Working with a controller

Marco Ruffini: marco.ruffini@tcd.ie

Lecture content

- POX controller structure and commands
- Packets and encapsulation
- ARP
- Learning switch

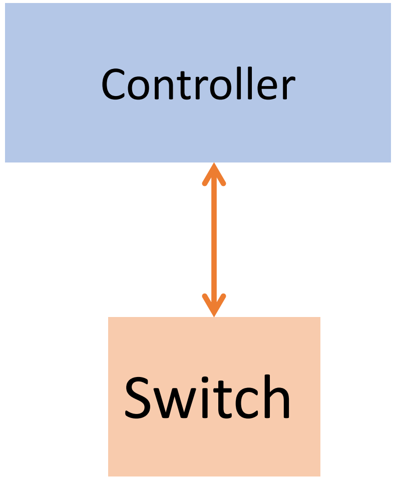
Working with a Python Controller

- We will use POX, a simple Python controller available in the mininet image.
- Packet manipulation: receive, send out, send flow
- ARP, ICMP, how to learn ports
- Packet stripping (payload)

- POX documentation: https://noxrepo.github.io/pox-doc/html/
- Most important libraries:
  - **libopenflow_01.py - https://github.com/att/pox/blob/master/pox/openflow/libopenflow_01.py**
  - **Packet classes - https://github.com/att/pox/tree/master/pox/lib/packet**

Connecting controller and switch

First thing is to initialize the Controller behaviour setting listeners for at least two actions



def launch ():

core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)

core.openflow.addListenerByName("PacketIn", _handle_PacketIn)

-> defines listners functions for ConnectionUp and Packet In

def _handle_ConnectionUp (event):

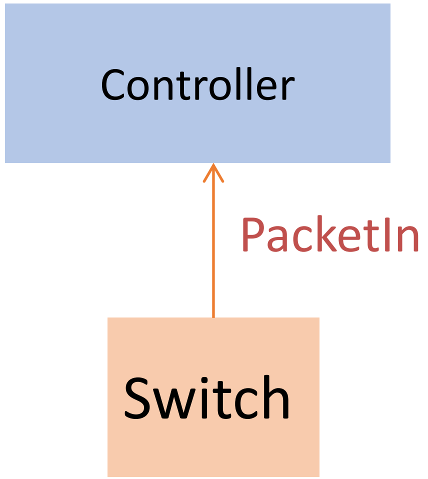
msg = of.ofp_flow_mod(command = of.OFPFC_DELETE)

event.connection.send(msg)

→ If a new connection is detected, by default we delete all active flow entries in the switch (resetting the switch)

Receiving a Packet

- The switch sends a PacketIn to the controller
  - because there is no matching flow in the switch
  - because it's the result of a precise action
- The information in the PacketIn message includes:
  - Input port
  - Reason (no-match, action)
  - Packet data
  - Buffer-id for packet



def _handle_PacketIn (event):

dpid = event.connection.dpid -> defines the connection (switch) from which the packet came in

sw=dpidToStr(event.dpid) -> store it as readable string

inport = event.port -> shows the input port where the packet arrived into the switch

packet = event.parsed -> parses the packet (this contains all packet headers and payload)

Sending a Packet

- The controller sends a data packet out to the switch
  - because it's the same packet that arrived in earlier on
  - it might create a new packet for operating a protocol

def _handle_PacketIn (event):

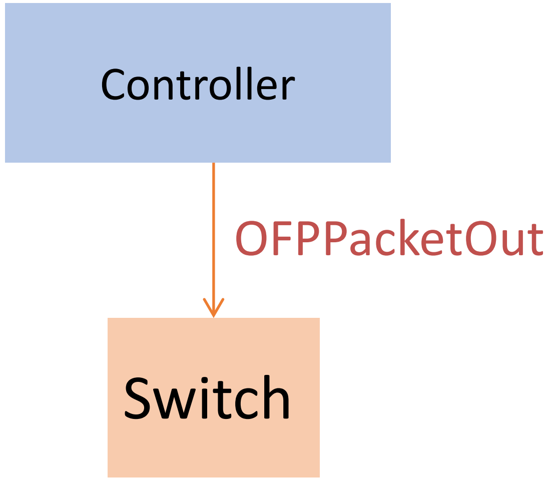
…

msg = of.ofp_packet_out(data = event.ofp) → creates a new message for the data plane

msg.actions.append(of.ofp_action_output(port = …))

