Identify a “WHOIS” Solution

PRIYANK SHAH

MEETIKA SHARMA

SHUBHAM KOTHARI

PARTH SHIROLAWALA

Environment

- Controller: POX Controller
- Topology: Linear Topology
- Tool: Mininet
- Language: Python
- Requirements: Python2.7

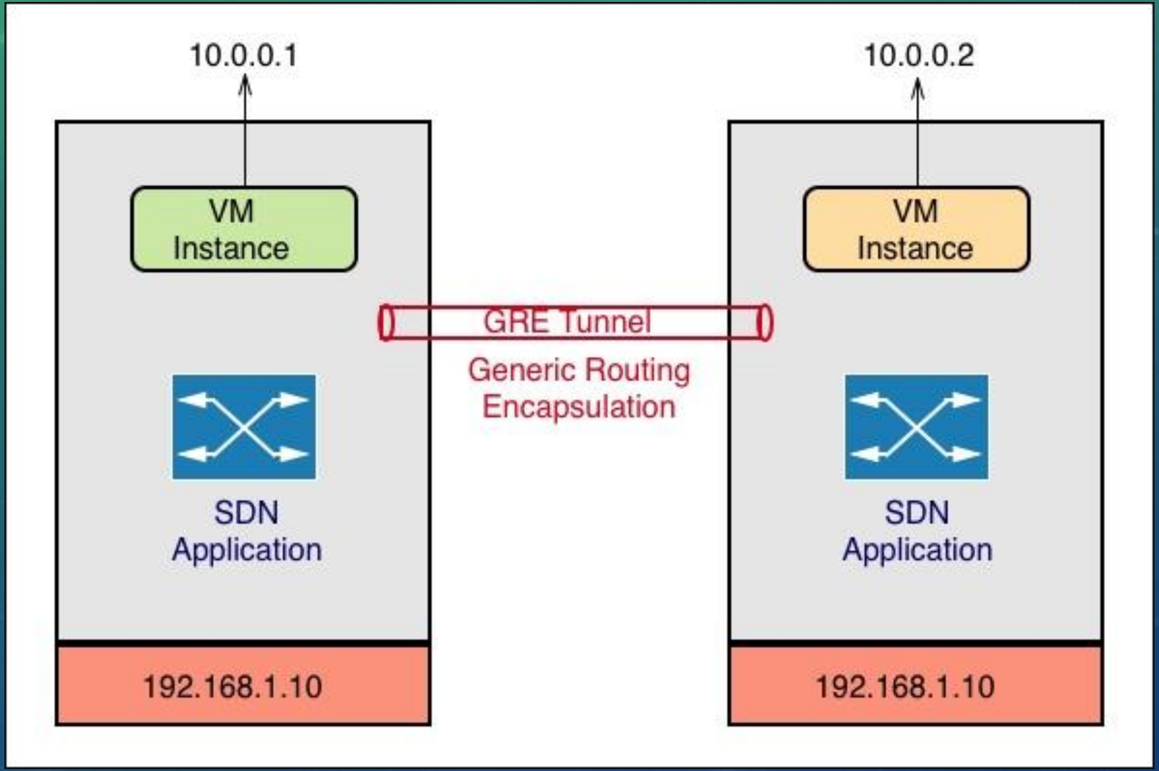
“WHOIS” Lookup Features

- WHOIS database - listing of registered domains
- WHOIS data – to identify and fix problems
- To combat spam or fraud
- To hold accountable and for legal purposes

