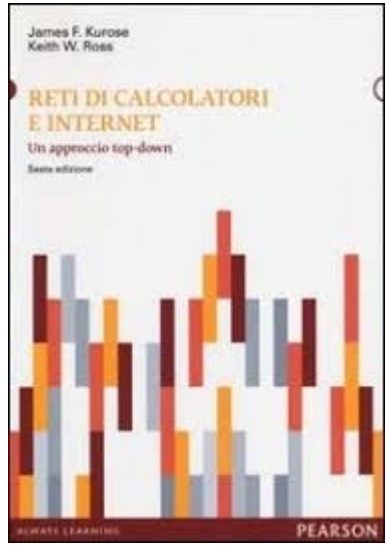
Capitolo 3
Livello di trasporto



Nota per l'utilizzo:

Abbiamo preparato queste slide con l'intenzione di renderle disponibili a tutti (professori, studenti, lettori). Sono in formato PowerPoint in modo che voi possiate aggiungere e cancellare slide (compresa questa) o modificarne il contenuto in base alle vostre esigenze.
Come potete facilmente immaginare, da parte nostra abbiamo fatto *un sacco* di lavoro. In cambio, vi chiediamo solo di rispettare le seguenti condizioni:

- se utilizzate queste slide (ad esempio, in aula) in una forma sostanzialmente inalterata, fate riferimento alla fonte (dopo tutto, ci piacerebbe che la gente usasse il nostro libro!)
- se rendete disponibili queste slide in una forma sostanzialmente inalterata su un sito web, indicate che si tratta di una adattamento (o che sono identiche) delle nostre slide, e inserite la nota relativa al copyright.

Thanks and enjoy! JFK/KWR

Reti di calcolatori e Internet:
Un approccio top-down

6ª edizione
Jim Kurose, Keith Ross

Pearson Paravia Bruno Mondadori Spa
©2013

Capitolo 3: Livello di trasporto

Obiettivi:

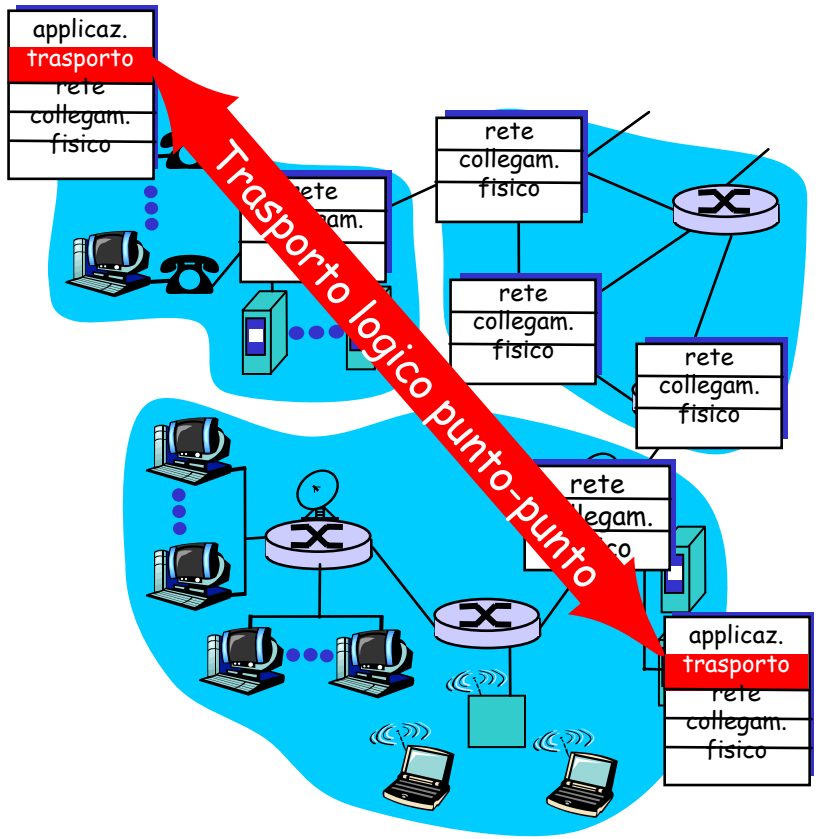
- Capire i principi che sono alla base dei servizi del livello di trasporto:
  - multiplexing/demultiplexing
  - trasferimento dati affidabile
  - controllo di flusso
  - controllo di congestione
- Descrivere i protocolli del livello di trasporto di Internet:
  - UDP: trasporto senza connessione
  - TCP: trasporto orientato alla connessione
  - controllo di congestione TCP

Capitolo 3: Livello di trasporto

- 3.1 Servizi a livello di trasporto
- 3.2 Multiplexing e demultiplexing
- 3.3 Trasporto senza connessione: UDP
- 3.4 Principi del trasferimento dati affidabile
- 3.5 Trasporto orientato alla connessione: TCP
  - struttura dei segmenti
  - trasferimento dati affidabile
  - controllo di flusso
  - gestione della connessione
- 3.6 Principi sul controllo di congestione
- 3.7 Controllo di congestione TCP

Servizi e protocolli di trasporto

- Forniscono la *comunicazione logica* tra processi applicativi di host differenti
- I protocolli di trasporto vengono eseguiti nei sistemi terminali
  - lato invio: scinde i messaggi in segmenti e li passa al livello di rete
  - lato ricezione: riassembla i segmenti in messaggi e li passa al livello di applicazione
- Più protocolli di trasporto sono a disposizione delle applicazioni
  - Internet: TCP e UDP



Relazione tra livello di trasporto e livello di rete

- *livello di rete:* comunicazione logica tra host
- *livello di trasporto:* comunicazione logica tra processi
  - si basa sui servizi del livello di rete

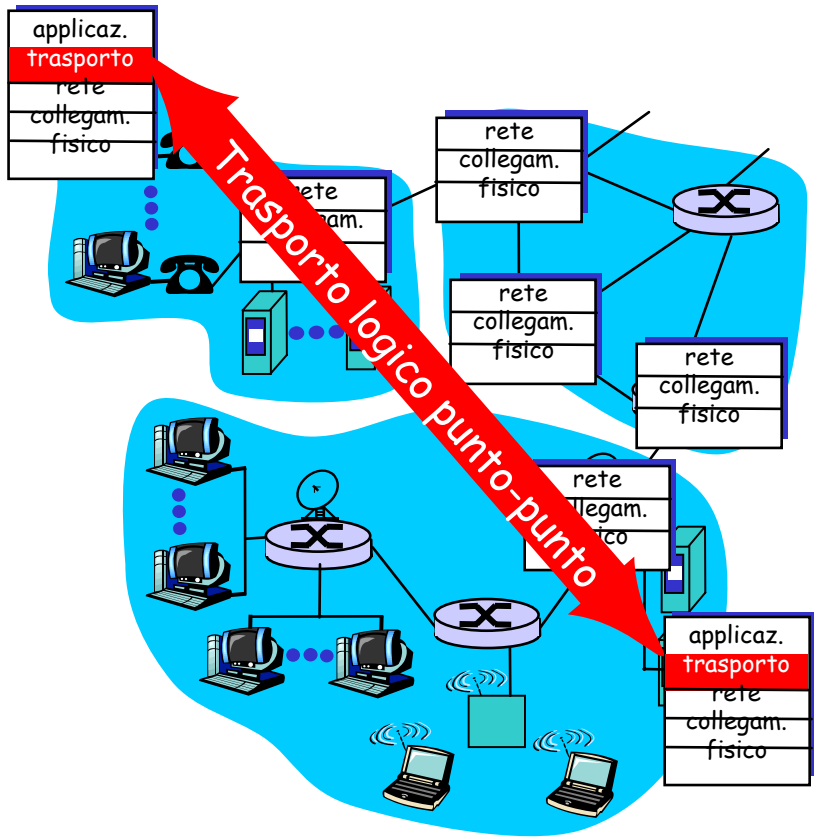
Analogia con la posta ordinaria:

12 ragazzi inviano lettere a 12 ragazzi

- processi = ragazzi
- messaggi delle applicazioni = lettere nelle buste
- host = case
- protocollo di trasporto = Anna e Andrea
- protocollo del livello di rete = servizio postale

Protocolli del livello di trasporto in Internet

- Affidabile, consegne nell'ordine originario (TCP)
  - controllo di congestione
  - controllo di flusso
  - setup della connessione
- Inaffidabile, consegne senz'ordine: UDP
  - estensione senza fronzoli del servizio di consegna a massimo sforzo
- Servizi non disponibili:
  - garanzia su ritardi
  - garanzia su ampiezza di banda



Capitolo 3: Livello di trasporto

- 3.1 Servizi a livello di trasporto
- 3.2 Multiplexing e demultiplexing
- 3.3 Trasporto senza connessione: UDP
- 3.4 Principi del trasferimento dati affidabile
- 3.5 Trasporto orientato alla connessione: TCP
  - struttura dei segmenti
  - trasferimento dati affidabile
  - controllo di flusso
  - gestione della connessione
- 3.6 Principi sul controllo di congestione
- 3.7 Controllo di congestione TCP

