# HANDS-ON SDN

***Introduction to Software-defined Networking***
Block Course – 13-17 March 2015

David Koll

# Where we are now

**You have now learned about:**

- Python programming
- 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:**

- Python programming
- 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?



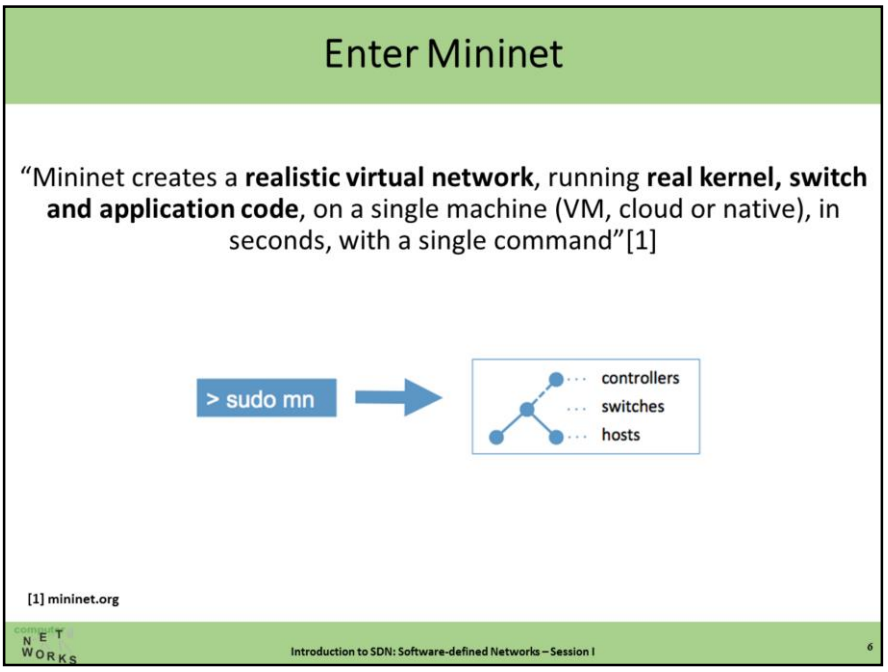
Network ossification: We see this today, main reason for SDN

## 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



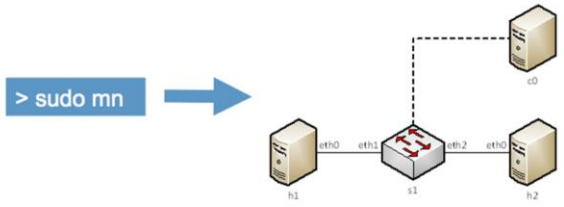
Also: innovation in applications far outpacing standardization;




Network ossification: We see this today, main reason for SDN

## Enter Mininet

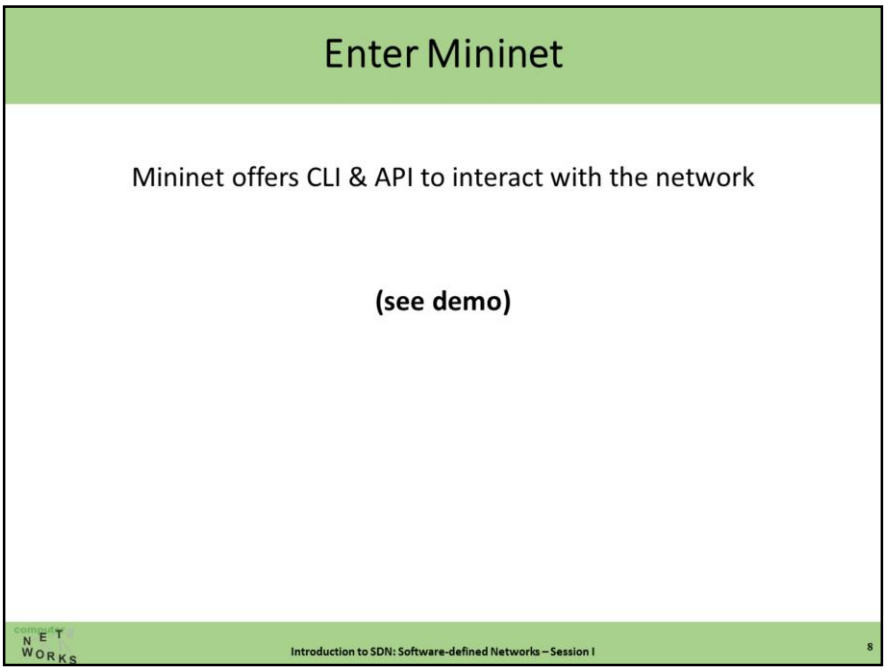
"Mininet creates a **realistic virtual network**, running **real kernel, switch and application code**, on a single machine (VM, cloud or native), in seconds, with a single command"[1]