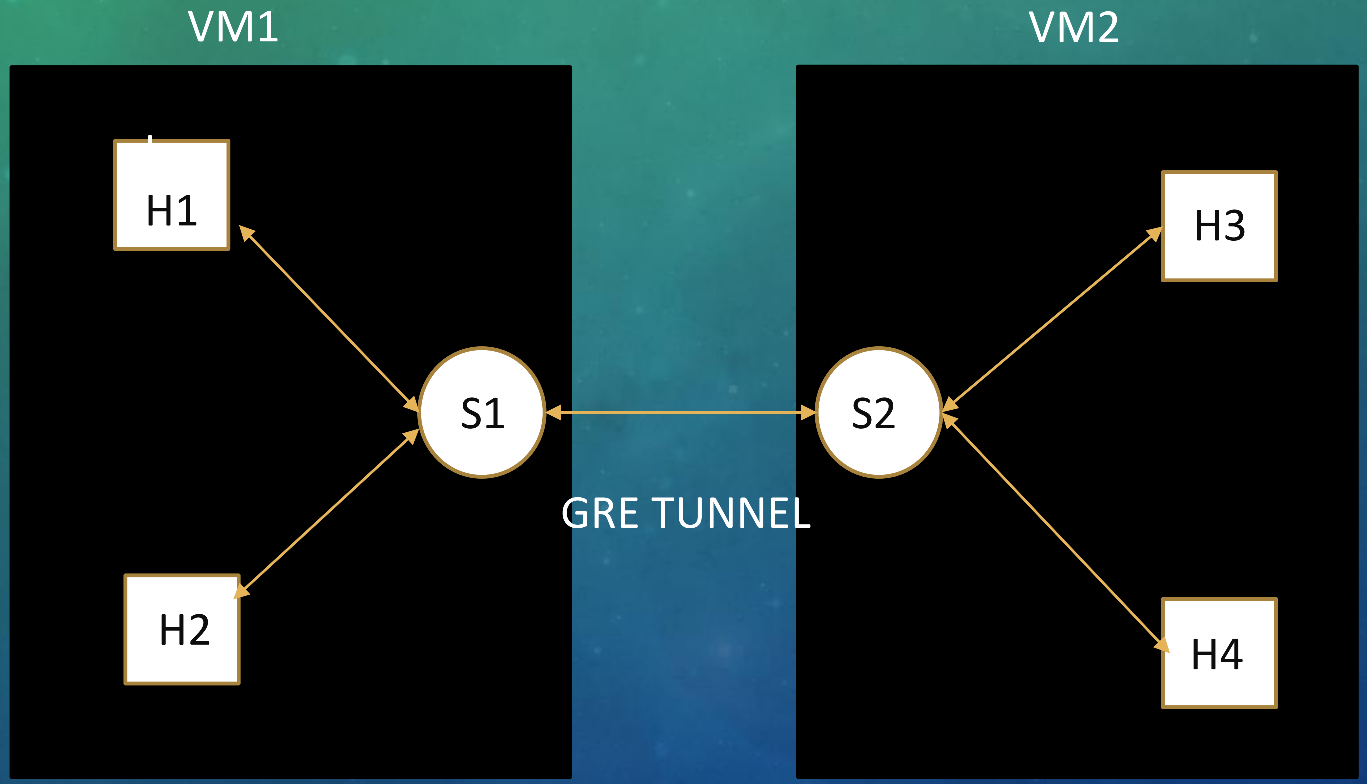
GRE Tunnel

- Generic Routing Encapsulation (GRE) is a tunneling protocol developed by Cisco Systems.
- Encapsulates packets to route other protocols over IP networks.
- To create point to point connections between two networks.

