Teaching Computer Networking with Mininet

Te-Yuan Huang, Vimalkumar Jeyakumar
Bob Lantz, Brian O'Connor
Nick Feamster
Keith Winstein, Anirudh Sivaraman

Wi-Fi: HHonors, SGC2014

Tutorial Goals

Learn how Mininet (and network emulation in general) works, and how it can be used in computer networking courses

Gain hands-on experience using Mininet for a network lab exercise

Find out what we've learned from using Mininet in on-campus courses and MOOCs

Tutorial Agenda

1. Introduction to Mininet

presentation, demos, short break

2. Hands-on Lab

presentation, lab, coffee break

3. Teaching with Mininet

presentations, discussion, done!

Teaching Computer Networking with Mininet

Session 1: **Introduction to Mininet**

Bob Lantz
Open Networking Laboratory

Introduction to Mininet

Platforms for Network/Systems Teaching

Network Emulator Architecture

Mininet: Basic Usage, CLI, API

Example Demos: Network Security

Conclusion and Questions

Experiential Learning for Networking

"**Learning by doing**" is memorable and leads to mastery.

In **computer systems courses**, this means building, modifying, using, and experimenting with working systems.

Networking (and distributed systems) courses require complicated **testbeds** including multiple **servers** and **switches.**

Platforms for Network/Systems Teaching (and Research)

Platform	Advantages	Disadvantages
Hardware Testbed	fast accurate: "ground truth"	expensive shared resource? hard to reconfigure hard to change hard to download
Simulator	inexpensive, flexible detailed (or abstract!) easy to download virtual time (can be "faster" than reality)	may require app changes might not run OS code detail != accuracy may not be "believable" may be slow/non-interactive
Emulator	**inexpensive, flexible** **real code** **reasonably accurate** **easy to download** **fast/interactive usage**	slower than hardware experiments may not fit possible inaccuracy from multiplexing

