# HANDS-ON SDN

Block Course – Winter 2016/17

David Koll

# Where we are now

**You have now learned about:**

•SDN basic principles

- Basic concepts (CP/DP separation etc.)
- De-facto standard interfaces (OpenFlow)
- Controllers (NOX, POX, …)
- Virtualization (FlowVisor)

# Where we want to go

**You have now learned about:**

•SDN basic principles

- Basic concepts (CP/DP separation etc.)
- De-facto standard interfaces (OpenFlow)
- Controllers (NOX, POX, …)
- Virtualization (FlowVisor)

• **Put the stuff learned into practice:**

- Implement OpenFlow?
- Implement controllers?
- Implement FlowVisor?

- Rather: *learn how to use and program them!*
  - Hands-on work on state-of-the-art tools

# How can we get there?

- Luckily, implementations are available.
  - Switches implementing OF
  - Controllers implementing OF

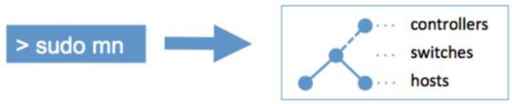
- So, how do we run them?
  - We don‘t have a hardware testbed at hand
  - We don‘t have access to a production network
  - We may want to test different things on different network topologies
  - Simulation?

# Emulation of Networks

- Network emulation means to run unmodified code interactively on virtual hardware
- Huge benefit:
  - Can actually port our applications seamlessly to hardware
- Challenges:
  - Scalability: need to model hosts, switches, links, controllers, …
  - Ease-of-Use: easily allow to create different topologies with varying parameters
  - Accuracy: results have to match results obtained from running same experiment on hardware

# Enter Mininet

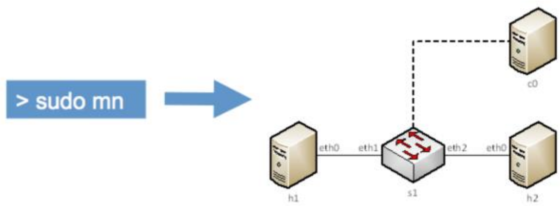
"Mininet creates a **realistic virtual network**, running **real kernel, switch and application code**, on a single machine (VM, cloud or native), in seconds, with a single command"[1]



