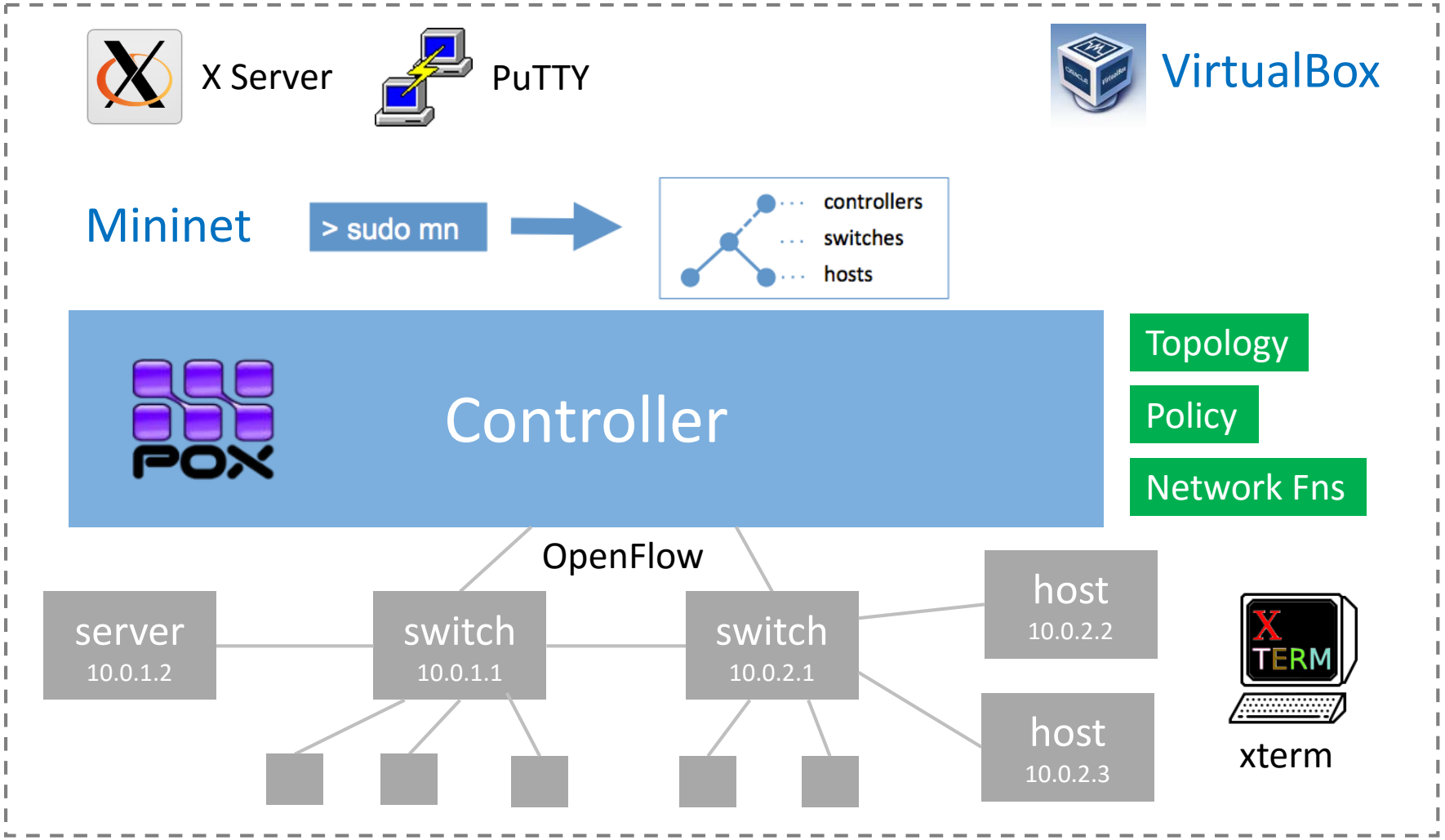
IERG5090: SDN Lab

Prepared by Jacky ZHAN

15 Mar, 2017

What is covered in this SDN Lab?



Overview

In this lab, you will learn

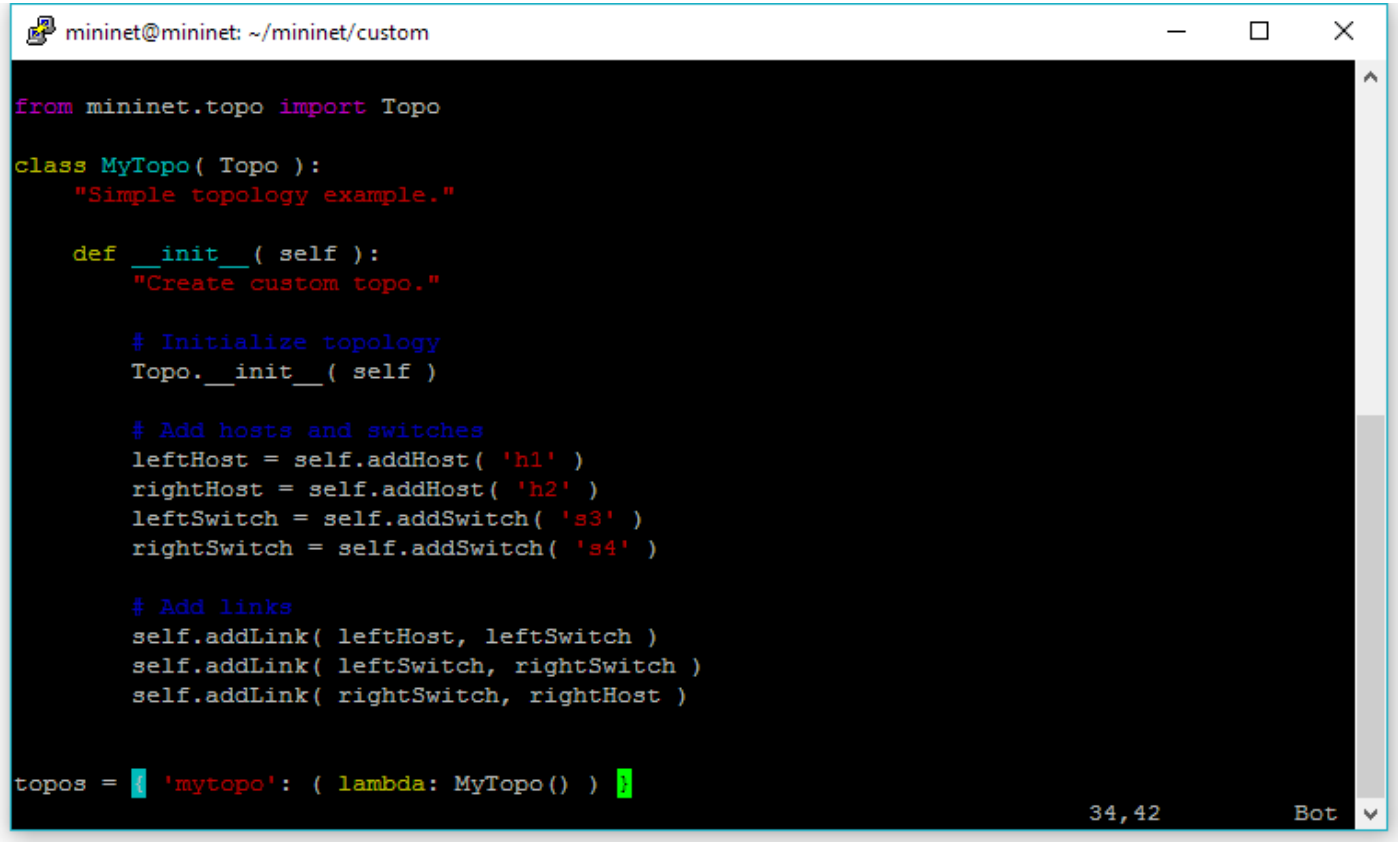
- how to set up a SDN emulation environment on your own laptop or PC; how to connect and access the environment from your laptop; simple examples of Mininet and some development tools;
- how to write and run custom topology in Mininet; the basic knowledge and APIs of the controller platform you choose (here we take POX as an example); how to implement network functions, such as hub, switch, and firewall, on the controller platform;
- the IP load balancer component of POX and its usage together with L2 learning switches; the problem of containing connection loop in network topology for SDN and how POX handles it via the spanning tree module; the load balancing via routing on multiple paths in SDN.

