



# Mininet & POX Controller Tutorial

**Instructor:** Dr. Irfan Ali

**Presenter:** Müge Erel-Özçevik

Department of Computer Engineering,  
Istanbul Technical University,  
Turkey  
Email:erelmu@itu.edu.tr

18.04.2017



# Outline

Introduction

Mininet Installation

Mininet Command Line Tool

Remote Controller

Custom Controller and Topology



# Outline

## Introduction

Platforms for Network/Systems

Mininet Architecture

Mininet Installation

Mininet Command Line Tool

Remote Controller

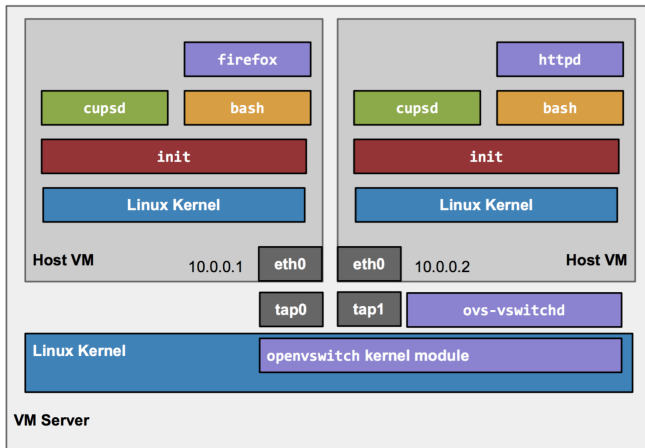
Custom Controller and Topology

# Platforms for Network/Systems

Platform	Advantages	Disadvantages
<b>Hardware Testbed</b>	fast accurate: "ground truth"	expensive shared resource? hard to reconfigure hard to change hard to download
<b>Simulator</b>	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
<b>Emulator</b>	inexpensive, flexible real code reasonably accurate easy to download fast/interactive usage	slower than hardware experiments may not fit possible inaccuracy from multiplexing

Figure: Platforms for Network/Systems

## Mininet Architecture



**Figure:** An example of Mininet Architecture for two hosts



# Outline

Introduction

**Mininet Installation**

Mininet Command Line Tool

Remote Controller

Custom Controller and Topology



## Mininet Installation

To get started install these softwares on your host machine:

- Install Vagrant, it is a wrapper around virtualization softwares like VirtualBox, VMWare etc.:  
<http://www.vagrantup.com/downloads>
- Install VirtualBox, this would be your VM provider:  
<https://www.virtualbox.org/wiki/Downloads>
- Install Git, it is a distributed version control system:  
<https://git-scm.com/downloads>
- Install X Server and SSH capable terminal
  - Linux comes pre-installed with X server and Gnome terminal + SSH (builtin)
  - »sudo apt-get install ubuntu-desktop



## Mininet Installation

To get started install these softwares on your host machine:

- Install Vagrant, it is a wrapper around virtualization softwares like VirtualBox, VMWare etc.:  
<http://www.vagrantup.com/downloads>
- Install VirtualBox, this would be your VM provider:  
<https://www.virtualbox.org/wiki/Downloads>
- Install Git, it is a distributed version control system:  
<https://git-scm.com/downloads>
- Install X Server and SSH capable terminal
  - Linux comes pre-installed with X server and Gnome terminal + SSH (builtin)
  - »sudo apt-get install ubuntu-desktop



## Mininet Installation

To get started install these softwares on your host machine:

- Install Vagrant, it is a wrapper around virtualization softwares like VirtualBox, VMWare etc.:  
<http://www.vagrantup.com/downloads>
- Install VirtualBox, this would be your VM provider:  
<https://www.virtualbox.org/wiki/Downloads>
- Install Git, it is a distributed version control system:  
<https://git-scm.com/downloads>
- Install X Server and SSH capable terminal
  - Linux comes pre-installed with X server and Gnome terminal + SSH (builtin)
  - »sudo apt-get install ubuntu-desktop