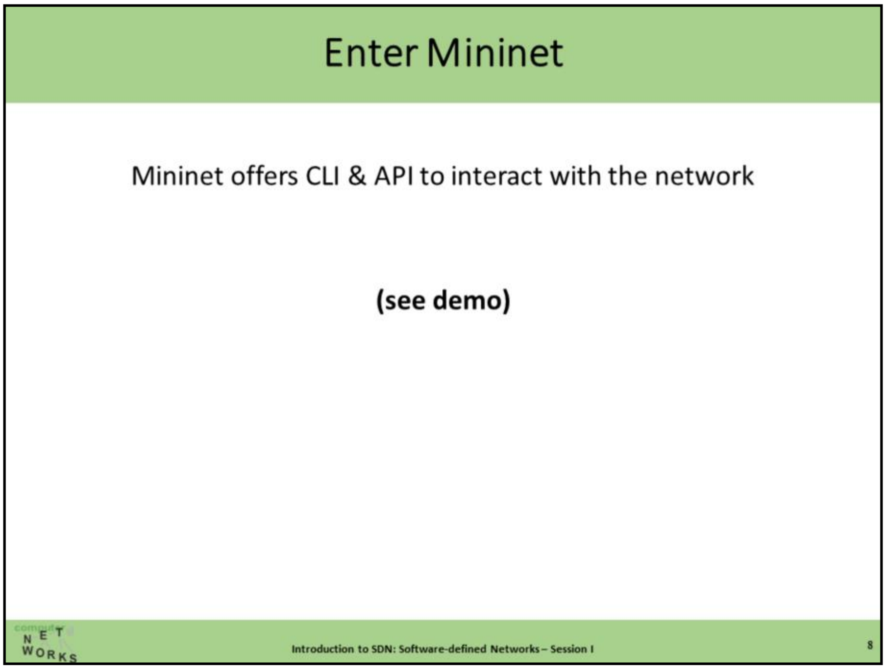

[1] mininet.org

# Enter Mininet

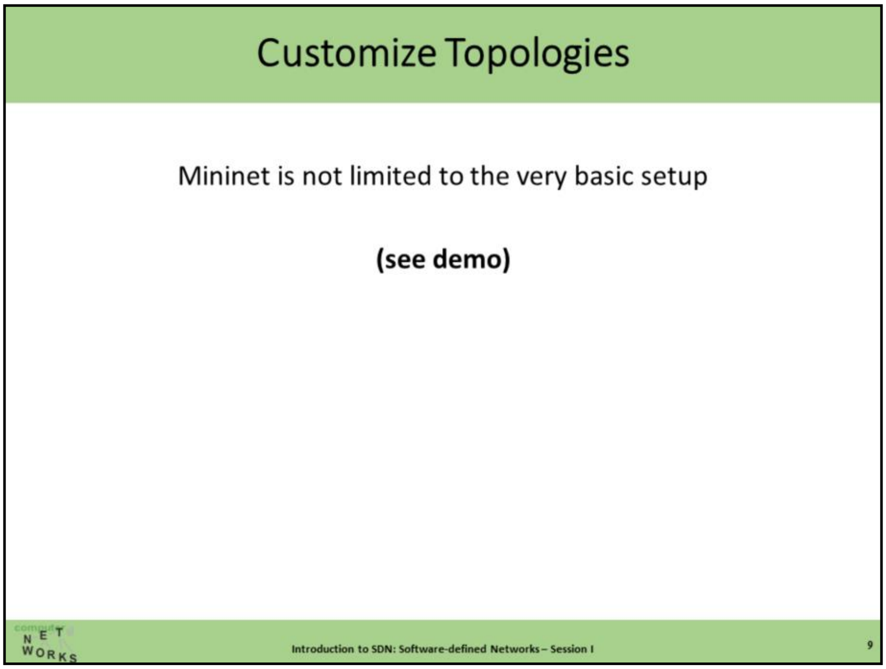
"Mininet creates a **realistic virtual network**, running **real kernel, switch and application code**, on a single machine (VM, cloud or native), in seconds, with a single command"[1]




[1] mininet.org




Sudo mn -> Pingall -> H1 ping h2 -> Iperf -> Nodes -> Xterm h1 -> Ifconfig –a
Complicated MAC addresses -> Sudo mn –c -> Sudo mn –mac ->xterm h1 -> ifconfig -a




Sudo mn - -topo linear,4 -> dump
Virtual network -> different namespaces
Everything else: not virtualized -> h1 ps –a == s1 ps –a

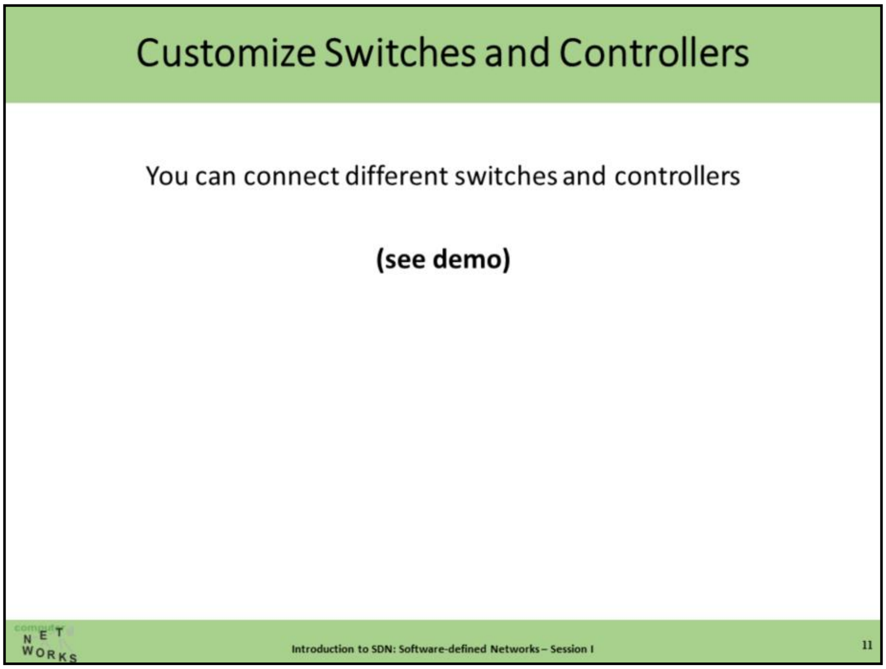
Can also change link capacities/delays :
iperf
Sudo mn –c
sudo mn –link tc,bw=1
iperf