Introduction to Mininet

Platforms for Network/Systems Teaching

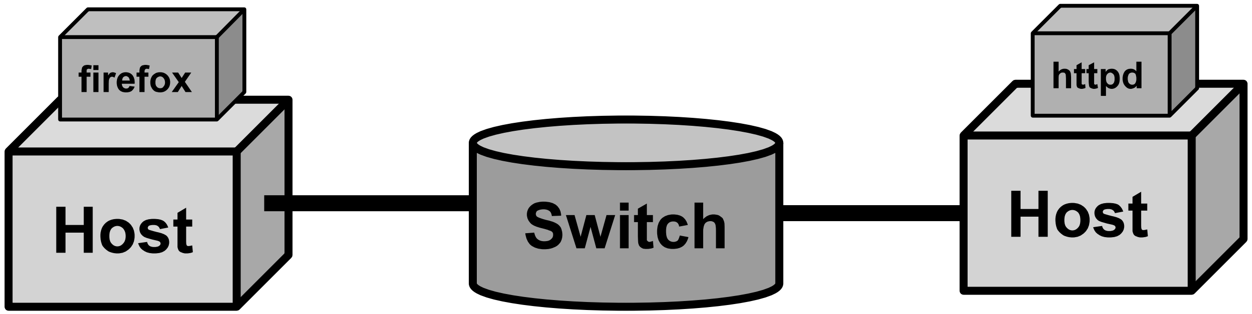
Network Emulator Architecture

Mininet: Basic Usage, CLI, API

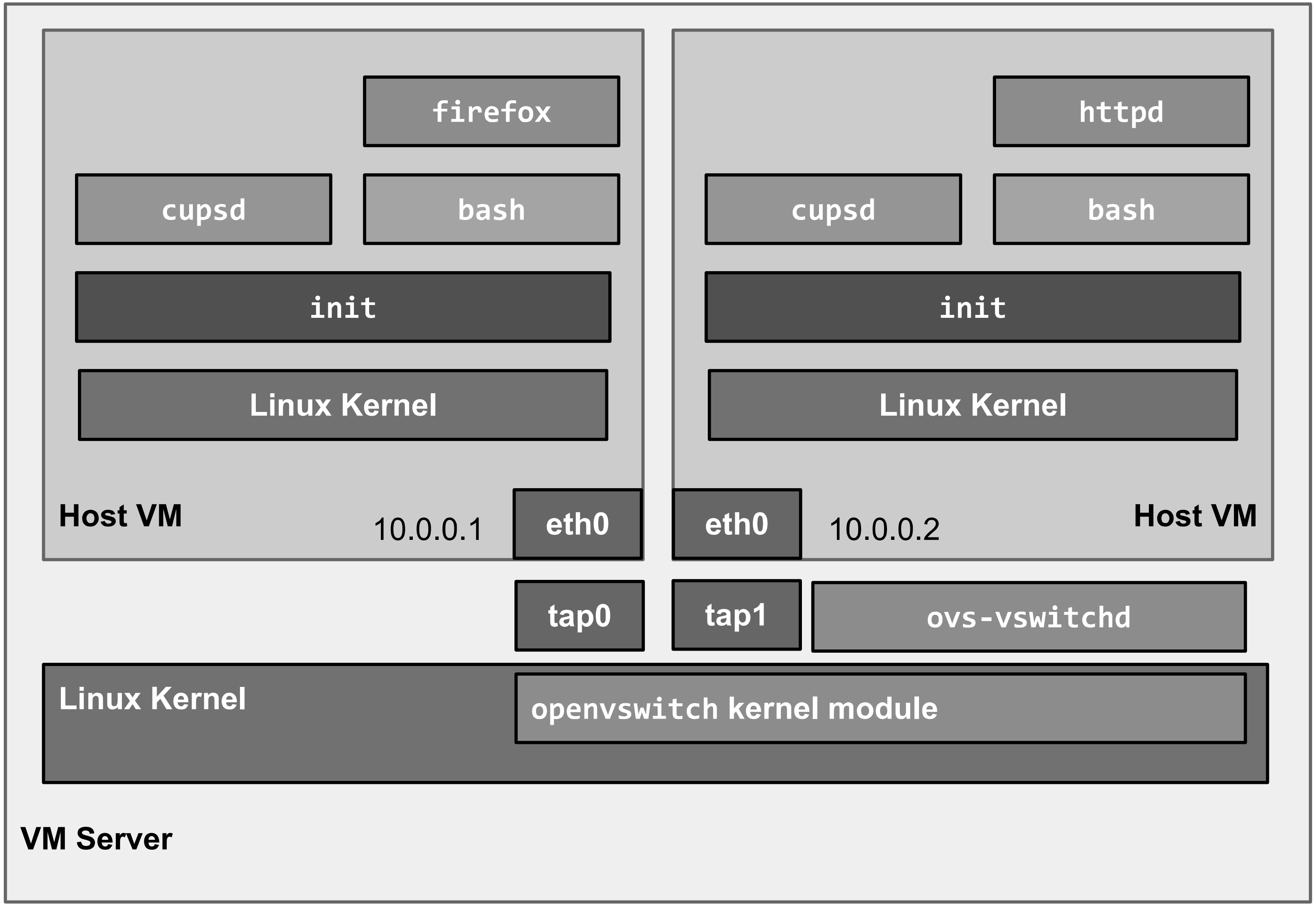
Example Demos: Network Security

Conclusion and Questions

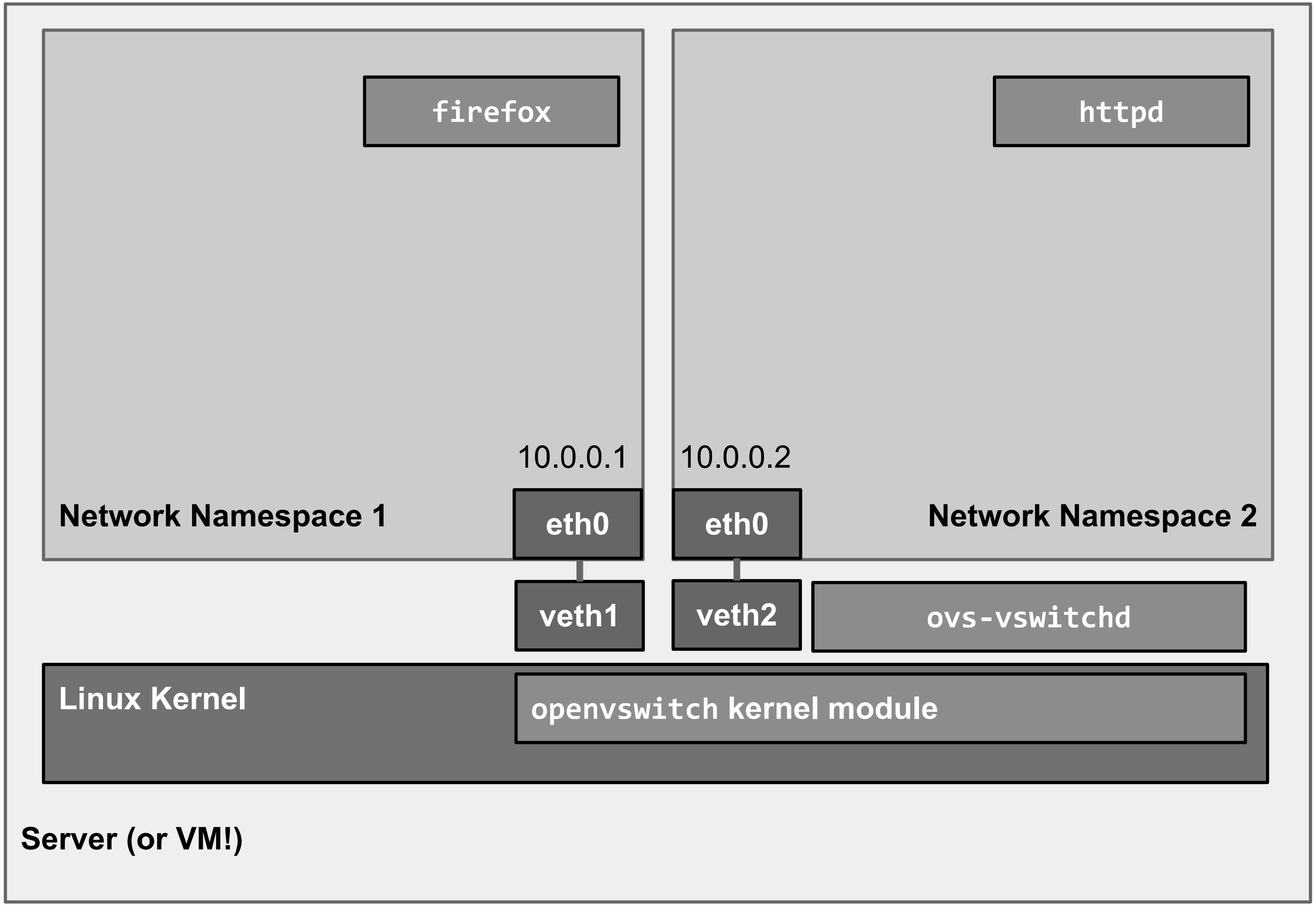
To start with,
a Very Simple Network



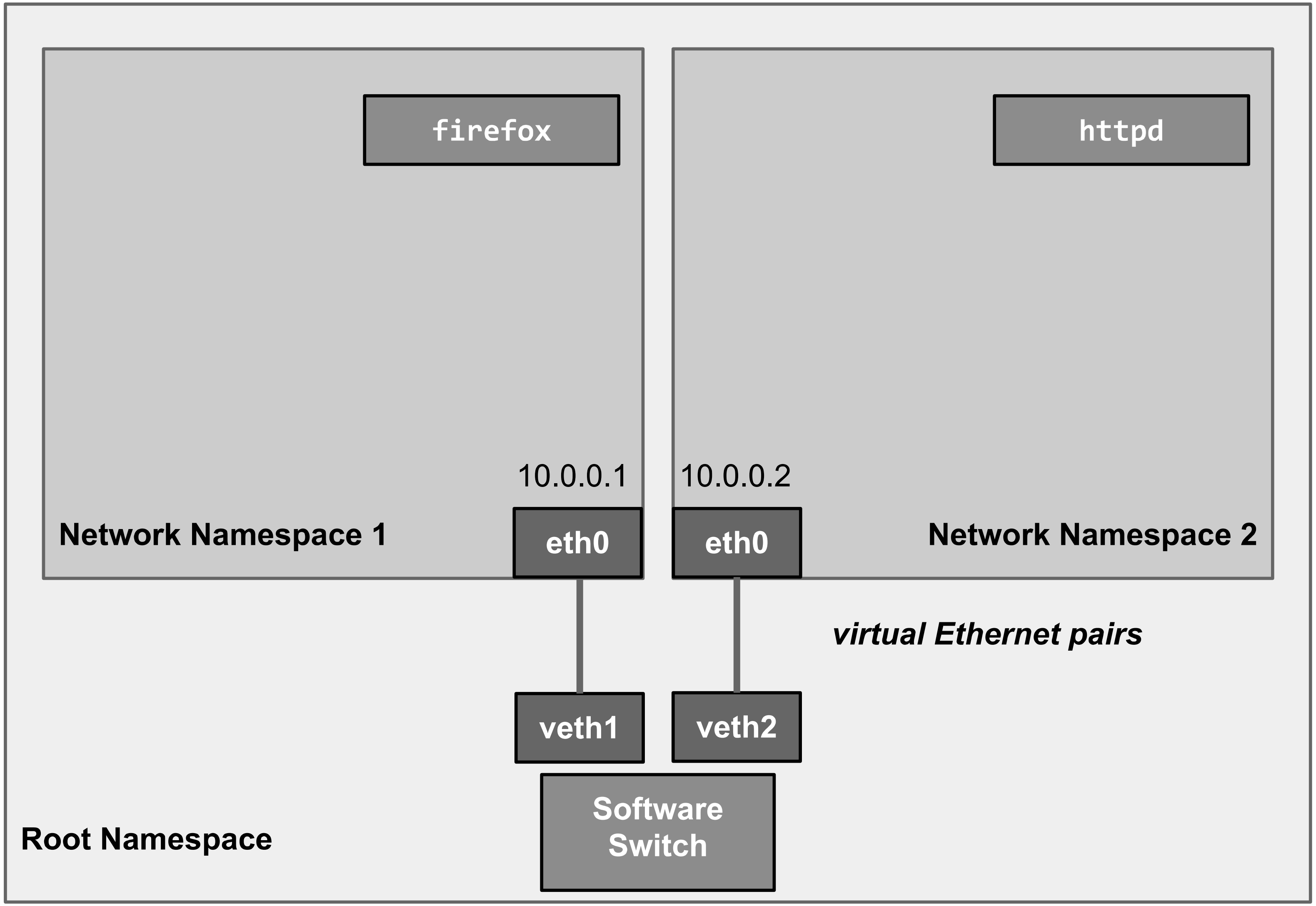
Very Simple Network using Full System Virtualization



Very Simple Network using Lightweight Virtualization



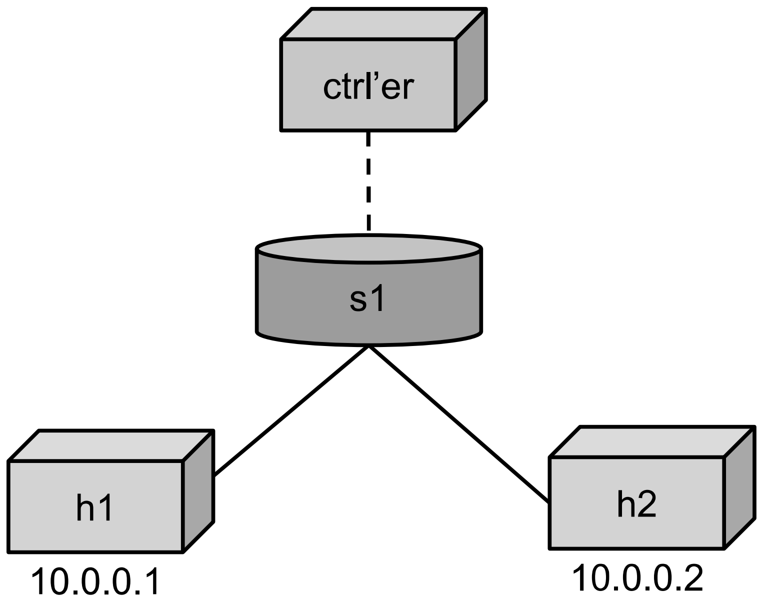
Mechanism: Network Namespaces and Virtual Ethernet Pairs



Creating it with Linux

```
sudo bash
# Create host namespaces
ip netns add h1
ip netns add h2
# Create switch
ovs-vsctl add-br s1
# Create links
ip link add h1-eth0 type veth peer name s1-eth1
ip link add h2-eth0 type veth peer name s1-eth2
ip link show
# Move host ports into namespaces
ip link set h1-eth0 netns h1
ip link set h2-eth0 netns h2
ip netns exec h1 ip link show
ip netns exec h2 ip link show
# Connect switch ports to OVS
ovs-vsctl add-port s1 s1-eth1
ovs-vsctl add-port s1 s1-eth2
ovs-vsctl show
# Set up OpenFlow controller
ovs-vsctl set-controller s1 tcp:127.0.0.1
ovs-controller ptcp: &
ovs-vsctl show
```

```
# Configure network
ip netns exec h1 ifconfig h1-eth0 10.1
ip netns exec h1 ifconfig lo up
ip netns exec h2 ifconfig h2-eth0 10.2
ip netns exec h1 ifconfig lo up
ifconfig s1-eth1 up
ifconfig s1-eth2 up
# Test network
ip netns exec h1 ping -c1 10.2
```

