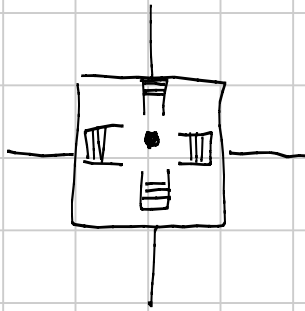


# Soluzione Esame 5 Luglio 2007

Note Title

15/07/2007

Il modello da sviluppare nell'esame del 5 luglio prevedeva una rete di nodi disposti in una topologia toroidale. Di fatto ogni nodo aveva 4 ingressi / uscite esplicite (e una implicita per l'ingresso / uscita dei pacchetti).



Lo stato del sistema era costituito, come in molti altri casi visti, dal numero di pacchetti nei buffer, e gli eventi sono tipicamente associati con l'ingresso e l'uscita di pacchetti dai buffer.

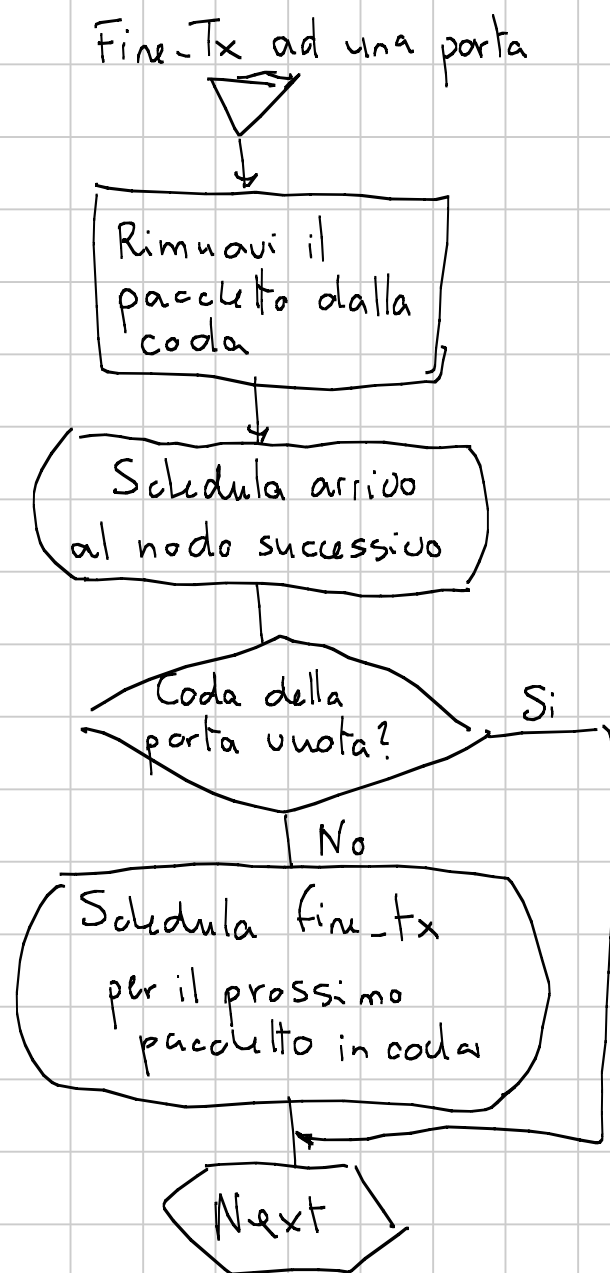
Il sistema era anche caratterizzato dall'aver canali di lunghezza variabile tra i nodi.

Per quel che riguarda il traffico, i punti chiave erano due: il traffico  $\mu$  è complessivo al sistema e veniva diviso tra i nodi secondo una matrice di traffico.