# Customize Topologies

```python
from mininet.topo import Topo
class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
    "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```



Sudo mn –custom custom_topo.py –topo mytopo –test pingall




Sudo mn –switch ovsk –controller remote
New terminal -> cd pox -> ./pox.py forwarding.hub

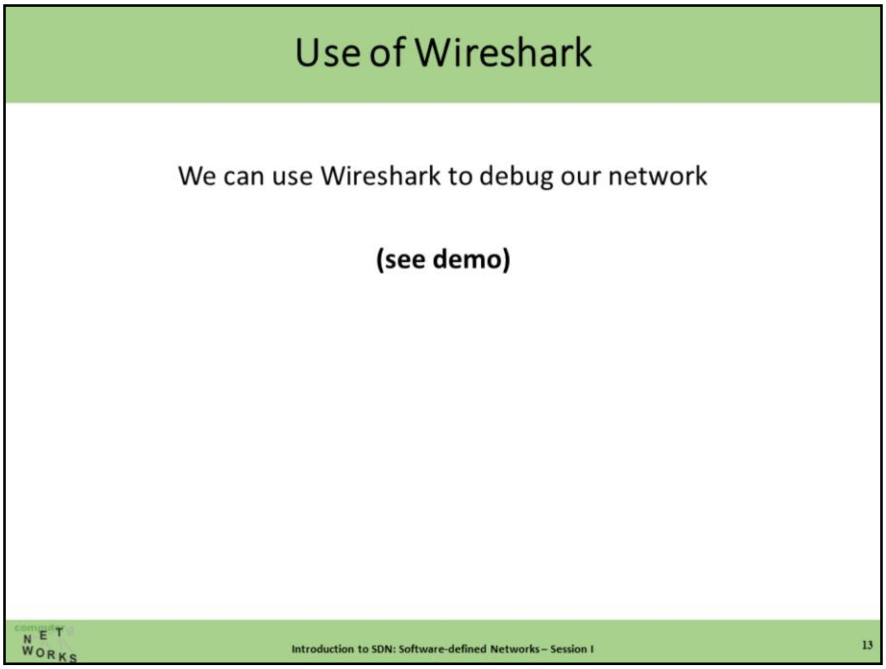
# Bring Links Up/Down

Change the topology at runtime

**(see demo)**



Pingall -> link h1 s1 down -> pingall -> link h1 s1 up -> pingall




Exit -> sudo mn –c -> sudo mn –controller remote
New terminal -> sudo wireshark & ->  filter of -> select lo
New terminal -> ./pox.py forwarding.hub
See of_hello of_features_request etc in wireshark

# Limitations?

Limited by single system resources
Limited to Linux kernel (e.g., portability to Windows?)
Limited to real-time

# NOTE:

Afternoon lecture today 1 hour later! Starts at 3.15pm!

# Exercise!

Time for Exercises 5a and 5b

http://windysdn.blogspot.de/2013/09/start-and-stop-pox-controller.html

# Custom Topologies with Mininet Python API

Mininet offers some topologies!

Eg: single switch, linear, tree

What if you want to replicate your very own production network?

Create a custom topology!

# Low-level API: Nodes and Links

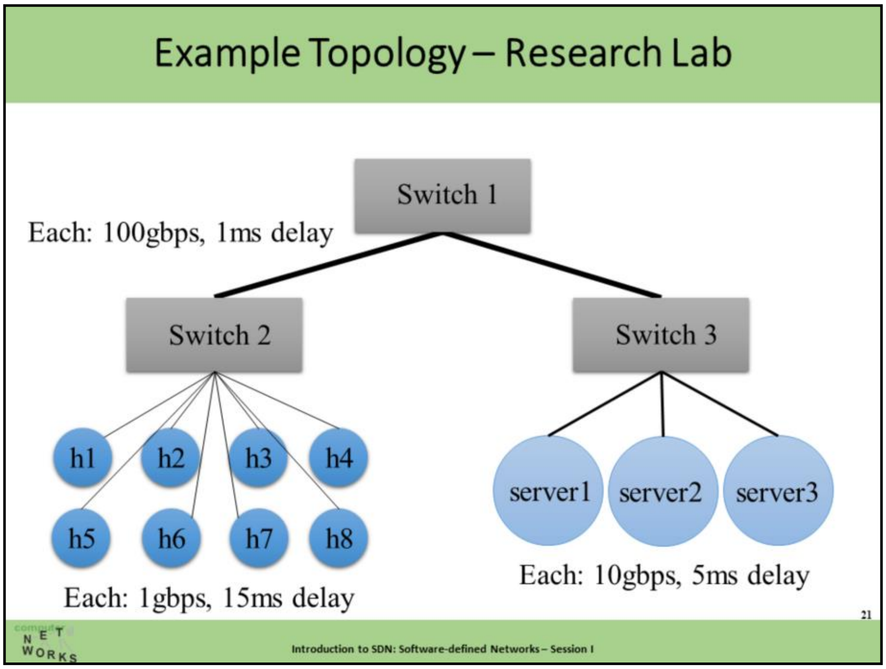
```
h1 = Host( 'h1' )
h2 = Host( 'h2' )
s1 = OVSSwitch( 's1', inNamespace=False )
c0 = Controller( 'c0', inNamespace=False )
Link( h1, s1 )
Link( h2, s1 )
h1.setIP( '10.1/8' )
h2.setIP( '10.2/8' )
c0.start()
s1.start( [ c0 ] )
print h1.cmd( 'ping -c1', h2.IP() )
s1.stop()
c0.stop()
```

# 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):
          Topo.__init__(self)
          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 = {'topo' : (lambda: SingleSwitchTopo())}
```