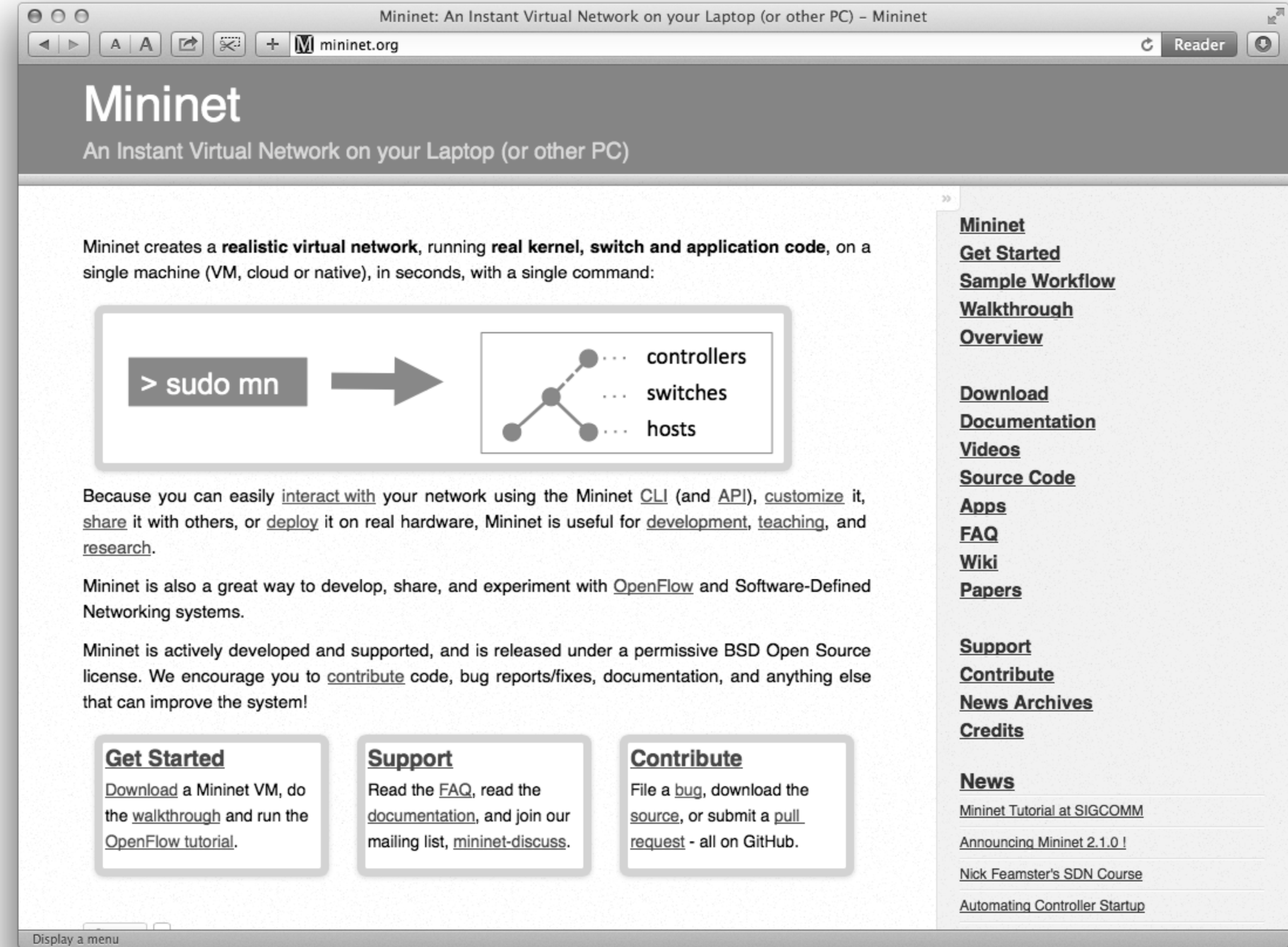


Wouldn’t it be great if...

We had a simple command-line tool and/or API that did this for us automatically?

It allowed us to easily create topologies of varying size, up to hundreds of nodes, and run tests on them?

It was already included in Ubuntu?


Mininet: An Instant Virtual Network on your Laptop (or other PC) – Mininet
mininet.org
Reader
Mininet
An Instant Virtual Network on your Laptop (or other PC)
Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native), in seconds, with a single command:
> sudo mn
controllers
switches
hosts
Because you can easily interact with your network using the Mininet CLI (and API), customize it, share it with others, or deploy it on real hardware, Mininet is useful for development, teaching, and research.
Mininet is also a great way to develop, share, and experiment with OpenFlow and Software-Defined Networking systems.
Mininet is actively developed and supported, and is released under a permissive BSD Open Source license. We encourage you to contribute code, bug reports/fixes, documentation, and anything else that can improve the system!
Get Started
Download a Mininet VM, do the walkthrough and run the OpenFlow tutorial.
Support
Read the FAQ, read the documentation, and join our mailing list, mininet-discuss.
Contribute
File a bug, download the source, or submit a pull request - all on GitHub.
Mininet
Get Started
Sample Workflow
Walkthrough
Overview
Download
Documentation
Videos
Source Code
Apps
FAQ
Wiki
Papers
Support
Contribute
News Archives
Credits
News
Mininet Tutorial at SIGCOMM
Announcing Mininet 2.1.0 !
Nick Feamster's SDN Course
Automating Controller Startup
Display a menu

Introduction to Mininet

- Platforms for Network/Systems Teaching
- Network Emulator Architecture
- **Mininet: Basic Usage, CLI, API**
- Example Demos: Network Security
- Conclusion and Questions

Mininet command line tool and CLI demo

```
# mn
# mn --topo tree,depth=3,fanout=3 --link=tc,bw=10
mininet> xterm h1 h2
h1# wireshark &
h2# python -m SimpleHTTPServer 80 &
h1# firefox &
# mn --topo linear,100
# mn --custom custom.py --topo mytopo
```

Mininet's Python API

Core of Mininet!! Everything is built on it.