Environment Setup

- Set up Virtual Machine
- Connect and Access VM
- Start Mininet
- Related Development Tools
- Mininet Walkthrough

Mininet Custom Topology #1

- Create Topology Template as mytopo.py



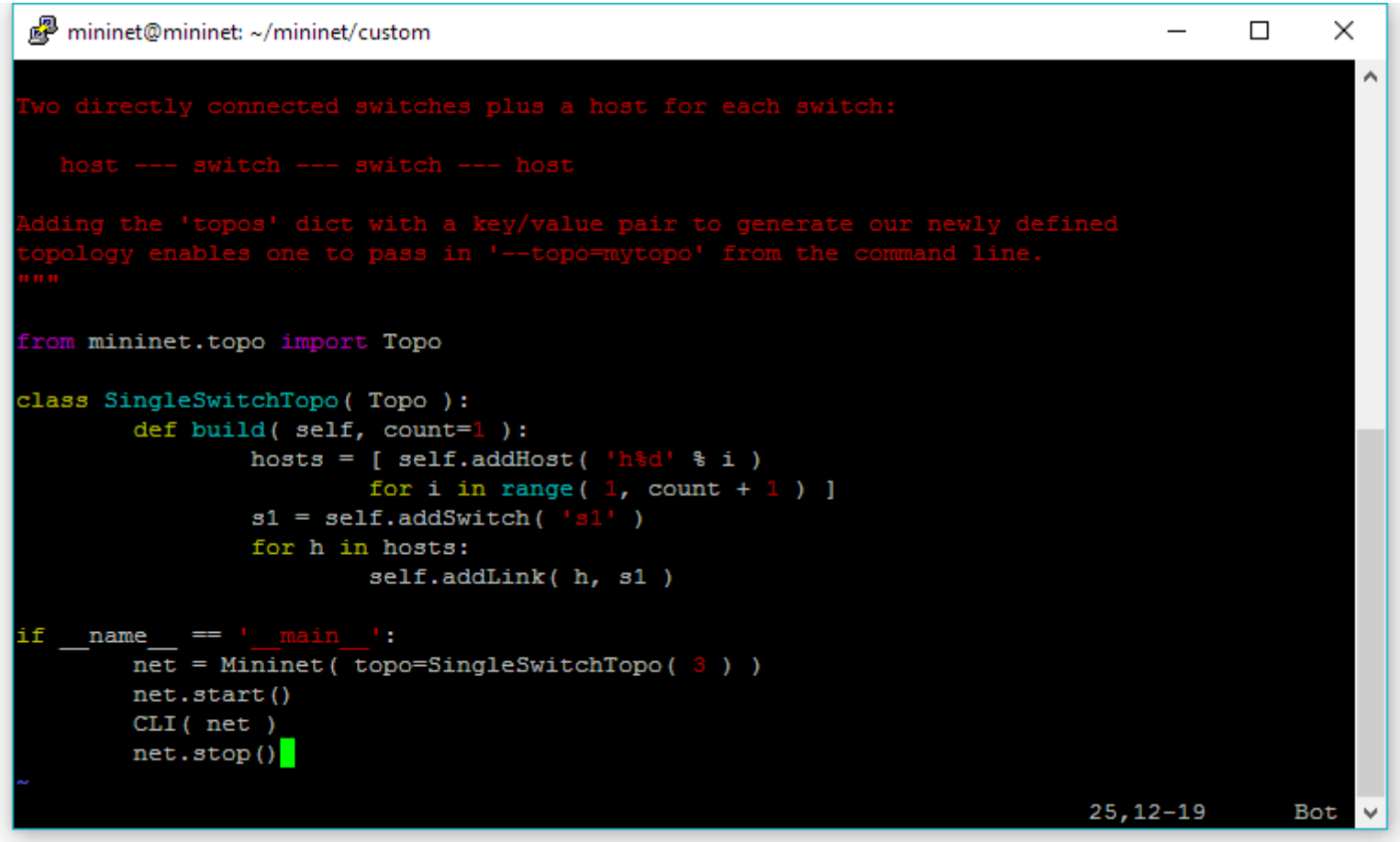
Mininet Custom Topology #1

- Run with sudo mn

  sudo mn --custom filepath --topo mytopo

Mininet Custom Topology #2

- Implement Mininet as Python script mynet.py



Mininet Custom Topology #2

- Run by executing Python

```
sudo python mynet.py
```

Update of Lab Task

- No need to run --test lab2test via sudo mn

Custom Topology

Apart from the built-in topologies and self-contained regression tests, Mininet supports customization of both test and topology in Python, see below:

```
# filename: mytopo.py
class MyTopo( Topo ):
    def build( self, ...):
def myTest( net ):
...
topos = { 'mytopo': MyTopo }
tests = { 'mytest': myTest }
```

The above example adds the *MyTopo* class to the topos dictionary and allows you to run the *myTest*. You could specify to use *MyTopo* and run *myTest* using the $--$*custom* to include the file following *sudomn*, see below:

```
$ sudo mn --custom mytopo.py --topo mytopo,3 --test mytest
```

Implementation of lab2Test

To verify your design of network topology, implement a test with name *lab2Test* to do the following tasks:

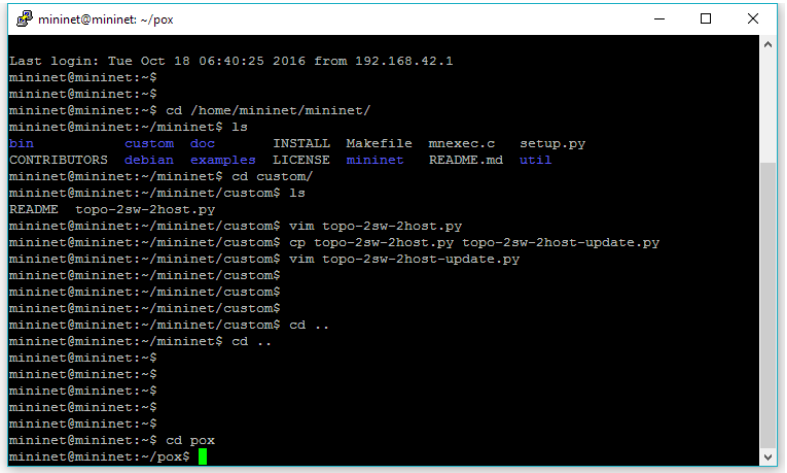
1. Dump all nodes and host connections;
2. Test the network connectivity;
3. Conduct iperf for all node pairs.

- Looking for methods in Mininet Python API
  - http://mininet.org/api/classmininet_1_1net_1_1Mininet.html
  - example:

```
def mininet.net.Mininet.iperf (  self,
                                 hosts = None,
                                 l4Type = 'TCP',
                                 udpBw = '10M',
                                 fmt = None,
                                 seconds = 5,
                                 port = 5001
                               )
```

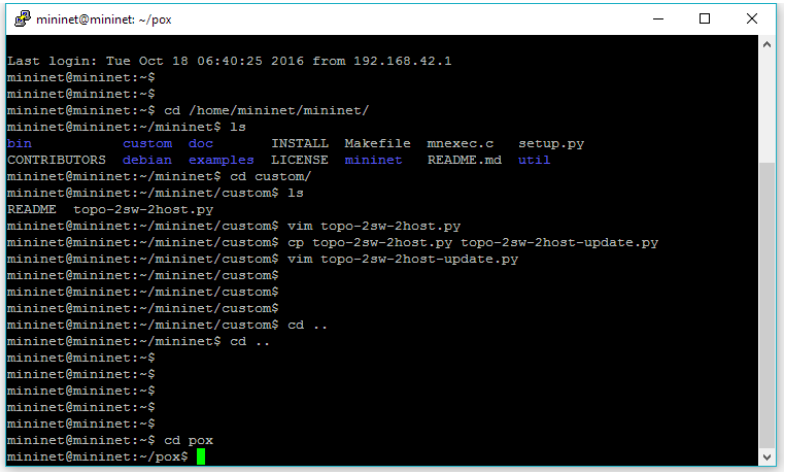
POX as Remote Controller

PuTTY Session 1



127.0.0.1 : 6633

PuTTY Session 2



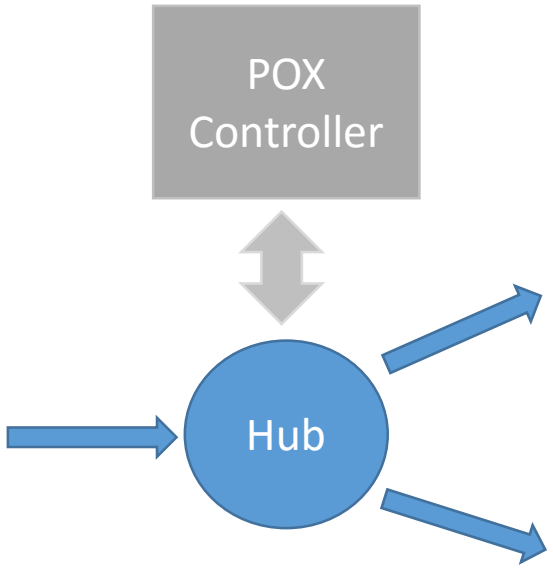
Run Mininet and enter CLI for conducting tests

Run POX as the Remote Controller

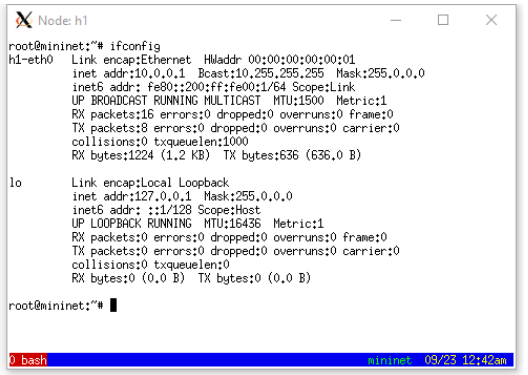
Hub Behavior

h1 xterm console

Conduct ping test to h2

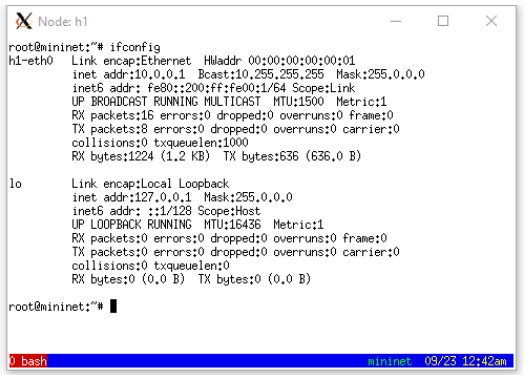


h2 xterm console



Run tcpdump to capture traffic

h3 xterm console



Run tcpdump to capture traffic

h2 and h3 receive the exact **Same Traffic**

Switch Behavior

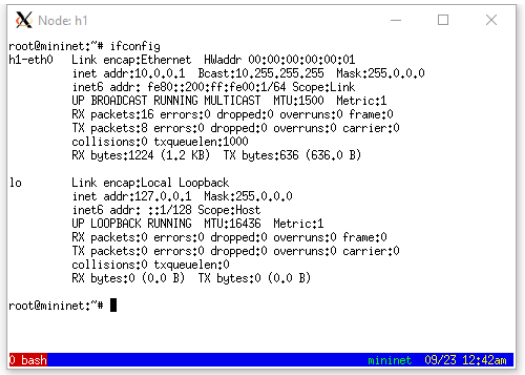
h1 xterm console

Conduct ping test to h2

POX Controller

Switch

h2 xterm console



Run tcpdump to capture traffic

h3 xterm console

Run tcpdump to capture traffic

Only

h2 will receive the traffic

Implementing Switch

- Look into the of_tutorial.py file in pox/pox/misc
  - create a copy of of_tutorial.py
  - implement the logic of act_like_switch()

```
def act_like_switch (self, packet, packet_in):
  """
  Implement switch-like behavior.
  """
```