Architecture

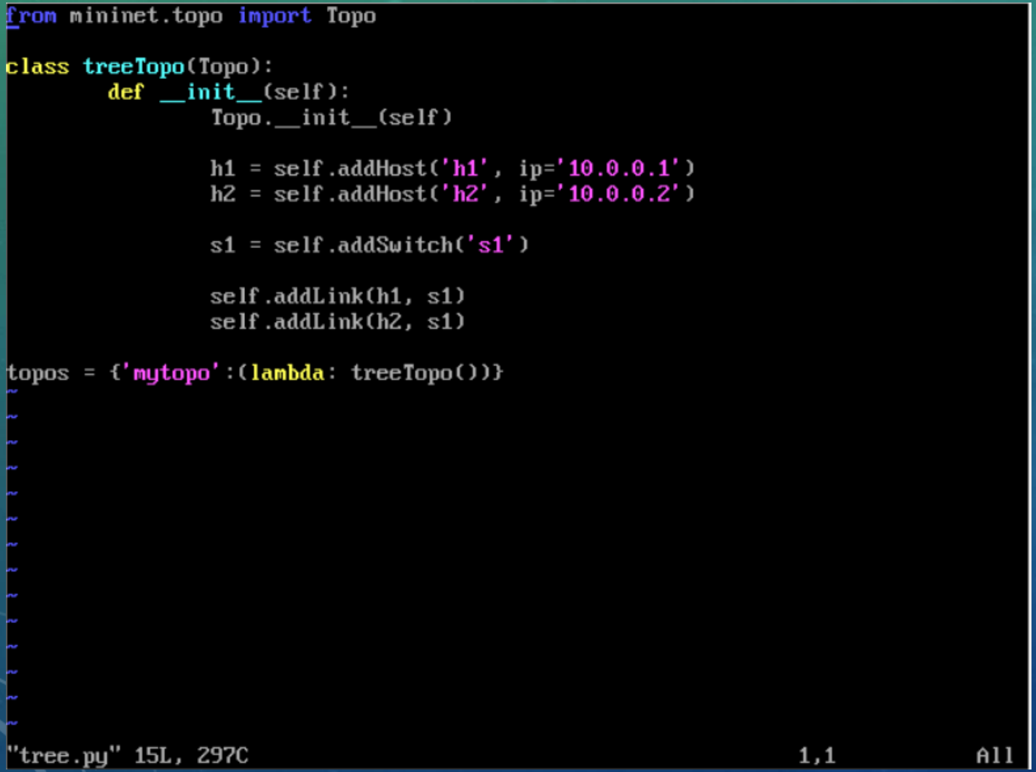


OUR TOPOLOGY



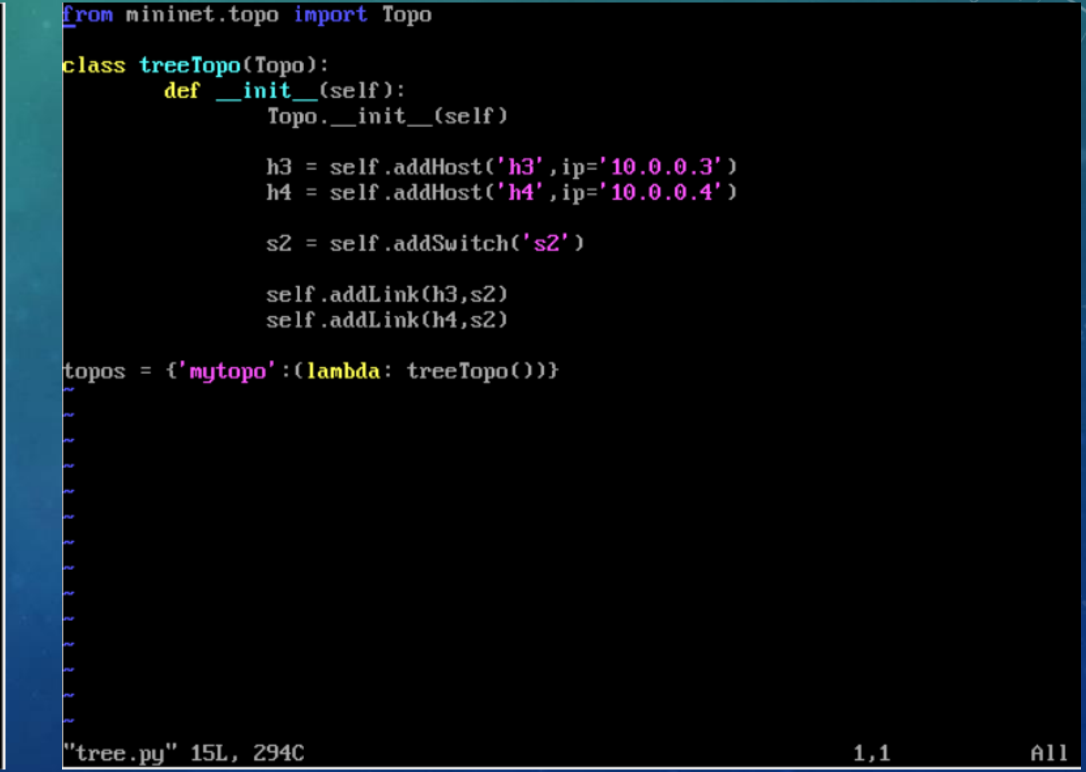
1) Run the Mininet VM instance
2) cd mininet
3) Create Tree.py with the following code

Mininet VM 1 - Tree.py



```
from mininet.topo import Topo

class treeTopo(Topo):
        def __init__(self):
                Topo.__init__(self)

                h1 = self.addHost('h1', ip='10.0.0.1')
                h2 = self.addHost('h2', ip='10.0.0.2')

                s1 = self.addSwitch('s1')

                self.addLink(h1, s1)
                self.addLink(h2, s1)

topos = {'mytopo':(lambda: treeTopo())}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"tree.py" 15L, 297C                                    1,1           All
```

