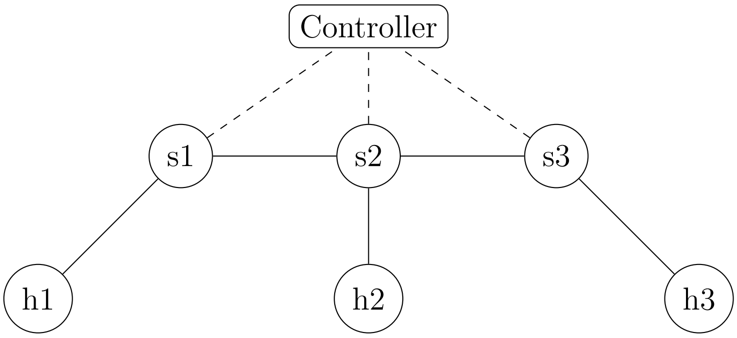
1

Switch con Ryu

Esercizio 1.1 Realizzare un hub usando regole openflow precaricate. Testarlo su una rete lineare con 3 nodi.

Soluzione Vediamo l'implementazione in python del controller (file `hub.py`). La topologia è la seguente:



Per creare la topologia:

```
sudo mn --topo=linear,3 --controller=remote
```

Bisogna specificare il controllore remoto e lanciare il controllore manualmente con il comando:

```
ryu-manager hub.py
```

L'idea è semplice caricare nella tabella openflow di s1, s2, s3 la seguente regola di default:

Priority	Match	Action
0	*	output FLOOD

Segue lo script `hub1.py`.

hub1.py

```
# Implementazione openflow di un hub
#
# In ogni switch viene caricata un'unica regola
# di default (table miss) con azione flooding

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3

# Classe principale, derivata da RyuApp
class PolimiHub(app_manager.RyuApp):
    # usiamo openflow 1.3
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    # Registriamo un handler dell'evento Switch Features
    # Il messaggio Switch Features e' inviato dallo switch
    # quando si registra al controllore
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures,
    ↪   CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Definizione della regola di default
        # priorita' 0
        # match di tutti i pacchetti
        # azione FLOOD
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_FLOOD)]
        inst = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_APPLY_ACTIONS,
                actions
            )
        ]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            priority=0,
            match=match,
            instructions=inst
        )
        datapath.send_msg(mod)
```

Esercizio 1.2 Realizzare un hub predendo decisioni al controller. Testarlo su una rete lineare con 3 nodi. In attesa della decisione, non bufferizzare i pacchetto allo switch.

Soluzione Nello switch carichiamo una regola che mandi tutto al controllore; per disattivare il buffering dei pacchetti specifichiamo come dimensione dei pacchetti da inviare al controllore la costante `ofproto.OFPCML_NO_BUFFER`. Attenzione che in OpenVSwitch questo non è possibile per la table miss (priorità 0), quindi caricheremo una regola con priorità 1:

Priority	Match	Action
1	*	output CONTROLLER,NO BUFFER

Segue lo script `hub.py`.

hub2.py

```
# Implementazione openflow di un hub tramite controller
#
# In ogni switch viene caricata un'unica regola
# di default (table miss) con azione di invio al controller
# dell'intero pacchetto. Il controller risponde con una
# packet out con azione flood
#
# NOTA: OpenVSwitch ignora l'opzione OFPCML_NO_BUFFER
# nelle regole table miss (priorita' 0); pertanto,
# carichiamo una regola con priorita' 1
```

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER,
↪  MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3

class PolimiHub(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures,
    ↪  CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        match = parser.OFPMatch()
        actions = [
            parser.OFPActionOutput(
                ofproto.OFPP_CONTROLLER,
                ofproto.OFPCML_NO_BUFFER
            )
        ]
        inst = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_APPLY_ACTIONS,
                actions
            )
        ]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            priority=1,
            match=match,
            instructions=inst
        )
        datapath.send_msg(mod)

    # Registriamo un handler dell'evento Packet In
    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Per come abbiamo scritto le regole nello switch
        # i pacchetti non devono essere bufferizzati allo switch
        assert msg.buffer_id == ofproto.OFP_NO_BUFFER

        # Recuperiamo dai metadati del pacchetto
        # la porta di ingresso allo switch
        in_port = msg.match['in_port']

        actions = [
            parser.OFPActionOutput(
                ofproto.OFPP_FLOOD
            )
        ]

        out = parser.OFPPacketOut(
            datapath=datapath,
            buffer_id=msg.buffer_id,
            in_port=in_port,
            actions=actions,
            data=msg.data
        )
        datapath.send_msg(out)
```

hub3.py

```
# Implementazione openflow di un hub tramite controller
#
# In ogni switch viene caricata un'unica regola
# di default (table miss) con azione di bufferizzazione
# del pacchetto e invio al controller dell'intestazione
# Il controller risponde con una  packet out con azione flood

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER,
↪  MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3

class PolimiHub(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]


    @set_ev_cls(ofp_event.EventOFPSwitchFeatures,
    ↪   CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        match = parser.OFPMatch()
        actions = [
            parser.OFPActionOutput(
                ofproto.OFPP_CONTROLLER,
                128
            )
        ]
        inst = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_APPLY_ACTIONS,
                actions
            )
        ]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            priority=0,
            match=match,
            instructions=inst
        )
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Per come abbiamo scritto le regole nello switch
        # i pacchetti devono essere bufferizzati allo switch
        assert msg.buffer_id != ofproto.OFP_NO_BUFFER

        in_port = msg.match['in_port']

        actions = [
            parser.OFPActionOutput(
                ofproto.OFPP_FLOOD
            )
        ]

        out = parser.OFPPacketOut(
            datapath=datapath,
            buffer_id=msg.buffer_id,
            in_port=in_port,
            actions=actions,
            data=None)
        datapath.send_msg(out)
```

switch1.py

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER,
↪  MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet, ethernet

# This implements a learning switch in the controller
# The switch sends all packet to the controller
# The controller implements the MAC table using a python dictionary

class PsrSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(PsrSwitch, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    # execute at switch registration
```

