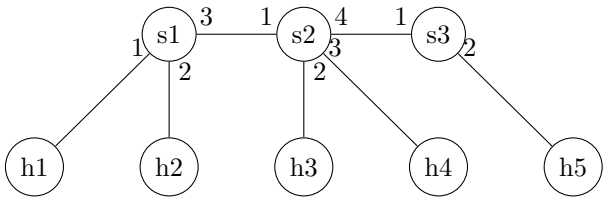
Temi d'esame

Esercizio 1

Si consideri la rete in figura. I nodi h1,..., hn contengono tabelle ARP prepopolate per tutta la rete. Gli indirizzi MAC e IP dei nodi h1,..., hn sono MAC1,..., MACn e IP1,..., IPn. Sui nodi h1,..., hn sono in esecuzione processi in ascolto su tutte le porte TCP e UDP di interesse. Il controllore Ryu è collegato ai nodi s1,..., sm ed è configurato come indicato nel Programma 1.



(a) Scrivere il contenuto delle tabelle openflow dopo l'accensione di tutti i nodi.

Soluzione:

s1

Priority	Match	Action	pkt cnt
0	*	output controller	0
1	eth_src = MAC1 eth_type = 0x800	output flood	0

s2

Priority	Match	Action	pkt cnt
0	*	output controller	0
1	eth_dst = MAC5 eth_type = 0x800 ip_proto = 17	set eth_dst=MAC4 set ip_dst=IP4 ouput 3	0

s3

Priority	Match	Action	pkt cnt
0	*	output controller	0

L'host h1, porta 123 invia un datagramma UDP a h5, porta 123.

(b) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s1.

Soluzione:
s1 → h2, s2: MAC1, MAC5, IPv4, IP1, IP5, TCP, 123, 123

(c) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s2 e s3.

Soluzione: s2 → h4: MAC1, MAC4, IP1, IP4, IPv4, 123, 123

L'host h3, porta 5432 invia una richiesta di apertura di connessione a h2, porta 80.

Programma 1 (1/2) – Switch Features

```
# intestazione omessa
@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=0,
                            match=match, instructions=inst)
    datapath.send_msg(mod)

    if (datapath.id == 1):
        match = parser.OFPMatch(
            eth_src=MAC1,
            eth_type=0x0800)    # IPv4
        actions = [ parser.OFPActionOutput(ofproto.OFPP_FLOOD) ]
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=1,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

    if (datapath.id == 2):
        match = parser.OFPMatch(
            eth_dst=MAC5,
            eth_type=0x0800,    # IPv4
            ip_proto=17)        # UDP
        actions = [
            parser.OFPActionSetField(eth_dst=MAC4),
            parser.OFPActionSetField(ip_dst=IP4),
            parser.OFPActionOutput(3)
            ]
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=1,
                                match=match, instructions=inst)
        datapath.send_msg(mod)
```

Programma 1 (2/2) – 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
    in_port = msg.match['in_port']
    pkt = packet.Packet(msg.data)
    pkt_eth = pkt.get_protocol(eth.eth)
    pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
    pkt_udp = pkt.get_protocol(udp.udp)
    pkt_tcp = pkt.get_protocol(tcp.tcp)

    if (datapath.id == 2) and (pkt_tcp is not None):
        actions = [parser.OFPActionOutput(ofproto.OFPP_FLOOD)]
        out = parser.OFPPacketOut(datapath=datapath, in_port=in_port,
                                  actions=actions, data=data)
        datapath.send_msg(out)

        match = parser.OFPMatch(
            eth_dst=pkt_eth.src,
            eth_type=0x0800,    # IPv4
            ip_proto=6)         # TCP
        actions = [ parser.OFPActionOutput(in_port) ]
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=1,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