  - change to evoke act_like_switch() in _handle_PacketIn()

```
  # Comment out the following line and uncomment the one after
  # when starting the exercise.
  self.act_like_hub(packet, packet_in)
  #self.act_like_switch(packet, packet_in)
```

Implementing Firewall

- Create a copy of your implementation of switch
  - implement the blocking logic in _handle_PacketIn()

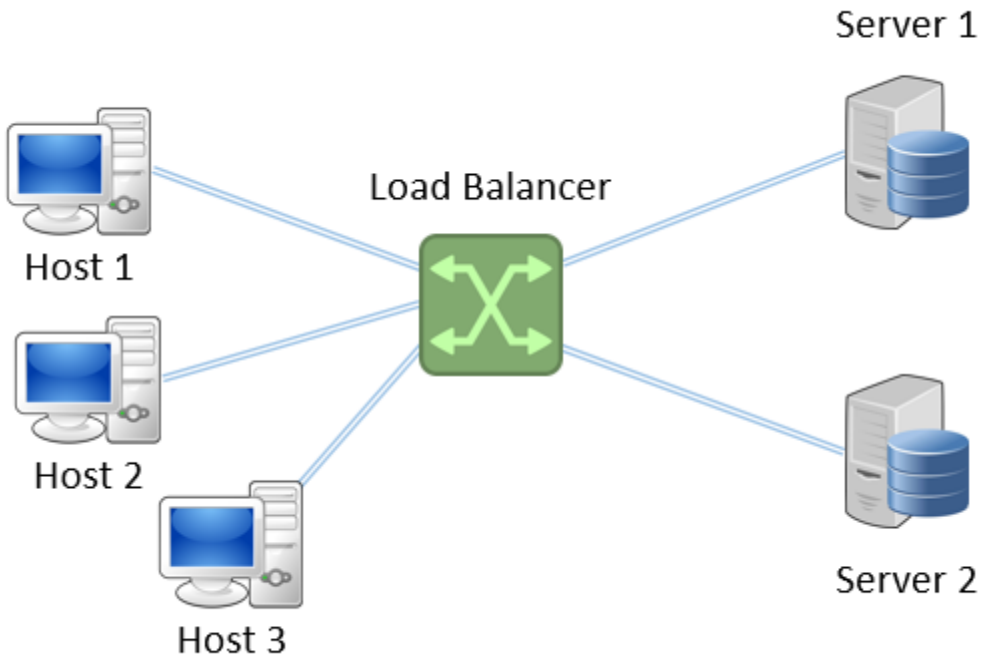
```
def _handle_PacketIn (self, event):
  """
  Handles packet in messages from the switch.
  """

  packet = event.parsed # This is the parsed packet data.
  if not packet.parsed:
    log.warning("Ignoring incomplete packet")
    return

  packet_in = event.ofp # The actual ofp_packet_in message.
```

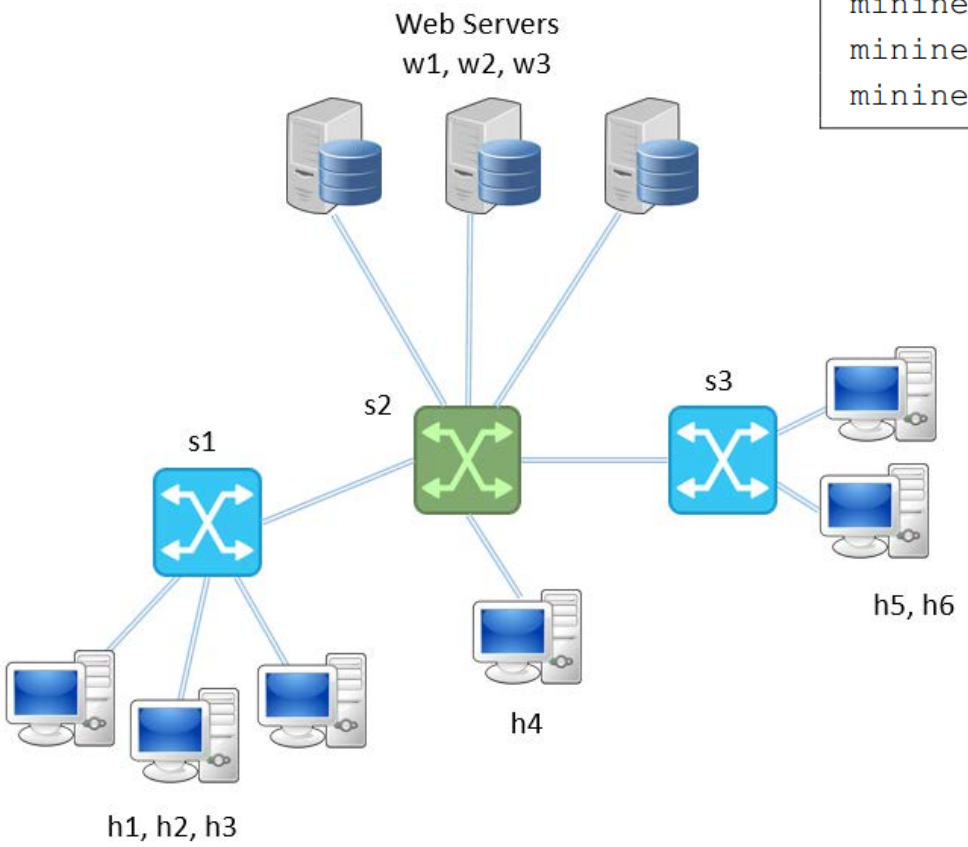
IP Load Balancer

```
mininet@mininet:~$ sudo mn --topo single,5 --controller remote
```




```
mininet@mininet:~/pox$ ./pox.py misc.ip_loadbalancer --ip=10.0.1.1
--servers=10.0.0.1,10.0.0.2
```

