Mininet: An instant Virtual Network on your Laptop

Bob Lantz, Brian O'Connor
work with Brandon Heller, Nikhil Handigol,
Vimal Jeyakumar, and the Mininet Contributors

Talk Outline

Network Emulation

- Why it's awesome

Challenges

- Scalability (demo)
- Ease of use (demo)
- Performance Accuracy (demo)

Interactive Demo

What is Network Emulation?

In this talk, ***emulation*** (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.)

Talk Outline

Network Emulation

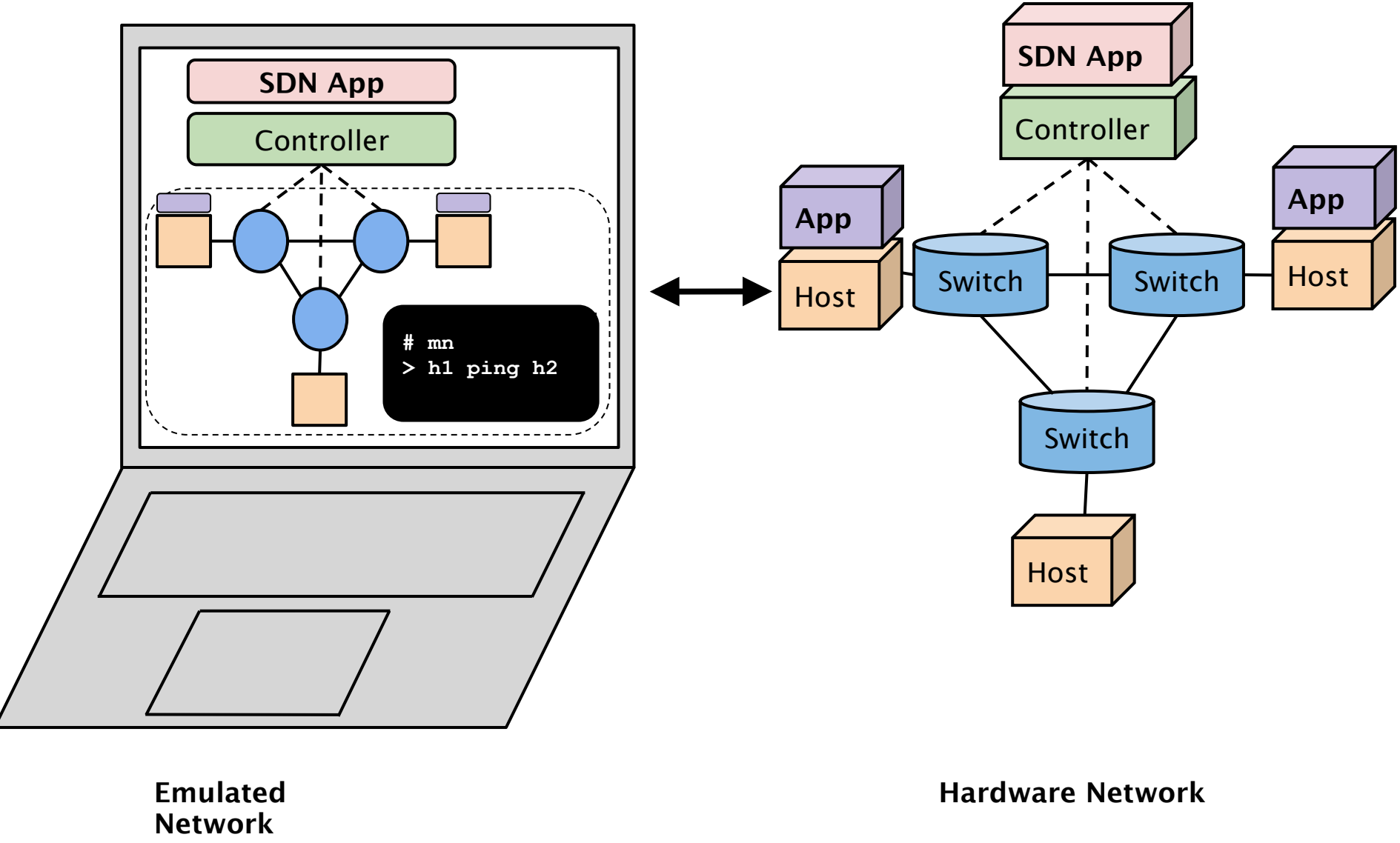
- **Why it's awesome**

Challenges

- Scalability (demo)
- Ease of use (demo)
- Performance Accuracy (demo)

Interactive Demo

Apps move seamlessly to/from hardware



Talk Outline

Network Emulation

– Why it's awesome

Challenges

– **Scalability** (demo)

– Ease of use (demo)

– Performance Accuracy (demo)

Interactive Demo

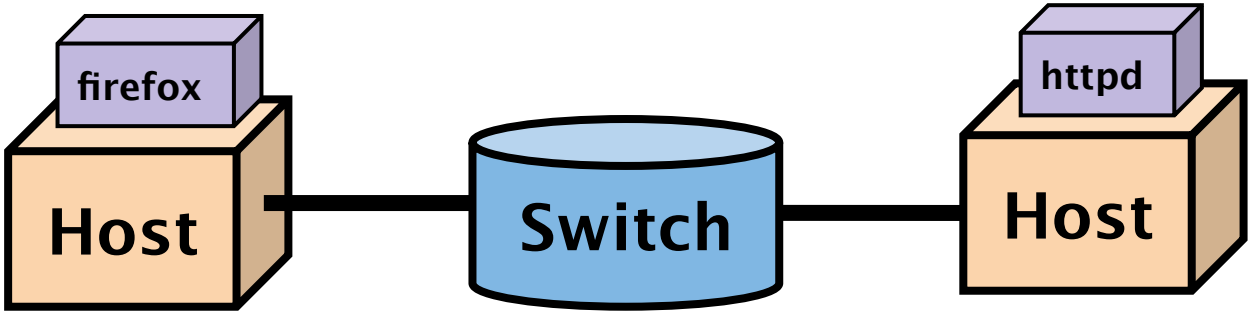
Creating a Network Emulator

Need to model **hosts**, **switches**, **links**, and (possibly) network **controllers** (for SDN/OpenFlow.)