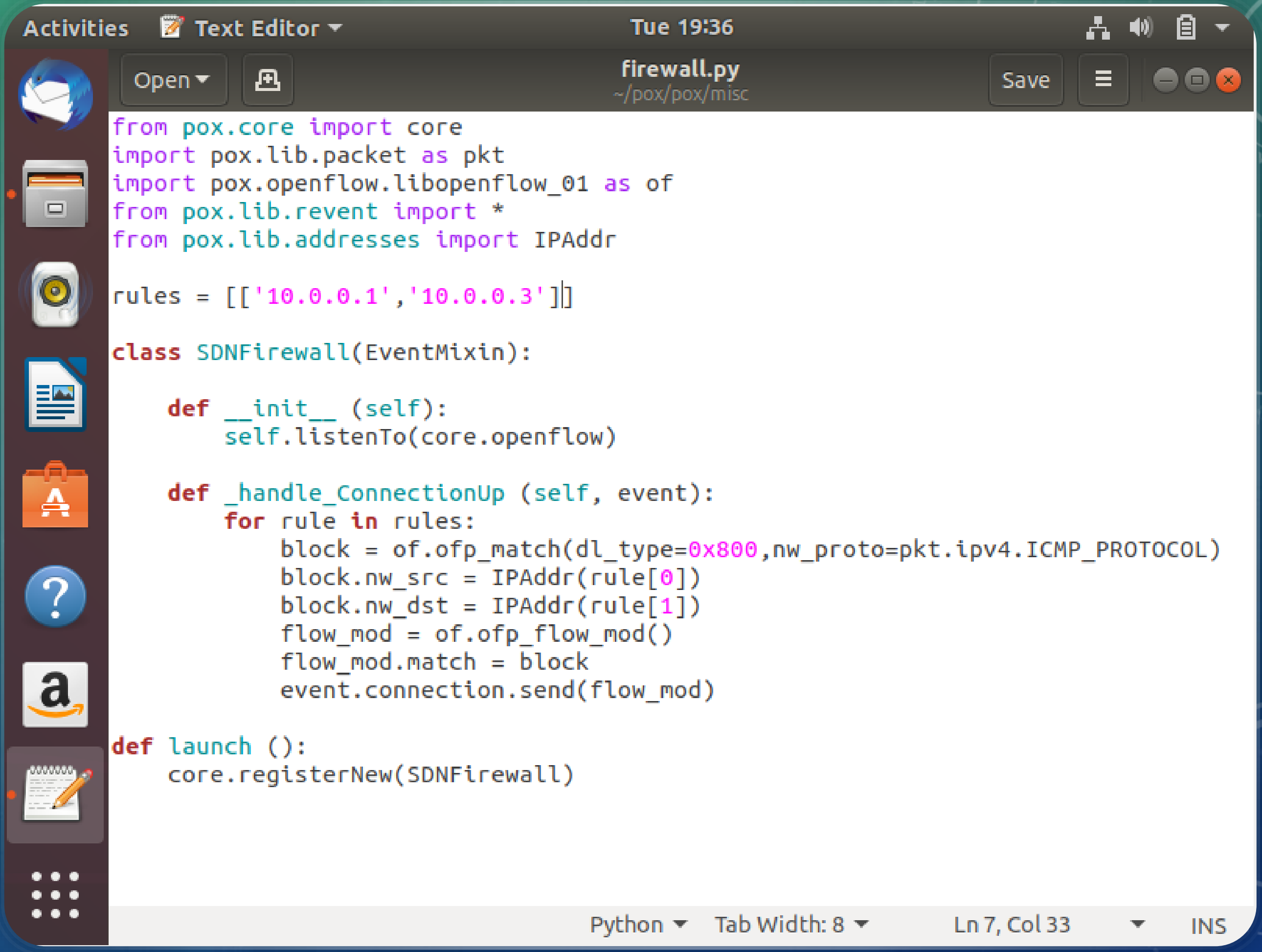
Mininet VM 2 - Tree.py



```
from mininet.topo import Topo

class treeTopo(Topo):
        def __init__(self):
                Topo.__init__(self)

                h3 = self.addHost('h3',ip='10.0.0.3')
                h4 = self.addHost('h4',ip='10.0.0.4')

                s2 = self.addSwitch('s2')

                self.addLink(h3,s2)
                self.addLink(h4,s2)

topos = {'mytopo':(lambda: treeTopo())}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"tree.py" 15L, 294C                                    1,1           All
```