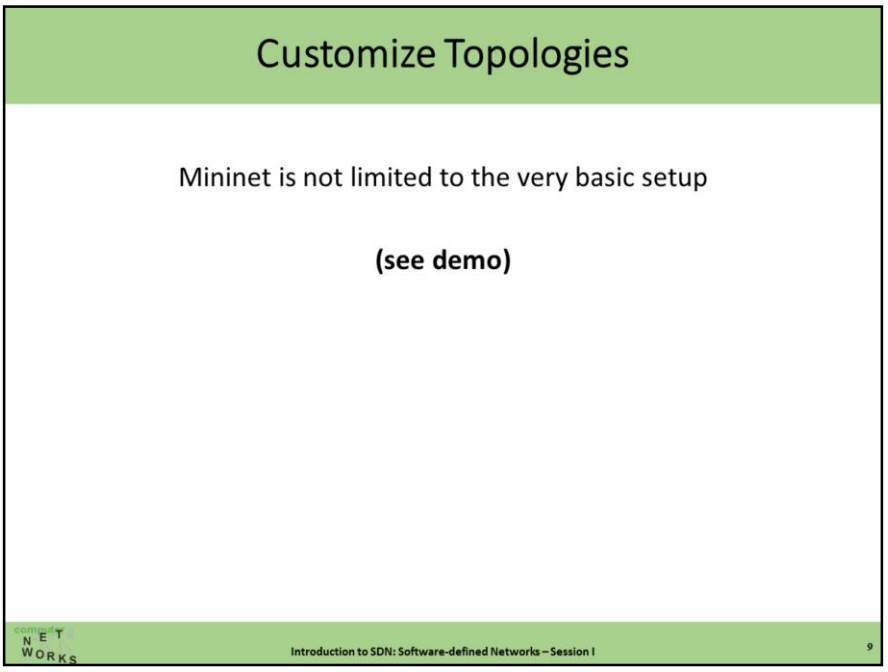

[1] mininet.org



Network ossification: We see this today, main reason for SDN




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

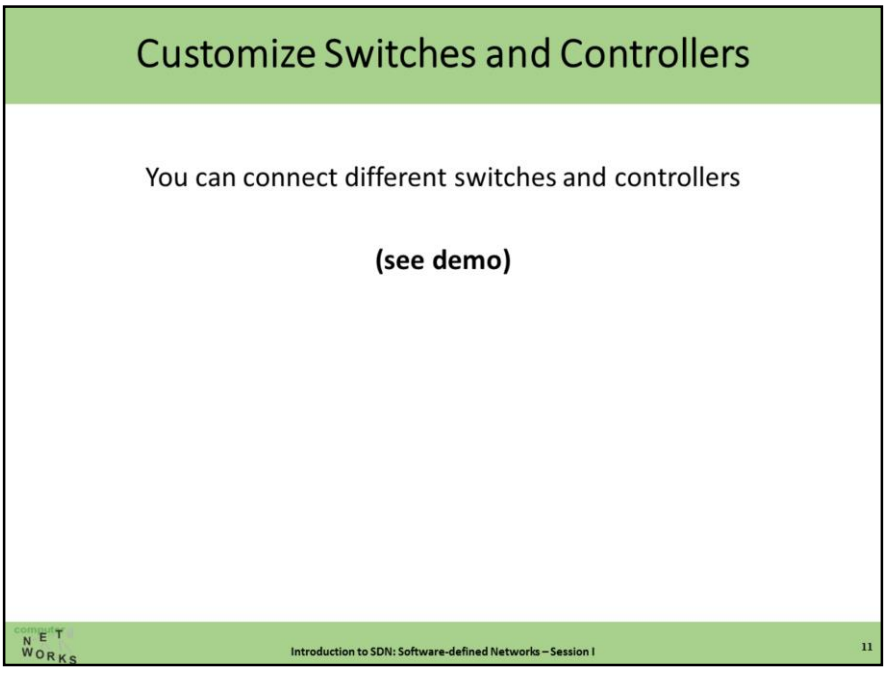
Can also change link capacities/delays :
iperf