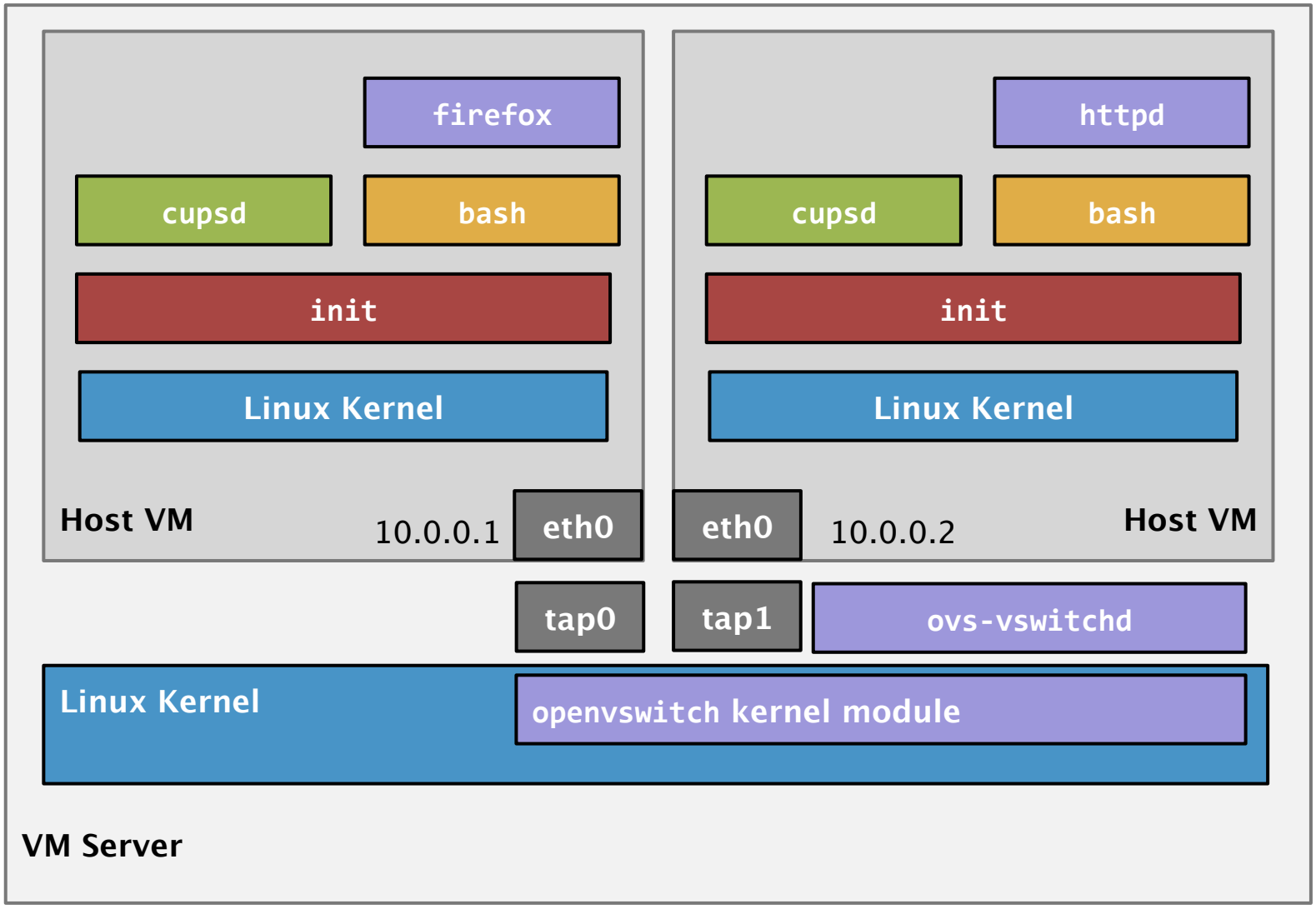
Scalability challenge: we would like to model networks of interesting size with practical performance.

How to do it? Virtualization!

