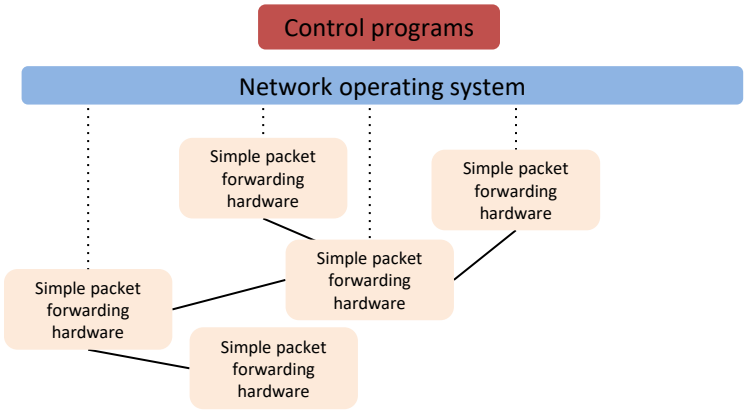
Software Defined Networking (SDN)

- Separation of control functions from forwarding functions
- Intelligence is concentrated in the control plane



1

Hands-On Experimentation With SDN

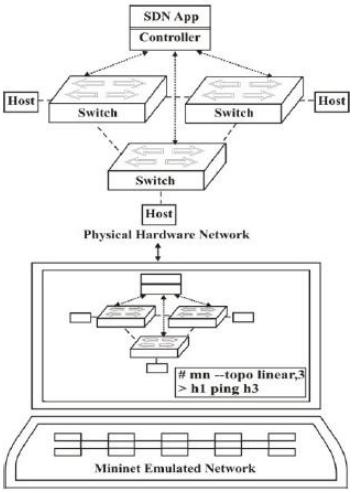
- Need a capability (simulator or emulator) to create real-world scenarios
- Mininet is an emulator that provides such capabilities
- Mininet uses the container-based abstractions
  - Namespaces, cgroups
  - Can create arbitrary topologies
  - Can emulate hosts and hosts can execute user-supplied logic
  - Supports open virtual switch
  - Implements OpenFlow controller
- On your laptop VM, do:
  - Clone the Mininet github repo and invoke the install.sh command with -a flag as discussed in the next few slides.
  - There is also a special Mininet VM available

2

Mininet: Basic Usage, CLI, API

Mininet: network emulator which creates realistic virtual network

- Runs real kernel, switch, and application code on a single machine
- Provides both Command Line Interface (CLI) and Application Programming Interface (API)
- Reasonably accurate, easy to download, and fast/interactive usage
- Abstraction
  - Host: emulated as an OS level process
  - Switch: emulated by using software-based switch
- **But, slower than hardware experiments: may not fit possible inaccuracy from multiplexing**



3

Mininet: Basic Usage, CLI, API (cont.)

Install Mininet (multiple different approaches; I used Git and install)

- **Mininet VM installation**
  - Download the Mininet VM image
- **Native installation from source (Preferred Approach)**
  - ***git clone git://github.com/mininet/mininet***
  - ***$ mininet/util/install.sh –a***
  - ***You may need to do "sudo apt-get install net-tools" for missing ifconfig***
- **Installation from packages**
  - ***$ sudo apt-get install mininet***
- **Upgrading an existing Mininet installation**
  - ***$ cd mininet && git fetch && git checkout master number or a specific version like 2.2.1***
  - ***$ git pull && sudo make install***

Mininet uses Linux network namespaces and some additional capabilities.

4

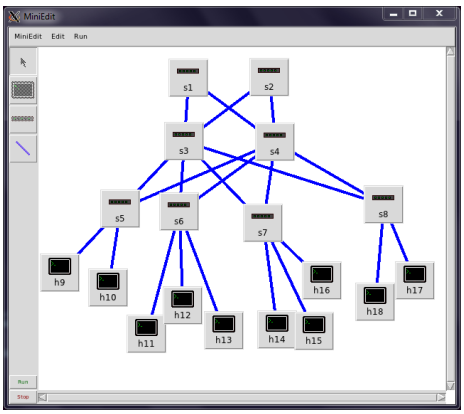
Neat Features of Mininet

- Ability to create arbitrarily large and complex networking topologies
- Ability to execute application logic in emulated hosts
- Ability to control network traffic characteristics
- Ability to integrate with Wireshark to observe traffic