IP Load Balancer with Learning Switches



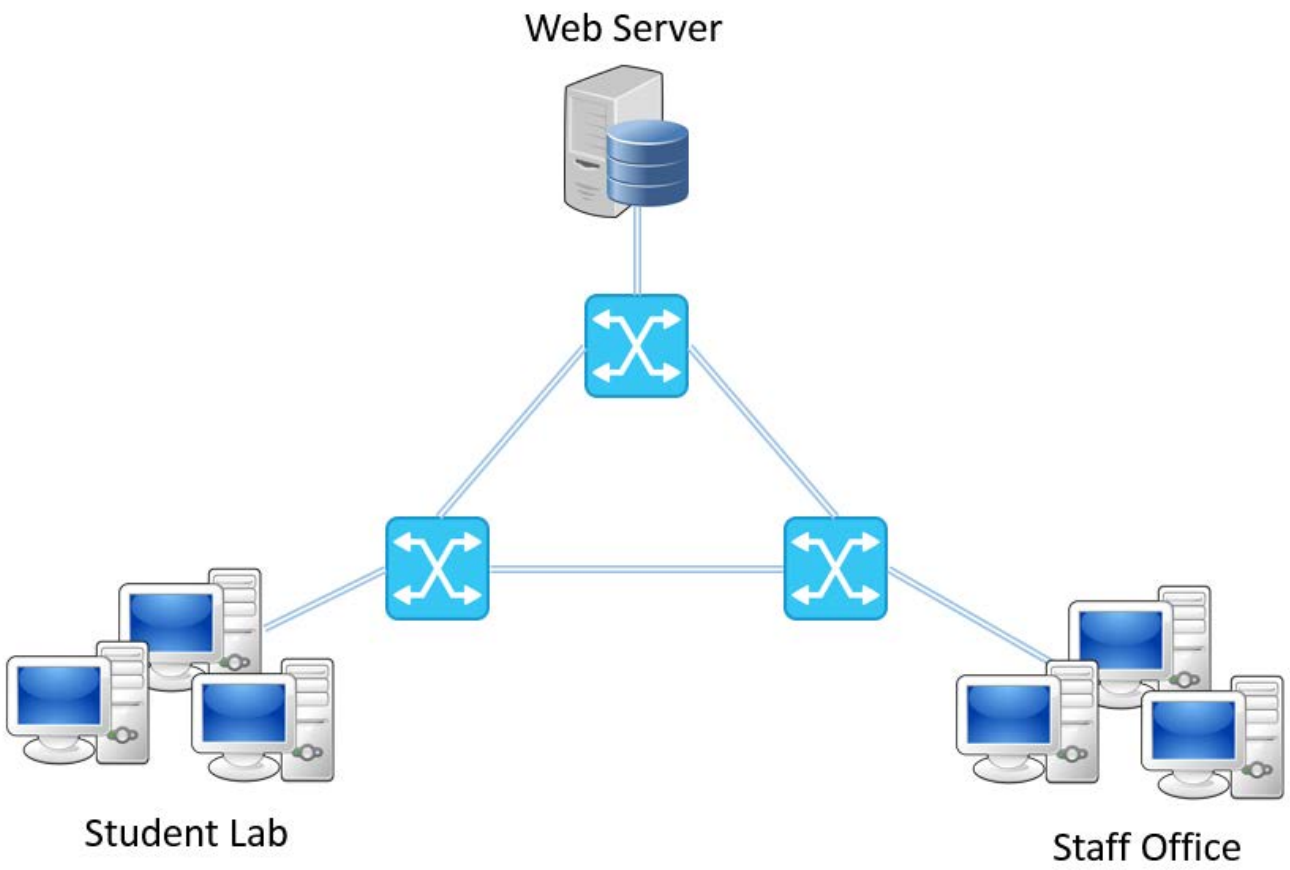

```
# remeber to backup you own files implemented under ./pox directory
mininet@mininet:~/pox$ sudo git checkout -b dart
mininet@mininet:~/pox$ sudo git clean -fd
mininet@mininet:~/pox$ sudo git pull origin dart
```

ext/selective_switch.py

```
1  """
2  More or less just l2_learning except it ignores a particular switch
3  """
4  from pox.core import core
5  from pox.lib.util import str_to_dpid
6  from pox.forwarding.l2_learning import LearningSwitch
7
8
9  def launch (ignore_dpid):
10   ignore_dpid = str_to_dpid(ignore_dpid)
11
12   def _handle_ConnectionUp (event):
13     if event.dpid != ignore_dpid:
14       core.getLogger().info("Connection %s" % (event.connection,))
15       LearningSwitch(event.connection, False)
16
17   core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
```

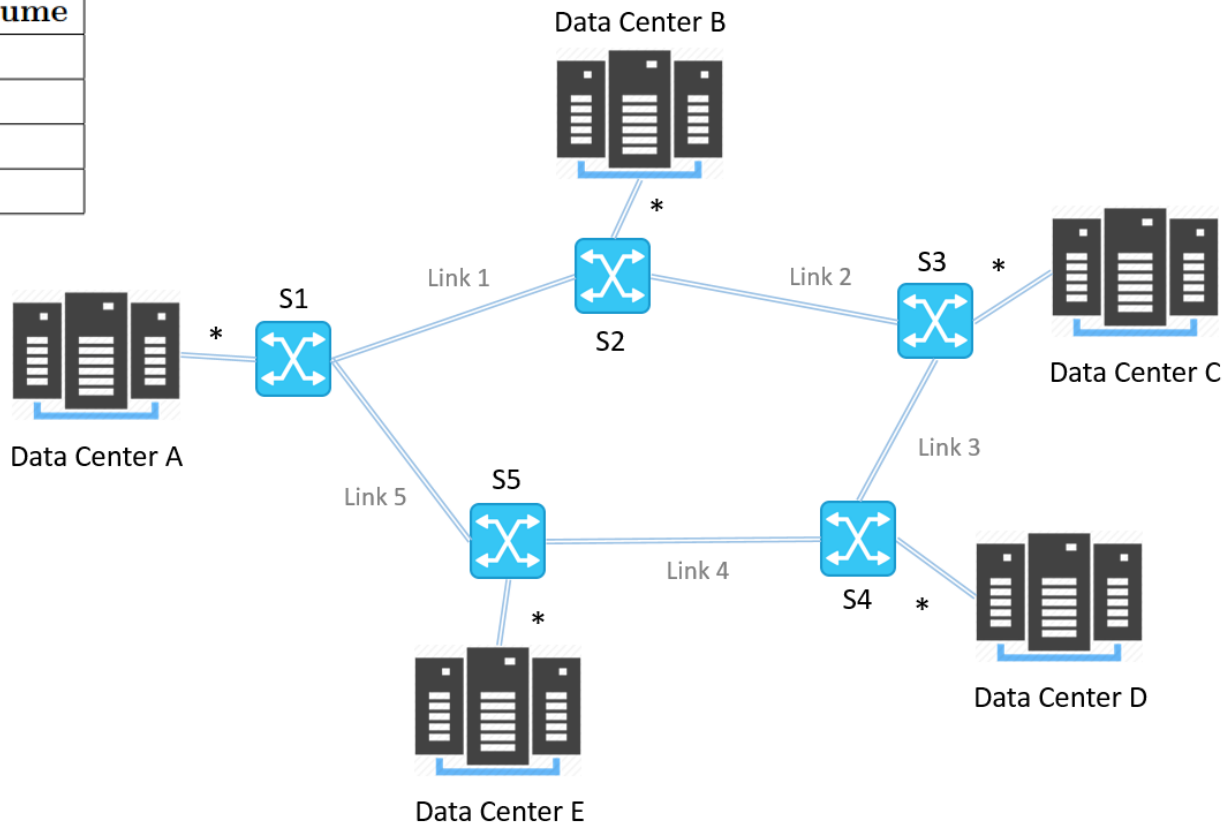
```
./pox.py misc.ip_loadbalancer --ip=10.0.1.1 --servers=10.0.0.1,10.0.0.2,..
--dpid=2 selective_switch --ignore-dpid=2
```