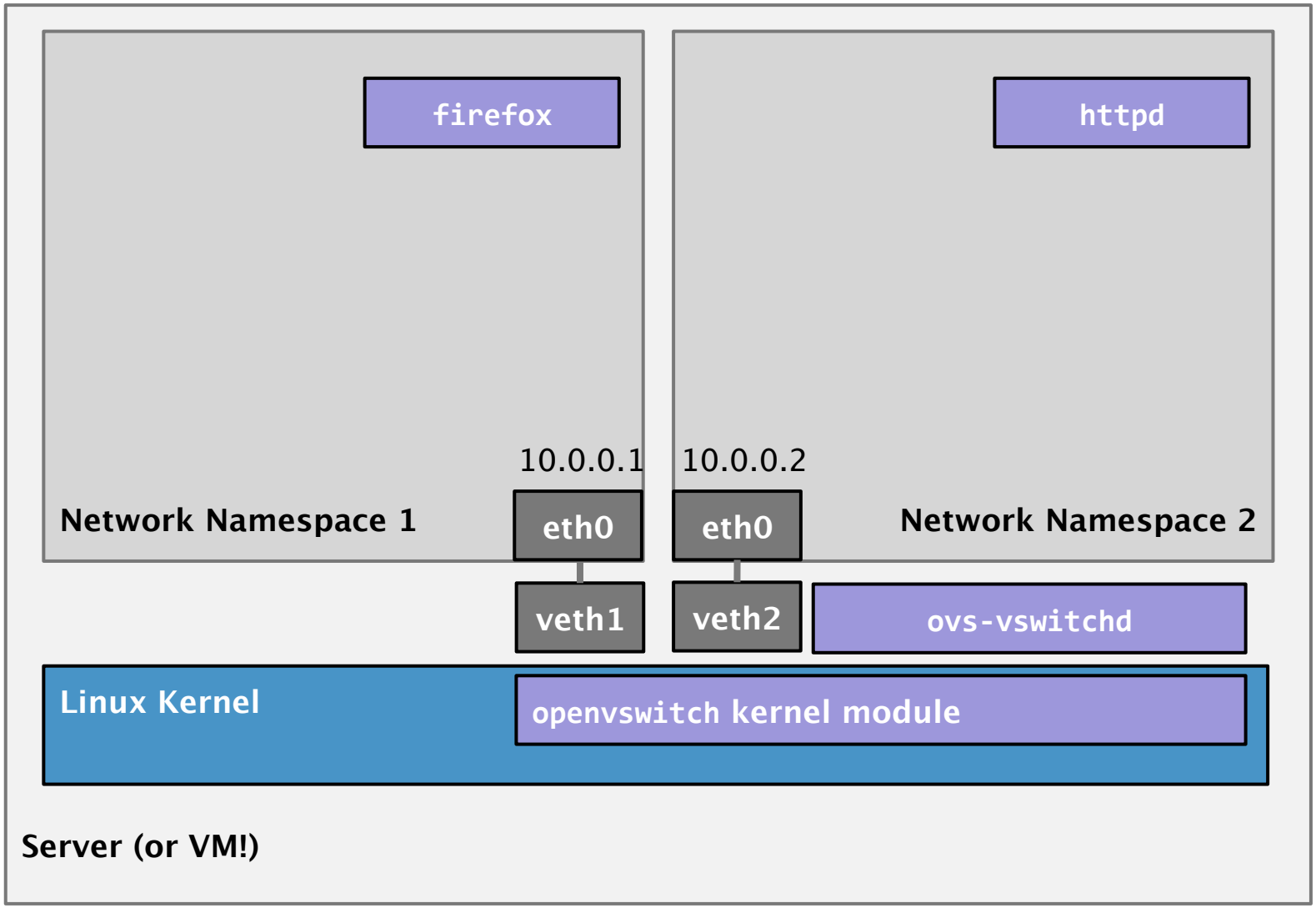
To start with,
a Very Simple Network



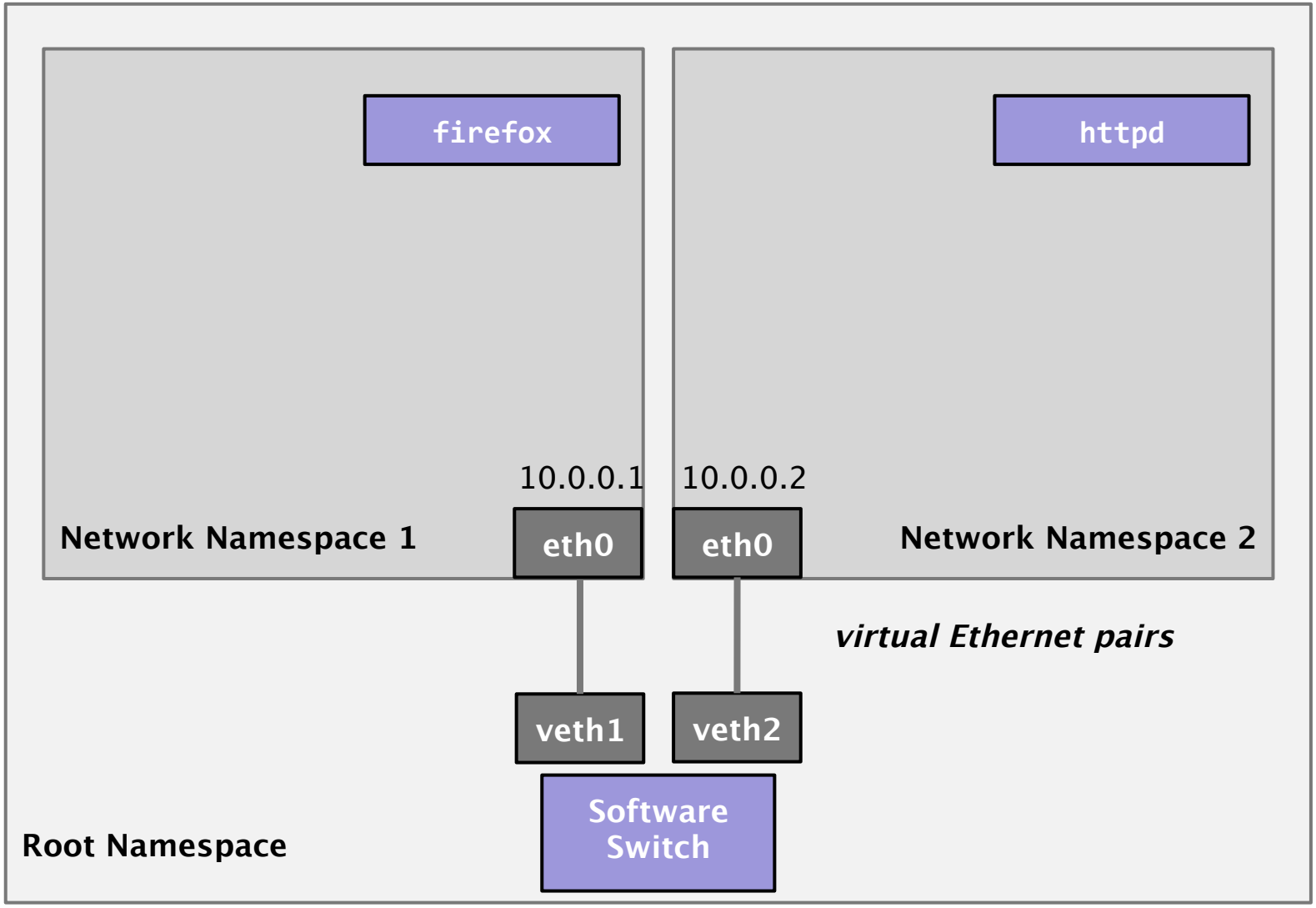
Very Simple Network using Full System Virtualization