Sudo mn –c
sudo mn –link tc,bw=1
iperf

# Customize Topologies

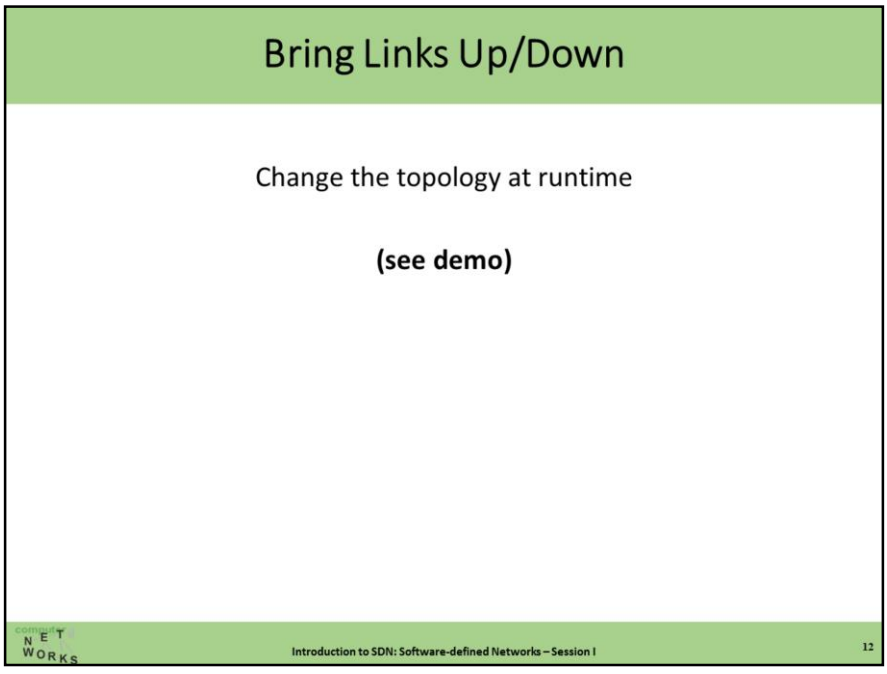
```
from mininet.topo import Topo
class MyTopo( Topo ):
    "Simple topology example."

    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() ) }
```



