# Introduction to Mininet & POX

Lakiotakis Emmanouil

HY436

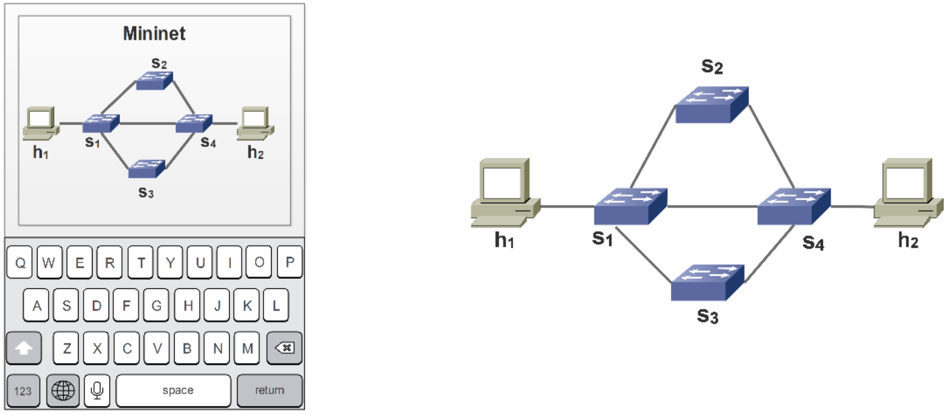
Fall 2020

# Outline

- Mininet
- OpenFlow
- POX

# Mininet

- Mininet is a **network emulator**
- Allows emulating **end-hosts, switches, routers and links** using software similar to real-world elements



(a) Emulated Network

(b) Hardware Network

# Mininet

- Mininet advantages
  - Fast network setup
  - Allows custom topologies for experiments
  - Supports OpenFlow enabling custom packet forwarding
  - Reduces cost for experiments since hardware is not required
  - Enables executing commands/scripts in each host (using XMing)
- Limitations
  - Suffers from resource limitations
    - All virtual network components run a single system
    - Resource requirements for large scale topologies
  - All end-hosts share the same file system
  - Lacks of virtual timing capabilities

# Mininet topologies

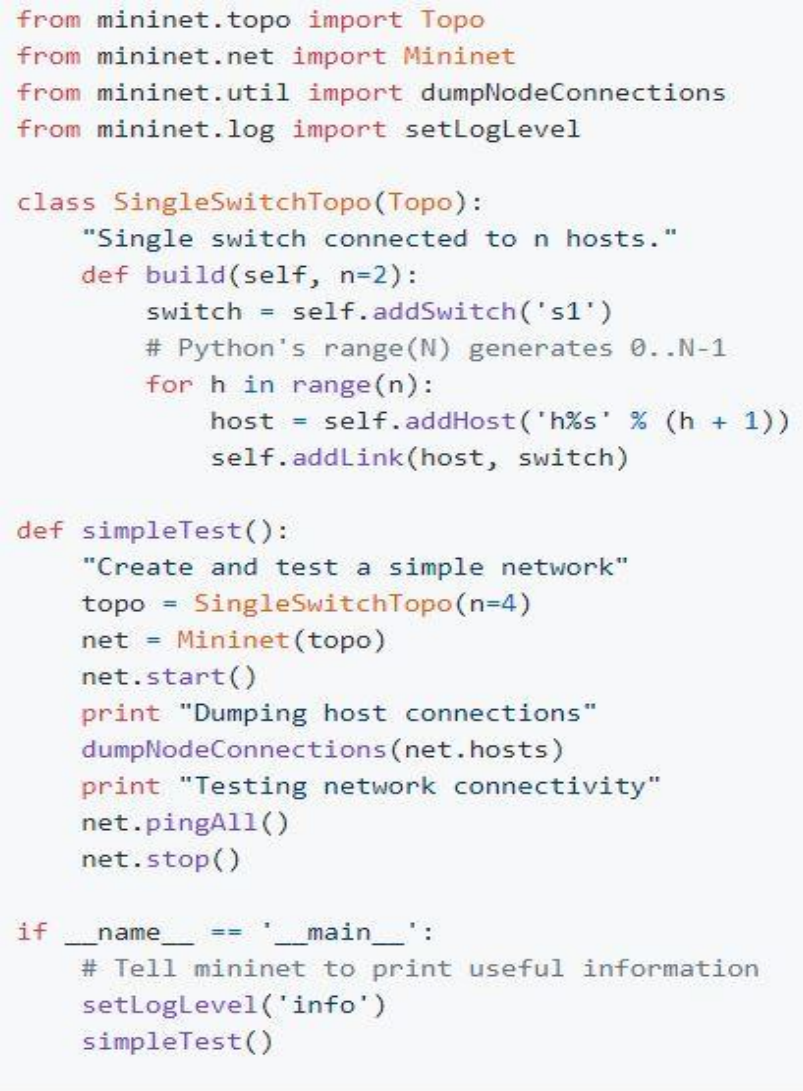
- Mininet allows creating parametrized topologies using Python scripts
- This is useful for experimenting with different network scenarios

- Useful methods

| Method | Meaning |
|---|---|
| addSwitch() | Adds a switch to the topology and returns the switch name |
| addHost() | Adds a host to the topology and returns the host name |
| addLink() | adds a bidirectional link to the topology |
| pingAll() | Connectivity test using ping between all hosts |
| net.hosts | List all the hosts in a network |
| dumpNodeConnections | Dumps connections to/from a set of nodes |

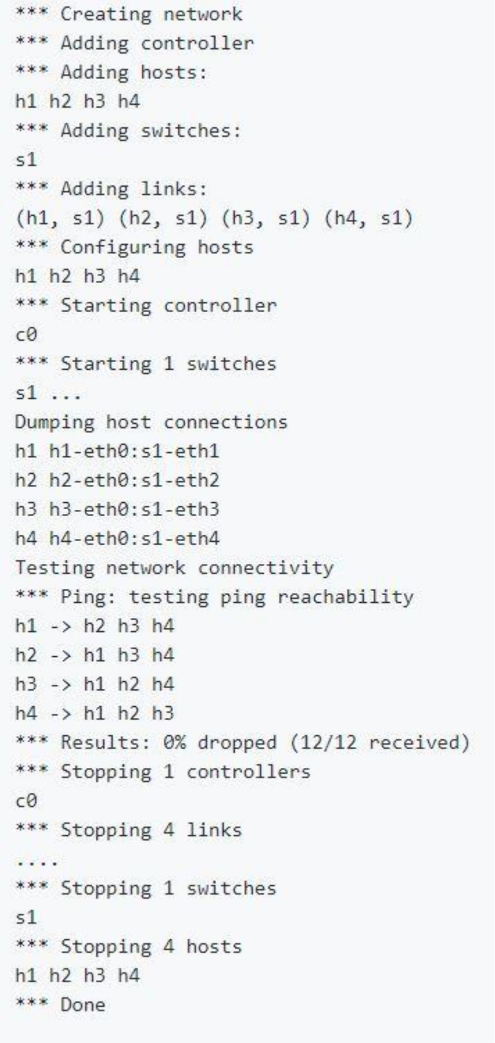
# Mininet Topology Example

## simple_test.py



```python
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def build(self, n=2):
        switch = self.addSwitch('s1')
        # Python's range(N) generates 0..N-1
        for h in range(n):
            host = self.addHost('h%s' % (h + 1))
            self.addLink(host, switch)

def simpleTest():
    "Create and test a simple network"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    net.stop()

if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTest()
```