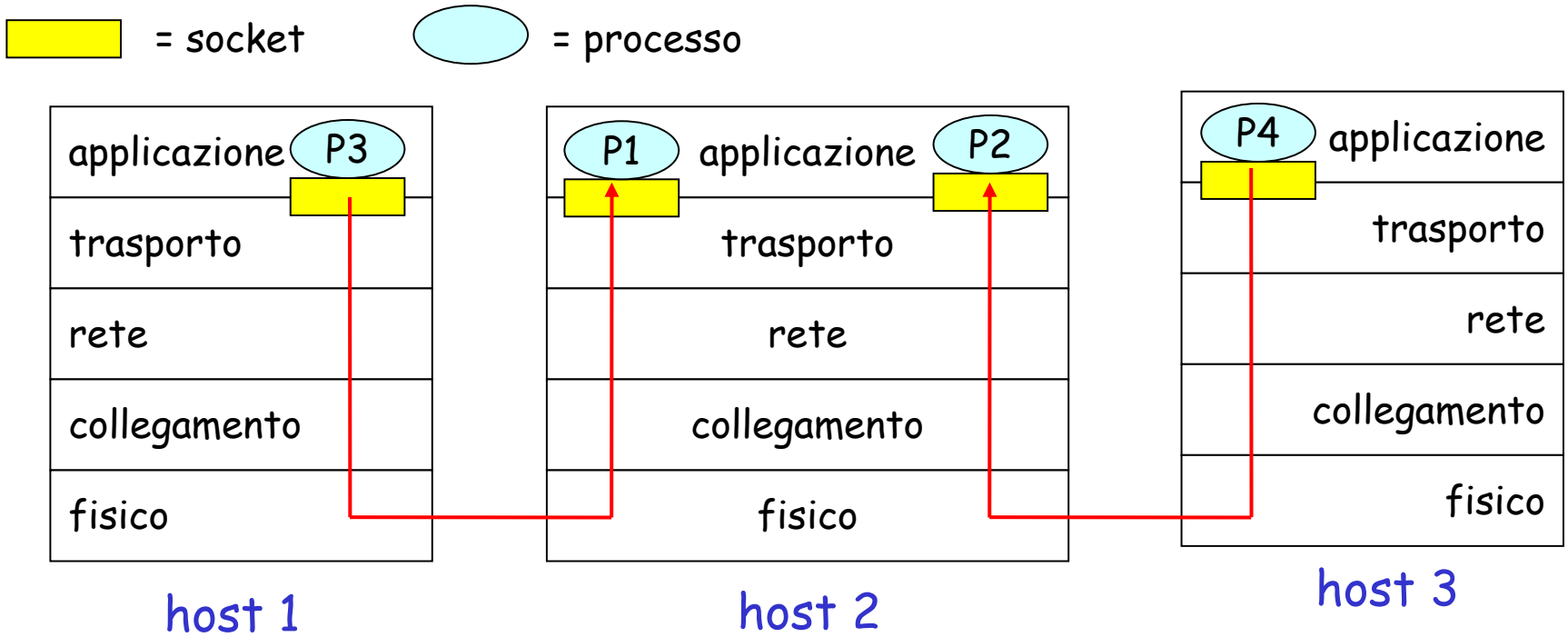
Multiplexing/demultiplexing

Demultiplexing nell'host ricevente:

consegnare i segmenti ricevuti alla socket appropriata

Multiplexing nell'host mittente:

raccogliere i dati da varie socket, incapsularli con l'intestazione (utilizzati poi per il demultiplexing)



Come funziona il demultiplexing

- L'host riceve i datagrammi IP
  - ogni datagramma ha un indirizzo IP di origine e un indirizzo IP di destinazione
  - ogni datagramma trasporta 1 segmento a livello di trasporto
  - ogni segmento ha un numero di porta di origine e un numero di porta di destinazione
- L'host usa gli indirizzi IP e i numeri di porta per inviare il segmento alla socket appropriata

Struttura del segmento TCP/UDP

Demultiplexing senza connessione

- Crea le socket con i numeri di porta:

```
DatagramSocket mySocket1 = new
   DatagramSocket(9111);
DatagramSocket mySocket2 = new
   DatagramSocket(9222);
```

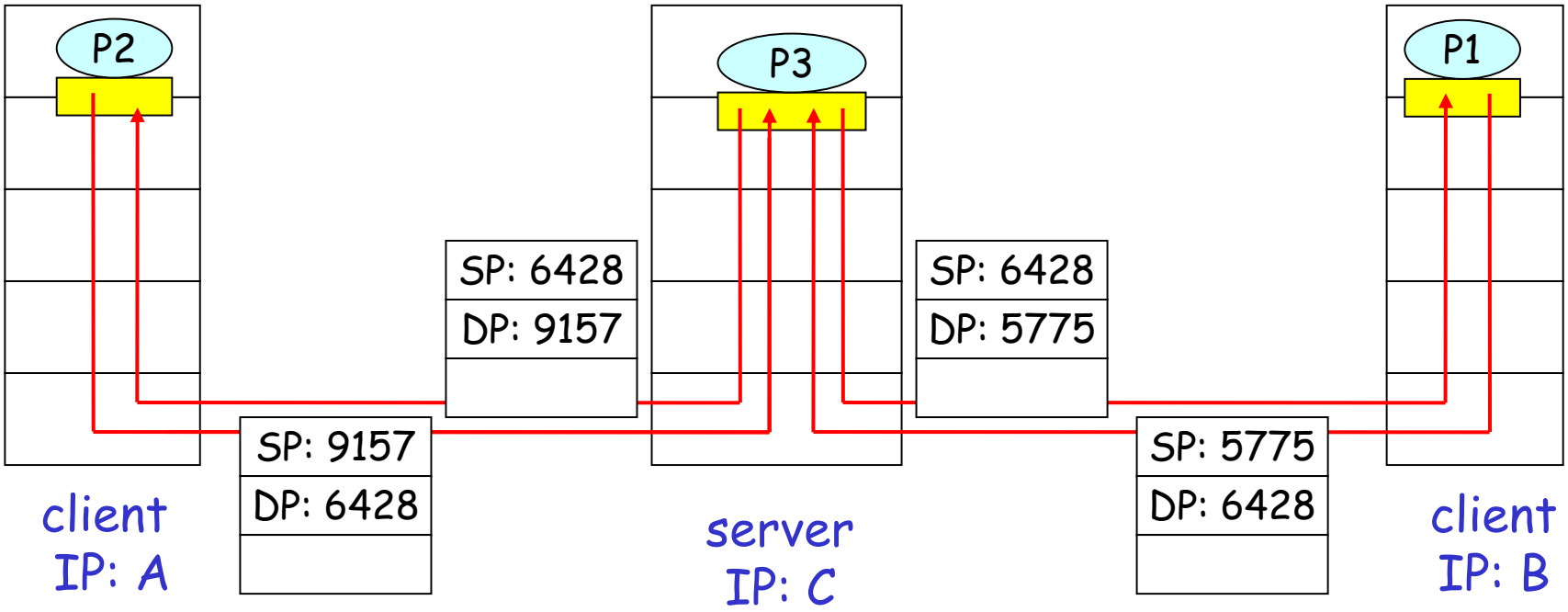
- La socket UDP è identificata da 2 parametri:

  (indirizzo IP di destinazione, numero della porta di destinazione)

- Quando l'host riceve il segmento UDP:
  - controlla il numero della porta di destinazione nel segmento
  - invia il segmento UDP alla socket con quel numero di porta
- I datagrammi IP con indirizzi IP di origine e/o numeri di porta di origine differenti vengono inviati alla stessa socket

Demultiplexing senza connessione (continua)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

