```
# Final Skeleton
#
# Hints/Reminders from Lab 3:
#
# To check the source and destination of an IP packet, you can use
# the header information... For example:
#
# ip_header = packet.find('ipv4')
#
# if ip_header.srcip == "1.1.1.1":
#   print "Packet is from 1.1.1.1"
#
# Important Note: the "is" comparison DOES NOT work for IP address
# comparisons in this way. You must use ==.
#
# To send an OpenFlow Message telling a switch to send packets out a
# port, do the following, replacing <PORT> with the port number the
# switch should send the packets out:
#
#    msg = of.ofp_flow_mod()
#    msg.match = of.ofp_match.from_packet(packet)
#    msg.idle_timeout = 30
#    msg.hard_timeout = 30
#
#    msg.actions.append(of.ofp_action_output(port = <PORT>))
#    msg.data = packet_in
#    self.connection.send(msg)
#
# To drop packets, simply omit the action.
#

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

log = core.getLogger()


class Final (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_final(self, packet, packet_in, port_on_switch, switch_id):
        # This is where you'll put your code. The following modifications have
        # been made from Lab 3:
        #   - port_on_switch: represents the port that the packet was received on.
        #   - switch_id represents the id of the switch that received the packet.
        #      (for example, s1 would have switch_id == 1, s2 would have switch_id
== 2, etc...)
        # You should use these to determine where a packet came from. To figure out
```

```
where a packet
        # is going, you can use the IP header information.
        # print "Example code."

        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet)
        msg.idle_timeout = 30
        msg.hard_timeout = 60
        msg.data = packet_in

        # search for ip and icmp packets
        ip = packet.find('ipv4')
        icmp = packet.find('icmp')
        ipport = 0

        # check if the packet is ip
        # need to specify ports for all IP traffic
        if ip:
            msg.priority = 50
            # set the destination
            dest = ip.dstip

            # check if the packet is icmp
            if icmp:
                if switch_id == 1:

                    if port_on_switch == 2 or port_on_switch == 3:

                        if dest == '10.2.5.50' or dest == '10.2.6.60' or dest ==
'10.2.7.70' or dest == '10.2.8.80':
                            self.connection.send(msg)
                            return

                    elif port_on_switch == 4 or port_on_switch == 5:
                        if dest == '10.1.1.10' or dest == '10.1.2.20' or dest ==
'10.1.3.30' or dest == '10.1.4.40':
                            self.connection.send(msg)
                            return

                    elif port_on_switch == 6:
                        if dest == '10.2.5.50' or dest == '10.2.6.60' or dest ==
'10.2.7.70' or dest == '10.2.8.80':
                            self.connection.send(msg)

                    # untrusted host
                    elif port_on_switch == 7:
                        self.connection.send(msg)
                        return

            # traffic is not icmp
            # match the destination ip with the correct ipport
            if switch_id == 1:
                if dest == '10.3.9.90' and port_on_switch < 6:
                    ipport = 1
                elif dest == '10.1.1.10' or dest == '10.1.2.20':
                    ipport = 2
                elif dest == '10.1.3.30' or dest == '10.1.4.40':
                    ipport = 3
                elif dest == '10.2.5.50' or dest == '10.2.6.60':
```

```
            ipport = 4
        elif dest == '10.2.7.70' or dest == '10.2.8.80':
            ipport = 5
        elif dest == '108.24.31.112':
            ipport = 6
        elif dest == '106.44.82.103':
            ipport = 7

    elif switch_id == 2:
        if dest == '10.1.3.30' or dest == '10.1.4.40':
            ipport = 1
        elif dest == '10.1.1.10':
            ipport = 3
        elif dest == '10.1.2.20':
            ipport = 4
        else:
            ipport = 1

    elif switch_id == 3:
        if dest == '10.1.1.10' or dest == '10.1.2.20':
            ipport = 1
        elif dest == '10.1.3.30':
            ipport = 3
        elif dest == '10.1.4.40':
            ipport = 4
        else:
            ipport = 1

    elif switch_id == 4:
        if dest == '10.2.7.70' or dest == '10.2.8.80':
            ipport = 1
        elif dest == '10.2.5.50':
            ipport = 3
        elif dest == '10.2.6.60':
            ipport = 4
        else:
            ipport = 1

    elif switch_id == 5:
        if dest == '10.2.5.50' or dest == '10.2.6.60':
            ipport = 1
        elif dest == '10.2.7.70':
            ipport = 3
        elif dest == '10.2.8.80':
            ipport = 4
        else:
            ipport = 1

    elif switch_id == 6:
        if port_on_switch == 1:
            ipport = 2
        elif port_on_switch == 2:
            ipport = 1

# traffic is not ip and ipport is none
else:
    msg.priority = 49
    ipport = None
```

```
        # traffic is ip so set port to ipport
        if ipport > 0:
            msg.actions.append(of.ofp_action_output(port=ipport))

        # traffic is not ip so flood
        elif ipport is None:
            msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))

        self.connection.send(msg)
        return

    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_final(packet, packet_in, event.port, event.dpid)


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