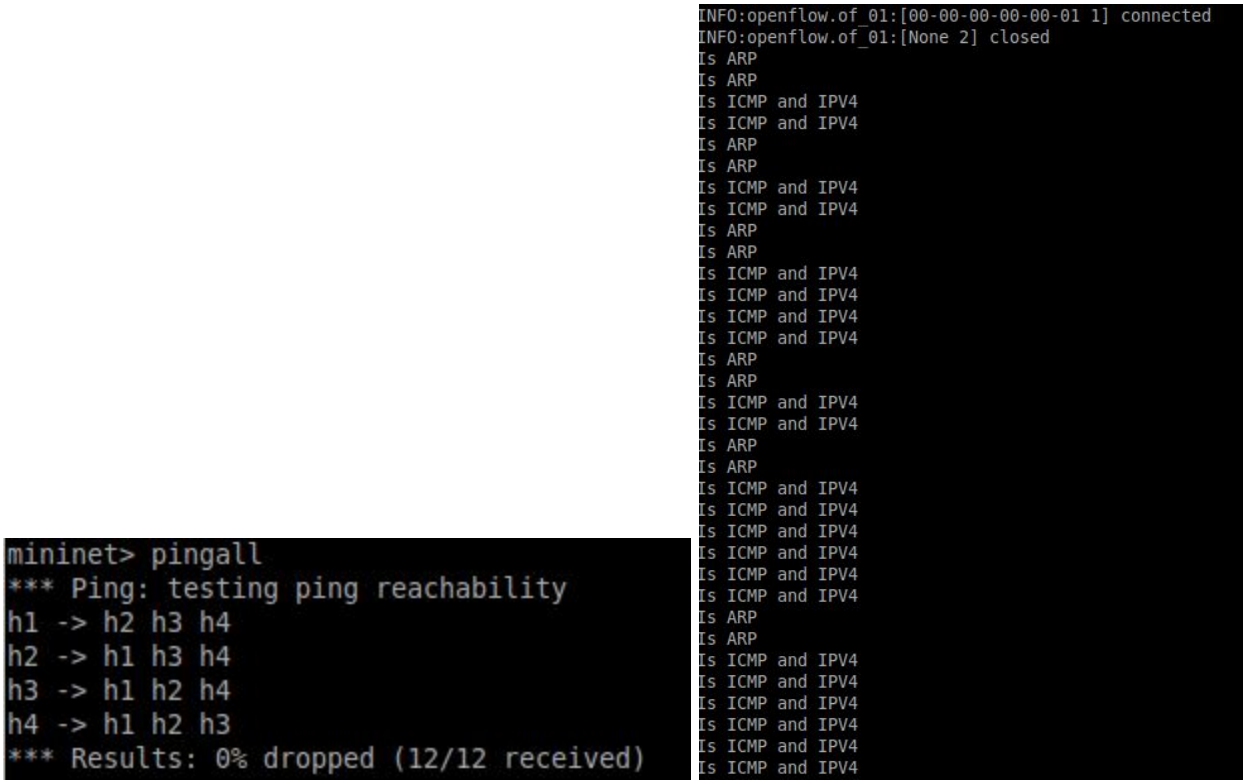
Drew von Zweck

CSE150/L

Lab3

Pingall: Using the pingall command uses ICMP and ARP traffic, which is allowed by the firewall and rules installed into the switch. There should be no packets dropped since all the hosts are able to communicate with this type of traffic.



Dpctl dump-flows: This command shows the type of traffic that was used during the pingall. The entries also show the actions that the controller took to send the packets. We can see the first

entry shows ICMP traffic and that it sent that rule which was to flood the packet.