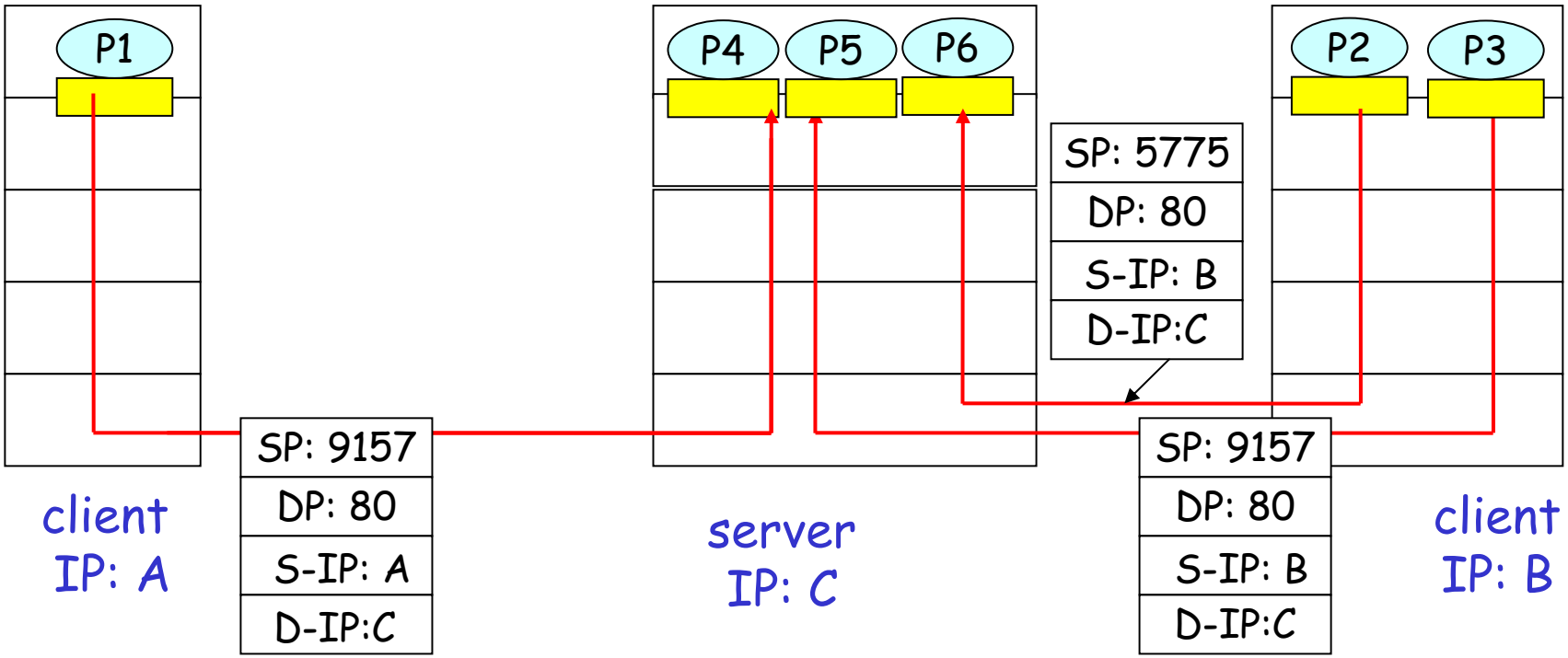


SP fornisce "l'indirizzo di ritorno"

Demultiplexing orientato alla connessione

- La socket TCP è identificata da 4 parametri:
  - indirizzo IP di origine
  - numero di porta di origine
  - indirizzo IP di destinazione
  - numero di porta di destinazione
- L'host ricevente usa i quattro parametri per inviare il segmento alla socket appropriata
- Un host server può supportare più socket TCP contemporanee:
  - ogni socket è identificata dai suoi 4 parametri
- I server web hanno socket differenti per ogni connessione client
  - con HTTP non-persistente si avrà una socket differente per ogni richiesta

Demultiplexing orientato alla connessione (continua)



Capitolo 3: Livello di trasporto

- 3.1 Servizi a livello di trasporto
- 3.2 Multiplexing e demultiplexing
- 3.3 Trasporto senza connessione: UDP
- 3.4 Principi del trasferimento dati affidabile
- 3.5 Trasporto orientato alla connessione: TCP
  - struttura dei segmenti
  - trasferimento dati affidabile
  - controllo di flusso
  - gestione della connessione
- 3.6 Principi sul controllo di congestione
- 3.7 Controllo di congestione TCP

UDP: User Datagram Protocol [RFC 768]

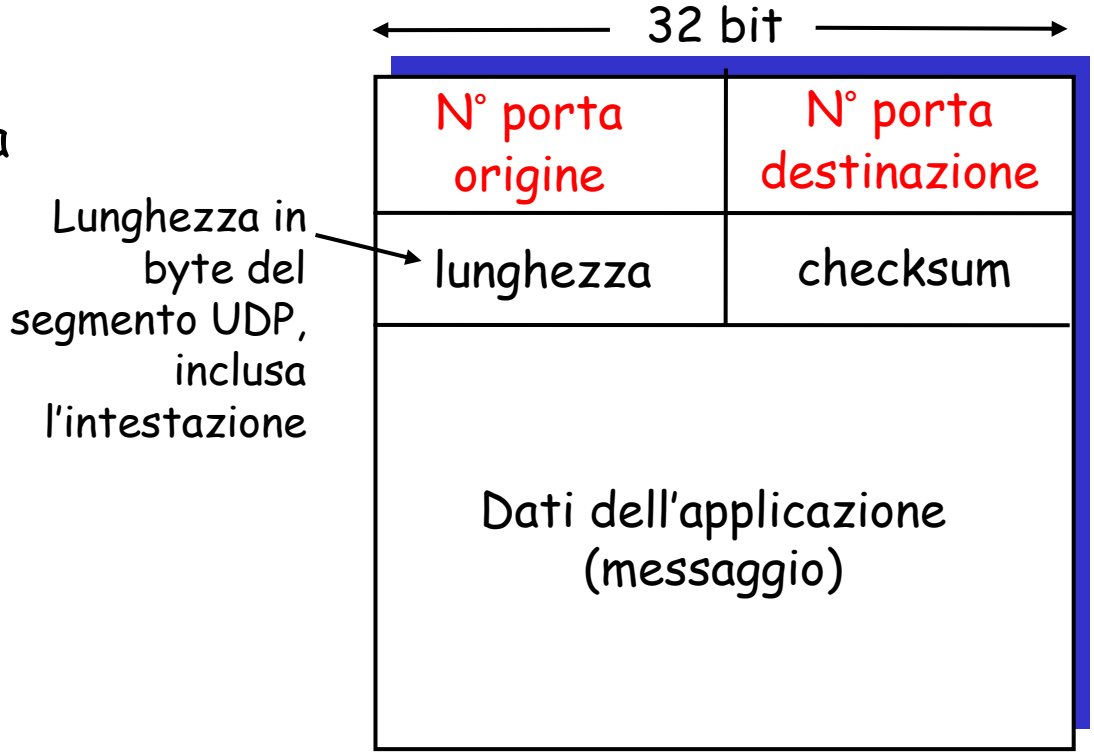
- Protocollo di trasporto "senza fronzoli"
- Servizio di consegna "a massimo sforzo", i segmenti UDP possono essere:
  - perduti
  - consegnati fuori sequenza all'applicazione
- *Senza connessione:*
  - no handshaking tra mittente e destinatario UDP
  - ogni segmento UDP è gestito indipendentemente dagli altri

Perché esiste UDP?

- Nessuna connessione stabilita (che potrebbe aggiungere un ritardo)
- Semplice: nessuno stato di connessione nel mittente e destinatario
- Intestazioni di segmento corte
- Senza controllo di congestione: UDP può sparare dati a raffica

UDP: altro

- Utilizzato spesso nelle applicazioni multimediali
  - tollera piccole perdite
  - sensibile alla frequenza
- Altri impieghi di UDP
  - DNS
  - SNMP
- Trasferimento affidabile con UDP: aggiungere affidabilità al livello di applicazione
  - Recupero degli errori delle applicazioni!



Struttura del segmento UDP

Checksum UDP

Obiettivo: rilevare gli "errori" (bit alterati) nel segmento trasmesso

Mittente:

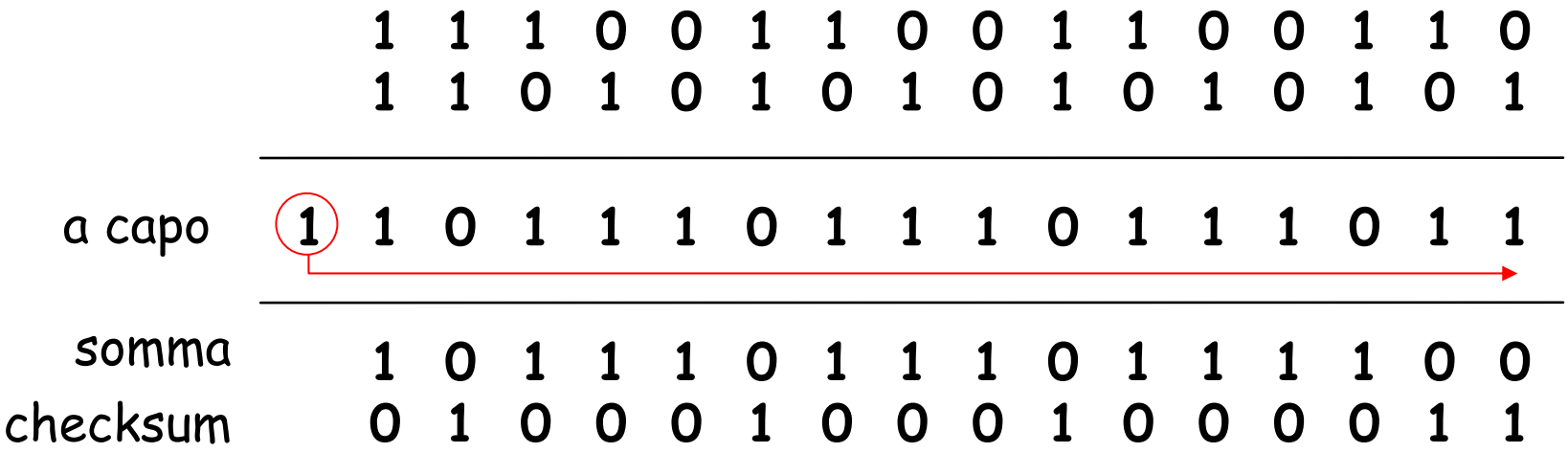
- Tratta il contenuto del segmento come una sequenza di interi da 16 bit
- checksum: somma (complemento a 1) i contenuti del segmento
- Il mittente pone il valore della checksum nel campo checksum del segmento UDP