5

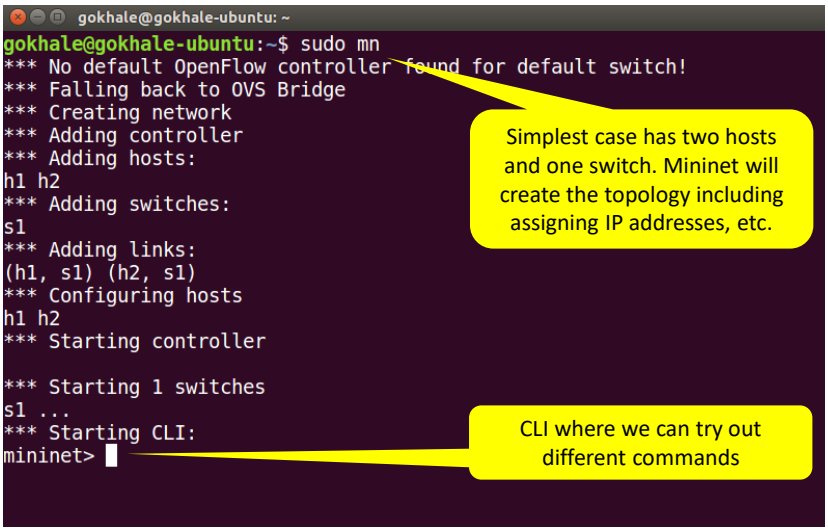
Mininet: Basic Usage, CLI, API

- **Mininet GUI**
  - Mininet provides a tool ("MiniEdit") to help creating real networks
    - Run the file mininet/examples/miniedit.py
- **Mininet CLI and features**
  - $sudo mn --topo tree,depth=3,fanout=3 --link=tc,bw=10
  - mininet> xterm h1 h2
- **Mininet's Python API**
  - net = Mininet() # net is a Mininet() object
  - h1 = net.addHost('h1') # h1 is a Host() object
  - h2 = net.addHost('h2') # h2 is a Host()
  - s1 = net.addSwitch('s1') # s1 is a Switch() object



6

Simple Command to Try



7

Simple Command to Try (cont.)

8

Mininet Help on the CLI

```
mininet>
mininet> help

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

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
```