Load Balancing via Path Routing



Path Routing for Data Sync Tasks

Task	From	To	Data Volume
Task 1	A	E	2 GB
Task 2	B	E	1 GB
Task 3	B	C	3 GB
Task 4	A	C	2 GB



Step 1: Create Custom Topology

Link	Bandwidth	Delay	Loss
*	1 Gbps	20 us	0%
Link 1	50 Mbps	1 ms	1%
Link 2	40 Mbps	2 ms	1%
Link 3	80 Mbps	1 ms	1%
Link 4	50 Mbps	2 ms	1%
Link 5	50 Mbps	1 ms	1%

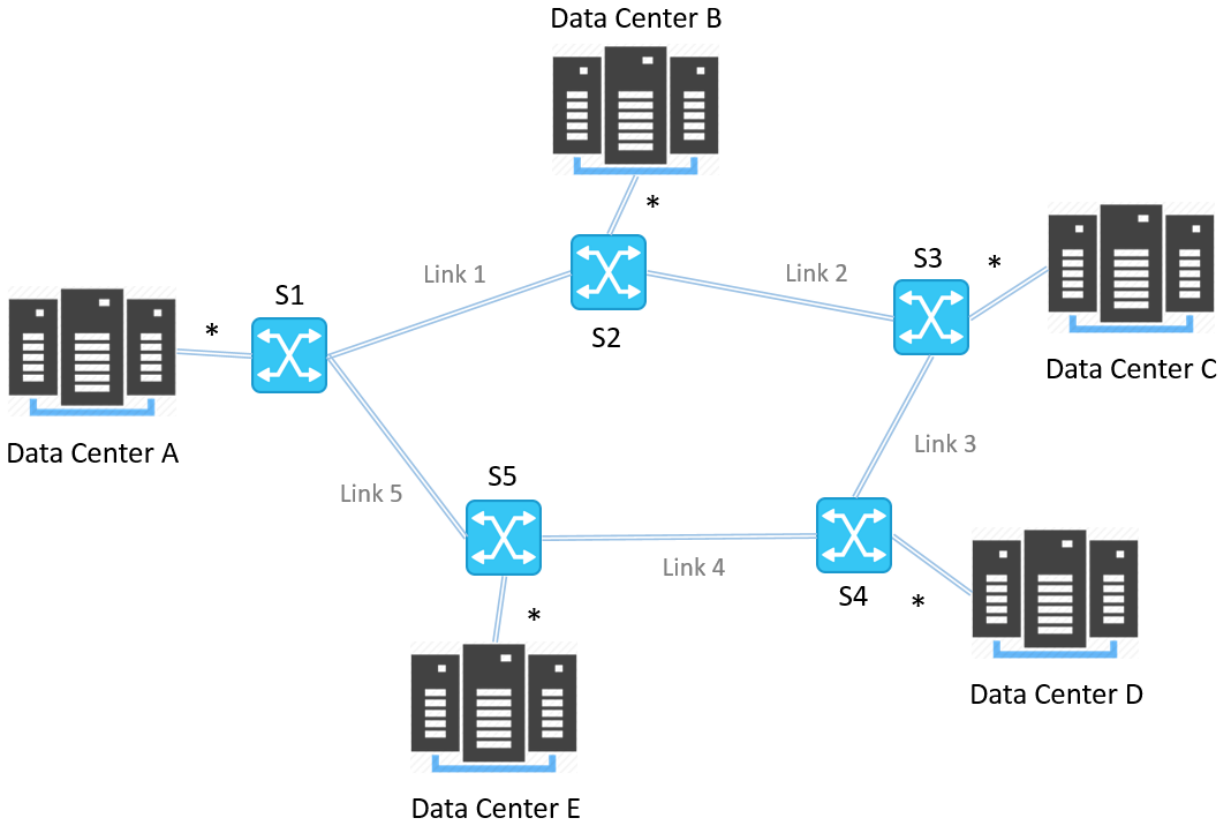
datacenter.py

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.cli import CLI

class Lab3Topo( Topo ):
        def __init__(self):

topos = { 'datacenter': (lambda: Lab3Topo()) }

if __name__ == '__main__':
        net = Mininet( topo=Lab3Topo() )
        net.start()
        CLI( net )
        net.stop()
```

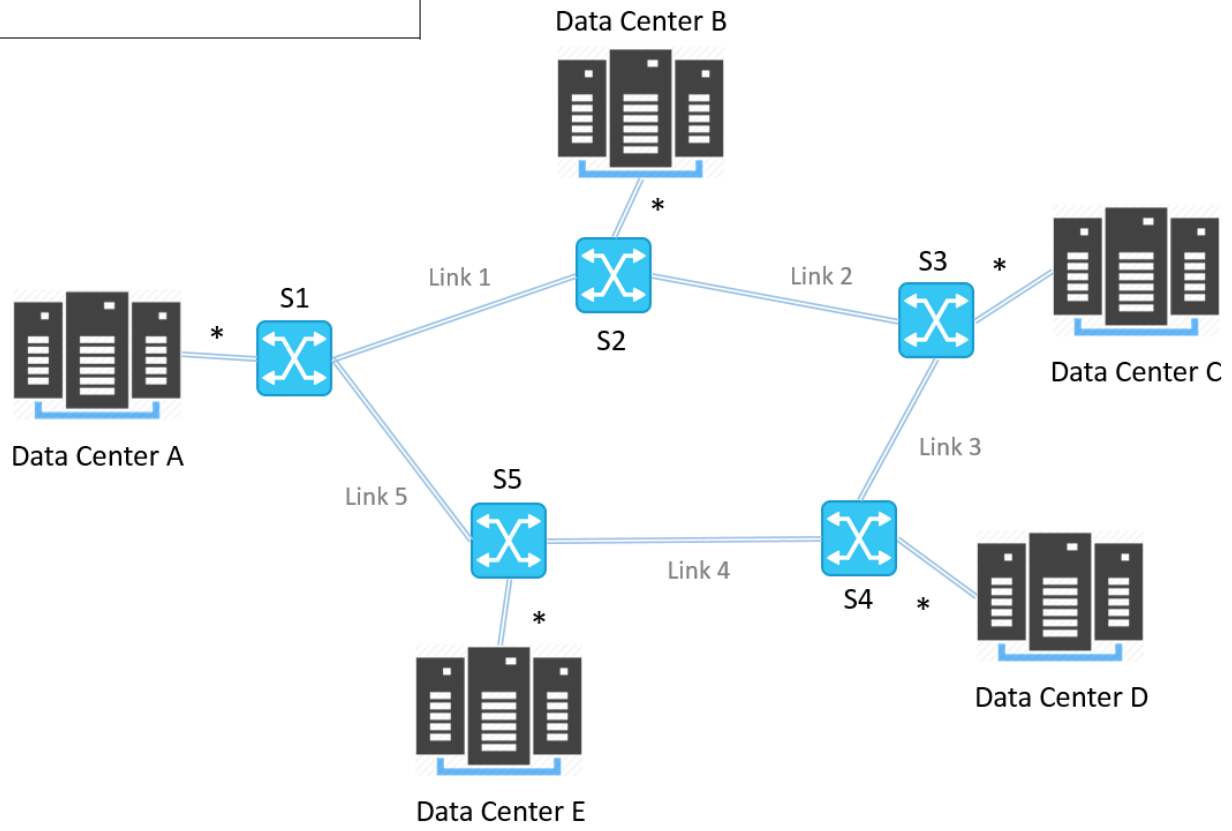


Step 2.1: Retrieve Connection Info. at Mininet