Ricevente:

- calcola la checksum del segmento ricevuto
- controlla se la checksum calcolata è uguale al valore del campo checksum:
  - No - errore rilevato
  - Sì - nessun errore rilevato. *Ma potrebbero esserci errori nonostante questo?* Altro più avanti …

Esempio di checksum

- Nota
  - Quando si sommano i numeri, un riporto dal bit più significativo deve essere sommato al risultato
- Esempio: sommare due interi da 16 bit



Capitolo 3: Livello di trasporto

- 3.1 Servizi a livello di trasporto
- 3.2 Multiplexing e demultiplexing
- 3.3 Trasporto senza connessione: UDP
- 3.4 Principi del trasferimento dati affidabile
- 3.5 Trasporto orientato alla connessione: TCP
  - struttura dei segmenti
  - trasferimento dati affidabile
  - controllo di flusso
  - gestione della connessione
- 3.6 Principi sul controllo di congestione
- 3.7 Controllo di congestione TCP

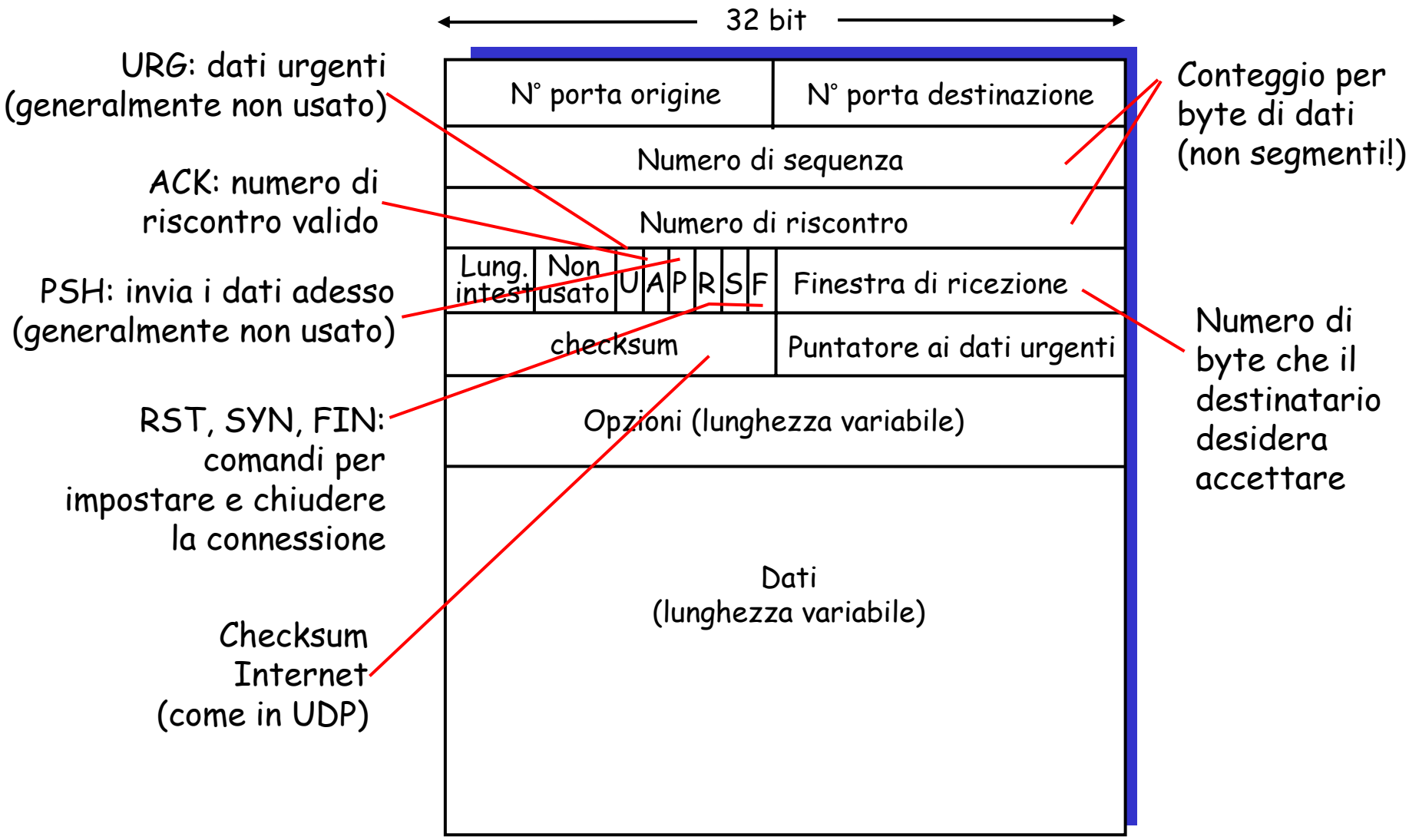
TCP: Panoramica

RFC: 793, 1122, 1323, 2018, 2581

- punto-punto:
  - un mittente, un destinatario
- flusso di byte affidabile, in sequenza:
  - nessun "confine ai messaggi"
- pipeline:
  - il controllo di flusso e di congestione TCP definiscono la dimensione della finestra
- *buffer d'invio e di ricezione*

- full duplex:
  - flusso di dati bidirezionale nella stessa connessione
  - MSS: dimensione massima di segmento (maximum segment size)
- orientato alla connessione:
  - l'handshaking (scambio di messaggi di controllo) inizializza lo stato del mittente e del destinatario prima di scambiare i dati
- flusso controllato:
  - il mittente non sovraccarica il destinatario

Struttura dei segmenti TCP



Numeri di sequenza e ACK di TCP

Numeri di sequenza:

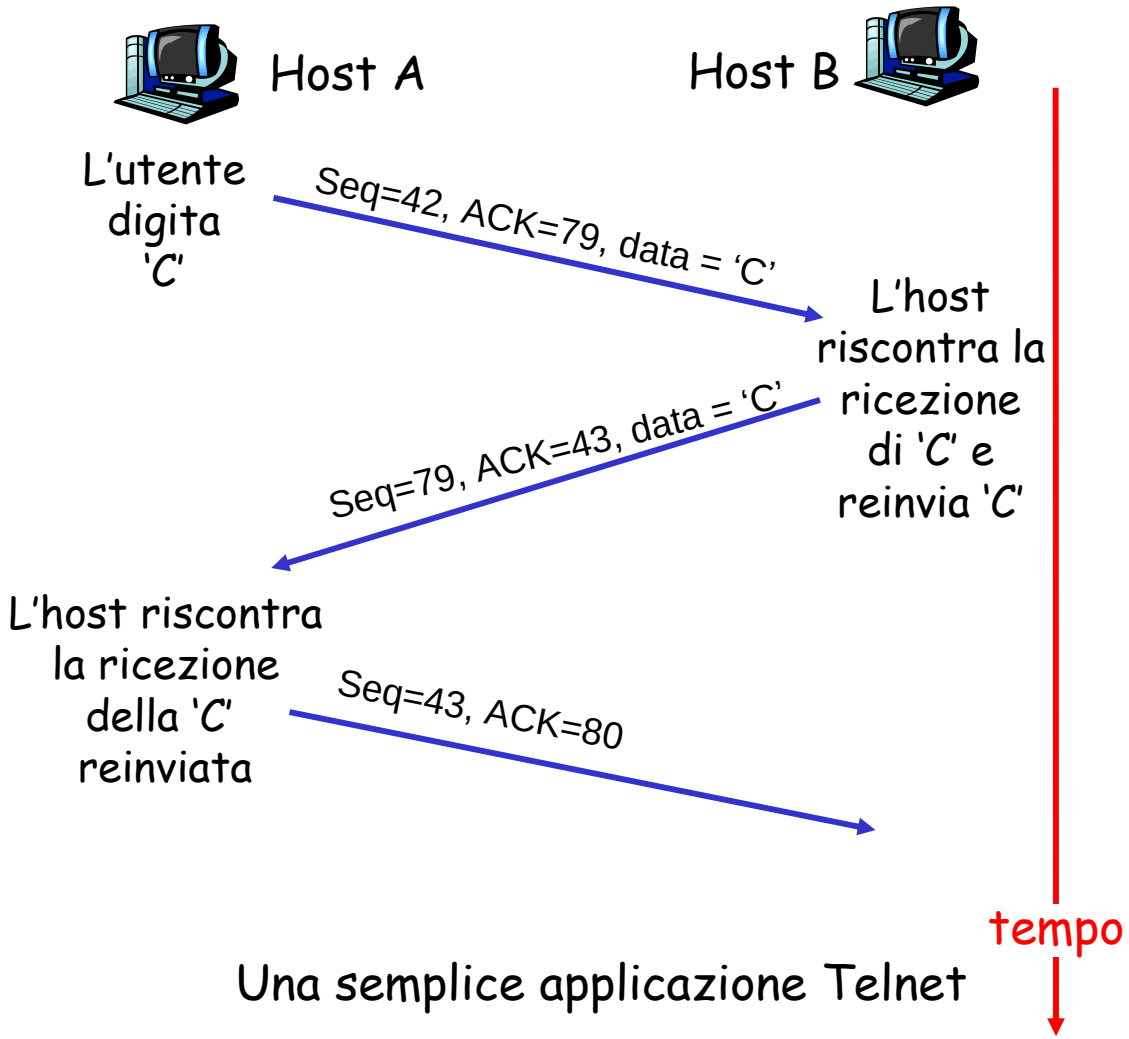
- "numero" del primo byte del segmento nel flusso di byte

