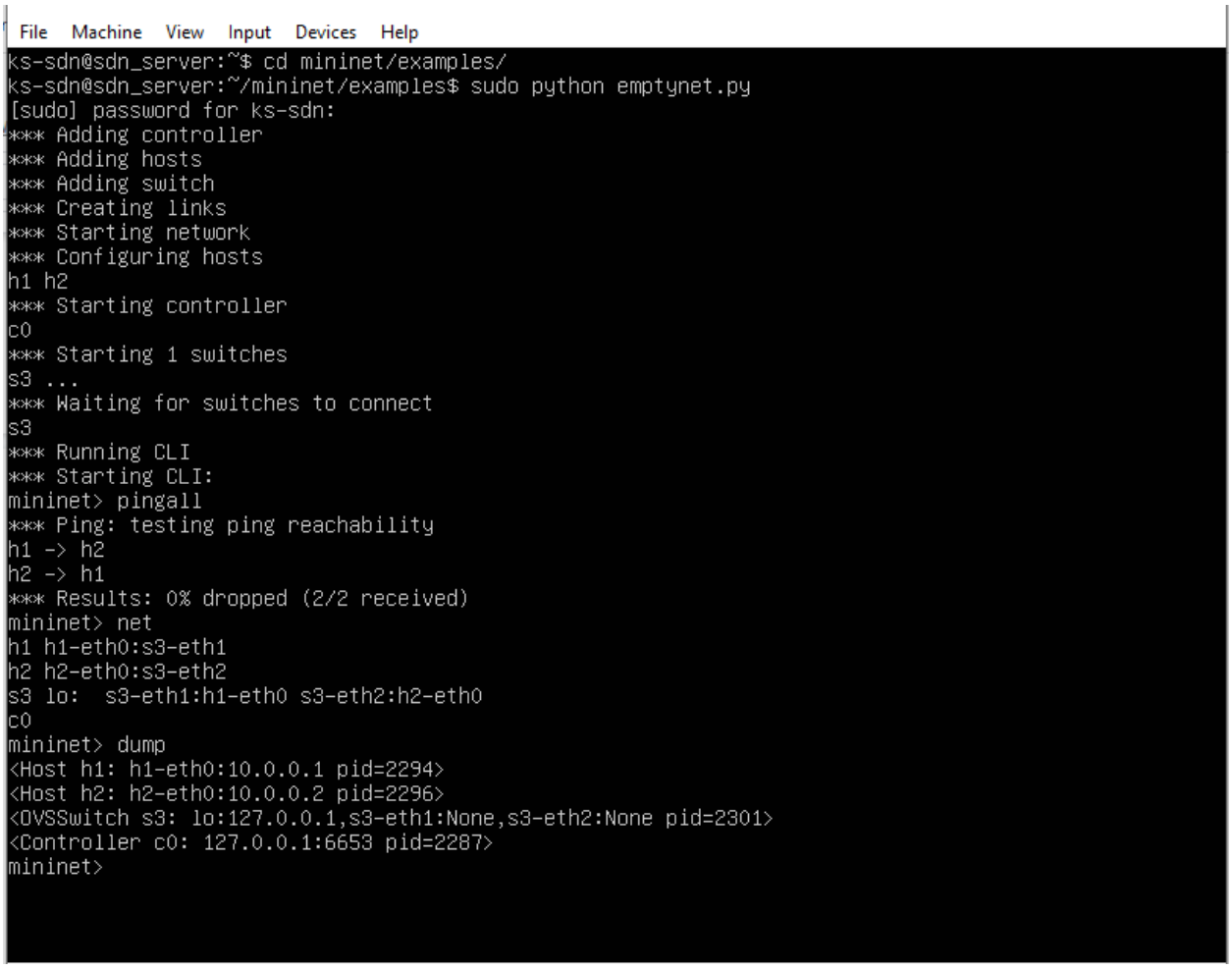
CODE EXPLANATION

- The first statement is a string literal, this is the module documentation string, explaining what the app does, with the help of docstring tools, user can interactively browse through the code.

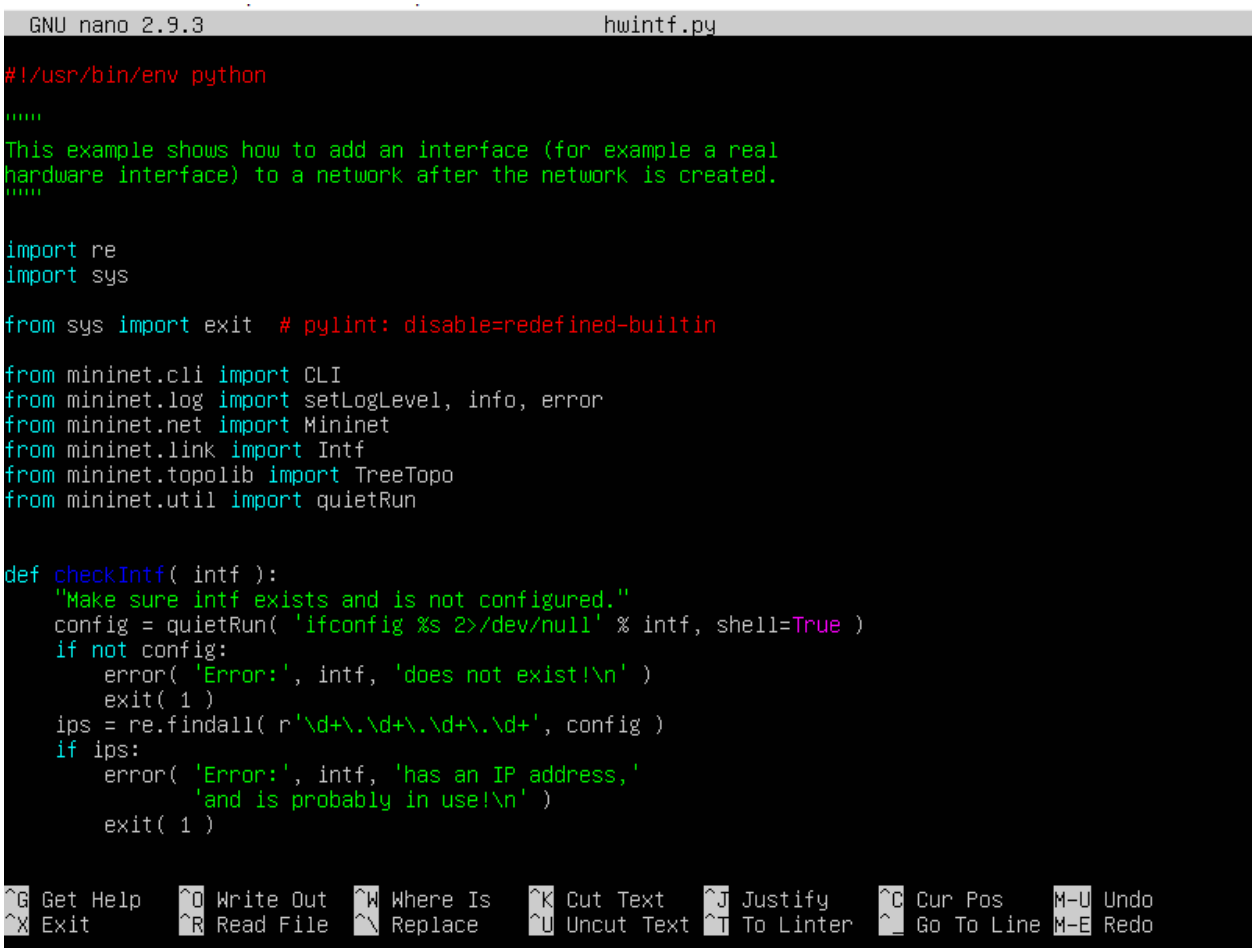
- The four import statements are used to **import Mininet, Controller, setLogLevel, Info, and CLI modules** from their parent modules respectively.
- The def statement defined a function named emptyNet which was used to:
    - instance of **Mininet** class was instantiated and assigned to a variable **net**
    - Reference **addController** method with the instance object **net**,
    - Reference **addHost** method with the instance object **net** to add hosts **h1, h2**
    - Reference **addSwitch** method with the instance object **net** to add switch **s3**
    - Reference **addLink** method with the instance object **net** to add links **(h1 - s3, h2 - s3)**
    - Start the controller, switch, and hosts with **net.start()**
    - Stop controller, switch, and hosts with **net.stop()**
    -