```
mininet@mininet:~$ sudo mn --custom datacenter.py --topo datacenter
--mac --controller remote --link tc
```

```
mininet> net
c0
s1 lo:  s1-eth1:ha-eth0 s1-eth2:s2-eth2 s1-eth3:s5-eth3
s2 lo:  s2-eth1:hb-eth0 s2-eth2:s1-eth2 s2-eth3:s3-eth2
s3 lo:  s3-eth1:hc-eth0 s3-eth2:s2-eth3 s3-eth3:s4-eth2
s4 lo:  s4-eth1:hd-eth0 s4-eth2:s3-eth3 s4-eth3:s5-eth2
s5 lo:  s5-eth1:he-eth0 s5-eth2:s4-eth3 s5-eth3:s1-eth3
ha ha-eth0:s1-eth1
hb hb-eth0:s2-eth1
hc hc-eth0:s3-eth1
hd hd-eth0:s4-eth1
he he-eth0:s5-eth1
```

Node	To	Interface	Port
s1	ha	s1-eth1	1
s1	s2	s2-eth2	2
…			
…			



Step 2.2: Confirm Port Info. at POX

of_tutorial_lab3.py

```
def _handle_PacketIn (self, event):
  """
  Handles packet in messages from the switch.
  """

  packet = event.parsed # This is the parsed packet data.
  if not packet.parsed:
    log.warning("Ignoring incomplete packet")
    return

  # the code to display the connection ports
  print "-------------"
  print "At switch with ID %s" % event.connection.dpid
  for p in event.connection.features.ports:
    print "port %s with name %s" % (p.port_no, p.name)

  packet_in = event.ofp # The actual ofp_packet_in message.

  # Comment out the following line and uncomment the one after
  # when starting the exercise.
  #self.act_like_hub(packet, packet_in)
  #self.act_like_switch(packet, packet_in)
```

start the POX controller

```
mininet@mininet:~/pox$ ./pox.py log.level --DEBUG misc.of_tutorial_lab3
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.3.0 (dart) going up...
DEBUG:core:Running on CPython (2.7.3/Apr 10 2013 05:46:21)
DEBUG:core:Platform is Linux-3.5.0-23-generic-i686-with-Ubuntu-12.04-precise
INFO:core:POX 0.3.0 (dart) is up.
```

conduct ping tests

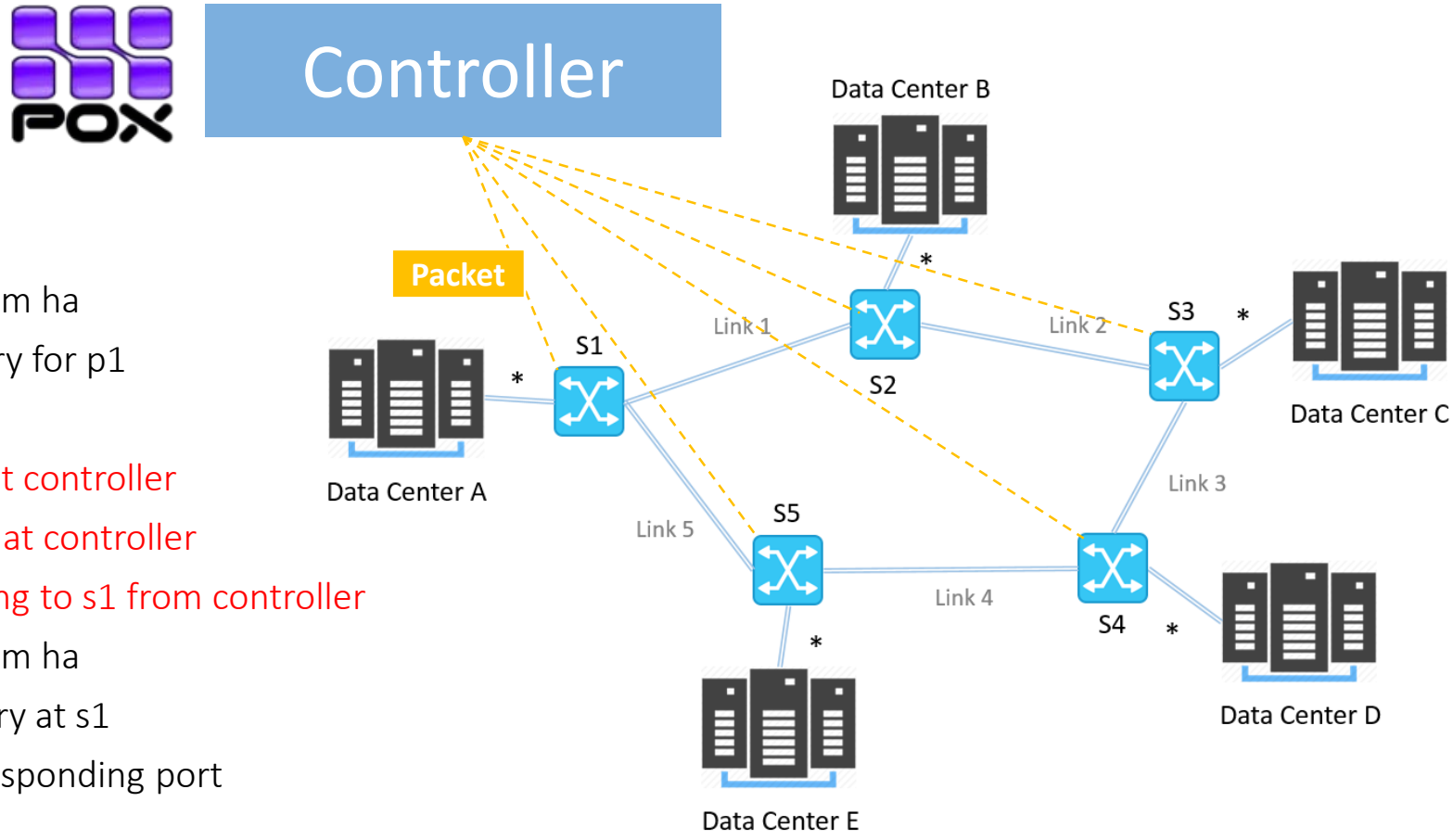
```
mininet> ha ping -c1 hb
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

output at the POX controller

```
-------------
At switch with ID 1
port 3 with name s1-eth3
port 2 with name s1-eth2
port 65534 with name s1
port 1 with name s1-eth1
```