        return
```

(d) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s2.

Soluzione:
s2 → C: packet_in(MAC3,MAC2, IPv4, IP3, IP2, TCP, 5432, 80)
s2 → s1,s3,h4: MAC3,MAC2, IPv4, IP3, IP2, TCP, 5432, 80

Il server risponde all'apertura di connessione

(e) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s2.

Soluzione:
Il pacchetto SYN è scartato in s1 e non arriva al server.

(f) Scrivere il contenuto delle tabelle openflow.

Soluzione:

s1

Priority	Match	Action	pkt cnt
0	*	output controller	0
1	eth_src = MAC1 eth_type = 0x800	output flood	1

s2

Priority	Match	Action	pkt cnt
0	*	output controller	0
1	eth_dst = MAC5 eth_type = 0x800 ip_proto = 17	set eth_dst=MAC4 set ip_dst=IP4 output 3	1
1	eth_dst = MAC3 eth_type = 0x800 ip_proto = 6	output 2	0

s3

Priority	Match	Action	pkt cnt
0	*	output controller	0

Esercizio 2

Si consideri un filtro di Bloom costituito da un array di 24 bit e 3 funzioni hash. Inizialmente tutti i bit dell'array sono posti a zero. Si assuma che ciascuna posizione all'interno dell'array possa essere selezionata da ciascuna funzione di hash con la medesima probabilità.

(a) Calcolare la probabilità che un dato bit sia posto ad 1 dopo aver effettuato l'inserimento di 3 elementi

(b) Siano le due funzioni di hash definite come $H_1(x) = 2x + 1 \bmod 24$, $H_2(x) = 4x + 1 \bmod 24$, $H_3(x) = 6x + 1 \bmod 24$. Dire se e quali funzioni hash scelte sono buone.

(c) Inizialmente tutti i bit dell'array sono posti a zero. Inserire 3 e 5 nel filtro di Bloom. Mostrare lo stato di ogni bit prima e dopo ogni inserimento.

(d) Conclusi gli inserimenti, verificare l'appartenza al dataset degli elementi 1 e 15 e commentare i risultati ottenuti (vi sono falsi positivi?)

Esercizio 3

Un sistema di backtracking usa filtri di Bloom per tenere traccia dei pacchetti transitati da un router. Il router riceve pacchetti alla velocità di 10 Mbit/s. Si assuma lunghezza minima dei pacchetti pari a 50 byte. Per memorizzare i pacchetti osservati si calcolano 3 funzioni hash di l bit sull'intero pacchetto.

(a) Si assumano funzioni hash ideali di $l = 16$ bit e un intervallo di osservazione di $T = 600\,\text{ms}$. Calcolare la probabilità di falso positivo.

Soluzione: Sia n il numero di inserimenti, k il numero di funzioni hash, m la lunghezza dell'array.

$$n = \frac{10 \cdot 10^6\,\text{bit/s}0{,}6\,\text{s}}{50 \times 8}$$
$$k = 3$$
$$m = 2^{16}$$
$$p = \left(1 - \exp\left(-\frac{kn}{m}\right)\right)^k = 0.12$$

(b) Anziché i singoli pacchetti si vuole effettuare il backtracking di connessioni TCP. Come si deve modificare l'algorito? C'è un impatto sulla probabilità di falso positivo?

Soluzione: Si effettua l'hash della quintupla che definisce il flusso. Poiché il numero di connessioni è minore del numero di pacchetti, n sarà più piccolo e ci saranno meno falsi positivi.

Si considerino le funzioni hash con $l = 3$.

$$h_1(x) = x + 1 \bmod 8$$
$$h_2(x) = 3x + 2 \bmod 8$$
$$h_3(x) = 7x + 3 \bmod 8$$

(c) Si inseriscano nel filtro i valori $x = 0$ e $x = 2$.

Soluzione: Per $x = 0$ si settano i bit 1, 2, 3.
Per $x = 2$ si settano i bit 0, 1, 3.
In totale si settano i bit 0, 1, 2, 3.

(d) Verificare se sono presenti nel filtro gli elementi $x = 5$ e $x = 10$. Quali di questi sono falsi positivi?

Soluzione: Per $x = 0$ si verificano i bit 1, 6. Non presente.
Per $x = 10$ si verificano i bit 0, 1, 3. Presente (falso positivo).

Esercizio 4

Si consideri un'interfaccia gestita con uno scheduler Deficit Round Robin. La velocità di trasmissione è 8 Mbit/s. Al tempo $t = 0$ sono presenti i seguenti pacchetti (è indicata la lunghezza in byte) ordinati dal più recente al più vecchio.

Coda 1 150, 75, 150

Coda 2 50, 150, 225

Coda 3 –

Il quanto è 100 byte per tutte le code. Al tempo $t = 100\,\mu s$ si presenta un pacchetto da 50 byte alla coda 3. Al tempo $t = 200\,\mu s$ si presenta alla coda 3 un pacchetto da 100 byte.

(a) Simulare l'algoritmo DRR indicando ad ogni tempo quali pacchetti concludono la trasmissione.

Soluzione:

Iter	Coda	Deficit	Lunghezza Pacchetto	Fine Trasmissione
1	1	100	–	–
1	2	100	–	–
1	3	–	–	–
2	1	–	150	150
2	2	200	–	–
2	3	50	50	200
3	1	75	75	275
3	2	75	225	500
3	3	–	100	600
4	1	–	150	750
4	2	25	150	900
4	3	–	–	–
5	1	–	–	–
5	2	–	50	950