## Mininet Installation

To get started install these softwares on your host machine:

- Install Vagrant, it is a wrapper around virtualization softwares like VirtualBox, VMWare etc.:  
<http://www.vagrantup.com/downloads>
- Install VirtualBox, this would be your VM provider:  
<https://www.virtualbox.org/wiki/Downloads>
- Install Git, it is a distributed version control system:  
<https://git-scm.com/downloads>
- Install X Server and SSH capable terminal
  - Linux comes pre-installed with X server and Gnome terminal + SSH (built-in)
  - » `sudo apt-get install ubuntu-desktop`

## Mininet Installation

To get started install these softwares on your host machine:

- Install Vagrant, it is a wrapper around virtualization softwares like VirtualBox, VMWare etc.:  
<http://www.vagrantup.com/downloads>
- Install VirtualBox, this would be your VM provider:  
<https://www.virtualbox.org/wiki/Downloads>
- Install Git, it is a distributed version control system:  
<https://git-scm.com/downloads>
- Install X Server and SSH capable terminal
  - Linux comes pre-installed with X server and Gnome terminal + SSH (builtin)
  - » `sudo apt-get install ubuntu-desktop`



## Mininet Installation

To get started install these softwares on your host machine:

- Install Vagrant, it is a wrapper around virtualization softwares like VirtualBox, VMWare etc.:  
<http://www.vagrantup.com/downloads>
- Install VirtualBox, this would be your VM provider:  
<https://www.virtualbox.org/wiki/Downloads>
- Install Git, it is a distributed version control system:  
<https://git-scm.com/downloads>
- Install X Server and SSH capable terminal
  - Linux comes pre-installed with X server and Gnome terminal + SSH (built in)
  - » `sudo apt-get install ubuntu-desktop`



# Outline

Introduction

Mininet Installation

**Mininet Command Line Tool**

Help

Linear topology

Flow table status

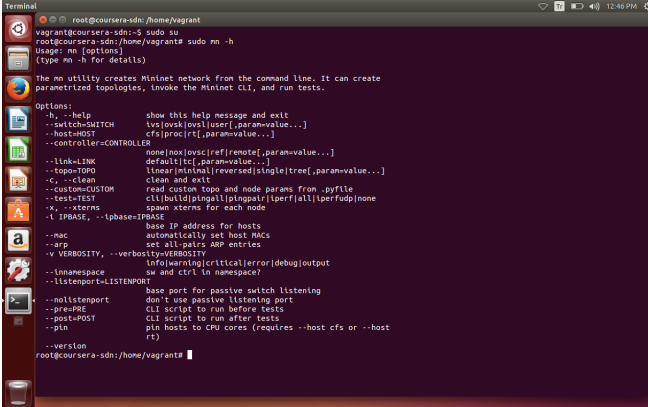
Port status

Remote Controller

Custom Controller and Topology

# Help

»sudo mn -h



```
Terminal
root@coursara-sdn: /home/vagrant
vagrant@coursara-sdn:~$ sudo su
root@coursara-sdn:~# 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.

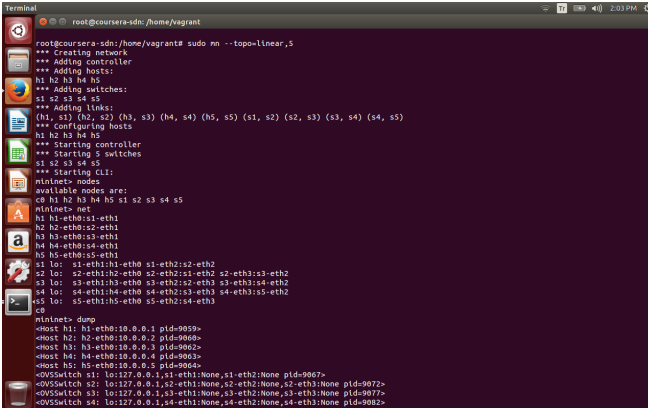
Options:
  -h, --help                show this help message and exit
  --switch=SWITCH           tvs|ovsk|ovsl|user[,param=value...]
  --host=HOST               cfs|proc|rt[,param=value...]
  --controller=CONTROLLER  none|nox|ovsc|ref|remote[,param=value...]
  --link=LINK               default|tc[,param=value...]
  --topo=TOPO               linear|minimal|reversed|single|tree[,param=value...]
  -c, --clean               clean and exit
  --custom=CUSTOM           read custom topo and node params from .pyfile
  --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 info|warning|critical|error|debug|output
  --lnamespace              sw and ctrl in namespace?
  --listenport=LISTENPORT   base port for passive switch listening
  --nolistenport            don't use passive listening port
  --pre=PRE                 CLI script to run before tests
  --post=POST               CLI script to run after tests
  --pin                     pin hosts to CPU cores (requires --host cfs or --host
                           rt)
  --version                 rt)

root@coursara-sdn: /home/vagrant#
```