Very Simple Network using Lightweight Virtualization



Network Namespaces and Virtual Ethernet Pairs

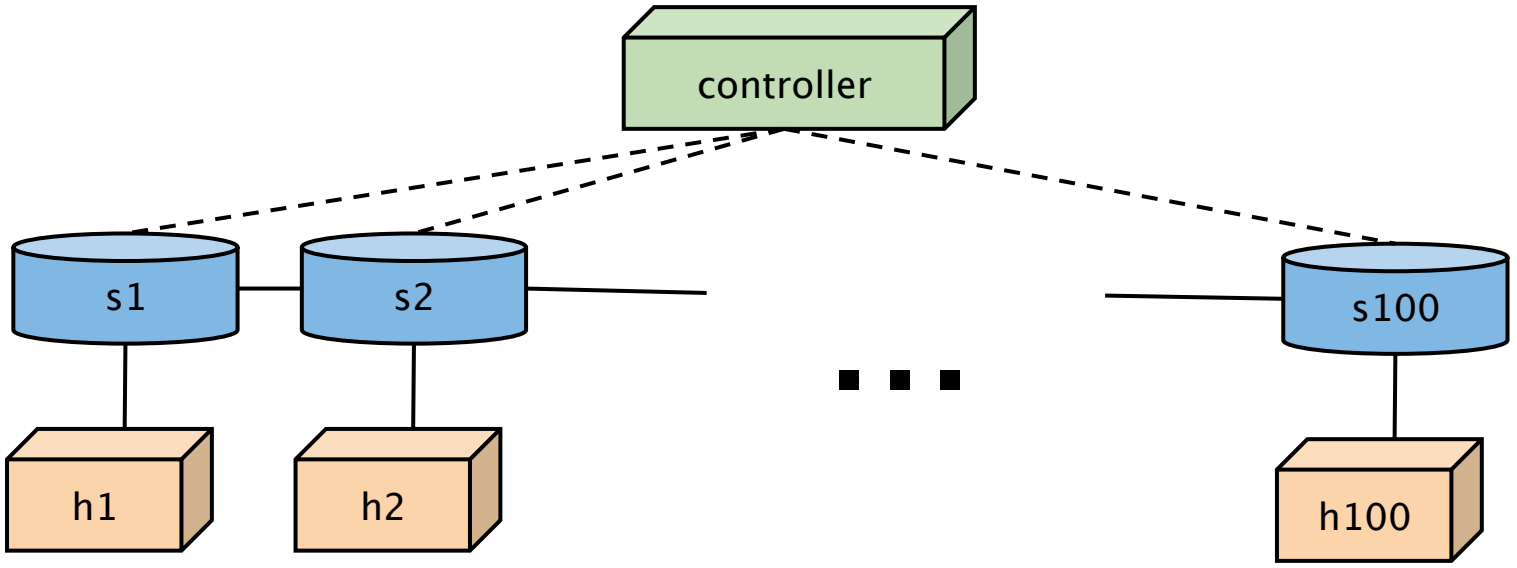


Lightweight Virtualization in Linux

Network component or property	Modeling mechanism	Configuration command(s)
Hosts	Processes in **network namespaces**	`ip netns`
Links	Virtual Ethernet pairs	`ip link`
Switches	Software switches (OVS)	`ovs-vsctl`
Controllers	Processes	`controller`

Demo

```
# mn --topo linear,100 --switch user
 --controller ref
```



Talk Outline

Network Emulation

- Why it's awesome

Challenges

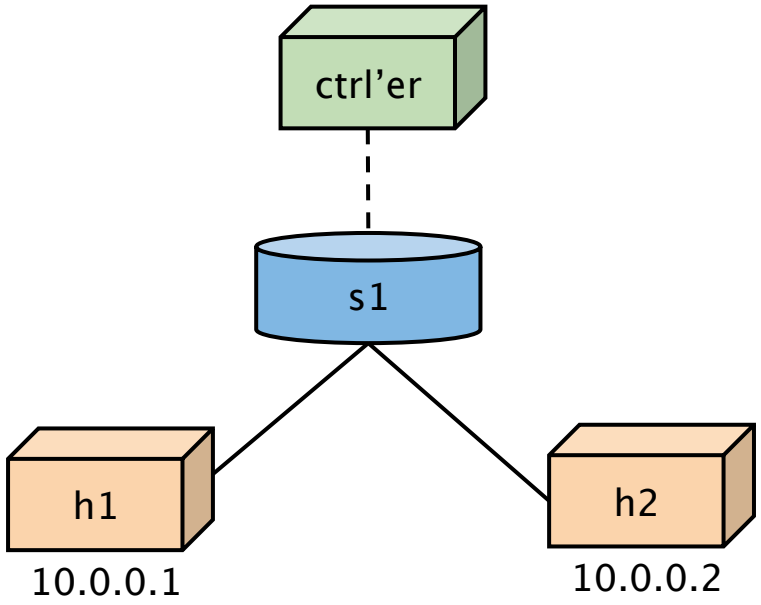
- Scalability (demo)
- **Ease of use** (demo)
- Performance Accuracy (demo)

Interactive Demo

Demo: basic network setup in 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...