(b) Considerando dimensione massima dei pacchetti pari a 500 byte e quanto di 100 byte, calcolare il tempo massimo che attende in coda un pacchetto di 50 byte che si presenta alla coda 3 trovandola vuota.

Soluzione: Un pacchetto in trasmissione per la coda 3. Per ciascuna delle code 1 e 2 un deficit accumulato di 499 e un quanto di 100.

$$t_{\max} = 500 + 2(499 + 100) = 1{,}698\,\text{ms}$$

(c) Simulare la stessa interfaccia della prima domanda, ma con uno scheduler a priorità semplice ($1 > 2 > 3$). Dire quanto tempo aspettano i due pacchetti della coda 3 nei due casi di scheduler DRR e a priorità.

Soluzione:

Coda	Lunghezza Pacchetto	Fine Trasmissione
1	150	150
1	75	225
1	150	375
2	225	600
2	150	750
2	50	800
3	50	850
3	100	950

Nel caso DRR il primo pacchetto aspetta 50 µs e il secondo 300 µs.

Nel caso a priorità il primo pacchetto aspetta 700 µs e il secondo 650 µs.

Esercizio 5

Una matrice di commutazione crossbar di uno switch 3×3 usa l'algoritmo Take-a-Ticket (TaT) per lo scheduling delle trasmissioni.

Al tempo $t = 0$ sono presenti i seguenti pacchetti, tutti di lunghezza L, in ordine dal più recente al più vecchio. Ogni pacchetto è etichettato con il numero dell'interfaccia di uscita cui è diretto.

- Ingresso 1: 1, 2, 1
- Ingresso 2: 3, 2, 1
- Ingresso 3: 2, 3, 3

(a) Svolgere l'algoritmo TaT fino allo svuotamento di tutte le code, specificando ad ogni round quali pacchetti sono trasmessi e quali ticket sono assegnati. L'algoritmo di assegnazione dei ticket privilegia sempre l'interfaccia di ingresso con l'identificativo più piccolo.

Soluzione:

Round		Ingresso 1	Ingresso 2	Ingresso 3
1	Assegnazione ticket	(1,1)	(1,2)	(3,1)
	Trasmissione	invio	attesa	invio
2	Assegnazione ticket	(2,1)	–	(3,2)
	Trasmissione	invio	invio	invio
3	Assegnazione ticket	(1,3)	(2,2)	(2,3)
	Trasmissione	invio	invio	attesa
4	Assegnazione ticket	–	(3,3)	–
	Trasmissione	–	invio	invio

Con la notazione (a,b) si intende l'interfaccia di uscita a e il numero di ticket b.

(b) Uno switch 3×3 implementa la seguente variante di TaT. Si i il numero del ticket del pacchetto che inizia la trasmissione su un'interfaccia di uscita. Se ci sono code in ingresso con numero di ticket $\geq i+2$ per quell'interfaccia, scartano il loro pacchetto e chiedono un ticket per il pacchetto successivo.

Considerando traffico in ingresso distribuito uniformemente e code sempre piene, calcolare il numero di pacchetti scartati per ogni intervallo temporale.

Soluzione:

Sia p il numero di pacchetti scartati

$$\Pr(p = 0) = \frac{3!}{3^3}$$
$$\Pr(p = 2) = \frac{3}{3^3}$$
$$\Pr(p = 1) = \frac{3^3 - 3! - 3}{3^3}$$

(c) Dire in cosa consiste il problema del Head-of-Line (HoL) blocking. Spiegare cosa si intende per *virtual output queues* e dire come possono affrontare il problema.

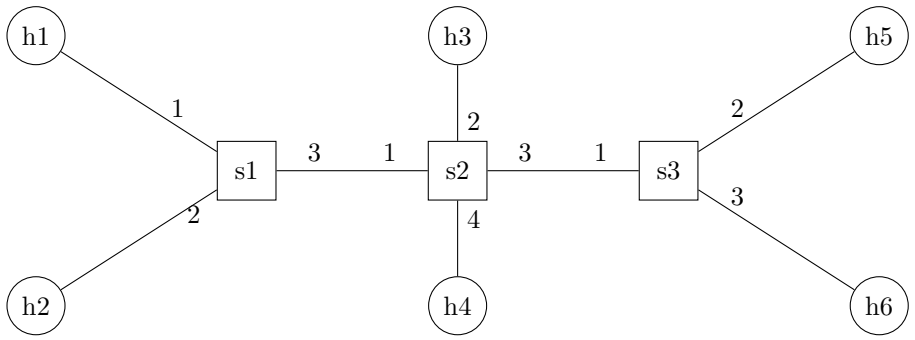
Soluzione: Si ha HoL blocking quando un pacchetto non può essere trasmesso anche se la sua uscita è libera perché si trova in fila di attesa dietro un pacchetto il quale non può essere trasmesso perché la sua uscita è occupata.

Può essere affrontato usando sulle interfacce in ingresso delle code di uscita virtuali.

Esercizio 6

Si consideri la rete in figura. I nodi h1,..., hn contengono tabelle ARP prepopolate per tutta la rete.

Gli indirizzi MAC e IP dei nodi h1,..., hn sono MAC1,..., MACn e IP1,..., IPn. Sui nodi h1,..., hn sono in esecuzione processi in ascolto su tutte le porte TCP e UDP di interesse. Il controllore Ryu è collegato ai nodi s1,..., sm ed è configurato come indicato nel Programma 1. Le interfacce tra gli switch e il controllore non sono indicate in figura.



(a) Scrivere il contenuto delle tabelle openflow dopo l'accensione di tutti i nodi.

Soluzione:

s1

Priority	Match	Action	pkt cnt
0	*	output controller	0
1	eth_dst = MAC1 eth_type = 0x800	output 1	0

s2

Priority	Match	Action	pkt cnt
0	*	output controller	0
1	eth_dst = MAC1 eth_type = 0x800	output 1	0
2	eth_dst = MAC3 eth_type = 0x800 ip_dst = IP3	set eth_dst=MAC4 set ip_dst=IP4 ouput 4	0
2	eth_src = MAC4 eth_type = 0x800 ip_src = IP4	set eth_src=MAC3 set ip_src=IP3 ouput flood	0

s3

Priority	Match	Action	pkt cnt
0	*	output controller	0
1	eth_dst = MAC1 eth_type = 0x800	output 1	0

L'host h5, porta 3456 invia un datagramma UDP a h1, porta 123.

(b) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s1 e s2.

Programma 1 (1/2) – Switch Features