Example Topology – Research Lab
Switch 1
Each: 100gbps, 1ms delay
Switch 2
Switch 3
h1
h2
h3
h4
h5
h6
h7
h8
server1
server2
server3
Each: 10gbps, 5ms delay
Each: 1gbps, 15ms delay
21
Introduction to SDN: Software-defined Networks – Session I

# Example Topology – Research Lab

```
#!/usr/bin/python
from mininet.topo import Topo

class ResearchLab ( Topo ) :
  """"Research Lab Topology"""
  def __init__ ( self ) :

    Topo.__init__(self)
    testbedhosts = [ self.addHost ( 'h%d' % i ) for i in range ( 1, 9) ]
    simservers = [self.addHost ( 'sim%d' % i ) for i in range ( 1, 4) ]
    s1 = self.addSwitch ( 's1' ) # TOR switch
    s2 = self.addSwitch ( 's2' ) # Testbed switch
    s3 = self.addSwitch ( 's3' ) # Server switch

    for h in testbedhosts:
      self.addLink ( h, s2 , bw=1, delay='15ms')

    for srv in simservers:
      self.addLink ( srv,s3, bw=10, delay='1ms')

    self.addLink (s2, s1, bw=100)
    self.addLink (s3, s1, bw=100)

topos = {'rlab' : (lambda: ResearchLab ()) }
```

```
sudo mn
--custom rlab.py
--topo rlab
--link=tc
```

# The POX Controller

- Invoke with: ./pox.py [options] <component>

- <options> can be:
    - --verbose : display debugging info
    - --no-openflow: do not automatically listen for OpenFlow connections

- <components> are the real meat!
  - There are some basic components we will use for this class
  - Intention: developers will build their own components

# The POX Controller - Components

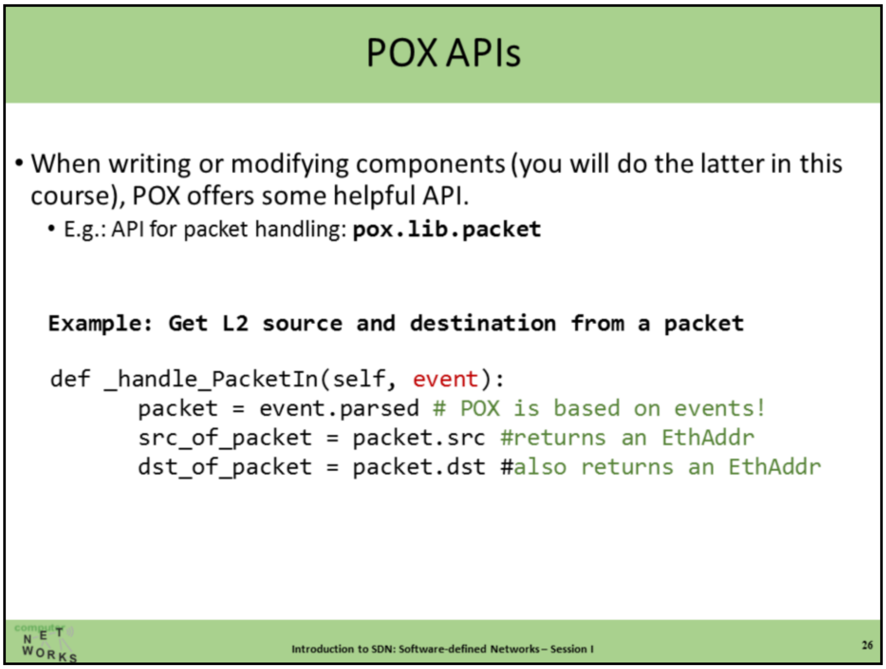
- Some stock components:
  - py
  - forwarding.hub
  - forwarding.l2_learning
  - forwarding.l2_pairs
  - forwarding.....

./pox.py forwarding.l2_learning ?