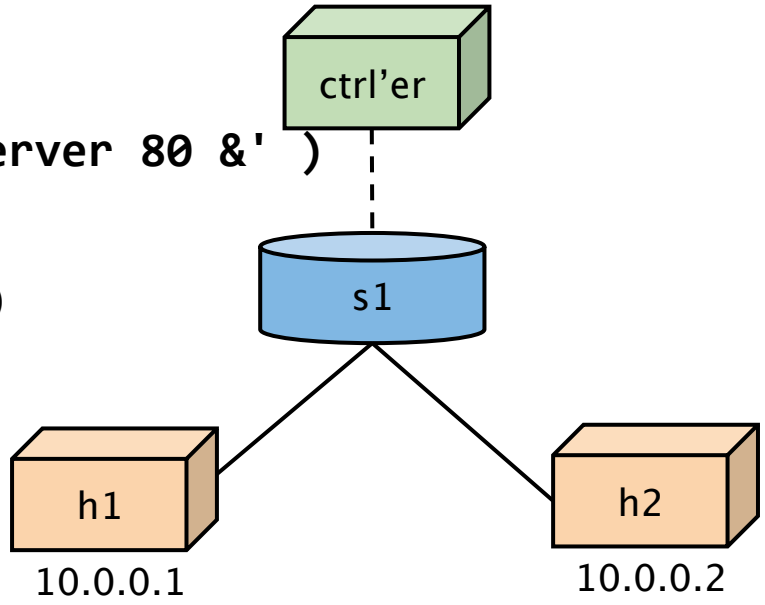
We had a simple command-line tool and/or API that did this for us automatically?

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?

Basic network setup in Mininet (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
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 )
print h1.cmd( 'curl', h2.IP() )
CLI( net )
h2.cmd('kill %python')
net.stop()
```



mn command and Mininet CLI demo

```
# mn --test pingall

# mn --topo tree,depth=3,fanout=3 --
 link=tc,bw=10

mininet> xterm h1 h2
h1# wireshark &
h2# python -m SimpleHTTPServer 80
h1# firefox &
```

Talk Outline

Network Emulation

– Why it's awesome

Challenges

– Scalability (demo)

– Ease of use (demo)

– **Performance Accuracy** (demo)

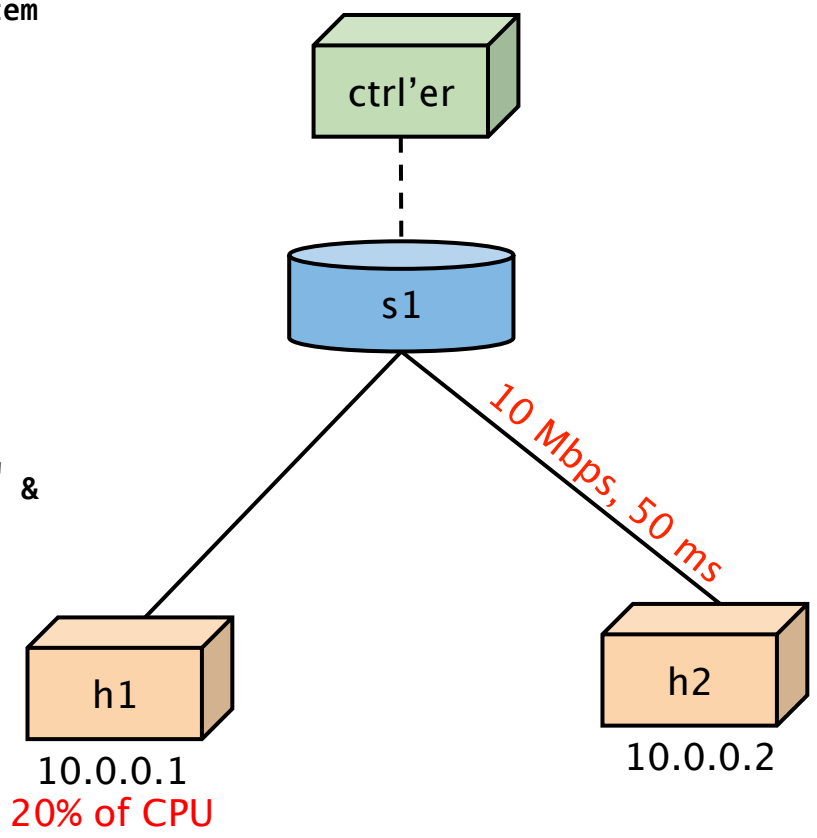
Interactive Demo

Performance modeling with Linux

Network component or property	Modeling mechanism	Configuration command(s)
Hosts	Processes in network namespaces	`ip netns`
Links	Virtual Ethernet pairs	`ip link`
Switches	Software switches (OVS)	`ovs-vsctl`
Controllers	Processes	`controller`
Link performance	Traffic Control (and `netem` subsystem)	`tc`
CPU performance	CPU Control Groups (CFS bandwidth limits)	`cg{create,set,delete, classify}`

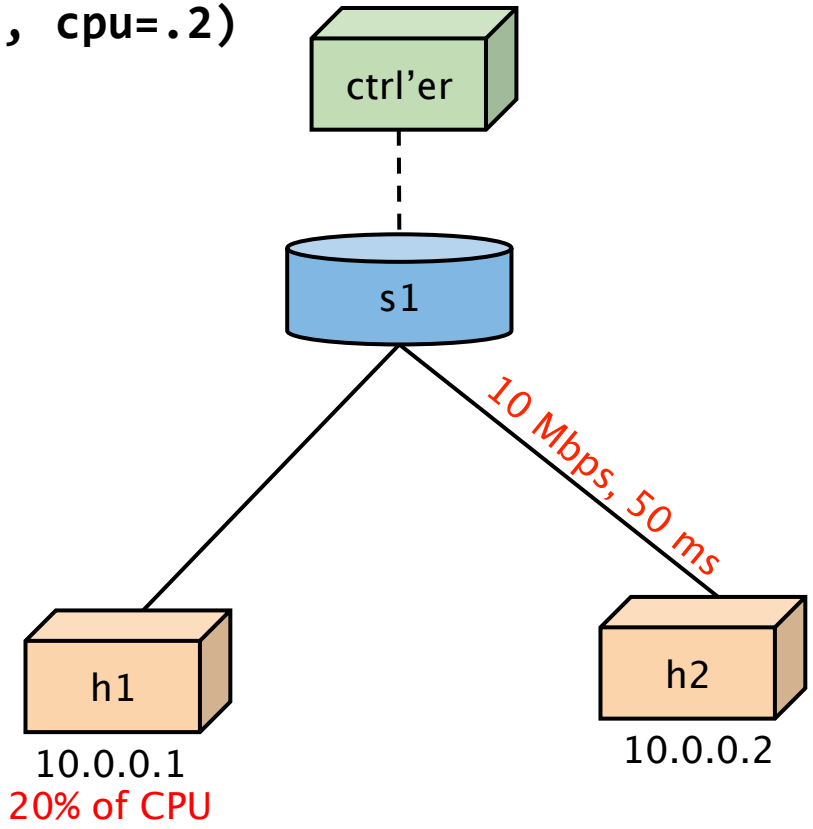
Demo: performance setup in Linux

