Protocolos y Tecnologías para Servicios Móviles y Multimedia
E.T.S. Telecomunicación
Máster en Ingeniería de Telecomunicación
Curso 2022/2023

Práctica 3: Creación de reglas en switch SDN con el controlador POX

El objetivo de esta práctica es realizar modificaciones en aplicaciones Pyhton de SDN Hub para optimizar su funcionamiento. Nota: se recomienda hacer una copia de seguridad de las aplicaciones que se van a usar y realizar las modificaciones oportunas sobre dicha copia.

1. Modificar el balanceador (aplicación tutorial_stateless_lb) para que:

   a. Se envíen 3 peticiones seguidas hacia el mismo servidor

   b. Detecte si un servidor web ha caído (aunque el ordenador sigue conectado a la red) y, en ese caso, eliminarlo de la lista de servidores del pool.

2. Inspirarse en el balanceador de carga para crear una aplicación que implemente un traductor de direcciones (NAT).

Se pide como resultado:

Nuevo código Python y un documento con capturas de pantallas y explicaciones de lo que ocurre a nivel de switch, hosts e intercambio entre switch-controlador. Se recomienda documentar todos los intentos.

Material de ayuda

Mensajes Openflow más relevantes:

HELLO, ECHO: confirma inicio y funcionamiento de conexión
FEATURES: Solicitud y envío de características del switch
STATS: Cuantificación del tráfico en el switch
SET_CONFIG: Solicitud y confirmación de que se han configurado parámetros
BARRIER: Solicitud y confirmación de que se han ejecutado las acciones pendientes
PACKET_IN: Transferencia de paquete completo al controlador
PACKET_OUT: Envío de un paquete a un puerto concreto del switch
FLOW_MOD: Crea reglas en el switch

Ver reglas activas en el switch como resultado de FLOW_MOD

sudo ovs-ofctl -O OpenFlow10 dump-tables s1 | more

Para modificar las tablas hay que usar el API de POX

https://noxrepo.github.io/pox-doc/html/

(a continuación se reproduce un extracto de ese tutorial)

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()    # CREACIÓN DEL MENSAJE OPENFLOW DE UN TIPO CONCRETO
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="192.168.101.101",
                                                           tp_dst=80 )))
```

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).

OpenFlow Actions (part..)

OpenFlow actions are applied to packets that match a rule installed at the datapath. The code snippets found here can be found in libopenflow_01.py in pox/openflow.

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.

Match Structure

- OpenFlow defines a match structure – `ofp_match` – which enables you to define a set of headers for packets to match against. You can either build a match from scratch, or use a factory method to create one based on an existing packet.
- The match structure is defined in pox/openflow/libopenflow_01.py in class `ofp_match`. Its attributes are derived from the members listed in the OpenFlow specification, so refer to that for more information, though they are summarized in the table below.
- `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")
```

- ➢ Para analizar el contenido ethernet transportado en PACKET_IN, ver apartado Working with packets: **pox.lib.packet** del tutorial.

- ➢ Todos los detalles de código del API en pox/openflow/libopenflow_01.py