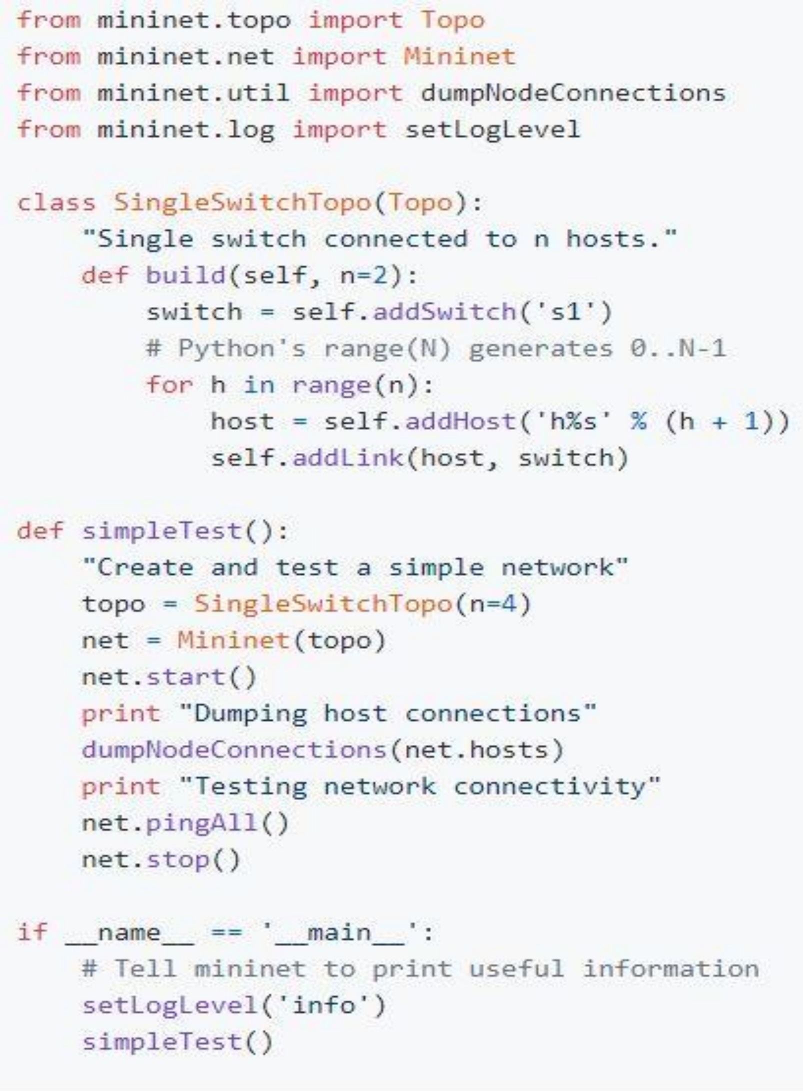
**sudo python simple_test.py**



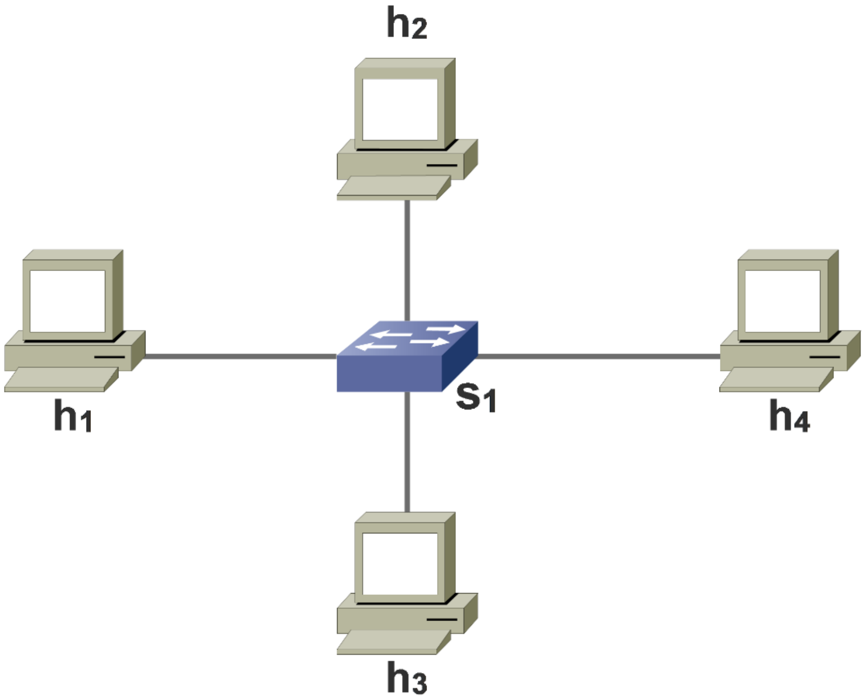
```
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
Testing network connectivity
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
*** Stopping 1 controllers
c0
*** Stopping 4 links
....
*** Stopping 1 switches
s1
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
```

# Mininet Topology Example

## simple_test.py



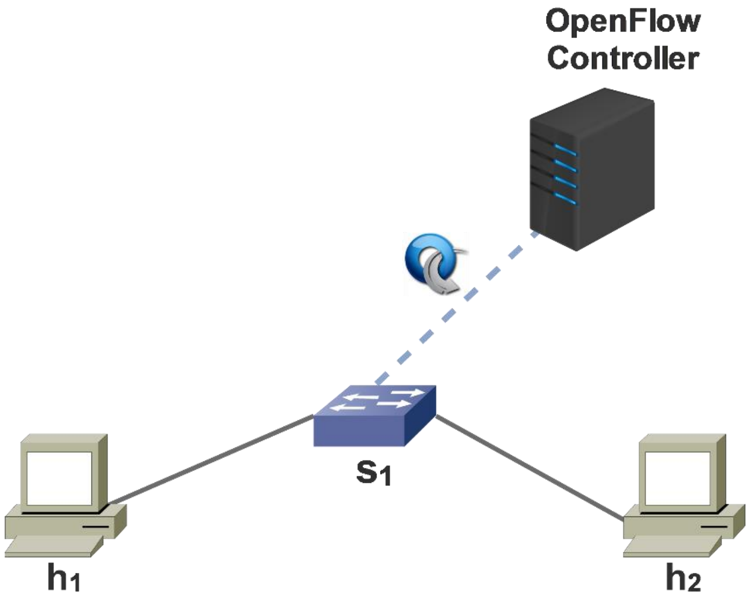
```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

class SingleSwitchTopo(Topo):
    "Single switch connected to n hosts."
    def build(self, n=2):
        switch = self.addSwitch('s1')
        # Python's range(N) generates 0..N-1
        for h in range(n):
            host = self.addHost('h%s' % (h + 1))
            self.addLink(host, switch)

def simpleTest():
    "Create and test a simple network"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo)
    net.start()
    print "Dumping host connections"
    dumpNodeConnections(net.hosts)
    print "Testing network connectivity"
    net.pingAll()
    net.stop()

if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')
    simpleTest()
```



# Mininet topologies

- Apart from writing Python scripts for creating network topologies, Mininet allows topology setup using command line extending existing default topologies

```
sudo mn
```



The default topology is the minimal topology, which includes **one OpenFlow switch** connected to **two hosts**, plus the **OpenFlow reference controller**.

# Mininet topologies

- Other useful Mininet commands

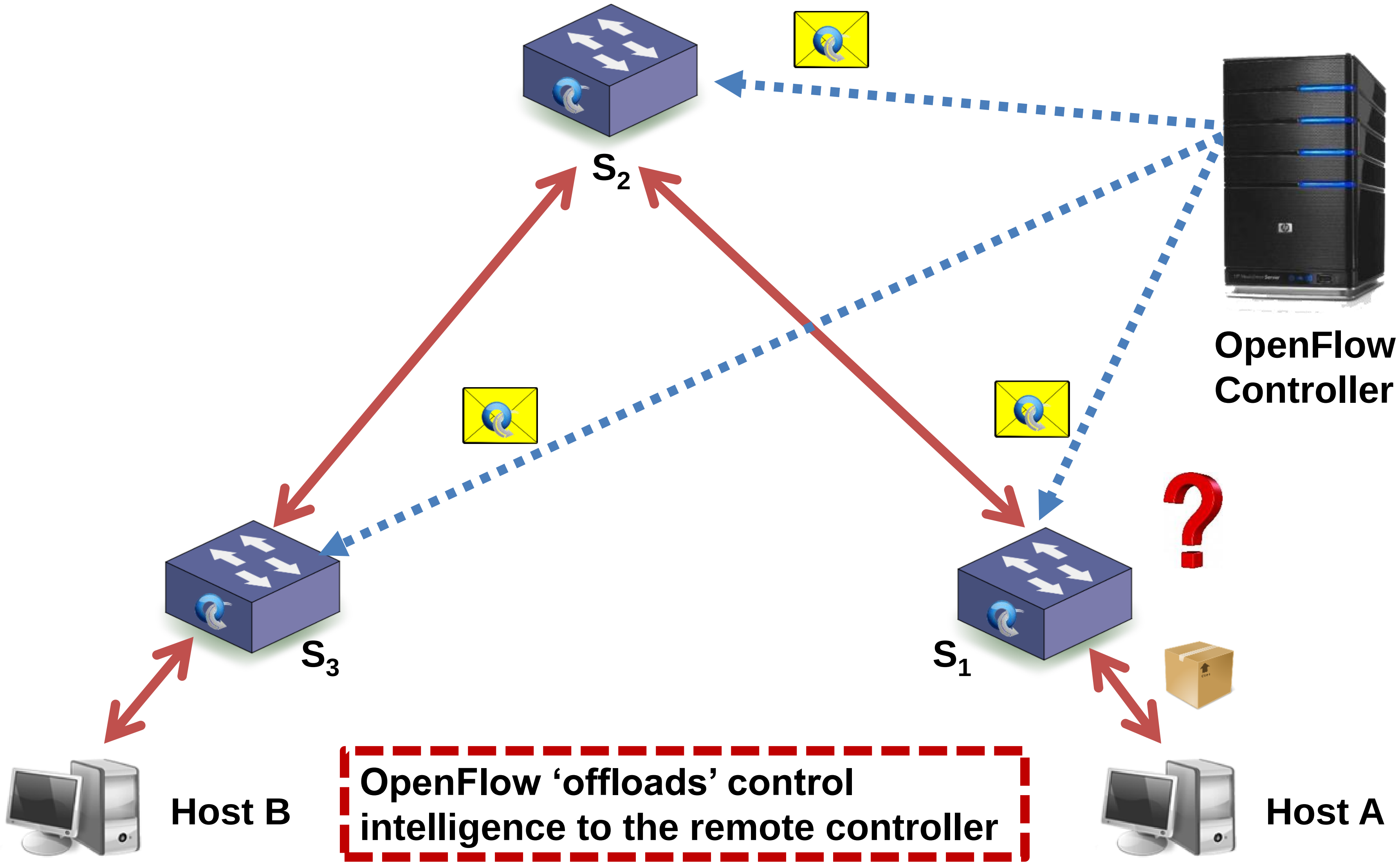
| Command | Meaning |
|---|---|
| help | Display available commands |
| nodes | List nodes |
| net | Display links |
| dump | Display dump information |
| xterm $h_i$ $h_j$ | Open terminal to hosts $h_i$, $h_j$ |
| exit/quit | Exit/quit Mininet |

- Mininet clean up: sudo mn –c (before each experiment)

# Mininet topologies

- Other topology example
  - sudo mn --topo single,3 --mac --switch ovsk --controller remote
  - This command creates a simple network topology with 3 hosts, each host n has a separate IP using 10.0.0.n format, the MAC for each host is a function of the assigned IP address (00:00:00:00:00:n) and the OpenFlow switch will be coordinated by a remote OpenFlow controller
- Usually, hosts are named h1,..,hN and switches s1,..,sN, host $h_1$'s default interface is h1-eth0 and switch $s_1$'s first port is s1-eth1