```
# Limit link bandwidth and add delay
tc qdisc add dev s1-eth2 root handle 5: tbf rate
  10Mbit burst 5k latency 12ms
tc qdisc add dev s1-eth2 parent 5:1 handle 10: netem
  delay 50ms
ip netns exec h1 ping -c4 10.2
ip netns exec h2 iperf -s >& /dev/null &
ip netns exec h1 iperf -t 5 -c 10.2
# Limit CPU bandwidth
cgcreate -g cpu:/h1
cgset -r cpu.cfs_period_us=100000 /h1
cgset -r cpu.cfs_quota_us=20000 /h1
ip netns exec h1 bash -c "while true; do a=1;done" &
cgclassify -g cpu:/h1 $!
```



Performance setup in Mininet

```
# Limit link bandwidth and add delay
net.addLink(h2, s1, cls=TCLink,
            bw=10, delay='50ms')
# Limit CPU bandwidth
net.addHost('h1', cls=CPULimitedHost, cpu=.2)
```



Accuracy = Matching hardware

Experiments on emulator should match results on hardware.

How to test? Micro/macrobenchmarks?

http://hci.stanford.edu/cstr/reports/2012-02.pdf

Better (and bigger) idea: Grad students!

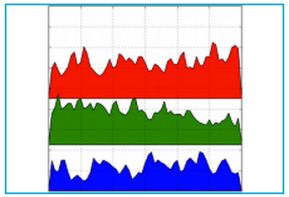
reproducingnetworkresearch.wordpress.com

REPRODUCING NETWORK RESEARCH

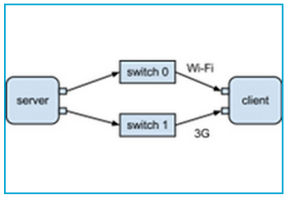
network systems experiments made accessible, runnable, and reproducible

projects / about / contribute

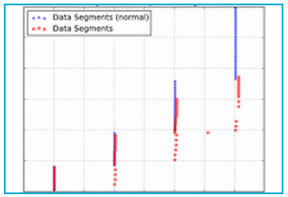
Posts by CS244 Spring 2012 Students



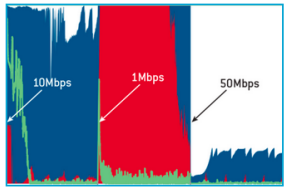
Exploring Outcast



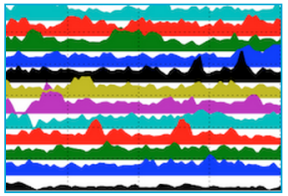
Multipath TCP over WiFi and 3G links



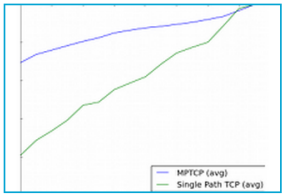
TCP Daytona: Congestion Control with a Misbehaving Receiver



