1

Routers

1.1 IP Address Lookup

Esercizio 1.1 Un possibile approccio per ridurre gli accessi a memoria durante il lookup è memorizzare un insieme di prefissi nella cache (fast memory), mentre la lista completa dei prefissi è contenuta nella memoria (slow memory).

Al momento di eseguire un lookup, viene prima individuato il miglior matching tra i prefissi memorizzati nella cache. Se tale matching esiste, viene utilizzato il next hop associato al prefisso selezionato. Altrimenti, si cerca un matching all'interno dell'intero database e il prefisso individuato viene inserito nella cache.

Periodicamente, i prefissi non utilizzati contenuti nella cache vengono cancellati.

a) Dimostrare con un esempio che questo algoritmo non garantisce l'individuazione del longest prefix match.

b) Modificare l'algoritmo in modo da assicurare un matching corretto.

Soluzione (a) Se viene individuato un match tra i prefissi della cache, potrebbe comunque esserci un match con un prefisso più lungo all'interno della memoria.

Vediamo un esempio. Consideriamo la cache inizialmente vuota, due prefissi P1 = 00*, P2 = 000* all'interno del database e un pacchetto A1 il cui indirizzo di destinazione comincia con 001.

Poichè il longest match è con P1, P1 = 00* è memorizzato nella cache.

Si consideri un successivo pacchetto A2 il cui indirizzo di destinazione comincia con 000.

Effettuando il prefix matching all'interno della cache, viene identificato un matching con P1 e il pacchetto viene erroneamente instradato all'indirizzo associato a P1, mentre il matching corretto è P2.

(b) Due possibili soluzioni.

- quando si identifica un matching P all'interno della cache, si ripete il lookup considerando solo i prefissi del database che iniziano con P
- memorizzare il prefisso di un match all'interno della cache solo se il database non contiene un prefisso più specifico che lo estenda.

Esercizio 1.2 Per codificare prefissi di lunghezza variabile con stringhe di lunghezza fissa si può usare il seguente schema: alla fine del prefisso si aggiunge un bit pari a uno seguito da bit pari a zero fino a raggiungere una lunghezza totale di 33 bit.

a) Dire quanti sono i possibili prefissi di 32 bit.

b) Codificare i prefissi 128/2, 128/3, 128/4 e mostrare che le codifiche sono diverse tra loro.

Soluzione (a) Il numero totale di prefissi di lunghezza n è 2^n. Perciò il numero di prefissi di 32 bit è:

$$\sum_{n=0}^{32} 2^n = 1 + 2 + \ldots 2^{32} = 2^{33} - 1$$

(b) Il prefissi sono codificati come segue:

