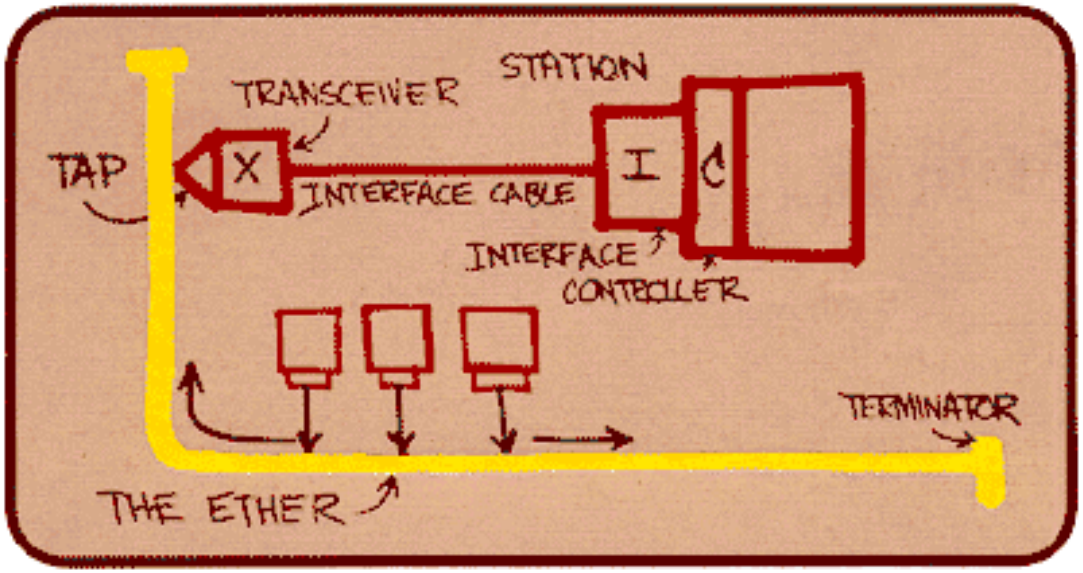
Uso della randomizzazione nelle reti

Randomizzazione

- la randomizzazione è usata in molti protocolli
- studieremo questi esempi:
  - protocollo di accesso multiplo Ethernet
  - rimozione di sincronizzazioni indesiderate
  - gestione attiva della coda
  - bilanciamento del carico in BitTorrent
  - la potenza delle due scelte (causali): bilanciamento del carico
  - instradamento randomizzato a due hop

Ethernet

- singolo canale broadcast condiviso
- 2+ trasmissioni simultanee dei nodi: interferenza
  - solo un nodo può trasmettere con successo in un dato istante
- *protocollo di accesso multiplo:* algoritmo distribuito che determina quali nodi accedono al canale, ovvero, quando un nodo può trasmettere



schizzo di Ethernet di Metcalfe

Ethernet: usa CSMA/CD

A: senti il canale, **se** libero

allora {

trasmetti, monitora il canale;

Se riveli un'altra trasmissione

allora {

abortisci e manda segnale di jamming;

aggiorna contatore di collisioni;

aspetta in base ad algoritmo di backoff esponenziale;

vai ad *A*

}

altrimenti {frame concluso; setta collisioni a zero}

}

altrimenti {aspetta finchè tx in corso finisce, vai ad *A*}

Ethernet CSMA/CD (cont.)

Backoff esponenziale:

- prima collisione per un pacchetto: scegli k a caso in {0,1}; attesa è k x 512 tempo di trasmissione di un bit. (K=2)
- dopo seconda collisione: scegli k a caso in {0,1,2,3}…(K=4)
- dopo ogni ulteriore collisione raddoppia K (e continua a raddoppiare K finchè…..)
- dopo dieci o più collisioni, scegli k a caso in {0,1,2,3,4,…,1023}

Uso della randomizzazione in Ethernet

- *comportamento risultante:* la probabilità di tentare una ritrasmissione (legata all'intervallo di randomizzazione) si adatta al carico corrente
  - semplice accesso multiplo adattativo

più collisioni → più spesso i nodi tentano di trasmettere → si randomizzano le ritrasmissioni su un intervallo di tempo più lungo, per ridurre la probabilità di collisione

Commenti su Ethernet

- Tetto massimo a K = 1023 limita intervallo
- si potrebbe tenere traccia dell'ultimo valore di K associato a trasmissione con successo (analogia con TCP: si tiene memoria del valore di congestion window precedente)
- D: perchè un backoff binario piuttosto di qualcosa di più sofisticato, come l'AIMD di TCP?
  - nota: Ethernet fa multiplicative-increase-complete-decrease

Analisi del protocollo CSMA/CD

Obiettivo: comprendere quantitativamente le prestazioni del protocollo CSMA

- lunghezza dei pacchetti fissa
- tempo di trasmissione di un pacchetto pari ad 1 unità di tempo
- ***throughput*** S – numero di pacchetti trasmessi con successo (senza collisione) nell'unità di tempo
- α – tempo di propagazione end-to-end
  - intervallo durante il quale possono avvenire collisioni

Modello di traffico per l'analisi di Ethernet

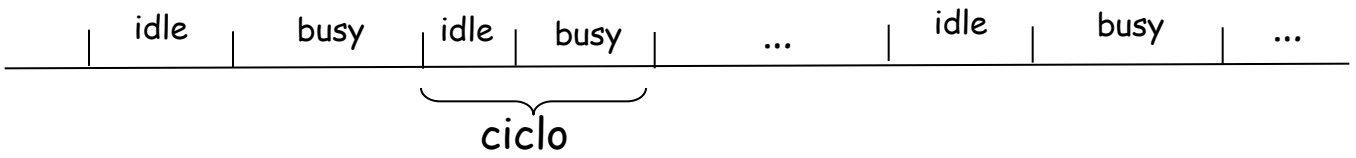
- *carico offerto* G – numero di trasmissioni tentate per unità di tempo
  - ***nota:*** $S < G$, S dipende da G
  - *modello Poisson:* probabilità di avere k tentativi di trasmissione in t unità di tempo

$$P[k\ trasmissioni\ in\ [0,t]] = (Gt)^k e^{-Gt}/k!$$

  - modelllo a popolazione infinita
- *capacità del protocollo di accesso multiplo:* massimo valore di S tra tutti i valori possibili di G

Analisi di CSMA/CD

Attività sul canale: cicli libero,occupato (idle,busy):



I – durata di periodo idle, B – durata di periodo occupato;

C = I + B: durata di un ciclo

Durate di ciclo successive sono indipendenti, per via dell'ipotesi Poisson

$$S = p/E[C] = p/(E[I]+E[B])$$

p - prob di trasmissione con successo durante un periodo busy

$$p = p(0 \text{ transmissions in } \alpha) = e^{-\alpha G}$$

- ogni periodo busy comincia con una trasmissione
- una trasmissione che comincia un periodo busy ha successo se non avvengono altre trasmissioni durante intervallo α

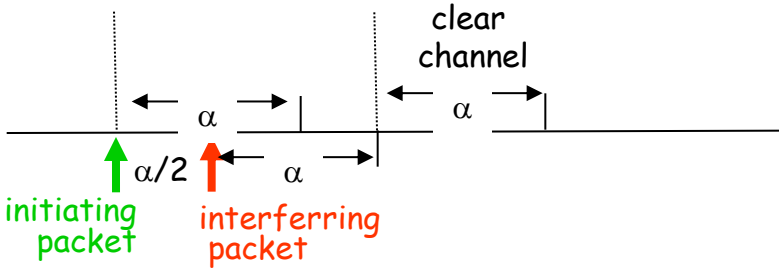
Analisi di CSMA/CD (cont.)

I distribuito esponenzialmente, a media 1/G (E[I] = 1/G)

Calcolo di E[B]:

Caso 1: no collisioni (NC): $E[B|NC] = 1 + \alpha$ con $P(NC) = e^{-\alpha G}$

Case 2: collisione (C): $P(C) = 1 - e^{-\alpha G}$



Calcolo approssimato di E[B]:

considero caso più probabile: una sola altra trasmissione interferisce col pacchetto che comincia il periodo busy

Il pacchetto interferente arriva in media $\alpha/2$ dopo l'inizio del periodo busy

Il pacchetto interferente trasmette per α unità di tempo prima che venga rivelata la collissione (caso peggiore)

Occorre tempo addizionale α prima che canale diventi idle

Perciò $E[B|C] = 5\alpha/2$

Analisi di CSMA/CD (cont.)