# OpenFlow



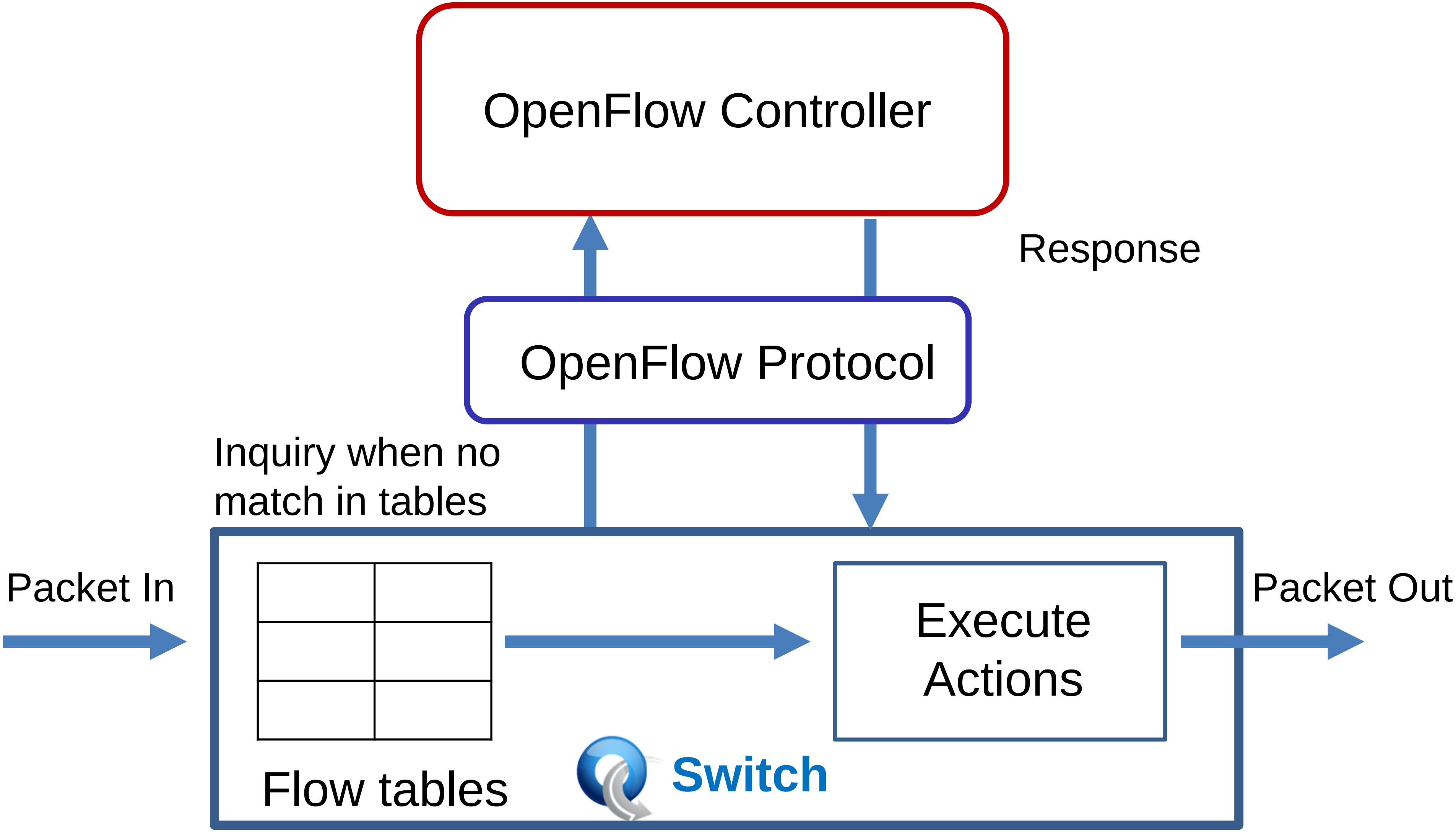
OpenFlow ‘offloads’ control intelligence to the remote controller

# OpenFlow differentiation

- **A conventional switch:**
  - Keeps data plane and control plane in the same device

- **An OpenFlow switch:**
  - Separates data plane and control plane
  - The data plane includes a Flow Table and a specific action for each flow entry
  - The control plane includes the controller that installs the flow entry in the flow table
  - The data plane still resides on the switch
  - The control plane is migrated to the controller

# Pipeline Processing



# What can I do with OpenFlow?

- Easily deploy innovative routing and switching protocols
- Secure networks
- Build scalable data center networks
- More energy-efficient networks
- Virtual machine mobility

# Introduction to POX

- What is POX?
  - Python-based OpenFlow controller framework
  - Supports OpenFlow specification version 1.0
  - Widely used and supported
  - Easy learning curve
  - Slow performance but easy and fast development for SDN controller applications
  - Generally applied for research, experimentation, demonstrations

# Introduction to POX

- How does POX work?
  - Event-driven programming model
  - Controller functionalities implemented on Components
  - Registers event listeners and/or handlers & creates objects of any Component class:
    - https://github.com/noxrepo/pox/blob/carp/pox/forwarding/hub.py
    - https://github.com/noxrepo/pox/blob/carp/pox/forwardi ng/l2_learning.py

# Introduction to POX

- Each switch in the topology has a unique datapath ID (DPID)
- POX controller and network switches exchange messages
  - Communication from the controller to the switch is performed by controller code which sends an OpenFlow message to a particular switch
  - When messages are coming from the switch, they show up in POX as events for which you can write event handlers
- DPIDs are used by the POX Controller to send messages to the switches and also recognize the switches that send messages/raise events to the Controller

# Events in POX

- **ConnectionUp**:
  - Raised in response to the establishment of a new control channel between a switch and the Controller
- **PacketIn**:
  - Fired when the Controller receives an OpenFlow packet-in message indicating that a packet arriving at a switch port has either failed to match all entries in the table, or the matching entry included an action specifying to send the packet to the controller
- Other events: ConnectionDown, PortStatus, FlowRemoved etc.
- Events are treated by handlers

# OpenFlow Messages in POX

- **ofp_packet_out:**
  - Instructs the switch to send (or sometimes discard) a packet, which might be constructed at the controller, or might be the one that the switch received, buffered in the datapath and forwarded to the controller
  - Attributes:
    - **buffer_id**: ID of the buffer in which the packet is stored at the datapath
    - **in_port**: The port number for the packet initially arrived on
    - **actions**: A list of actions to apply
    - **data**:  The data to be sent (or None if sending an existing buffer via its buffer_id).

# OpenFlow Messages in POX

- **ofp_flow_mod**:
  - Instructs the switch to install a flow table entry
  - Flow table entries match some fields of the incoming packets, and execute a list of actions on the matched packets
  - Major fields are:
    - **idle_timeout**: The rule will expire if it is not matched in 'idle_timeout' seconds
    - **hard_timeout**: The rule will expire after 'hard_timeout' seconds
    - **actions**: A list of actions that will be applied to the matched packets, each desired action object is then appended to this list and **they are executed in order**
    - **buffer_id**: The ID of the buffer on the datapath that the new flow will be applied to
    - **priority**: The priority at which a rule will match, higher numbers higher priority
    - **match**: The match structure for the rule to match on

# OpenFlow Actions

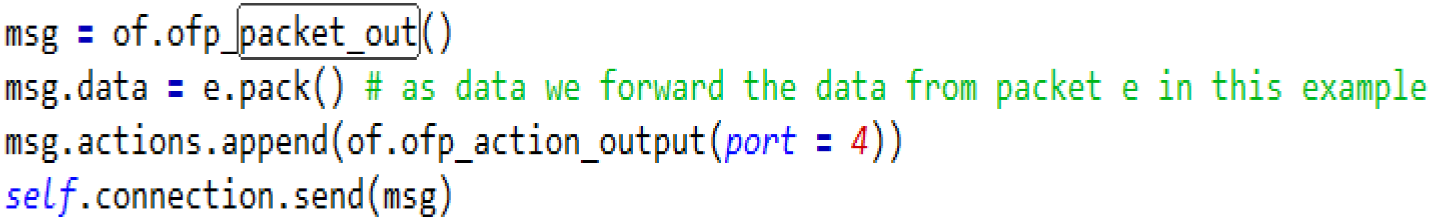
- OpenFlow actions are applied to packets that match a rule installed at the switch
- **ofp_action_output**:
  - An action for use with **ofp_packet_out** and **ofp_flow_mod**, which specifies a switch port that you wish to send the packet out of
  - Example: Create an output action that would send packets to all ports (flooding):
    - out_action=of.ofp_action_output(port=of.OFPP_FLOOD)

# OpenFlow Actions

- Other OpenFlow actions allow to set/modify:
  - VLAN ID
  - Ethernet source or destination address
  - IP source or destination address
  - IP Type of Service
  - TCP/UDP source or destination port
- For more details, please visit:
  - https://openflow.stanford.edu/display/ONL/POX+Wiki