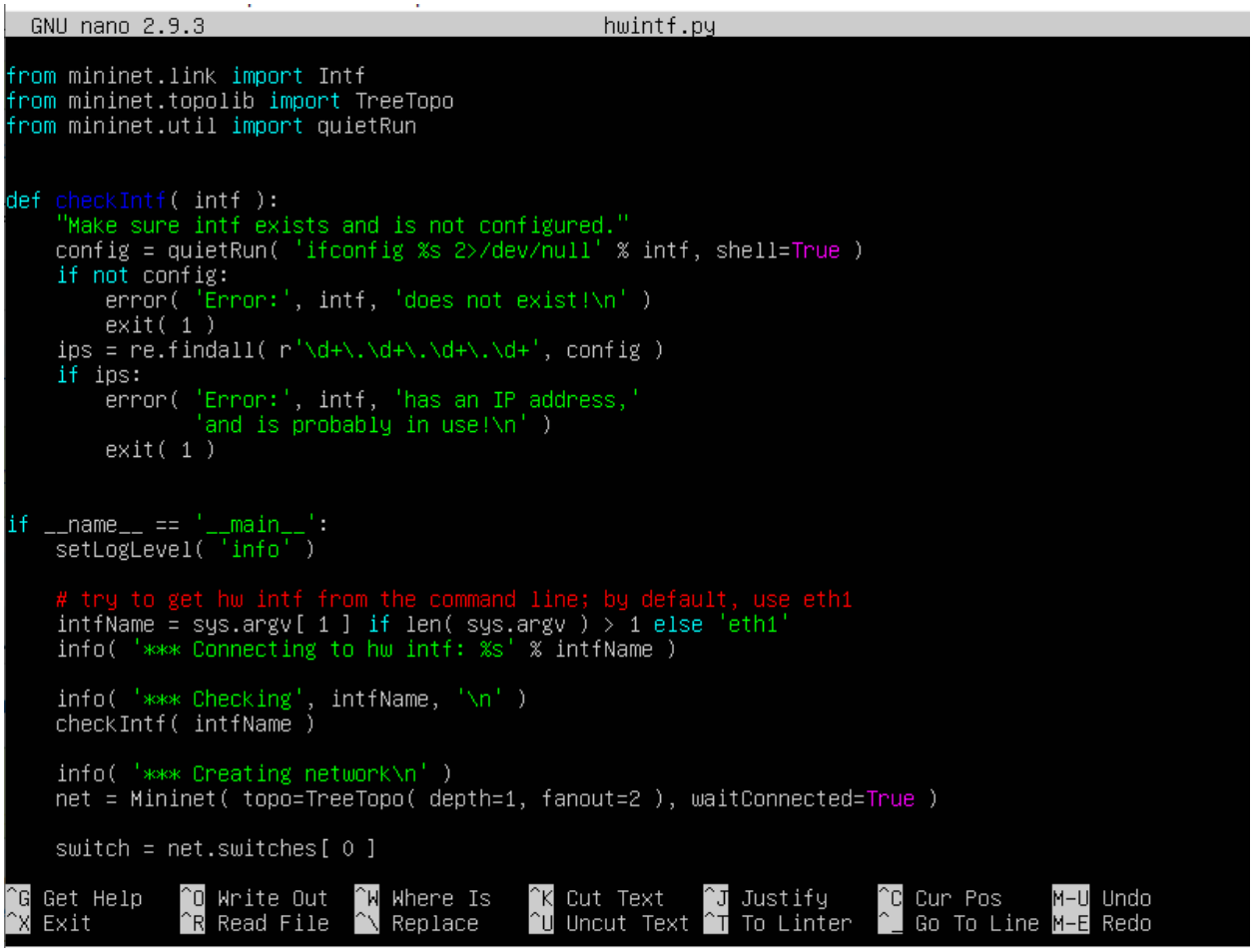
- The **if __name__ = “ __main__”:** statement makes the file usable as a script and importable module.

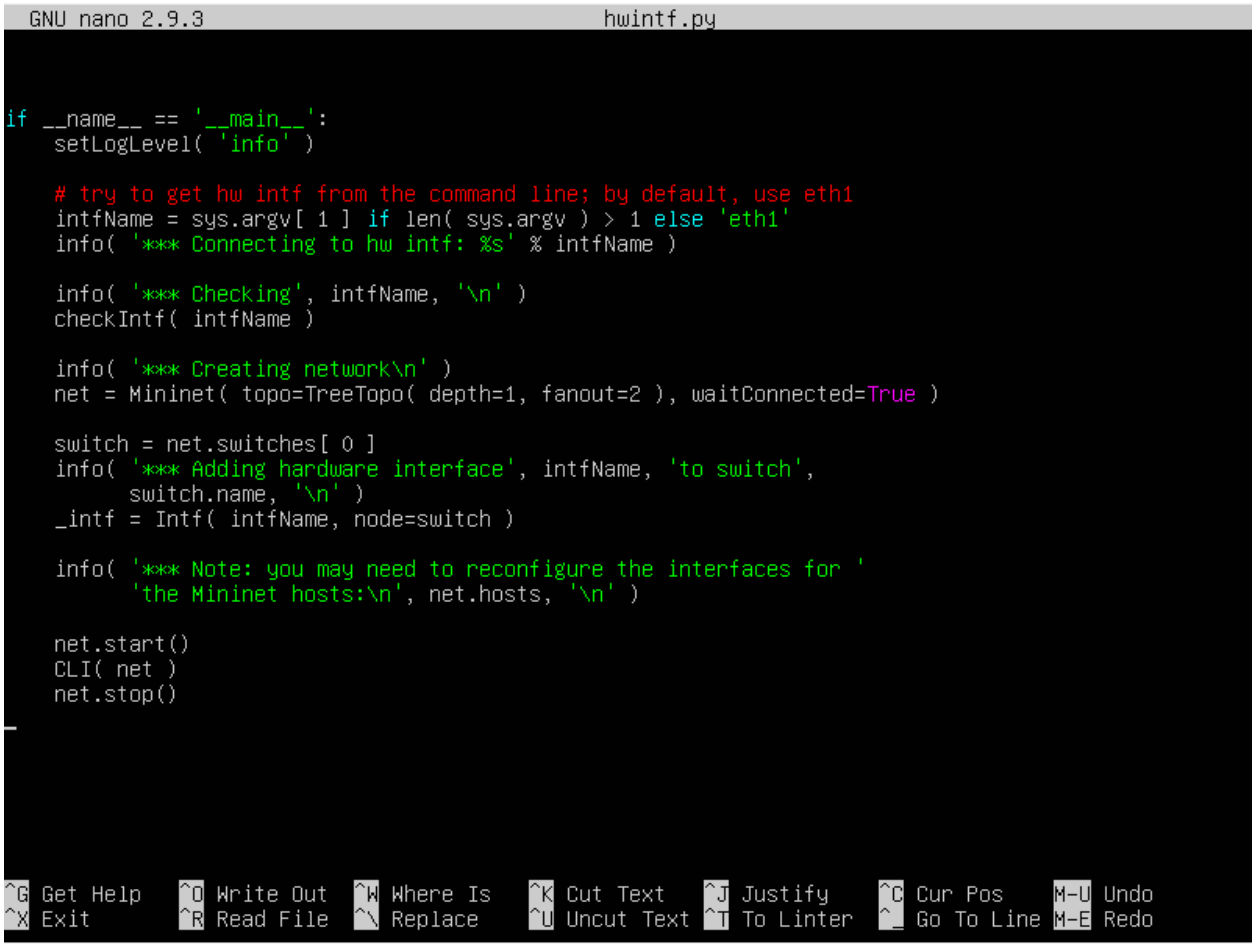
RESULT:



APP - 5


GNU nano 2.9.3 hwintf.py
#!/usr/bin/env python
"""
This example shows how to add an interface (for example a real
hardware interface) to a network after the network is created.
"""
import re
import sys
from sys import exit # pylint: disable=redefined-builtin
from mininet.cli import CLI
from mininet.log import setLogLevel, info, error
from mininet.net import Mininet
from mininet.link import Intf
from mininet.topolib import TreeTopo
from mininet.util import quietRun
def checkIntf(intf):
"Make sure intf exists and is not configured."
config = quietRun('ifconfig %s 2>/dev/null' % intf, shell=True)
if not config:
error('Error:', intf, 'does not exist!\n')
exit(1)
ips = re.findall(r'\d+\.\d+\.\d+\.\d+', config)
if ips:
error('Error:', intf, 'has an IP address,'
'and is probably in use!\n')
exit(1)
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line M-E Redo