$$E[B] = P(NC)\, E[B|NC] + P(C)\, E[B|C]$$

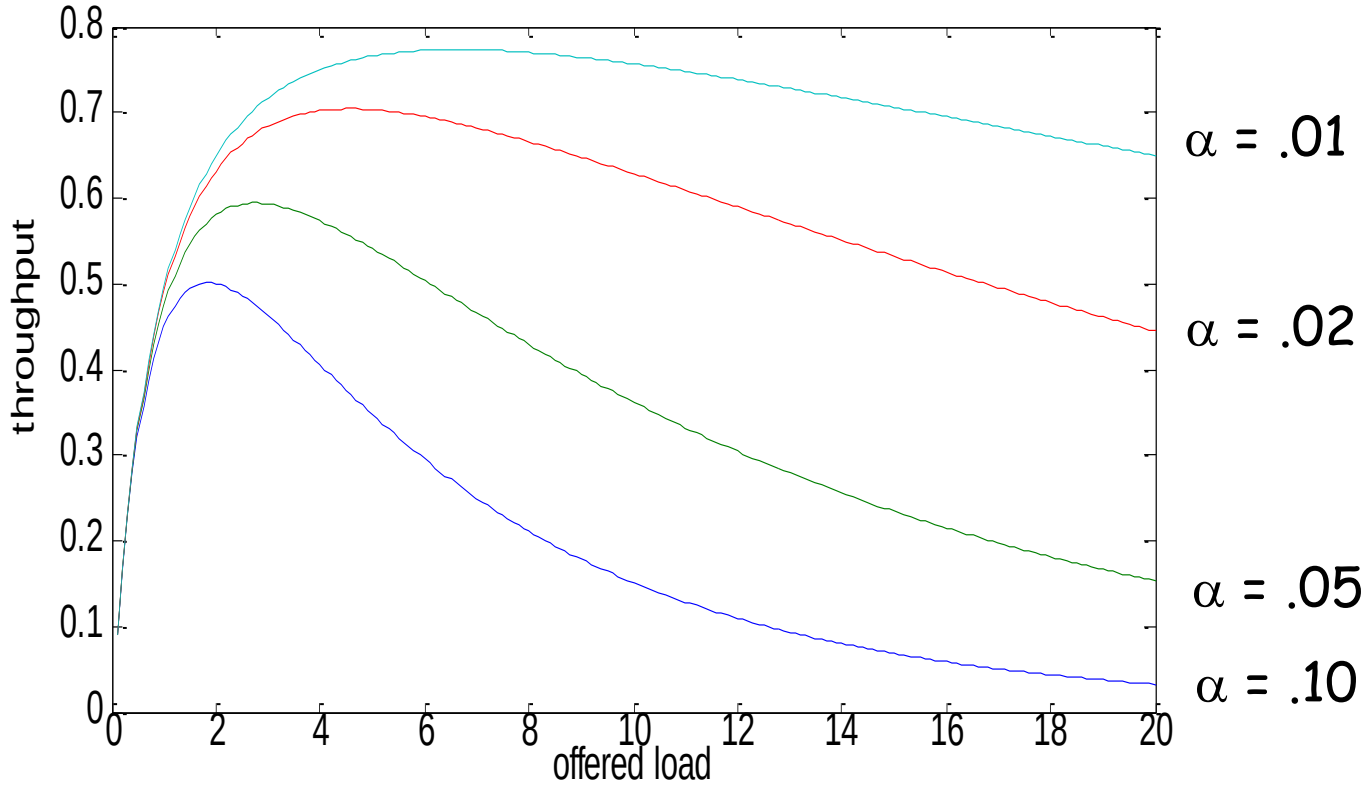
$$= e^{-\alpha G}(1 + \alpha) + (1 - e^{-\alpha G})5\alpha/2$$

dunque:

$$S = p/(E[I]+E[B])$$

$$= e^{-\alpha G}/(1/G + e^{-\alpha G}(1 - 3\alpha/2) + 5\alpha/2)$$

Ethernet: impatto del carico, di α sulle prestazioni

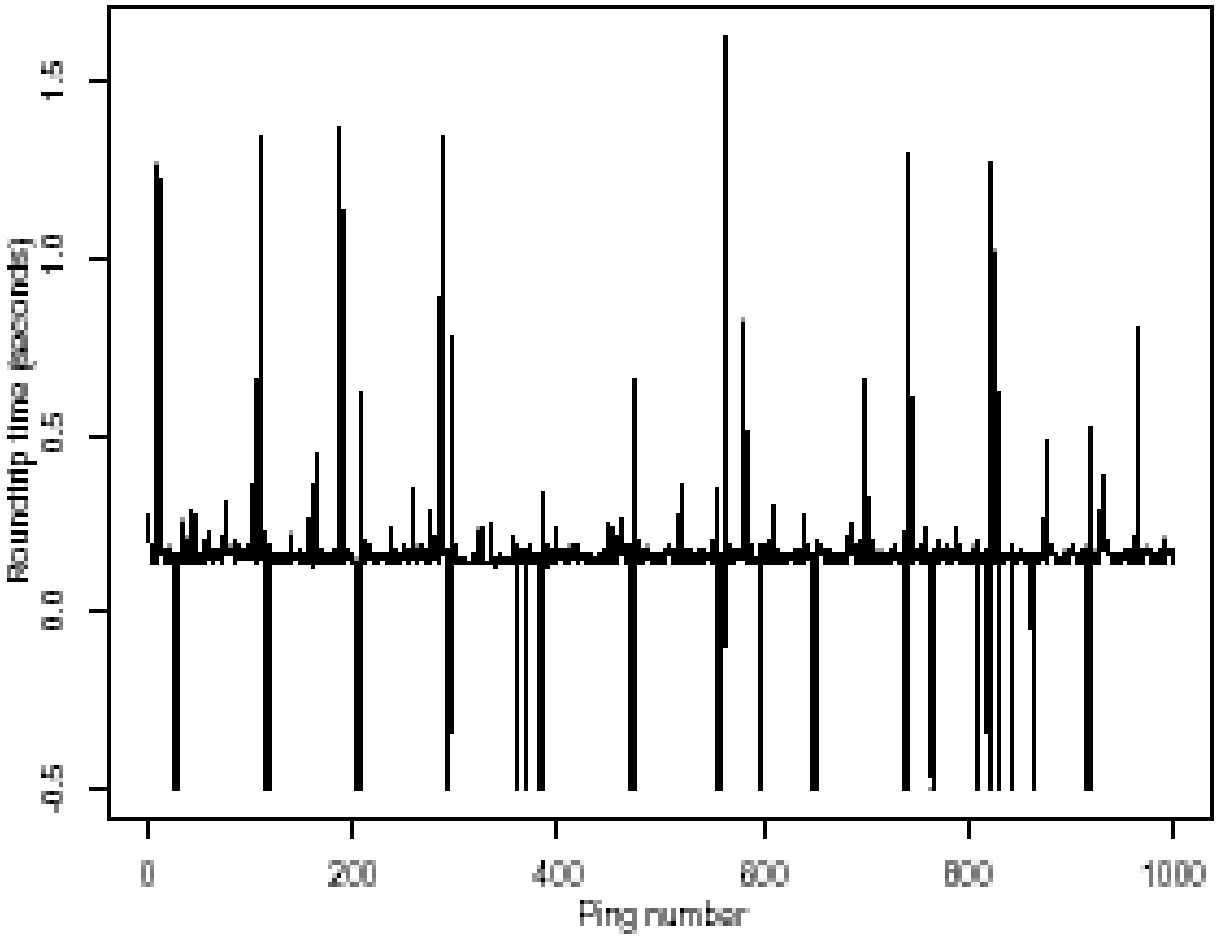


Morale

Perchè Ethernet usa la randomizzazione: per desincronizzare le trasmissioni – un algoritmo distribuito adattativo per spargere il carico nel tempo in caso di contesa per l'accesso multiplo al canale