# OpenFlow Messages in POX

- **ofp_packet_out example:**

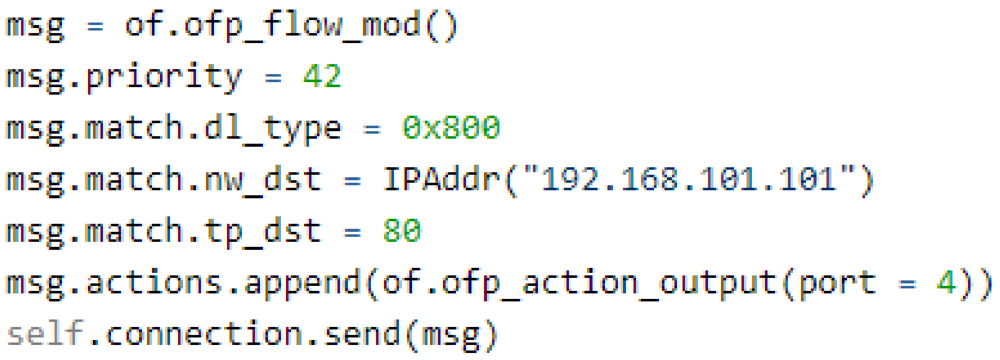


```
msg = of.ofp_packet_out()
msg.data = e.pack() # as data we forward the data from packet e in this example
msg.actions.append(of.ofp_action_output(port = 4))
self.connection.send(msg)
```

**Packet e is forwarded from switch port 4**

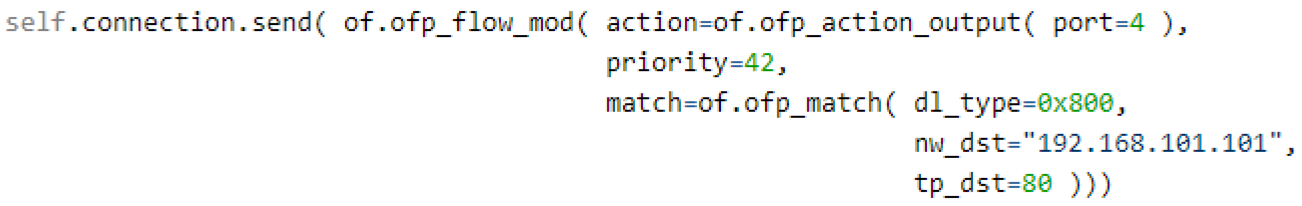
# OpenFlow Messages in POX

- **ofp_flow_mod example:**



```
msg = of.ofp_flow_mod()
msg.priority = 42
msg.match.dl_type = 0x800
msg.match.nw_dst = IPAddr("192.168.101.101")
msg.match.tp_dst = 80
msg.actions.append(of.ofp_action_output(port = 4))
self.connection.send(msg)
```

**OR**



```
self.connection.send( of.ofp_flow_mod( action=of.ofp_action_output( port=4 ),
                                       priority=42,
                                       match=of.ofp_match( dl_type=0x800,
                                                           nw_dst="192.168.101.101",
                                                           tp_dst=80 )))
```

**Traffic to 192.168.101.101:80 should be sent out from switch port 4**

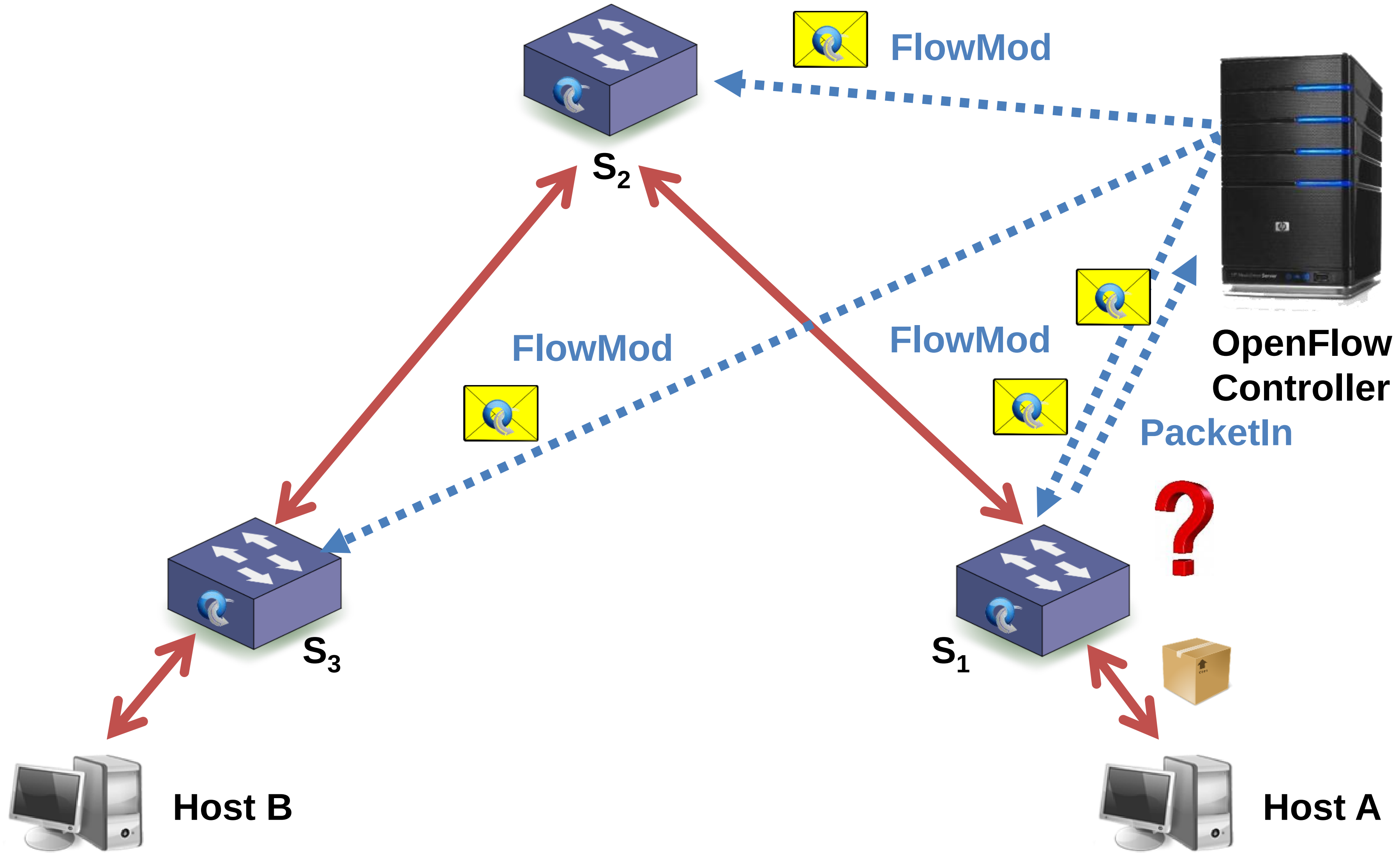
# OpenFlow Match Structure

- Major **ofp_match** class attributes:

| Attribute | Meaning |
|---|---|
| in_port | Switch port number the packet arrived on |
| dl_src | Ethernet (MAC) source address |
| dl_dst | Ethernet (MAC) destination address |
| dl_type | Ethertype / length (e.g. 0x0800 = IPv4) |
| nw_proto | IP protocol (e.g., 6 denotes TCP) |
| nw_src | IP source address |
| nw_dst | IP destination address |
| tp_src | TCP/UDP source port |
| tp_dst | TCP/UDP destination port |

- OpenFlow also allows match from an existing packet using the ofp_match.from_packet() method

# OpenFlow in an example



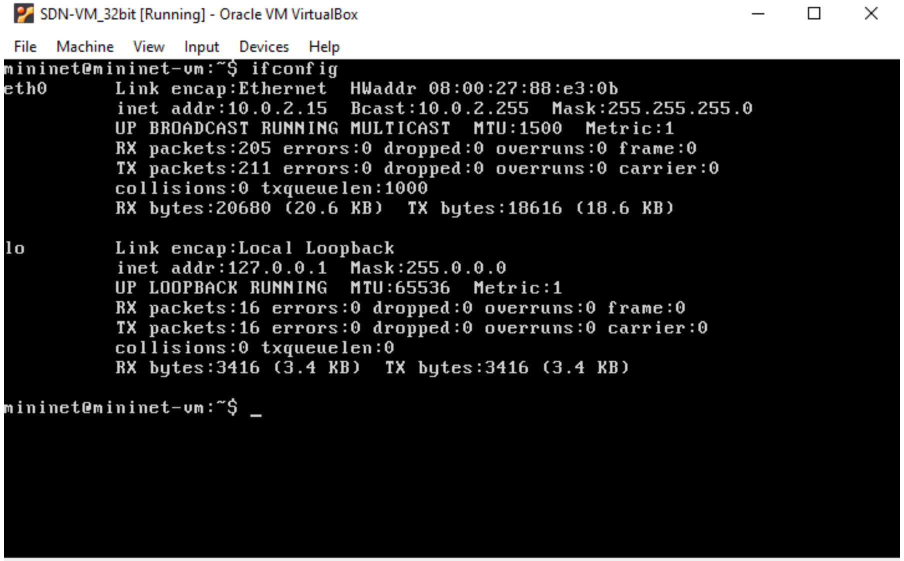
# Other useful methods

- Displaying a debug message in POX:
  - log.debug(‘saw new MAC!’)
- Displaying an error message in POX:
  - log.error(‘unexpected operation’)
- Python print command also works for debugging or displaying variables values

