

# Reti di Elaboratori

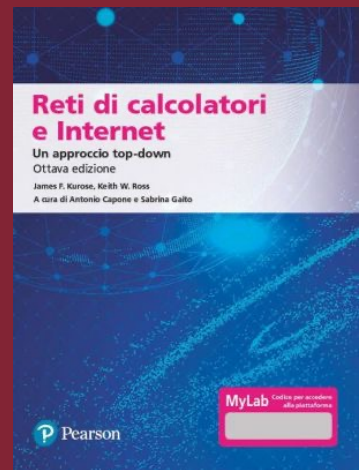
Livello di Trasporto: controllo della congestione



SAPIENZA  
UNIVERSITÀ DI ROMA

Alessandro Checco

[alessandro.checco@uniroma1.it](mailto:alessandro.checco@uniroma1.it)



Capitolo 3

# Chiusura di una connessione TCP

- client, server chiudono ciascuno il proprio lato di connessione
  - invia il segmento TCP con il bit FIN = 1
- rispondere al FIN ricevuto con ACK
  - alla ricezione del FIN, l'ACK può essere combinato con il proprio FIN
- possono essere gestiti scambi FIN simultanei

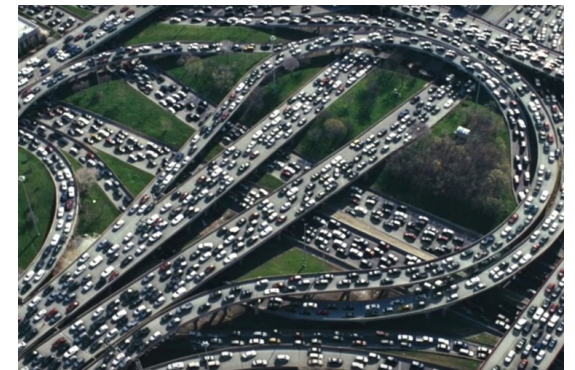
# Livello di trasporto: sommario

- Servizi a livello di trasporto
- Multiplexing e demultiplexing
- Trasporto senza connessione: UDP
- Principi di trasferimento affidabile dei dati
- Trasporto orientato alla connessione: TCP
  - struttura del segmento
  - trasferimento affidabile dei dati
  - controllo del flusso
  - gestione della connessione
- Principi di controllo della congestione
- Controllo della congestione TCP
- Evoluzione della funzionalità del livello di trasporto

# Principi di controllo della congestione

## Congestione:

- informalmente: "troppe fonti che inviano troppi dati troppo velocemente per essere gestiti dalla *rete*"
- manifestazioni:
  - lunghi ritardi (accodamento nei buffer del router)
  - pacchetti (buffer overflow sui router)
- diverso dal controllo di flusso!
- un problema nei top 10!



**controllo della congestione:** troppi mittenti, invio troppo veloce

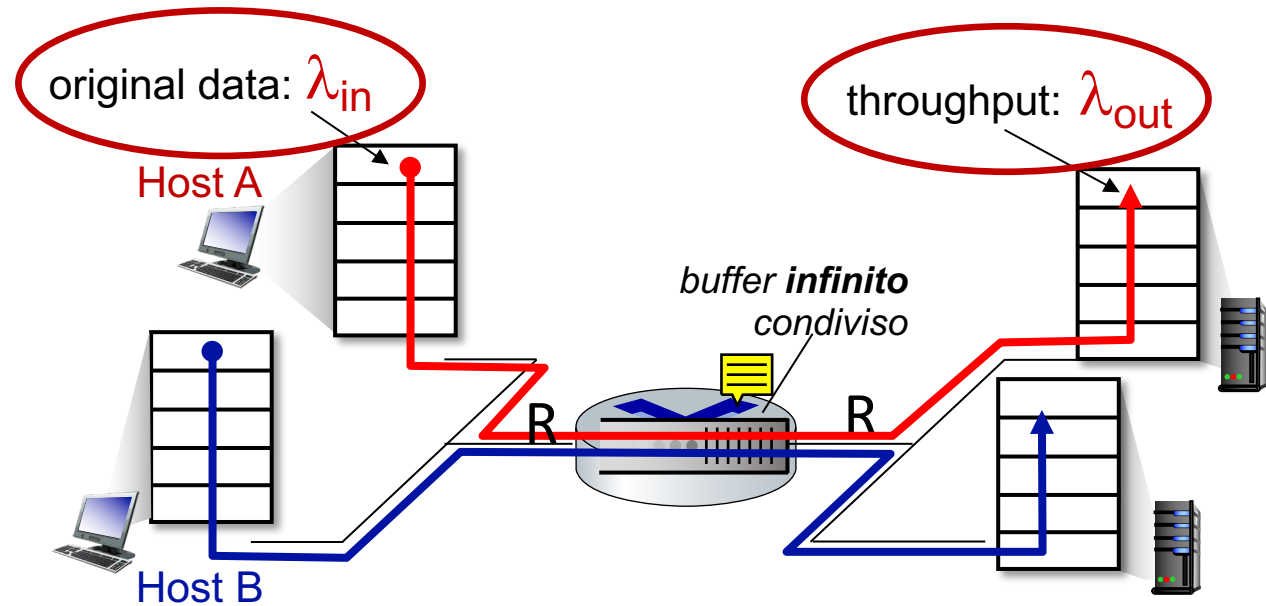


**controllo del flusso:** un mittente troppo veloce per un destinatario

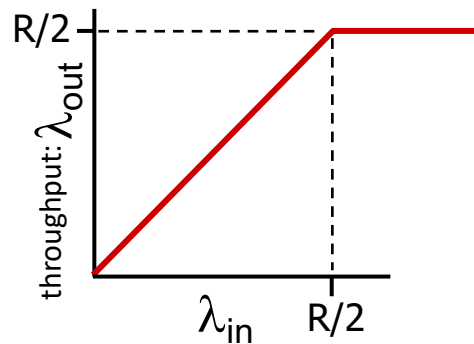
# Cause/costi della congestione: scenario 1

Scenario più semplice:

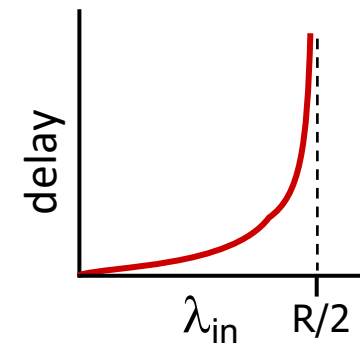
- un router, buffer infiniti
- rate dei link  $R$
- due flussi
- non sono necessarie ritrasmissioni



**D:** Cosa succede quando il tasso di arrivo è  $\lambda_{in}$  si avvicina a  $R/2$ ?



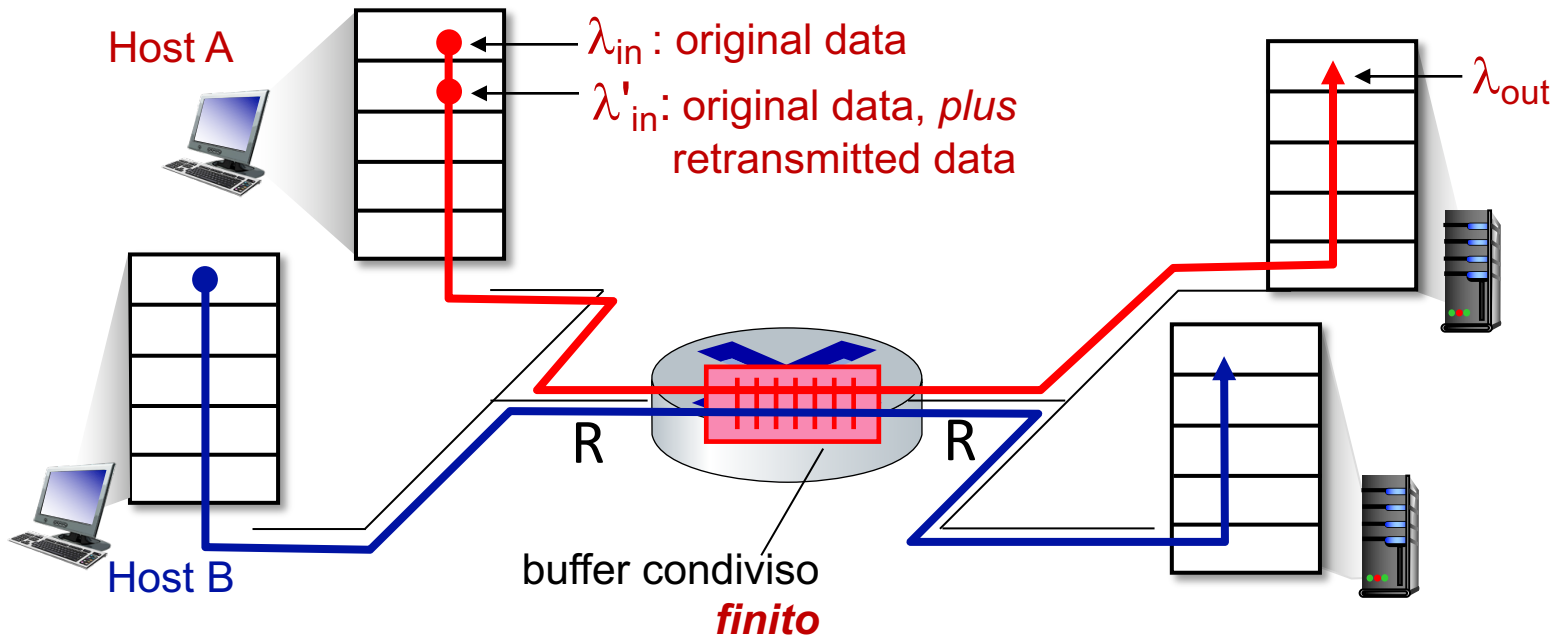
throughput massimo per connessione:  $R/2$