(De)Sincronizzazione di aggiornamenti di routing periodici

- perdite periodiche osservate in traffico Internet end-to-end

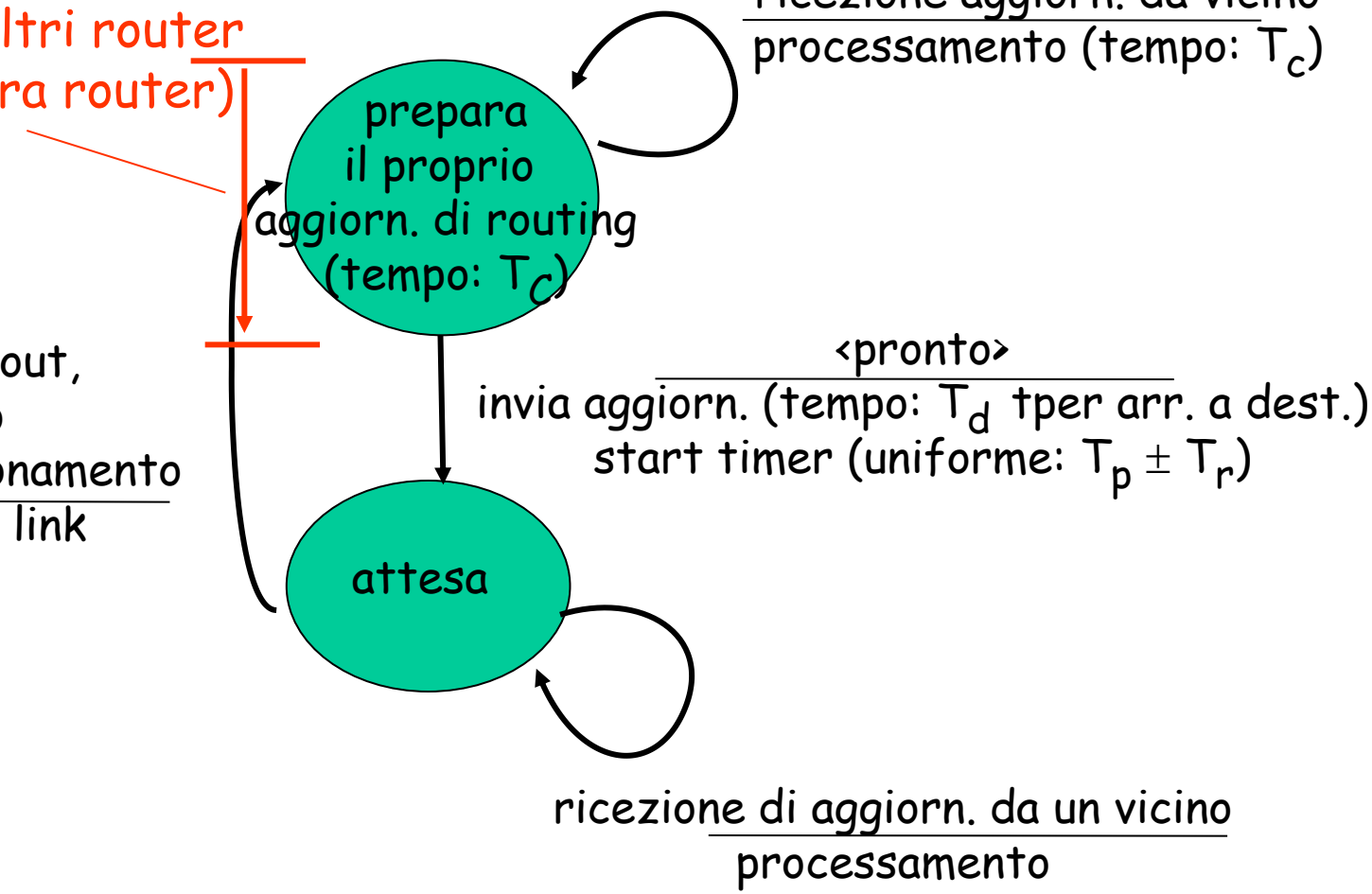


source: Floyd, Jacobson 1994

Perchè?

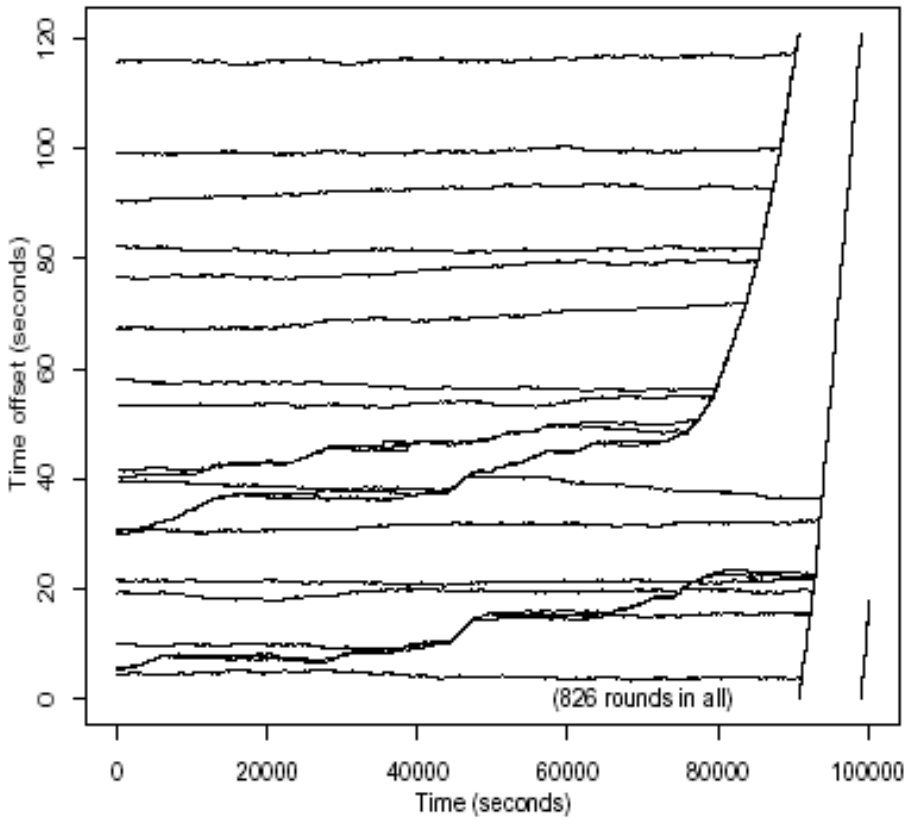
Dinamiche di aggiornamento in un router

il tempo speso nello stato dipende dai messaggi ricevuti dagli altri router (correlazione tra router)



Sincronizzazione dei router

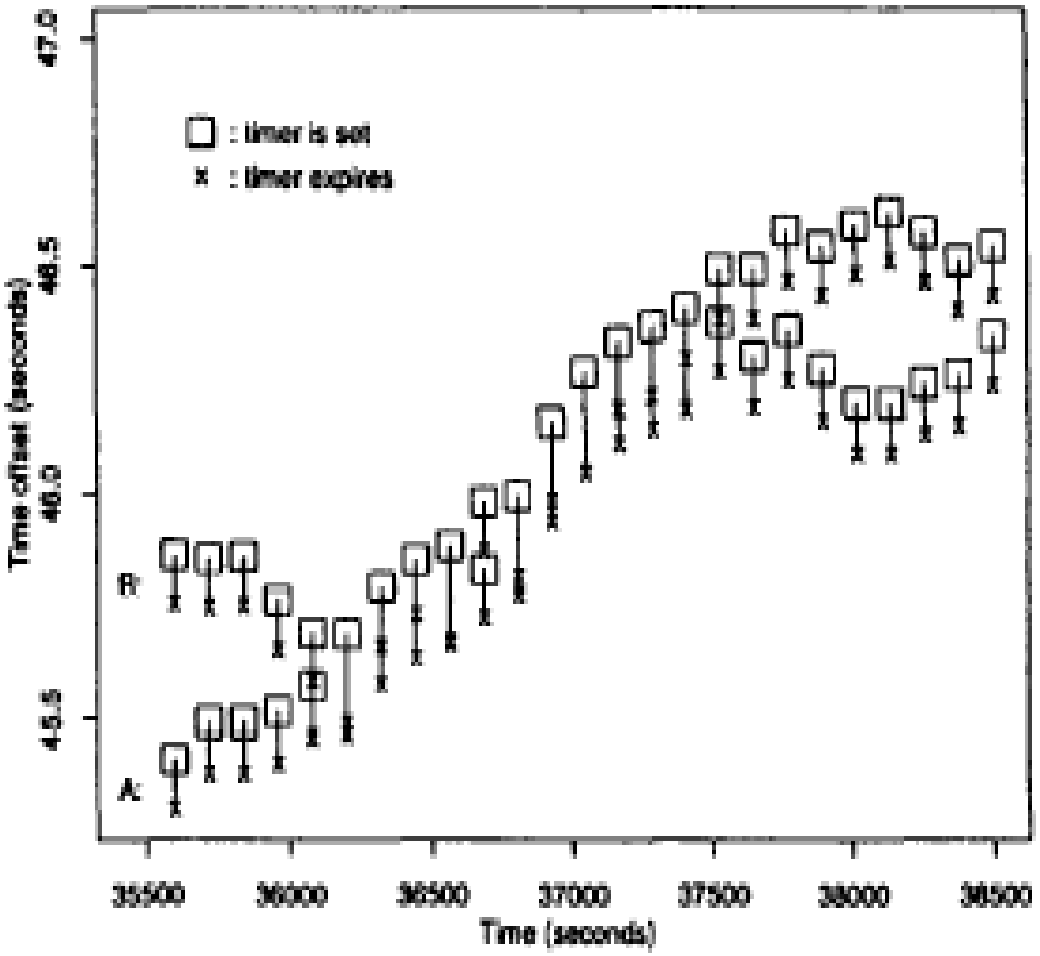
- 20 router (simulati) che si mandano tra loro aggiornamenti
- asse-y: distanza di tempo (offset) nell'invio degli aggiornamenti rispetto inizio periodo
- at tempo t=100,000 *tutti* i router si sono sincronizzati!
- la possibilità che si crei sincronizzazione dipende dai parametri del sistema



T_p = 121sec, T_r=0.1 sec, T_c=0.11 sec

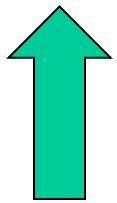
Sincronizzazione dei router

- zoom del grafico precedente
- si noti l'espansione del periodo di processamento
  → periodo più lungo