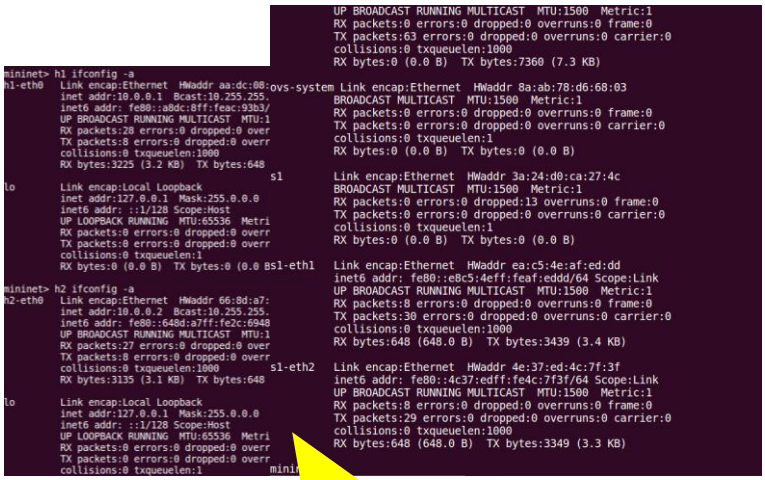
9

Command Line Parameters to “sudo mn”

```
gokhale@gokhale-ubuntu:~$ sudo mn -h
Usage: mn [options]
(type mn -h for details)

The mn utility creates Mininet network from the command line. It can create
parametrized topologies, invoke the Mininet CLI, and run tests.

Options:
  -h, --help            show this help message and exit
  --switch=SWITCH       default|ivs|lxbr|ovs|ovsbr|ovsk|user[,param=value...]
                        ovs=OVSSwitch default=OVSSwitch ovsk=OVSSwitch
                        lxbr=LinuxBridge user=UserSwitch ivs=IVSSwitch
                        ovsbr=OVSBridge
  --host=HOST           cfs|proc|rt[,param=value...]
                        rt=CPULimitedHost{'sched': 'rt'} proc=Host
                        cfs=CPULimitedHost{'sched': 'cfs'}
  --controller=CONTROLLER
                        default|none|nox|ovsc|ref|remote|ryu[,param=value...]
                        ovsc=OVSController none=NullController
                        remote=RemoteController default=DefaultController
                        nox=NOX ryu=Ryu ref=Controller
  --link=LINK           default|ovs|tc[,param=value...] default=Link
                        ovs=OVSLink tc=TCLink
  --topo=TOPO           linear|minimal|reversed|single|torus|tree[,param=value
                        ...] linear=LinearTopo torus=TorusTopo tree=TreeTopo
                        single=SingleSwitchTopo
                        reversed=SingleSwitchReversedTopo minimal=MinimalTopo
  -c, --clean           clean and exit
  --custom=CUSTOM       read custom classes or params from .py file(s)
  --test=TEST           cli|build|pingall|pingpair|iperf|all|iperfudp|none
  -x, --xterms          spawn xterms for each node
  -i IPBASE, --ipbase=IPBASE
                        base IP address for hosts
  --mac                 automatically set host MACs
  --arp                 set all-pairs ARP entries
  -v VERBOSITY, --verbosity=VERBOSITY
```

10

Executing Commands on Hosts, Switches, and Controllers



11

Executing Commands on Hosts, Switches, and Controllers (cont.)

- To execute a specific command inside a given entity (e.g., a host or switch), specify the entity name firs,t followed by the command you wish to execute.
- Example here shows running the ifconfig command, ps command, pinging other host in the network, etc.

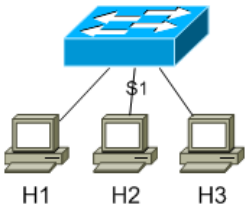
```
mininet> h1 ifconfig -a
h1-eth0   Link encap:Ethernet  HWaddr da:c0:8b:04:2b:7e
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::d8c0:8bff:fe04:2b7e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:30 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3439 (3.4 KB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          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:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

mininet> h1 ps
  PID TTY          TIME CMD
 8426 pts/17   00:00:00 bash
 8781 pts/17   00:00:00 ps
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.259 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.135 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.124 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.104 ms
```

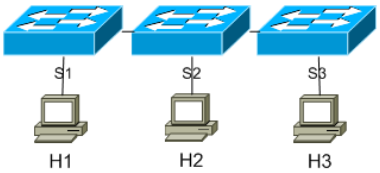
12

Mininet: Slightly Advanced Usage, CLI, API

Example 1: create a network and run a simple performance test

- **Minimal topology:** *$sudo mn --topo single,3*
  - ***Minimal is very simple topology that contains one OpenFlow switch and three hosts.***
- **Reversed topology:** *$sudo mn --topo reversed,4*
  - ***It is similar to single topology but connection order is reversed.***
- **Linear topology:** *$sudo mn --topo linear,3*
  - ***Linear topology contains k switches and k hosts.***
  - ***It also creates a link between each switch and each host and among the switches.***



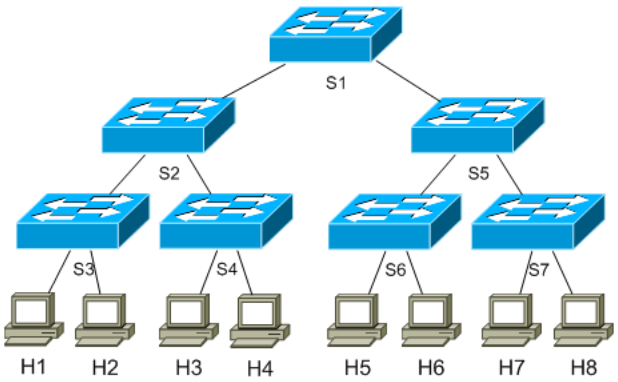


13

Mininet: Slightly Advanced Usage, CLI, API (cont.)