ritardi alti quando  $\lambda_{in}$  si avvicina a  $R/2$

# Cause/costi della congestione: scenario 2

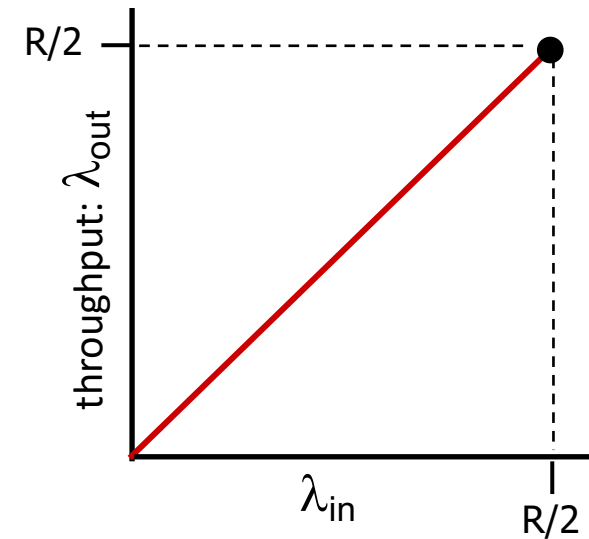
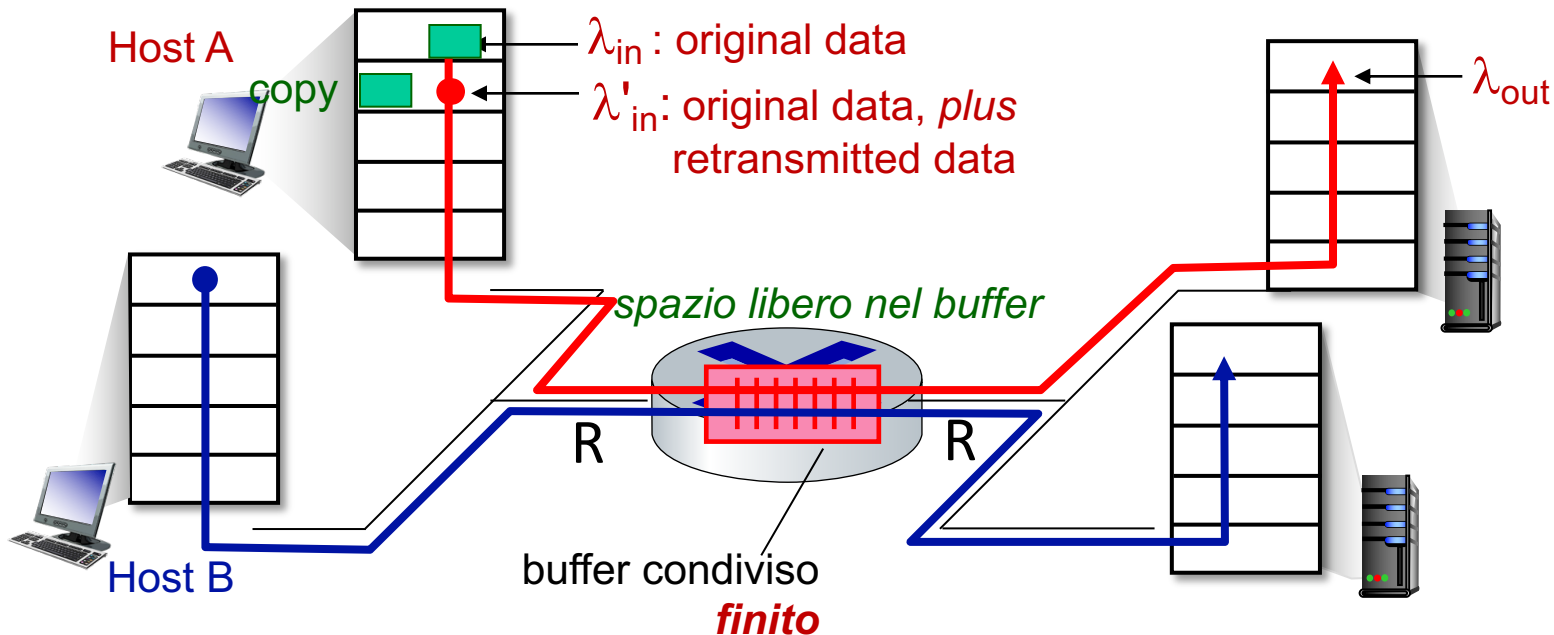
- un router, buffer *finiti*
- mittente ritrasmette pacchetti persi (timeout)
  - application-layer input = application-layer output:  $\lambda_{in} = \lambda_{out}$
  - transport-layer input include *ritrasmissioni*:  $\lambda'_{in} \geq \lambda_{in}$



# Cause/costi della congestione: scenario 2

Assunzione idealizzata: **perfect knowledge**

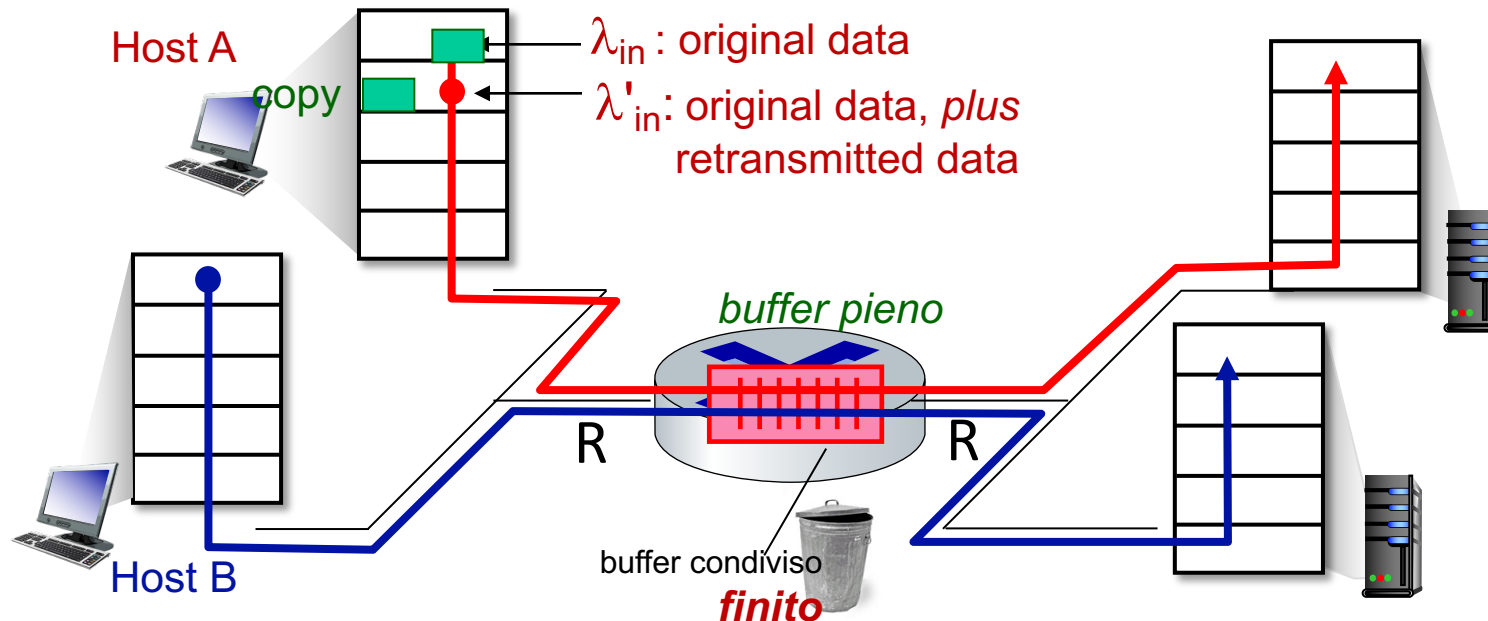
- il mittente invia solo se sa che i buffer nei router sono liberi
- tutto a posto, ma ritardi comunque crescono senza limite all'approcciare di  $R/2$