```
@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=0,
                            match=match, instructions=inst)
    datapath.send_msg(mod)

    match = parser.OFPMatch(
        eth_dst=MAC1,
        eth_type=0x0800)    # IPv4
    actions = [ 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)

    if (datapath.id == 2):
        match = parser.OFPMatch(
            eth_dst=MAC3,
            eth_type=0x0800,    # IPv4
            ip_dst=IP3)
        actions = [
            parser.OFPActionSetField(eth_dst=MAC4),
            parser.OFPActionSetField(ip_dst=IP4),
            parser.OFPActionOutput(4)
            ]
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=2,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

        match = parser.OFPMatch(
            eth_src=MAC4,
            eth_type=0x0800,    # IPv4
            ip_src=IP4)
        actions = [
            parser.OFPActionSetField(eth_src=MAC3),
            parser.OFPActionSetField(ip_src=IP3),
            parser.OFPActionOutput(ofproto.OFPP_FLOOD)
            ]
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=2,
                                match=match, instructions=inst)
        datapath.send_msg(mod)
```

Programma 1 (2/2) – 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
    in_port = msg.match['in_port']
    pkt = packet.Packet(msg.data)
    pkt_eth = pkt.get_protocol(eth.eth)
    pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
    pkt_udp = pkt.get_protocol(udp.udp)
    pkt_tcp = pkt.get_protocol(tcp.tcp)

    if (pkt_tcp is not None) and (pkt_tcp.dst==80):
        match = parser.OFPMatch(
            eth_dst=pkt_eth.src,
            eth_type=0x0800,    # IPv4
            ip_proto=6)         # TCP
        actions = [ parser.OFPActionOutput(in_port) ]
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=3,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

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

Soluzione:
$s2 \rightarrow s1$: MAC5 > MAC1 IP5 > IP1 UDP 3456 > 123
$s1 \rightarrow h1$: MAC5,MAC1,IP5,IP1,UDP,3456,123

L'host h1, porta 5432 invia una richiesta di apertura di connessione a h3, porta 123.

(c) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s2.

Soluzione:
$s2 \rightarrow h4$: MAC1,MAC4,IP1,IP4,TCP,5432,123 [SYN]

Il server risponde.

(d) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s2.

Soluzione:
$s2 \rightarrow s1,s3,h3$: MAC3,MAC1,IP3,IP1,TCP,5432,123 [SYN ACK]

L'host h2, porta 5432 invia una richiesta di apertura di connessione a h6, porta 80. Nel caso il pacchetto non arrivi a destinazione ripetere l'invio altre 2 volte.

(e) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s2.

Soluzione:
$s2 \rightarrow \mathcal{C}$: packet_in (MAC2,MAC6,IP2,IP6,TCP,5432,80 [SYN])
$s2 \rightarrow s1,s2,h3,h4$: MAC2,MAC6,IP2,IP6,TCP,5432,80 [SYN]

Il server risponde.

(f) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s2.

Soluzione:
$s2 \rightarrow s1$: MAC6,MAC2,IP6,IP2,TCP,80,5432 [SYN ACK]

(g) Scrivere il contenuto finale delle tabelle openflow.

Soluzione:

s1

Priority	Match	Action	pkt cnt
0	*	output controller	?
1	eth_dst = MAC1 eth_type = 0x800	output 1	?
3	eth_src = MAC6 eth_type = 0x800 ip_proto = 6	output 2	?

s2

Priority	Match	Action	pkt cnt
0	*	output controller	?
1	eth_dst = MAC1 eth_type = 0x800	output 1	?
2	eth_dst = MAC3 eth_type = 0x800 ip_dst = IP3	set eth_dst=MAC4 set ip_dst=IP4 ouput 4	?
2	eth_src = MAC4 eth_type = 0x800 ip_src = IP4	set eth_src=MAC3 set ip_src=IP3 ouput flood	?
3	eth_src = MAC6 eth_type = 0x800 ip_proto = 6	output 1	?

s3

Priority	Match	Action	pkt cnt
0	*	output controller	?
1	eth_dst = MAC1 eth_type = 0x800	output 1	?
3	eth_src = MAC6 eth_type = 0x800 ip_proto = 6	output 1	?

Esercizio 7

Si consideri il seguente set di regole, per il quale implementare un meccanismo di Bit Vector Linear Search:

Source Address	Destination Address	Source Port	Destination Port	Protocol
64.0.0.0/2	*	80	*	TCP
64.0.0.0/3	128.0.0.0/2	80	22	TCP
*	96.0.0.0/3	*	53	UDP
64.0.0.0/2	*	*	123	UDP
128.0.0.0/2	64.0.0.0/2	23	*	UDP
96.0.0.0/3	64.0.0.0/3	25	80	*

(a) Definire il bitvector set per ciascun campo.

Soluzione:

Source Address	
64.0.0.0/2	101100
64.0.0.0/3	111101
96.0.0.0/3	101101
128.0.0.0/2	001010
*	001000

Destination Address	
64.0.0.0/2	100110
64.0.0.0/3	100101
96.0.0.0/3	101010
128.0.0.0/2	110100
*	100100

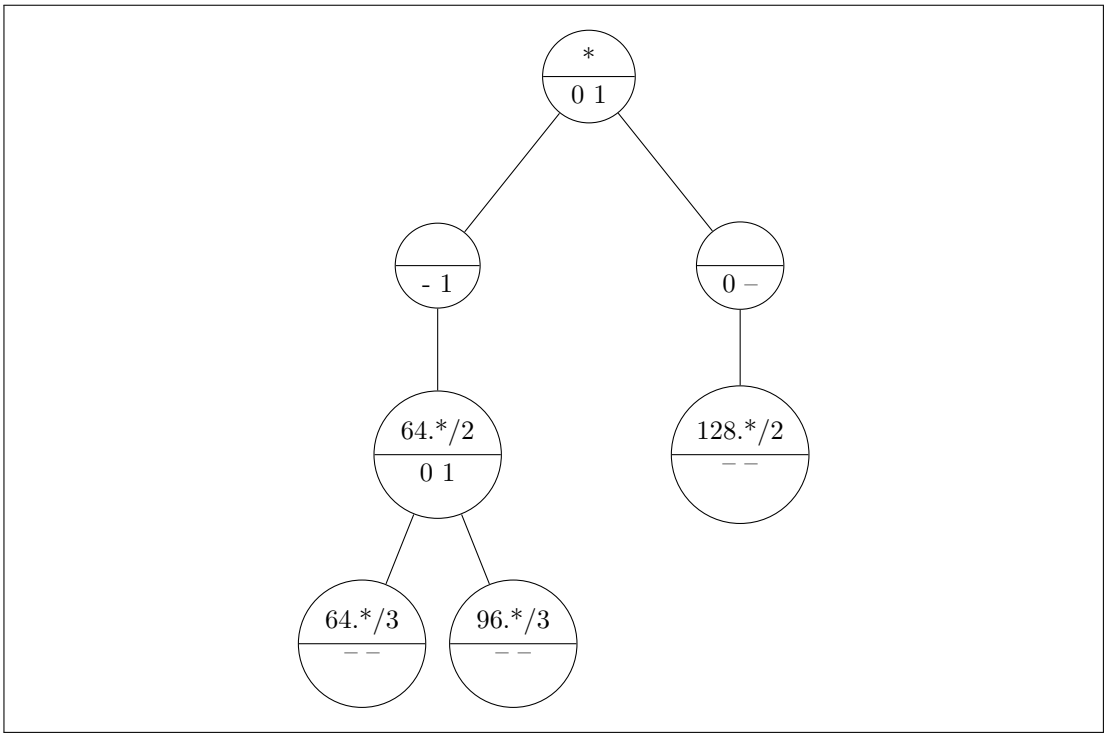
Source Port	
23	001110
25	001101
80	111100
*	001100

Dest. Port	
22	110010
53	101010
123	100110
80	100011
*	100010

protocol	
TCP	110001
UDP	001111
*	000001

(b) L'algoritmo di ricerca del prefisso IP sorgente (prima colonna) è implementato usando uno unibit trie. Disegnare il trie.

Soluzione:



(c) Si considerino i pacchetti (64.10.0.1, 96.0.0.2, 80, 53, UDP) e (97.50.0.1, 96.0.0.2, 80, 123, TCP). Applicare ad entrambi la procedura di BVLS per identificare gli eventuali match.

Soluzione: 1) 111101 AND 101010 AND 111100 AND 101010 AND 001111 = 001000
La terza regola.
2) 101101 AND 100101 AND 111100 AND 100110 AND 110001 = 100000
Il pacchetto soddisfa la regola 1.

(d) Si dica qual è la profondità massima dello unibit trie nei due casi IPv4 e IPv6.

Soluzione: Nel caso IPv4 32 bit. Nel caso IP64 128 bit.

Esercizio 8

Una matrice di commutazione crossbar di uno switch 4×4 usa un algoritmo per lo scheduling delle trasmissioni. Si assumano pacchetti tutti di lunghezza L,

Al tempo $t = 0$ sono presenti i seguenti pacchetti, in ordine dal più recente al più vecchio. Ogni pacchetto è etichettato con il numero dell'interfaccia di uscita cui è diretto.

Ingresso 1: 4, 2, 1

Ingresso 2: 2, 2, 3

Ingresso 3: –, 4, 3

Ingresso 4: –, 4, 1

(a) Svolgere l'algoritmo Take-a-Ticket (TaT) fino allo svuotamento di tutte le code, specificando ad ogni round quali pacchetti sono trasmessi e quali ticket sono assegnati. L'algoritmo di assegnazione dei ticket privilegia sempre l'interfaccia di ingresso con l'identificativo più piccolo.