Figure: Calling Mininet Help

# Linear topology

(»sudo mn - - topo=linear,5) (»nodes) (»net) (»dump)



```

root@course-sdn:/home/vagrant
root@course-sdn:/home/vagrant# sudo mn --topo=linear,5
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s5) (s1, s2) (s2, s3) (s3, s4) (s4, s5)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
*** Starting 5 switches
s1 s2 s3 s4 s5
*** Starting CLI:
mininet> nodes
available nodes are:
c0 h1 h2 h3 h4 h5 s1 s2 s3 s4 s5
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s3-eth1
h4 h4-eth0:s4-eth1
h5 h5-eth0:s5-eth1
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth2
s2 lo: s2-eth1:h2-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo: s3-eth1:h3-eth0 s3-eth2:s2-eth3 s3-eth3:s4-eth2
s4 lo: s4-eth1:h4-eth0 s4-eth2:s3-eth3 s4-eth3:s5-eth2
s5 lo: s5-eth1:h5-eth0 s5-eth2:s4-eth3
c0
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=9059>
<Host h2: h2-eth0:10.0.0.2 pid=9060>
<Host h3: h3-eth0:10.0.0.3 pid=9062>
<Host h4: h4-eth0:10.0.0.4 pid=9063>
<Host h5: h5-eth0:10.0.0.5 pid=9064>
<OVSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=9067>
<OVSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=9072>
<OVSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=9077>
<OVSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=9082>
  
```

Figure: Examining Linear Topology with 5 OpenFlow switches

```
>>dpctl dump-flows
```

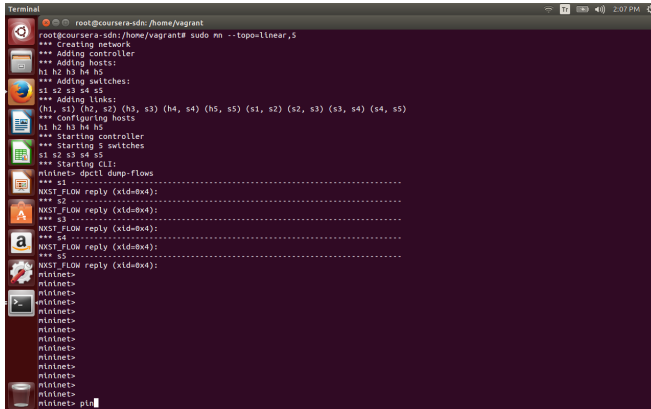
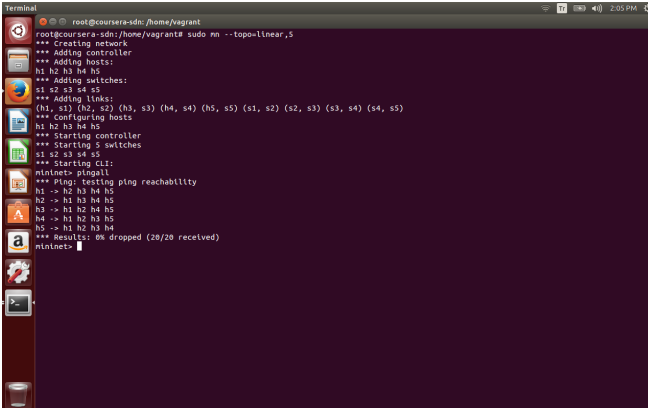