GNU nano 2.9.3 hwintf.py
from mininet.link import Intf
from mininet.topolib import TreeTopo
from mininet.util import quietRun
def checkIntf(intf):
"Make sure intf exists and is not configured."
config = quietRun('ifconfig %s 2>/dev/null' % intf, shell=True)
if not config:
error('Error:', intf, 'does not exist!\n')
exit(1)
ips = re.findall(r'\d+\.\d+\.\d+\.\d+', config)
if ips:
error('Error:', intf, 'has an IP address,'
'and is probably in use!\n')
exit(1)
if __name__ == '__main__':
setLogLevel('info')
# try to get hw intf from the command line; by default, use eth1
intfName = sys.argv[1] if len(sys.argv) > 1 else 'eth1'
info('*** Connecting to hw intf: %s' % intfName)
info('*** Checking', intfName, '\n')
checkIntf(intfName)
info('*** Creating network\n')
net = Mininet(topo=TreeTopo(depth=1, fanout=2), waitConnected=True)
switch = net.switches[0]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^_ Go To Line M-E Redo



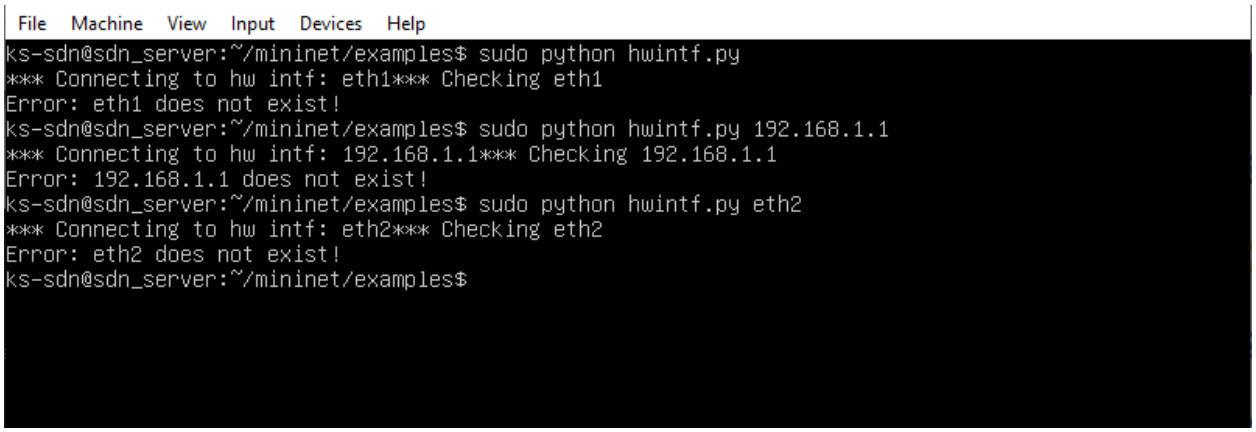
CODE EXPLANATION

- The first statement is a string literal, this is the module documentation string, explaining what the app does, with the help of docstring tools, user can interactively browse through the code.

- The nine **import** statements are used to import **re, sys, CLI, setLogLevel, Info, error, Mininet, Intf, TreeTopo, quietRun** from their parent modules respectively.

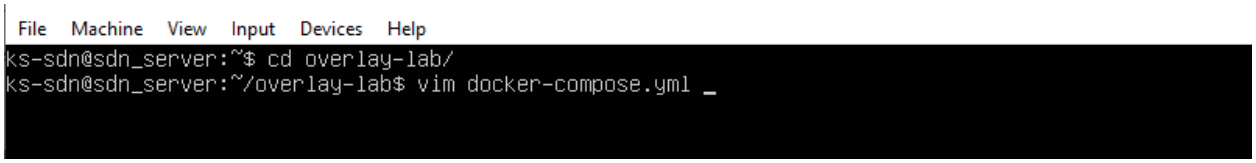
- The **def** statement defined **checkIntf** method, that check if an interface exist and configured by:
    - Calling a **quietRun** method that performed an **ifconfig** command, redirect the **stderr** to a null device, and assigned to variable **config**
    - Used **if not** statement to verify **config** which raised an **exception** and exit with **code 1:** operation not permitted.
    - Called the **findall** method and assigned return value to **ips**
    - Used **if** statement to check **ips**, raised **exception** and exit with **code 1**, operation not permitted.

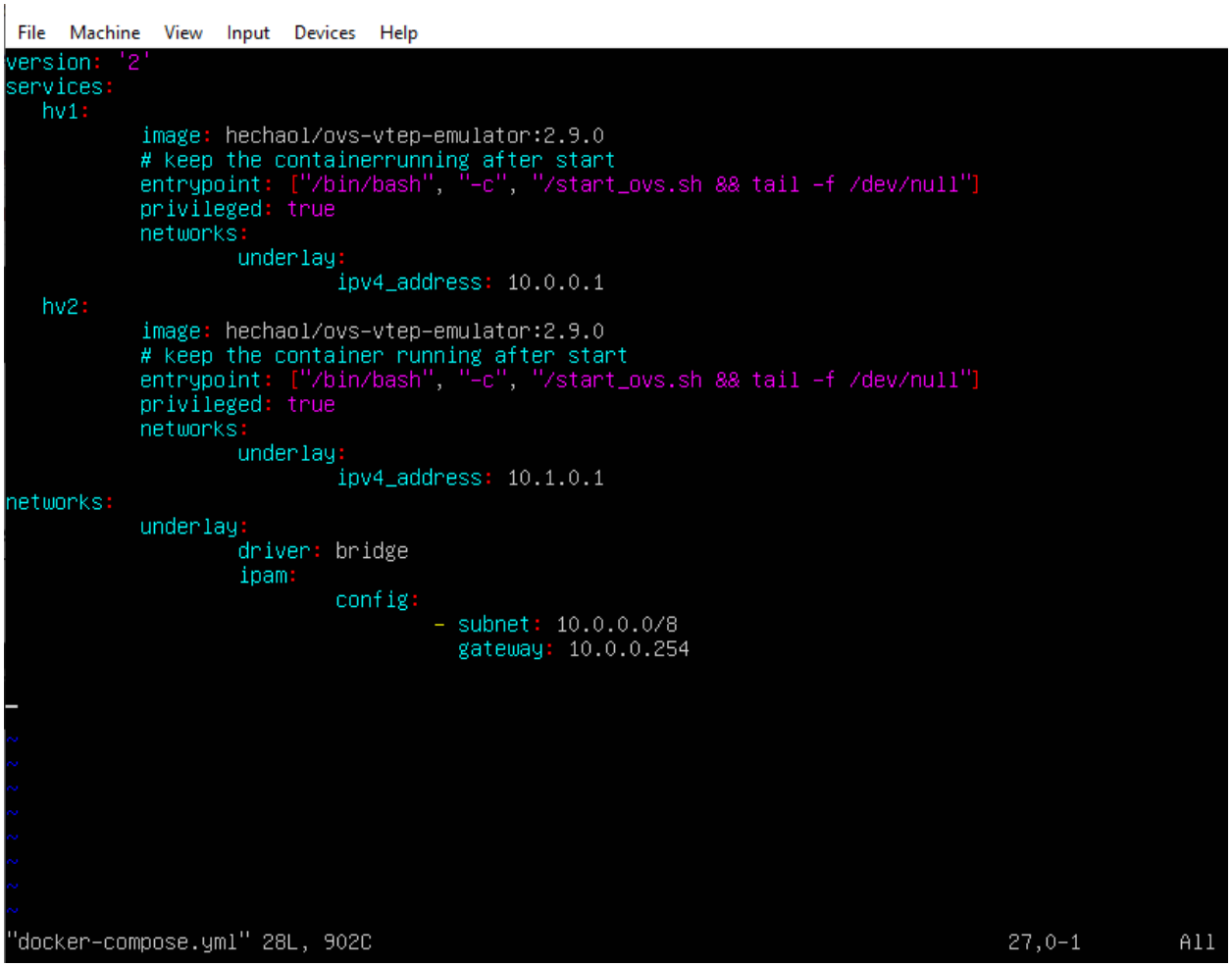
- The **if __name__ = “ __main__”:** statement makes the file usable as a script and importable module.
    - Used **sys.argv** to accept input from the command line, set the default to **eth1**, assigned to variable intfName
    - Called **checkIntf** method with **intfName** as argument
    - Instantiates **Mininet** class and assigned it to variable **net**
    - Reference the **Mininet** class method with **net.switches[0]**
    - Start mininet network
    - **CLI(net)** runs a batch or interactive mode for the network.
    - The **net.stop()** stop controllers, switches, and hosts.