Soluzione:

Round		Ingresso 1	2	3	4
1	Assegnazione	(1,1)	(3,1)	(3,2)	(1,2)
	Trasmissione	1	3	–	–
2	Assegnazione	(2,1)	(2,2)	–	–
	Trasmissione	2	–	3	1
3	Assegnazione	(4,1)	–	(4,2)	(4,3)
	Trasmissione	4	2	–	–
4	Assegnazione	–	(2,3)	–	–
	Trasmissione	–	2	4	–
5	Assegnazione	–	–	–	–
	Trasmissione	–	–	–	4

(b) Facendo riferimento all'algoritmo TaT con assegnazione casuale, si consideri uno scenario di traffico uniforme e code sempre sature. Qual è la probabilità che tutte le uscite siano usate?

Soluzione:

$$p = \frac{4!}{4^4} = \frac{3}{32} \simeq 9\%$$

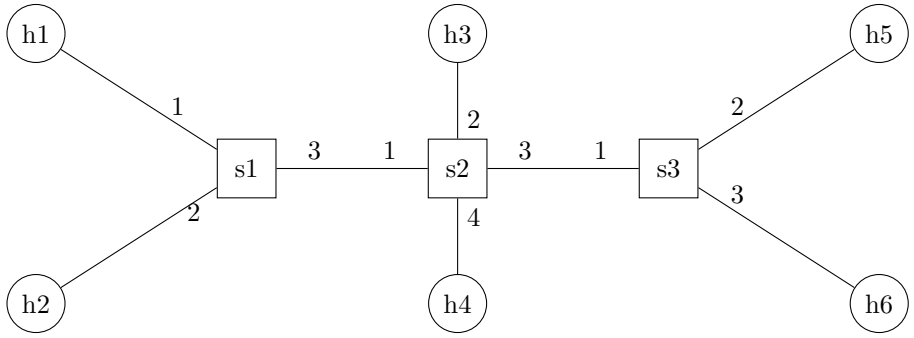
(c) Si consideri il caso in cui i pacchetti diretti alla coda 3 abbiano lunghezza $2L$. Ripetere la simulazione dell'algoritmo TaT.

Soluzione:

Round		Ingresso 1	2	3	4
1	Assegnazione	(1,1)	(3,1)	(3,2)	(1,2)
	Trasmissione	1	3	–	–
2	Assegnazione	(2,1)	–	–	–
	Trasmissione	2	(3)	–	1
3	Assegnazione	(4,1)	(2,2)	–	(4,2)
	Trasmissione	4	2	3	–
4	Assegnazione	–	(2,3)	–	–
	Trasmissione	–	2	(3)	4
5	Assegnazione	–	–	(4,3)	–
	Trasmissione	–	–	4	–

Esercizio 9

Si consideri la rete in figura. I nodi h1,..., hn contengono tabelle ARP prepopolate per tutta la rete. Gli indirizzi MAC e IP dei nodi h1,..., hn sono MAC1,..., MACn e IP1,..., IPn. Sui nodi h1,..., hn sono in esecuzione processi in ascolto su tutte le porte TCP e UDP di interesse. Il controllore Ryu è collegato ai nodi s1,..., sm ed è configurato come indicato nel Programma 1. Le interfacce tra gli switch e il controllore non sono indicate in figura. Se non altrimenti specificato, il TTL iniziale è 200.



(a) Scrivere il contenuto delle tabelle openflow dopo l'accensione di tutti i nodi.

Soluzione:

s1

Priority	Match	Action
0	*	output controller

s2

Priority	Match	Action
0	*	output controller
1	eth_dst = MAC3 eth_type = IPv4 ip_proto = TCP	output 2

s3

Priority	Match	Action
0	*	output controller

L'host h2, porta 7843 invia un datagramma UDP a h6, porta 543 con TTL iniziale 10.

(b) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s1, s2, 3.

Soluzione:

$s3 \rightarrow C$:	packet_in(MAC6 > MAC2 IPv4 IP6 > IP2 UDP 7843 > 543)
$s3 \rightarrow s2$:	MAC6 > MAC2 IPv4 IP6 > IP2 UDP 7843 > 544
$s2 \rightarrow C$:	packet_in(MAC6 > MAC2 IPv4 IP6 > IP2 UDP 7843 > 544)
$s2 \rightarrow s1$,$h3$,$h4$:	MAC6 > MAC2 IPv4 IP6 > IP2 UDP 7843 > 545
$s1 \rightarrow C$:	packet_in(MAC6 > MAC2 IPv4 IP6 > IP2 UDP 7843 > 545)
$s1 \rightarrow h1$,$h2$:	MAC6 > MAC2 IPv4 IP6 > IP2 UDP 7843 > 546

L'host h1, porta 3456 apre una connessione verso h5, porta 123.

(c) Scrivere tutti i pacchetti uscenti da tutte le interfacce di s1, s2, s3.

Soluzione:

$s1 \rightarrow C$:	packet_in(MAC1 > MAC3 IPv4 IP1 > IP3 TCP[SYN] 3456 > 123) nota: viene inserita una regola in s1
$s1 \rightarrow s2$:	MAC1 > MAC3 IPv4 IP1 > IP3 TCP[SYN] 3456 > 123
$s2 \rightarrow h3$:	MAC1 > MAC3 IPv4 IP1 > IP3 TCP[SYN] 3456 > 123

Programma 1 (1/2) – Switch Features