Python >> JSON/XML/etc.

Easy and (hopefully) fun

Python is used for *orchestration*, but emulation is performed by compiled C code (Linux + switches + apps)

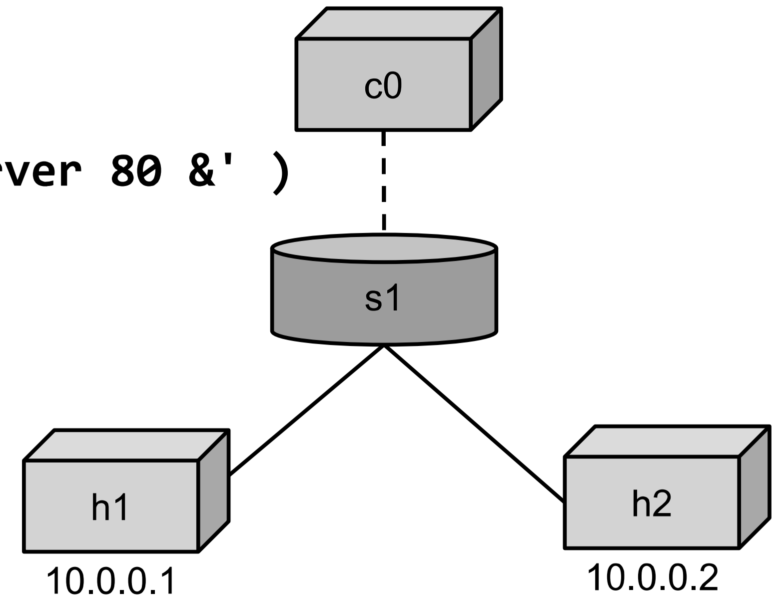
api.mininet.org

docs.mininet.org

Introduction to Mininet

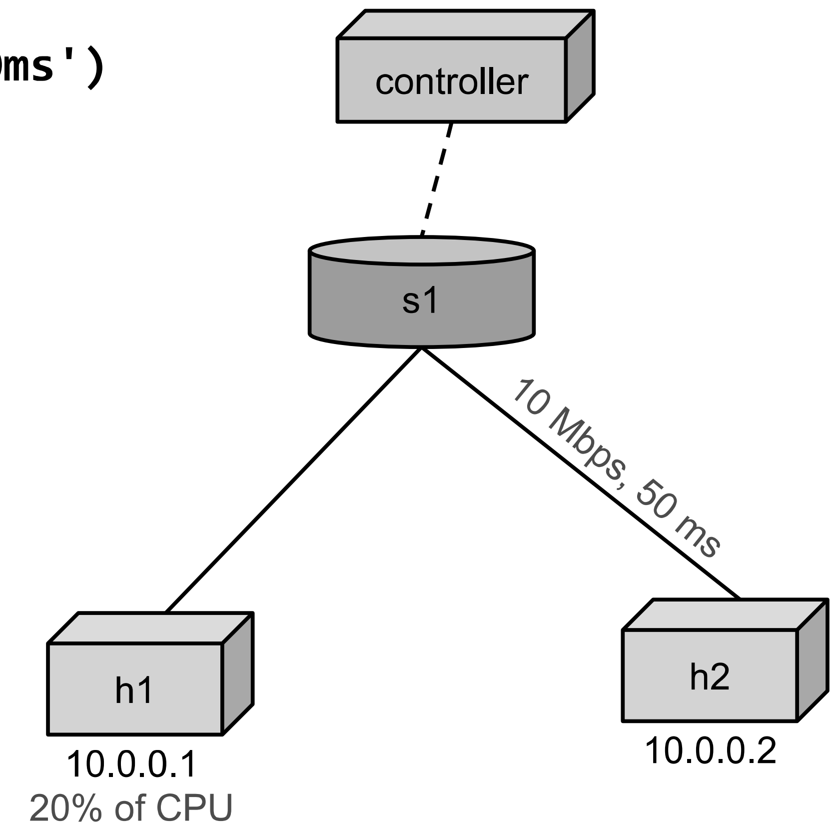
Mininet API basics

```
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
c0 = net.addController( 'c0' )   # c0 is a Controller()
net.addLink( h1, s1 )            # creates a Link() object
net.addLink( h2, s1 )
net.start()
h2.cmd( 'python -m SimpleHTTPServer 80 &' )
sleep( 2 )
h1.cmd( 'curl', h2.IP() )
CLI( net )
h2.cmd('kill %python')
net.stop()
```



Performance modeling in Mininet

```
# Use performance-modeling link and host classes
net = Mininet(link=TCLink, host=CPULimitedHost)
# Limit link bandwidth and add delay
net.addLink(h2, s1, bw=10, delay='50ms')
# Limit CPU bandwidth
net.addHost('h1', cpu=.2)
```



Low-level API: Nodes and Links

```
h1 = Host( 'h1' )
h2 = Host( 'h2' )
s1 = OVSSwitch( 's1', inNamespace=False )
c0 = Controller( 'c0', inNamespace=False )
Link( h1, s1 )
Link( h2, s1 )
h1.setIP( '10.1/8' )
h2.setIP( '10.2/8' )
c0.start()
s1.start( [ c0 ] )
print h1.cmd( 'ping -c1', h2.IP() )
s1.stop()
c0.stop()
```

