EECE.7290 Selected Topics on Software Defined Networking

Lab 1: Experimenting OpenFlow with Mininet.

Name : Jigar Makwana

ID : 0171137

Email : jigar_makwana@student.uml.edu

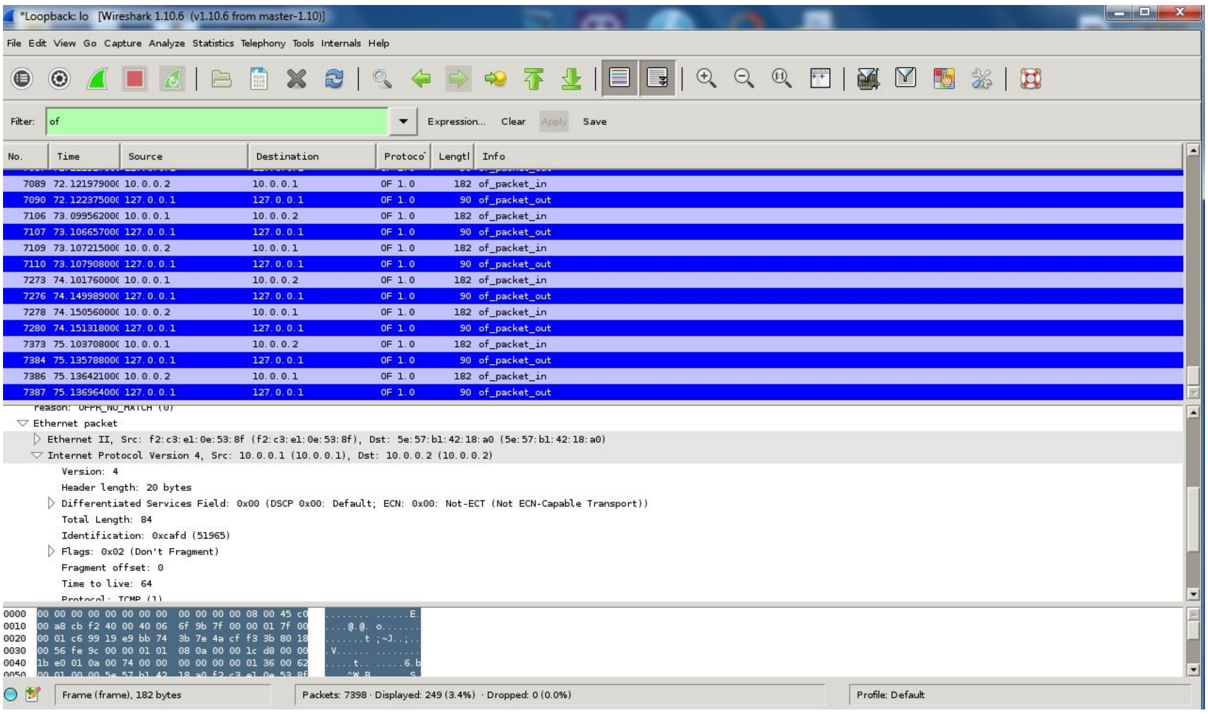
Purpose:

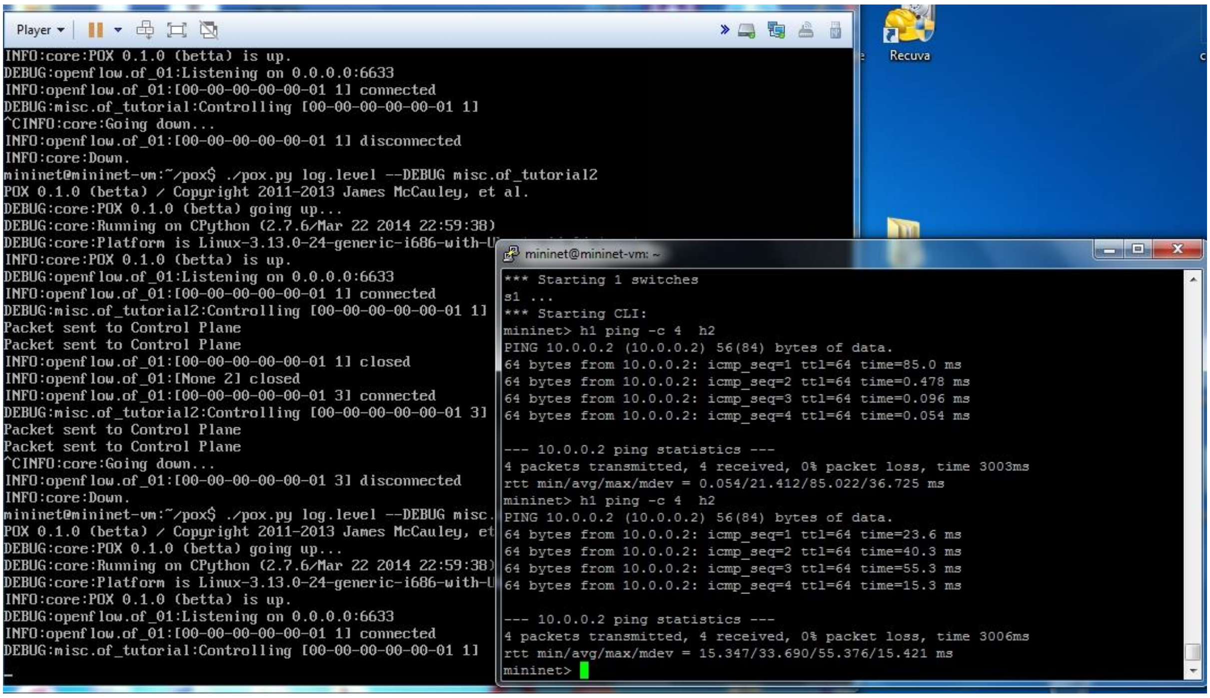
The main purpose this lab is to experiment Open-flow with mini-net software to understand the working of l2 learning switch. We need to make use of POX controller to design l2 learning switch in Python programming language. Open-flow packets are captured using Wireshark.