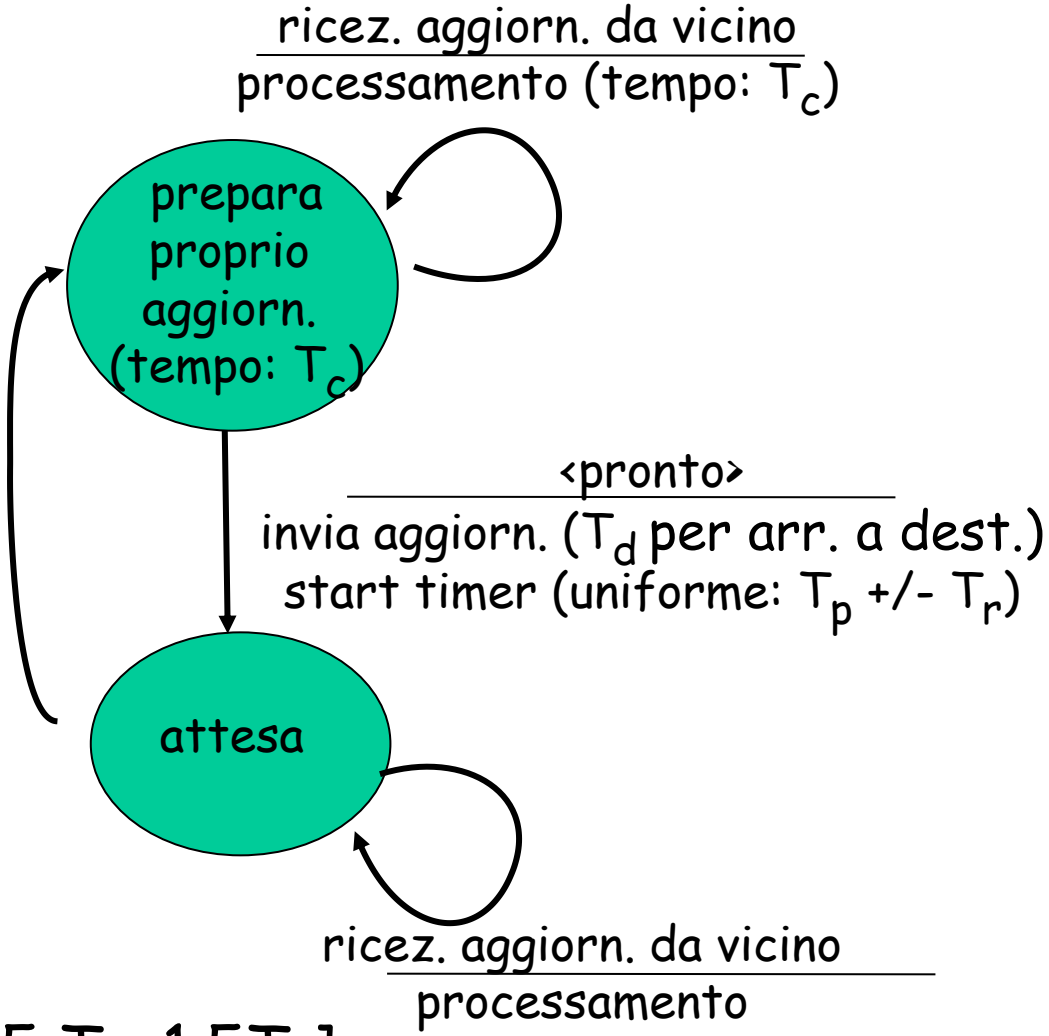


Come evitare la sincronizzazione

- scegliere componente random del timer, T_r grande (e.g., parecchi multipli di T_c)



Aggiungere *abbastanza* randomizzazione da evitare la sincronizzazione



Articolo suggerisce $T_r \in [.5\ T_p, 1.5 T_p]$

Sincronizzazione

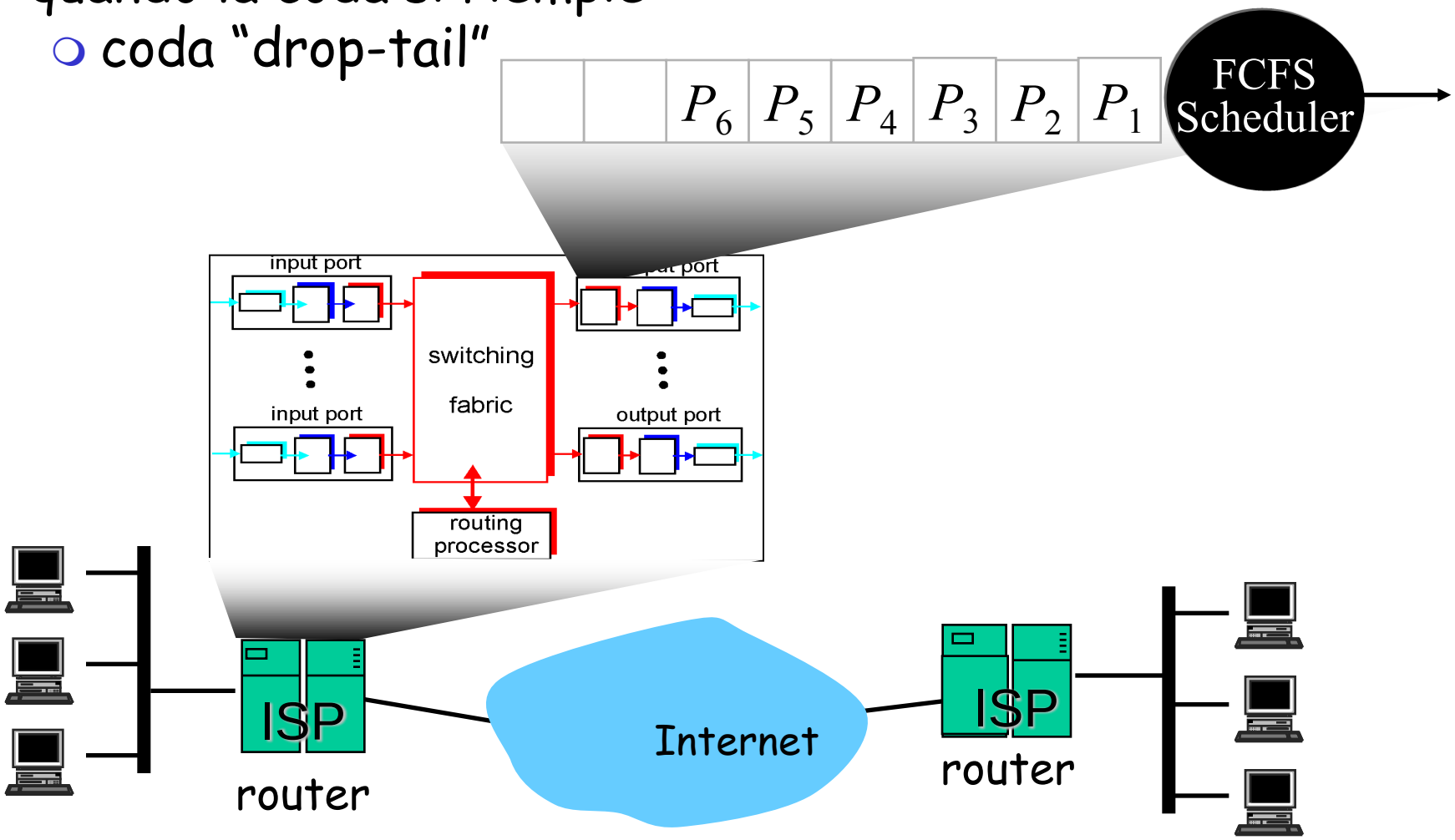
- router si correlano tra loro
- esempi di sincronizzazione spontanea
  - lucciole
  - pesci/uccelli
  - battito cardiaco
  - per saperne di più?

    Steven Strogatz . *Sync*, Hyperion Books, 2003.

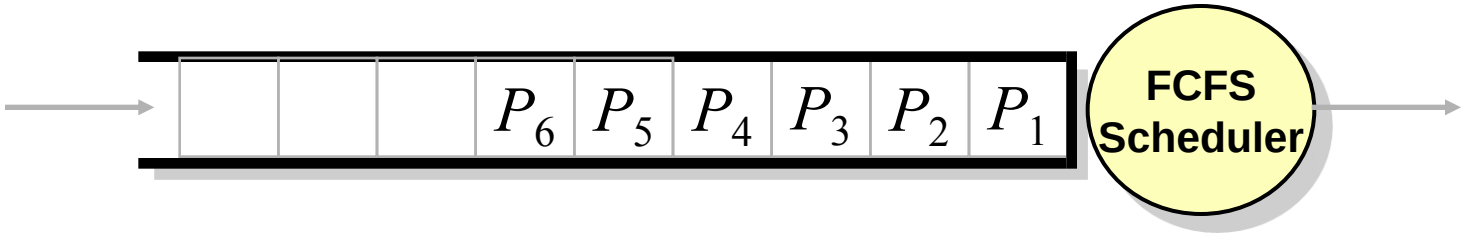
    disponibile anche un TED talk sottotitolato

Randomizzazione nella gestione attiva delle code (Active Queue Management)

- normalmente, i pacchetti verngono scartati solo quando la coda si riempie
  - coda "drop-tail"

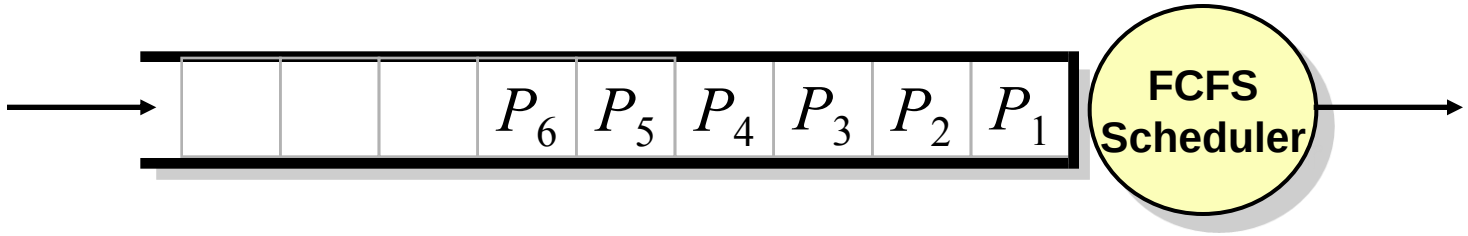


Aspetti negativi della gestione drop-tail



- code lunghe nei router sono una "cattiva cosa"
  - ritardo end-to-end dominato da lunghezza delle code nei commutatori della rete
- permettere che le code vadano in "overflow" è una "cattiva cosa"
  - le connessioni che trasmettono a tasso alto possono uccidere quelle che trasmettono a tasso basso
  - le connessioni possono *sincronizzare* la loro risposta alla congestione

