1

Modifica dei pacchetti con Ryu

Le tabelle openflow possono effettuare match su vari campi del pacchetto:

in_port
eth_src/eth_dst
eth_type
ip_proto
ipv4_src/ipv4_dst
tcp_src/tcp_dst
udp_src/udp_dst
mpls_label
ipv6_src/ipv6_dst/ipv6_flabel

Esempio in ryu:

```
match = parser.OFPMatch(eth_type=0x0800,ipv4_src="10.0.0.2")
```

Per effettuare il match su alcuni campi occorre che il match comprenda **anche** altri campi:

Match field	Prerequisito
ip_proto	eth_type=ETH_TYPE_IP eth_type=ETH_TYPE_IPV6
ipv4_src/ipv4_dst	eth_type=ETH_TYPE_IP
tcp_src/tcp_dst	ip_proto=IPPROTO_TCP
udp_src/udp_dst	ip_proto=IPPROTO_UDP
ipv6_src/ipv6_dst/ipv6_flabel	eth_type=ETH_TYPE_IPV6
mpls_label	eth_type=ETH_TYPE_MPLS

Prima della `ActionOutput` è possibile specificare una o più azioni `SetField` che sovrascrivono ciascuna un campo. I nomi dei campi sono gli stessi del'operazione di match. Esempio in ryu:

```
actions = [
 parser.OFPActionSetField(tcp_src=80),
 parser.OFPActionOutput(ofproto.OFPP_FLOOD)
 ]
```

Esercizio 1.1 Considerare una rete lineare con 3 nodi. Nello switch numero 2 implementare un meccanismo di port translation che, per tutte le connessioni TCP verso h3 porta 80, inoltri verso h2 porta 8080. Implementare un meccanismo proattivo e considerare MAC degli host noti e tabelle ARP prepopolate. Mandare tutto il resto del traffico in flooding.

Soluzione Possiamo usare una coppia di regole proattive in s2.

Priority	Match	Action
1	eth_type = ETH_TYPE_IP, ip_dst = 10.0.0.3, proto = IPPROTO_TCP, tcp_dst = 80	SetField(eth_dst=00:00:00:00:00:02), SetField(ip_dst=10.0.0.2), SetField(tcp_dst=8080), output(1)
1	eth_type = ETH_TYPE_IP, ip_src = 10.0.0.2, proto = IPPROTO_TCP, tcp_src = 8080	SetField(eth_src=00:00:00:00:00:03), SetField(ip_src=10.0.0.3), SetField(tcp_src=80), output(2)
0	*	output FLOOD

hubrewrite1.py

```
# Questo switch presuppone
# mn --mac --arp --topo linear,3 --controller=remote
# Lo switch 2 dirotta il traffico tcp a h3:80 verso h2:8080

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
from ryu.ofproto import inet, ether

class PolimiHubRewrite(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

        # table miss flooding per tutti gli switch
        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)
```

```
        if (datapath.id == 2):

            match = parser.OFPMatch(
                eth_type=ether.ETH_TYPE_IP,
                ipv4_dst="10.0.0.3",
                ip_proto=inet.IPPROTO_TCP,
                tcp_dst=80)
            # send broadcast
            actions = [
                parser.OFPActionSetField(
                    eth_dst="00:00:00:00:00:02"
                    ),
                parser.OFPActionSetField(ipv4_dst="10.0.0.2"),
                parser.OFPActionSetField(tcp_dst=8080),
                parser.OFPActionOutput(1)
                ]
            inst = [
                parser.OFPInstructionActions(
                    ofproto.OFPIT_APPLY_ACTIONS,
                    actions
                )
            ]
            mod = parser.OFPFlowMod(
                datapath=datapath,
                priority=1,
                match=match,
                instructions=inst
            )
            datapath.send_msg(mod)

            match = parser.OFPMatch(
                eth_type=ether.ETH_TYPE_IP,
                ipv4_src="10.0.0.2",
                ip_proto=inet.IPPROTO_TCP,
                tcp_src=8080)
            # send broadcast
            actions = [
                parser.OFPActionSetField(
                    eth_src="00:00:00:00:00:03"
                    ),
                parser.OFPActionSetField(ipv4_src="10.0.0.3"),
                parser.OFPActionSetField(tcp_src=80),
```

```
            parser.OFPActionOutput(2)
            ]
        inst = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_APPLY_ACTIONS,
                actions
            )
        ]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            priority=1,
            match=match,
            instructions=inst
        )
        datapath.send_msg(mod)
```

Esercizio 1.2 Come il precedente passando dal controller.

Soluzione

hubrewrite2.py

```
# Questo switch presuppone
# mn --mac --arp --topo linear,3 --controller=remote
# Lo switch 2 dirotta il traffico tcp a h3:80 verso h2:8080

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
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import tcp
# from array import array

class PolimiHubRewrite(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

        # manda al controllore tutti i pacchetti
        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)

    @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']
        out_port = ofproto.OFPP_FLOOD

        pkt = packet.Packet(data=msg.data)

        pkt_ethernet = pkt.get_protocol(ethernet.ethernet)
        pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
        pkt_tcp = pkt.get_protocol(tcp.tcp)

        if (datapath.id == 2
            and pkt_tcp is not None
            and pkt_ipv4.dst == '10.0.0.3'
            and pkt_tcp.dst_port == 80):
```

```
            pkt_ethernet.dst='00:00:00:00:00:02'
            pkt_ipv4.dst='10.0.0.2'
            pkt_tcp.dst_port=8080
            pkt_tcp.csum=0 # il checksum va ricalcolato
            pkt.serialize()

            out_port = 1

        elif (datapath.id == 2
            and pkt_tcp is not None
            and pkt_ipv4.src == '10.0.0.2'
            and pkt_tcp.src_port == 8080):

            pkt_ethernet.src='00:00:00:00:00:03'
            pkt_ipv4.src='10.0.0.3'
            pkt_tcp.src_port=80
            pkt_tcp.csum=0 # il checksum va ricalcolato
            pkt.serialize()

            out_port = 2

        actions = [parser.OFPActionOutput(out_port)]
        data = pkt.data
        out = parser.OFPPacketOut(
            datapath=datapath,
            buffer_id=msg.buffer_id,
            in_port=in_port,
            actions=actions,
            data=data)
        datapath.send_msg(out)
```

Esercizio 1.3 In s1 tutto il traffico UDP/DNS per h2 deve essere inviato invece ad h3. In s2 tutto il traffico TCP/http per h2 deve essere mappato su 8080.

In s1 e s2 aggiungere dinamicamente una regola per ogni connessione TCP che invii i pacchetti sulla porta corretta. Il restante traffico deve essere inviato broadcast.