  - openflow.webservice
    - Creates a webinterface to interact with OpenFlow
  - openflow.of_01
    - Communicates with OpenFlow 1.0 switches

# The POX Controller - Components

- Developing your own components:
  - https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-DevelopingyourownComponents
- In general: POX wiki a good place to look for help
  - https://openflow.stanford.edu/display/ONL/POX+Wiki


POX APIs
• When writing or modifying components (you will do the latter in this course), POX offers some helpful API.
• E.g.: API for packet handling: pox.lib.packet
Example: Get L2 source and destination from a packet
def _handle_PacketIn(self, event):
packet = event.parsed # POX is based on events!
src_of_packet = packet.src #returns an EthAddr
dst_of_packet = packet.dst #also returns an EthAddr
Introduction to SDN: Software-defined Networks – Session I
26

# POX APIs

- When writing or modifying components (you will do the latter in this course), POX offers some helpful API.
  - E.g.: API for packet handling: **pox.lib.packet**

**Example: Get source IP from a packet**

```
def _handle_PacketIn(self, event):
      "check if packet is an IP packet"
      packet = event.parsed
      ip = packet.find('ipv4') #check if packet is IP
      if ip is None: #packet is not IP
            return
      print "Source IP: ", ip.srcip
```

# POX and Openflow

- Up front: Best to read POX wiki:
  - https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-OpenFlowinPOX
- Usually, switches connect to POX automatically via OpenFlow
  - Exception: `no-openflow` option (see previous slides)
- So – how do we communicate with them?

# Coding in POX – Connection Elements

- Upon connecting to POX, a switch is associated with a `Connection` object
- Use that object's `send()` method to send messages to the switch
- `Connection` object will raise events on the corresponding switch
  - Create **event handlers** for events you are interested in

# In Practice

- Launch our component.
- Add one event listener for PacketIn

```
from pox.core import core
import pox.openflow.libopenflow_01 as of

log = core.getLogger()

def launch ():
    "Starts the Component"
    core.openflow.addListenerByName("PacketIn",
                       _handle_packetin)

    log.info("Switch running.")
```

## In Practice

- Write packet handler (here: flood packet)

```
def _handle_packetin (event):
    “Handle PacketIn“
    packet = event.parsed
    send_packet(event, of.OFPP_ALL) #broadcast

    log.debug("Broadcasting %s.%i -> %s.%i" %
                 (packet.src, event.ofp.in_port,
                 packet.dst, of.OFPP_ALL))
```

# In Practice

- Write send_packet method (simplified)

```
def send_packet (event, dst_port):
     "Instructs switch to send packet via dst_port"
      msg = of.ofp_packet_out(in_port=event.ofp.in_port)
      msg.data = event.ofp.data
      msg.actions.append(of.ofp_action_output(port = dst_port))

      event.connection.send(msg)
```

## In Practice

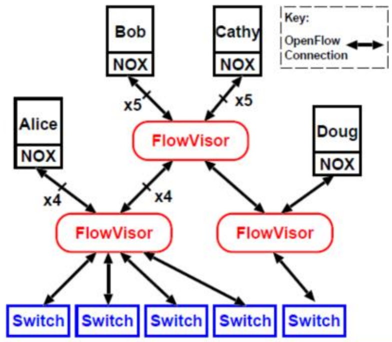
- Code on previous slides implemented a hub behaviour
- Exercise: modify hub behaviour to learning switch behaviour

# Exercise!

Time for Exercise 6

# FlowVisor

- Exercise 5: You have already installed FlowVisor
- Recall: FlowVisor is an extra layer between controllers and switches



# FlowVisor

- Basic procedure:
  - Create and start your network topology with Mininet
  - Connect Flowvisor to switches on standard port
  - Slice network with Flowvisor
  - Connect Controllers to Flowvisor slices

# FlowVisor

- Basic procedure:
  - Create and start your network topology with Mininet
  - Connect Flowvisor to switches on standard port
  - Slice network with Flowvisor
  - Connect Controllers to Flowvisor slices

# Connecting FlowVisor

- FlowVisor operates outside of Mininet!

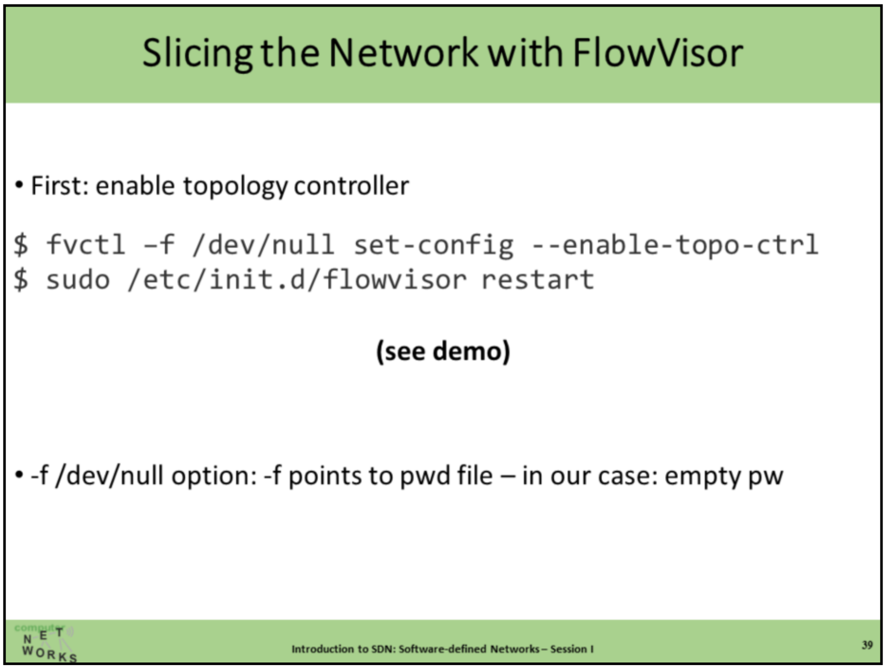
```
$ sudo /etc/init.d/flowvisor start
```