# Cause/costi della congestione: scenario 2

Assunzione: *parte* di perfect knowledge

- i pacchetti possono essere scartati per buffer del router pieno
- il mittente ha conoscenza perfetta di quali pacchetti sono persi: ritrasmette solo i pacchetti che sa essere persi

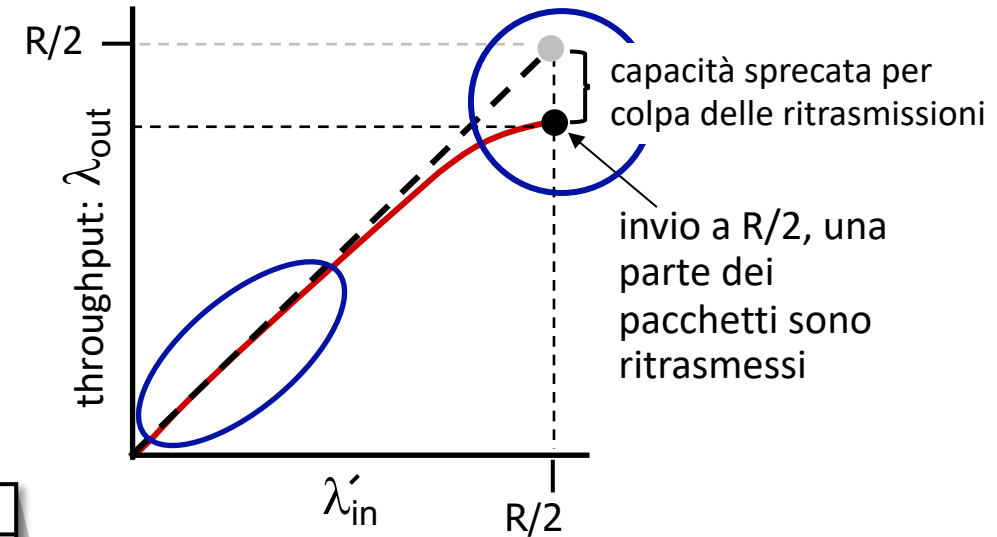
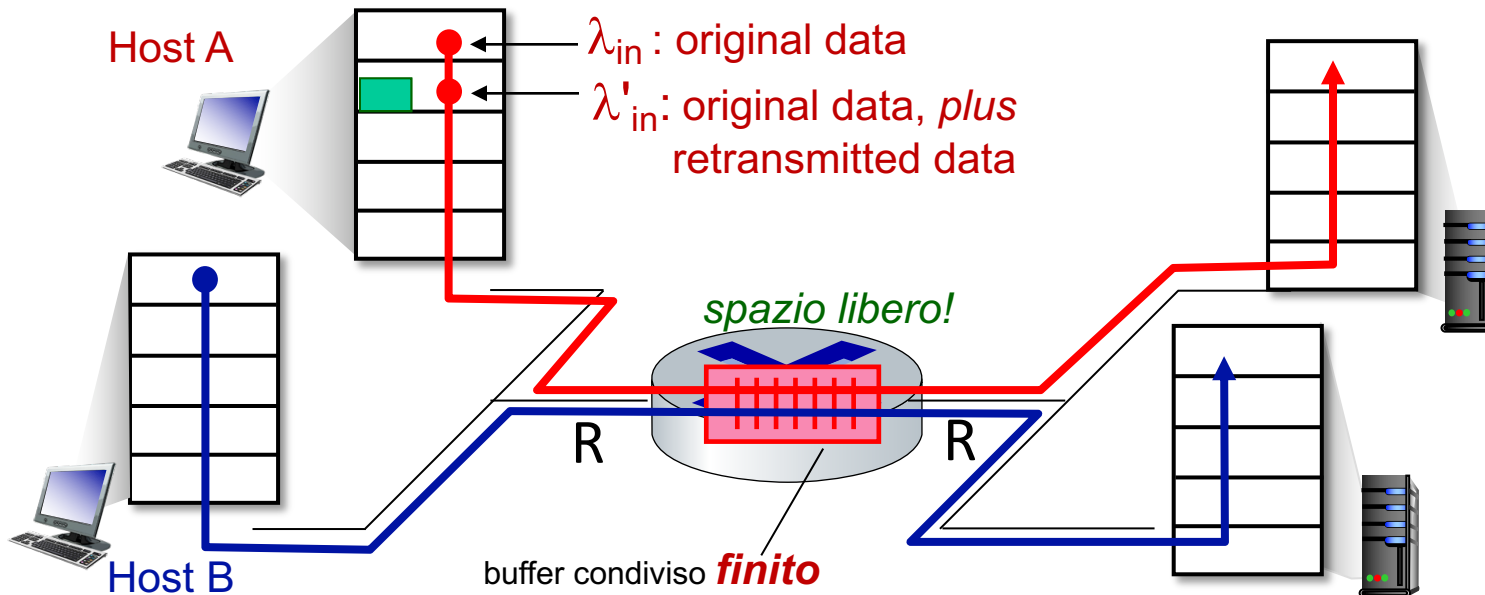




# Cause/costi della congestione: scenario 2

Assunzione: *parte* di perfect knowledge

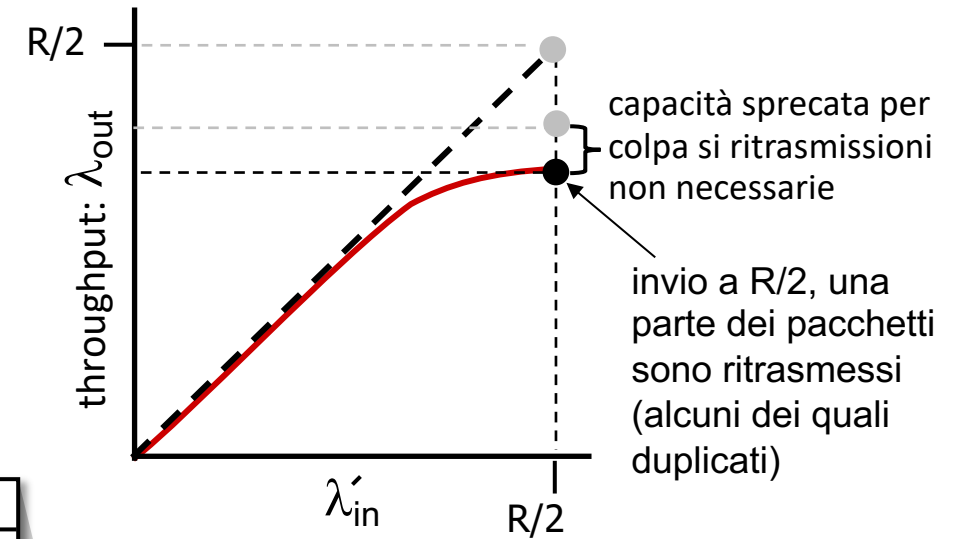
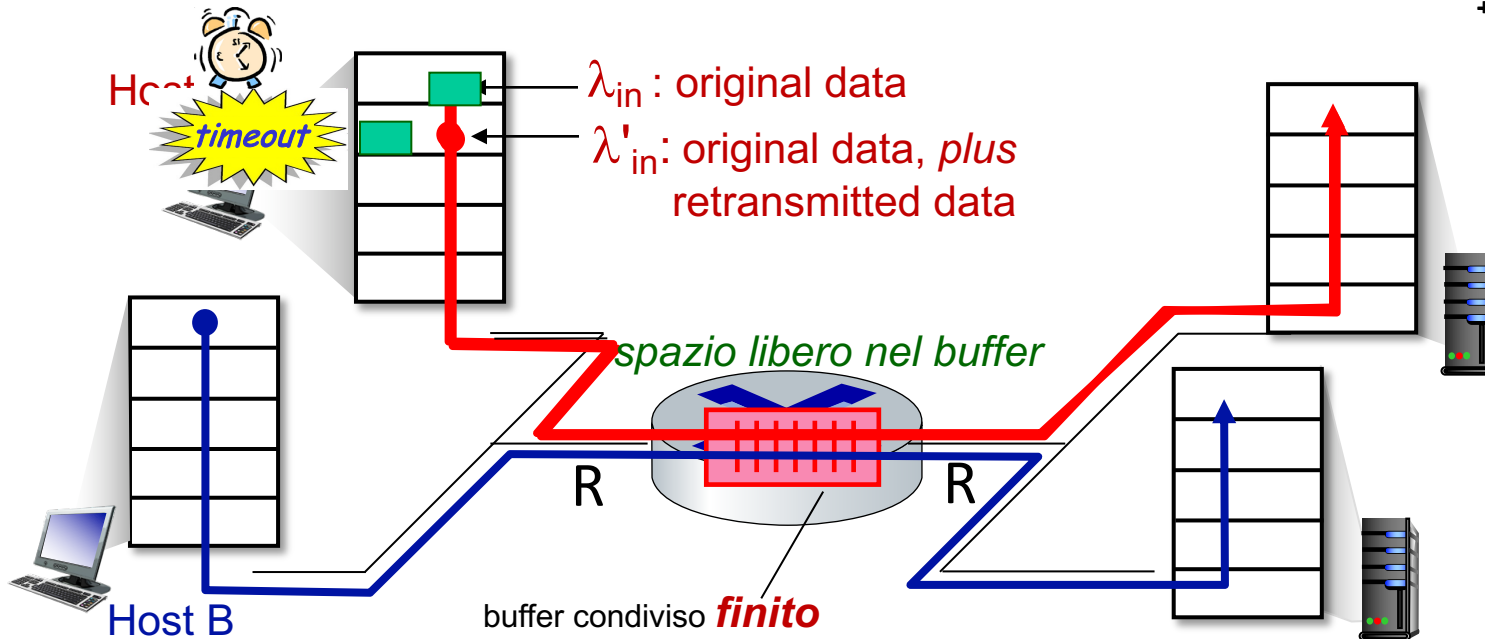
- i pacchetti possono essere scartati per buffer del router pieno
- il mittente ha conoscenza perfetta di quali pacchetti sono persi: ritrasmette solo i pacchetti che sa essere persi