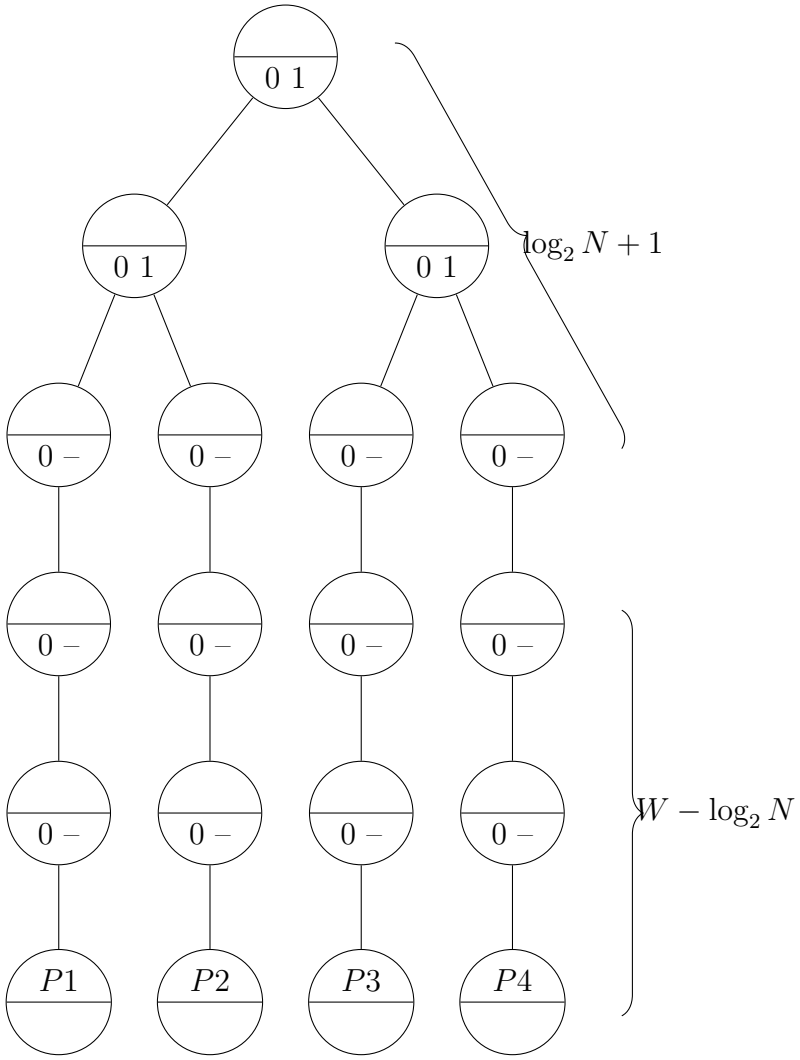
128/2	10100000000000000000000000000000
128/3	10010000000000000000000000000000
128/4	10001000000000000000000000000000

Esercizio 1.3 Per uno unibit trie in assenza di compressione dei rami one-way, mostrare che il massimo numero di nodi di trie è $O(NW)$, ove N è il numero di prefissi e W è la lunghezza massima del prefisso.

Suggerimento: considerare N una potenza di due e generare un trie completo con $\log_2 N + 1$ livelli per avere N nodi al livello inferiore, poi aggiungere una stringa di $W - \log_2 N$ bit da ciascuno degli N nodi.)

Soluzione Nell'esempio suggerito, il numero totale di nodi $N - 1 + N(W - \log_2 N)$, pertanto la complessità è $O(NW)$

Nel disegno abbiamo il caso $N = 4, W = 5$. L'albero ha $\log_2 4 + 1 = 3$ livelli e dal livello inferiore si estende una stringa di $5 - 2 = 3$ nodi.



La tabella di routing corrispondente contiene i prefissi:

P1	00000
P2	01000
P3	10000
P4	11000

Nota: gli ultimi tre bit di ogni prefisso possono differire.

Esercizio 1.4 Si consideri l'algoritmo di ricerca binaria su prefix lengths. Supponiamo di utilizzare soltanto markers e di non effettuare nessuna precomputazione del longest match associato al marker. Ciò abbatterebbe i tempi di inserimento.

a) Mostrare un esempio in cui il backtracking ha complessità lineare e non logaritmica.

Suggerimento: Considerare una lookup table con 7 possibili lunghezze di prefissi, e i prefissi $P1 = 00000004$ e $P2 = 001$.

Soluzione ..

1	2	3	4	5	6	7
–	$M = 00*$	$P1 = 001*$	$M = 0000*$	–	$M = 000000*$	$P2 = 0000000$

Per gestire correttamente $P1$ servono i marker 0000* alla lunghezza 4 e 000000* alla lunghezza 6. Inoltre per gestire $P2$ c'è un marker 00* alla lunghezza 2.

Consideriamo ora l'indirizzo di destinazione 0000001. La ricerca inizia alla lunghezza 4, individua il marker e passa alla lunghezza 6, ove il marker successivo sposta la ricerca alla lunghezza 7. Lì la ricerca fallisce e ritorna alla lunghezza 5, ove non trova alcun match. Allora la ricerca si sposta alla lunghezza 2, il marker la sposta alla lunghezza 3, ove il match con 001 dà esito negativo. Infine, la ricerca ritorna alla lunghezza 1 e fallisce. Con questo procedimento, sono state considerate tutte le possibili lunghezze tra 1 e 7 (dipendenza lineare).