Example 2: create a network and run a simple performance test

- **Tree topology:** *$sudo mn --topo tree,3*
  - ***Tree topology contains k levels and two hosts are attached to per switch.***



14

Namespaces in Mininet

- Note that Mininet uses Linux namespaces for isolating the hosts, switches, and controllers from each other.
- Note that it uses only the network namespace and nothing else.
  - The process hierarchy and file system is the same across all hosts and switches and controllers.
  - One could use additional isolation dimensions if needed using the specific option provided by Mininet.
    - Option to use: --innamespace

15

Cleaning Up

- Type exit on the mininet CLI to tear down the network and cleanup
- On an exception or if mininet crashes:
  - sudo mn -c

16

Mininet: More Complex Example (Example 3)

- sudo mn --link tc,bw=10,delay=10ms
- Two hosts and one switch scenario
- Each link has 10 msec delay and a 10 Mbps capacity
- Host h1 ping to h2 will traverse a path from h1 to s1 to h2 and back to s1 to h1 for a total of 40 ms round-trip
- Notice how the first ping took more time because of route setup time

```
gokhale@gokhale-ubuntu:~$ sudo mn --link tc,bw=10,delay=10ms
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(10.00Mbit 10ms delay) *** Error: RTNETLINK answers: No such file or directory
(10.00Mbit 10ms delay) *** Error: RTNETLINK answers: No such file or directory
(h1, s1) (10.00Mbit 10ms delay) *** Error: RTNETLINK answers: No such file or di
rectory
(10.00Mbit 10ms delay) *** Error: RTNETLINK answers: No such file or directory
(h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...(10.00Mbit 10ms delay) (10.00Mbit 10ms delay)
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=99.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=41.4 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=42.4 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=41.8 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=41.9 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 41.429/53.380/99.234/22.929 ms
mininet>
```

17

Mininet: Programming API: Part I

Example 4: create a customized network topology

- Using Mininet CLI: $sudo mn --topo tree,depth=3,fanout=3 --link=tc,bw=10
- Needs just writing a few lines of Python

```
#!/usr/bin/python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import CPULimitedHost
from mininet.link import TCLink
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def __init__(self, n=2, **opts):
        Topo.__init__(self, **opts)
        switch = self.addSwitch('s1')
        for h in range(n):
            #Each host gets 50%/n of system CPU
            host = self.addHost('h%s' % (h + 1), cpu=.5/n)
            #10 Mbps, 5ms delay, 0% Loss, 1000 packet queue
            self.addLink(host, switch, bw=10, delay='5ms',
loss=0, max_queue_size=1000, use_htb=True)
```

```
def perfTest():
    "Create network and run simple performance test"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo=topo,host=CPULimitedHost, link=TCLink)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    print "Testing bandwidth between h1 and h4"
    h1, h4 = net.get('h1', 'h4')
    net.iperf((h1, h4))
    net.stop()

if __name__=='__main__':
    setLogLevel('info')
    perfTest()
```

18

Mininet: Programming API: Part II

Example 5: create a simple network and use a POX controller

Note that POX, Ryu etc are old SDN controllers; Need to switch to P4, etc

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller
from mininet.topo import SingleSwitchTopo
from mininet.log import setLogLevel

import os

class POXBridge( Controller ):
    "Custom Controller class to invoke POX forwarding.l2_learning"
    def start( self ):
        "Start POX learning switch"
        self.pox = '%s/pox/pox.py ' % os.environ[ 'HOME' ]
        self.cmd( self.pox, 'forwarding.l2_learning &' )
    def stop( self ):
        "Stop POX"
        self.cmd('kill %' + self.pox )
```

```
controllers = { 'poxbridge': POXBridge }

if __name__ == '__main__':
    setLogLevel( 'info' )
    net = Mininet( topo=SingleSwitchTopo( 2 ), controller=POXBridge )
    net.start()
    net.pingAll()
    net.stop()
```