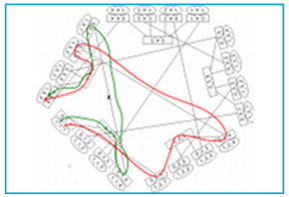
Solving Bufferbloat – The CoDel Way



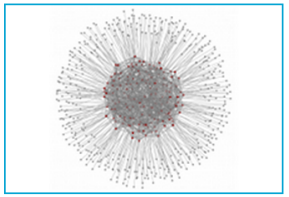
Life's not fair, neither is TCP (… under the following conditions)



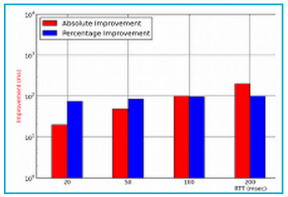
Fairness of Jellyfish vs. Fat-Tree



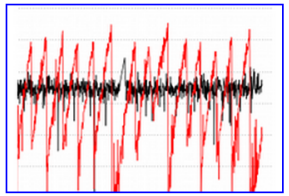
DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers



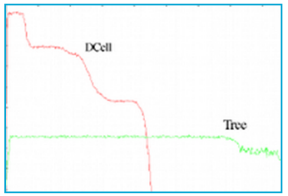
Jellyfish vs. Fat Tree



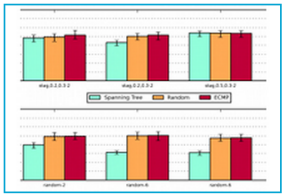
Choosing the Default Initial Congestion Window



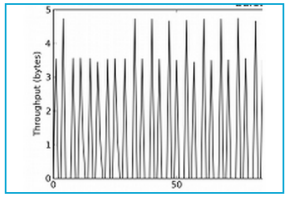
DCTCP and Queues



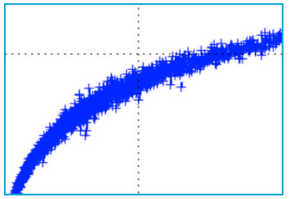
DCell : A Scalable and Fault-Tolerant Network Structure for Data Centers



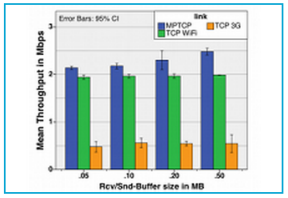
Hedera



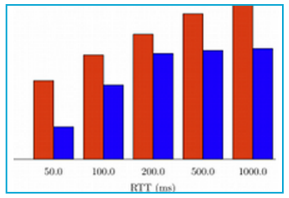
Seeing RED



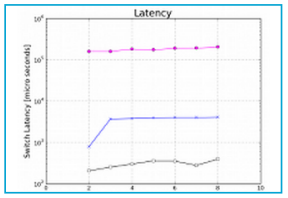
Why Flow-Completion Time is the Right Metric for Congestion Control



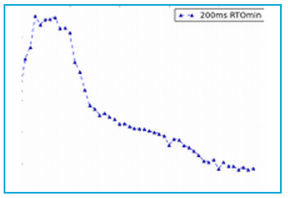
MPTCP Wireless Performance



Increasing TCP's Initial Congestion Window



HULL: High Bandwidth, Ultra Low Latency



TCP Incast Collapse

Talk Outline

Network Emulation

- Why it's awesome

Challenges

- Scalability (demo)
- Ease of use (demo)
- Performance Accuracy (demo)

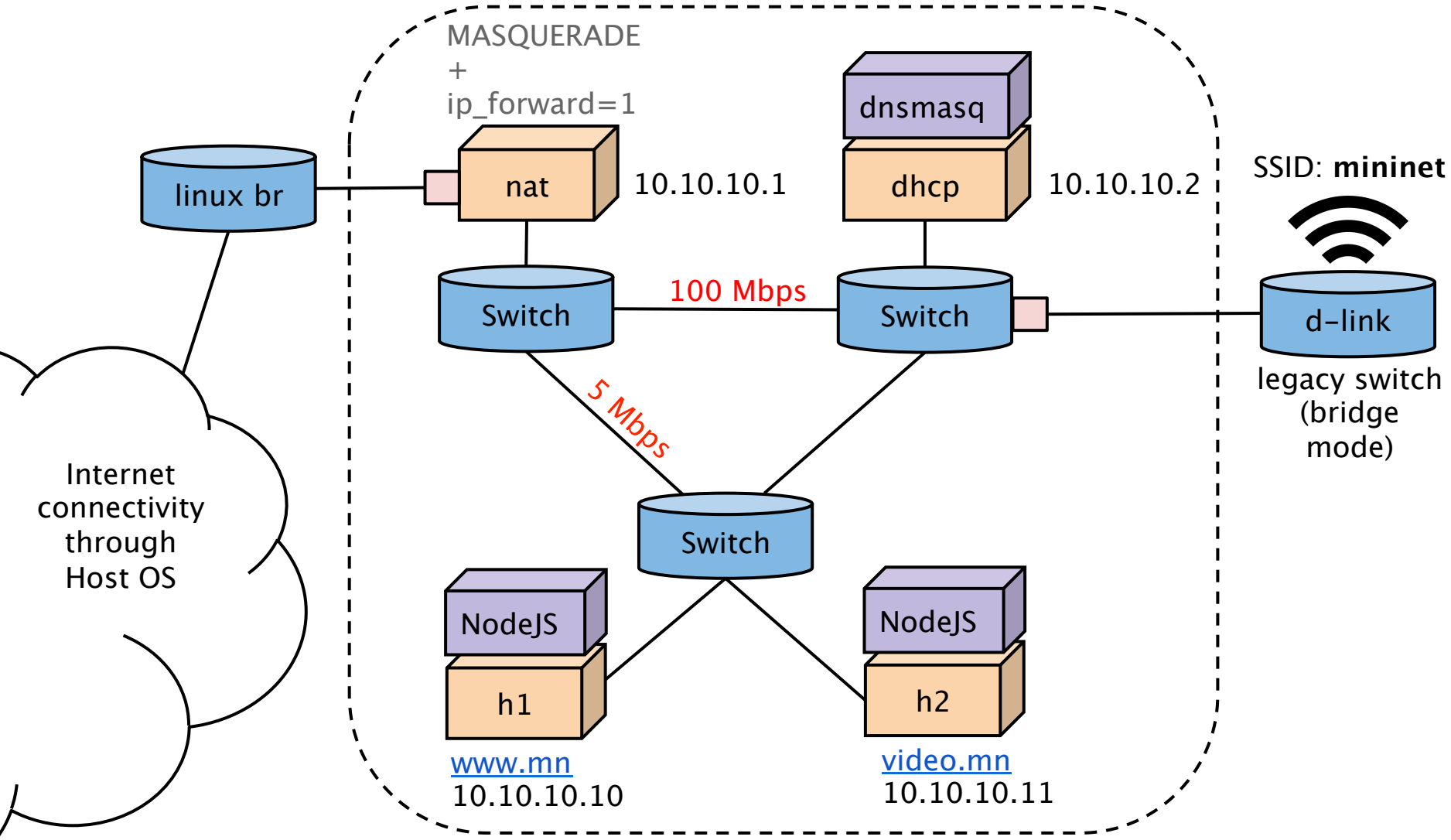
Interactive Demo

Demo: Connect to Mininet with your phone or laptop!

Interactive demo here in Gates 104:

1) **connect to "mininet" wi-fi network**
2) check your IP address if you like
3) go to www.mn
4) go to google.com or any other site

Demo Topology