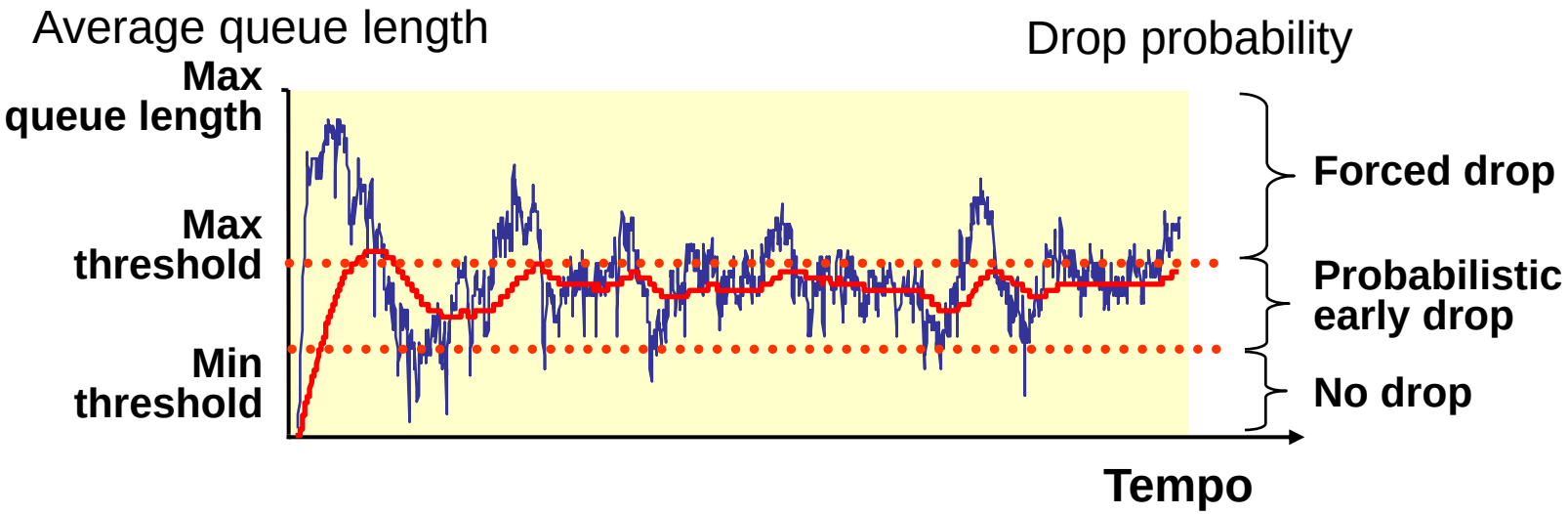
Idea: scarto casuale in anticipo



Quando la lunghezza della coda eccede una certa soglia, i pacchetti vengono scartati con *probabilità* dipendente dalla lunghezza della coda

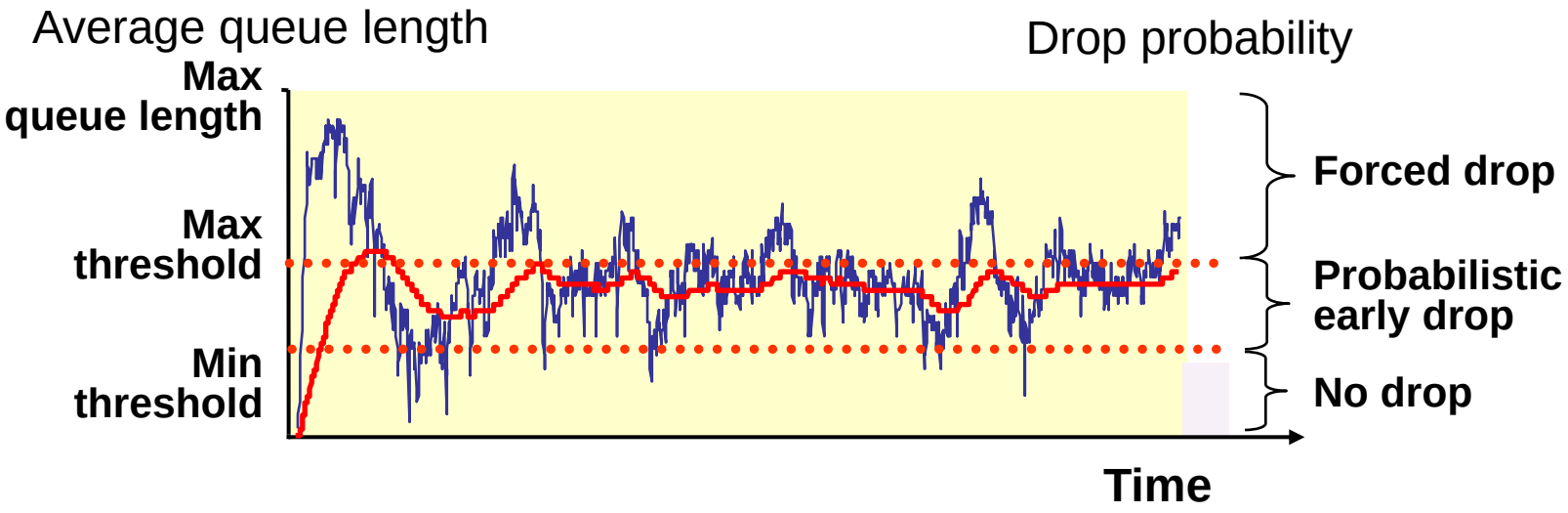
- scarto probabilistico di pacchetti: i flussi subiscono lo *stesso* tasso di perdita
- problema: picchi di traffico (bursts) (che arrivano quando la coda è prossima alla soglia) possono venire troppo penalizzati

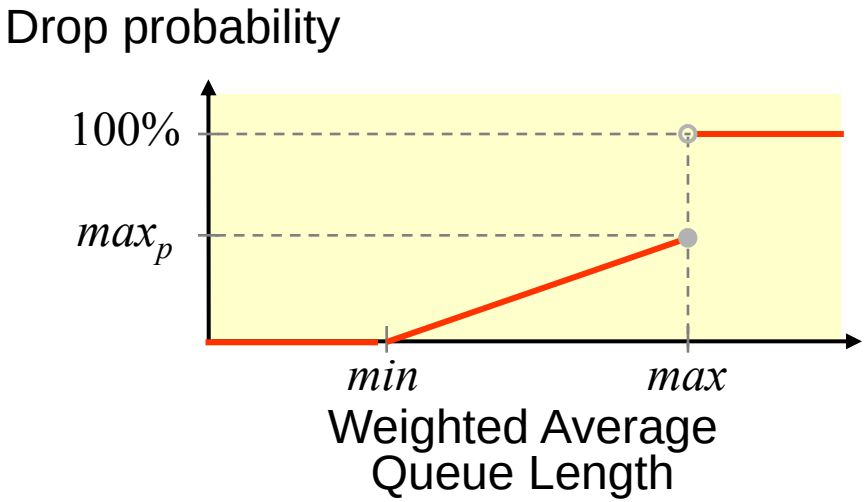
Random Early Detection (RED)



- usa *media* mobile esponenziale per determinare la probabilità di perdita
  - evita di penalizzare troppo i bursts di traffico
  - reagisce a cambiamenti su scala di tempo più lunga
- lega la prob. di perdita alla lunghezza media pesata della coda
  - evita di reagire in modo troppo forte a condizioni di sovraccarico leggere

Random Early Detection (RED)





RED: perchè scarto probabilistico?

- creare una transizione morbida da nessuno-persi a tutti-persi
  - fornisce un avvertimento "gentile" alle sorgenti
- produce lo stesso tasso di perdita a tutti i flussi:
  - usando drop-tail, i flussi a tasso basso possono venire completamente uccisi
- evita la sincronizzazione degli eventi di perdita percepiti dalle sorgenti
  - evita cicli di grandi-perdite seguiti da basso carico

Random Early Detection (RED)

- grande numero (5) di parametri: difficili da configurare (ad es. traffico http)
- vantaggi rispetto a drop-tail FCFS non così significativi
- implementato nei router ma pochissimo utilizzato…