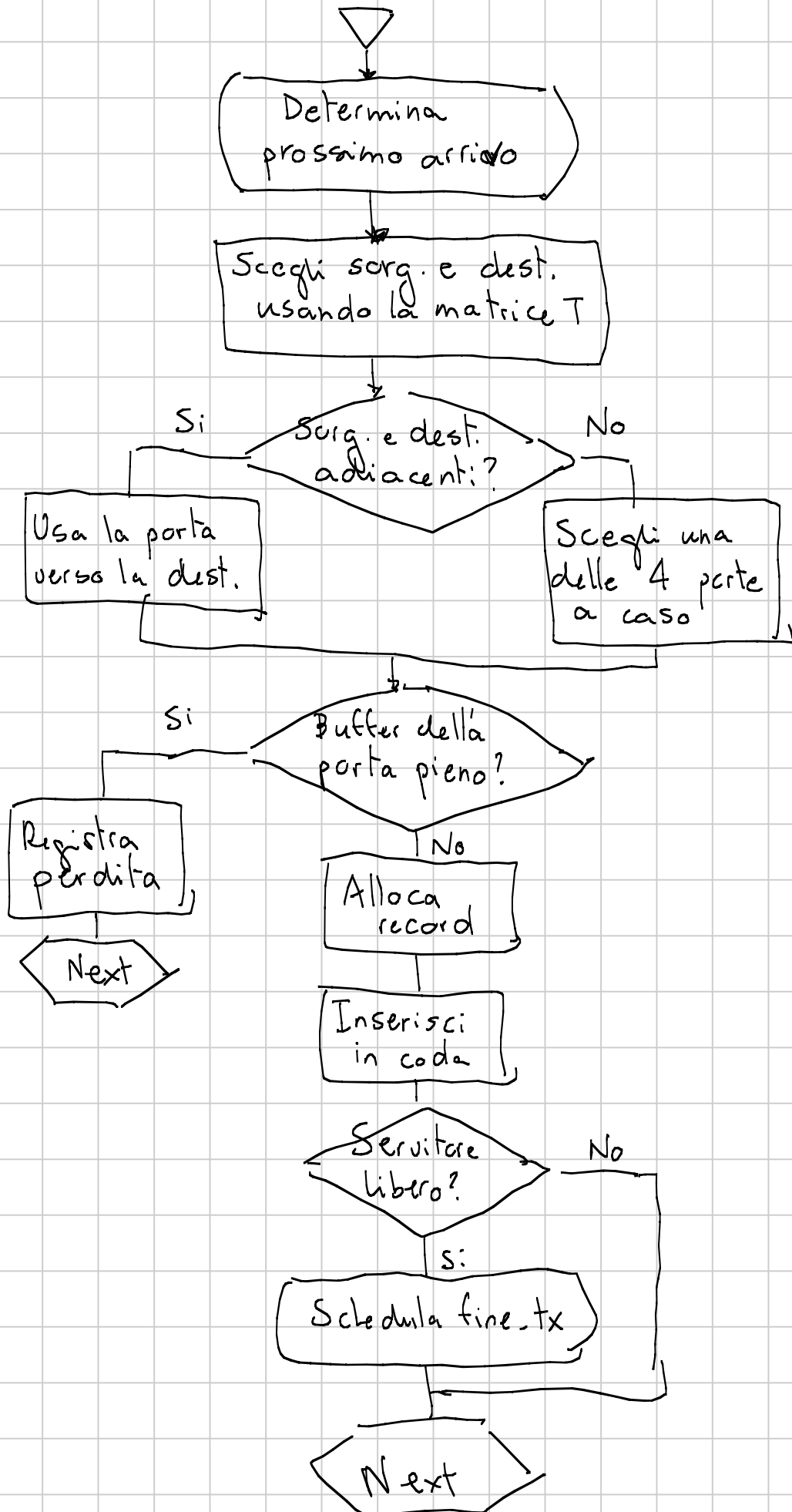
Dal punto di vista dell'implementazione era molto importante scegliere le strutture dati in maniera da semplificare le operazioni (anche eventualmente a costo di una inizializzazione complicata)

Cominciamo però con lo schema a blocchi del sistema, che è indipendente dal tipo di struttura scelta.  
Una possibile soluzione è quella che prevede 3 eventi:

- Arrivo di un nuovo pacchetto al sistema
- Fine Tx di un pacchetto da parte dell'interfaccia di un nodo
- Arrivo di un pacchetto sull'interfaccia di un nodo da un nodo adiacente



Arrivo Pacchetto al sistema



# Arrivo pacchetto ad un nodo



La scelta delle strutture dati è funzionale alle scelte implementative:

- struttura per ogni nodo con campi

int id-adiacenti [4]

Nodi adiacenti:

int id-porta remota [4]

id porta al nodo adiacente

Record \* queue [4]

Lista per la coda

int queue size [4]

