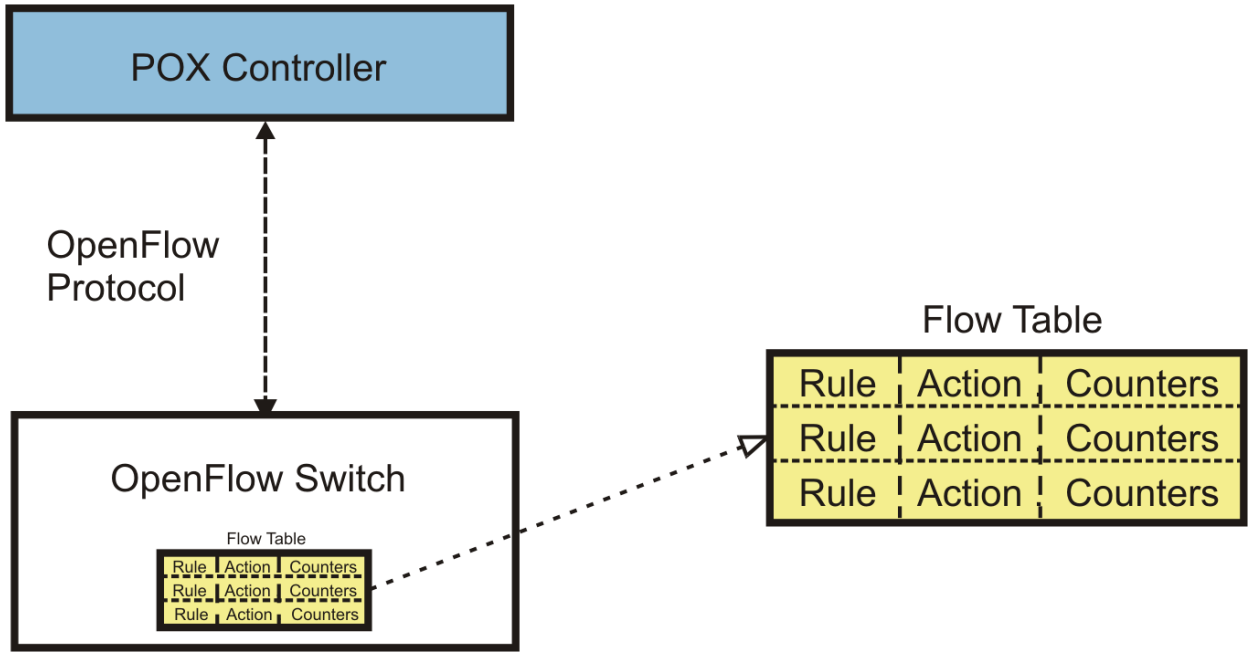
POX Controller

Vipin Gupta
BE,RHCE,CEH,CCNA,MCSE,MCSA
vipin2411@gmail.com
Mobile: 93563-10379
www.linuxexpert.in

POX Controller



Controller in a software defined network (SDN) is the brain of the network. POX is an open source controller for developing SDN applications. POX is Python based Controller.

POX controller provides an efficient way to implement the OpenFlow protocol which is the de facto communication protocol between the controllers and the switches. Using POX controller you can run different applications like hub, switch, load balancer and firewall.

Main Functions & Classes of POX Controller

ofp_match: This class describes packet header fields and an input port to match on. All fields are optional. Fields not specified will be treated as wildcard & will match on anything. Some of the fields are:

dl_type: It is used to specify the whether the packet is arp (0x806) or ip(0x800) type
dl_src,dl_dst: for specifying layer 2 source & destination mac address.
in_port: the port through which packet came
tp_dst: for specifying tcp/udp destination port
nw_src, nw_dst: for specifying IP addresses.