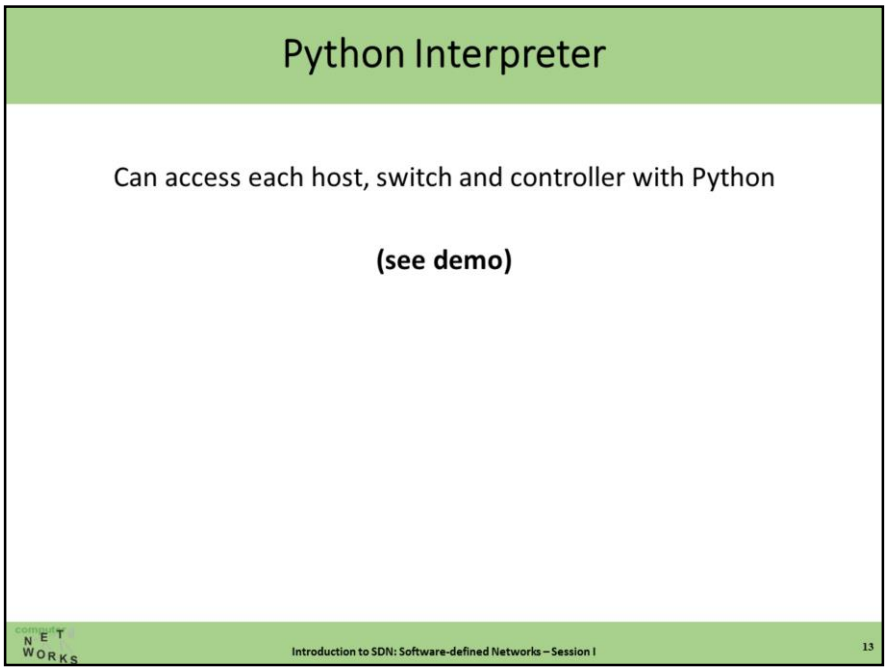
Sudo mn –custom custom_topo.py –topo mytopo –test pingall



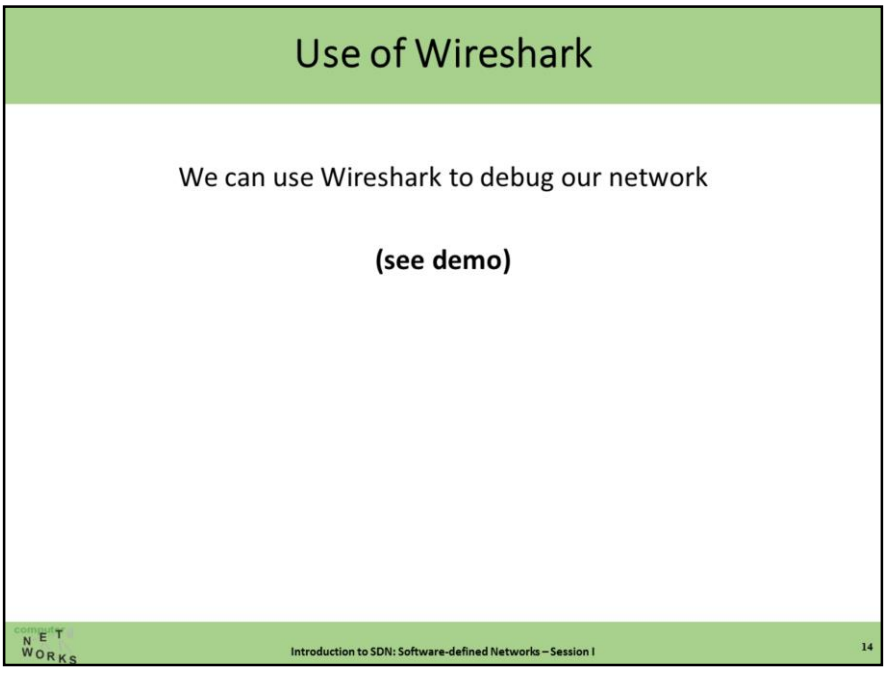

Sudo mn –switch ovsk –controller remote
New terminal -> cd pox -> ./pox.py forwarding.hub