1.2 Packet Classification

Esercizio 1.5 Si consideri il seguente ruleset in ordine di priorità:

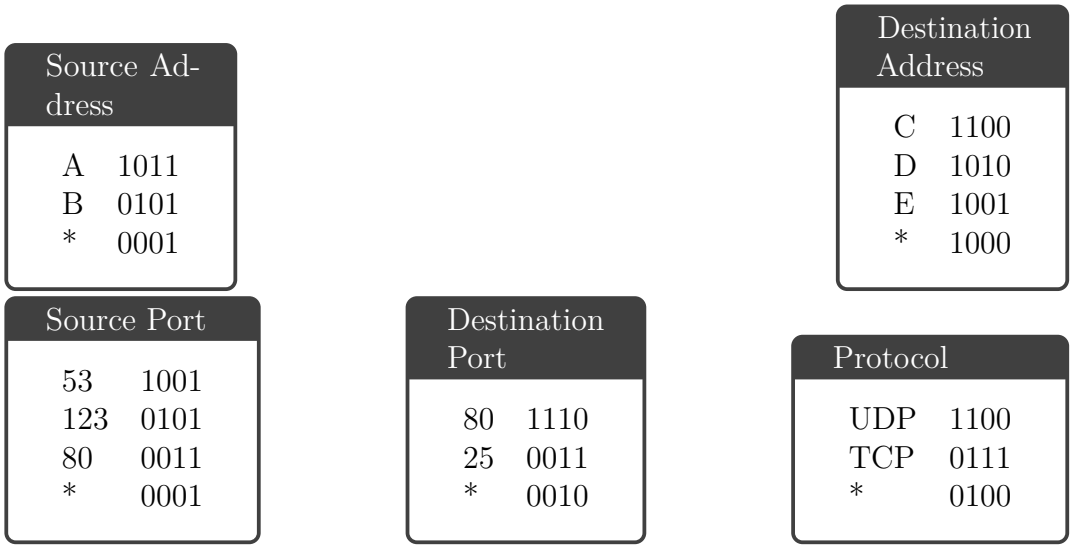
Source Address	Destination Address	Source Port	Destination Port	Protocol
A	*	53	80	UDP
B	C	123	80	*
A	D	80	*	TCP
*	E	*	25	TCP

La classificazione è effettuata con Bitvector Linear Search.

a) Definire il bitvector set per ciascun campo

b) Applicare l'algoritmo al pacchetto (A,D,80,23,TCP) per identificare la eventuale regola corrispondente.

Soluzione ..



Applicando le regole al pacchetto (A,D,80,23,TCP) si ottiene:

Campo	valore	stringa
Source Address	A	1011
Dest. Address	D	1010
Source Port	80	0011
Dest. Port	23	0010
Protocol	TCP	0111
AND		0010

Per cui la regola corrispondente è la terza.

Esercizio 1.6 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
*	E	*	80	TCP
A	F	25	*	UDP
B	*	21	22	TCP
C	G	80	*	*
*	*	53	80	UDP
D	E	*	123	*
B	H	80	*	UDP
C	*	21	*	TCP

a) Definire il bitvector set per ciascun campo.

Soluzione .

Source Address		Dest. Address		Source Port		Dest. Port		Protocol	
A	11001000	E	10101101	21	10100101	22	01110011	TCP	10110101
B	10101010	F	01101001	25	11000100	80	11011011	UDP	01011110
C	10011001	G	00111001	53	10001100	123	01010111	*	00010100
D	10001100	H	00101011	80	10010110	*	01010011		
*	10001000	*	00101001	*	10000100				

b) Assumendo che gli indirizzi sorgente/destinazione siano codificati su 32 bit, i numeri di porta sorgente/destinazione su 16 bit e il tipo di protocollo su 2 bit, calcolare la dimensione totale (in bit) del bitvector set.