→ add some action to the packet (i.e., which port to send out to)

event.connection.send(msg) -> sends the message (packet) out to the switch



Installing a Rule

- Install a flow rule into the switch:
  - because I want to make a rule permanent into the switch

msg = of.ofp_flow_mod() → create a message for the switch table (control message rather than data plane message)

msg.priority=… -> define message priority (if a packet matches two rules the higher priority will be considered)

msg.match.dl_dst = … -> add packet Ethernet destination address

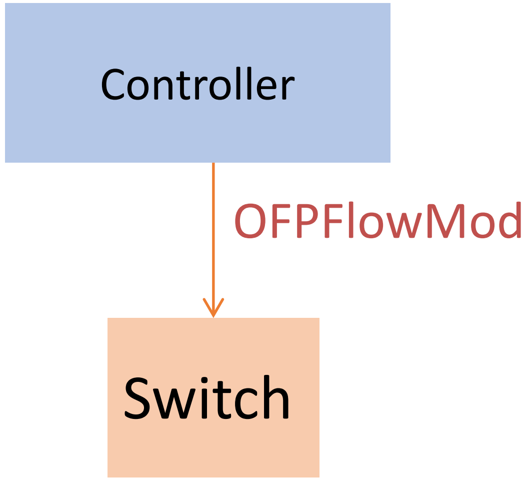
msg.match.dl_src = … -> add packet Ethernet source address

msg.hard_timeout = … -> add hard timeout (rule is deleted in x seconds from installation)

msg.soft_timeout = … -> add soft timeout (rule is deleted after x seconds from the last time a packet matched the rule)

msg.actions.append(of.ofp_action_output(port = …)) -> add output port

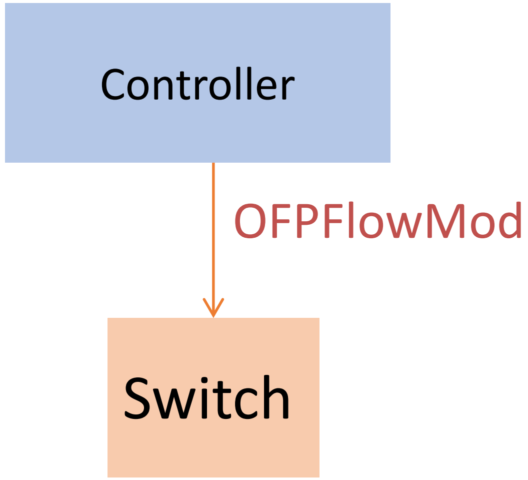
event.connection.send(msg) -> sends the message (packet) out to the switch



Matching a packet to a rule

msg.match.dl_dst = …

msg.match.dl_src = …



Other possibilities for *…match.*

in_port -> match on switch port where the packet arrived

dl_type -> match on the protocol type (ethertype)

nw_src -> match on network (IP) source address

nw_dst -> match on network (IP) destination address

nw_proto -> match on type of IP protocol

tp_src -> match on TCP/UDP source port

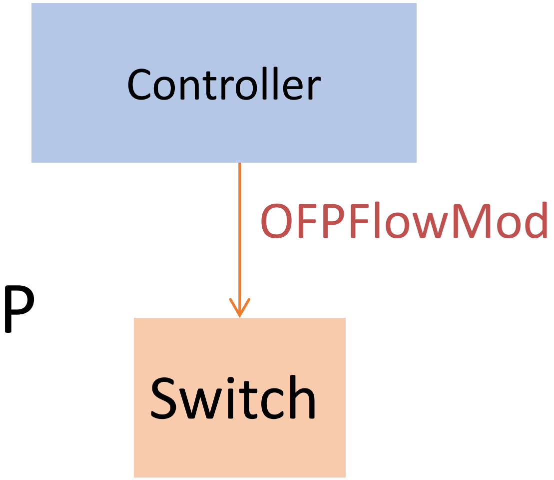
tp_dst -> match on TCP/UDP destination port

Matching to an IP network mask

match.nw_src = ”134.226.0.0/16"

match.nw_src = (IPAddr("134.226.0.0"), 16)

match.nw_src = " 134.226.0.0 /255.255.0.0”



A network mask is a wildcard, it means it matches a group of IP addresses.

For example 134.226.0.0/16 matches all IP addresses whose source (in this case) is 134.226 (independently on the last two fields).

Similarly 134.226.32.0/24 matches all IP addresses whose source is 134.226.32 (independently on the last field).

Actions

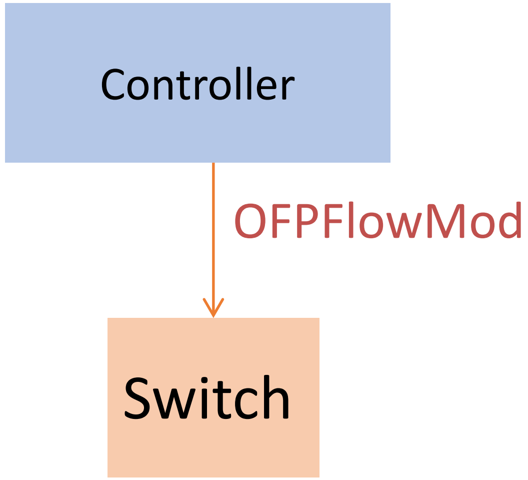
msg.actions.append(of.ofp_action_output(port = (int)))