Lunghezza della coda

double capacity [4]

} dati dell'interfaccia

double length [4]

per la trasmissione e

la propagazione  
presti da  $C_{ij}$  e  $L_{ij}$

Servono poi i vettori / matrici per generare

il traffico secondo la matrice

double cumul\_sorg [MAX\_NODE]

MAX\_NODE = M \* N

double cumul\_dest [MAX\_NODE][MAX\_NODE]

Come nella procedura vista negli esercizi in aula, all'inizializzazione si riempiono le due strutture

```
for (i = 0; i < MAX_NODE; i++)
```

```
{ prob_sorg[i] = 0;
```

```
for (j = 0; j < MAX_NODE; j++)
```

```
prob_sorg[i] += T[i][j];
```

/x prob. di  
generazione  
sorgenti

```
for (j = 0; j < MAX_NODE; j++)
```

```
cond_dest[i][j] = T[i][j] / prob_sorg[i];
```

```
} cumul_sorg[0] = prob_sorg[0]
```

```
for (i = 1; i < MAX_NODE; i++)
```

```
{ cumul_sorg[i] = cumul_sorg[i-1] + prob_sorg[i];
```

```
}
```

```
for (i = 0; i < MAX_NODE; i++)
```

```
{ cumul_dest[i][0] = cond_dest[i][0];
```

```
for (j = 1; j < MAX_NODE; j++)
```

```
cumul_dest[i][j] = cumul_dest[i][j-1] +
```

```
+ cond_dest[i][j];
```

```
}
```

Ci sono anche altri metodi per generare la coppia sorgente destinazione, l'importante è eseguirli correttamente.

## Inizializzazione

Azzerare i vari vettori/matrici per le misure

Azzerare le strutture per i nodi, che sono in un vettore nodi [MAX\_NODI]

Scheduliamo il primo arrivo

```
schedule (ARRIVO_PACCHETTO, current_time +  
negexp(1/μ, &seme), ∅, ∅, NULL);
```

La schedule usata prende 5 parametri

- Evento
- Tempo
- ID\_Nodo param 1
- ID\_Porta param 2
- Record pointer

Definiamo i 3 eventi: (x l'event-loop)

```
ARRIVO_PACCHETTO → arrivo ()  
FINE_TX → fine_tx (ev → param 1, ev → param 2);  
RICEZIONE → ricezione (ev → param 1,  
ev → param 2,  
(Record *) ev → pointer)
```

Le strutture dati per le misure le vedremo man mano  
che servono

```
void arrivo ()
```

```
{ Time delta;
```

```
  int source, dest;
```

```
  double u;
```

```
  int adjacent, port_out;
```

```
  Record * rec;
```

```
/* Prossimo arrivo */
```

```
delta = negexp (1.0/mu, & seme);
```

```
schedule (ARRIVO_PACCHETTO, current_time + delta,  
          0, 0, NULL);
```

```
/* scelta sorg / dest */
```

```
u = uniform 01 (& seme)
```

```
for (source = 0; source < MAX_NODE; source++)
```

```
{ if (u < cumul_sorg[source])
```

```
  break;
```

```
}
```

```
u = uniform 01 (& seme)
```

```
for (dest = 0; dest < MAX_NODE; dest++)
```

```
{ if (u < cumul_dest[source][dest])
```

```
  break;
```

```
}
```

```
total_arrivi[source][dest]++;
```

```
tot_arrivi++;
```

*/\* controllo adiacenza \*/*

adjacent = 0

for (i = 0; i < 4; i++)

{ if (nodi[source].id\_adiacenti[i] == dest)

{ adjacent = 1;

port\_out = i;

break;

}

}

if (!adjacent)

{ port\_out = (int) floor(uniform(0.0, 4.0, &seme));  
}

*/\* Ora posso inserire nel buffer opportuno \*/*

if (nodi[source].queue\_size[port\_out] < BSize)

{ rec = new\_record();

rec->source = source;

rec->destination = dest;

rec->arrival = current\_time;

rec->count = 0

in\_list(&nodi[source].queue[port\_out], rec);

nodi[source].queue\_size[port\_out]++;

if (nodi[source].queue\_size[port\_out] == 1)

{ delta = PktSize \* 8.0 / (nodi[source].capacity[port\_out] \* 1000);

schedule(FINE\_TX, current\_time + delta,

source, port\_out, NULL);