Steps to install POX controller with custom Firewall

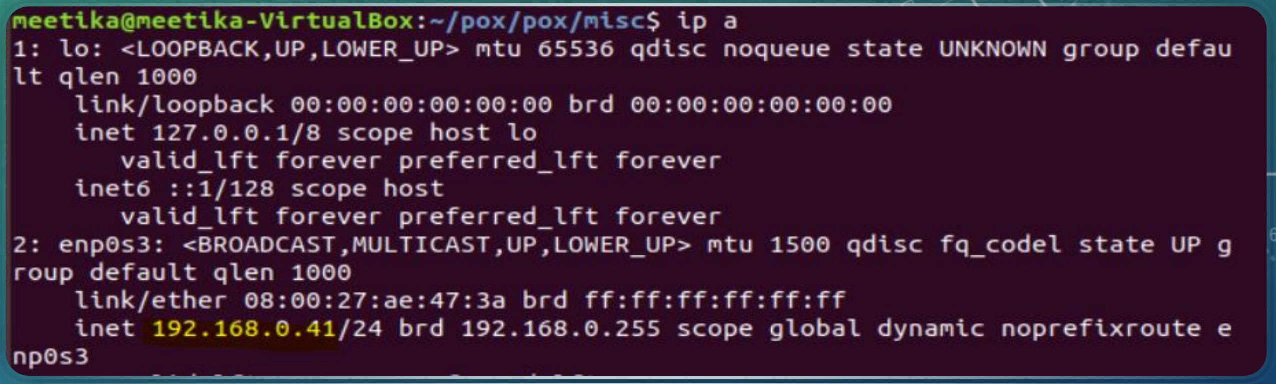
- Clone the git repository - git clone https://github.com/noxrepo/pox
- cd into the POX directory – cd pox/pox/misc
- Create/Update firewall.py with the following code
- Rules is a list of list which contains the IP addresses of the hosts to be blocked.
- We can add multiple rules to block multiple flows.



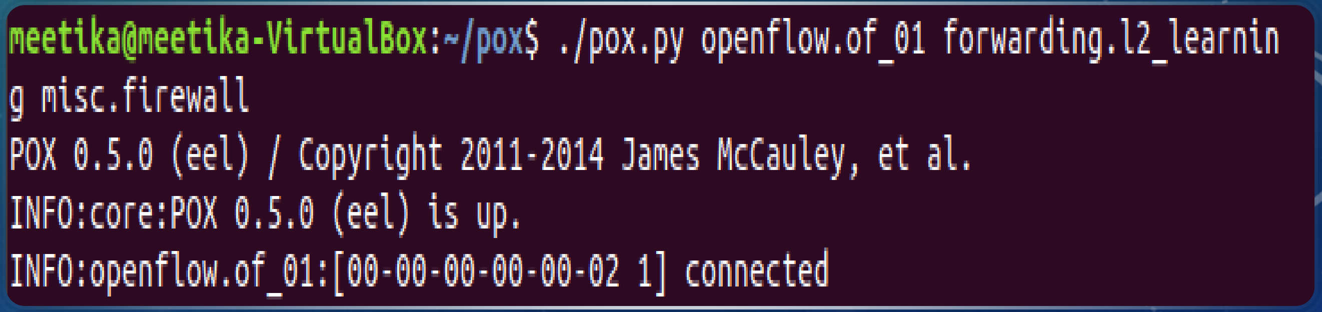
```python
from pox.core import core
import pox.lib.packet as pkt
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.addresses import IPAddr

rules = [['10.0.0.1','10.0.0.3']]

class SDNFirewall(EventMixin):

    def __init__ (self):
        self.listenTo(core.openflow)

    def _handle_ConnectionUp (self, event):
        for rule in rules:
            block = of.ofp_match(dl_type=0x800,nw_proto=pkt.ipv4.ICMP_PROTOCOL)
            block.nw_src = IPAddr(rule[0])
            block.nw_dst = IPAddr(rule[1])
            flow_mod = of.ofp_flow_mod()
            flow_mod.match = block
            event.connection.send(flow_mod)

def launch ():
    core.registerNew(SDNFirewall)
```

STEPS TO RUN THE POX CONTROLLER ON UBUNTU

- Get the IP address of the machine where POX controller will run using following command – ip a
- E.g. 192.168.0.41
- cd pox
- ./pox.py openflow.of_01 forwarding.l2_learning misc.firewall