Overlays: OpenVswitch + VXLAN

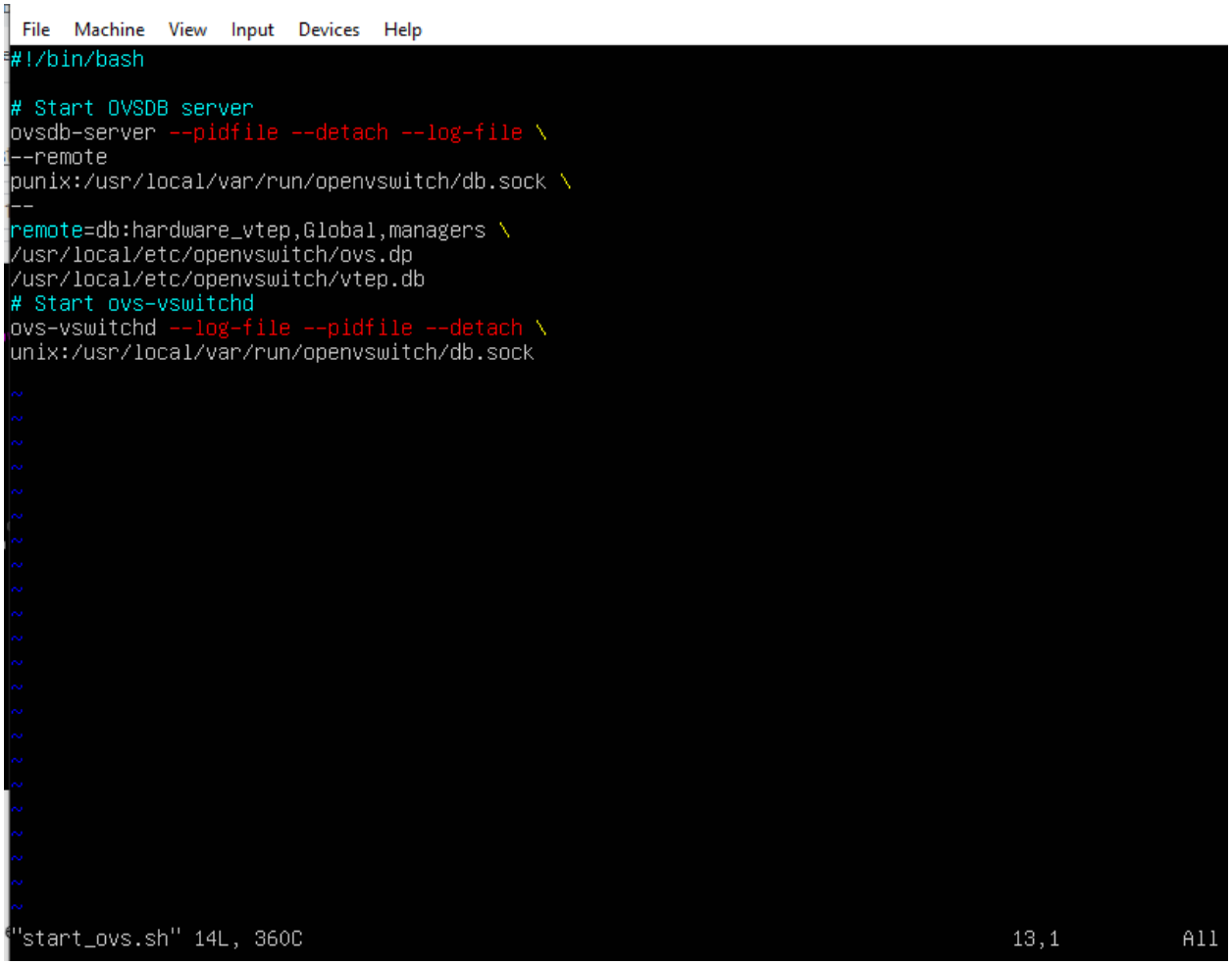
- All required software already install in iteration 1
- Create a directory named **overlay-lab** where all files for the project will be kept
- Create a **.yml file**, name it **docker-compose.yml** with the vi editor as shown below:



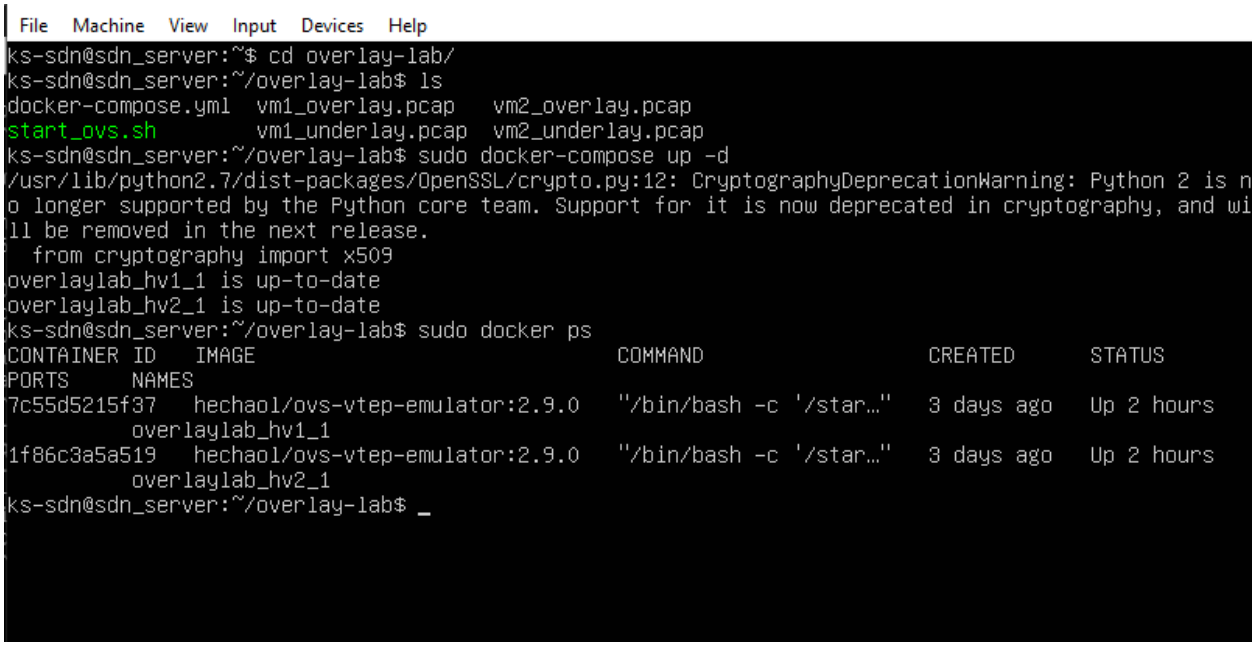


Create the VM OVS startup file:

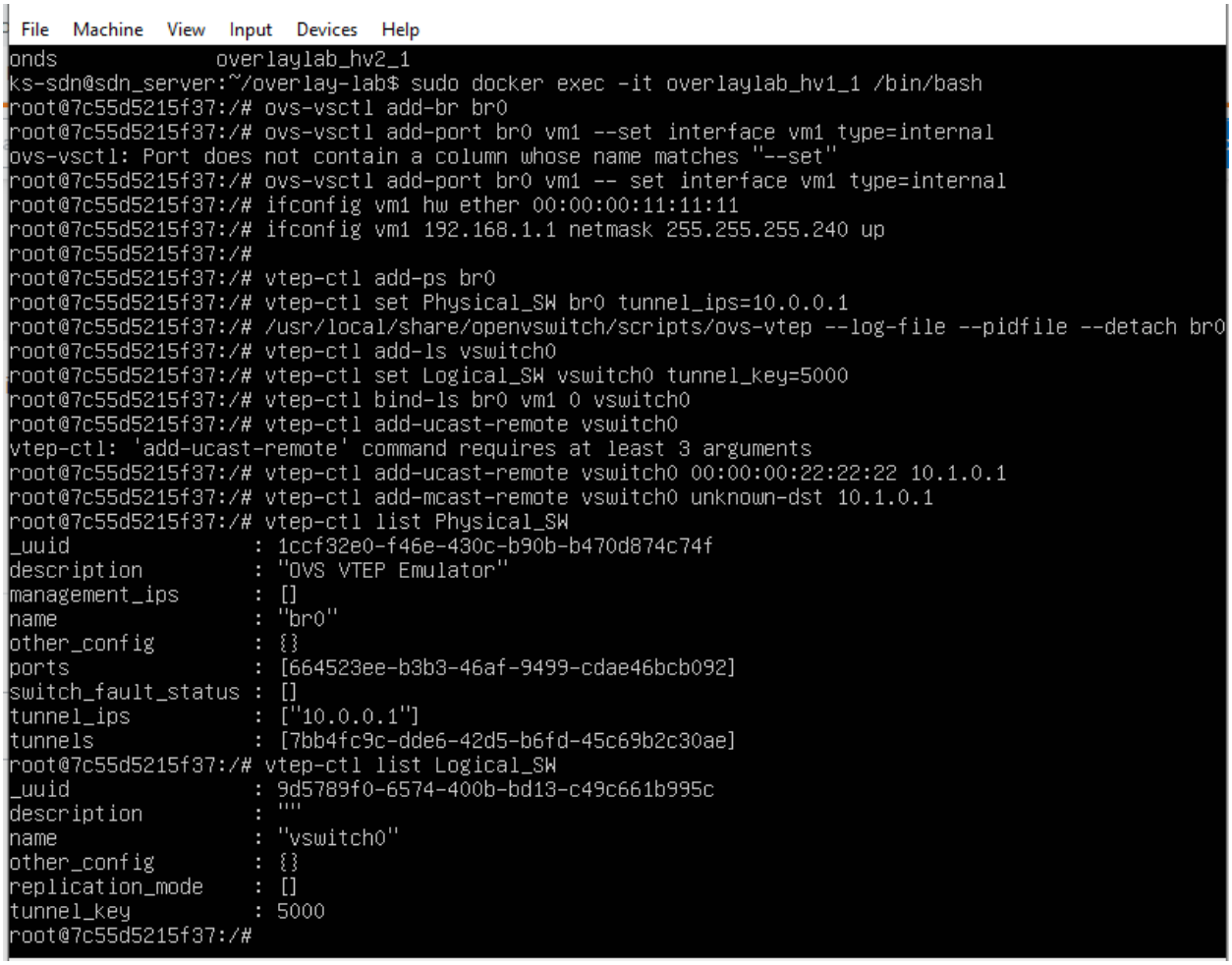
$ touch start_ovs.sh
$ sudo chmod +x start_ovs.sh # make it executable
$ vim start_ovs.sh