Figure: Empty flow tables



# Pingall

»pingall



```

Terminal
root@coursiera-sdn: /home/vagrant
root@coursiera-sdn:/home/vagrant# sudo mn --topo=linear,s
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (h4, s4) (h5, s5) (s1, s2) (s2, s3) (s3, s4) (s4, s5)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
*** Starting 5 switches
s1 s2 s3 s4 s5
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet>
  
```

Figure: Examining Linear Topology with 5 OpenFlow switches

# Flow tables after iperf

»iperf h1 h2

```

Terminal
root@coursera-sdn: /home/vagrant

mininet>
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['7.97 Gbits/sec', '7.99 Gbits/sec']
mininet> dpctl dump-flows
*** s1 *****
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=9.655s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=9, priority=65535,arp,in_port=2,vlan_tci=0x0
000,d_l_src=12:7a:4c:c5:6f:76,d_l_dst=0e:f2:6e:bb:28,arp_spac=10.0.0.2,arp_tpa=10.0.0.1,arp_op=2 actions=output:1
cookie=0x0, duration=9.616s, table=0, n_packets=1, n_bytes=66, idle_timeout=60, idle_age=9, priority=65535,tcp,in_port=1,vlan_tci=0x0
000,d_l_src=0e:f2:6e:bb:28,d_l_dst=12:7a:4c:c5:6f:76,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=54616,tp_dst=5001 actions=output
:2
cookie=0x0, duration=9.623s, table=0, n_packets=133548, n_bytes=5807113848, idle_timeout=60, idle_age=4, priority=65535,tcp,in_port=1
,vlan_tci=0x0000,d_l_src=0e:f2:6e:bb:28,d_l_dst=12:7a:4c:c5:6f:76,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=54617,tp_dst=5001 a
ctions=output:2
cookie=0x0, duration=9.653s, table=0, n_packets=3, n_bytes=206, idle_timeout=60, idle_age=9, priority=65535,tcp,in_port=2,vlan_tci=0x
0000,d_l_src=12:7a:4c:c5:6f:76,d_l_dst=0e:f2:6e:bb:28,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst=54616 actions=output
:1
cookie=0x0, duration=9.617s, table=0, n_packets=38302, n_bytes=2527940, idle_timeout=60, idle_age=4, priority=65535,tcp,in_port=2,vla
n_tci=0x0000,d_l_src=12:7a:4c:c5:6f:76,d_l_dst=0e:f2:6e:bb:28,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst=54617 actio
ns=output:1
*** s2 *****
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=9.668s, table=0, n_packets=1, n_bytes=42, idle_timeout=60, idle_age=9, priority=65535,arp,in_port=1,vlan_tci=0x0
000,d_l_src=12:7a:4c:c5:6f:76,d_l_dst=0e:f2:6e:bb:28,arp_spac=10.0.0.2,arp_tpa=10.0.0.1,arp_op=2 actions=output:2
cookie=0x0, duration=9.666s, table=0, n_packets=3, n_bytes=206, idle_timeout=60, idle_age=9, priority=65535,tcp,in_port=1,vlan_tci=0x
0000,d_l_src=12:7a:4c:c5:6f:76,d_l_dst=0e:f2:6e:bb:28,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst=54616 actions=output
:1
cookie=0x0, duration=9.636s, table=0, n_packets=38302, n_bytes=2527940, idle_timeout=60, idle_age=4, priority=65535,tcp,in_port=1,vla
n_tci=0x0000,d_l_src=12:7a:4c:c5:6f:76,d_l_dst=0e:f2:6e:bb:28,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=5001,tp_dst=54617 actio
ns=output:2
cookie=0x0, duration=9.634s, table=0, n_packets=133548, n_bytes=5807113848, idle_timeout=60, idle_age=4, priority=65535,tcp,in_port=2
,vlan_tci=0x0000,d_l_src=0e:f2:6e:bb:28,d_l_dst=12:7a:4c:c5:6f:76,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=54617,tp_dst=5001 a
ctions=output:1
cookie=0x0, duration=9.618s, table=0, n_packets=1, n_bytes=66, idle_timeout=60, idle_age=9, priority=65535,tcp,in_port=2,vlan_tci=0x0
000,d_l_src=0e:f2:6e:bb:28,d_l_dst=12:7a:4c:c5:6f:76,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=54616,tp_dst=5001 actions=output
:1
*** s3 *****
NXST_FLOW reply (xid=0x4):
*** s4 *****
NXST_FLOW reply (xid=0x4):