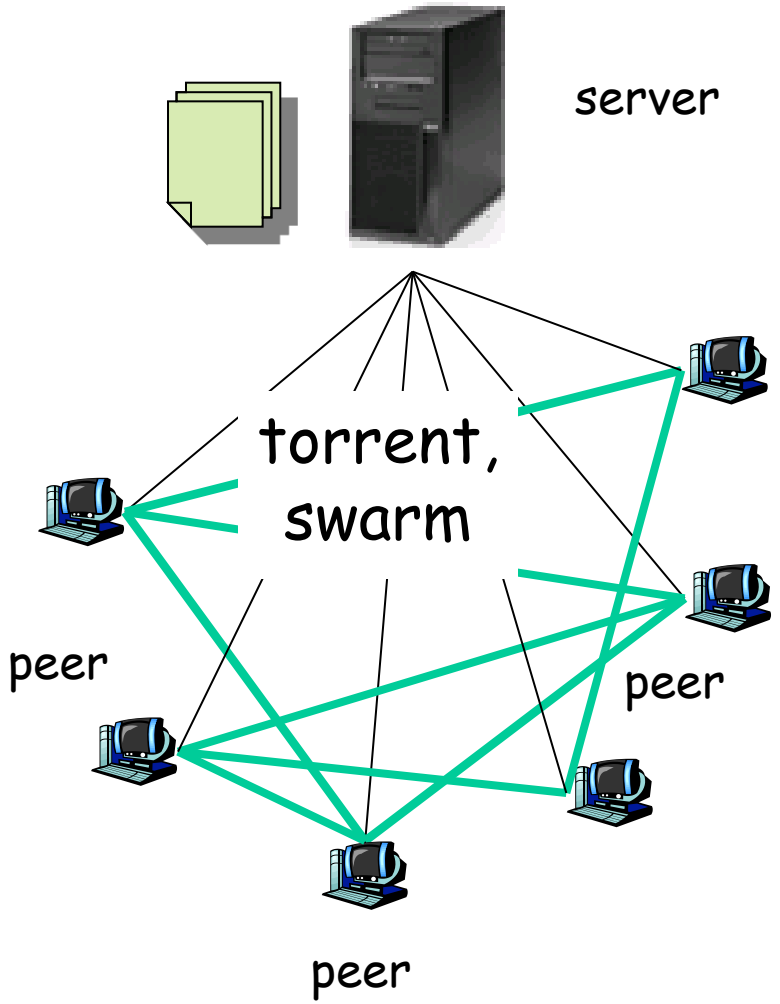
Ci torneremo in seguito!

BitTorrent

- protocollo P2P proposto ~ 2002
- grande successo: approssimativamnete 35% del traffico Internet (secondo CacheLogic)

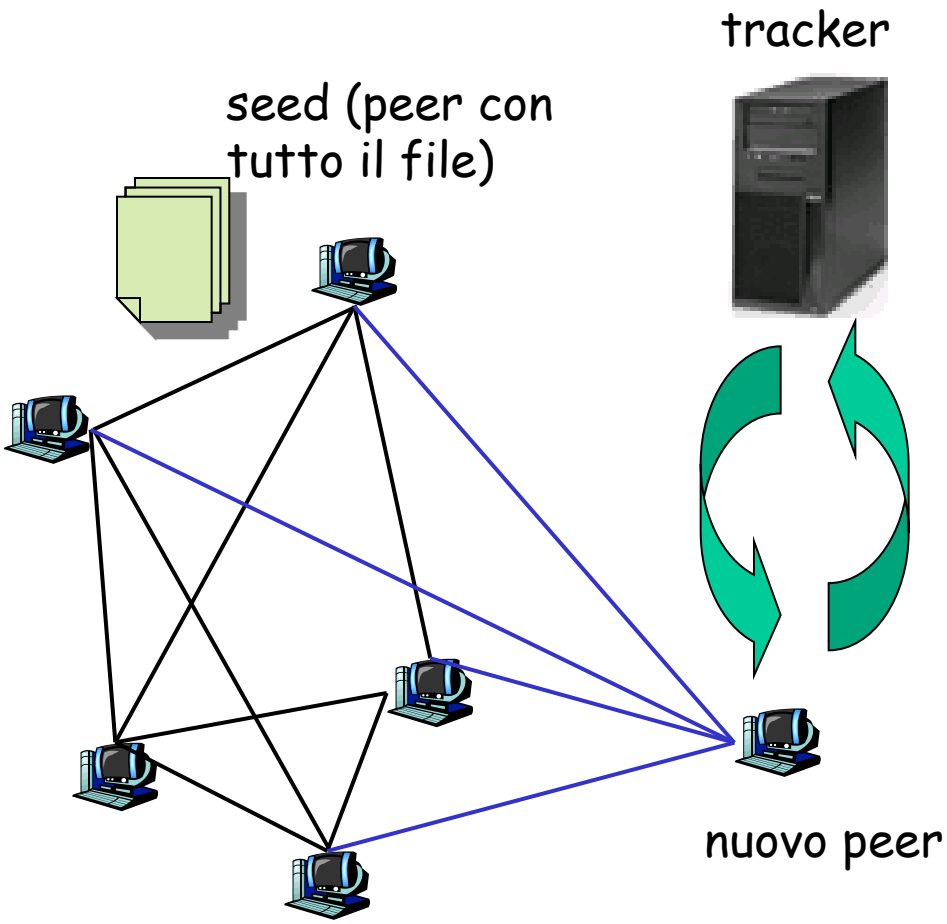
BitTorrent – le basi (Cohen)

- pensato per grandi file (~GB), molti nodi interessati (decine/migliaia o più di clienti)
- divisione del file in piccole parti (256KB).
- utilizzare tutta la capacità di upload dei peers



BitTorrent - funzionamento

- un nuovo peer riceve una lista casuale di altri peer (~50)
- ne sceglie 4 a caso dalla lista



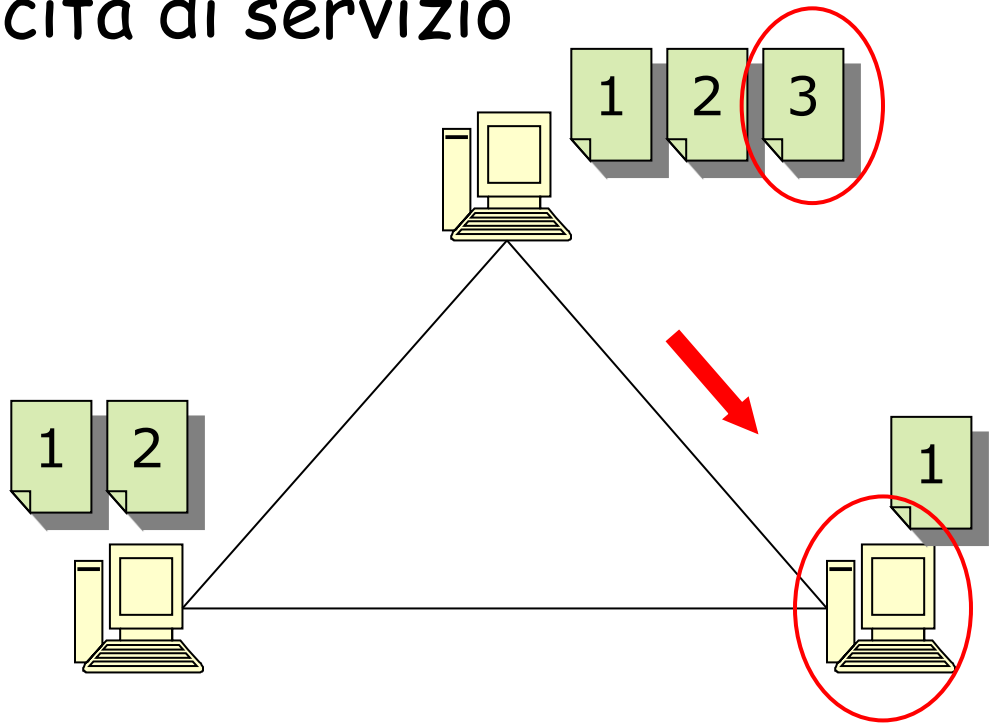
BitTorrent: a chi faccio upload?

Tit-for-tat: upload verso i peer da cui ho maggiormente scaricato durante gli ultimi 30 seconds (per default 4 peers)

$\Rightarrow$ incentivo a fare upload per essere scelti dagli altri peer!

BitTorrent : quale pezzo mandare?

- rarest-first: priorità al pezzo più raro tra quelli posseduti dai vicini
- permette al sistema di utilizzare tutta la capacità di servizio



Selezione dei pezzi (dettagli)

- quando il download comincia: scegli a caso
  - scaricare pezzi completi il più in fretta possibile
  - ottenere qualcosa da "barattare"
- dopo aver ottenuto 4 pezzi: rarest first (locale)
  - ottenere replicazione più veloce dei pezzi rari
  - ottenere qualcosa di valore da scambiare
  - scaricare pezzi unici dai seed
- fase "fine-del-gioco"
  - difesa dal "problema dell'ultimo blocco"
  - invia richieste per sotto-pezzi mancanti a tutti i peer della propria lista
  - invia messaggi di cancellazione alla ricezione di un sotto-pezzo

Problema dell'ultimo pezzo

- verso la fine del download, un peer può avere difficoltà a trovare i pezzi mancanti
- basato su evidenza empirica
- dovuto solo a cattive implementazioni del client, oppure un vero problema del sistema?
- possibile soluzione
  - network coding [Gkantsidis *et al.*, Infocom'05]

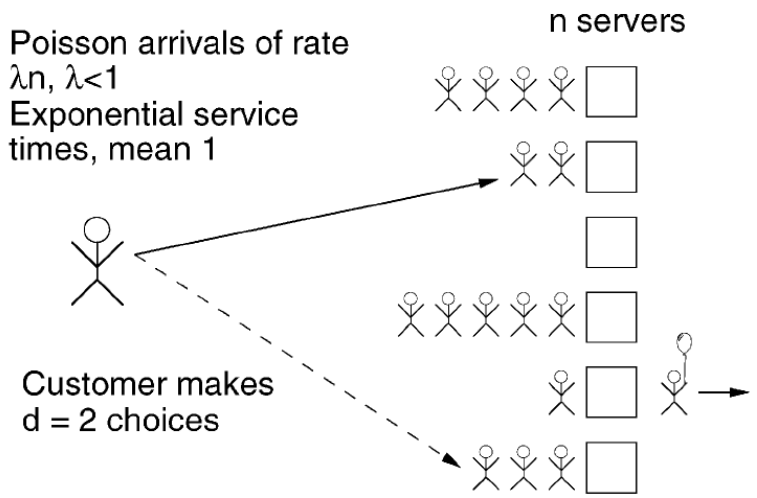
Selezione dei peer

- quattro peer "unchoked" contemporaneamente all'inizio
- ogni 30 secondi
  - scarta peer con il più basso tasso di download
  - unchoke di un nuovo peer casuale
- è un meccanismo con molti scopi
  - permettere l'ingresso di nuovi client
  - bilanciare il carico tra i peer -> ritardi più bassi
  - mantenere la rete connessa: ogni peer ha una probabilità non nulla di interagire con un altro peer qualunque (dello swarm)

La potenza delle due scelte: bilanciamento del carico

Problema di bilanciamento del carico: in quale di N code entrare?