OFPP_IN_PORT -> send packet to same port where it arrived

OFPP_ALL -> send a copy of the packet out on all ports, except where it came from

OFPP_CONTROLLER -> sends the packet to the controller

OFPP_NONE -> drop the packet

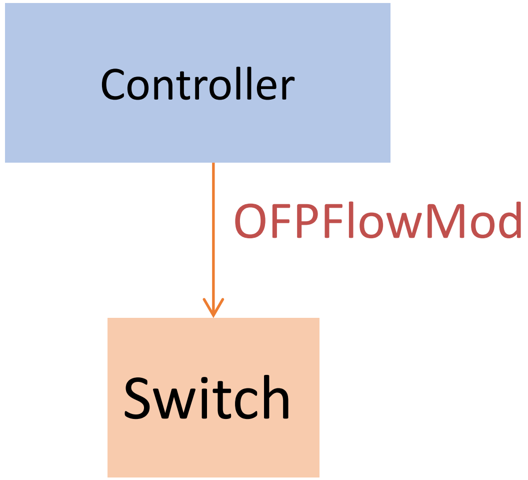


msg.actions.append(of.ofp_action_enqueue (port = (int), queue_id=(int))) *-> send the packet to a specific queue on that port – the queue must be previously configured i.e., with ovs-vsctl*

Actions

You can also do other (simple) actions.

Remember these are flow rules, so the switch will carry out these actions automatically (at line rate) when packets arrive and match a specific flow rule.

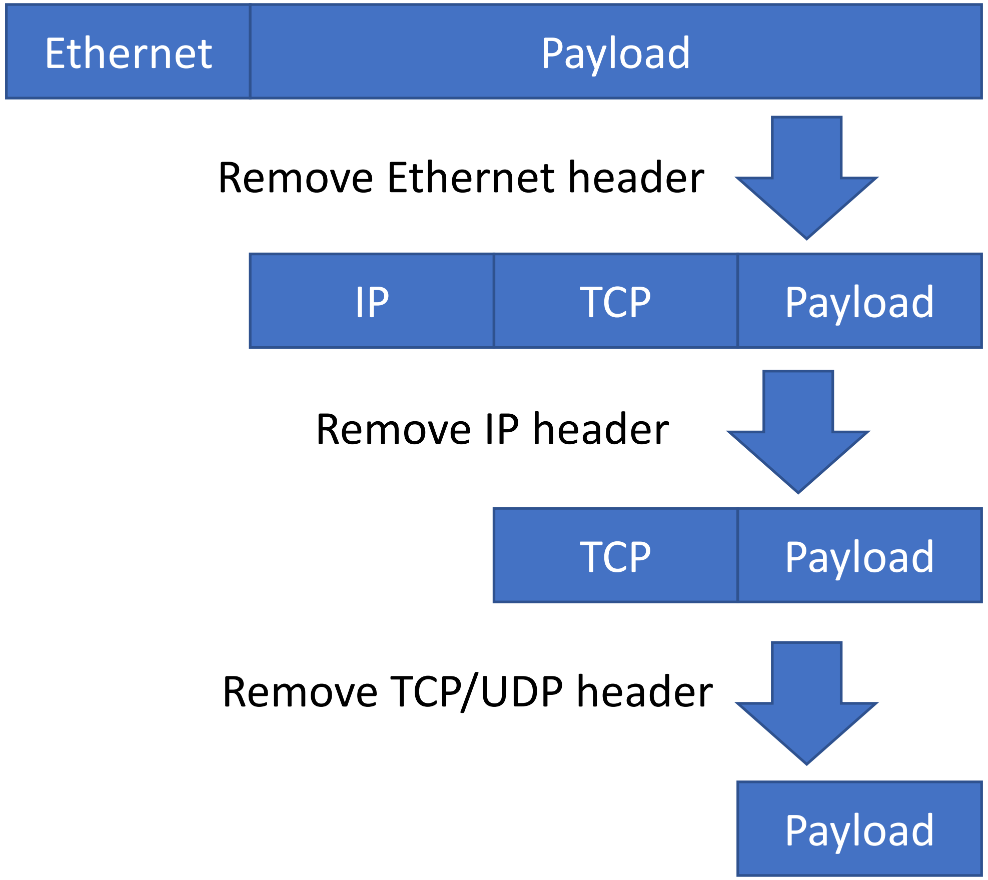


For example, can change/set VLAN ID, change Ethernet or IP addresses, change TCP port.