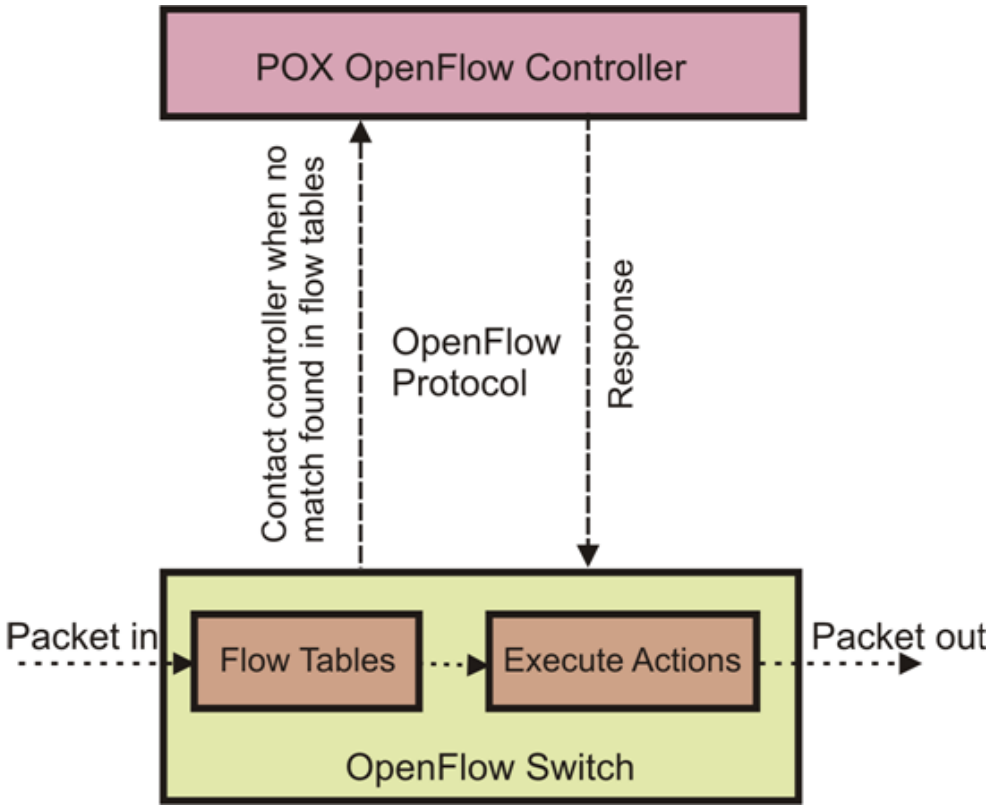
example 1: *create a match for packets arriving on port 1*

```
match=of.ofp_match()
match.in_port=1
```

example 2: *for creating match for* IP *packets arriving on port 2 & going towards port 80 of host 10.0.0.4*

```
match=of.ofp_match()
match.in_port=2
match.dl_type=0x800
match.nw_dst=10.0.0.4
match.tp_dst=80
```

Main Functions & Classes of POX Controller



connection.send(): is used for sending messages (instructions)

ofp_action_output: This is used with *ofp_packet_out & ofp_flow_mod* class. Here we have to specify the port through which we want to send the packet out.

example 1: *create an output action that will send the packet out from port 4.*
output_action=of.ofp_action_output(port=4)

example 2: *create an output action that will send the packet to the controller.*
output_action=of.ofp_action_output(port=of.OFPP_CONTROLLER)

example 3: *create an output action that will send the packet out from all port except in_port.*
output_action=of.ofp_action_output(port=of.OFPP_FLOOD)

Main Functions & Classes of POX Controller

ofp_flow_mod: It is openflow message(instruction) sent by the controller to the switch to install flow entries into flow table. Incoming packets are matched against these flow entries & action is performed on these packets as specified in flow entries. The main fields of ofp_flow_mod message are:

hard_timeout: After how many seconds the flow entry will be removed. Default is no timeout.
idle_timeout: After how many seconds the idle flow entries will be removed. Default is no timeout.
priority: relative importance of flow entries.
actions: list of actions to be performed on matched entries.
buffer_id: The buffer id of packet.
in_port: The port at which the packet arrived.
match: for specifying ofp_match. Default is match all.

example 1: *create a flow modification message that sends ip packets having source ip address 10.0.0.1 from port 1 to port 3.*

```
msg=of.ofp_flow_mod()
msg.match.in_port=1
msg.match.dl_type=0x800
msg.match.nw_src=10.0.0.1
msg.actions.append(of.ofp_action_output(port=3))
```

Main Functions & Classes of POX Controller

ofp_packet_out: It is openflow message sent by the controller to the switch to send the packet out. The packet sent out could be the one that was received by the switch & forwarded to the controller after buffering or the packet created by the controller itself. The main fields are:

data: raw data that you want to send. No need if sending buffered data.
buffer_id: The buffer id of the packet
action: list of actions
in_port: the port on which the packet arrived. Specify OFPP_NONE for packet created at controller.

`ofp_match` attributes:

Attribute	Meaning
in_port	Switch port number the packet arrived on
dl_src	Ethernet source address
dl_dst	Ethernet destination address
dl_vlan	VLAN ID
dl_vlan_pcp	VLAN priority
dl_type	Ethertype / length (e.g. 0x0800 = IPv4)
nw_tos	IP TOS/DS bits
nw_proto	IP protocol (e.g., 6 = TCP) or lower 8 bits of ARP opcode
nw_src	IP source address
nw_dst	IP destination address
tp_src	TCP/UDP source port
tp_dst	TCP/UDP destination port

Attributes may be specified either on a match object or during its initialization. That is, the following are equivalent:

```
my_match = of.ofp_match(in_port = 5, dl_dst = EthAddr("01:02:03:04:05:06"))
#.. or ..
my_match = of.ofp_match()
my_match.in_port = 5
my_match.dl_dst = EthAddr("01:02:03:04:05:06")
```

Output

Forward packets out of a physical or virtual port. Physical ports are referenced to by their integral value, while virtual ports have symbolic names. Physical ports should have port numbers less than 0xFF00.