Mid-level API: Network object

```
net = Mininet()
h1 = net.addHost( 'h1' )
h2 = net.addHost( 'h2' )
s1 = net.addSwitch( 's1' )
c0 = net.addController( 'c0' )
net.addLink( h1, s1 )
net.addLink( h2, s1 )
net.start()
print h1.cmd( 'ping -c1', h2.IP() )
CLI( net )
net.stop()
```

High-level API: Topology templates

```
class SingleSwitchTopo( Topo ):
    "Single Switch Topology"
    def build( self, count=1):
        hosts = [ self.addHost( 'h%d' % i )
                  for i in range( 1, count + 1 ) ]
        s1 = self.addSwitch( 's1' )
        for h in hosts:
            self.addLink( h, s1 )

net = Mininet( topo=SingleSwitchTopo( 3 ) )
net.start()
CLI( net )
net.stop()
```

more examples and info available at docs.mininet.org

Custom Topology Files

```
# cat custom.py
from mininet.topo import Topo
class SingleSwitchTopo( Topo ):
    "Single Switch Topology"
    def build( self, count=1):
        hosts = [ self.addHost( 'h%d' % i )
                  for i in range( 1, count + 1 ) ]
        s1 = self.addSwitch( 's1' )
        for h in hosts:
            self.addLink( h, s1 )
topos = { 'mytopo': SingleSwitchTopo }
# mn --custom custom.py --topo mytopo,3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
```

Introduction to Mininet

Platforms for Network/Systems Teaching

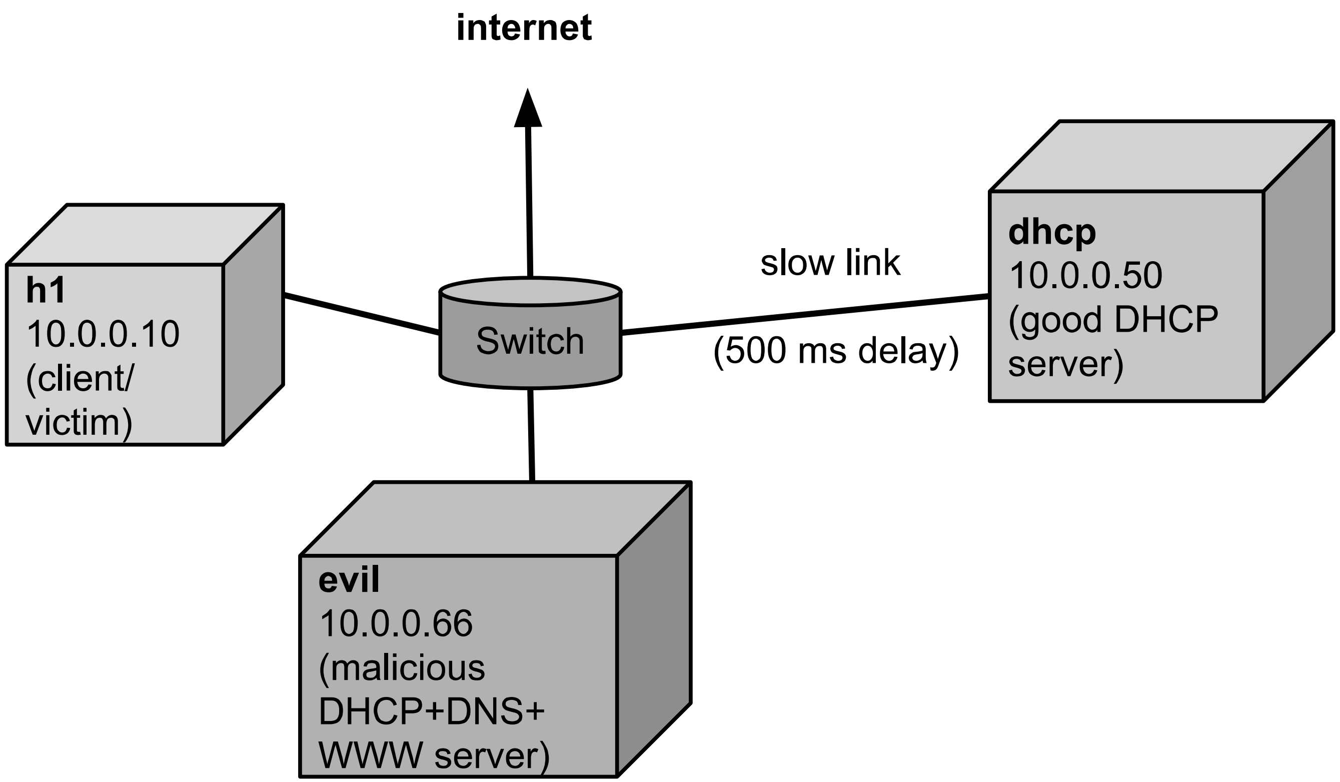
Network Emulator Architecture

Mininet: Basic Usage, CLI, API

Example Demos: Network Security

Conclusion and Questions

Security Demo #1: DHCP Attack



Security Demo #2: BGP

More Demos!

MiniEdit

Consoles.py

Cluster prototype

Introduction to Mininet

Platforms for Network/Systems Teaching

Network Emulator Architecture

Mininet: Basic Usage, CLI, API

Example Demos: Network Security

Conclusion and Questions

Conclusion and Questions

Network Emulators **can facilitate teaching networking via realistic live demos, interactive labs and course assignments**

- inexpensive, interactive, real apps and OS, reasonably accurate
- downloadable, fast setup

Mininet **is a lightweight virtualization/container based emulator**

- modest hardware requirements, fast startup, hundreds of nodes
- command line tool, CLI, simple Python API
- SDN as well as Ethernet/IP networking as well as SD
- install using VM, Ubuntu package, or source

mininet.org: Tutorials, walkthroughs, API documentation and examples

teaching.mininet.org: Mininet-based **course assignments and labs**

open source: hosted on github, permissive BSD license

Next up: short break, then hands-on lab!