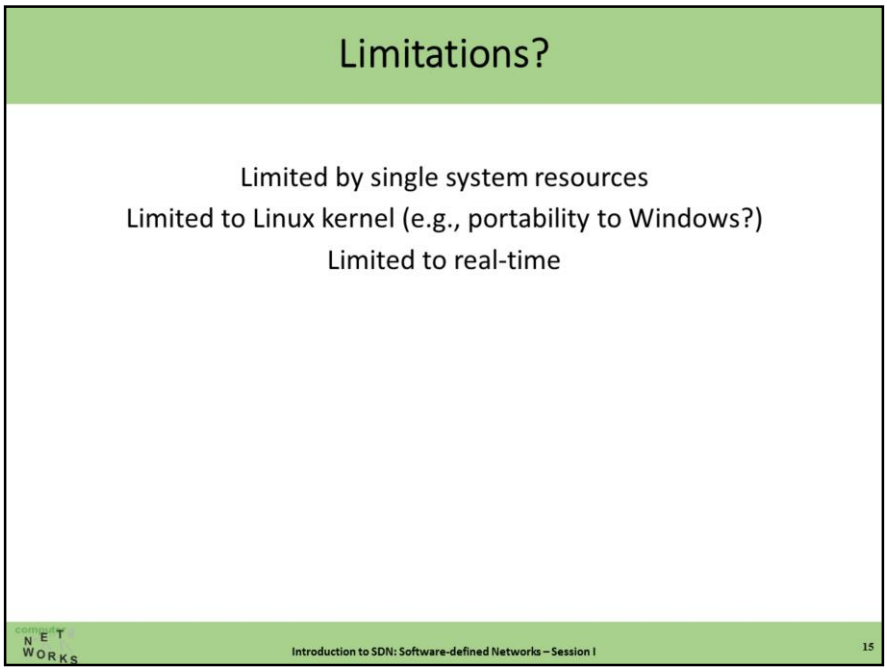

Pingall -> link h1 s1 down -> pingall -> link h1 s1 up -> pingall



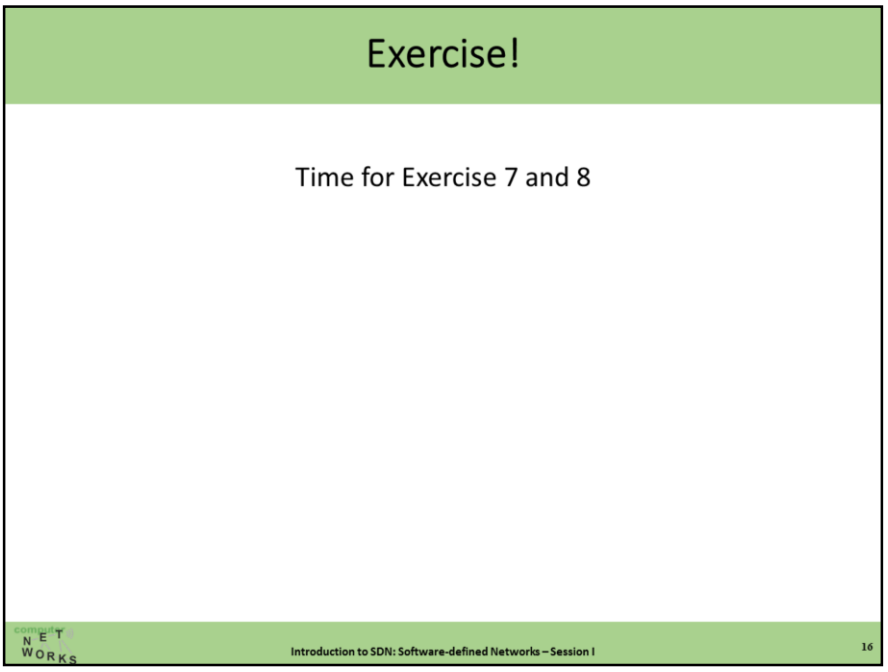

Py „hello“ + „world“
Py h1.IP()




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




Network ossification: We see this today, main reason for SDN




Network ossification: We see this today, main reason for SDN