ACK:

- numero di sequenza del prossimo byte atteso dall'altro lato
- ACK cumulativo

D: come gestisce il destinatario i segmenti fuori sequenza?

- R: la specifica TCP non lo dice – dipende dall'implementatore



Una semplice applicazione Telnet

TCP: tempo di andata e ritorno e timeout

D: come impostare il valore del timeout di TCP?

- Più grande di RTT
  - ma RTT varia
- Troppo piccolo: timeout prematuro
  - ritrasmissioni non necessarie
- Troppo grande: reazione lenta alla perdita dei segmenti

D: come stimare RTT?

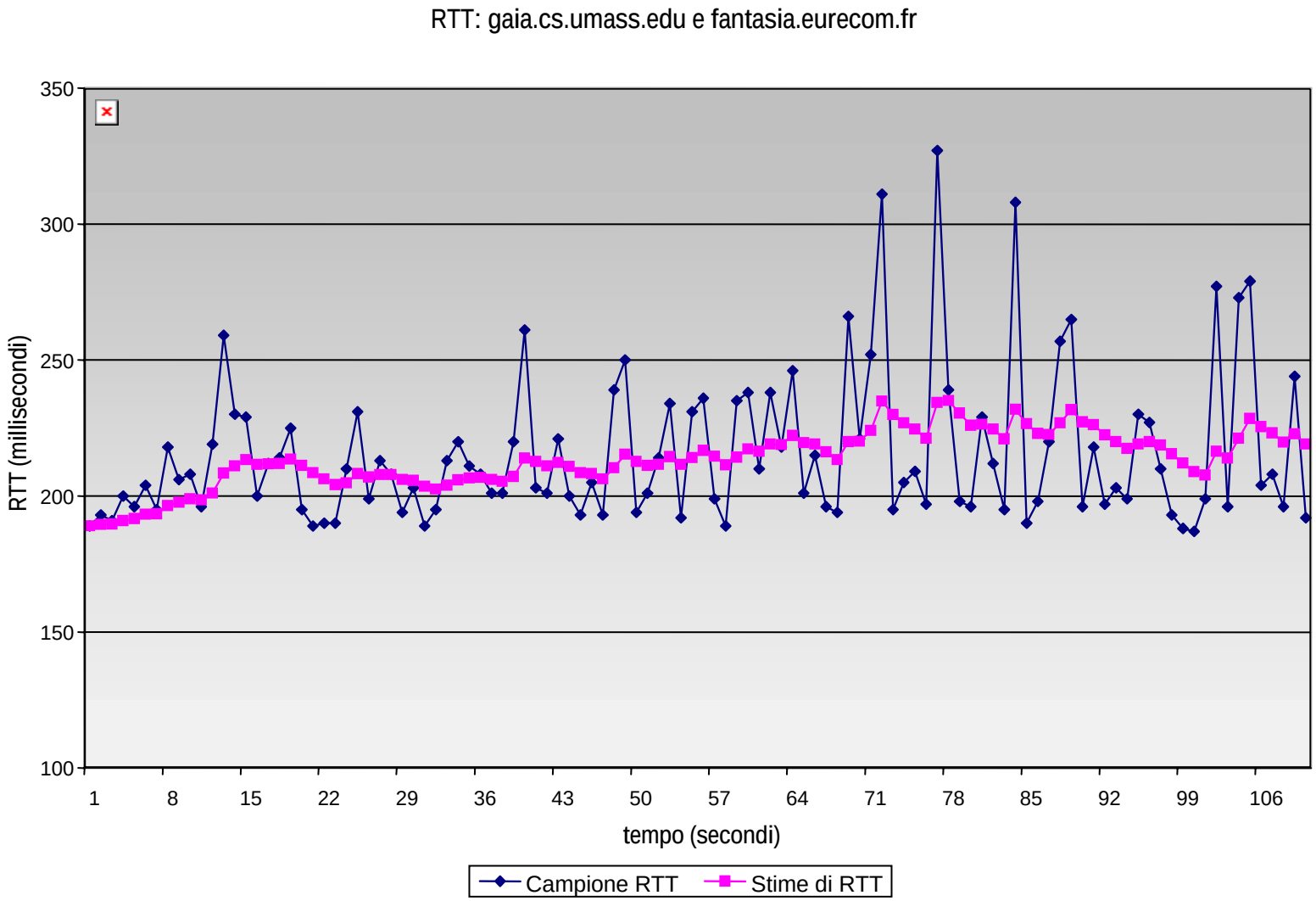
- **SampleRTT**: tempo misurato dalla trasmissione del segmento fino alla ricezione di ACK
  - ignora le ritrasmissioni
- **SampleRTT** varia, quindi occorre una stima "più livellata" di RTT
  - media di più misure recenti, non semplicemente il valore corrente di **SampleRTT**

TCP: tempo di andata e ritorno e timeout

EstimatedRTT = (1 - α)*EstimatedRTT + α*SampleRTT

- Media mobile esponenziale ponderata
- L'influenza dei vecchi campioni decresce esponenzialmente
- Valore tipico: α = 0,125

Esempio di stima di RTT:



TCP: tempo di andata e ritorno e timeout

Impostazione del timeout

- **EstimtedRTT** più un "margine di sicurezza"
  - grande variazione di **EstimatedRTT** -> margine di sicurezza maggiore
- Stimare innanzitutto di quanto SampleRTT si discosta da EstimatedRTT:

DevRTT = (1-β)*DevRTT +
β*|SampleRTT-EstimatedRTT|

(tipicamente, β = 0,25)

Poi impostare l'intervallo di timeout:

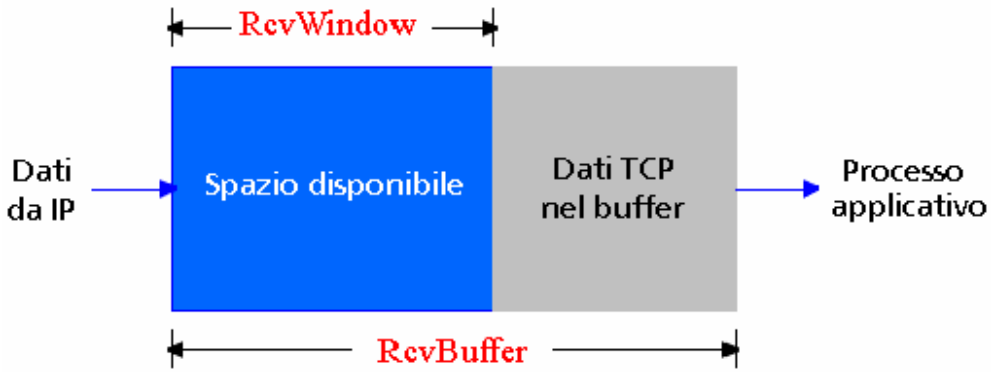
TimeoutInterval = EstimatedRTT + 4*DevRTT

Capitolo 3: Livello di trasporto

- 3.1 Servizi a livello di trasporto
- 3.2 Multiplexing e demultiplexing
- 3.3 Trasporto senza connessione: UDP
- 3.4 Principi del trasferimento dati affidabile
- 3.5 Trasporto orientato alla connessione: TCP
  - struttura dei segmenti
  - trasferimento dati affidabile
  - controllo di flusso
  - gestione della connessione
- 3.6 Principi sul controllo di congestione
- 3.7 Controllo di congestione TCP