```
@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=0,
                            match=match, instructions=inst)
    datapath.send_msg(mod)

    if (datapath.id == 2):
        match = parser.OFPMatch(in_port=1, eth_type=MPLS)
        actions = [ parser.OFPActionOutput(3) ]
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=1,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

    if (datapath.id == 3):
        match = parser.OFPMatch(eth_type=MPLS)
        actions = [ parser.OFPActionPopMpls(),
                    parser.OFPActionOutput(ofproto.OFPP_FLOOD) ]
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=1,
                                match=match, instructions=inst)
        datapath.send_msg(mod)
```

Programma 1 (2/2) – 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
    in_port = msg.match['in_port']
    pkt = packet.Packet(msg.data)
    pkt_eth = pkt.get_protocol(eth.eth)
    pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
    pkt_udp = pkt.get_protocol(udp.udp)
    pkt_tcp = pkt.get_protocol(tcp.tcp)
    data = msg.data

    if (datapath.id == 1) and (pkt_tcp is not None) and (pkt_ipv4.dst==IP5):
        match = parser.OFPMatch(
            eth_dst=pkt_eth.src,eth_src=pkt_eth.dst,eth_type=IPv4,ip_proto=TCP)
        actions = [
            parser.OFPActionPushMpls(),
            parser.OFPActionSetField(mpls_label=1000),
            parser.OFPActionOutput(3)
        ]
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=2,
                                match=match, instructions=inst)
        datapath.send_msg(mod)
    elif (pkt_udp is not None):
        if ( pkt_ipv4.ttl <= 5):
            return
        pkt_ipv4.ttl = pkt_ipv4.ttl - 5
        pkt_udp.csum = 0 # azzera checksum esistente
        pkt.serialize()  # calcola nuovi checksum
        data = pkt.data

    if (datapath.id == 1) and (in_port < 3):
        outport = 3
    elif (datapath.id == 3) and (in_port > 1):
        outport = 1
    else:
        outport = ofproto.OFPP_FLOOD
    actions = [parser.OFPActionOutput(outport)]
    out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                              in_port=in_port, actions=actions, data=data)
    datapath.send_msg(out)
```

(d) L'host h3 risponde. Scrivere tutti i pacchetti uscenti da tutte le interfacce di s1, s2, s3.

Soluzione:

$s2 \rightarrow C$:	packet_in(MAC3 > MAC1 IPv4 IP3 > IP1 TCP[SYN,ACK] 123 > 3456)
$s2 \rightarrow s1, s2, h4$:	MAC3 > MAC1 IPv4 IP3 > IP1 TCP[SYN,ACK] 123 > 3456
$s1 \rightarrow h1$:	MAC3 > MAC1 IPv4 IP3 > IP1 TCP[SYN,ACK] 123 > 3456
$s3 \rightarrow C$:	packet_in(MAC3 > MAC1 IPv4 IP3 > IP1 TCP[SYN,ACK] 123 > 3456)
$s3 \rightarrow h5$,$h6$:	MAC3 > MAC1 IPv4 IP3 > IP1 TCP[SYN,ACK] 123 > 3456

(e) L'host h1 riscontra la risposta. Scrivere tutti i pacchetti uscenti da tutte le interfacce di s1, s2, s3.

Soluzione:

$s1 \rightarrow C$:	packet_in(MAC1 > MAC3 IPv4 IP1 > IP3 TCP[SYN] 3456 > 123) nota: viene inserita una regola in s1, che sovrascrive la regola esistente
$s1 \rightarrow s2$:	MAC1 > MAC3 IPv4 IP1 > IP3 TCP[SYN] 3456 > 123
$s2 \rightarrow h3$:	MAC1 > MAC3 IPv4 IP1 > IP3 TCP[SYN] 3456 > 123

(f) Scrivere il contenuto finale delle tabelle openflow.

Soluzione:

s1

Priority	Match	Action
0	*	output controller
3	eth_dst = MAC1 eth_src = MAC 3 eth_type = IPv4 ip_proto = TCP	output 1

s2

Priority	Match	Action
0	*	output controller
1	eth_dst = MAC3 eth_type = IPv4 ip_proto = TCP	output 2

s3

Priority	Match	Action
0	*	output controller

Esercizio 10

Si consideri un sistema di misura del traffico in un router. Ad ogni pacchetto entranti è associato un identificativo di flusso x ottenuto concatenando alcuni campi del pacchetto. La lunghezza massima dei pacchetti è 1024 byte.

Sull'identificativo x sono calcolati n hash usando la famiglia di funzioni indipendenti $h_i(x)$ con $1 \leq i \leq n$. Tali hash sono usati come indici in n array distinti $A_1, \ldots, A_n$ di m elementi ciascuno. Ad ogni pacchetto arrivato si incrementano di L gli elemento $A_i[h_i(x)]$ per ogni i, dove L è la lunghezza del pacchetto.

Se $A_i[h_i(x)] \geq S$ per ogni i, l'indentificativo x viene salvato su disco.

Tutti gli array sono azzerati ogni $T = 1\,\text{s}$.

(a) Considerando quattro link entranti con capacità $C = 1\,\text{Gbit/s}$ ciascuno, dire quanti bit occorrono per rappresentare gli n array.

Soluzione: Nel caso peggiore, in un secondo si accumulano 10^9 bit in un unico contatore. Contando in ottetti, servono

$$\lceil \log_2 10^9/8 \rceil mn = 27mn \text{ bit}$$

(b) Dire quali campi occorre scegliere per rappresentare l'identificativo del flusso per avere nello stesso flusso tutti i pacchetti della stessa connessione TCP in entrambe le direzioni.

Soluzione:

$$x = \text{IPs} \| \text{IPd} \| \text{proto} \| \text{ports} \| \text{portd}$$

(c) Dire come scegliere la soglia S in modo che si scriva su disco l'identificativo di tutti i flussi di almeno 50 Mbit/s. Ci sono falsi positivi? Ci sono falsi negativi?

Soluzione: La soglia va fissata a $S = 125\,000$ ottetti. Ci possono essere falsi positivi, non ci sono falsi negativi.

(d) Dall'inizia della misura si è osservato un solo flusso y da 75 Mbit/s. Considerando $m = 2^7$ e $n = 4$, dire qual è la probabilità che il primo pacchetto di un nuovo flusso sia copiato su disco.