# Cause/costi della congestione: scenario 2

## Scenario realistico: *duplicati non necessari*

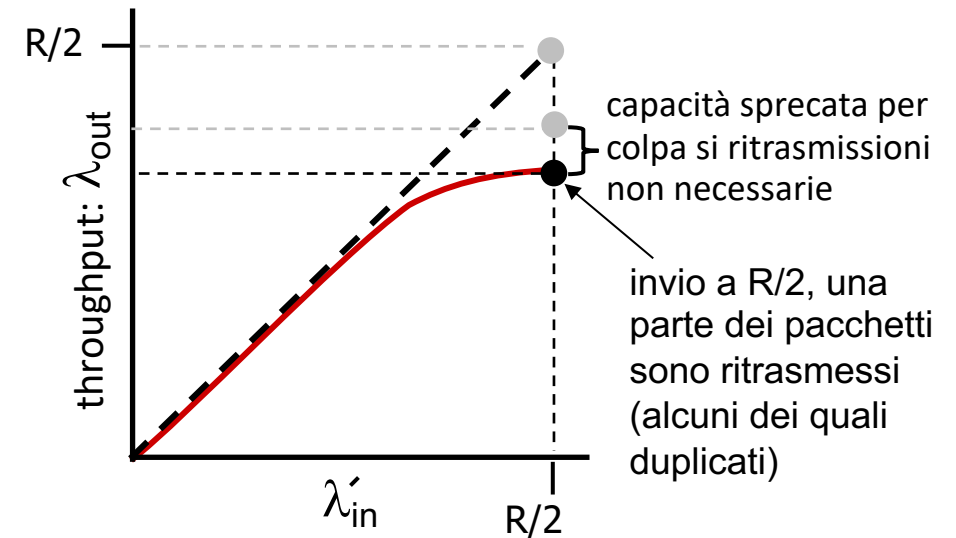
- i pacchetti possono essere persi, scartati dal router a causa di buffer pieni, richiedendo ritrasmissioni
- ma i tempi del mittente possono scadere prematuramente, inviandone *due* copie, *entrambe* consegnate



# Cause/costi della congestione: scenario 2

## Scenario realistico: *duplicati non necessari*

- i pacchetti possono essere persi, scartati dal router a causa di buffer pieni, richiedendo ritrasmissioni
- ma i tempi del mittente possono scadere prematuramente, inviandone *due* copie, *entrambe* consegnate



## “costi” della congestione:

- serve più lavoro (ritrasmissioni) per un dato throughput durante congestione
- ritrasmissioni non necessarie: il collegamento trasporta più copie di un pacchetto
  - diminuzione del throughput massimo ottenibile

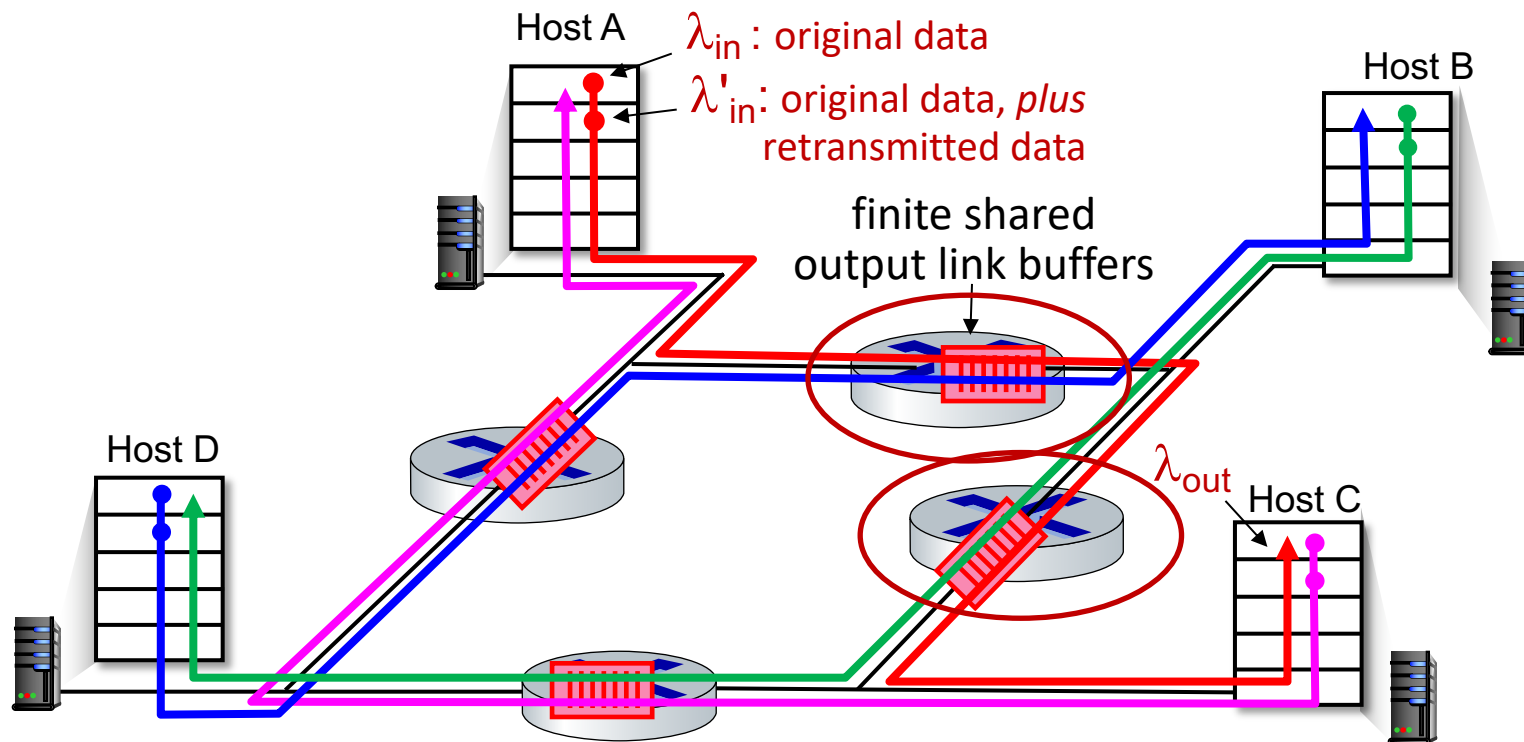
# Cause/costi della congestione: scenario 3



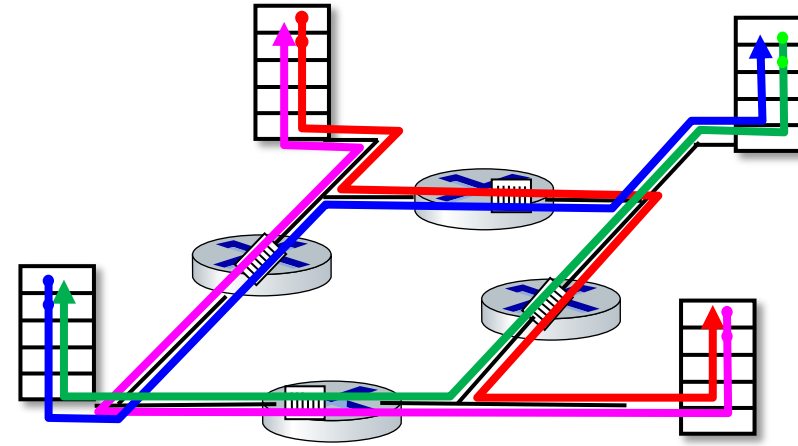
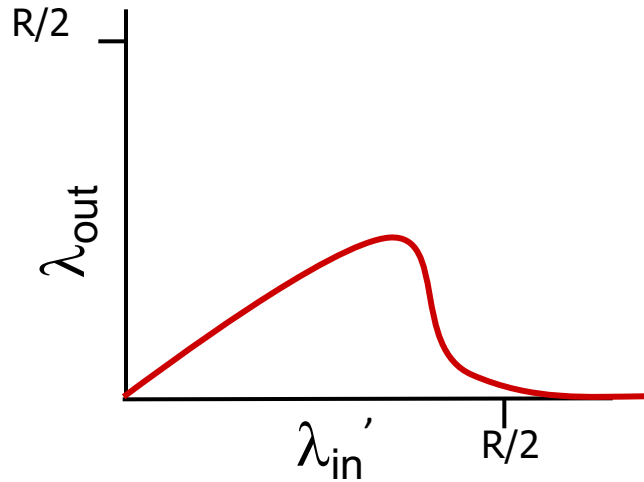
- *quattro* mittenti
- percorsi *multi-hop*
- timeout/ritrasmissione

Q: cosa succede quando  $\lambda_{in}$  e  $\lambda'_{in}$  crescono?