TCP: controllo di flusso

- Il lato ricevente della connessione TCP ha un buffer di ricezione:



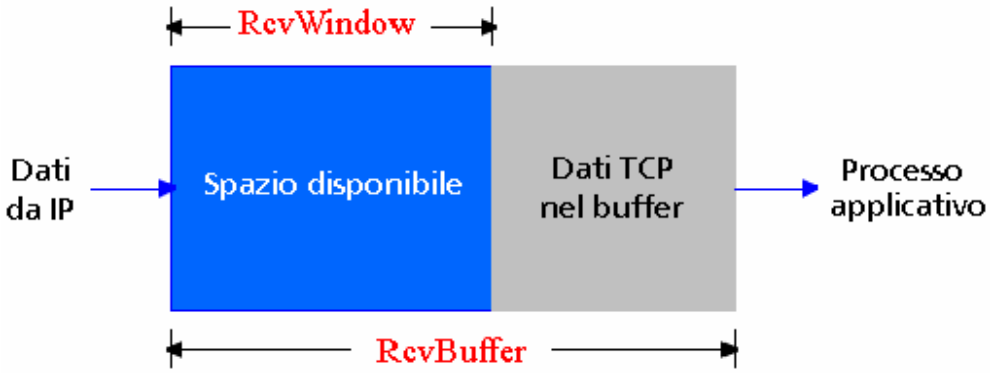
- Il processo applicativo potrebbe essere rallentato dalla lettura nel buffer

Controllo di flusso

Il mittente non vuole sovraccaricare il buffer del destinatario trasmettendo troppi dati, troppo velocemente

- Servizio di corrispondenza delle velocità: la frequenza d'invio deve corrispondere alla frequenza di lettura dell'applicazione ricevente

TCP: funzionamento del controllo di flusso



(supponiamo che il destinatario TCP scarti i segmenti fuori sequenza)

- Spazio disponibile nel buffer

= RcvWindow

= RcvBuffer - [LastByteRcvd - LastByteRead]

- Il mittente comunica lo spazio disponibile includendo il valore di **RcvWindow** nei segmenti
- Il mittente limita i dati non riscontrati a **RcvWindow**
  - garantisce che il buffer di ricezione non vada in overflow

Capitolo 3: Livello di trasporto

- 3.1 Servizi a livello di trasporto
- 3.2 Multiplexing e demultiplexing
- 3.3 Trasporto senza connessione: UDP
- 3.4 Principi del trasferimento dati affidabile
- 3.5 Trasporto orientato alla connessione: TCP
  - struttura dei segmenti
  - trasferimento dati affidabile
  - controllo di flusso
  - gestione della connessione
- 3.6 Principi sul controllo di congestione
- 3.7 Controllo di congestione TCP

Gestione della connessione TCP

Ricordiamo: mittente e destinatario TCP stabiliscono una "connessione" prima di scambiare i segmenti di dati

- inizializzano le variabili TCP:
  - numeri di sequenza
  - buffer, informazioni per il controllo di flusso (per esempio, **RcvWindow**)
- *client:* avvia la connessione

```
Socket clientSocket = new
Socket("hostname","portnumber");
```

- *server:* contattato dal client

```
Socket connectionSocket =
welcomeSocket.accept();
```

Handshake a tre vie:

Passo 1: il client invia un segmento SYN al server
- specifica il numero di sequenza iniziale
- nessun dato

Passo 2: il server riceve SYN e risponde con un segmento SYNACK
- il server alloca i buffer
- specifica il numero di sequenza iniziale del server

Passo 3: il client riceve SYNACK e risponde con un segmento ACK, che può contenere dati

Gestione della connessione TCP (continua)

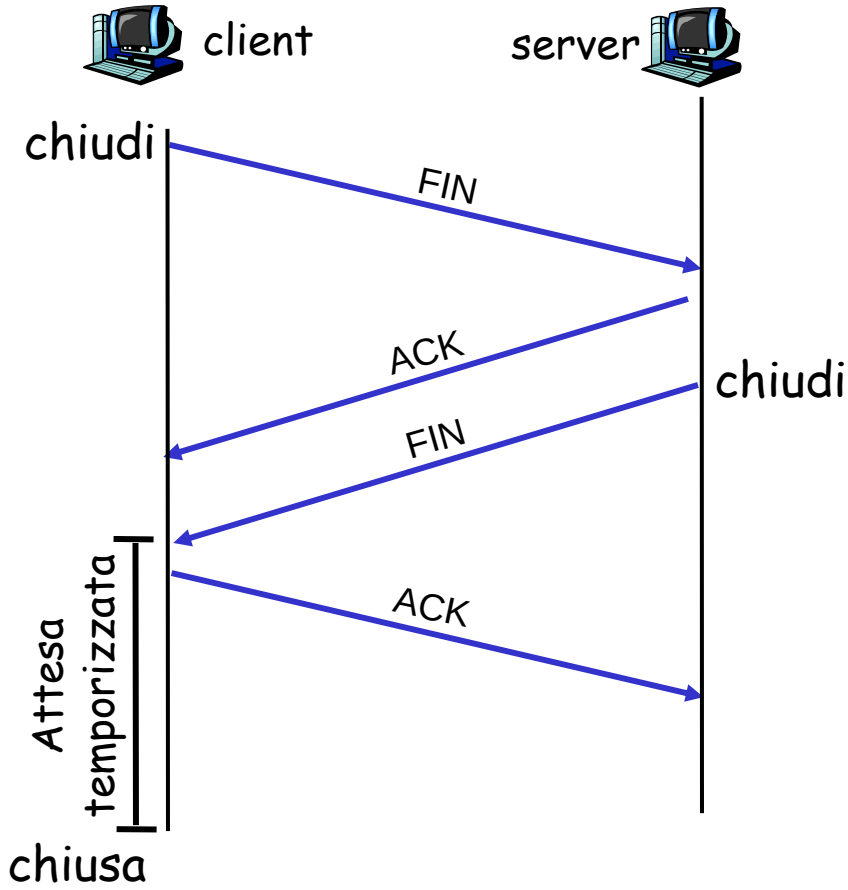
Chiudere una connessione:

Il client chiude la socket:

```
clientSocket.close();
```

Passo 1: il client invia un segmento di controllo FIN al server.

Passo 2: il server riceve il segmento FIN e risponde con un ACK. Chiude la connessione e invia un FIN.



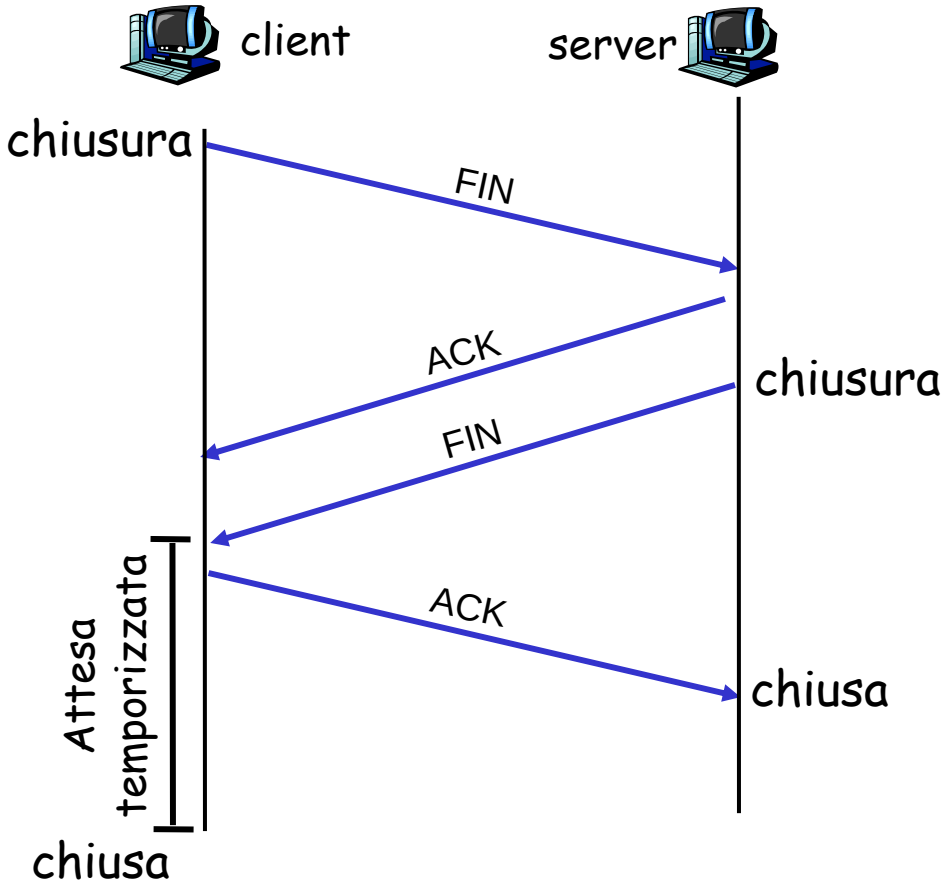
Gestione della connessione TCP (continua)

Passo 3: il client riceve FIN e risponde con un ACK.