**(see demo)**

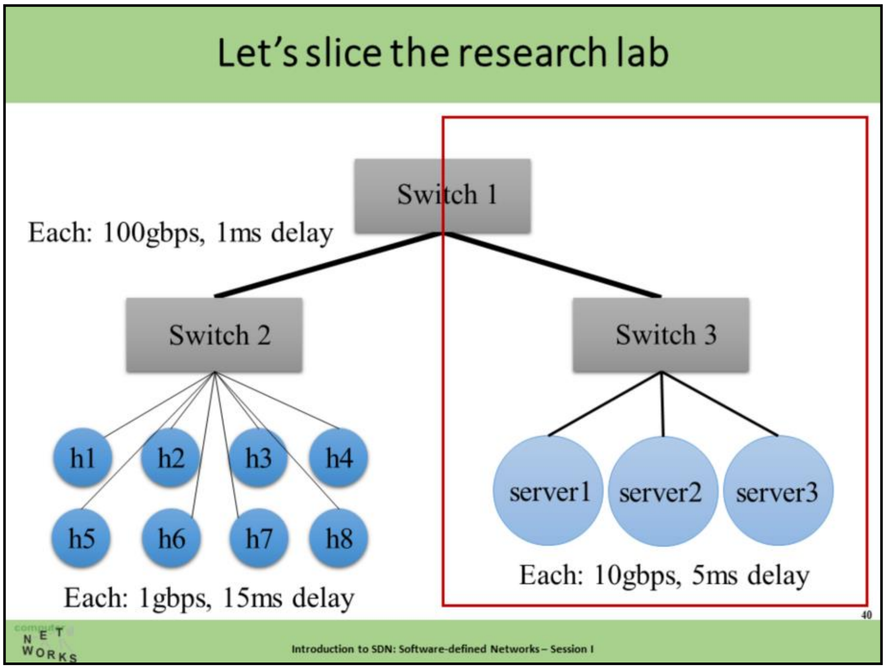
- Afterwards: use flowvisor control (command: `fvctl`) to slice