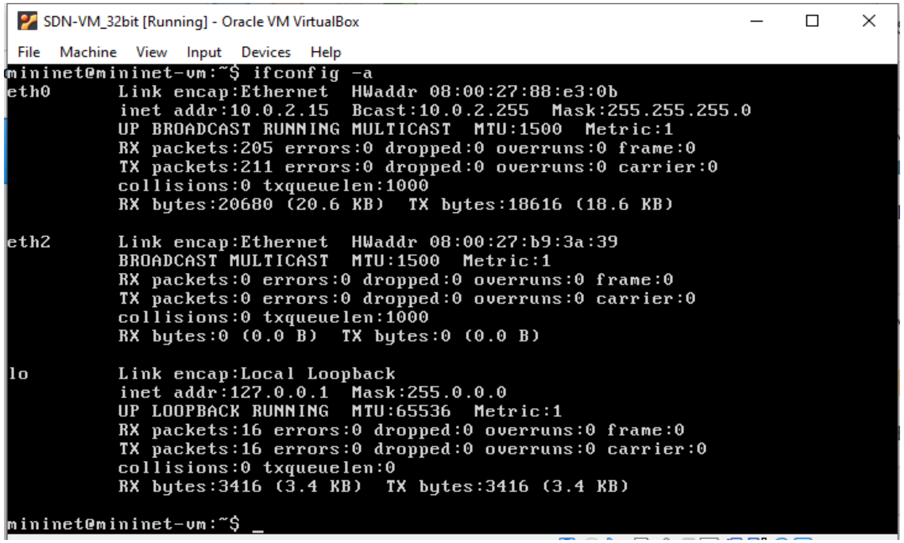
# VM Setup

- Import Virtual Machine Image
- Log in using as credentials
  - Username: mininet
  - Password: mininet
- Your VM should have 2 network interfaces
  - One NAT interface that can be used for Internet access and
  - One host-only interface to enable communication with the host machine

In order to list all network interfaces (also those without IP address) :



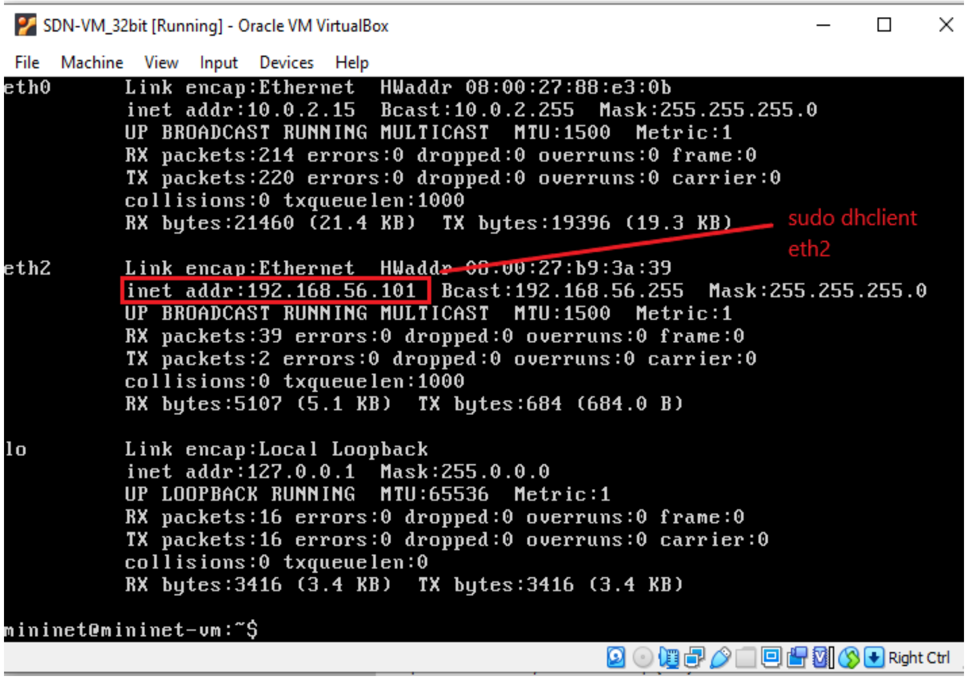
```
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:88:e3:0b
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:205 errors:0 dropped:0 overruns:0 frame:0
          TX packets:211 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20680 (20.6 KB)  TX bytes:18616 (18.6 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3416 (3.4 KB)  TX bytes:3416 (3.4 KB)

mininet@mininet-vm:~$ _
```



```
mininet@mininet-vm:~$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr 08:00:27:88:e3:0b
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:205 errors:0 dropped:0 overruns:0 frame:0
          TX packets:211 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20680 (20.6 KB)  TX bytes:18616 (18.6 KB)

eth2      Link encap:Ethernet  HWaddr 08:00:27:b9:3a:39
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3416 (3.4 KB)  TX bytes:3416 (3.4 KB)

mininet@mininet-vm:~$ _
```

# VM Setup

- In case of a network interface without IP address, you can assign an IP using command
  - sudo dhclient ‘ethX’
  - “ethX’ is the interface name
- The host only interface  will be used for SSH access to the VM

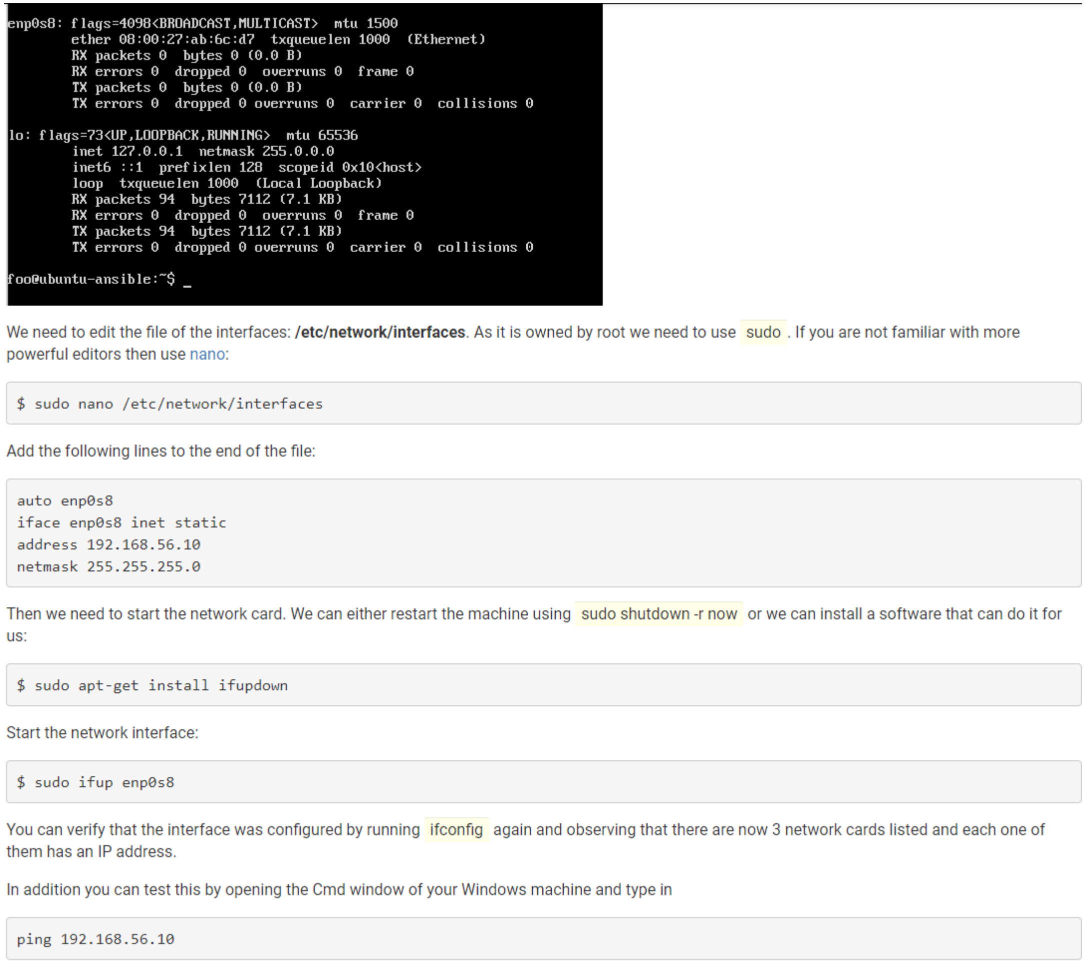


```
eth0      Link encap:Ethernet  HWaddr 08:00:27:88:e3:0b
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:214 errors:0 dropped:0 overruns:0 frame:0
          TX packets:220 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:21460 (21.4 KB)  TX bytes:19396 (19.3 KB)

eth2      Link encap:Ethernet  HWaddr 08:00:27:b9:3a:39
          inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:39 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:5107 (5.1 KB)  TX bytes:684 (684.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3416 (3.4 KB)  TX bytes:3416 (3.4 KB)

mininet@mininet-vm:~$
```

sudo dhclient eth2

# VM Setup

- In case that dhclient command does not work, please follow the instructions on the right



```
enp0s8: flags=4098<BROADCAST,MULTICAST>  mtu 1500
        ether 08:00:27:ab:6c:d7  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 94  bytes 7112 (7.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 94  bytes 7112 (7.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

foo@ubuntu-ansible:~$ _
```

We need to edit the file of the interfaces: **/etc/network/interfaces**. As it is owned by root we need to use `sudo`. If you are not familiar with more powerful editors then use nano:

```
$ sudo nano /etc/network/interfaces
```

Add the following lines to the end of the file:

```
auto enp0s8
iface enp0s8 inet static
address 192.168.56.10
netmask 255.255.255.0
```

Then we need to start the network card. We can either restart the machine using `sudo shutdown -r now` or we can install a software that can do it for us:

```
$ sudo apt-get install ifupdown
```