- inizia l'attesa temporizzata - risponde con un ACK ai FIN che riceve

Passo 4: il server riceve un ACK. La connessione viene chiusa.

Nota: con una piccola modifica può gestire segmenti FIN simultanei.



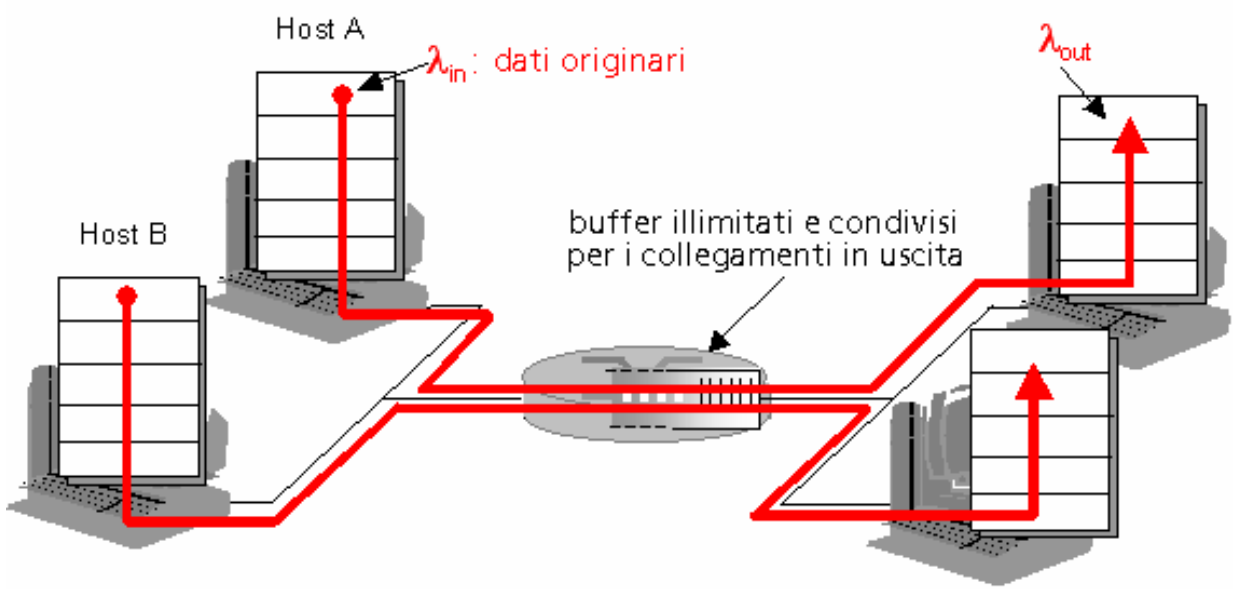
Principi sul controllo di congestione

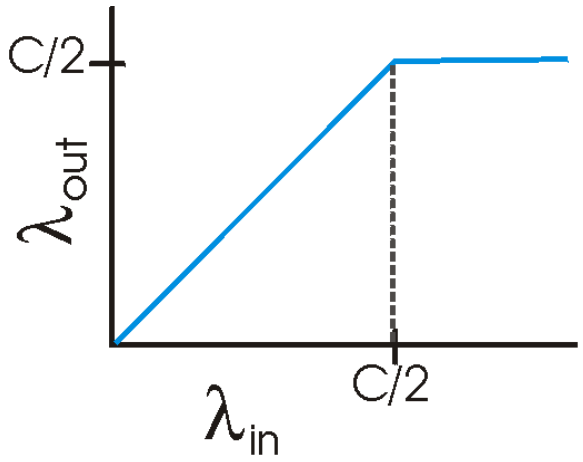
Congestione:

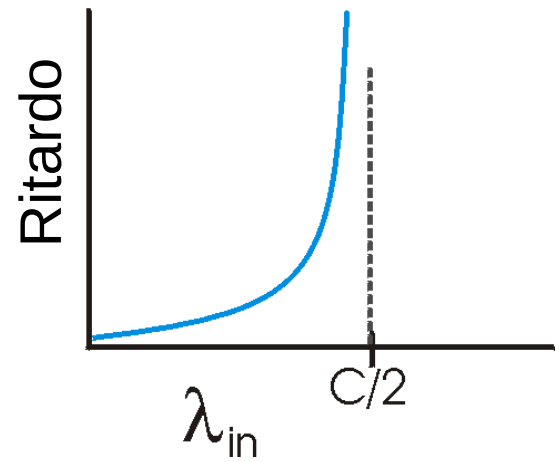
- informalmente: "troppe sorgenti trasmettono troppi dati, a una velocità talmente elevata che la *rete* non è in grado di gestirli"
- differisce dal controllo di flusso!
- manifestazioni:
  - pacchetti smarriti (overflow nei buffer dei router)
  - lunghi ritardi (accodamento nei buffer dei router)
- tra i dieci problemi più importanti del networking!

Cause/costi della congestione: scenario 1

- due mittenti, due destinatari
- un router con buffer illimitati
- nessuna ritrasmissione