Soluzione (32+8)*10+(16+8)*9+(2+8)*3= 646 bit

c) Si considerino i pacchetti (B, E, 23, 53, UDP) e (C, E, 21, 123, TCP). Applicare ad entrambi la procedura di BVLS per identificare gli eventuali matching.

Soluzione 1) 10101010 AND 10101101 AND 10000100 AND 01010011 AND 01011110 = 00000000 Nessuna regola soddisfatta.

2) 10011001 AND 10101101 AND 10100101 AND 01010111 AND 10110101 = 00000001 Il pacchetto soddisfa la regola 8.

d) Spiegare brevemente come può essere implementata la ricerca della parola binaria nei campi Source Address e Protocol.

1.3 Switching

Esercizio 1.7 Una matrice di commutazione crossbar di uno switch 3×3 usa un algoritmo per lo scheduling delle trasmissioni. Si assuma 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: 2, 3, 1

Ingresso 2: 1, 2, 3

Ingresso 3: 1, 3, 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
1	Assegnazione	(1,1)	(3,1)	(1,2)
	Trasmissione	1	3	–
2	Assegnazione	(3,2)	(2,1)	–
	Trasmissione	3	2	1
3	Assegnazione	(2,2)	(1,3)	(3,3)
	Trasmissione	2	1	3
4	Assegnazione	–	–	(1,4)
	Trasmissione	–	–	1

b) Svolgere l'algoritmo PIM fino allo svuotamento di tutte le code, specificando ad ogni round quali pacchetti sono trasmessi e quali grant sono richiesti e sono assegnati. L'algoritmo privilegia sempre l'interfaccia di ingresso o di uscita con l'identificativo più piccolo.

Soluzione

Round		Ingresso 1	2	3
1	Richiesta	{1,2,3}	{1,2,3}	{1,3}
	Assegnazione	{1,2,3}	–	–
	Trasmissione	1	–	–
2	Richiesta	{2,3}	{1,2,3}	{1,3}
	Assegnazione	{2,3}	{1}	–
	Trasmissione	2	1	–
3	Richiesta	{3}	{2,3}	{1,3}
	Assegnazione	{3}	{2}	{1}
	Trasmissione	3	2	1
4	Richiesta	–	{3}	{1,3}
	Assegnazione	–	{3}	{1}
	Trasmissione	–	3	1
5	Richiesta	–	–	{3}
	Assegnazione	–	–	{3}
	Trasmissione	–	–	–

c) Facendo riferimento all'algoritmo TaT con assegnazione casuale, si consideri uno scenario di traffico uniforme e code sempre sature. In quale caso la matrice lavora alla massima efficienza, ovvero tutte le uscite sono usate? Qual è la probabilità che ciò accada?

Soluzione Quando i tre ingressi hanno pacchetti per tre uscite distinte.

$$p = \frac{3!}{3^3} = \frac{6}{27} \simeq 22\%$$

d) Si consideri la variante dell'algoritmo TaT in cui i pacchetti che trovano l'interfaccia d'uscita occupata sono inviati alla prima interfaccia libera disponibile (*deflection routing*), anche se non corretta. Stimare la percentuale di pacchetti deflessi.

Soluzione

0 pacchetti deflessi tre pacchetti per tre uscite distinte

$$\Pr[0] = \frac{3!}{3^3}$$

2 pacchetti deflessi tre pacchetti per la stessa uscita

$$\Pr[2] = \frac{3}{3^3}$$

1 pacchetto deflesso due pacchetti per la stessa uscita e uno per un'uscita diversa