A: quando  $\lambda'_{in}$  cresce, i pacchetti hanno bassa probabilità di non essere scartati a ogni hop  $\rightarrow 0$



# Cause/costi della congestione: scenario 3

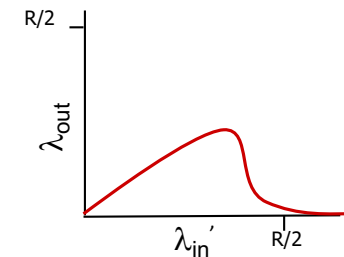
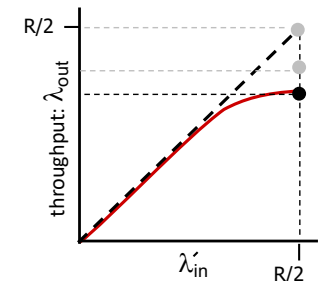
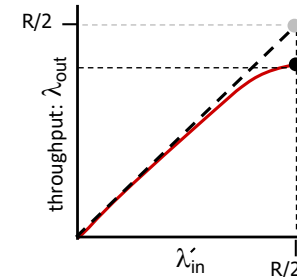
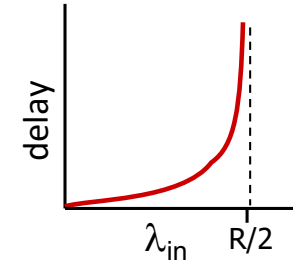
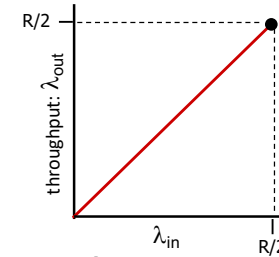


## un altro “costo” della congestione:

- quando il pacchetto è scartato, tutta la capacità di trasmissione upstream e il buffering utilizzati per quel pacchetto sono stati sprecati!

# Cause/costi della congestione: approfondimenti

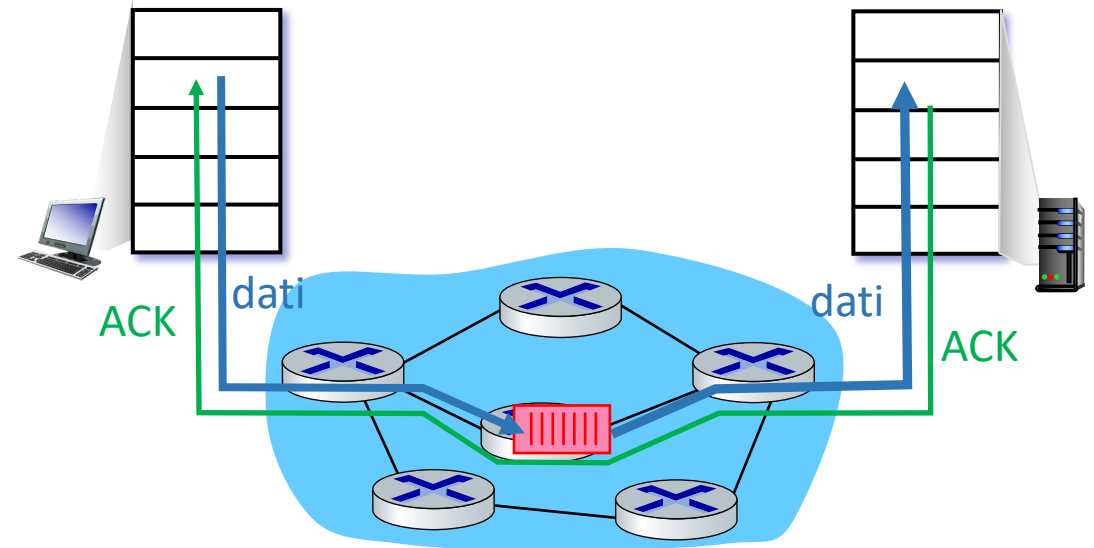
- il throughput non può mai superare la capacità
- il ritardo aumenta con l'avvicinarsi alla capacità
- la perdita/ritrasmissione riduce il throughput effettivo
- i duplicati non necessari riducono ulteriormente il throughput effettivo
- capacità di trasmissione/buffering upstream sprecata per i pacchetti persi downstream (reti multi-hop)



# Approcci al controllo della congestione

controllo della congestione end-to-end (punto punto):

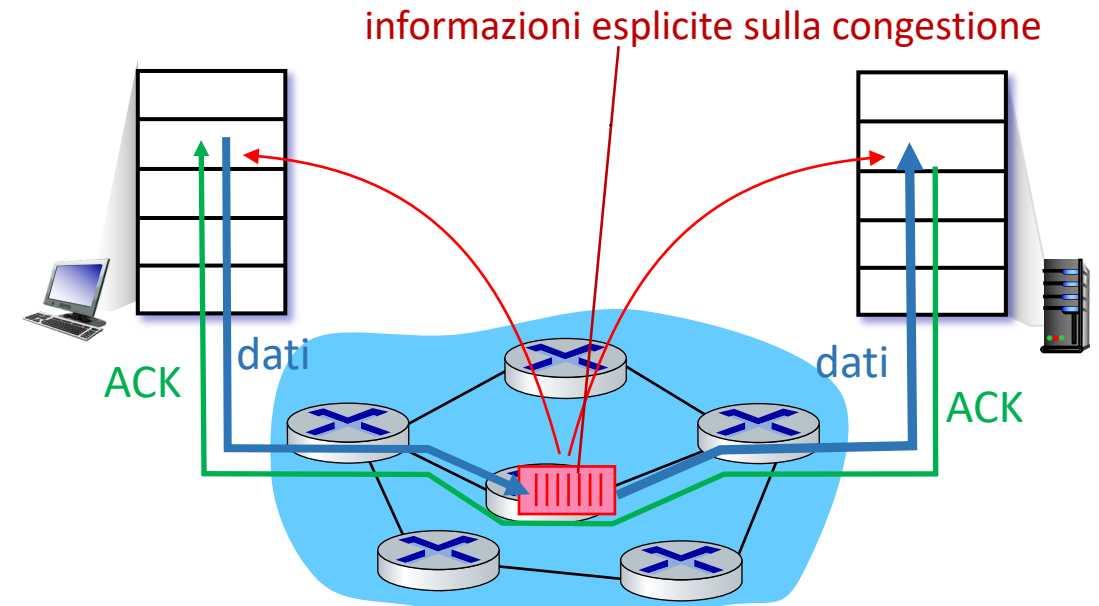
- nessun feedback esplicito dalla rete
- congestione *dedotta* dalle perdite (e ritardi) osservati
- approccio adottato dal TCP



# Approcci al controllo della congestione

## Controllo della congestione assistito dalla rete:

- i router forniscono un feedback *diretto* agli host di invio/ricezione con flussi che passano attraverso router congestionati
- può indicare il livello di congestione o la velocità di invio impostata in modo esplicito
- Protocolli TCP ECN, ATM, DECbit





# Livello di trasporto: sommario

- Servizi a livello di trasporto
- Multiplexing e demultiplexing
- Trasporto senza connessione: UDP
- Principi di trasferimento affidabile dei dati
- Trasporto orientato alla connessione: TCP
  - struttura del segmento
  - trasferimento affidabile dei dati
  - controllo del flusso
  - gestione della connessione
- Principi di controllo della congestione
- **Controllo della congestione TCP**
- Evoluzione della funzionalità del livello di trasporto