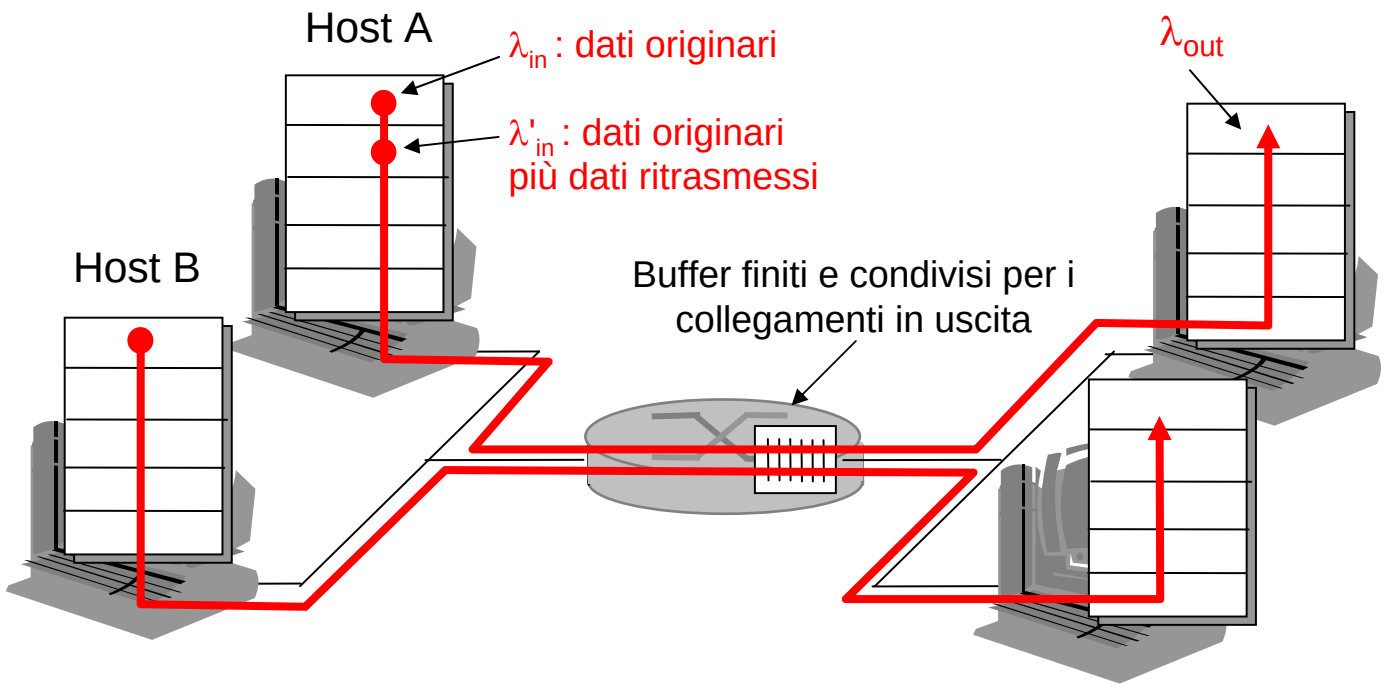




- grandi ritardi se congestionati
- throughput massimo

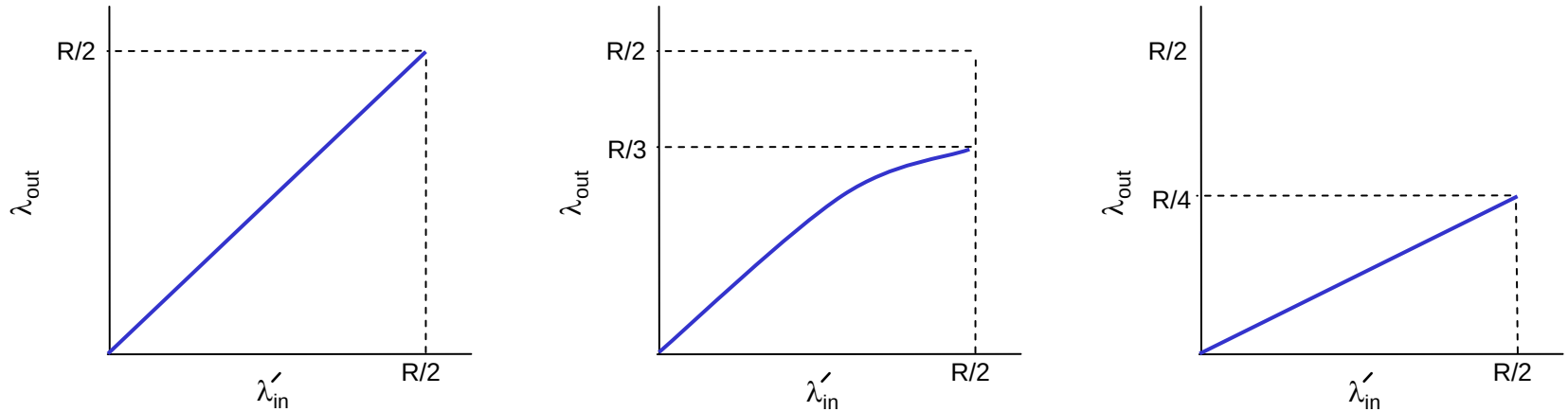
Cause/costi della congestione: scenario 2

- un router, buffer *finiti*
- il mittente ritrasmette il pacchetto perduto



Cause/costi della congestione: scenario 2

- Sempre: $\lambda_{in} = \lambda_{out}$ (goodput)
- Ritrasmissione "perfetta" solo quando la perdita: $\lambda'_{in} > \lambda_{out}$
- La ritrasmissione del pacchetto ritardato (non perduto) rende λ'_{in} più grande (rispetto al caso perfetto) per lo stesso λ_{out}



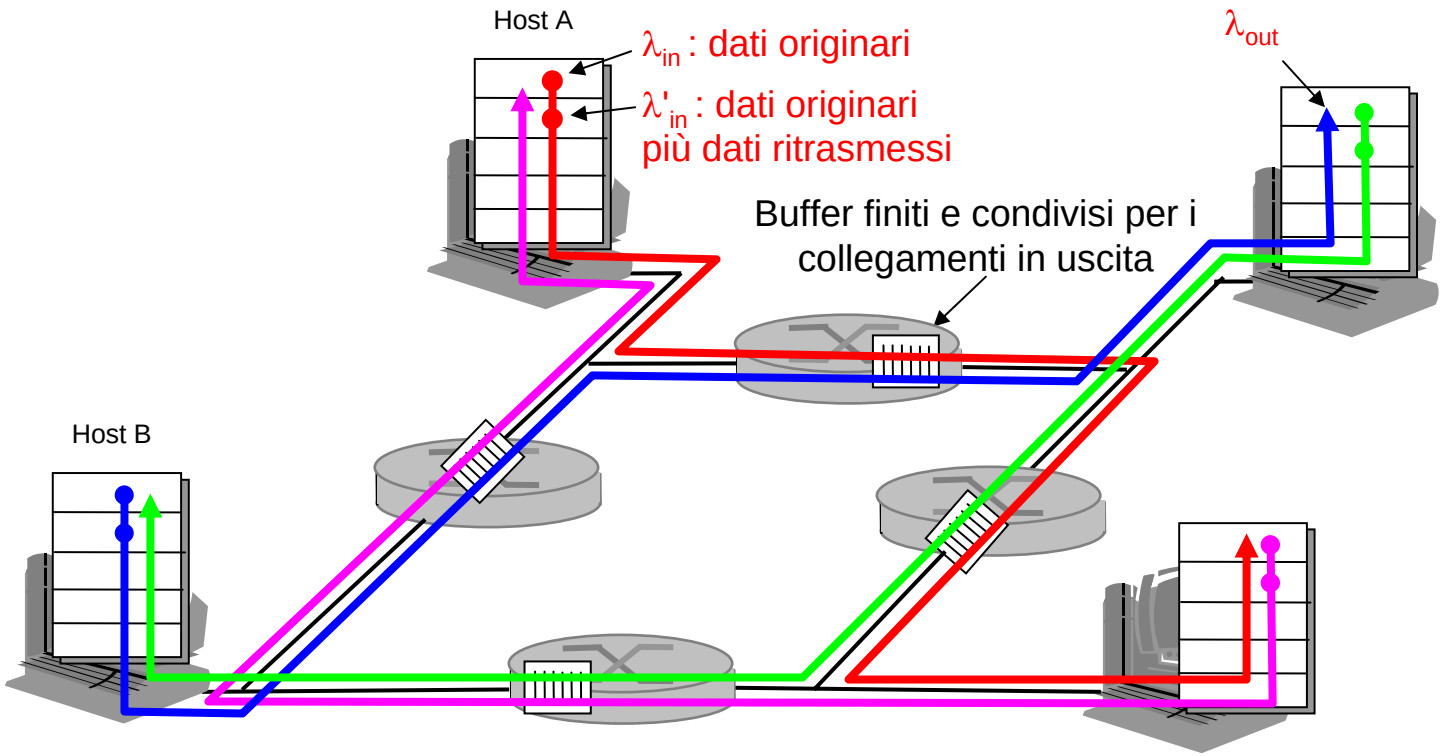
"Costi" della congestione:

- Più lavoro (ritrasmissioni) per un dato "goodput"
- Ritrasmissioni non necessarie: il collegamento trasporta più copie del pacchetto

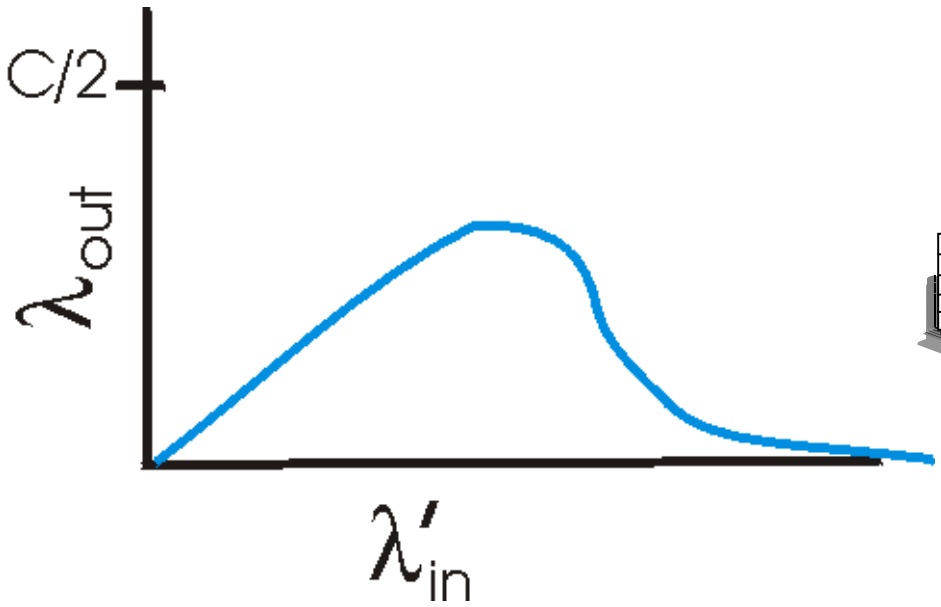
Cause/costi della congestione: scenario 3

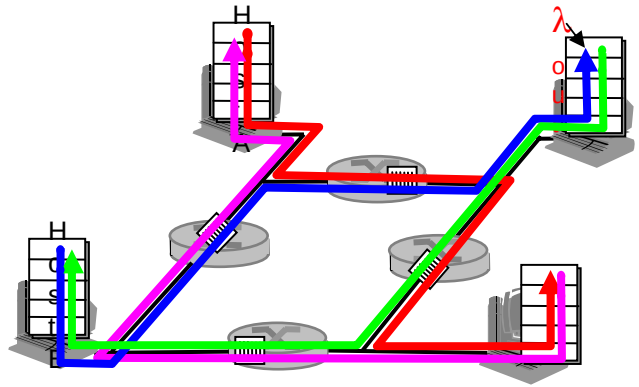
- Quattro mittenti
- Percorsi multihop
- timeout/ritrasmissione

D: Che cosa accade quando λ_{in} e λ'_{in} aumentano?



Cause/costi della congestione: scenario 3





Un altro "costo" della congestione:

- Quando il pacchetto viene scartato, la "capacità trasmissiva utilizzata sui collegamenti di upstream per instradare il pacchetto risulta sprecata!