```
class DemoTopo( Topo ):
  def __init__(self, inetIntf, wlanIntf, **opts):
    Topo.__init__(self, **opts)
    s1 = self.addSwitch( 's1' )
    s2 = self.addSwitch( 's2' )
    s3 = self.addSwitch( 's3' )
    # connect switches in a triangle
    self.addLink( s1, s2, cls=TCLink, bw=5, delay='1ms', max_queue_size=20 )
    self.addLink( s2, s3 )
    self.addLink( s1, s3, cls=TCLink, bw=100, delay='1ms', max_queue_size=20 )
    # add servers and connect to s2
    h1 = self.addHost( 'h1', ip='10.10.10.10/24', defaultRoute='via 10.10.10.1' )
    h2 = self.addHost( 'h2', ip='10.10.10.11/24', defaultRoute='via 10.10.10.1' )
    self.addLink( s2, h1 )
    self.addLink( s2, h2 )
    # add NAT and connect it to s1
    nat = self.addNode( 'nat', ip='10.10.10.1/24', cls=NAT,
                        subnet=10.10.10.0/24, inetIntf=inetIntf, inNamespace=False )
    self.addLink( s1, nat )
    # add DHCP server and connect it to s3
    dhcp = self.addNode( 'dhcp', ip='10.10.10.2/24', cls=DHCPServer, defaultRoute='via 10.10.10.1' )
    self.addLink( s3, dhcp )
    # add the WiFi interface to s3
    self.addLink( s3, s3, cls=HWIntfLink, intfName1=wlanIntf, cls2=NoneIntf )
```

Enjoy Mininet!

mininet.org

docs.mininet.org

teaching.mininet.org

reproducingnetworkresearch.wordpress.com

Network Emulation
- Why it's awesome

Challenges
- Scalability (demo)
- Performance (demo)
- Ease of use (demo)

Interactive Demo

Backup Slides

Predictive Accuracy: *Network Invariants*

Idea: Emulator should not violate conditions that we know must be true.

For example, for non-empty switch queue, output data rate should be constant and packets should be evenly spaced.

If we can specify and monitor invariants, simulation is more likely to be accurate.

More info at:

http://purl.stanford.edu/zk853sv3422

Demo: Reproducible experiments and a “Runnable” Paper

Reproducible Network Experiments using Container-Based Emulation

http://conferences.sigcomm.org/co-next/2012/eproceedings/conext/p253.pdf

Mininet(.org)

- command line tool and interface (`mn`)
- simple Python API
- parametrized topologies
- link modeling and CPU limits
- scales to hundreds of nodes on a laptop
- free/open source - you can (and should) contribute code on GitHub
- `mininet-discuss` mailing list
- pre-made VM image (easy to run/share)
- included in Ubuntu!

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 __init__( self, count=1, **params ):
        Topo.__init__( self, **params )
        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

Example Mininet use cases

… for Teaching	organization(s)	link
Buffer Bloat (in-class lab)	Stanford (CS144)	https://github.com/mininet/mininet/wiki/Bufferbloat
MAC Overflow Attack (demo)	Stanford (CS144)	https://github.com/mininet/mininet/wiki/Mac-address-table-overflow-attack
Simple **OpenFlow hub and firewall** (and other course assignments)	Georgia Tech, Coursera (MOOC)	
… for Research	organization(s)	link
ElasticTree	Stanford, Berkeley	http://dl.acm.org/citation.cfm?id=1855728
Pyretic	Princeton, Cornell	http://frenetic-lang.org/pyretic/
NetSight	Stanford	
***Reproducible* Research Experiments**	Stanford (and hopefully everywhere!)	http://reproducingnetworkresearch.wordpress.com
… in Industry	organization(s)	link
Floodlight	Big Switch	http://www.projectfloodlight.org/floodlight/
HP Controller Platform (and SDK)	HP	http://h17007.www1.hp.com/us/en/networking/solutions/technology/sdn/
???	Insieme/Cisco	http://www.insiemenetworks.com

Mininet usage and adoption

Students: use it at many universities (e.g. Stanford CS144, CS244), MOOCs

Researchers: have used it for many SDN projects (e.g. Beacon, Netsight, ONOS, …)

Teachers: use it for labs, demos and course projects – teaching.mininet.org

Companies: used by Big Switch, HP, Insieme, others (liberal BSD license so we don't know)

IT services: used at Stanford for experimenting with SDN applications

Mininet usage: indirect numbers

Mininet.org

~133,000 visits since inception

October 2013: 16,000 visits

Pre-installed Virtual Machine Downloads

Mininet 2.0 – 217,134 (12/1/12 – 9/20/13)

Mininet 2.1 – 12,714 (9/20/13 – 11/3/13)

Github: 142 Forks (private copies of the source hosted on Github)

mininet-discuss Mailing List: ~180-250 messages/month, ~975 members

CiteSeerX: 31+ citations (and growing!)

Experiences running an Open Source software project

+ Increases impact, benefit, and visibility (for the project at least)

+/−: code (and other) contributions

− A lot of work to go from research prototype to production (documentation, web site, tests, examples, build system, launchpad…)

− Large support overhead (mailing list is a self-filtering system)

− More public criticism (some inaccurate)

− Work appropriation (“HP Mininet”)

Mininet Enhancements/Futures

Cluster Edition/Distributed Mininet: Tunnels are easy! Work at ARCCN, U. Paderborn, ON.Lab enables large networks (ARCCN: 30,000 nodes?!)

Time Dilation: Work at Stanford (Vimal J., Antonin Bas) decouples virtual time from real time, allowing "faster" simulations

GSoC 2013 projects: "Clone" a physical topology on Mininet; wireless link modeling integrating ns-3 model

Some related work

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

VM-based emulators: DieCast

UML-based emulators: NetKit

Simulators: ns-3, OPNET

Testbeds: Emulab, GENI, PlanetLab, ORBIT