# Controllo della congestione TCP: AIMD

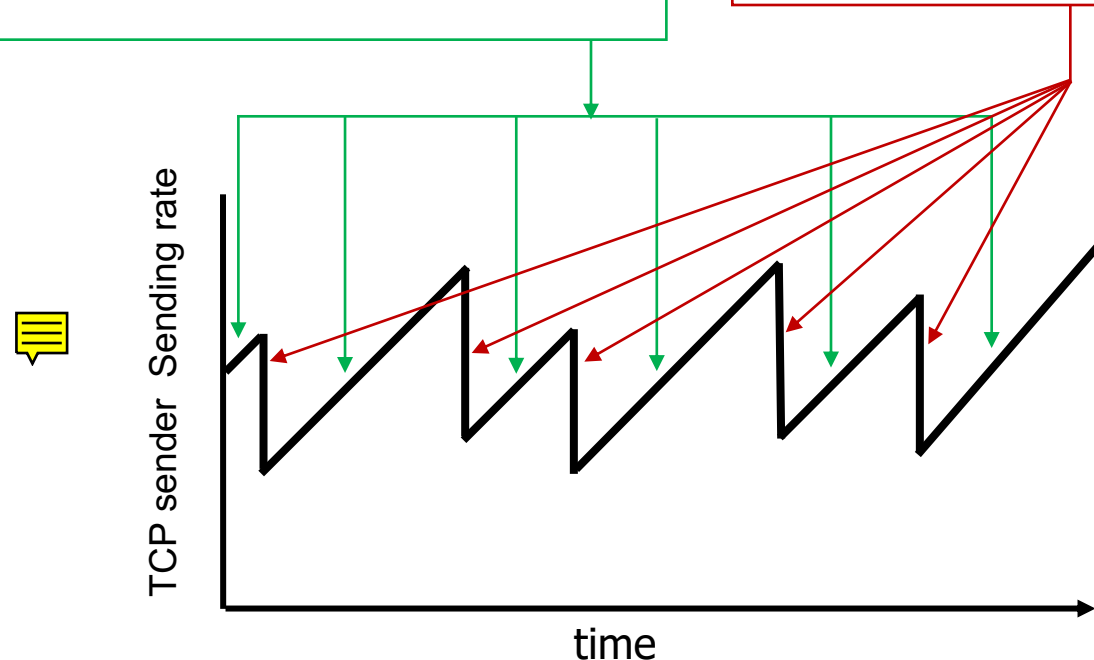
- *approccio*: i mittenti possono aumentare la velocità di invio fino a quando si verifica la perdita di pacchetti (congestione), quindi diminuire la velocità di invio in caso di pacchetti persi

## Additive Increase

aumenta il rate di 1 MSS ogni RTT fino a quando una perdita viene osservata

## Multiplicative Decrease

a ogni evento di perdita dividi per due il rate di invio



**AIMD:** comportamento  
a dente di sega:  
*sondando*  
la larghezza di banda

# TCP AIMD

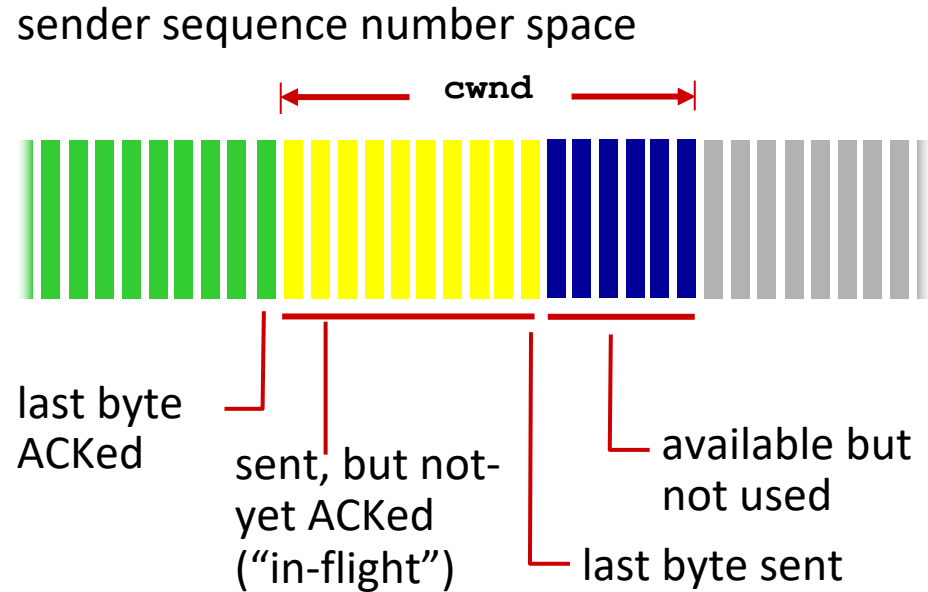
*Multiplicative decrease*: il rate di invio (prop. a numero di MSS in volo) è

- Dimezzato in caso di perdita rilevata dal triplo ACK duplicato (TCP Reno fast recovery)
- Diminuita a 1 MSS (dimensione massima del segmento) quando la perdita viene rilevata dal timeout (TCP Tahoe slow start)

## Perché AIMD?

- AIMD, un algoritmo asincrono distribuito, ha dimostrato di:
  - ottimizzare le portate congestionate su tutta la rete!
  - hanno proprietà di stabilità desiderabili

# Controllo della congestione TCP: dettagli



Comportamento di invio TCP:

- *approssimativamente*: invia `cwnd` byte, attendi RTT per ACKS, quindi invia più byte

$$\text{TCP rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

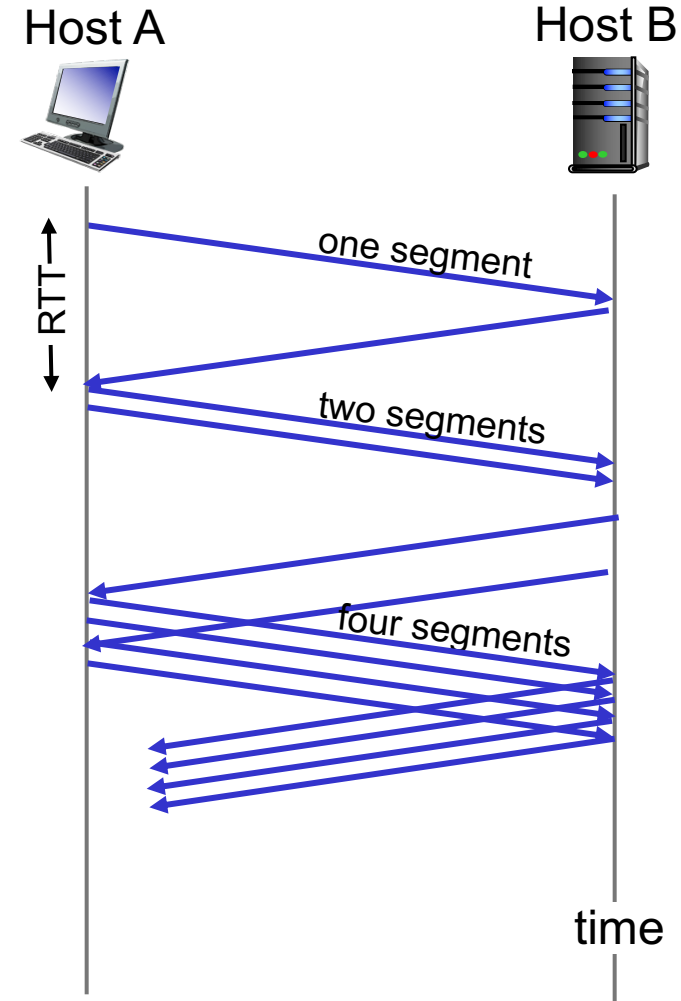
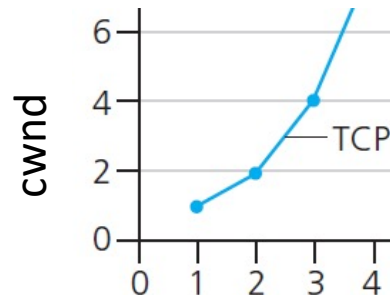


- Mittente limita la trasmissione:  $\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$
- `cwnd` è cambiato dinamicamente reagendo alla congestione osservata

Additive increase: aumenta il rate di 1 MSS ogni RTT significa approx  $\text{cwnd} = \text{cwnd} + (1/\text{cwnd})$  a ogni ACK, quindi dopo una intera `cwnd` ottendo  $\text{cwnd} = \text{cwnd} + 1$

# TCP slow start

- quando inizia la connessione, aumenta la velocità in modo esponenziale (+1 MSS per ogni ACK) fino alla prima perdita:
  - inizialmente **cwnd** = 1 MSS
  - raddoppia **cwnd** ogni RTT
- *slow?* Il rate iniziale è basso, ma aumenta esponenzialmente