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=7.095s, table=0, n_packets=1, n_bytes=98, idle_timeout=50,
 hard_timeout=50, idle_age=7, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_d
st=00:00:00:00:00:04,nw_src=10.0.1.10,nw_dst=10.0.1.40,nw_tos=0,icmp_type=8,icmp
_code=0 actions=FLOOD
 cookie=0x0, duration=7.116s, table=0, n_packets=1, n_bytes=98, idle_timeout=50,
 hard_timeout=50, idle_age=7, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_d
st=00:00:00:00:00:02,nw_src=10.0.1.10,nw_dst=10.0.1.20,nw_tos=0,icmp_type=8,icmp
_code=0 actions=FLOOD
 cookie=0x0, duration=7.093s, table=0, n_packets=1, n_bytes=98, idle_timeout=50,
 hard_timeout=50, idle_age=7, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:04,dl_d
st=00:00:00:00:00:01,nw_src=10.0.1.40,nw_dst=10.0.1.10,nw_tos=0,icmp_type=0,icmp
_code=0 actions=FLOOD
 cookie=0x0, duration=7.037s, table=0, n_packets=1, n_bytes=98, idle_timeout=50,
 hard_timeout=50, idle_age=7, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl_d
st=00:00:00:00:00:03,nw_src=10.0.1.20,nw_dst=10.0.1.30,nw_tos=0,icmp_type=0,icmp
_code=0 actions=FLOOD
 cookie=0x0, duration=7.028s, table=0, n_packets=1, n_bytes=98, idle_timeout=50,
 hard_timeout=50, idle_age=7, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_d
st=00:00:00:00:00:04,nw_src=10.0.1.30,nw_dst=10.0.1.40,nw_tos=0,icmp_type=8,icmp
_code=0 actions=FLOOD
 cookie=0x0, duration=7.085s, table=0, n_packets=1, n_bytes=98, idle_timeout=50,
 hard_timeout=50, idle_age=7, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_d
st=00:00:00:00:00:02,nw_src=10.0.1.10,nw_dst=10.0.1.20,nw_tos=0,icmp_type=0,icmp
_code=0 actions=FLOOD
 cookie=0x0, duration=7.104s, table=0, n_packets=1, n_bytes=98, idle_timeout=50,
 hard_timeout=50, idle_age=7, icmp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_d
```

Iperf: This command uses TCP to test the connection between the hosts. Since I added a rule to the controller that drops any packet besides ICMP and ARP, when using iperf the connection will hang for an indefinite amount of time, this means that the packet was dropped(because it is TCP)

```
mininet> iperf
*** Iperf: testing TCP bandwidth between h1 and h4
```

`Dropping packet` this is the message printed by lab3controller.py

Screenshots of Code

```
# Lab 3 Skeleton
#
# Based on of_tutorial by James McCauley

from pox.core import core
import pox.openflow.libopenflow_01 as of

log = core.getLogger()

class Firewall (object):
  """
  A Firewall object is created for each switch that connects.
  A Connection object for that switch is passed to the __init__ function.
  """
  def __init__ (self, connection):
    # Keep track of the connection to the switch so that we can
    # send it messages!
    self.connection = connection

    # This binds our PacketIn event listener
    connection.addListeners(self)

  def do_firewall (self, packet, packet_in):
    # The code in here will be executed for every packet.
    ##first we make a variable for isICMP and isARP to check the packet against each
    isICMP = packet.find('icmp')
    isARP = packet.find('arp')
    isIPV4 = packet.find('ipv4')
```

```
### if statement that will check if the packet is ARP or ICMP... by seeing if
### the variables are null, or .find returned something(which tells us that it matched)
if isICMP is not None or isARP is not None:

  ## here we see if its ICMP and IPV4
  if isICMP is not None and isIPV4 is not None:
    print "Is ICMP and IPV4"
    #allow ICMP traffic
    msg = of.ofp_flow_mod()
    msg.match = of.ofp_match.from_packet(packet)
    msg.data = packet_in
    msg.idle_timeout = 50
    msg.hard_timeout = 50
    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    self.connection.send(msg)

  #if its not IPV4 and ICMP then check if its ARP
  elif isARP is not None:
    #allow ARP traffic
    print "Is ARP"
    msg = of.ofp_flow_mod()
    msg.match = of.ofp_match.from_packet(packet)
    msg.data = packet_in
    msg.idle_timeout = 50
    msg.hard_timeout = 50
    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    self.connection.send(msg)
```

```
    ###Any other type of traffic should be dropped regardless of what protocol it is
    else:
       ### we drop packet
       print "Dropping packet"
       msg = of.ofp_flow_mod()
       msg.match = of.ofp_match.from_packet(packet)
       msg.idle_timeout = 50
       msg.hard_timeout = 50
       self.connection.send(msg)
```

```
  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.
    self.do_firewall(packet, packet_in)
```

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