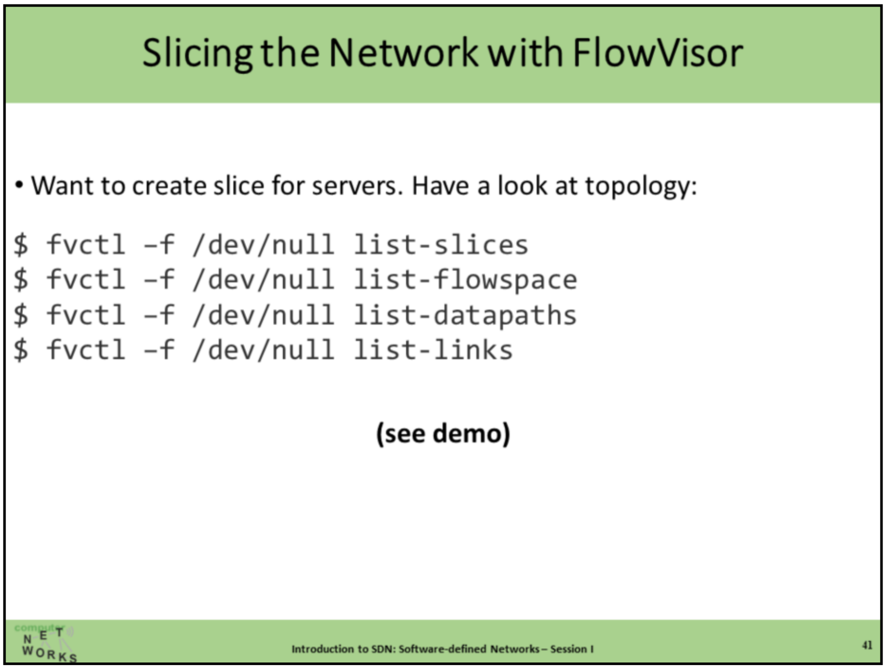


$ sudo /etc/init.d/flowvisor start

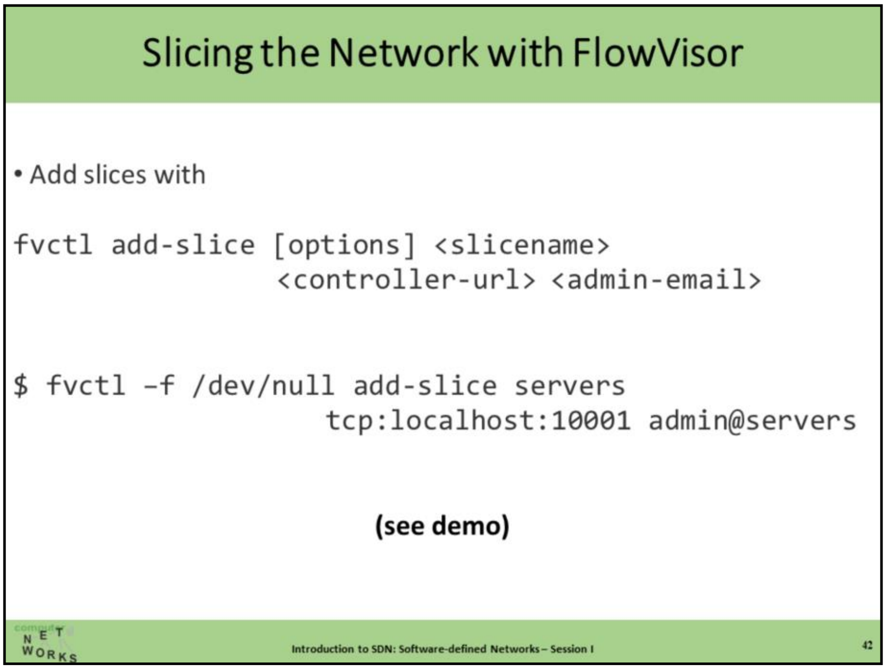



```
$ fvctl –f /dev/null set-config --enable-topo-ctrl
$ sudo /etc/init.d/flowvisor restart
```

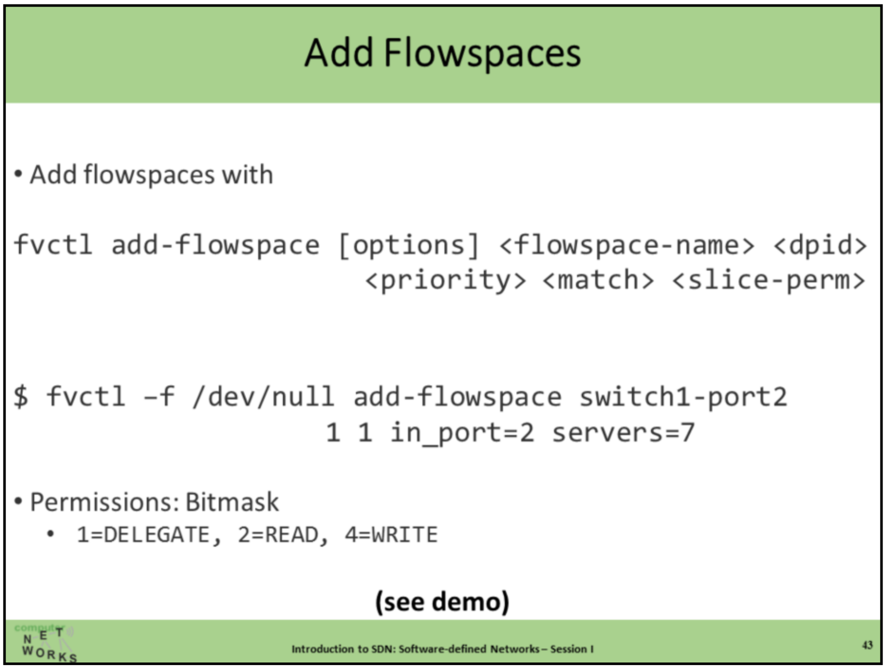

Let's slice the research lab
Switch 1
Each: 100gbps, 1ms delay
Switch 2
Switch 3
h1
h2
h3
h4
h5
h6
h7
h8
server1
server2
server3
Each: 10gbps, 5ms delay
Each: 1gbps, 15ms delay
40
Introduction to SDN: Software-defined Networks – Session I