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures,
 ↪  CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    self.mac_to_port.setdefault(datapath.id, {})

    # match all packets
    match = parser.OFPMatch()
    # send to controller
    actions = [
        parser.OFPActionOutput(
            ofproto.OFPP_CONTROLLER,
            128
        )
    ]
    inst = [
        parser.OFPInstructionActions(
            ofproto.OFPIT_APPLY_ACTIONS,
            actions
       )
   ]
    mod = parser.OFPFlowMod(
        datapath=datapath,
        priority=0,
        match=match,
        instructions=inst
    )
    datapath.send_msg(mod)


@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']
    dpid = datapath.id
```

```
        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)

        assert eth is not None

        dst = eth.dst
        src = eth.src

        self.mac_to_port[dpid][src] = in_port

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

#        self.logger.info("packet in %s %s %s %s %s", dpid, src,
↪   dst, in_port, out_port)

        actions = [
            parser.OFPActionOutput(out_port)
        ]

        assert msg.buffer_id != ofproto.OFP_NO_BUFFER

        out = parser.OFPPacketOut(
            datapath=datapath,
            buffer_id=msg.buffer_id,
            in_port=in_port,
            actions=actions,
            data=None
        )
        datapath.send_msg(out)
```

switch2.py

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER,
↪   MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet, ethernet

# This implements a learning switch in the controller
# The switch sends all packet to the controller
# The controller implements the MAC table using a python dictionary
# If the MAC dst is known, add rule to the switch
class PsrSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(PsrSwitch, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    # execute at switch registration
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures,
    ↪   CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        self.mac_to_port.setdefault(datapath.id, {})

        match = parser.OFPMatch()
        actions = [
            parser.OFPActionOutput(
                ofproto.OFPP_CONTROLLER,
                128
            )
        ]
        inst = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_APPLY_ACTIONS,
                actions
            )
        ]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            priority=0,
            match=match,
            instructions=inst
        )
        datapath.send_msg(mod)


    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
        dpid = datapath.id

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)

        assert eth is not None

        dst = eth.dst
        src = eth.src

        self.mac_to_port[dpid][src] = in_port

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

#        self.logger.info("packet in %s %s %s %s %s", dpid, src,
↪   dst, in_port, out_port)

        actions = [
            parser.OFPActionOutput(out_port)
        ]

        assert msg.buffer_id != ofproto.OFP_NO_BUFFER

        if out_port != ofproto.OFPP_FLOOD:
          # install a flow and send the packet
            match = parser.OFPMatch(
                eth_src=src,
                eth_dst=dst
            )
            inst = [
                parser.OFPInstructionActions(
                    ofproto.OFPIT_APPLY_ACTIONS,
                    actions
                )
            ]
            ofmsg = parser.OFPFlowMod(
                datapath=datapath,
                priority=1,
                match=match,
                instructions=inst,
                buffer_id=msg.buffer_id
            )
        else:
            # only send the packet
            ofmsg = parser.OFPPacketOut(
                datapath=datapath,
                buffer_id=msg.buffer_id,
                in_port=in_port,
                actions=actions,
                data=None
            )

        datapath.send_msg(ofmsg)
```

switch3.py

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER,
↪  MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet, ethernet

# This implements a learning switch in the of switch
# The switch uses two tables:
#   table 0 for source address
#       if source present -> go to table 1
#       if source mac missing -> send to controller & go to table 1
#   table 1 for destination address
#       if destination present -> send to destination
#       if destination missing -> flood
# Controller adds source mac to both tables
class PsrSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    # execute at switch registration
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures,
    ↪   CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        match = parser.OFPMatch()
        actions = [
            parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,128)
        ]
        inst = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_APPLY_ACTIONS,
                actions
            ),
            parser.OFPInstructionGotoTable(1)
        ]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            table_id=0,
            priority=0,
            match=match,
            instructions=inst
        )
        datapath.send_msg(mod)

        match = parser.OFPMatch()
        actions = [
            parser.OFPActionOutput(ofproto.OFPP_FLOOD)
        ]
        inst = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_APPLY_ACTIONS,
                actions
            )
        ]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            table_id=1,
            priority=0,
            match=match,
            instructions=inst
        )
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
        dpid = datapath.id

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)

        assert eth is not None

        src = eth.src

#        self.logger.info("packet in %s %s %s", dpid, src, in_port)

        # add source address to table 0
        # to stop sending to the controller
        match = parser.OFPMatch(eth_src=src)
        inst = [
            parser.OFPInstructionGotoTable(1)
        ]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            table_id=0,
            priority=1,
            match=match,
            instructions=inst
        )
        datapath.send_msg(mod)

        # add source address to table 1
        # to send to the correct port
        match = parser.OFPMatch(eth_dst=src)
        actions = [
            parser.OFPActionOutput(in_port)
        ]
        inst = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_APPLY_ACTIONS,
                actions
            )
        ]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            table_id=1,
            priority=1,
            match=match,
            instructions=inst
        )
        datapath.send_msg(mod)
```