Step 3: Install Flow Entry at Switches



e.g. ha ping hb

1) packet p1 approaches s1 from ha
2) s1 has no matched flow entry for p1
3) s1 sends p1 to controller
4) trigger _handle_PacketIn() at controller
5) create all flow entries for s1 at controller
6) send flow entries for installing to s1 from controller
7) packet p2 approaches s1 from ha
8) p2 is matched with flow entry at s1
9) p2 is sent to s2 via the corresponding port

Step 3.1: Logic at Controller

general function to install a flow entry

identify the switch id and install all entries for the switch

```
from pox.core import core
from pox.lib.addresses import EthAddr
import pox.openflow.libopenflow_01 as of

log = core.getLogger()


class Tutorial (object):
  """
  A Tutorial object is created for each switch that connects.
  A Connection object for that switch is passed to the __init__ function.
  """
  def __init__ (self, connection):

  def resend_packet (self, packet_in, out_port):

  def install_flow_entry (self, src, dst, port_no):

  def act_in_lab3 (self, switch_id):

  def _handle_PacketIn (self, event):

def launch ():
  """
  Starts the component
  """
  def start_switch (event):
    log.debug("Controlling %s" % (event.connection,))
    Tutorial(event.connection)
  core.openflow.addListenerByName("ConnectionUp", start_switch)
```

Step 3.2: Install Flow Entry Function

```
def install_flow_entry (self, src, dst, port_no):
  log.debug("Flow entry from %s to %s at port %s" % (src, dst, port_no))
  # creat a new flow_message (0-4)
  msg = of.ofp_flow_mod(command=0)
  msg.priority = 3

  # set src and dst MAC address of matching
  msg.match.dl_type = 0x800
  msg.match.nw_src = src
  msg.match.nw_dst = dst

  # forward the packet to certain port X at s1 to s2
  msg.actions.append(of.ofp_action_output(port=port_no))

  # send out the message
  self.connection.send(msg)

  # add the arp flow entry
  msg.match.dl_type = 0x806
  self.connection.send(msg)
```

for the purpose of ping tests

Protocol	Dependency	Name	Match Field
Port	none	Port ID	in_port
Ethernet	none	Source MAC	dl_src
		Destination MAC	dl_dst
		Type/Length	dl_type
		VLAN ID	dl_vlan
		VLAN Priority	dl_vlan_pcp
⚠ ARP	dl_type = 0x0806	Opcode	nw_proto
		Sender Protocol Address	nw_src
		Target Protocol Address	nw_dst
IPv4	dl_type = 0x0800	Type of Service	nw_tos
		Protocol	nw_proto
		Source Address	nw_src
		Destination Address	nw_dst
TCP	nw_proto = 6	Source Port	tp_src
		Destination Port	tp_dst

Step 3.3: Identify Switch and Install Entries

_handle_Packet_in

```
self.act_in_lab3(event.connection.dpid)
```

act_in_lab3(self, switch_id)

```
    if switch_id == 1:
    elif switch_id == 2:
    elif switch_id == 3:
    elif switch_id == 4:
    elif switch_id == 5:
```

install flow entries for all possibilities of src, dst pairs at the corresponding switch

```
if switch_id == 1:
  self.install_flow_entry(ips['ha'], ips['he'], ports['s1-s5'])
  self.install_flow_entry(ips['he'], ips['ha'], ports['s1-ha'])
  self.install_flow_entry(ips['hb'], ips['he'], ports['s1-s5'])
  self.install_flow_entry(ips['he'], ips['hb'], ports['s1-s2'])
  self.install_flow_entry(ips['ha'], ips['hc'], ports['s1-s2'])
  self.install_flow_entry(ips['hc'], ips['ha'], ports['s1-ha'])
```

Task	Path
Task 1	A - S1 - S5 - E
Task 2	B - S2 - S1 - S5 - E
Task 3	B - S2 - S3 - C
Task 4	A - S1 - S2 - S3 - C

build the IP and port dictionary according to you connection table

Step 4: Validate the Installed Routing Rules

Conduct following ping tests to trigger the flow entry installation for all switches at controller

- *ha ping –c1 hb*
- *hb ping –c1 hc*
- *hc ping –c1 hd*
- *hd ping –c1 he*
- *he ping –c1 ha*

It is okay if the ping tests fail

Task	From	To	Data Volume
Task 1	A	E	2 GB
Task 2	B	E	1 GB
Task 3	B	C	3 GB
Task 4	A	C	2 GB

Test the connectivity for each tasks then

- *ha ping –c1 he*
- *he ping –c1 ha*
- *hb ping –c1 he*
- *he ping –c1 hb*
- *hb ping –c1 hc*
- *hc ping –c1 hb*
- *ha ping –c1 hc*
- *hc ping –c1 ha*

The ping tests between other host pairs will fail

Step 5: Traffic Generation

at xterm ha and hb

```
dd if=/dev/urandom of=traffic bs=10M count=10
python -m SimpleHTTPServer 80
```

create a 100MB file

setup a HTTP server

for multi-thread HTTP server

python MultithreadedSimpleHTTPServer.py 80

download_ha.py

```
import urllib
import sys
import time

def download(count=1):
        start = time.time()
        for i in range(count):
                urllib.urlretrieve("http://10.0.0.1/traffic")
        print "Download %s00MB from A takes %s seconds" % (count, time.time() - start)

if __name__ == '__main__':
        count = int(sys.argv[1])
        download(count)
```

create a similar one download_hb.py

Run only Task 1
at xterm he

python download_ha.py 20

Run only Task 1 & 2 at the same time
at xterm he

python download_ha.py 20 > he_ha.txt & python download_hb.py 10 > he_hb.txt &

outputs will be at .txt files

Step 6: Run Experiments

Run Task 1 & 2 at xterm he

```
python download_ha.py 20 > he_ha.txt & python download_hb.py 10 > he_hb.txt &
```

Run Task 3 & 4 at xterm hc

```
python download_hb.py 30 > hc_hb.txt & python download_ha.py 20 > hc_ha.txt &
```

Check the running time at .txt files

Conduct the experiment for the given paths

Task	Path
Task 1	A - S1 - S5 - E
Task 2	B - S2 - S1 - S5 - E
Task 3	B - S2 - S3 - C
Task 4	A - S1 - S2 - S3 - C

Conduct the experiment for the your planning of paths

Q&A