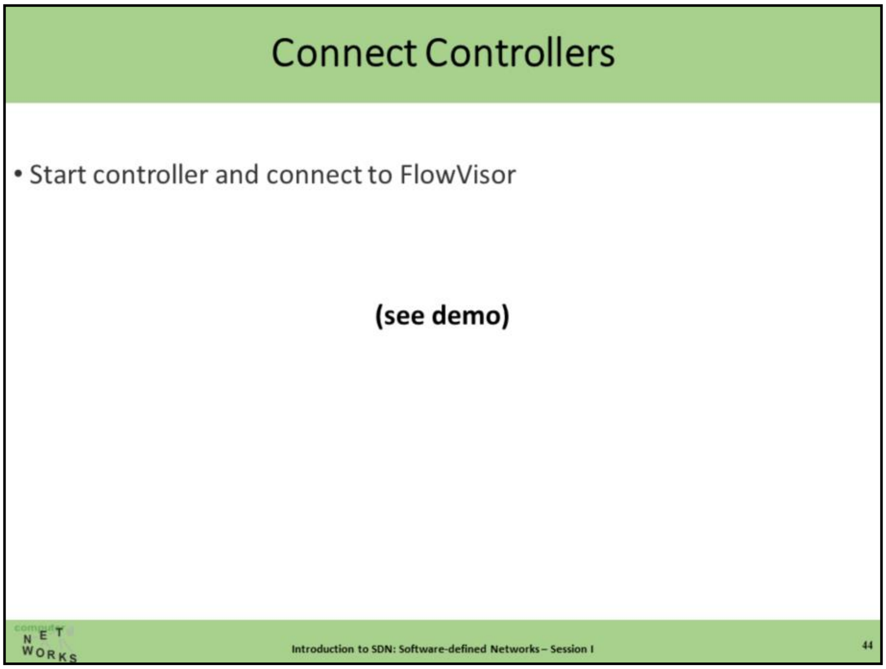

```
$ fvctl -f /dev/null list-slices
```




```
$ fvctl –f /dev/null add-slice servers tcp:localhost:10001
admin@servers
```

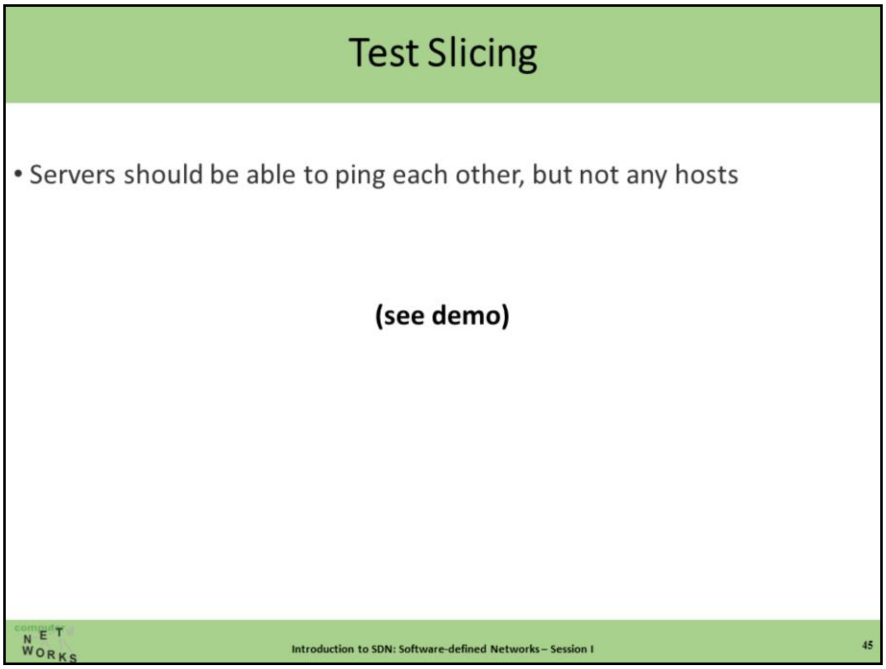



```
$ fvctl -f /dev/null add-flowspace switch1-port2 1 1 in_port=2
servers=7
$ fvctl -f /dev/null add-flowspace switch3-port1 3 1 in_port=1
servers=7
$ fvctl -f /dev/null add-flowspace switch3-port2 3 1 in_port=2
servers=7
$ fvctl -f /dev/null add-flowspace switch3-port3 3 1 in_port=3
servers=7
$ fvctl -f /dev/null add-flowspace switch3-port4 3 1 in_port=4
servers=7

$ fvctl -f /dev/null list-flowspace
```




```
NEW TERMINAL
$ ./pox.py openflow.of_01 --port 10001 forwarding.l2_pairs
```

Comment: need to load openflow with custom parameters here (instead of default that would be loaded when just loading forwarding.l2_pairs)




```
Mininet console> sim1 ping –c1 sim2
Mininet console> sim1 ping –c1 h1
```

# Exercise!

Time for Exercise 7