$$\Pr[1] = \frac{3^3 - 3! - 3}{3^3}$$

$$p = \frac{3^3 - 3! - 3}{3^3} + 2\frac{3}{3^3} = 1 - \frac{1}{9} = 0.89$$

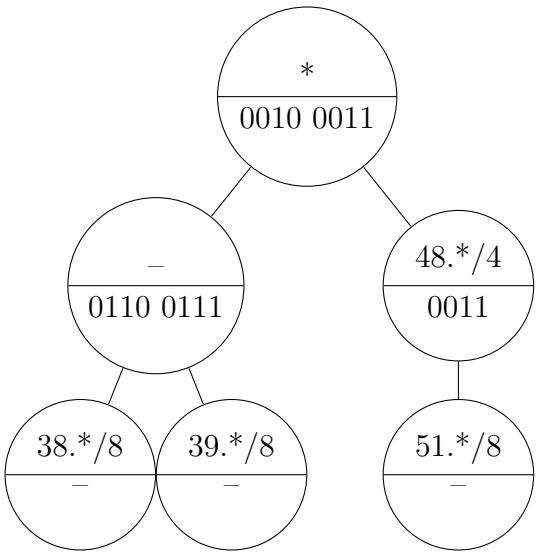
e) Facendo riferimento all'algoritmo PIM con scelta casuale, si consideri uno scenario di traffico uniforme e code infinite. In quale caso la matrice lavora alla massima efficienza, ovvero tutte le uscite sono usate? Qual è la probabilità che ciò accada?

Soluzione Quando le tre uscite assegnano i grant a tre ingressi distinti.

$$p = \frac{3!}{3^3} = \frac{6}{27} \simeq 22\%$$

Esercizio 1.8 Si consideri una tabella di routing contenente i seguenti prefissi: 38.0.0.0/7, 38.0.0.0/8, 48.0.0.0/4, 51.0.0.0/8, default. L'algoritmo di ricerca del prefisso è implementato usando un multibit trie con passo 4. Disegnare il trie.

Soluzione



1.4 Buffer Management

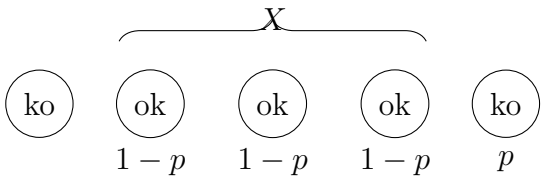
Esercizio 1.9 Si consideri un algoritmo di buffer management RED. Si assuma che in un certo periodo temporale la lunghezza media della coda sia costante e che ad essa corrisponda un parametro p. Calcolare la distribuzione di probabilità e valore atteso del numero di pacchetti che intercorrono tra due eventi successivi di scarto nei seguenti casi:

a) Il router scarta pacchetti con probabilità p.

b) Il router scarta pacchetti con probabilità $p_d = \frac{p}{1-cp}$, dove c è il numero di pacchetti non scartati dopo l'ultimo evento di scarto.

Soluzione

a) Sia X la variabile casuale che indica il numero di pacchetti tra due eventi di scarto.

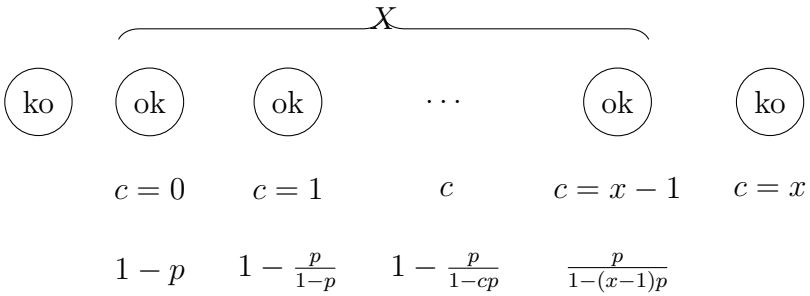


Si ha:

$$\Pr(X = x) = p(1-p)^x \quad x \geq 0$$

che è una variabile casuale con distribuzione geometrica e media $1/p$.

b) In questo caso si ha:



$$\Pr(X = x) = \frac{p}{1 - xp} \prod_{c=0}^{x-1} \left(1 - \frac{p}{1 - cp}\right) = \begin{cases} p & \text{se } 0 \leq x < 1/p \\ 0 & \text{se } x \geq 1/p \end{cases}$$

Pertanto X è una variabile casuale con distribuzione uniforme. Il valore atteso è:

$$\mathrm{E}(X) = \sum_{x=0}^{1/p} xp = p\frac{1}{2p}\left(\frac{1}{p} - 1\right) = \frac{1}{2p} - \frac{1}{2}$$

1.5 Scheduling

L'algoritmo DRR fornisce un servizio Round Robin fair anche in presenza di pacchetti di lunghezza variabile.