```
$ sudo mn --custom poxbridge.py --controller poxbridge --topo tree,2,2 --test pingall -v output
```

19

Mininet: Programming API: Part III

Example 6: dynamically change the network parameters—change link delay

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Node
from mininet.link import *
from mininet.log import setLogLevel, info
from threading import Timer
def myNet():
    "Create network from scratch using Open vSwitch."
    info( "*** Creating nodes\n" )
    switch0 = Node( 's0', inNamespace=False )
    switch1 = Node( 's1', inNamespace=False )
    h0 = Node( 'h0' )
    h1 = Node( 'h1' )
info( "*** Creating links\n" )
    linkopts0=dict(bw=100, delay= '1ms', loss=0)
    link1=TCLink( h0, switch0, **linkopts0)
    TCLink( switch0, switch1, **linkopts0)
    link2=TCLink( h1, switch4, **linkopts0)
info( "*** Configuring hosts\ns" )
    h0.setIP( '192.168.123.1/24' )
    h1.setIP( '192.168.123.2/24' )
    info( str( h0 ) + '\n' )
    info( str( h1 ) + '\n' )

info( "*** Starting network using Open vSwitch\n" )
    switch0.cmd( 'ovs-vsctl del-br dp0' )
    switch0.cmd( 'ovs-vsctl add-br dp0' )
    switch1.cmd( 'ovs-vsctl del-br dp1' )
    switch1.cmd( 'ovs-vsctl add-br dp1' )

    for intf in switch0.intfs.values():
        print intf
        print switch0.cmd( 'ovs-vsctl add-port dp0 %s' % intf )

    for intf in switch1.intfs.values():
        print intf
        print switch1.cmd( 'ovs-vsctl add-port dp1 %s' % intf )
```

20

Mininet: Programming API: Part IV

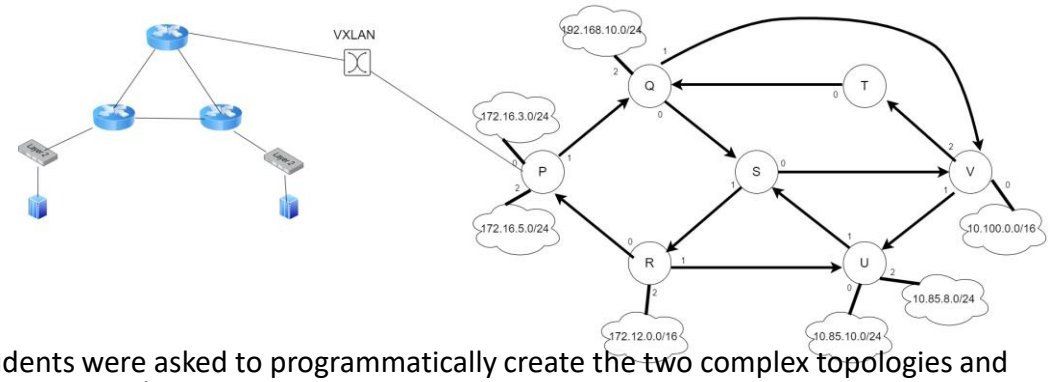
Example 7: dynamically change the network parameters—change link delay

```
print switch1.cmd(r'ovs-ofctl add-flow dp1 idle_timeout=0,priority=1,in_port=1,actions=flood' )
print switch1.cmd(r'ovs-ofctl add-flow dp1 idle_timeout=0,priority=1,in_port=1,actions=output:2' )
print switch0.cmd(r'ovs-ofctl add-flow dp0 idle_timeout=0,priority=10,ip,nw_dst=192.168.123.2,nw_tos=0x10,actions=output:2')
print switch0.cmd(r'ovs-ofctl add-flow dp0 idle_timeout=0,priority=10,ip,nw_dst=192.168.123.2,nw_tos=0x20,actions=output:3')
print switch0.cmd(r'ovs-ofctl add-flow dp0 idle_timeout=0,priority=10,ip,nw_dst=192.168.123.2,nw_tos=0x30,actions=output:4')

def cDelay1():
    h1.cmdPrint('ethtool -K h1-eth0 gro off')
    h1.cmdPrint('tc qdisc del dev h1-eth0 root')
    h1.cmdPrint('tc qdisc add dev h1-eth0 root handle 10: netem delay 100ms')

  def hello(a, b='4'):
      print a
      print b
  t=Timer(5.0, cDelay1)
  t.start()
```

```
info( "*** Running test\n" )
  h0.cmdPrint( 'ping -Q 0x30 -c 10' + h1.IP() )
  info( "*** Stopping network\n" )
  switch0.cmd( 'ovs-vsctl del-br dp0' )
  switch0.deleteIntfs()
  switch1.cmd( 'ovs-vsctl del-br dp1' )
  switch1.deleteIntfs()
if __name__ == '__main__':
  setLogLevel( 'info' )
  info( '*** Scratch network demo (kernel datapath)\n' )
  Mininet.init()
  myNet()
```