File Machine View Input Devices Help
#!/bin/bash
Start OVSDB server
ovsdb-server --pidfile --detach --log-file \
--remote
punix:/usr/local/var/run/openvswitch/db.sock \
--
remote=db:hardware_vtep,Global,managers \
/usr/local/etc/openvswitch/ovs.dp
/usr/local/etc/openvswitch/vtep.db
Start ovs-vswitchd
ovs-vswitchd --log-file --pidfile --detach \
unix:/usr/local/var/run/openvswitch/db.sock
"start_ovs.sh" 14L, 360C
13,1
All

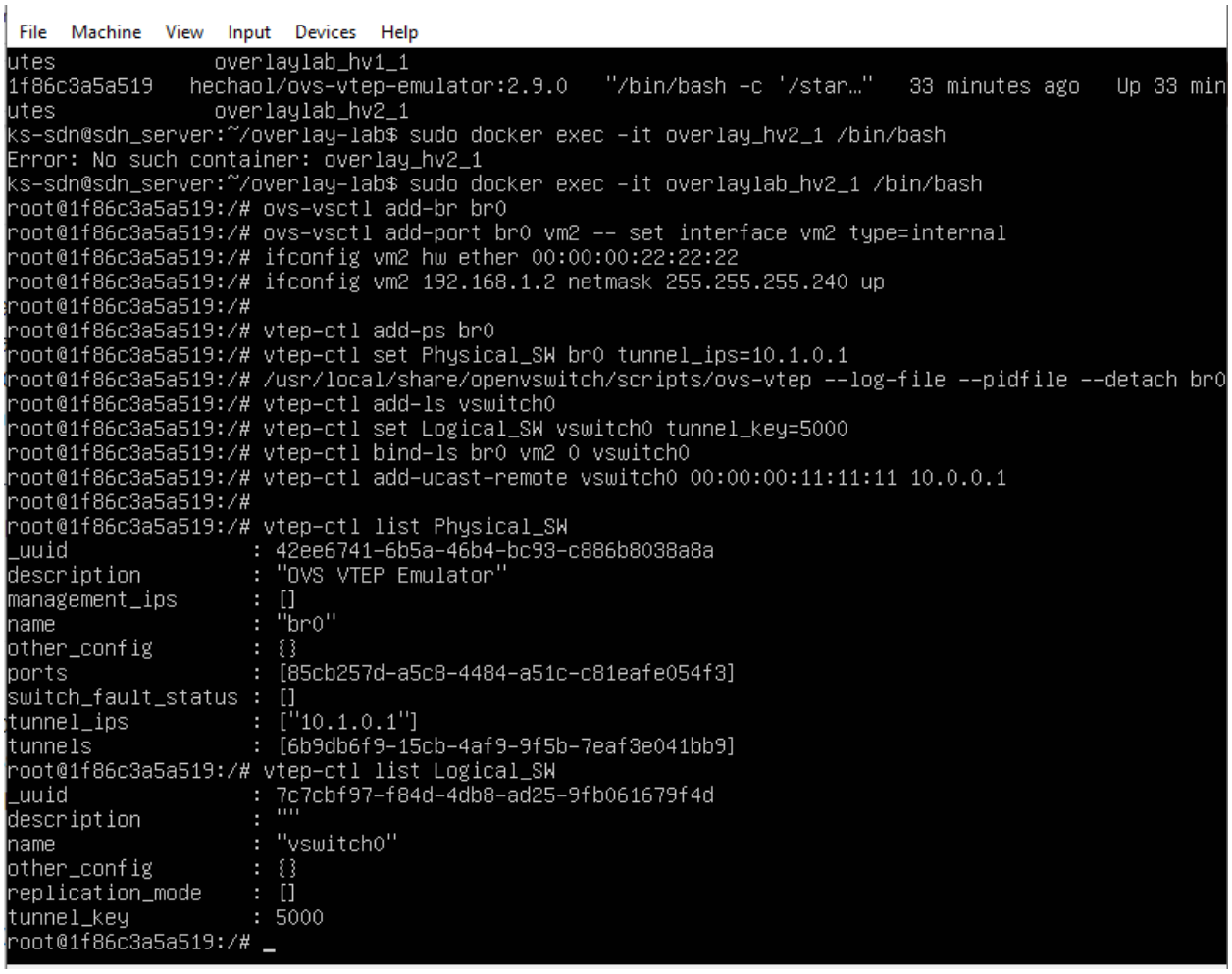
Docker UP



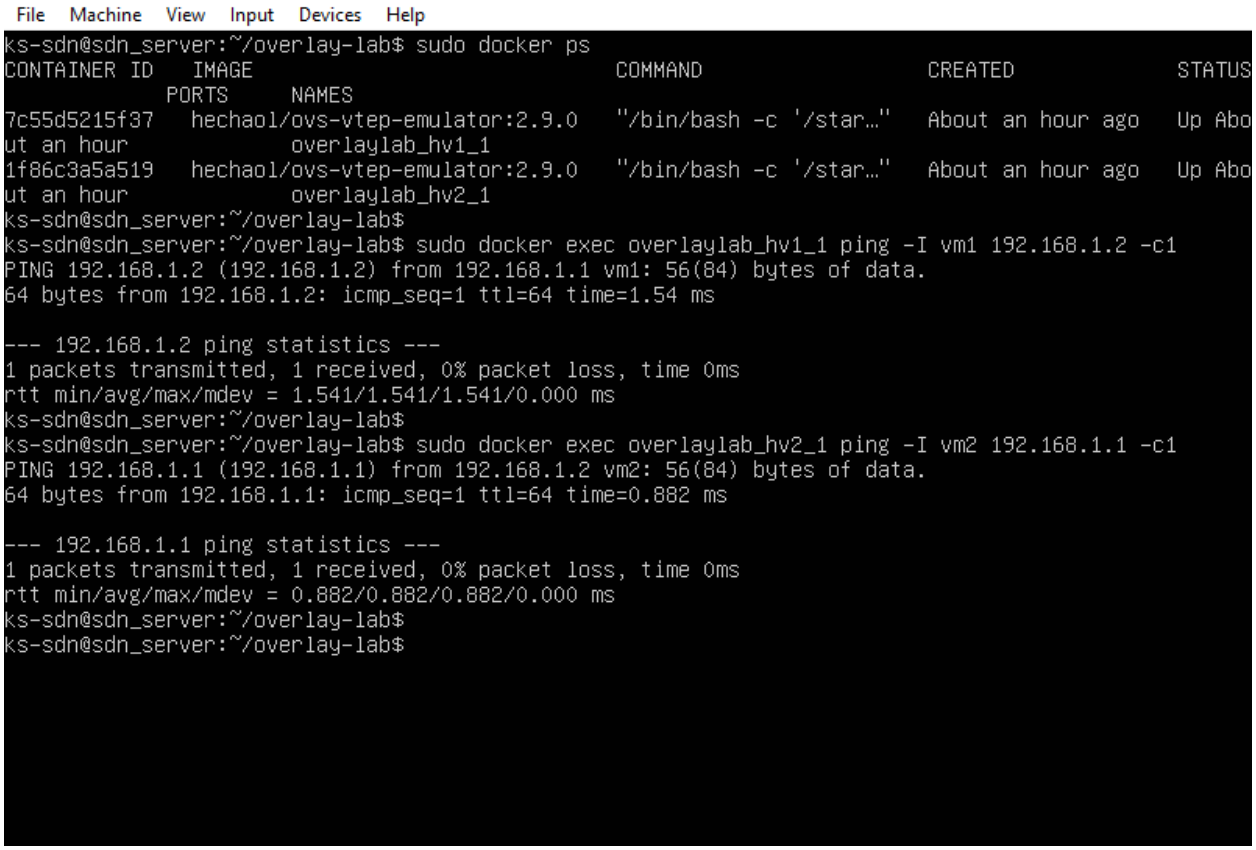
Configure Container 1, bridge and interfaces, local interface, and VTEP emulator:



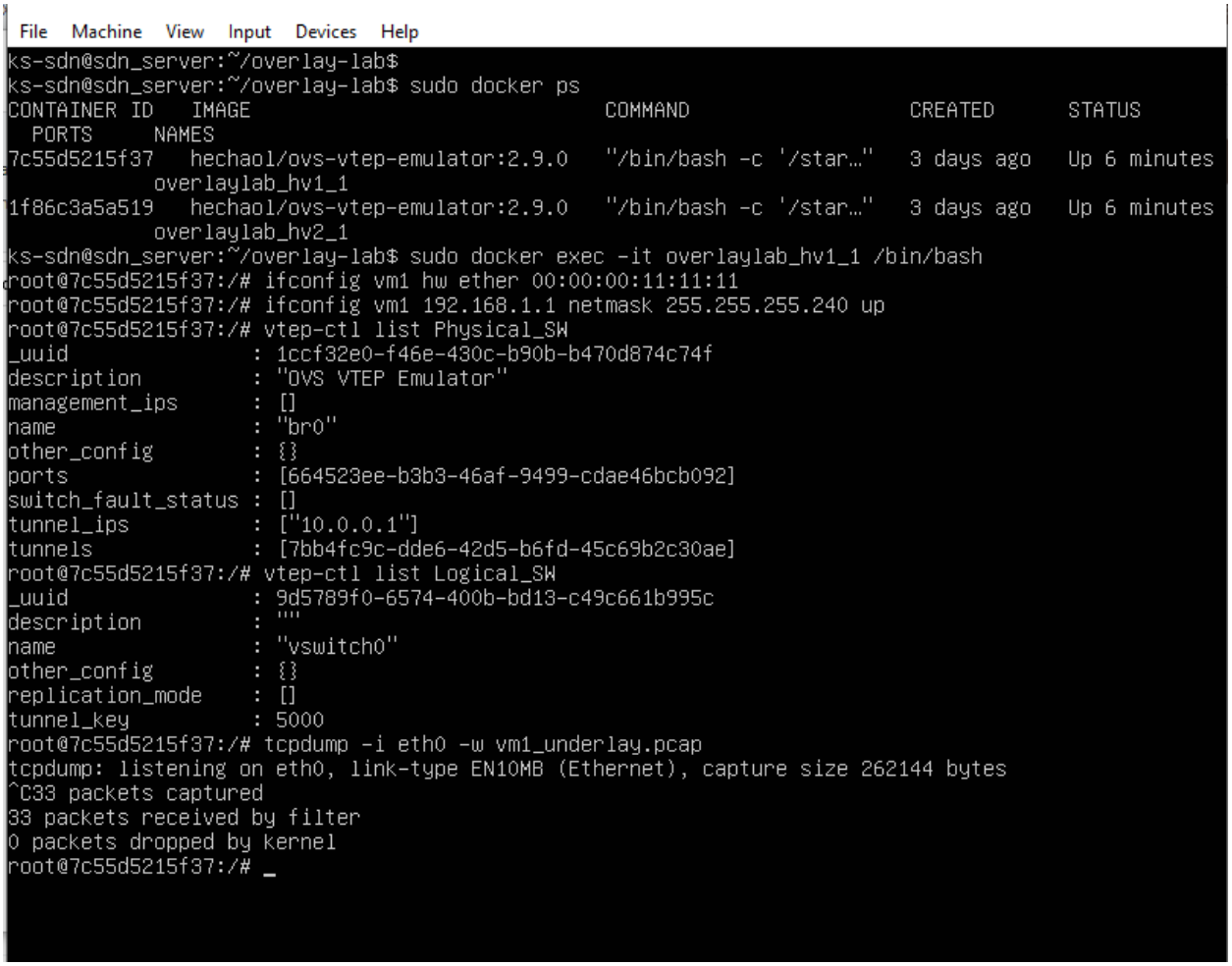
Configure Container 2, bridge and interfaces, local interface, and VTEP emulator.

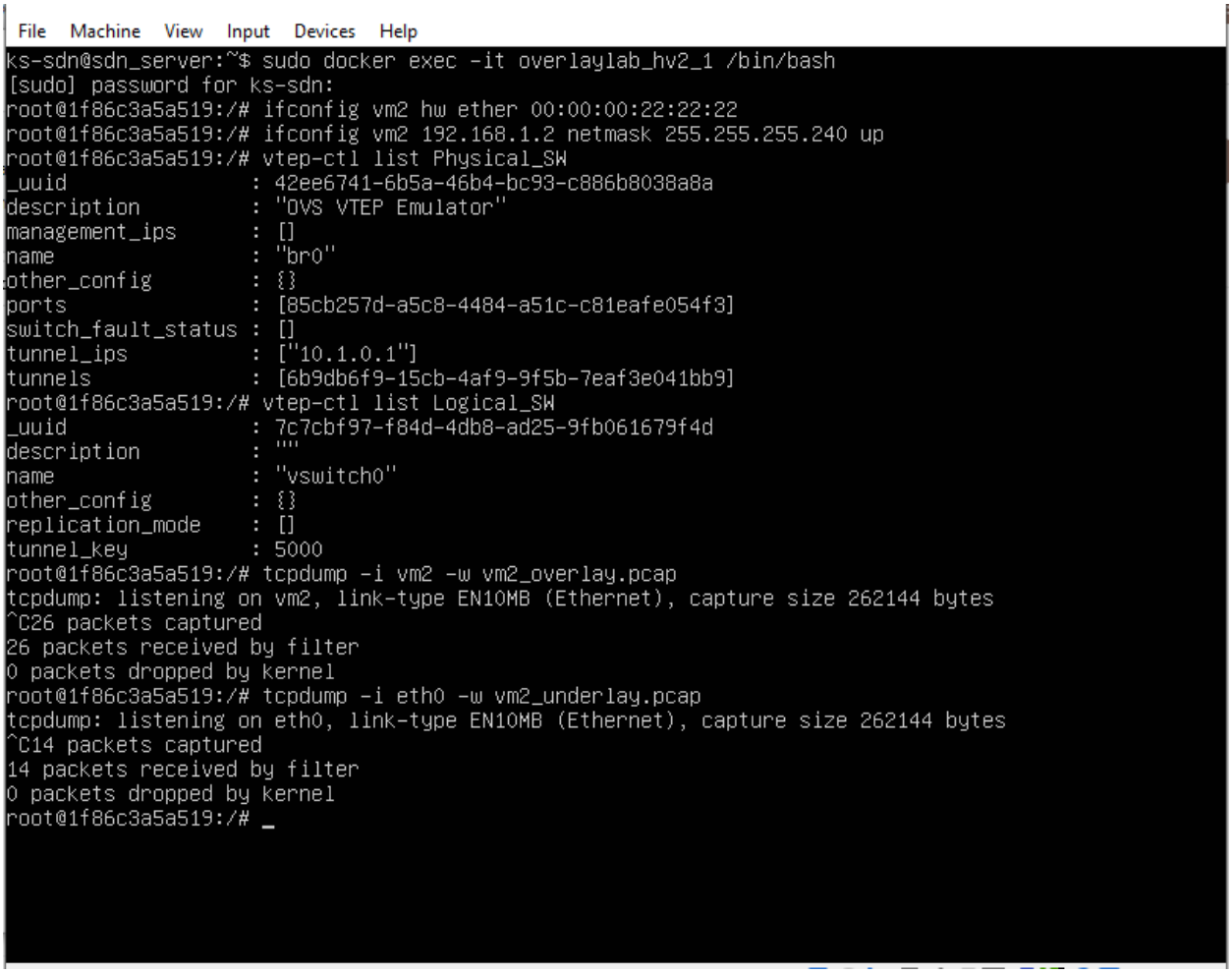


Testing Configurations:



Sniff traffics using tcpdump command:


File Machine View Input Devices Help
ks-sdn@sdn_server:~/overlay-lab$
ks-sdn@sdn_server:~/overlay-lab$ sudo docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
7c55d5215f37 hechaol/ovs-vtep-emulator:2.9.0 "/bin/bash -c '/star…" 3 days ago Up 6 minutes
overlaylab_hv1_1
1f86c3a5a519 hechaol/ovs-vtep-emulator:2.9.0 "/bin/bash -c '/star…" 3 days ago Up 6 minutes
overlaylab_hv2_1
ks-sdn@sdn_server:~/overlay-lab$ sudo docker exec -it overlaylab_hv1_1 /bin/bash
root@7c55d5215f37:/# ifconfig vm1 hw ether 00:00:00:11:11:11
root@7c55d5215f37:/# ifconfig vm1 192.168.1.1 netmask 255.255.255.240 up
root@7c55d5215f37:/# vtep-ctl list Physical_SW
_uuid : 1ccf32e0-f46e-430c-b90b-b470d874c74f
description : "OVS VTEP Emulator"
management_ips : []
name : "br0"
other_config : {}
ports : [664523ee-b3b3-46af-9499-cdae46bcb092]
switch_fault_status : []
tunnel_ips : ["10.0.0.1"]
tunnels : [7bb4fc9c-dde6-42d5-b6fd-45c69b2c30ae]
root@7c55d5215f37:/# vtep-ctl list Logical_SW
_uuid : 9d5789f0-6574-400b-bd13-c49c661b995c
description : ""
name : "vswitch0"
other_config : {}
replication_mode : []
tunnel_key : 5000
root@7c55d5215f37:/# tcpdump -i eth0 -w vm1_underlay.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C33 packets captured
33 packets received by filter
0 packets dropped by kernel
root@7c55d5215f37:/# _


File Machine View Input Devices Help
ks-sdn@sdn_server:~$ sudo docker exec -it overlaylab_hv2_1 /bin/bash
[sudo] password for ks-sdn:
root@1f86c3a5a519:/# ifconfig vm2 hw ether 00:00:00:22:22:22
root@1f86c3a5a519:/# ifconfig vm2 192.168.1.2 netmask 255.255.255.240 up
root@1f86c3a5a519:/# vtep-ctl list Physical_SW
_uuid : 42ee6741-6b5a-46b4-bc93-c886b8038a8a
description : "OVS VTEP Emulator"
management_ips : []
name : "br0"
other_config : {}
ports : [85cb257d-a5c8-4484-a51c-c81eafe054f3]
switch_fault_status : []
tunnel_ips : ["10.1.0.1"]
tunnels : [6b9db6f9-15cb-4af9-9f5b-7eaf3e041bb9]
root@1f86c3a5a519:/# vtep-ctl list Logical_SW
_uuid : 7c7cbf97-f84d-4db8-ad25-9fb061679f4d
description : ""
name : "vswitch0"
other_config : {}
replication_mode : []
tunnel_key : 5000
root@1f86c3a5a519:/# tcpdump -i vm2 -w vm2_overlay.pcap
tcpdump: listening on vm2, link-type EN10MB (Ethernet), capture size 262144 bytes
^C26 packets captured
26 packets received by filter
0 packets dropped by kernel
root@1f86c3a5a519:/# tcpdump -i eth0 -w vm2_underlay.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C14 packets captured
14 packets received by filter
0 packets dropped by kernel
root@1f86c3a5a519:/# _

```
File  Machine  View  Input  Devices  Help
ks-sdn@sdn_server:~/overlay-lab$
ks-sdn@sdn_server:~/overlay-lab$ sudo docker exec overlaylab_hv1_1 ping -I vm1 192.168.1.2 -c 5
PING 192.168.1.2 (192.168.1.2) from 192.168.1.1 vm1: 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=2.61 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.195 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.199 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.169 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.230 ms

--- 192.168.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4056ms
rtt min/avg/max/mdev = 0.169/0.682/2.618/0.968 ms
ks-sdn@sdn_server:~/overlay-lab$ sudo docker exec overlaylab_hv2_1 ping -I vm2 192.168.1.1 -c 5
PING 192.168.1.1 (192.168.1.1) from 192.168.1.2 vm2: 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.12 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.190 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.187 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=0.190 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=0.222 ms

--- 192.168.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4044ms
rtt min/avg/max/mdev = 0.187/0.383/1.129/0.373 ms
ks-sdn@sdn_server:~/overlay-lab$ sudo docker exec overlaylab_hv2_1 ping -I vm2 192.168.1.1 -c 5
PING 192.168.1.1 (192.168.1.1) from 192.168.1.2 vm2: 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.907 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.186 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.186 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=0.186 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=0.247 ms

--- 192.168.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4041ms
rtt min/avg/max/mdev = 0.186/0.342/0.907/0.283 ms
ks-sdn@sdn_server:~/overlay-lab$ _
```

Transfer Container Captured files to current directory:

```
File  Machine  View  Input  Devices  Help
ks-sdn@sdn_server:~/overlay-lab$ sudo docker cp overlaylab_hv1_1:/vm1_overlay.pcap .
[sudo] password for ks-sdn:
ks-sdn@sdn_server:~/overlay-lab$ sudo docker cp overlaylab_hv2_1:/vm2_overlay.pcap .
ks-sdn@sdn_server:~/overlay-lab$ sudo docker cp overlaylab_hv1_1:/vm1_underlay.pcap .
ks-sdn@sdn_server:~/overlay-lab$ sudo docker cp overlaylab_hv2_1:/vm2_underlay.pcap .
ks-sdn@sdn_server:~/overlay-lab$ ls -l *.pcap
-rw-r--r-- 1 root root 2536 Jun 30 22:16 vm1_overlay.pcap
-rw-r--r-- 1 root root 4386 Jul  3 14:21 vm1_underlay.pcap
-rw-r--r-- 1 root root 2652 Jul  3 14:23 vm2_overlay.pcap
-rw-r--r-- 1 root root 1996 Jul  3 14:25 vm2_underlay.pcap
ks-sdn@sdn_server:~/overlay-lab$
ks-sdn@sdn_server:~/overlay-lab$ _
```

View files using tshark:

```
File  Machine  View  Input  Devices  Help
ks-sdn@sdn_server:~/overlay-lab$ tshark -V -r vm1_overlay.pcap -c 1 --color_
```

```
File  Machine  View  Input  Devices  Help
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        0000 00.. = Differentiated Services Codepoint: Default (0)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 84
    Identification: 0x1c71 (7281)
    Flags: 0x4000, Don't fragment
        0... .... .... .... = Reserved bit: Not set
        .1.. .... .... .... = Don't fragment: Set
        ..0. .... .... .... = More fragments: Not set
        ...0 0000 0000 0000 = Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0x9ae4 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.1.1
    Destination: 192.168.1.2
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x1a2f [correct]
    [Checksum Status: Good]
    Identifier (BE): 82 (0x0052)
    Identifier (LE): 20992 (0x5200)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    Timestamp from icmp data: Jun 30, 2022 21:53:16.000000000 UTC
    [Timestamp from icmp data (relative): 0.339323000 seconds]
    Data (48 bytes)

0000  0f 2d 05 00 00 00 00 00 10 11 12 13 14 15 16 17   .-..............
0010  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27   ........ !"#$%&'
0020  28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37   ()*+,-./01234567
        Data: 0f2d050000000000101112131415161718191a1b1c1d1e1f...
        [Length: 48]

ks-sdn@sdn_server:~/overlay-lab$
```