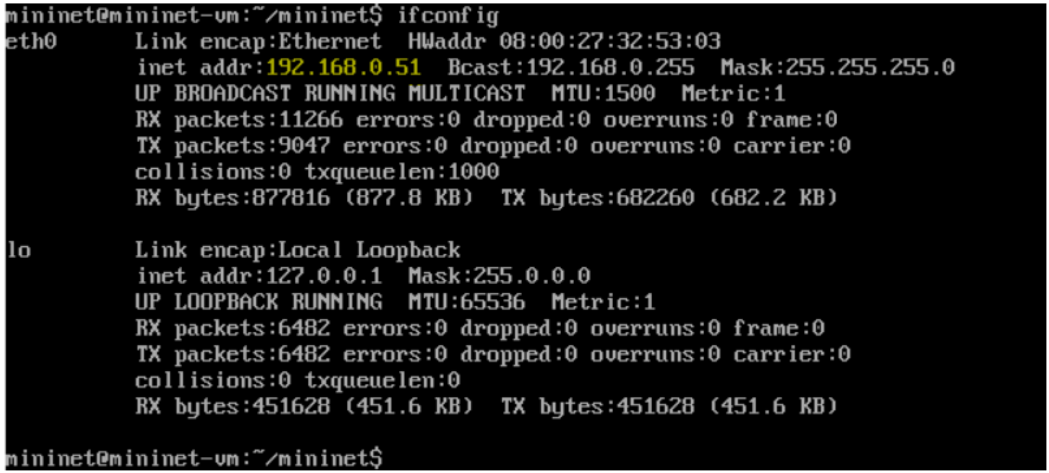


```
meetika@meetika-VirtualBox:~/pox/pox/misc$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defau
lt qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP g
roup default qlen 1000
    link/ether 08:00:27:ae:47:3a brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.41/24 brd 192.168.0.255 scope global dynamic noprefixroute e
np0s3
```



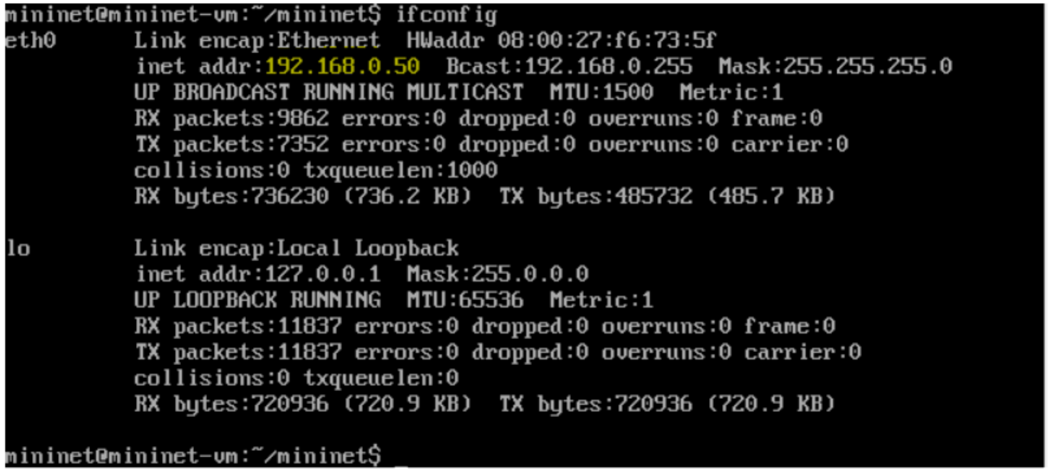
```
meetika@meetika-VirtualBox:~/pox$ ./pox.py openflow.of_01 forwarding.l2_learnin
g misc.firewall
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-02 1] connected
```

Mininet VM 1 - ifconfig



```
mininet@mininet-vm:~/mininet$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:32:53:03
          inet addr:192.168.0.51  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:11266 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9047 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:877816 (877.8 KB)  TX bytes:682260 (682.2 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:6482 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6482 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:451628 (451.6 KB)  TX bytes:451628 (451.6 KB)

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

Mininet VM 2 - ifconfig



```
mininet@mininet-vm:~/mininet$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:f6:73:5f
          inet addr:192.168.0.50  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:9862 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7352 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:736230 (736.2 KB)  TX bytes:485732 (485.7 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:11837 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11837 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:720936 (720.9 KB)  TX bytes:720936 (720.9 KB)