All'ingresso nel router i pacchetti sono divisi in flussi; i pacchetti di ogni flusso sono inseriti in una distinta coda FIFO.

Per ogni flusso i, il nodo mantiene una *quantum size* Q_i e un *deficit counter* D_i. Entrambe sono misurate in ottetti. La quantum size è un parametro di configurazione che serve per dare un peso ai diversi flussi. Il DRR darà al flusso $i$esimo una porzione di banda proporzionale a Q_i diviso la somma dei Q_j di tutti i flussi attivi. Il deficit counter è un contatore aggiornato dinamicamente durante l'esecuzione dell'algoritmo. È una misura di quanto servizio il flusso $i$esimo **non** ha ricevuto per via della lunghezza variabile dei pacchetti.

Ciascuna coda viene visitata ciclicamente. Per ogni flusso i, si aggiorna il deficit counter

$$D_i \leftarrow D_i + Q_i$$

Se D_i è maggiore o uguale alla lunghezza L_i del primo pacchetto nella coda i, si trasmette il pacchetto e si aggiorna

$$D_i \leftarrow D_i - L_i$$

Questa operazione si ripete finché non è più possibile trasmettere pacchetti da questa coda. Si passa quindi alla successiva.

Osservazioni:

- DRR è fair su lungo periodo
- La possibilità di inviare più di un pacchetto per round può portare a condizioni di non equità (e ritardi non predicibili) sul breve periodo
- Quando una coda è vuota il suo deficit è azzerato, altrimenti il deficit crescerebbe all'infinito quando il flusso non invia dati

- Se ci sono molte code, si perde molto tempo nel visitare code vuote

Se ci sono molte code, si perde molto tempo nel visitare code vuote. Per risolvere il problema si può mantenere una coda ausiliaria (chiamata `ActiveList`) di puntatori ai flussi aventi code non vuote. In questo modo si visitano solo le i flussi nella `ActiveList`. Alla fine della visita, se la coda è vuota, si rimuove dalla `ActiveList`, altrimenti si sposta in fondo.

Se una coda vuota diventa attiva perché arriva un nuovo pacchetto, la si colloca in fondo alla lista.

Iil costo per pacchetto dell'esecuzione dell'algoritmo DRR è costante se i Q_i sono maggiori o uguali della dimensione massima di un pacchetto. In questo modo viene inviato almeno un pacchetto ad ogni coda visitata.

Esercizio 1.10 Si consideri una interfaccia di uscita gestita da uno scheduler Deficit Round Robin. Al tempo $t = 0$ sono presenti i seguenti pacchetti con la seguenti lunghezze in byte, ordinati dal più recente al più vecchio:

Coda 1 20, 750, 200

Coda 2 500, 500

Coda 3 200, 600, 100

Coda 4 50, 700, 180

I quanti sono di 500 byte per tutte le code. La veliocità di trasmissione è di 1 byte per unità di tempo. Simulare l'algoritmo DRR, indicando ad ogni tempo quali pacchetti concludono la trasmissione.

Soluzione ..

Round	Coda	D_i iniziale	D_i finale	Lunghezza pacchetto Tx	Tempo fine Tx
1	1	0	300	200	200
1	2	0	0	500	700
1	3	0	400	100	800
1	4	0	320	180	980
2	1	300	50	750	1730
		50	0	20	1750
2	2	0	0	500	2250
2	3	400	300	600	2850
		300	0	200	3050
2	4	320	120	700	3750
		120	0	50	3800

Esercizio 1.11 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, 1, 1
- Ingresso 2: 3, 2, 1
- Ingresso 3: 2, 3, 1

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	2	3
1	Assegnazione ticket	(1,1)	(1,2)	(1,3)
	Trasmissione	invio	attesa	attesa
2	Assegnazione ticket	(1,4)	–	–
	Trasmissione	attesa	invio	attesa
3	Assegnazione ticket	–	(2,1)	–
	Trasmissione	attesa	invio	invio
4	Assegnazione ticket	–	(3,1)	(3,2)
	Trasmissione	invio	invio	attesa
5	Assegnazione ticket	(1,5)	–	–
	Trasmissione	invio	–	invio
6	Assegnazione ticket	–	–	(2,2)
	Trasmissione	–	–	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. Se tutte le interfacce sono bloccate tranne una, si scarta il pacchetto con ticket più alto.