Start the network interface:

```
$ sudo ifup enp0s8
```

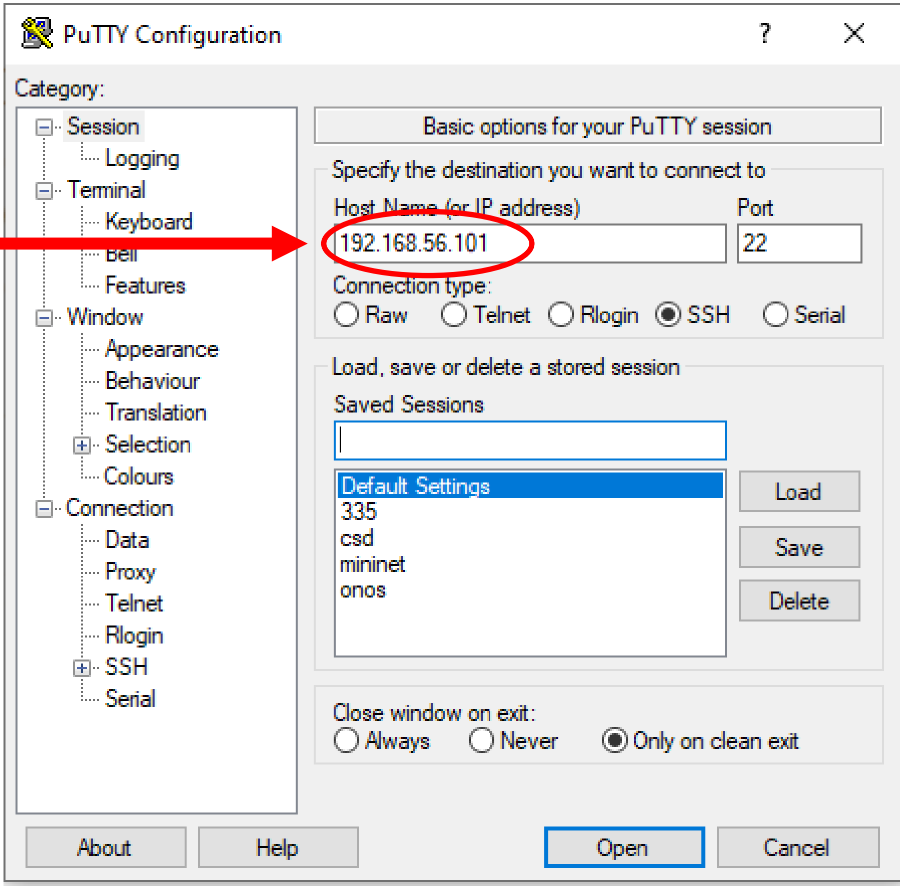
You can verify that the interface was configured by running `ifconfig` again and observing that there are now 3 network cards listed and each one of them has an IP address.

In addition you can test this by opening the Cmd window of your Windows machine and type in

```
ping 192.168.56.10
```

# Access VM via SSH

- Windows users should use SSH client software (e.g. PuTTy) with the assigned IP to the host-only interface
- Also Xming server should be installed as well as enabling X11 forwarding **before establishing the SSH session** (Session-> SSH -> X11 -> Enable X11 forwarding)
- For Linux/Mac OS X:
  - ssh -X mininet@[192.168.56.101]



PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): 192.168.56.101

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Load, save or delete a stored session

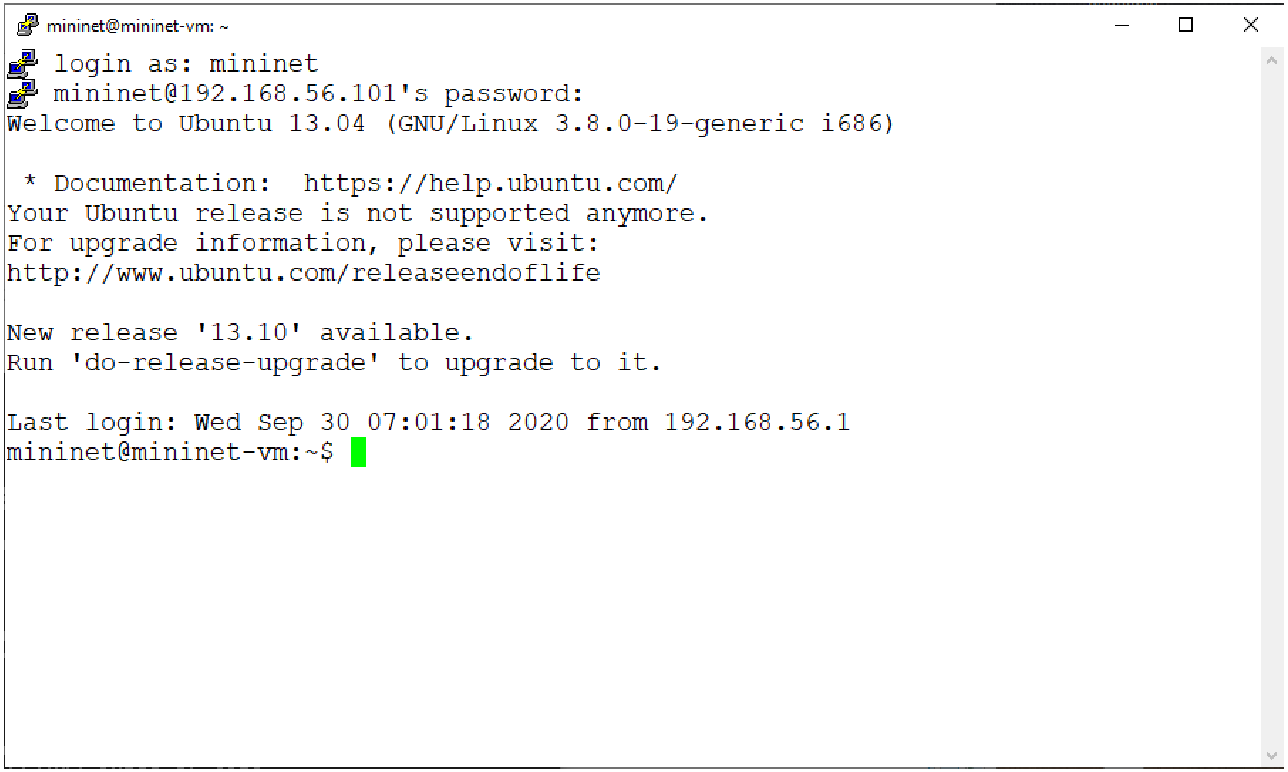
Saved Sessions

Default Settings
335
csd
mininet
onos

Close window on exit: ○ Always ○ Never ● Only on clean exit

# Access VM via SSH

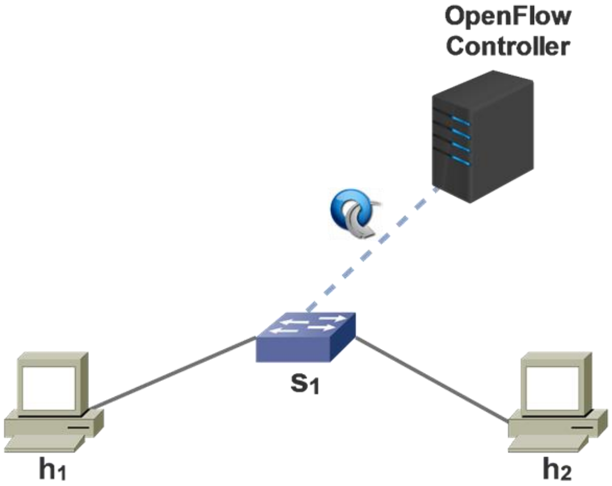
- After establishing the SSH session:



```
login as: mininet
mininet@192.168.56.101's password:
Welcome to Ubuntu 13.04 (GNU/Linux 3.8.0-19-generic i686)

 * Documentation:  https://help.ubuntu.com/
Your Ubuntu release is not supported anymore.
For upgrade information, please visit:
http://www.ubuntu.com/releaseendoflife

New release '13.10' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Sep 30 07:01:18 2020 from 192.168.56.1
mininet@mininet-vm:~$
```

# Combine Mininet with POX

- Set up your network topology:
  - sudo mn --topo single,2 --mac --switch ovsk --controller remote



- Fire up the controller:
  - cd ~/pox
  - ./pox.py log.level --DEBUG forwarding.hub

https://github.com/noxrepo/pox/blob/carp/pox/forwarding/hub.py

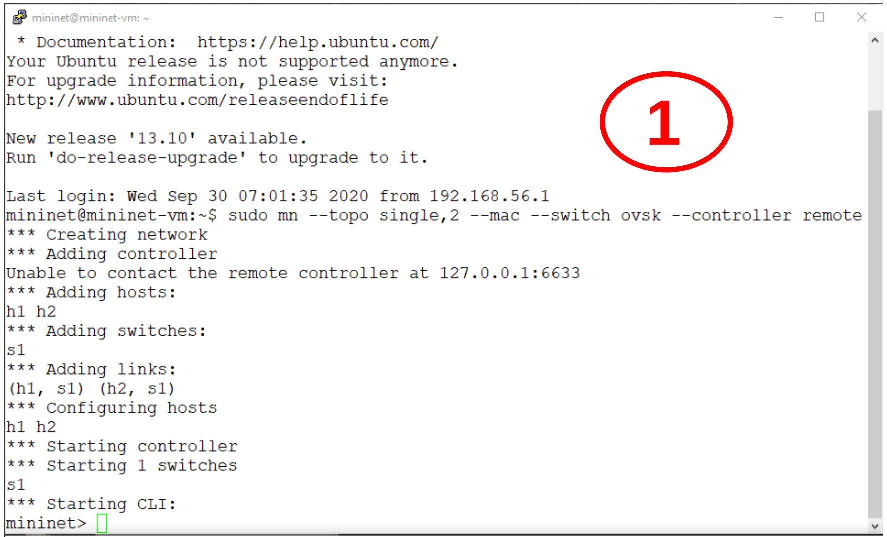
- Try also:
  - ./pox.py log.level --DEBUG forwarding.l2_learning