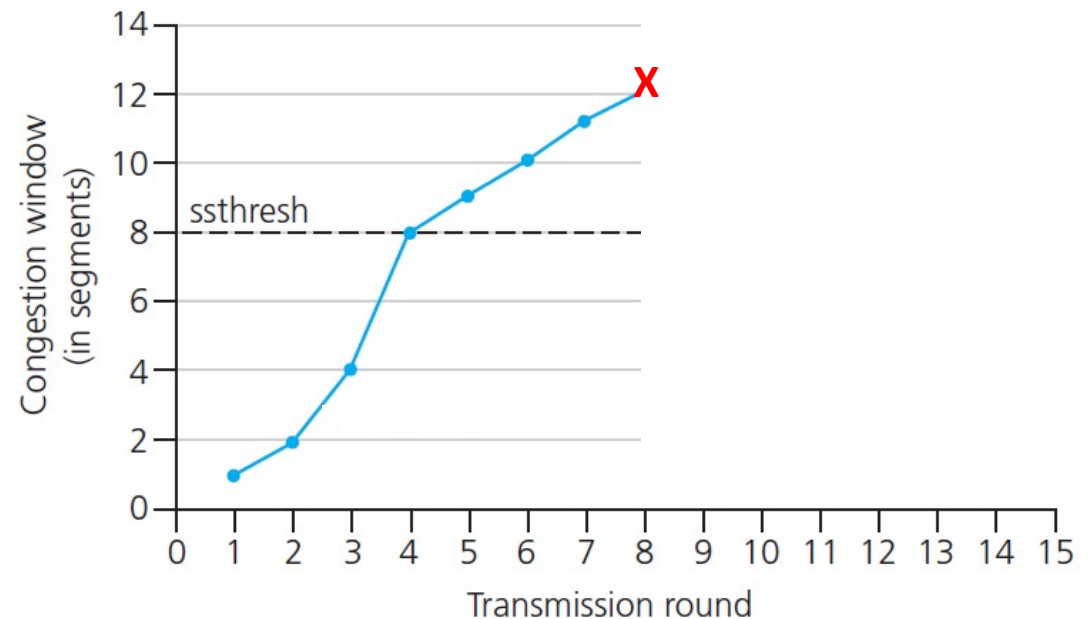
# TCP: transizione slow start/congestion avoidance

**D:** quando passare da crescita esponenziale a lineare?

**A:** quando **cwnd** arriva a 1/2 del suo valore prima dell'ultima perdita di pacchetto

## Implementazione:

- variabile **ssthresh**
- in caso di perdita, **ssthresh** è impostato su 1/2 di **cwnd** appena prima dell'evento di perdita



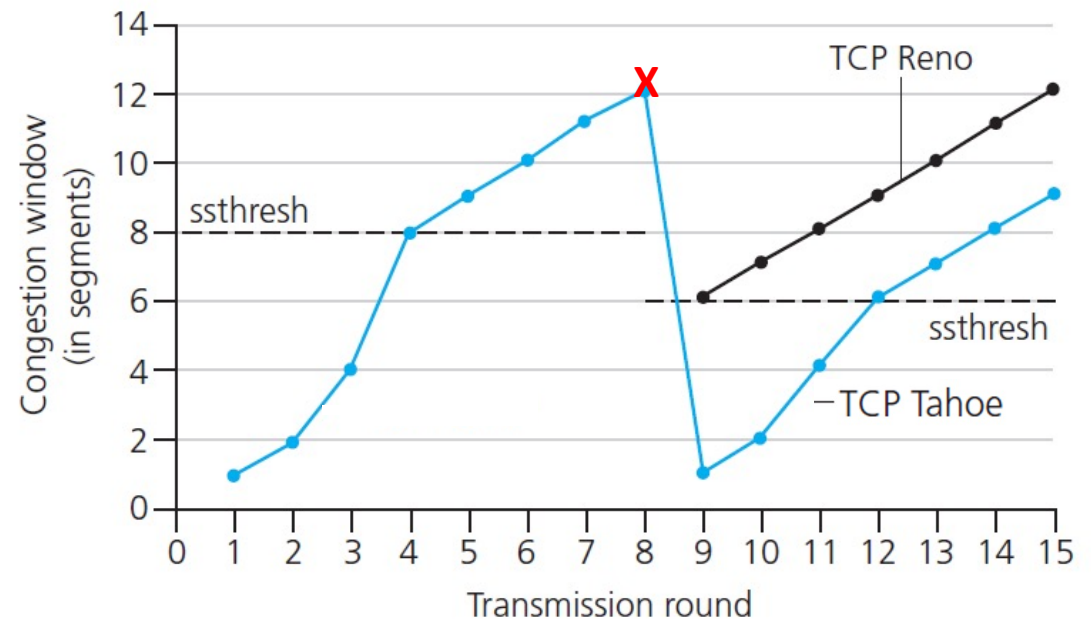
# TCP: transizione slow start/congestion avoidance

**D:** quando passare da crescita esponenziale a lineare?

**A:** quando **cwnd** arriva a 1/2 del suo valore prima dell'ultima perdita di pacchetto

## Implementazione:

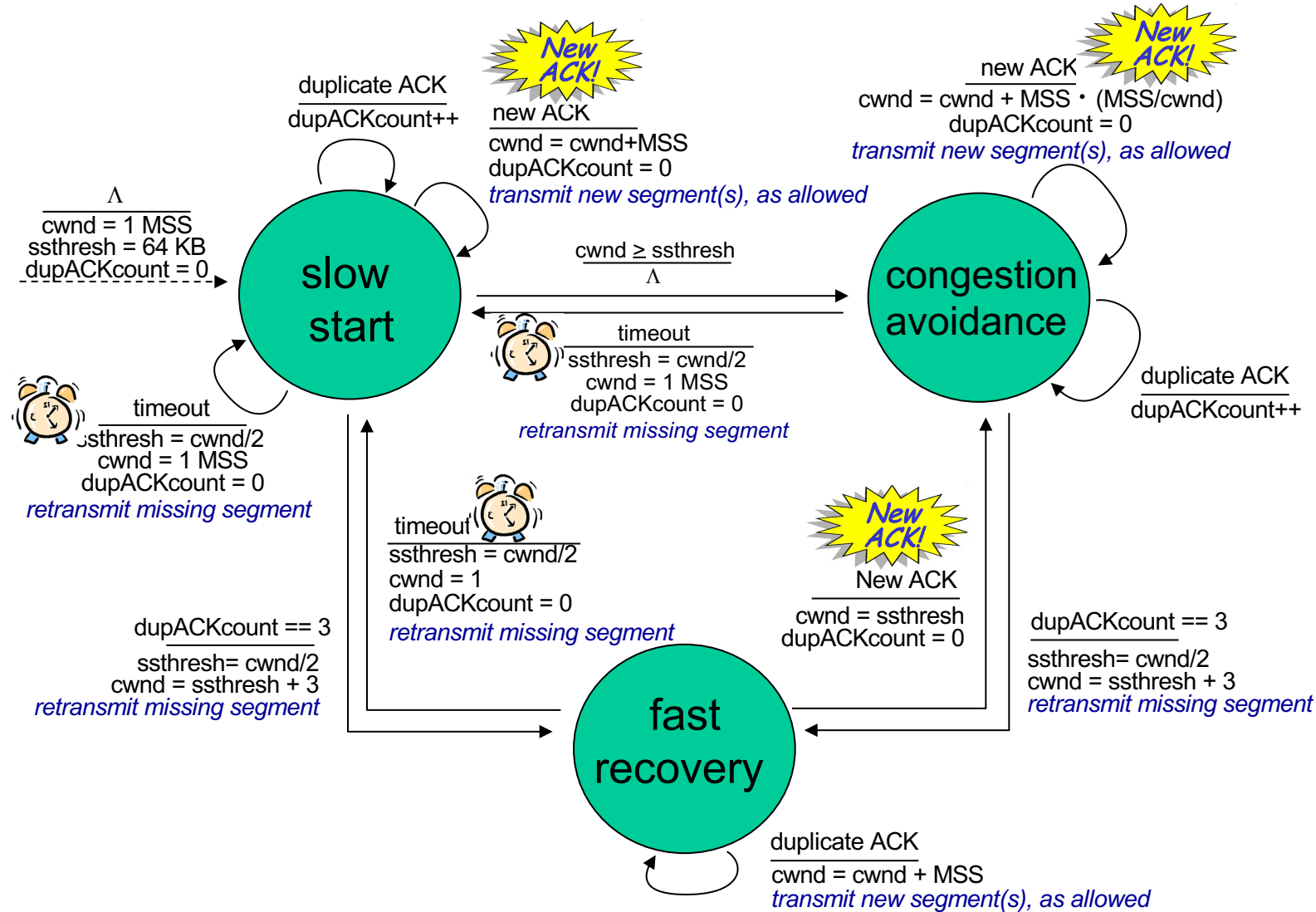
- variabile **ssthresh**
- in caso di perdita, **ssthresh** è impostato su 1/2 di **cwnd** appena prima dell'evento di perdita
- TCP Reno passa direttamente alla fase di congestion avoidance quando la perdita è causata da 3 dup ACK (dopo una breve fase di *fast recovery non mostrata nel grafico*)



# Sommario: controllo della congestione TCP: Reno

Slow start +  
(AIMD) + (Fast  
Retransmit) +  
**Congestion  
Avoidance** =  
TCP Tahoe