- applicazioni
  - inoltro di richieste a server web
  - instradamento di pacchetti su link/percorsi
- politiche:
  - random
  - *SQ:* join shortest queue: richiede molta informazione
  - *SQ(d):* seleziona *d* code a caso, entra in quella più corta

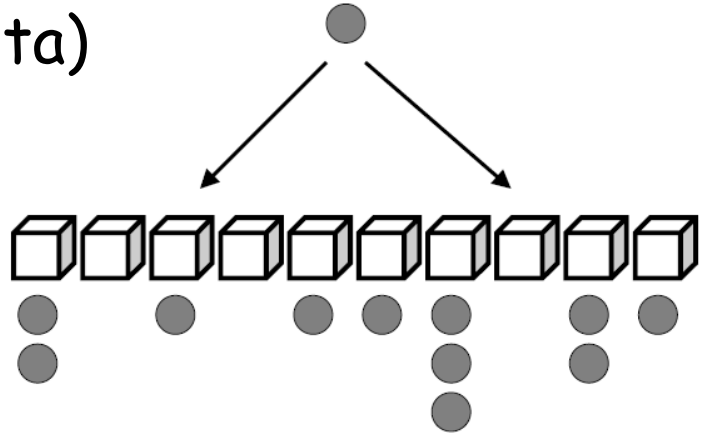


The supermarket model, with $d = 2$.

Formulazione alternativa: problema delle "balls and bins"

Mettere *m* palline in *n* urne (una alla volta)

- random: *SQ(1)*
- *SQ(d):* scegli *d urne a caso,* metti la pallina in quella col minore numero di palline



- per m ≈ n:
  - SQ(1): max carico è O(log n)
  - SQ(2): max carico è O(log log n)
  - SQ(3): solo un fattore costante di miglioramento rispetto a Q(2)

si ottiene un *grande* miglioramento con poca informazione (d=2), e si guadagnerebbe poco avendo ulteriore informazione

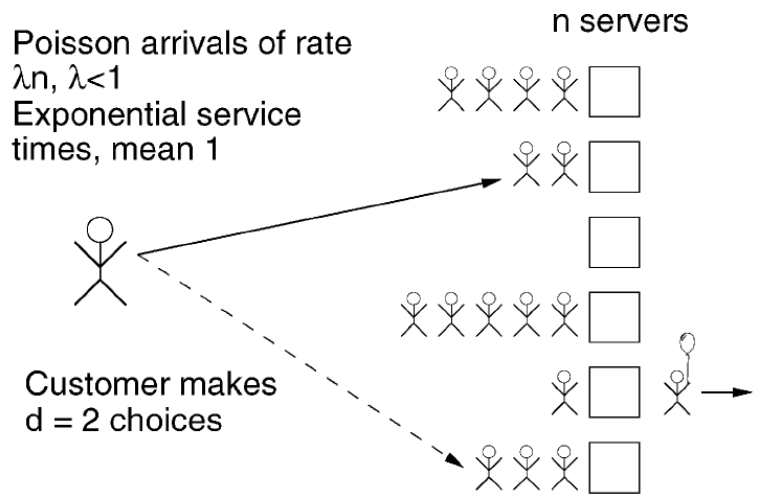
La potenza delle due scelte

- *Rand:* facile da implementare, scarse prestazioni
- *SQ:* difficile da implementare, ottime prestazioni
- *SQ(2):* facile da implementare, funziona esponenzialmente meglio di Rand, ottenendo gran parte del benefici possibili con SQ

Modello del supermarket

- T_d ritardo atteso nel caso di d scelte
- *SQ(1):* $T_1 = \frac{1}{\mu - \lambda}$
- *SQ(d):* $T_d = \frac{\log T_1}{\log d}$, $d > 1$

(con velocità di servizio: $\mu > \lambda$)



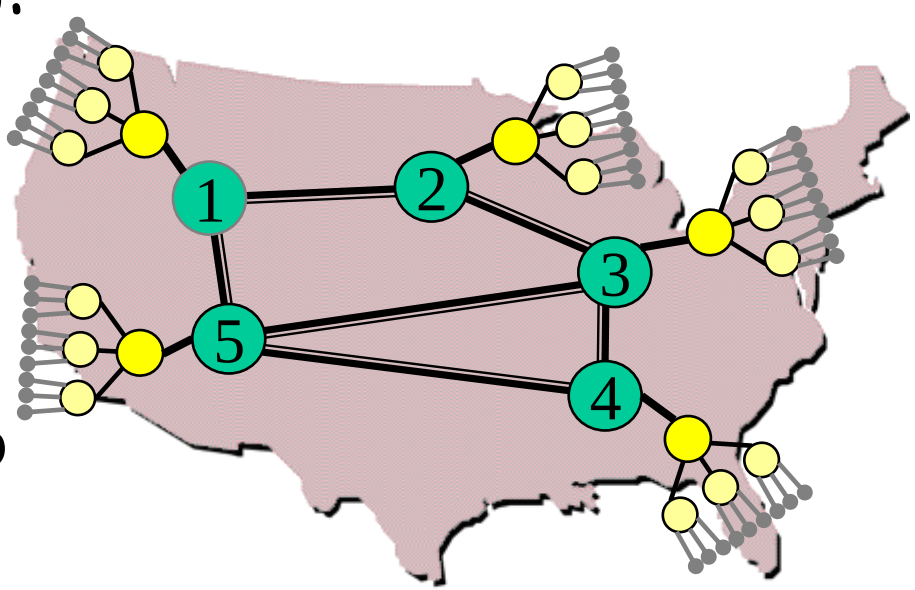
The supermarket model, with $d = 2$.

Ref: "The power of two choices in randomized load balancing,"
M. Mitzenmacher, IEEE Trans. Parallel & Distributed Sys, Oct 2001

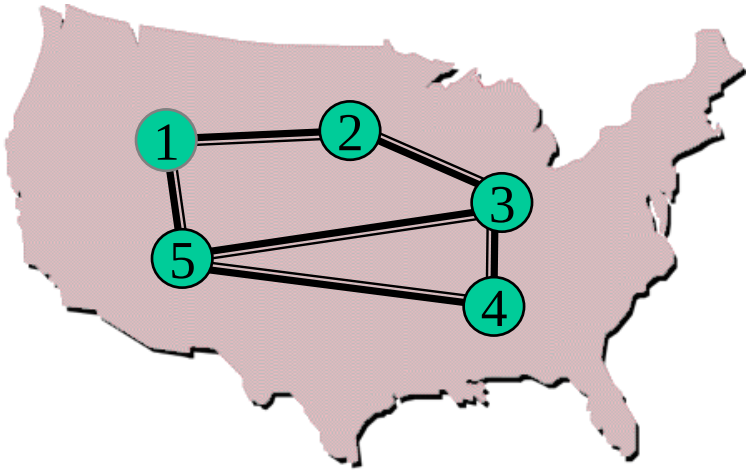
Instradamento randomizzato a due hop

Obiettivo: scegliere percorsi da router di ingresso a router di uscita in modo:

- *robusto* a cambiamenti nel traffico:
  - i link non vadano in sovraccarico, anche se i tassi sorg/dest cambiano
- *con minimo overhead*
  - di aggiornamenti di instradamento al variare del traffico