Soluzione: Si ha una collisione se il secondo flusso x collide con y su tutti gli hash. Quiesto accade con probabilità:

$$\left(\frac{1}{m}\right)^n = \frac{1}{2^{24}} = 6 \cdot 10^{-8}$$

Esercizio 11

Si consideri il seguente set di regole, per il quale implementare un meccanismo di Bit Vector Linear Search:

Source Address	Destination Address	Source Port	Destination Port	Transport Protocol
112.0.0.0/6	*	*	8080	TCP
104.0.0.0/8	32.0.0.0/3	80	*	TCP
*	104.0.0.0/8	*	53	*
104.0.0.0/6	*	*	123	UDP
32.0.0.0/3	104.0.0.0/8	23	*	UDP
*	104.0.0.0/6	25	80	*

(a) Definire il bitvector set per ciascun campo.

Soluzione:

Source Address	
38.0.0.0/7	011001
38.0.0.0/8	011101
48.0.0.0/4	101001
51.0.0.0/8	101011
*	001001

Destination Address	
38.0.0.0/7	100101
38.0.0.0/8	100111
48.0.0.0/4	101100
51.0.0.0/8	111100
*	100100

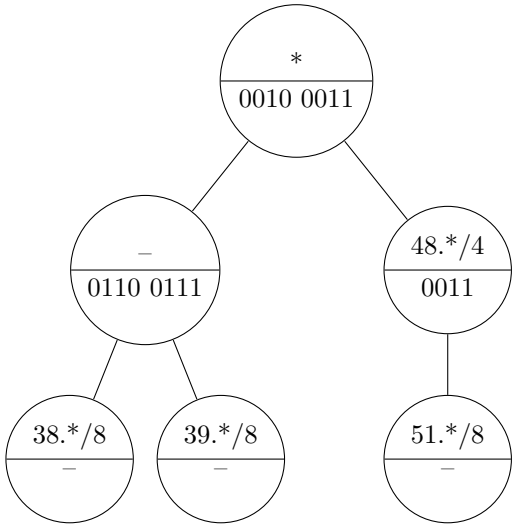
Source Port	
23	101110
25	101101
80	111100
*	101100

Dest. Port	
22	010010
53	001010
80	100011
123	000110
*	000010

protocol	
TCP	001111
UDP	111001
*	001001

(b) L'algoritmo di ricerca del prefisso IP sorgente (prima colonna) è implementato usando un multibit trie con passo 3. Disegnare il trie.

Soluzione:



(c) Si considerino il pacchetto (213.10.0.1, 81.10.0.2, 80, 123, UDP). Elencare le maschere identificate per ogni campo e identificare gli eventuali match.

Soluzione: Il pacchetto soddisfa la regola 6.

Esercizio 12

Una matrice di commutazione crossbar di uno switch 4×4 usa un algoritmo per lo scheduling delle trasmissioni. Si assumano pacchetti tutti di lunghezza L,

Al tempo $t = 0$ sono presenti i seguenti pacchetti, in ordine dal più recente al più vecchio. Ogni pacchetto è etichettato con il numero dell'interfaccia di uscita cui è diretto.

Ingresso 1: 4, 2, 1

Ingresso 2: 1, 3, 2

Ingresso 3: –, 4, 3

Ingresso 4: –, 4, 1

(a) Svolgere l'algoritmo Take-a-Ticket (TaT) fino allo svuotamento di tutte le code, specificando ad ogni round quali pacchetti sono trasmessi e quali ticket sono assegnati. L'algoritmo di assegnazione dei ticket privilegia sempre l'interfaccia di ingresso con l'identificativo più piccolo.

Soluzione:

Round		Ingresso 1	2	3	4
1	Assegnazione	(1,1)	(3,1)	(3,2)	(1,2)
	Trasmissione	1	3	–	–
2	Assegnazione	(2,1)	(2,2)	–	–
	Trasmissione	2	–	3	1
3	Assegnazione	(4,1)	–	(4,2)	(4,3)
	Trasmissione	4	2	–	–
4	Assegnazione	–	(2,3)	–	–
	Trasmissione	–	2	4	–
5	Assegnazione	–	–	–	–
	Trasmissione	–	–	–	4

(b) Si consideri il caso in cui i pacchetti diretti alla coda 2 abbiano lunghezza $2L$. Ripetere la simulazione dell'algoritmo TaT.

Soluzione:

Round		Ingresso 1	2	3	4
1	Assegnazione	(1,1)	(3,1)	(3,2)	(1,2)
	Trasmissione	1	3	–	–
2	Assegnazione	(2,1)	–	–	–
	Trasmissione	2	(3)	–	1
3	Assegnazione	(4,1)	(2,2)	–	(4,2)
	Trasmissione	4	2	3	–
4	Assegnazione	–	(2,3)	–	–
	Trasmissione	–	2	(3)	4
5	Assegnazione	–	–	(4,3)	–
	Trasmissione	–	–	4	–

(c) Facendo riferimento all'algoritmo TaT con assegnazione casuale, si consideri uno scenario di traffico uniforme e code sempre sature. Qual è la probabilità che tre ingressi su quattro siano bloccati?

Soluzione:

$$p = \frac{4!}{4^4} = \frac{3}{32} \simeq 9\%$$