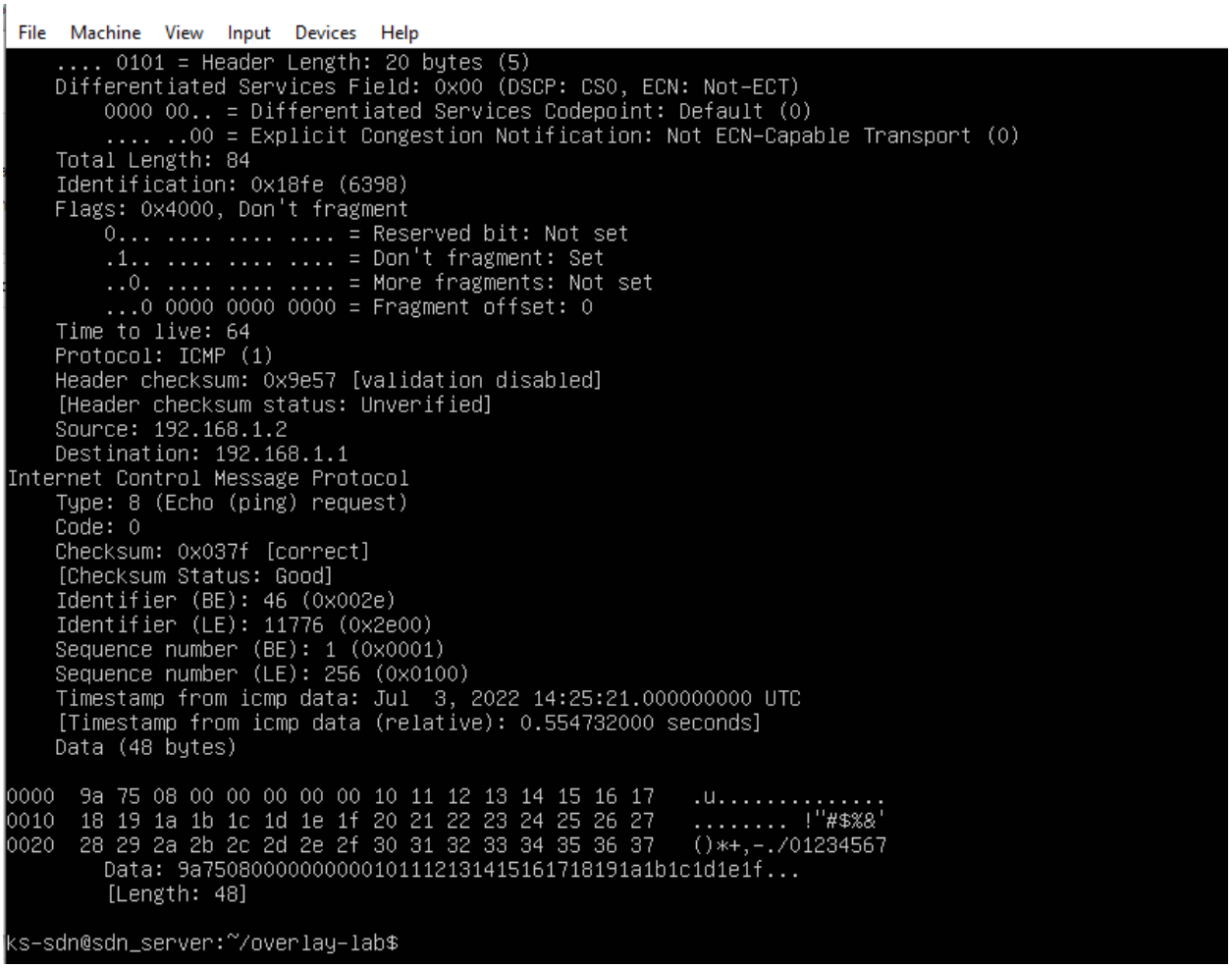
```
File  Machine  View  Input  Devices  Help
ks-sdn@sdn_server:~/overlay-lab$ ls -l *.pcap
-rw-r--r-- 1 root root 2536 Jun 30 22:16 vm1_overlay.pcap
-rw-r--r-- 1 root root 4386 Jul  3 14:21 vm1_underlay.pcap
-rw-r--r-- 1 root root 2652 Jul  3 14:23 vm2_overlay.pcap
-rw-r--r-- 1 root root 1996 Jul  3 14:25 vm2_underlay.pcap
ks-sdn@sdn_server:~/overlay-lab$ tshark -V -r vm1_underlay.pcap -c 1 --color
```

```
File  Machine  View  Input  Devices  Help
    [Coloring Rule String: eth[0] & 1]
Ethernet II, Src: 42:91:e9:ca:5d:f6 (42:91:e9:ca:5d:f6), Dst: IPv6mcast_02 (33:33:00:00:00:02)
    Destination: IPv6mcast_02 (33:33:00:00:00:02)
        Address: IPv6mcast_02 (33:33:00:00:00:02)
        .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factor
y default)
        .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
    Source: 42:91:e9:ca:5d:f6 (42:91:e9:ca:5d:f6)
        Address: 42:91:e9:ca:5d:f6 (42:91:e9:ca:5d:f6)
        .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factor
y default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv6 (0x86dd)
Internet Protocol Version 6, Src: fe80::4091:e9ff:feca:5df6, Dst: ff02::2
    0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
        .... 0000 00.. .... .... .... .... .... = Differentiated Services Codepoint: Default (0)
        .... .... ..00 .... .... .... .... .... = Explicit Congestion Notification: Not ECN-Capable
Transport (0)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 16
    Next Header: ICMPv6 (58)
    Hop Limit: 255
    Source: fe80::4091:e9ff:feca:5df6
    Destination: ff02::2
Internet Control Message Protocol v6
    Type: Router Solicitation (133)
    Code: 0
    Checksum: 0x6a89 [correct]
    [Checksum Status: Good]
    Reserved: 00000000
    ICMPv6 Option (Source link-layer address : 42:91:e9:ca:5d:f6)
        Type: Source link-layer address (1)
        Length: 1 (8 bytes)
        Link-layer address: 42:91:e9:ca:5d:f6 (42:91:e9:ca:5d:f6)

ks-sdn@sdn_server:~/overlay-lab$ _
```

```
File  Machine  View  Input  Devices  Help
ks-sdn@sdn_server:~/overlay-lab$ ls -l *.pcap
-rw-r--r-- 1 root root 2536 Jun 30 22:16 vm1_overlay.pcap
-rw-r--r-- 1 root root 4386 Jul  3 14:21 vm1_underlay.pcap
-rw-r--r-- 1 root root 2652 Jul  3 14:23 vm2_overlay.pcap
-rw-r--r-- 1 root root 1996 Jul  3 14:25 vm2_underlay.pcap
ks-sdn@sdn_server:~/overlay-lab$ tshark -V -r vm2_overlay.pcap -c 1 --color _
```

```
File  Machine  View  Input  Devices  Help
    Arrival Time: Jul  3, 2022 14:17:02.499671000 UTC
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1656857822.499671000 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 42 bytes (336 bits)
    Capture Length: 42 bytes (336 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:arp]
    [Coloring Rule Name: ARP]
    [Coloring Rule String: arp]
Ethernet II, Src: 00:00:00_11:11:11 (00:00:00:11:11:11), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
    Destination: Broadcast (ff:ff:ff:ff:ff:ff)
        Address: Broadcast (ff:ff:ff:ff:ff:ff)
        .... ..1. .... .... .... .... = LG bit: Locally administered address (this is NOT the factor
y default)
        .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
    Source: 00:00:00_11:11:11 (00:00:00:11:11:11)
        Address: 00:00:00_11:11:11 (00:00:00:11:11:11)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: ARP (0x0806)
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 00:00:00_11:11:11 (00:00:00:11:11:11)
    Sender IP address: 192.168.1.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.1.2

ks-sdn@sdn_server:~/overlay-lab$
```

```
File  Machine  View  Input  Devices  Help
ks-sdn@sdn_server:~/overlay-lab$ ls -l *.pcap
-rw-r--r-- 1 root root 2536 Jun 30 22:16 vm1_overlay.pcap
-rw-r--r-- 1 root root 4386 Jul  3 14:21 vm1_underlay.pcap
-rw-r--r-- 1 root root 2652 Jul  3 14:23 vm2_overlay.pcap
-rw-r--r-- 1 root root 1996 Jul  3 14:25 vm2_underlay.pcap
ks-sdn@sdn_server:~/overlay-lab$ tshark -V -r vm2_underlay.pcap -c 1 --color
```


File Machine View Input Devices Help
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
0000 00.. = Differentiated Services Codepoint: Default (0)
.... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
Total Length: 84
Identification: 0x18fe (6398)
Flags: 0x4000, Don't fragment
0... .... = Reserved bit: Not set
.1.. .... = Don't fragment: Set
..0. .... = More fragments: Not set
...0 0000 0000 0000 = Fragment offset: 0
Time to live: 64
Protocol: ICMP (1)
Header checksum: 0x9e57 [validation disabled]
[Header checksum status: Unverified]
Source: 192.168.1.2
Destination: 192.168.1.1
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x037f [correct]
[Checksum Status: Good]
Identifier (BE): 46 (0x002e)
Identifier (LE): 11776 (0x2e00)
Sequence number (BE): 1 (0x0001)
Sequence number (LE): 256 (0x0100)
Timestamp from icmp data: Jul 3, 2022 14:25:21.000000000 UTC
[Timestamp from icmp data (relative): 0.554732000 seconds]
Data (48 bytes)
0000 9a 75 08 00 00 00 00 00 10 11 12 13 14 15 16 17 .u..............
0010 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 .... !"#$%&'
0020 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37 ()*+,-./01234567
Data: 9a750800000000001011121314151617181 9a1b1c1d1e1f...
[Length: 48]
ks-sdn@sdn_server:~/overlay-lab$

QED