https://github.com/noxrepo/pox/blob/carp/pox/forwarding/l2_learning.py

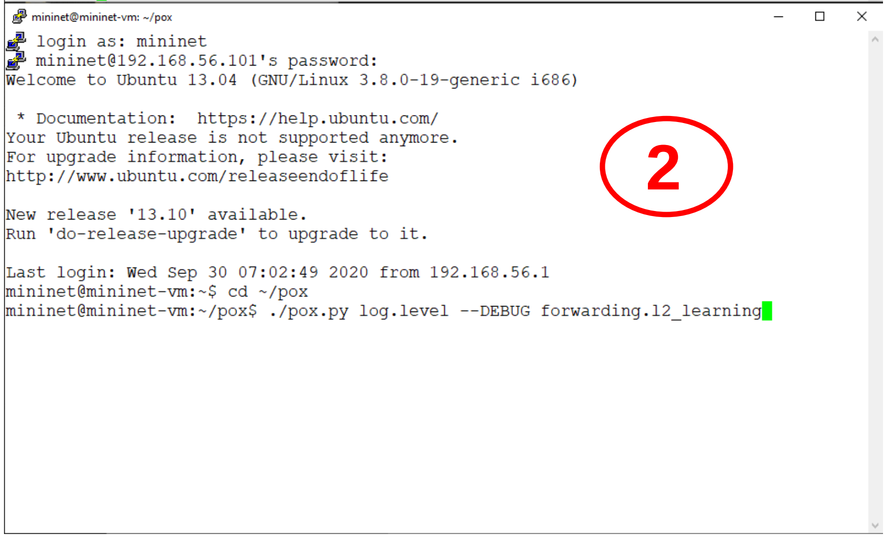
# Combine Mininet with POX

- Set up your network topology by typing:
  - sudo mn --topo single,2 --mac --switch ovsk --controller remote

- In a new SSH session, fire up the POX controller
  - ./pox.py log.level --DEBUG forwarding.l2_learning



```
 * Documentation:  https://help.ubuntu.com/
Your Ubuntu release is not supported anymore.
For upgrade information, please visit:
http://www.ubuntu.com/releaseendoflife

New release '13.10' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Sep 30 07:01:35 2020 from 192.168.56.1
mininet@mininet-vm:~$ sudo mn --topo single,2 --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet>
```

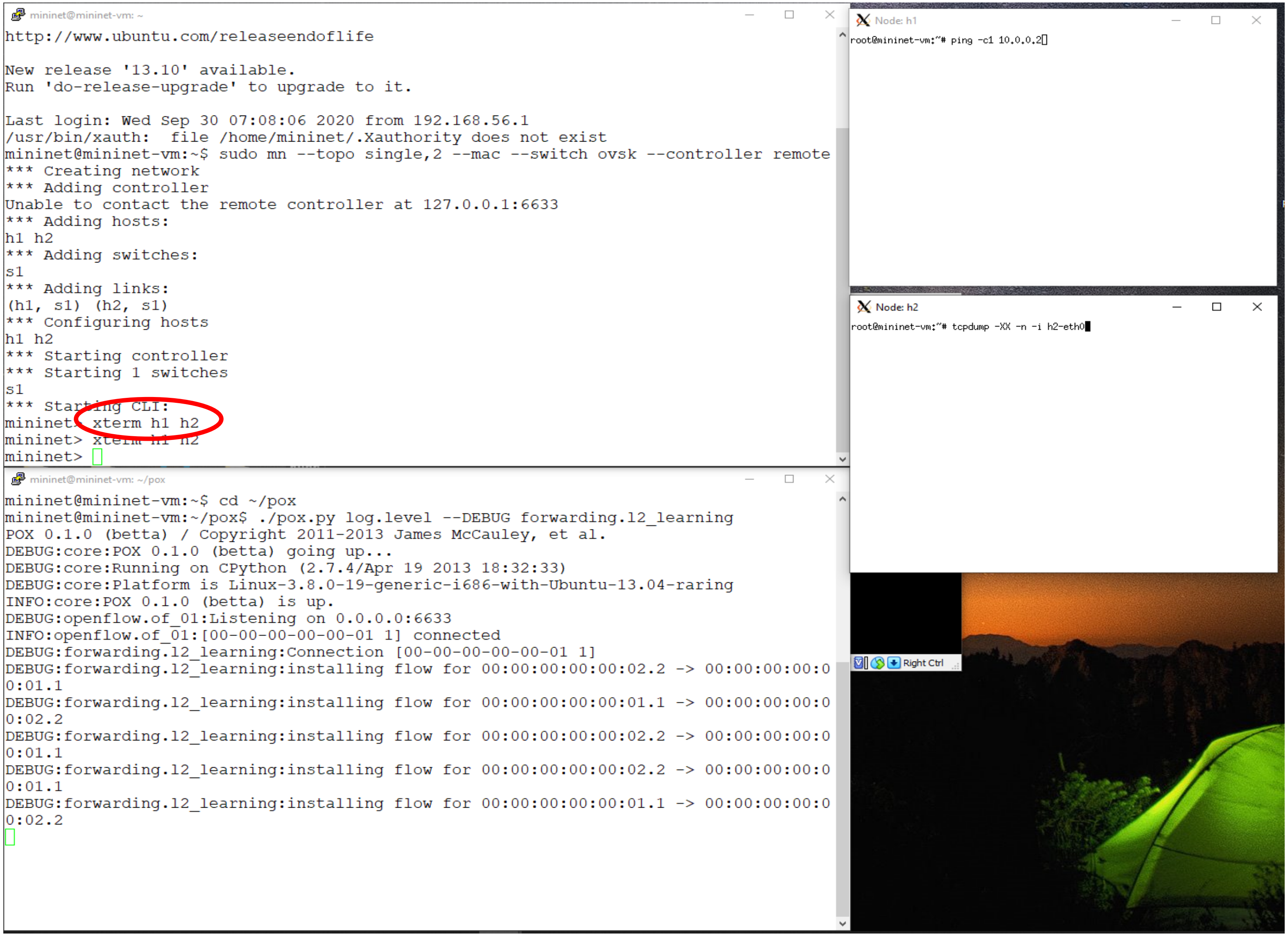


```
 login as: mininet
 mininet@192.168.56.101's password:
Welcome to Ubuntu 13.04 (GNU/Linux 3.8.0-19-generic i686)

 * Documentation:  https://help.ubuntu.com/
Your Ubuntu release is not supported anymore.
For upgrade information, please visit:
http://www.ubuntu.com/releaseendoflife

New release '13.10' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Sep 30 07:02:49 2020 from 192.168.56.1
mininet@mininet-vm:~$ cd ~/pox
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG forwarding.l2_learning
```

# Executing commands in hosts

- Xming Server is required for Windows users
- Hosts h1,…, hN are accessible by typing (when Mininet is running)
  - xterm h1 h2 … hN
- You can execute commands such as ping or tcpdump for debug purposes after executing xterm command in the xterm of a host (e.h h2)
  - ping -c1 10.0.0.3 (ping the host with IP 10.0.0.3)
  - tcpdump -XX -n -i h2-eth0 (use tcpdump for the interface h2-eth0)

# Executing commands in hosts



```
http://www.ubuntu.com/releaseendoflife

New release '13.10' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Sep 30 07:08:06 2020 from 192.168.56.1
/usr/bin/xauth:  file /home/mininet/.Xauthority does not exist
mininet@mininet-vm:~$ sudo mn --topo single,2 --mac --switch ovsk --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> xterm h1 h2
mininet> xterm h1 h2
mininet> 
```

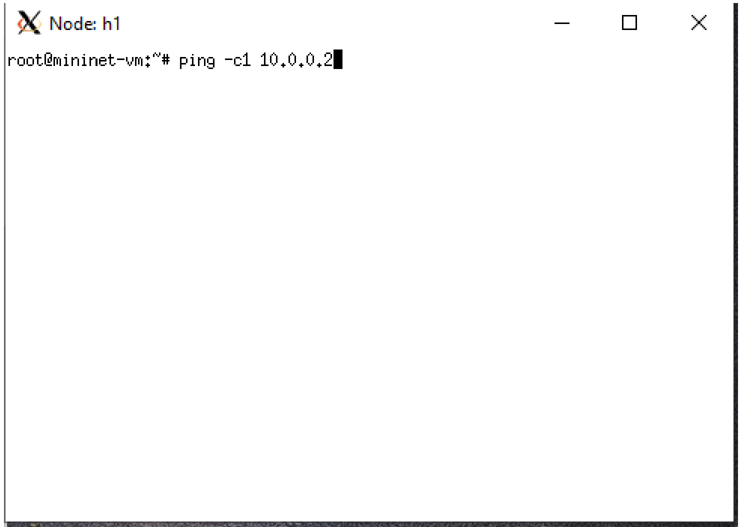
```
root@mininet-vm:~# ping -c1 10.0.0.2
```

```
root@mininet-vm:~# tcpdump -XX -n -i h2-eth0
```

```
mininet@mininet-vm:~$ cd ~/pox
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG forwarding.l2_learning
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.4/Apr 19 2013 18:32:33)
DEBUG:core:Platform is Linux-3.8.0-19-generic-i686-with-Ubuntu-13.04-raring
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 1]
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:0
0:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:0
0:02.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:0
0:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:0
0:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:0
0:02.2
```