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

Connect both Mininet VM to remote POX controller using the following command

1) cd mininet
2) sudo mn --custom Tree.py --topo mytopo --mac --controller=remote,ip=192.168.0.41,port=6633

Tree.py – filename of your topology

192.168.0.41 – IP address of the machine where POX controller is running

Create a GRE tunnel between mininets using the following command

On VM 1

```
sh ovs-vsctl add-port s1 hello -- set interface hello type=gre options:remote_ip=192.168.0.50
```

On VM 2

```
sh ovs-vsctl add-port s2 hello -- set interface hello type=gre options:remote_ip=192.168.0.51
```

s1, s2 – Name of the switch used in the topology

192.168.0.XX – IP address of the other mininet

VM1 contains

- 1 Switch (s1)
- 2 Hosts (H1 (IP = 10.0.0.1), H2 (IP = 10.0.0.2))

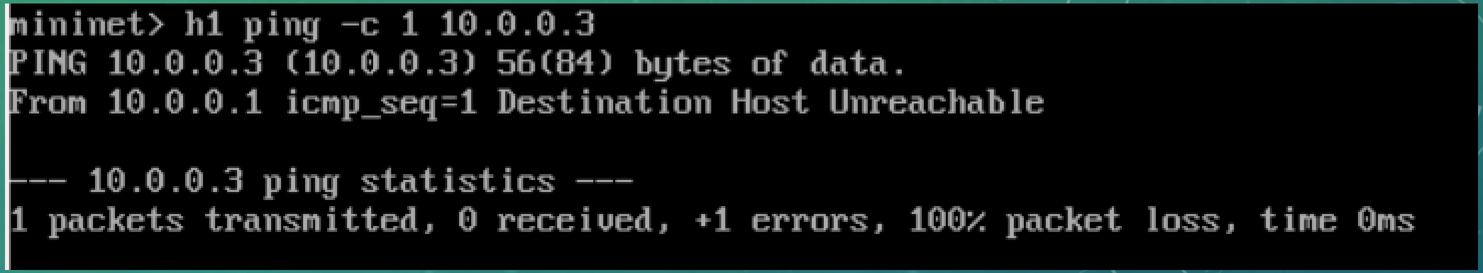
VM2 contains

- 1 Switch (s2)
- 2 Hosts (H3 (IP = 10.0.0.3), H4 (IP = 10.0.0.4))

Flow from Host1 to Host3 is blocked.

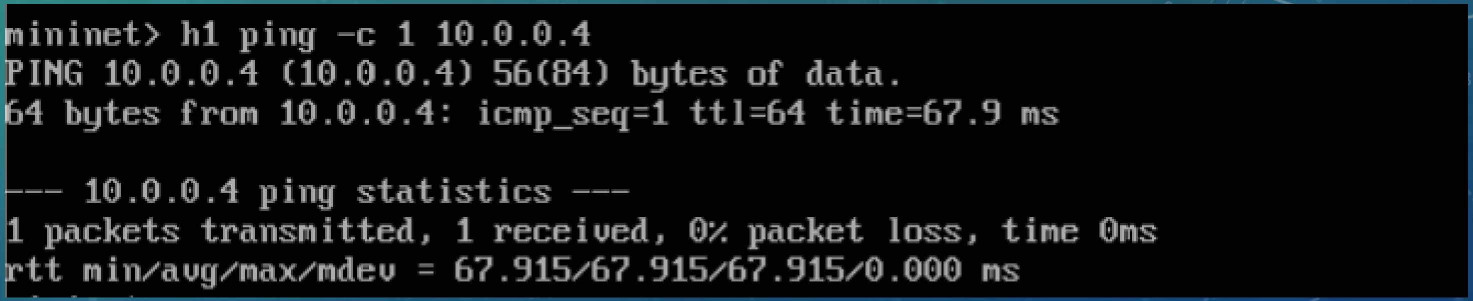
This is mentioned in the rules in firewall.py

H1 cannot ping H3



```
mininet> h1 ping -c 1 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable

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

H1 can ping H4



```
mininet> h1 ping -c 1 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=67.9 ms

--- 10.0.0.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 67.915/67.915/67.915/0.000 ms
```

CHALLENGES FACED

- Accessing mininet, controller on UTD network.
- Configuring the GRE Tunnel.
- We blacklisted the IP, but we faced issues while doing reverse job i.e. Whitelisting the IP for certain hosts.

Thank You