Structure definition:

```
class ofp_action_output (object):
  def __init__ (self, **kw):
    self.port = None # Purposely bad -- require specification
```

- port (int) the output port for this packet. Value could be an actual port number or one of the following virtual ports:
  - OFPP_IN_PORT - Send back out the port the packet was received on. Except possibly OFPP_NORMAL, *this is the only way to send a packet back out its incoming port.*
  - OFPP_TABLE - Perform actions specified in flowtable. Note: Only applies to ofp_packet_out messages.
  - OFPP_NORMAL - Process via normal L2/L3 legacy switch configuration (if available – switch dependent)
  - OFPP_FLOOD - output all openflow ports except the input port and those with flooding disabled via the OFPPC_NO_FLOOD port config bit (generally, this is done for STP)
  - OFPP_ALL - output all openflow ports except the in port.
  - OFPP_CONTROLLER - Send to the controller.
  - OFPP_LOCAL - Output to local openflow port.
  - OFPP_NONE - Output to no where.

ofp_flow_mod - Flow table modification

```
class ofp_flow_mod (ofp_header):
  def __init__ (self, **kw):
    ofp_header.__init__(self)
    self.header_type = OFPT_FLOW_MOD
    if 'match' in kw:
      self.match = None
    else:
      self.match = ofp_match()
    self.cookie = 0
    self.command = OFPFC_ADD
    self.idle_timeout = OFP_FLOW_PERMANENT
    self.hard_timeout = OFP_FLOW_PERMANENT
    self.priority = OFP_DEFAULT_PRIORITY
    self.buffer_id = None
    self.out_port = OFPP_NONE
    self.flags = 0
    self.actions = []
```

- cookie (int) - identifier for this flow rule. (optional)
- command (int) - One of the following values:
    - OFPFC_ADD - add a rule to the datapath (default)
    - OFPFC_MODIFY - modify any matching rules
    - OFPFC_MODIFY_STRICT - modify rules which strictly match wildcard values.
    - OFPFC_DELETE - delete any matching rules
    - OFPFC_DELETE_STRICT - delete rules which strictly match wildcard values.
- idle_timeout (int) - rule will expire if it is not matched in 'idle_timeout' seconds. A value of OFP_FLOW_PERMANENT means there is no idle_timeout (the default).
- hard_timeout (int) - rule will expire after 'hard_timeout' seconds. A value of OFP_FLOW_PERMANENT means it will never expire (the default)
- priority (int) - the priority at which a rule will match, higher numbers higher priority. Note: Exact matches will have highest priority.
- buffer_id (int) - A buffer on the datapath that the new flow will be applied to. Use None for none. Not meaningful for flow deletion.
- out_port (int) - This field is used to match for DELETE commands.OFPP_NONE may be used to indicate that there is no restriction.
- flags (int) - Integer bitfield in which the following flag bits may be set:
    - OFPFF_SEND_FLOW_REM - Send flow removed message to the controller when rule expires
    - OFPFF_CHECK_OVERLAP - Check for overlapping entries when installing. If one exists, then an error is send to controller
    - OFPFF_EMERG - Consider this flow as an emergency flow and only use it when the switch controller connection is down.
- actions (list) - actions are defined below, each desired action object is then appended to this list and they are executed in order.
- match (ofp_match) - the match structure for the rule to match on (see below).

Example: Installing a table entry

```
# Traffic to 192.168.101.101:80 should be sent out switch port 4

# One thing at a time...
msg = of.ofp_flow_mod()
msg.priority = 42
msg.match.dl_type = 0x800
msg.match.nw_dst = IPAddr("192.168.101.101")
msg.match.tp_dst = 80
msg.actions.append(of.ofp_action_output(port = 4))
self.connection.send(msg)

# Same exact thing, but in a single line...
self.connection.send( of.ofp_flow_mod( action=of.ofp_action_output( port=4
                                       priority=42,
                                       match=of.ofp_match( dl_type=0x800,
                                                           nw_dst="192.168.
                                                           tp_dst=80 )))
```

Example: Clearing tables on all switches

```
# create ofp_flow_mod message to delete all flows
# (note that flow_mods match all flows by default)
msg = of.ofp_flow_mod(command=of.OFPFC_DELETE)

# iterate over all connected switches and delete all their flows
for connection in core.openflow.connections: # _connections.values() before
  connection.send(msg)
  log.debug("Clearing all flows from %s." % (dpidToStr(connection.dpid),))
```

ofp_packet_out - Sending packets from the switch

The main purpose of this message is to instruct a switch to send a packet (or enqueue it). However it can also be useful as a way to instruct a switch to discard a buffered packet (by simply not specifying any actions).

attribute	type	default	notes
buffer_id	int/None	None	ID of the buffer in which the packet is stored at the datapath. If you're not resending a buffer by ID, use None.
in_port	int	OFPP_NONE	Switch port that the packet arrived on if resending a packet.
actions	list of ofp_action_XXXX	[ ]	If you have a single item, you can also specify this using the named parameter "action" of the initializer.
data	bytes / ethernet / ofp_packet_in	''	The data to be sent (or None if sending an existing buffer via its buffer_id). If you specify an `ofp_packet_in` for this, `in_port`, `buffer_id`, and `data` will all be set correctly – this is the easiest way to resend a packet.

Note:

If you receive an `ofp_packet_in` and wish to resend it, you can simply use it as the `data` attribute.