21

Complex Example Given to Networking Class



- Students were asked to programmatically create the two complex topologies and interconnect them
- The left-side topology was to be deployed on one VM and right-side topology on another VM
- The Mininets were then required to communicate with each other
- This demonstrates a way to build very large and complex topologies, and test complex applications end-to-end without incurring the limitation of a single laptop's capacity
  - E.g., CAGE2 topology

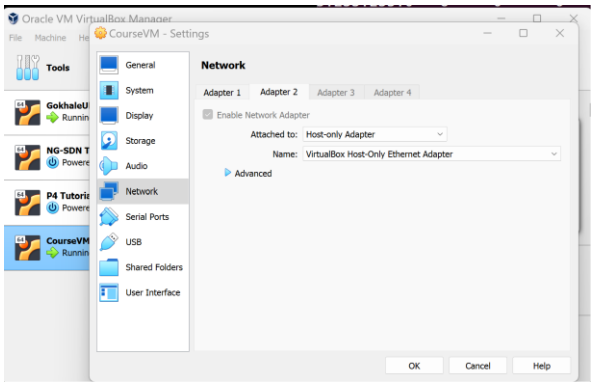
22

Approach to Making Mininets Communicate (1/7)

- Ideally, a simple solution comprising an additional routing table entry on the two VMs which host individual Mininets should work
  - This was shown to work on two VMs that used a common host-only network on the laptop
- Here is what was done to make that work
  - Start each mininet with --nat option and a -i <unique subnet> so that the hosts will get IP address from that subnet
  - Then, each VM's routing table was updated to relay traffic to the other Mininet via the underlying VM and through a specific interface
  - See screenshots next slide

23

Approach to Making Mininets Communicate (2/7)




```
gokhale@coursevm: $ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 enp0s3
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
192.168.56.0    0.0.0.0         255.255.255.0   U     101    0        0 enp0s8
```

```
gokhale@CourseVM: $ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
10.121.68.0     0.0.0.0         255.255.255.0   U     0      0        0 mpqemubr
0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 enp0s3
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.18.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker_g
wbridge
192.168.56.0    0.0.0.0         255.255.255.0   U     101    0        0 enp0s8
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
```

- Each VM had a second adapter for host-only network, which gets IP addresses from the 192.168.56.0/24 subnet
- This is very similar to our Chameleon VMs, which get their IP addresses in the 192.168.5.0/24 subnet

- These were the routing tables on my 2 VMs on the laptop
- Notice the rule for the 192.168.56.0 prefix, which sends packet to the network it is connected via the enp0s8 interface
  - The interface that is specified is important

24

Approach to Making Mininets Communicate (3/7)

```
gokhale@coursevm:~$ sudo mn --nat -i 10.10.1.0/24
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Warning: loopback address in /etc/resolv.conf may break host DNS over NAT
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

- The first VM starts a mininet with 10.10.1.0/24 subnet
- Host h1 will get 10.10.1.1, Host h2 will get 10.10.1.2 IP addresses
- We avoid 10.0.2.0/24 subnet as it is used by our NAT

- The second VM starts another Mininet with 10.10.2.0/24 subnet
- Host h1 will get 10.10.2.1, Host h2 will get 10.10.2.2 IP addresses
- We avoid 10.0.2.0/24 subnet as it is used by our NAT

```
gokhale@CourseVM:~$ sudo mn --nat -i 10.10.2.0/24
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Warning: loopback address in /etc/resolv.conf may break host DNS over NAT
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

25

Approach to Making Mininets Communicate (4/7)

```
gokhale@coursevm:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
10.10.1.0       0.0.0.0         255.255.255.0   U     0      0        0 nat0-eth
0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 enp0s3
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
192.168.56.0    0.0.0.0         255.255.255.0   U     101    0        0 enp0s8
```

- This is the routing table on VM1 after Mininet was started
- Notice the entry for 10.10.1.0, which is the address space for the hosts on that mininet

- The second VM starts another Mininet with 10.10.2.0/24 subnet
- Host h1 will get 10.10.2.1, Host h2 will get 10.10.2.2 IP addresses
- We avoid 10.0.2.0/24 subnet as it is used by our NAT

```
gokhale@CourseVM:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
10.10.2.0       0.0.0.0         255.255.255.0   U     0      0        0 nat0-eth
0
10.121.68.0     0.0.0.0         255.255.255.0   U     0      0        0 mpqemubr
0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 enp0s3
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.18.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker_g
wbridge
192.168.56.0    0.0.0.0         255.255.255.0   U     101    0        0 enp0s8
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
```