```

Figure: Flow tables are filled via TCP flow entries after iperf

# Port status

» dpctl dump-ports

```

Terminal
root@coursera-sdn: /home/vagrant

port 2: rx pkts=281, bytes=50090, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=92, bytes=15234, drop=0, errs=0, coll=0
*** s2 -----
OFFST_PORT reply (xid=0x2): 4 ports
port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=0, bytes=0, drop=261, errs=0, coll=0
port 1: rx pkts=27, bytes=2290, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=343, bytes=62414, drop=0, errs=0, coll=0
port 2: rx pkts=92, bytes=15234, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=281, bytes=50090, drop=0, errs=0, coll=0
port 3: rx pkts=219, bytes=38264, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=159, bytes=27019, drop=0, errs=0, coll=0
*** s3 -----
OFFST_PORT reply (xid=0x2): 4 ports
port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=0, bytes=0, drop=254, errs=0, coll=0
port 1: rx pkts=27, bytes=2290, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=336, bytes=61284, drop=1, errs=0, coll=0
port 2: rx pkts=159, bytes=27019, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=219, bytes=38264, drop=0, errs=0, coll=0
port 3: rx pkts=152, bytes=25476, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=228, bytes=48003, drop=0, errs=0, coll=0
*** s4 -----
OFFST_PORT reply (xid=0x2): 4 ports
port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=0, bytes=0, drop=246, errs=0, coll=0
port 1: rx pkts=27, bytes=2290, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=328, bytes=59340, drop=1, errs=0, coll=0
port 2: rx pkts=228, bytes=48003, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=152, bytes=25476, drop=0, errs=0, coll=0
port 3: rx pkts=75, bytes=11977, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=263, bytes=47197, drop=0, errs=0, coll=0
*** s5 -----
OFFST_PORT reply (xid=0x2): 3 ports
port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=0, bytes=0, drop=239, errs=0, coll=0
port 1: rx pkts=27, bytes=2290, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=322, bytes=58544, drop=0, errs=0, coll=0
port 2: rx pkts=263, bytes=47197, drop=0, errs=0, frame=0, over=0, crc=0
tx pkts=75, bytes=11977, drop=0, errs=0, coll=0
mininet:

```

Figure: Port status

# Outline

Introduction

Mininet Installation

Mininet Command Line Tool

**Remote Controller**

- POX controller default running

- POX controller with extension modules

Custom Controller and Topology



# POX controller default running

- Run Controller with:
  - »python pox.py
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables
  - Destination Host Unreachable while try to ping it



# POX controller default running

- Run Controller with:
  - »python pox.py
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables
  - Destination Host Unreachable while try to ping it



## POX controller default running

- Run Controller with:
  - »python pox.py
- Run topology from command line with remote controller:
  - `sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633`
- Results:
  - Empty flow tables
  - Destination Host Unreachable while try to ping it