}

else

{ lost\_packets[source][port\_out]++;

}

return;

}

*/\* Nella struttura record avremo aggiunto i campi opportuni \*/*

La fine trasmissione elimina i pacchetti: dalla coda in cui si trovano e ne schedula l'arrivo all'opportuna interfaccia del nodo all'altro capo del canale

```
void fine_tx(int source, int port_out)
```

```
{ Time delta;  
  Record *rec;  
  int i;  
  int dest, dest_port;
```

```
/* toghiamo il pacchetto dalla coda del vecchio nodo */  
rec = ant_list(&(nodi[source].queue[port_out]));  
nodi[source].queue_size[port_out]--;
```

```
/* Determiniamo a che porta e che nodo va consegnato */  
dest = nodi[source].idadiacenti[port_out];  
for (dest_port = 0; dest_port < 4; dest_port++)  
  { if (nodi[dest].idadiscenti[dest_port] == source)  
    break;  
  }
```

```
schedule(RICEZIONE, current_time +  
         nodi[source].lunghezza[port_out] / 2e5,  
         dest, dest_port, rec);
```

```
/* Trasmettiamo il prossimo in coda */  
if (nodi[source].queue_size[port_out] != 0)  
  { delta = PckSize * 8.0 / (1000.0 * nodi[source].  
                             capacity[port_out]);  
    schedule(FINE_TX, current_time + delta, source,  
            port_out, NULL);
```

```
  }  
} return;
```

Infine abbiamo l'arrivo di un pacchetto all'interfaccia di un nodo

```
void ricezione(int node, int port-in, Record * rec)
```

```
{ Time delta;
```

```
  int i;
```

```
  int port-out, dest, dest-port;
```

```
  int adjacent;
```

```
  rec->counter++; /* Conto i nodi attraversati */
```

```
  /* Sono arrivato a destinazione? */
```

```
  dest = rec->destination;
```

```
  if (node == dest)
```

```
  { total-delivered[rec->source][node]++;
```

```
    tot-delivered++;
```

```
    total-delay += current-time - rec->arrival
```

```
    histo[rec->counter < MAX-HISTO ?
```

```
          rec->counter : MAX-HISTO]++;
```

```
    release_record(rec);
```

```
  }
```

```
  else
```

```
  { /* Non sono ancora arrivato: ripeti le  
    operazioni di relay */
```

```
    total-relay[node]++;
```

```
    adjacent = 0;
```

```
    /* controlla se destinazione adiacente */
```

```
    for (i=0; i<4; i++)
```

```
    { if (nodi[node].idadiacenti[i] == dest)
```

```
      { adjacent = 1;
```

```
        port-out = i; break;
```

```
      } }
```

```

if (! adjacent)
{
do
{
port_out = (int) floor (uniform (0.0, 4.0, &some));
}
while (port_out == port_in);
}
/* Escludo la porta di ingresso dalla
scelta casuale */

```

```

/* Ora che ho scelto una porta, accedo */
if (nodi [node]. queue_size [port_out] < Bsize)
{
in_lst (&(nodi [node]. queue [port_out]), rec);
nodi [node]. queue_size [port_out] ++;
if (nodi [node]. queue_size [port_out] == 1)
{
delta = PckSize * 8.0 / (1000 *
nodi [node]. capacity [port_out]);
schedule (FINI_TX, current_time + delta,
node, port_out, NULL);
}
}
else
{
release_record (rec);
lost_packets [node] [port_out] ++;
}
}
return;
}

```

Infine rimangono da stampare le misure, aggregando i dati che sono stati raccolti per nodo e per porta

$$\text{tot\_lost} = \sum_{i=0}^{\text{MAX\_NODE}-1} \sum_{j=0}^3 \text{lost\_packets}[i][j]$$

Prob. perdita  $\frac{\text{tot\_lost}}{\text{tot\_arrivi}}$

Distribuzione  $\frac{\text{histo}[i]}{\text{tot\_delivered}}$   $i = 0 \dots \text{MAX\_HISTO}$

Ritardo  $\frac{\text{total\_delay}}{\text{tot\_delivered}}$

Traffico di relay  $\frac{\text{total\_relay}[i]}{\text{current\_time}}$   $i = 0 \dots \text{MAX\_NODE}$