26

Approach to Making Mininets Communicate (5/7)

```
mininet> h2 ping 10.10.2.1
PING 10.10.2.1 (10.10.2.1) 56(84) bytes of data.
```

```
mininet> h1 ping 10.10.1.2
PING 10.10.1.2 (10.10.1.2) 56(84) bytes of data.
```

- Pinging from a host of one mininet to another host of another mininet does not work but we can ping on hosts of the same mininet
- There is no route between the mininets
- The –nat option only allows us to go out from the boundaries of

```
enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.106  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::f347:4108:fc28:10c6  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:41:df:89  txqueuelen 1000  (Ethernet)
        RX packets 1156  bytes 121153 (121.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 111  bytes 21002 (21.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.103  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::b419:36ab:824a:ab00  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:e5:28:8a  txqueuelen 1000  (Ethernet)
        RX packets 1559  bytes 163303 (163.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 129  bytes 22526 (22.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- We need to add routing rules to both VMsto make this happen
- For that we need to understand our IP address on the host-only network because that is the one on which the VMs can converse
- VM1: 192.168.56.106, VM2: 192.168.156.103

27

Approach to Making Mininets Communicate (6/7)

```
gokhale@coursevm:~$ sudo ip route add 10.10.2.0/24 via 192.168.56.103 dev enp0s8
gokhale@coursevm:~$
gokhale@coursevm:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
10.10.1.0       0.0.0.0         255.255.255.0   U     0      0        0 nat0-eth
0
10.10.2.0       192.168.56.103  255.255.255.0   UG    0      0        0 enp0s8
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 enp0s3
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
192.168.56.0    0.0.0.0         255.255.255.0   U     101    0        0 enp0s8
```

On VM1

```
gokhale@CourseVM:~$ sudo ip route add 10.10.1.0/24 via 192.168.56.106 dev enp0s8
gokhale@CourseVM:~$
gokhale@CourseVM:~$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.2.2        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
10.10.1.0       192.168.56.106  255.255.255.0   UG    0      0        0 enp0s8
10.10.2.0       0.0.0.0         255.255.255.0   U     0      0        0 nat0-eth
0
10.121.68.0     0.0.0.0         255.255.255.0   U     0      0        0 mpqemubr
0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 enp0s3
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.18.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker_g
wbridge
192.168.56.0    0.0.0.0         255.255.255.0   U     101    0        0 enp0s8
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
```

On VM2

28

Approach to Making Mininets Communicate (7/7)

```
mininet> h2 ping 10.10.2.1
PING 10.10.2.1 (10.10.2.1) 56(84) bytes of data.
64 bytes from 10.10.2.1: icmp_seq=1 ttl=62 time=2.61 ms
64 bytes from 10.10.2.1: icmp_seq=2 ttl=62 time=3.03 ms
64 bytes from 10.10.2.1: icmp_seq=3 ttl=62 time=217 ms
64 bytes from 10.10.2.1: icmp_seq=4 ttl=62 time=3.10 ms
64 bytes from 10.10.2.1: icmp_seq=5 ttl=62 time=2.47 ms
64 bytes from 10.10.2.1: icmp_seq=6 ttl=62 time=1.76 ms
^C
--- 10.10.2.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5023ms
rtt min/avg/max/mdev = 1.756/38.369/217.256/80.001 ms
```

On VM1