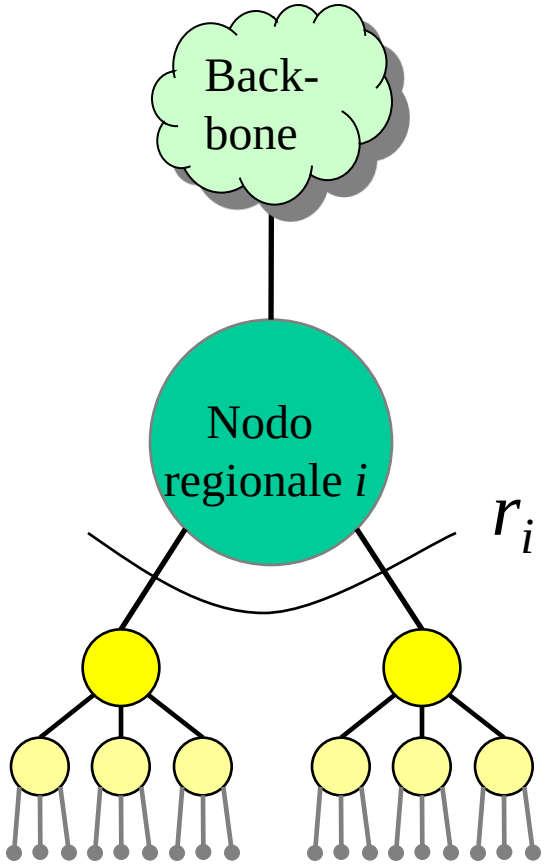
Matrici di traffico



da \ a	1	2	3	4	5
1	0	3	5	3	4
2	2	0	8	1	2
3	1	4	0	7	4
4	8	2	1	0	5
5	4	4	2	5	0

r_i

la matrice di traffico è difficile da stimare

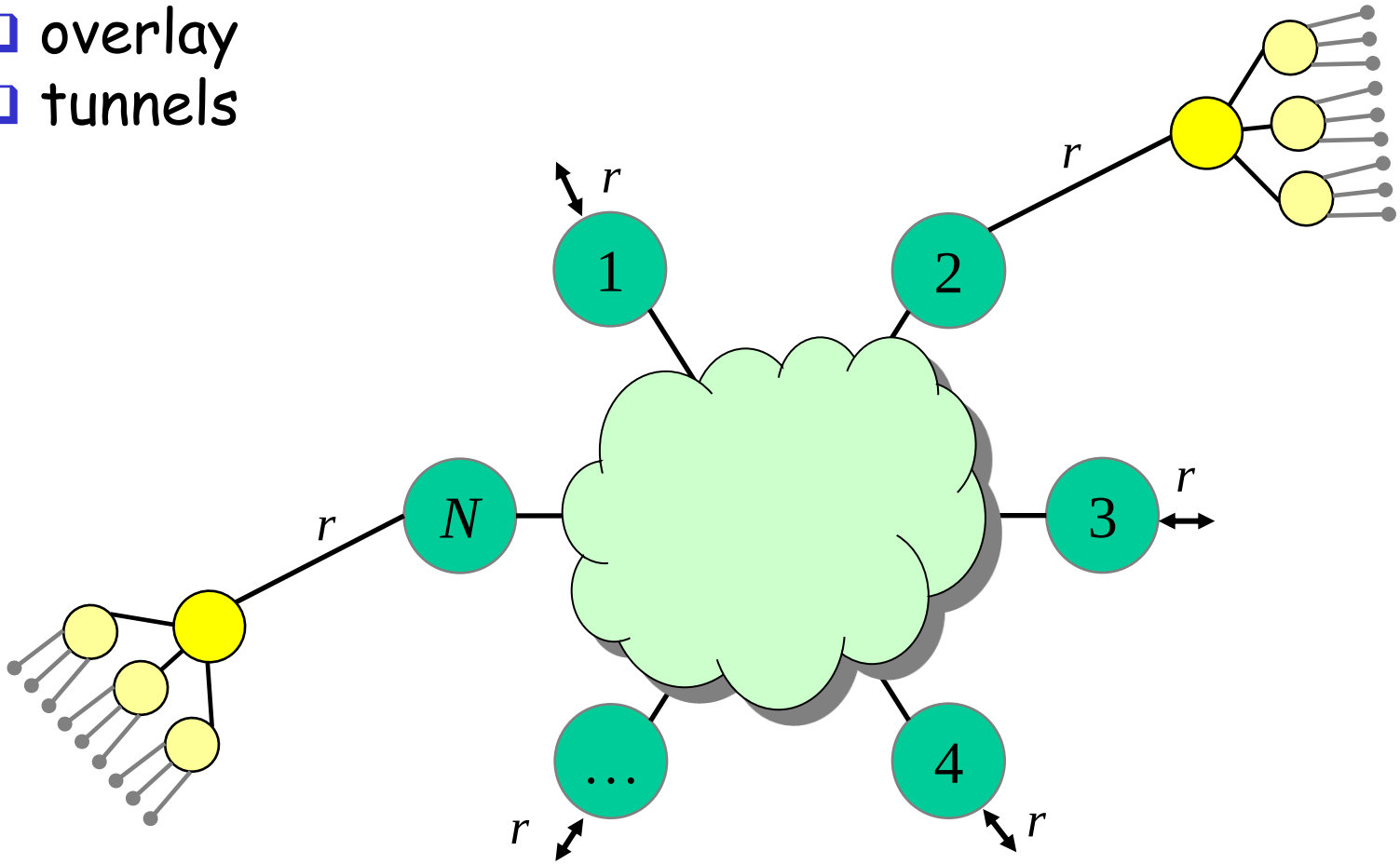


r_i aggregato è più facile da stimare

"Valiant" Load-Balancing

Modello: connessioni logiche 1-hop tra nodi

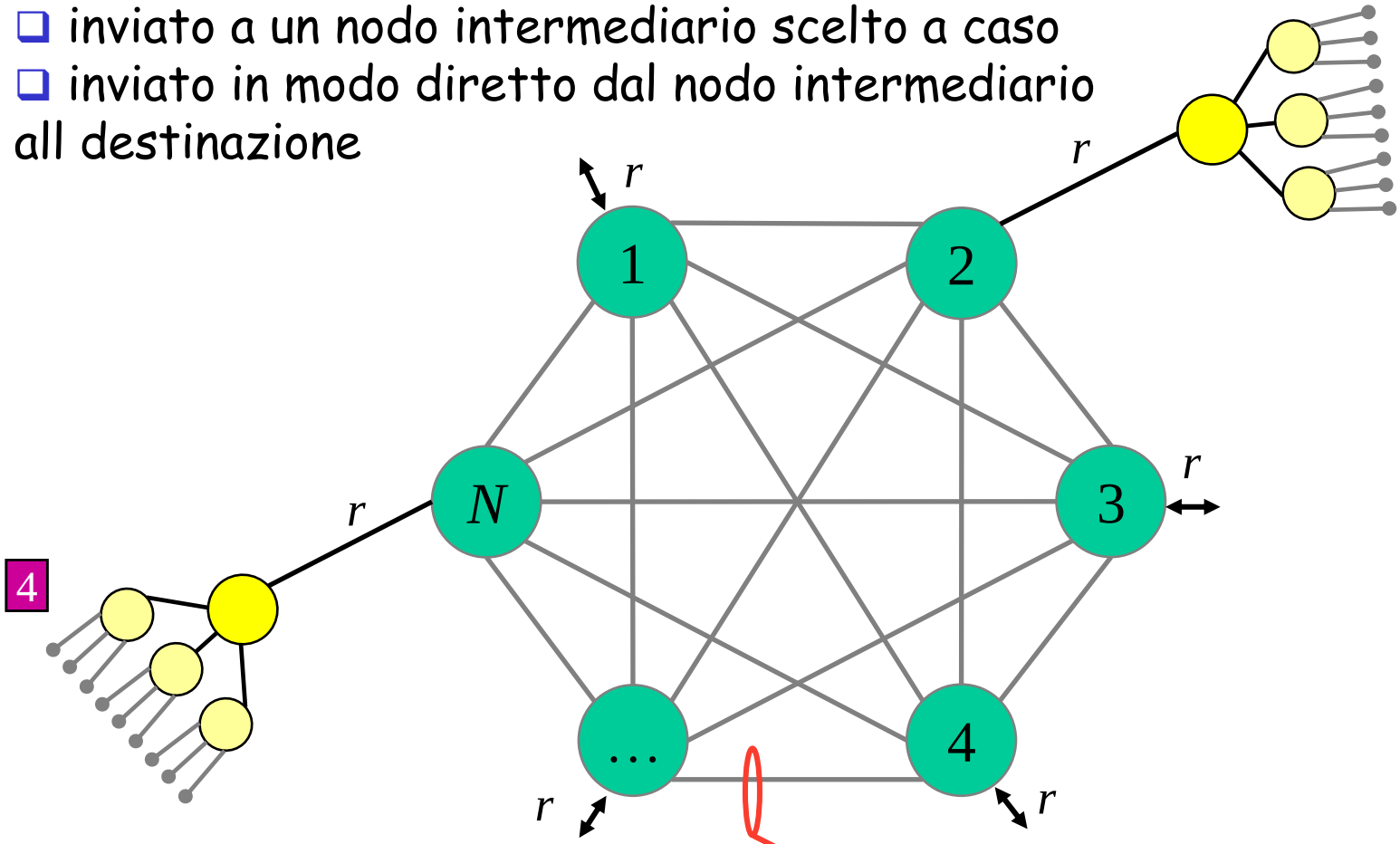
- overlay
- tunnels



Valiant Load-Balancing

Ogni pacchetto che entra nella rete:

- inviato a un nodo intermediario scelto a caso
- inviato in modo diretto dal nodo intermediario all destinazione



D: capacità richiesta su ogni link?

R: no load balancing: r

valiant load balancing: 2r/(N-1)

Discussione

- *sembra inefficiente: perchè non seguire il percorso diretto sorg/dest?*
  -
- quale è il ruolo della *randomizzazione* qui?
  -

Ref: R. Zhang-Shen, N. McKeown, "designing a Predictable Internet Backbone Network with valiant Load Balancing, IWQoS'05, 2005.