# The given below is the modified file for learning switches with the license.

# Copyright 2012 James McCauley

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at:

#

# http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

"""

This component is for use with the OpenFlow tutorial.

It acts as a simple hub, but can be modified to act like an L2

learning switch.

It's roughly similar to the one Brandon Heller did for NOX.

"""

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):

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.

"""

# Here's some psuedocode to start you off implementing a learning

# switch. You'll need to rewrite it as real Python code.

# Learn the port for the source MAC

self.mac\_to\_port[packet.src]=packet\_in.in\_port

if packet.dest in self.mac\_to\_port:

# Send packet out the associated port

self.resend\_packet(packet\_in, self.mac\_to\_port[packet.dst])

# Once you have the above working, try pushing a flow entry

# instead of resending the packet (comment out the above and

# uncomment and complete the below.)

log.debug("Installing flow for %s.%i -> %s.%i"%(packet.src,packet\_in.in\_port,packet.dst,port )

# Maybe the log statement should have source/destination/port?

msg = of.ofp\_flow\_mod()

#Set fields to match received packet

msg.match = of.ofp\_match.from\_packet(packet)

msg.idle\_timeout = 10

msg.hard\_timeout = 30

msg.actions.append(of.ofp\_action\_output(port=self.mac\_to\_port[packet.dst]))

self.connection.send(msg)

#< Add an output action, and send -- similar to resend\_packet() >

else:

# Flood the packet out everything but the input port

# This part looks familiar, right?

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)