TTM4180 - Applied Networking
Lab 2: Python and POX

By S. Xiao Fernández Marín

Q1: What is one of the primary purposes for using POX?

As we are working with python, we need to have access to some specific libraries and functions. Those are implemented in the API (code written by someone and that give us access to use it) POX and are going to help us getting the documentation and syntax checking.
We get to have these libraries and functions communicating with the control plane (implemented in OpenVSwitch) using the OpenFlow protocol. In this way, when we are creating the Python application with help of PyCharm, out laptop or computer will communicate with the POX, that will communicate withe the controller using this protocol.

Now that we have seen how it works, we are going to use POX to create Python applications and run them in the controller, as POX is a Python based SDN controller. For that we have to install PyCharm or any other IDE in the container, for the IDE to have access to the directories of the controller. In this way, we have the libraries and functions we need to complete these course labs.

Q2: Go to section "OpenFlow Actions" in the POX documentation and explain what the ofp_action_output() class is used for.

It forward packets from a physical or virtual port. The physical ports are referenced by their integral value (less than 0xFF00) but the value of the virtual ones are specified in variables.

Structure definition:

```
class ofp_action_output (object):
  def __init__ (self, **kw): # initialize the data entered as a variable in the
  ↪  function
    self.port = None # name/number of the port
```

Q3: What is the purpose of OFPP_ALL?

As a variable given by POX, it is used in the ofp_action_output class and it sends the package received to all the openflow virtual ports, but not the one that is used as entrance port.

Q4: What is the purpose of ofp_match? How do you specify a match on the IP 10.0.0.4?

The ofp_match is a structure that enables you to define headers for packages to match. You can build this structure from scratch or using some base structure from some existing package. Some of the attributes are: The match on the IP 10.0.0.4 will be

Attribute	Meaning
in_port	Switch port number the packet arrived on
dl_src	Ethernet source address
dl_dst	Ethernet destination address
nw_src	IP source address
nw_dst	IP destination address

Table 1: ofp_match attributes table

```
my_match = of.ofp_match(..., nw_src = "10.0.0.4", ...) # if you want a match in
↪  the source address

my_match = of.ofp_match(..., nw_dst = "10.0.0.4", ...) # if you want a match in
↪  the destination address
```

Q5: Which POX library contains the packet class, and which types of packets are supported?

```
import pox.lib.packet # library for parsing and constructing packages. It works
↪  by looking at the package header and seeing the type of package is and if
↪  they have some payload (another package inside it) i.e. ethernet most times
↪  contains ipv4 pacakges wich sometimes have tcp packages
```

The packet types supported are ethernet, ARP, IPv4, ICMP, TCP, UDP, DHCP, DNS, LLDP and VLAN.

Q6: The packet class has a method named find() associated with it. What is it used for, and how could it be used to determine if a packet is IPv4?

Navigate the encapsulated packets. The find() method can be used to determine if a packet is IPv4 in the next way:

```
def check_IP (packet):
  ip = packet.find('ipv4') # tries to find if it is IP
  if ip is None: # This packet isn't IP!
    return False
  return True
```

Q7: How would you obtain an IP packet's source IP?

```
def check_IP (packet):
  ip = packet.find('ipv4') # tries to find if it is IP
  if ip is None: # This packet isn't IP!
    print "Not IP found"
    return
  print "Package srcIP", ip.srcip
```

Q8: What is the Connection object?

The connection object is in charge of communicate the datapaths to the code. When a switch connects to POX (via OpenFlow), there is an associated Connection object. If your code has a reference to that Connection object, you can send commands to switches and are sources of events from them, between others.

Q9: What is a DPID and what is its relation to the Connection objects?

The DPID is the datapath identifier of the switch, in other words, if you want to send any message or get any connection with a certain switch, you use this.

Q10: What is the purpose of the ofp_packet_out message?

The ofp_packet_out function is to tell a switch to send or enqueue a package, but also to tell it to discard a buffered package.

Q11: The POX controller receives a packet, and the packetIn event is triggered. Submit code where the event is handled in the following way. If the packet is an ARP request, an ARP reply should be instantiated with fields that allows it to be sent back to the source of the request. If the packet is not an ARP request, do nothing.

Next code extracted from McCauley 2015. I modified it a bit and added the comments explaining what it does.

```
def _handle_PacketIn (self, event): # trigger for the PackageIn event
  packet = event.parsed # parse the event
  if packet.type == packet.ARP_TYPE: # check if it is ARP
```

```
if packet.payload.opcode == arp.REQUEST: # check if it is a request
  arp_reply = arp() # builds an ip address

  # verify it is the main server
  arp_reply.hwsrc = <requested mac address>
  arp_reply.hwdst = packet.src
  arp_reply.opcode = arp.REPLY # it is an ARP reply
  arp_reply.protosrc = <IP of requested mac-associated machine>
  arp_reply.protodst = packet.payload.protosrc

  # creates ethernet package
  ether = ethernet()
  ether.type = ethernet.ARP_TYPE
  ether.dst = packet.src
  ether.src = <requested mac address>
  ether.payload = arp_reply

  #send this packet to the switch
else:
  return # if not, do nothing
```

Bibliography

McCauley, et al.. (2015). *POX Documemtation, Example: ARP Messages*. Last accessed 22 February 2022. URL: https://noxrepo.github.io/pox-doc/html/#example-arp-messages.