More complex operations will need to be carried out by the controller (i.e., software level)

Packet encapsulation

- Remember that packets are encapsulated, for example:



Each header will bring specific information about its layer (Ethernet =L2, IP = L3, TCP/UDP = L4)

By decapsulating the lower layer header you get access to the inner header (going up the stack)

While going down you do the reverse (encapsulation)

Example of decapsulation from Ethernet to IP in POX: *ip_packet = eth_packet.payload*

Ethernet packet

From Wikipedia

Layer	Preamble	Start frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence(32-bit CRC)	Interpacket gap
	7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	46-1500 octets	4 octets	12 octets
Layer 2 Ethernet frame			← 64–1522 octets →	**<u>This is the part visible to the L2 protocol</u>**					
Layer 1 Ethernet packet & IPG	← 72–1530 octets →								← 12 octets →

Ethertype

- The Ethertype defines the type of packet (protocol) that is carried by the Ethernet packet
- Some notable examples are:
  - 0x0800 -> IPv4
  - 0x0806 -> ARP
  - 0x86DD -> IPv6
  - 0x8847 -> MPLS (unicast)
- The Ethertype has values above 1536.
- Values below 1500 instead indicate the field is used to state the payload length (in bytes)

IPv4 packet

From Wikipedia

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><th>0</th><th>0</th><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="6">DSCP</td><td colspan="2">ECN</td><td colspan="16">Total Length</td></tr>
<tr><th>4</th><th>32</th><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><th>8</th><th>64</th><td colspan="8">Time To Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><th>12</th><th>96</th><td colspan="32">Source IP Address</td></tr>
<tr><th>16</th><th>128</th><td colspan="32">Destination IP Address</td></tr>
<tr><th>20</th><th>160</th><td colspan="32" rowspan="3">Options (if IHL > 5)</td></tr>
<tr><th>⋮</th><th>⋮</th></tr>
<tr><th>56</th><th>448</th></tr>
</table>

<u>Some notable Protocols:</u>

- 0x01 -> ICMP (ping)
- 0x06 -> TCP
- 0x08 ->EGP Exterior gateway protocol
- 0x09 -> IGP Interior gateway protocol
- 0x11 -> UDP
- 0x29 -> IPv6 encapsulation

<u>Meaning of some of the fields:</u>

- Internet Header Length (IHL): size of IP header
- Differentiated Services Code Point (DSCP): used for packet priority (QoS)
- Explicit Congestion Notification (ECN): notification of network congestion
- Identification: fragment identification
- Flags: related to fragmentation

IP and TCP packet fields

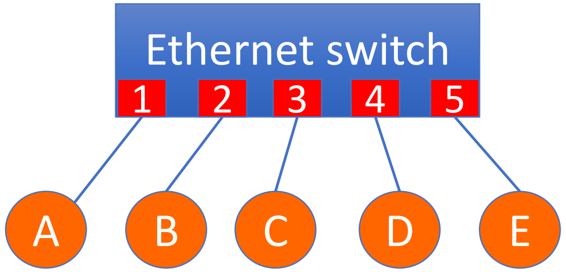
- *ip_packet = eth_packet.payload*
  - *ip_packet.protocol* ➔ *protocol type, e.g. ip.ICMP_PROTOCOL, ip.TPC_PROTOCOL, ip.UDP_PROTOCOL, etc..*
  - *ip_packet.srcip* ➔ *ip address of source node*
  - *ip_packet.dstip* ➔ *ip address of destination node*
  - *ip_packet.ttl* ➔ *time to live field of the IP header*
  - …
- If the packet is a TCP packet:
  - tcp_packet=ip_packet.payload
  - tcp_packet.srcport ➔ this is the source port for the TCP protocol
  - tcp_packet.dstport ➔ this is the destination port for the TCP protocol
  - …

How does a host know the MAC address of a given host?

- The Address Resolution Protocol - https://en.wikipedia.org/wiki/Address_Resolution_Protocol
- It's a protocol that asks around what is the Ethernet address associated to a given IP address.
- It's needed because typically applications make use of IP addresses to reach a given destination (a network address)
- However, the actual transmission is carried out by the Layer 2 (and 1, which has its own independent addressing system).
- The host sends an ARP request: "Who has IP address 134.225.23.4"
  - This is a broadcast address at layer 2 (i.e. within the subnet)
- The other host that has that IP address replies with its Ethernet address

Ethernet learning switch

- Typical Ethernet configurations use switches ➔ star topology



- The switch builds a switching table from the messages it receives
  - Initially the forwarding table is empty:

Dest	Port

  - A sends a message to C ➔ The switch sees a message from port 1 with source A and Dest C
  - C is not on the forwarding table, so the switch broadcasts the message to all ports except 1
  - The switch learns A is reachable through port 1 and updates the table:

Dest	Port
A	1

  - As other messages are received from other nodes, the switching table is updated
  - The entries in the table expire after some time if no message is received from that node

  - The exchange of ARP messages also helps to build up the switching table