Simulare l'algoritmo TaT modificato.

Soluzione ..

Round		Ingresso 1	2	3
1	Assegnazione ticket	(1,1)	(1,2)	(1,3)
	Trasmissione	invio	attesa	scarto
2	Assegnazione ticket	(1,4)	–	(3,1)
	Trasmissione	attesa	invio	invio
3	Assegnazione ticket	–	(2,1)	(2,2)
	Trasmissione	invio	invio	attesa
4	Assegnazione ticket	(1,5)	(3,2)	–
	Trasmissione	invio	invio	invio

E' stato scartato un pacchetto.

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

Soluzione Si scartano pacchetti se tutte le interfacce di uscita sono in coda per lo stesso ingresso. Questo accade con probabilità:

$$p = \frac{3}{3^3} = 0{,}11$$

d) Si calcoli il numero di pacchetti scartati dall'algoritmo precedente nel caso di switch 4×4.

Soluzione Si scartano pacchetti se tutte le interfacce di uscita sono in coda per lo stesso ingresso. Questo accade con probabilità:

$$p = \frac{4}{4^4} = 0{,}015$$

e) 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 1.12 Si consideri una interfaccia di uscita gestita da uno scheduler Deficit Round Robin. Al tempo $t = 0$ sono presenti i seguenti con la seguenti lunghezze in byte, ordinati dal più recente al più vecchio:

Coda 1 150,75,125

Coda 2 50,150,225

Coda 3 100,50,175

a) Assumendo quanti di grandezza 100 byte per tutte le code, simulare l'algoritmo DRR, indicando ad ogni tempo quali pacchetti concludono la trasmissione.

Soluzione .

Round		Coda 1	2	3
1	Quanti	100	100	100
	Trasmissioni	–	–	–
2	Quanti	0	200	25
	Trasmissioni	125 + 75 byte, T=200	–	175 byte, T=375
3	Quanti	100	75	75
	Trasmissioni	–	225 byte, T=600	50 byte, T=650
4	Quanti	0	25	0
	Trasmissioni	150 byte, T=800	150 byte, T=950	100 byte, T=1050
5	Quanti	0	0	0
	Trasmissioni	stop	50 byte, T=1100	stop

b) Assumendo quanti di grandezza 75,150 e 50 byte per le code 1,2 e 3 rispettivamente, e che al tempo T=400 arrivi un pacchetto da 200 byte alla coda 2 simulare l'algoritmo DRR, indicando ad ogni tempo quali pacchetti lasciano il sistema.

Soluzione .

Round		Coda 1	2	3
1	Quanti	75	150	50
	Trasmissioni	–	–	–
2	Quanti	25	75	100
	Trasmissioni	125 byte, T=125	225 byte, T=350	–
3	Quanti	25	25	150
	Trasmissioni	75 byte, T=425	150+50 byte, T=625	–
4	Quanti	100	175	25
	Trasmissioni	–	–	175 byte, T=800
5	Quanti	0	0	25
	Trasmissioni	150 byte, T=950	200 byte, T=1150	50 byte, T=1200
6	Quanti	0	0	75
	Trasmissioni	stop	stop	–
6	Quanti	0	0	0
	Trasmissioni	stop	stop	100 byte, T=1300

c) Calcolare i quanti da attribuire ad ogni coda in maniera che ad ogni round possa sicuramente essere trasmesso un pacchetto di 50 byte e che la coda 2 abbia banda doppia rispetto alle code 1 e 3.

Soluzione Una possibile scelta dei quanti è 50, 100, 50