## POX controller default running

- Run Controller with:
  - »python pox.py
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote,  
ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables
  - Destination Host Unreachable while try to ping it





## POX controller default running

- Run Controller with:
  - »python pox.py
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables
  - Destination Host Unreachable while try to ping it



## POX controller default running

- Run Controller with:
  - »python pox.py
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables
  - Destination Host Unreachable while try to ping it



## POX controller default running

- Run Controller with:
  - »python pox.py
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables
  - Destination Host Unreachable while try to ping it



# POX controller with layer2 learning

- Run Controller with:
  - »python pox.py forwarding.l2\_learning
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables initially
  - When it is tried to ping from one host to another,
    - Forwarding l2\_learning module is run in controller, per flow
    - It installs destination rules for each flow.
    - Hosts are not reachable



# POX controller with layer2 learning

- Run Controller with:
  - »python pox.py forwarding.l2\_learning
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables initially
  - When it is tried to ping from one host to another, forwarding l2\_learning module is run in controller, per flow.
    - It installs destination rules for each flow.
    - Hosts are not connected



## POX controller with layer2 learning

- Run Controller with:
  - »python pox.py forwarding.l2\_learning
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables initially
  - When it is tried to ping from one host to another, the pox controller will learn the destination ip address and install the necessary rules for each flow.
  - This is called as layer2 learning.



## POX controller with layer2 learning

- Run Controller with:
  - »python pox.py forwarding.l2\_learning
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables initially
  - When it is tried to ping from one host to another, the controller learns the MAC addresses of the hosts and creates the necessary rules for each flow.



## POX controller with layer2 learning

- Run Controller with:
  - »python pox.py forwarding.l2\_learning
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables initially
  - When it is tried to ping from one host to another,
    - forwarding.l2\_learning module is run in controller per flow
    - it installs exact-match rules for each flow.
    - Host becomes reachable





## POX controller with layer2 learning

- Run Controller with:
  - »python pox.py forwarding.l2\_learning
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables initially
  - When it is tried to ping from one host to another,
    - forwarding.l2\_learning module is run in controller per flow
    - it installs exact-match rules for each flow.
    - Host becomes reachable



## POX controller with layer2 learning

- Run Controller with:
  - »python pox.py forwarding.l2\_learning
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables initially
  - When it is tried to ping from one host to another,
    - forwarding.l2\_learning module is run in controller per flow
    - it installs exact-match rules for each flow.
    - Host becomes reachable



## POX controller with layer2 learning

- Run Controller with:
  - »python pox.py forwarding.l2\_learning
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables initially
  - When it is tried to ping from one host to another,
    - forwarding.l2\_learning module is run in controller per flow
    - it installs exact-match rules for each flow.
    - Host becomes reachable



## POX controller with layer2 learning

- Run Controller with:
  - »python pox.py forwarding.l2\_learning
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables initially
  - When it is tried to ping from one host to another,
    - forwarding.l2\_learning module is run in controller per flow
    - it installs exact-match rules for each flow.
    - Host becomes reachable



## POX controller with layer2 learning

- Run Controller with:
  - »python pox.py forwarding.l2\_learning
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote, ip=127.0.0.1, port=6633
- Results:
  - Empty flow tables initially
  - When it is tried to ping from one host to another,
    - forwarding.l2\_learning module is run in controller per flow
    - it installs exact-match rules for each flow.
    - Host becomes reachable



# POX controller with spanning tree in DEBUG mode

- Run Controller with:
  - »python pox.py forwarding.l2\_learning  
openflow.spanning\_tree -no-flood -hold-down log.level  
-DEBUG samples.pretty\_log openflow.discovery  
host\_tracker info.packet\_dump
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote,  
ip=127.0.0.1, port=6633



# POX controller with spanning tree in DEBUG mode

- Run Controller with:
  - »python pox.py forwarding.l2\_learning  
openflow.spanning\_tree -no-flood -hold-down log.level  
-DEBUG samples.pretty\_log openflow.discovery  
host\_tracker info.packet\_dump
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote,  
ip=127.0.0.1, port=6633