Lab Procedure:

Act Like a Hub: Before we start to program POX controller to act like a learning switch, we need to run it like a hub to troubleshoot the setup. By default, of_tutorial.py in misc folder of pox library is configured to flood the output ports of the switch for every packet it received. You can start the POX controller from the pox library by using the command: **./pox.py log.level - -DEBUG misc.of_tutorial**

Wireshark showing the capture of OpenFlow packets as a hub

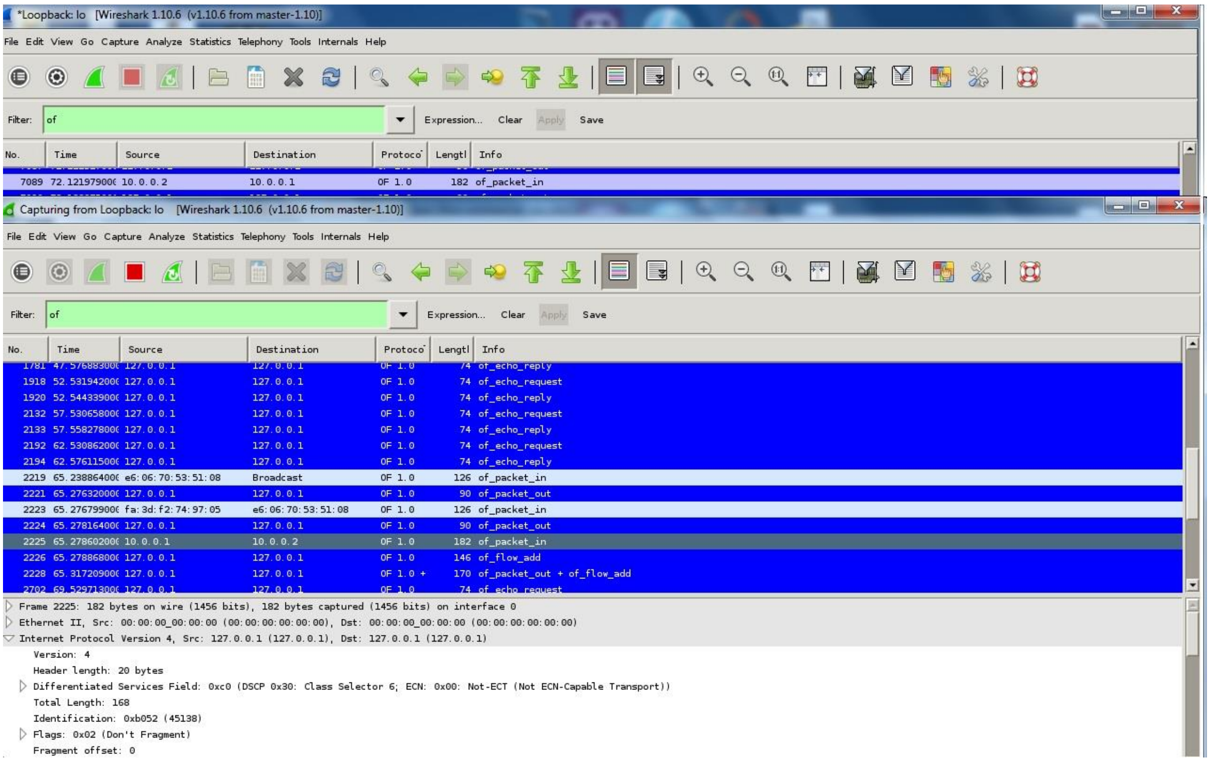




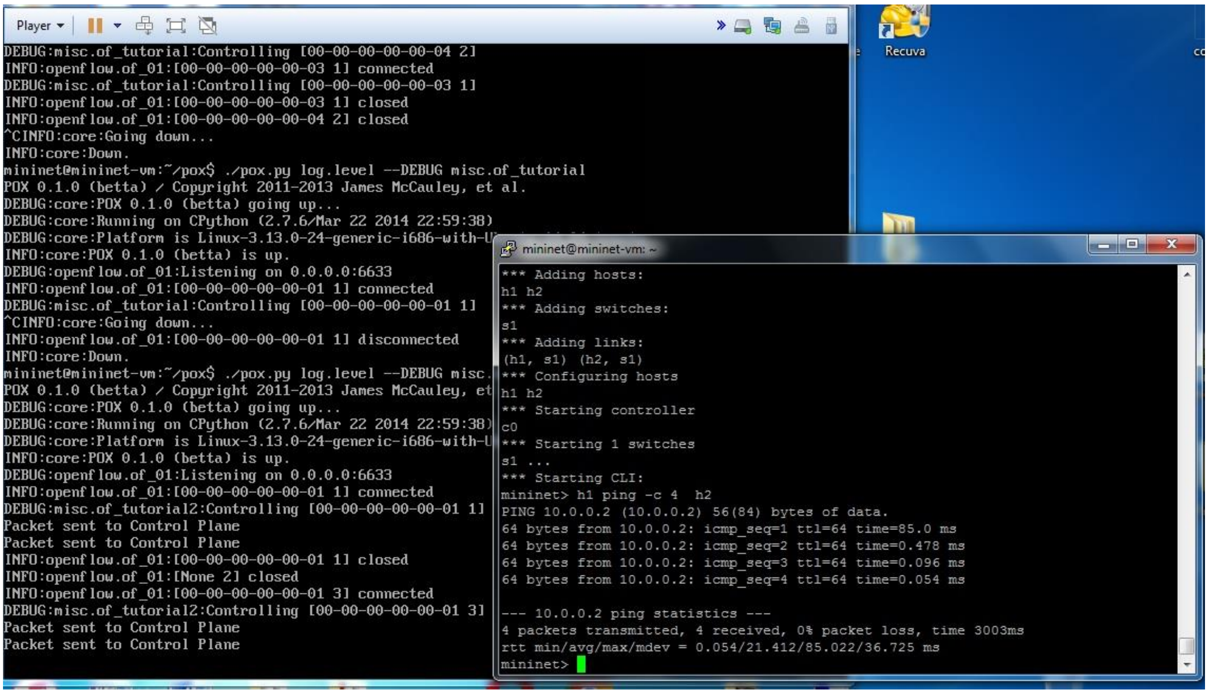
Default topology from the mini-net can started using the command below. It contains two hosts, one switch and it is direct to connect to POX controller listening on port 6633 in the localhost. You can try to ping h2 from h1 using the command: **h1 ping h2** (-c 4 for 4 pings).

Wireshark showing the capture of OpenFlow packets as a L2 Learning Switch

Act Like a L2 Learning Switch: After configuring the controller to act like a switch, you can flow instructions given in above section to start the controller and topology. Also, in this case, we use the same network topology. Then when we observe the packet capture, it contains few PACKET_IN, PACKET_OUT initially but after FLOW_MOD messages, you can’t find any.



As shown above, you can see few PACKET_IN and PACKET_OUT. And FLOW_MOD messages also appear. If you see the flow modification message, it has time outs, buffer id etc.



Shown in the above fig., the controller is installing the flows in the switch by sending flow modification messages.

The source code of a L2 Learning switch controller for POX

```
from pox.core import core

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):

    # 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)
  # Use this table to keep track of which ethernet address is on
  # which switch port (keys are MACs, values are ports).
  self.mac_to_port = {}