Tutorial Agenda

1. Introduction to Mininet

presentation, demos, **short break**

2. Hands-on Lab

presentation, lab, coffee break

3. Teaching with Mininet

presentations, discussion, done!

Backup/Supplementary Slides

Mininet is a *Network Emulator*

In this talk, ***emulation*** (or running on an **emulator**) means running *unmodified* code *interactively* on *virtual hardware* on a *regular PC*, providing convenience and realism at low cost – with some limitations (e.g. speed, detail.)

This is in contrast to running on a **hardware testbed** (fast, accurate, expensive/shared) or a **simulator** (cheap, detailed, but perhaps slow and requiring code modifications.)

Context: Platforms for Network Experimentation and Development

Container-based emulators: CORE, virtual Emulab, Trellis, Imunes, even ns-3 (in emulation mode), **Mininet**

VM-based emulators: DieCast

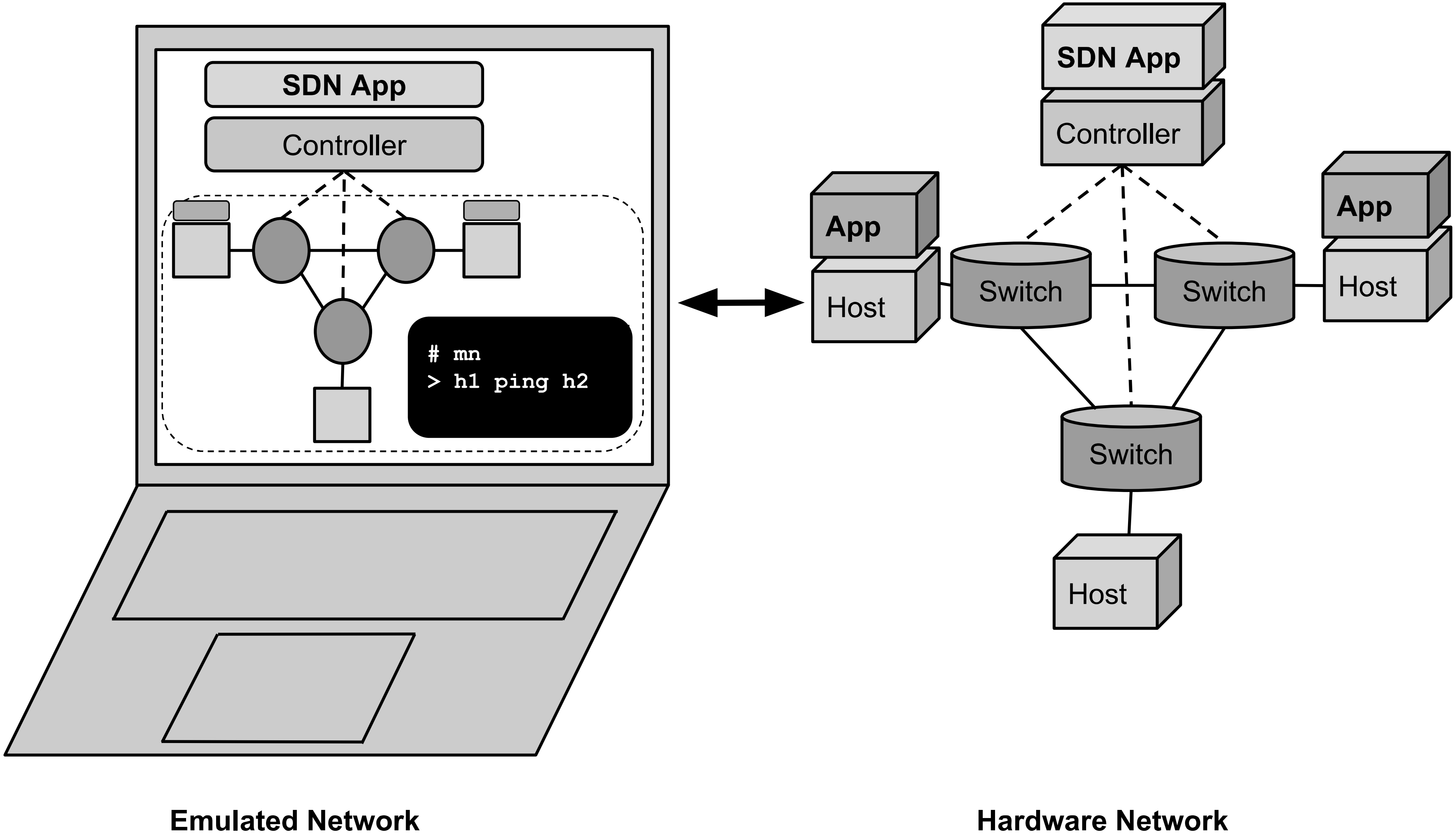
UML-based emulators: NetKit

Simulators: ns-3, OPNET

Testbeds: Emulab, GENI, PlanetLab, ORBIT

All of these are fine, but we think Emulators are particularly useful! Why? Because...