```
mininet> h1 ping 10.10.1.2
PING 10.10.1.2 (10.10.1.2) 56(84) bytes of data.
64 bytes from 10.10.1.2: icmp_seq=1 ttl=62 time=15.7 ms
64 bytes from 10.10.1.2: icmp_seq=2 ttl=62 time=5.61 ms
64 bytes from 10.10.1.2: icmp_seq=3 ttl=62 time=2.72 ms
64 bytes from 10.10.1.2: icmp_seq=4 ttl=62 time=1.58 ms
64 bytes from 10.10.1.2: icmp_seq=5 ttl=62 time=2.29 ms
^C
--- 10.10.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4015ms
rtt min/avg/max/mdev = 1.582/5.577/15.690/5.238 ms
```

On VM2

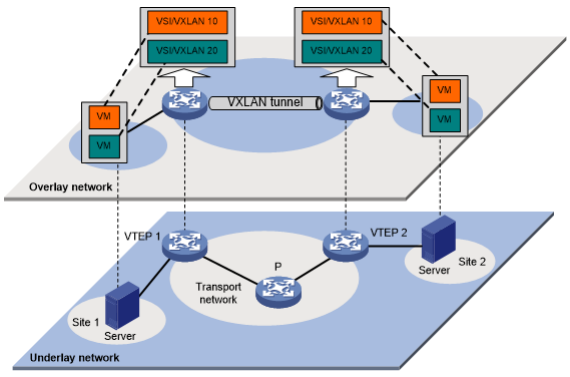
29

Some Problems Encountered on Chameleon

- Although the setup is very much the same on Chameleon, there are problems getting the approach to work
- First, Chameleon has “ufw” active, which gives rise to numerous internal iptables rules
- Despite disabling ufw and flushing out all iptables rules in all tables, the solution did not work
  - Somewhere after the POSTROUTING, the reply was getting either lost or dropped
- VxLAN tunneling shows promise
  - But we could not get Open Virtual Switch-based VxLANs to work
  - So, we ended up using native Linux-based Bridging and VxLANs using the “ip” command

30

VxLAN Approach



Source: techhub.hpe.com

- VxLAN (Virtual Extensible LANs) provides an illusion of a single LAN segment to distributed resources on possibly different physical networks
- The concept is realized by tunneling Layer 2 traffic over Layer 3, i.e., ethernet traffic is carried over network traffic
- In this case, UDP is used as the protocol for tunneling
- Our responsibility is to create the tunnel endpoints and let our traffic flow through the tunnel
- This way, we can enable two or more Mininets on different VMs to communicate with each other

31

Creating VxLAN Tunnel (1/4)

- The tunnel has to be created on both VMs with each tunnel provided with an IP address (that we make up ourselves)
- The tunnel should have a unique, currently unused, ID
- In my setup, my VMs had the following IP addresses:
  - VM1: 192.168.5.115
  - VM2: 192.168.5.84
  - Your VMs will have different IP addresses; so be careful when you copy-paste these commands

32

Creating VxLAN Tunnel (2/4)

- Step 1: Create the VxLAN (this was executed on my VM2)

sudo ip link add vxlan0 type vxlan id 100 local 192.168.5.84 remote 192.168.5.115 dev ens3 dstport 4789

- where,
  - "ip link add" adds a new interface/link; must be done with "sudo"
  - vxlan0 is the name of the VxLAN and is of type "vxlan"
  - 100 is the id that we are giving this VxLAN and must be the same on the other side
  - local is the IP of the VM on which this command is executed; remote is the IP of the other VM
  - Since these IP addresses of the VM are reachable over the "ens3" interface (do ifconfig and verify), we use the "device" as "ens3)
  - VxLAN logic needs to use a UDP port, which usually is 4789
- If there are problems, you may need to add a "ufw rule"
  - sudo ufw limit 4789/udp
  - Alternately, you may just disable ufw (but take this step if things are not working)

33

Creating VxLAN Tunnel (3/4)

- Step 2: Create the VxLAN (this was executed on my VM2; issue similar command on other VM)

sudo ip addr add 192.168.100.2/24 dev vxlan0

- where,
  - Using the "ip addr add" command, we attach an arbitrary, unused, private IP address to this side of the VxLAN tunnel
    - For the other VM, use the same subnet but a different IP address
  - Effectively, we should see it show up in our ifconfig output with that ip address

34

Creating VxLAN Tunnel (4/4)

- Step 3: Activating the VxLAN (Execute this on both VMs)

sudo ip link set vxlan0 up

- where,
  - Using the "ip addr set" command, we set a property on our VxLAN
  - Here, we activate it

- Step 4: Verify creation using ifconfig or ip link show

sudo ip link show

Or simply

ifconfig

- where,
  - We should see a vxlan0 interface just like we have ens3 and others
  - There should be an IP address associated with it

35

Getting the Mininets to Communicate

- Start the Mininets on the two VMs using two separate subnets as discussed/shown before
- Use the route table manipulation steps shown earlier but this time use the IP addresses from the VxLAN and that interface
- Then test out pinging, iperf and finally our application code by putting a server on one host of one of the mininets, and the client on another host of another mininet
- Similar approach can be used to deploy CASTLE workflows end-to-end

36