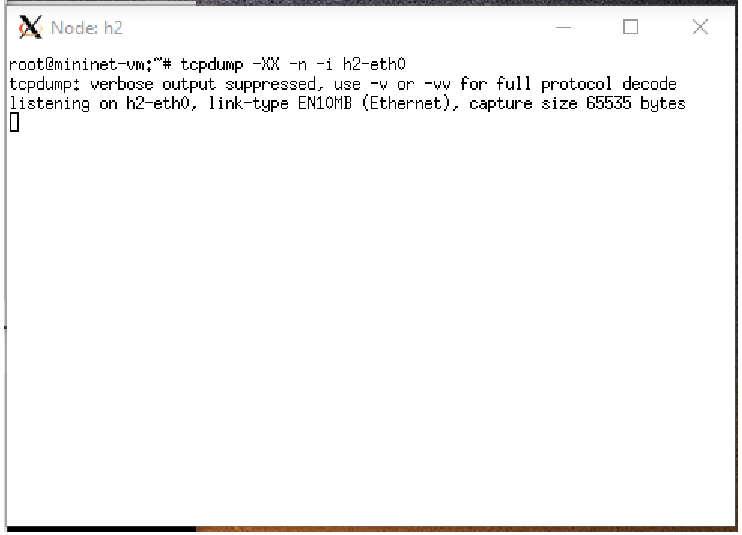
# Executing commands in hosts

Host h1 pings 10.0.0.2 (h2)



```
root@mininet-vm:~# ping -c1 10.0.0.2
```

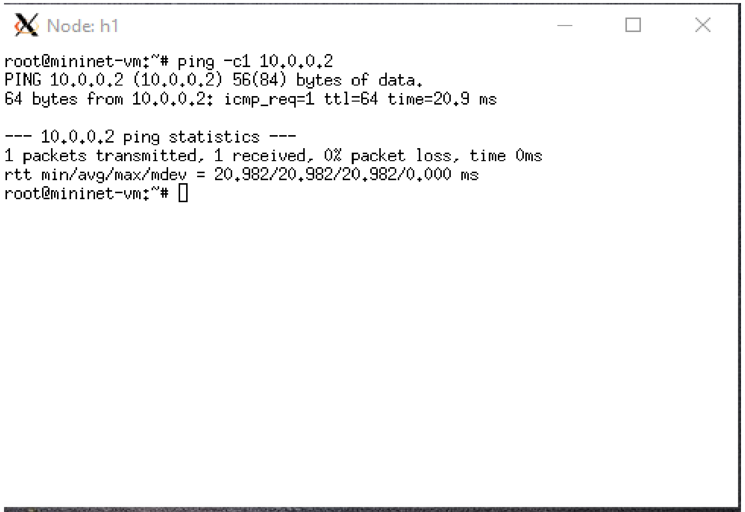
Host h2 uses tcpdump for traffic monitoring



```
root@mininet-vm:~# tcpdump -XX -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

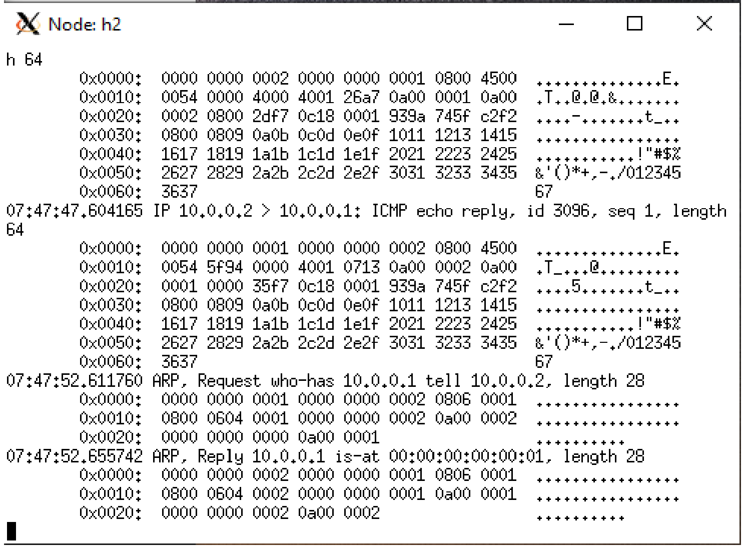
# Executing commands in hosts

Host h1 pings 10.0.0.2 (h2)



```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=20.9 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 20.982/20.982/20.982/0.000 ms
root@mininet-vm:~# 
```

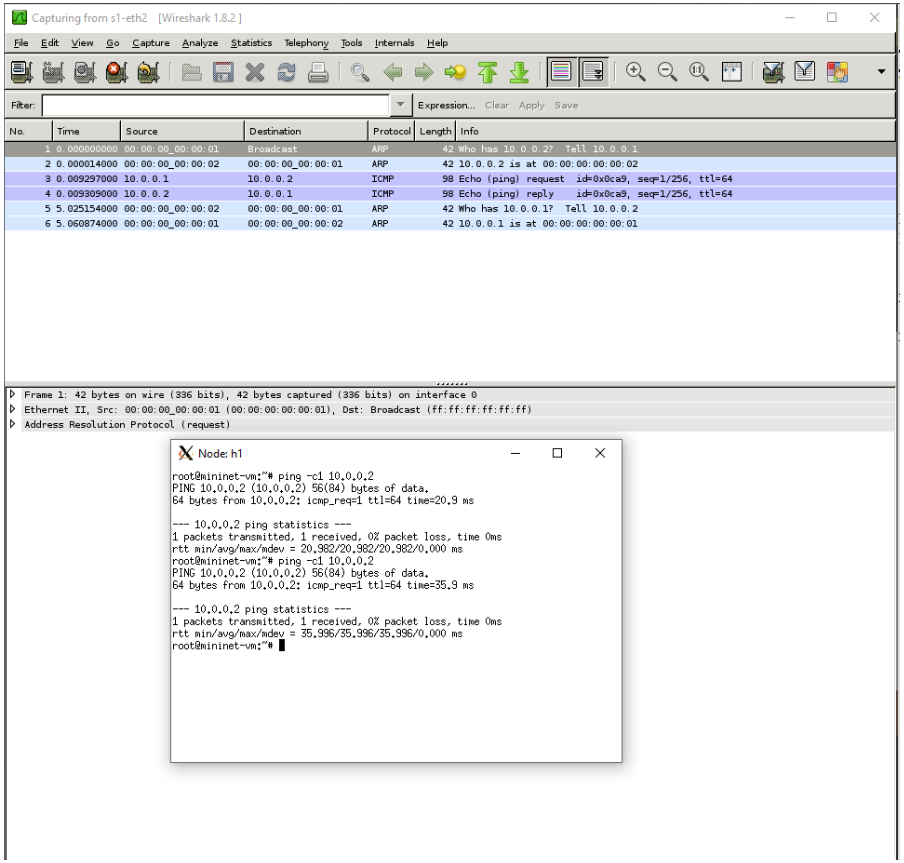
Host h2 captures the traffic using from host h1 using tcpdump



```
h 64
        0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  ..............E.
        0x0010:  0054 0000 4000 4001 26a7 0a00 0001 0a00  .T..@.@.&.......
        0x0020:  0002 0800 2df7 0c18 0001 939a 745f c2f2  ....-.......t_..
        0x0030:  0800 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .......... !"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
07:47:47.604165 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 3096, seq 1, length
64
        0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  ..............E.
        0x0010:  0054 5f94 0000 4001 0713 0a00 0002 0a00  .T_...@.........
        0x0020:  0001 0000 35f7 0c18 0001 939a 745f c2f2  ....5.......t_..
        0x0030:  0800 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .......... !"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
07:47:52.611760 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
        0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  ................
        0x0010:  0800 0604 0001 0000 0000 0002 0a00 0002  ................
        0x0020:  0000 0000 0000 0a00 0001                 ..........
07:47:52.655742 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
        0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  ................
        0x0010:  0800 0604 0002 0000 0000 0001 0a00 0001  ................
        0x0020:  0000 0000 0002 0a00 0002                 ..........
```

# Traffic monitoring

- Wireshark is also installed in your vm for debug/traffic monitoring purposes
- Wireshark can monitor all interfaces in the network
- In a new SSH session, please type:
  - **<u>sudo</u>** wireshark &



Capturing from s1-eth2 [Wireshark 1.8.2 ]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 00:00:00_00:00:01 | Broadcast | ARP | 42 | Who has 10.0.0.2? Tell 10.0.0.1 |
| 2 | 0.000014000 | 00:00:00_00:00:02 | 00:00:00_00:00:01 | ARP | 42 | 10.0.0.2 is at 00:00:00:00:00:02 |
| 3 | 0.009297000 | 10.0.0.1 | 10.0.0.2 | ICMP | 98 | Echo (ping) request id=0x0ca9, seq=1/256, ttl=64 |
| 4 | 0.009309000 | 10.0.0.2 | 10.0.0.1 | ICMP | 98 | Echo (ping) reply id=0x0ca9, seq=1/256, ttl=64 |
| 5 | 5.025154000 | 00:00:00_00:00:02 | 00:00:00_00:00:01 | ARP | 42 | Who has 10.0.0.1? Tell 10.0.0.2 |
| 6 | 5.060874000 | 00:00:00_00:00:01 | 00:00:00_00:00:02 | ARP | 42 | 10.0.0.1 is at 00:00:00:00:00:01 |

```
Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)
```

```
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=20.9 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 20.982/20.982/20.982/0.000 ms
root@mininet-vm:~# ping -c1 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=35.9 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 35.996/35.996/35.996/0.000 ms
root@mininet-vm:~# 
```

# References

- http://mininet.org/
- https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
- https://www.clear.rice.edu/comp529/www/papers/tutorial_4.pdf
- https://openflow.stanford.edu/display/ONL/POX+Wiki