# POX controller with spanning tree in DEBUG mode

- Run Controller with:
  - »python pox.py forwarding.l2\_learning  
openflow.spanning\_tree -no-flood -hold-down log.level  
-DEBUG samples.pretty\_log openflow.discovery  
host\_tracker info.packet\_dump
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote,  
ip=127.0.0.1, port=6633





# POX controller with spanning tree in DEBUG mode

- Run Controller with:
  - »python pox.py forwarding.l2\_learning  
openflow.spanning\_tree -no-flood -hold-down log.level  
-DEBUG samples.pretty\_log openflow.discovery  
host\_tracker info.packet\_dump
- Run topology from command line with remote controller:
  - sudo mn - - topo=linear,5 - - controller=remote,  
ip=127.0.0.1, port=6633



- Results:

- OpenFlow switches may optionally support 802.1D Spanning Tree Protocol.
- Those switches process all 802.1D packets locally before performing flow lookup.
  - Flow tables are filled with entries including "actions=CONTROLLER"
- After pinging from one host to another,
  - forwarding.l2\_learning module is run in controller per flow
  - it installs exact-match rules for each flow.
  - Host becomes reachable
  - Switches that do not support 802.1D spanning tree must allow the controller to specify the port status for packet flooding through the port-mod messages.



- Results:

- OpenFlow switches may optionally support 802.1D Spanning Tree Protocol.
- Those switches process all 802.1D packets locally before performing flow lookup.
  - Flow tables are filled with entries including "actions=CONTROLLER"
- After pinging from one host to another,
  - forwarding.l2\_learning module is run in controller per flow
  - it installs exact-match rules for each flow.
  - Host becomes reachable
  - Switches that do not support 802.1D spanning tree must allow the controller to specify the port status for packet flooding through the port-mod messages.



- Results:

- OpenFlow switches may optionally support 802.1D Spanning Tree Protocol.
- Those switches process all 802.1D packets locally before performing flow lookup.
  - Flow tables are filled with entries including "actions=CONTROLLER"
- After pinging from one host to another,
  - forwarding.l2\_learning module is run in controller per flow
  - it installs exact-match rules for each flow.
  - Host becomes reachable
  - Switches that do not support 802.1D spanning tree must allow the controller to specify the port status for packet flooding through the port-mod messages.



- Results:

- OpenFlow switches may optionally support 802.1D Spanning Tree Protocol.
- Those switches process all 802.1D packets locally before performing flow lookup.
  - Flow tables are filled with entries including "actions=CONTROLLER"
- After pinging from one host to another,
  - forwarding.l2\_learning module is run in controller per flow
  - it installs exact-match rules for each flow.
  - Host becomes reachable
  - Switches that do not support 802.1D spanning tree must allow the controller to specify the port status for packet flooding through the port-mod messages.



- Results:

- OpenFlow switches may optionally support 802.1D Spanning Tree Protocol.
- Those switches process all 802.1D packets locally before performing flow lookup.
  - Flow tables are filled with entries including "actions=CONTROLLER"
- After pinging from one host to another,
  - forwarding.l2\_learning module is run in controller per flow
  - it installs exact-match rules for each flow.
  - Host becomes reachable
  - Switches that do not support 802.1D spanning tree must allow the controller to specify the port status for packet flooding through the port-mod messages.



- Results:

- OpenFlow switches may optionally support 802.1D Spanning Tree Protocol.
- Those switches process all 802.1D packets locally before performing flow lookup.
  - Flow tables are filled with entries including "actions=CONTROLLER"
- After pinging from one host to another,
  - forwarding.l2\_learning module is run in controller per flow
  - it installs exact-match rules for each flow.
  - Host becomes reachable
  - Switches that do not support 802.1D spanning tree must allow the controller to specify the port status for packet flooding through the port-mod messages.