def resend_packet (self, packet_in, out_port):
  """
  Instructs the switch to resend a packet that it had sent to us.
  "packet_in" is the ofp_packet_in object the switch had sent to the
  controller due to a table-miss.
  """
  msg = of.ofp_packet_out()
  msg.data = packet_in
  # Add an action to send to the specified port
  action = of.ofp_action_output(port = out_port)
  msg.actions.append(action)
  # Send message to switch
  self.connection.send(msg)
def act_like_hub (self, packet, packet_in):
  """
  Implement hub-like behavior -- send all packets to all ports besides
  the input port.
  """
  # We want to output to all ports -- we do that using the special
  # OFPP_ALL port as the output port.  (We could have also used
  # OFPP_FLOOD.)
  self.resend_packet(packet_in, of.OFPP_ALL)
  # Note that if we didn't get a valid buffer_id, a slightly better
```

```
  # implementation would check that we got the full data before
  # sending it (len(packet_in.data) should be == packet_in.total_len)).
def act_like_switch (self, packet, packet_in):
  """
  Implement switch-like behavior.
  """
  #Learn the port fo source MAC
  self.mac_to_port[packet.src] = packet_in.in_port
  # if the port associated with the destination MAC of the packet is known:
  if packet.dst in self.mac_to_port:
    # Send the packet to the associated port
    self.resend_packet(packet_in,
                self.mac_to_port[packet.dst])
    log.debug(("Installing flow...Source" )
    # Install a flow
    msg = of.ofp_flow_mod()
    msg.match.dl_dst = packet.dst
    msg.actions.append(of.ofp_action_output(port=self.mac_to_port[packet.dst]))
    self.connection.send(msg)
  else:
    self.resend_packet(packet_in, of.OFPP_ALL)
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.
```

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