Apps move seamlessly to/from hardware

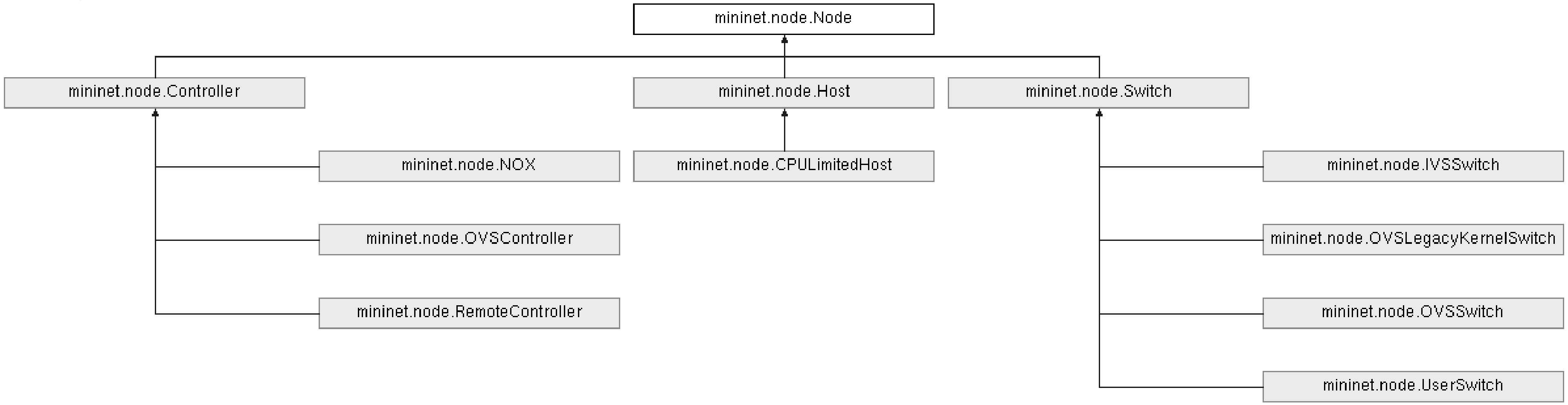


Appendix: Mininet Subclassing for Fun and Profit

Bob Lantz, Brian O'Connor

Classes in Mininet

Inheritance diagram for mininet.node.Node:



Example

```
class Host( Node ):
    "A host is simply a Node"
    pass
```

What do you want to customize?

```
class Node( object ):
    def config( self, mac=None, ip=None,
                defaultRoute=None, lo='up', **_params ):

        # If we were overriding this method, we would call
        # the superclass config method here as follows:
        # r = Parent.config( **_params )
        r = {}
        self.setParam( r, 'setMAC', mac=mac )
        self.setParam( r, 'setIP', ip=ip )
        self.setParam( r, 'setDefaultRoute', defaultRoute=defaultRoute )
        self.cmd( 'ifconfig lo ' + lo )
        return r
```

Customizing `Host()`

```
class VLANHost( Host ):
   def config( self, vlan=100, **params ):
        """Configure VLANHost according to (optional) parameters:
           vlan: VLAN ID for default interface"""
        r = super( Host, self ).config( **params )
        intf = self.defaultIntf()
        self.cmd( 'ifconfig %s inet 0' % intf )  # remove IP from default, "physical" interface
        self.cmd( 'vconfig add %s %d' % ( intf, vlan ) )  # create VLAN interface
        self.cmd( 'ifconfig %s.%d inet %s' % ( intf, vlan, params['ip'] ) ) # assign the host's IP to
                                                                            the VLAN interface
        # to maintain CLI compatibility
        newName = '%s.%d' % ( intf, vlan )  # update the intf name and host's intf map
        intf.name = newName  # update the (Mininet) interface to refer to VLAN interface name
        self.nameToIntf[ newName ] = intf  # add VLAN interface to host's name to intf map
        return r

hosts = { 'vlan': VLANHost }
```

Using Custom Hosts

In Python:

```
def run( vlan ):
    # vlan (type: int): VLAN ID to be used by all hosts
    host = partial( VLANHost, vlan=vlan )

    # Start a basic network using our VLANHost
    topo = SingleSwitchTopo( k=2 )
    net = Mininet( host=host, topo=topo )
    net.start()
    CLI( net )
    net.stop()
```

From the CLI:

```
sudo mn --custom vlanhost.py --host vlan,vlan=1000
```

Customizing `Switch()`

```
class LinuxBridge( Switch ):
      "Linux Bridge"

      prio = 0

      def __init__( self, name, stp=True, **kwargs ):
            self.stp = stp
            Switch.__init__( self, name, **kwargs )  # BL doesn’t care about multiple inheritance

      def start( self, controllers ):
            self.cmd( 'ifconfig', self, 'down' )
            self.cmd( 'brctl delbr', self )
            self.cmd( 'brctl addbr', self )
            if self.stp:
                  self.cmd( 'brctl setbridgeprio', self.prio )
                  self.cmd( 'brctl stp', self, 'on' )
                  LinuxBridge.prio += 1
            for i in self.intfList():
                  if self.name in i.name:
                        self.cmd( 'brctl addif', self, i )
            self.cmd( 'ifconfig', self, 'up' )

      def stop( self ):
            self.cmd( 'ifconfig', self, 'down' )
            self.cmd( 'brctl delbr', self )

switches = { 'lxbr': LinuxBridge }
```

Customizing `Switch()`

demo

```
openflow@ubuntu13:~$ sudo mn --custom torus3.py --switch lxbr --topo torus,3,3
…
mininet> sh brctl showstp s0x0
…
```

Customizing Switch()

```
c0 = Controller( 'c0', port=6633 )
c1 = Controller( 'c1', port=6634 )
c2 = RemoteController( 'c2', ip='127.0.0.1' )

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 )
for c in [ c0, c1 ]:
     net.addController(c)
net.start()
CLI( net )
net.stop()
```

DEMO: controllers.py

Customizing Controller()

```
from mininet.node import Controller
from os import environ

POXDIR = environ[ 'HOME' ] + '/pox'

class POX( Controller ):
      def __init__( self, name, cdir=POXDIR,
                  command='python pox.py',
                  cargs=( 'openflow.of_01 --port=%s '
                              'forwarding.l2_learning' ),
                  **kwargs ):
      Controller.__init__( self, name, cdir=cdir, command=command, cargs=cargs, **kwargs )

controllers={ 'pox': POX }
```

Customizing Controller()

```
from mininet.node import Controller
from os import environ
from functools import partial

POXDIR = environ[ 'HOME' ] + '/pox'

POX = partial( Controller, cdir=POXDIR,
                command='python pox.py',
                cargs=( 'openflow.of_01 --port=%s '
                           'forwarding.l2_learning' ) )
controllers = { 'pox': POX }
```