- Results:

- OpenFlow switches may optionally support 802.1D Spanning Tree Protocol.
- Those switches process all 802.1D packets locally before performing flow lookup.
  - Flow tables are filled with entries including "actions=CONTROLLER"
- After pinging from one host to another,
  - forwarding.l2\_learning module is run in controller per flow
  - it installs exact-match rules for each flow.
  - Host becomes reachable
  - Switches that do not support 802.1D spanning tree must allow the controller to specify the port status for packet flooding through the port-mod messages.





- Results:

- OpenFlow switches may optionally support 802.1D Spanning Tree Protocol.
- Those switches process all 802.1D packets locally before performing flow lookup.
  - Flow tables are filled with entries including "actions=CONTROLLER"
- After pinging from one host to another,
  - forwarding.l2\_learning module is run in controller per flow
  - it installs exact-match rules for each flow.
  - Host becomes reachable
  - Switches that do not support 802.1D spanning tree must allow the controller to specify the port status for packet flooding through the port-mod messages.



- Results:

- OpenFlow switches may optionally support 802.1D Spanning Tree Protocol.
- Those switches process all 802.1D packets locally before performing flow lookup.
  - Flow tables are filled with entries including "actions=CONTROLLER"
- After pinging from one host to another,
  - forwarding.l2\_learning module is run in controller per flow
  - it installs exact-match rules for each flow.
  - Host becomes reachable
  - Switches that do not support 802.1D spanning tree must allow the controller to specify the port status for packet flooding through the port-mod messages.



## Link down/up

- Afterwards, try to link *down* in running topology:
  - Down the link between switch1 and switch2 by following command:
    - »link s1 s2 down
    - »h1 ping h2 -c 3 results:
      - In controller side: openflow.spanning\_tree tries to find alternative link between h1 and h2. However, openflow.discovery→timeout
      - In mininet side: Destination Host Unreachable



## Link down/up

- Afterwards, try to link *down* in running topology:
  - Down the link between switch1 and switch2 by following command:
    - »link s1 s2 down
    - »h1 ping h2 -c 3 results:
      - In controller side: openflow.spanning\_tree tries to find alternative link between h1 and h2. However, openflow.discovery→timeout
      - In mininet side: Destination Host Unreachable



## Link down/up

- Afterwards, try to link *down* in running topology:
  - Down the link between switch1 and switch2 by following command:
    - »link s1 s2 down
    - »h1 ping h2 -c 3 results:
      - In controller side: openflow.spanning\_tree tries to find alternative link between h1 and h2. However, openflow.discovery→timeout
      - In mininet side: Destination Host Unreachable



## Link down/up

- Afterwards, try to link *down* in running topology:
  - Down the link between switch1 and switch2 by following command:
    - »link s1 s2 down
    - »h1 ping h2 -c 3 results:
      - In controller side: openflow.spanning\_tree tries to find alternative link between h1 and h2. However, openflow.discovery→timeout
      - In mininet side: Destination Host Unreachable



## Link down/up

- Afterwards, try to link *down* in running topology:
  - Down the link between switch1 and switch2 by following command:
    - »link s1 s2 down
    - »h1 ping h2 -c 3 results:
      - In controller side: openflow.spanning\_tree tries to find alternative link between h1 and h2. However, openflow.discovery→timeout
      - In mininet side: Destination Host Unreachable



## Link down/up

- Afterwards, try to link *down* in running topology:
  - Down the link between switch1 and switch2 by following command:
    - »link s1 s2 down
    - »h1 ping h2 -c 3 results:
      - In controller side: openflow.spanning\_tree tries to find alternative link between h1 and h2. However, openflow.discovery→timeout
      - In mininet side: Destination Host Unreachable





# Outline

Introduction

Mininet Installation

Mininet Command Line Tool

Remote Controller

**Custom Controller and Topology**



# Custom Controller and Topology written in python language

DEMO!!!



Thank you for your attention.

Any questions?