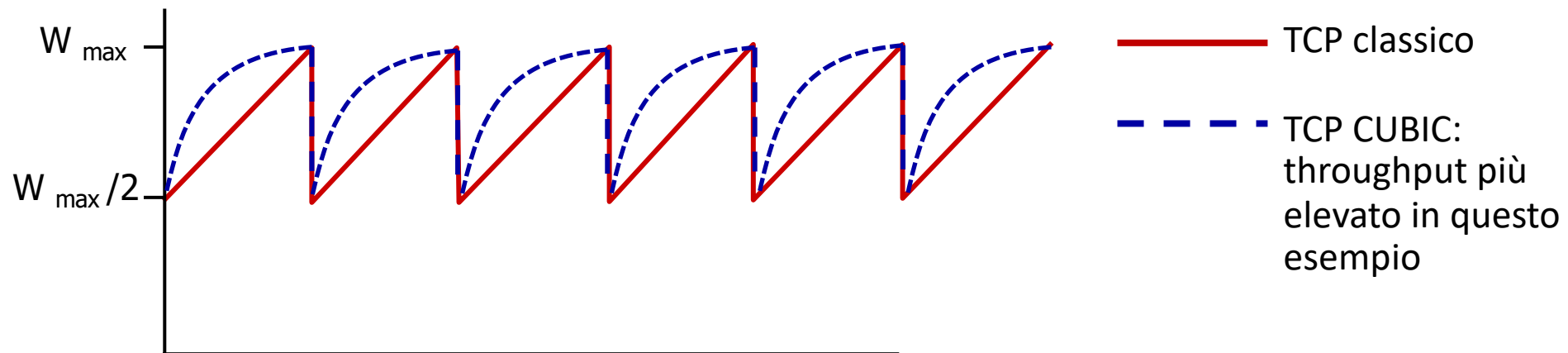
Tahoe + Fast  
Recovery =  
TCP Reno





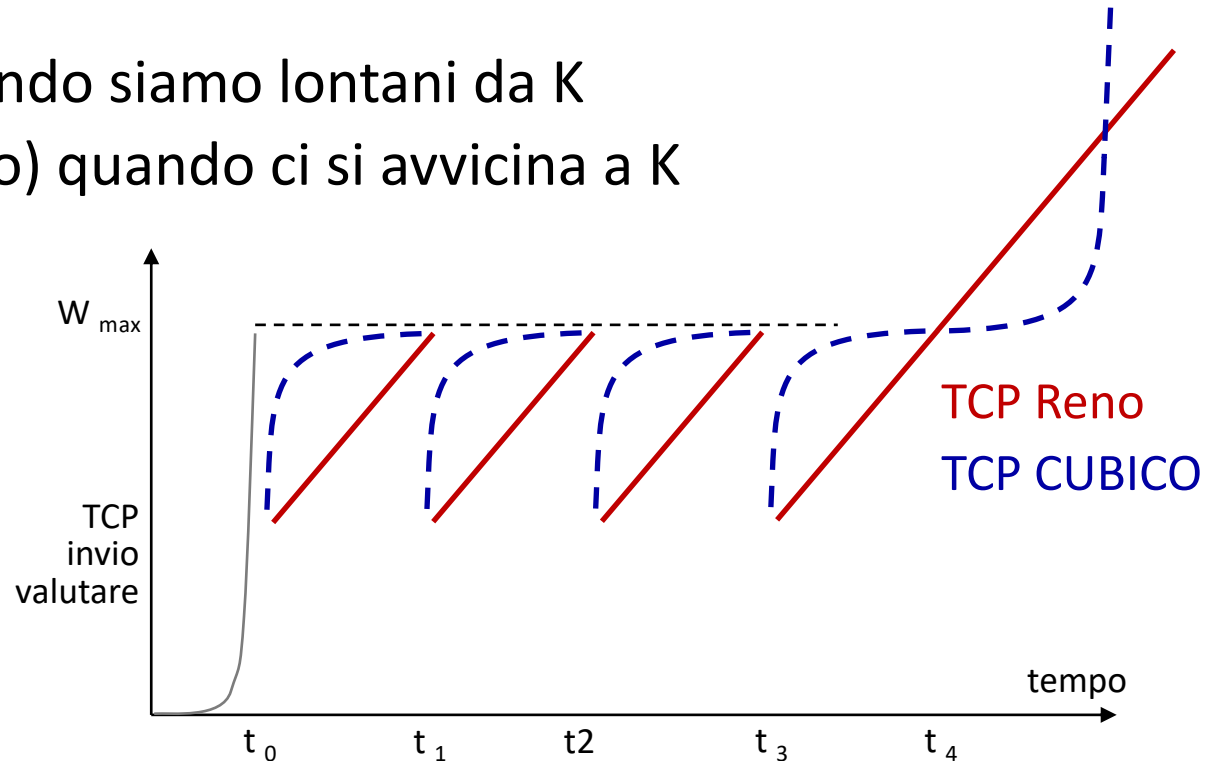
# TCP CUBIC

- Esiste un modo migliore di AIMD per "sondare" la larghezza di banda utilizzabile?
- Intuizione:
  - $W_{\max}$ : velocità di invio quando è stata rilevata la perdita di congestione
  - lo stato di congestione del probabilmente (?) nel frattempo non è cambiato molto
  - dopo aver dimezzato la frequenza/finestra in caso di perdita, vogliamo salire fino a  $W_{\max}$  *velocemente*, ma poi avvicinarci a  $W_{\max}$  *lentamente* (rispetto a TCP classico)



# TCP CUBIC

- K: momento stimato in cui la dimensione della finestra TCP raggiungerà  $W_{\max}$ 
  - La stima è definita in RFC 8312
- aumentare  $W$  in funzione del *cu*bo della distanza tra il tempo corrente e K
  - aumenta velocemente quando siamo lontani da K
  - aumenta lentamente (cauto) quando ci si avvicina a K
- TCP CUBIC  
predefinito in Linux, il TCP più popolare per i server Web più diffusi

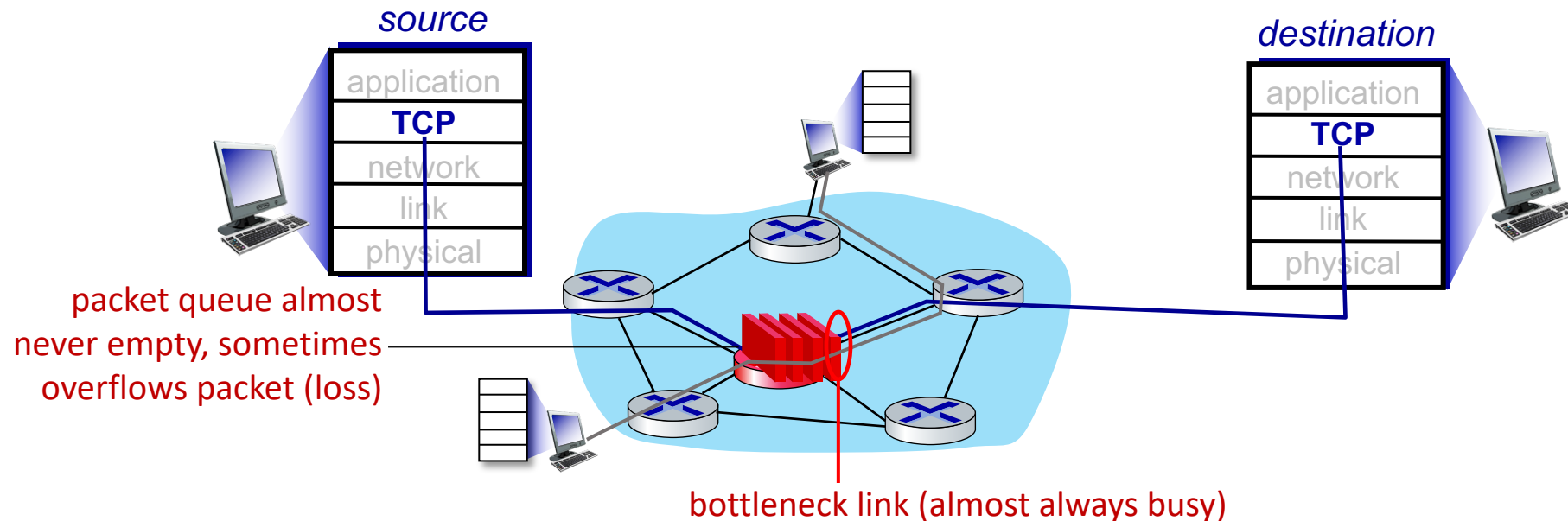


# Livello di trasporto: sommario

- Servizi a livello di trasporto
- Multiplexing e demultiplexing
- Trasporto senza connessione: UDP
- Principi di trasferimento affidabile dei dati
- Trasporto orientato alla connessione: TCP
  - struttura del segmento
  - trasferimento affidabile dei dati
  - controllo del flusso
  - gestione della connessione
- Principi di controllo della congestione
- **Controllo della congestione TCP**
  - **Approcci delay-based e network-assisted**
- Evoluzione della funzionalità del livello di trasporto

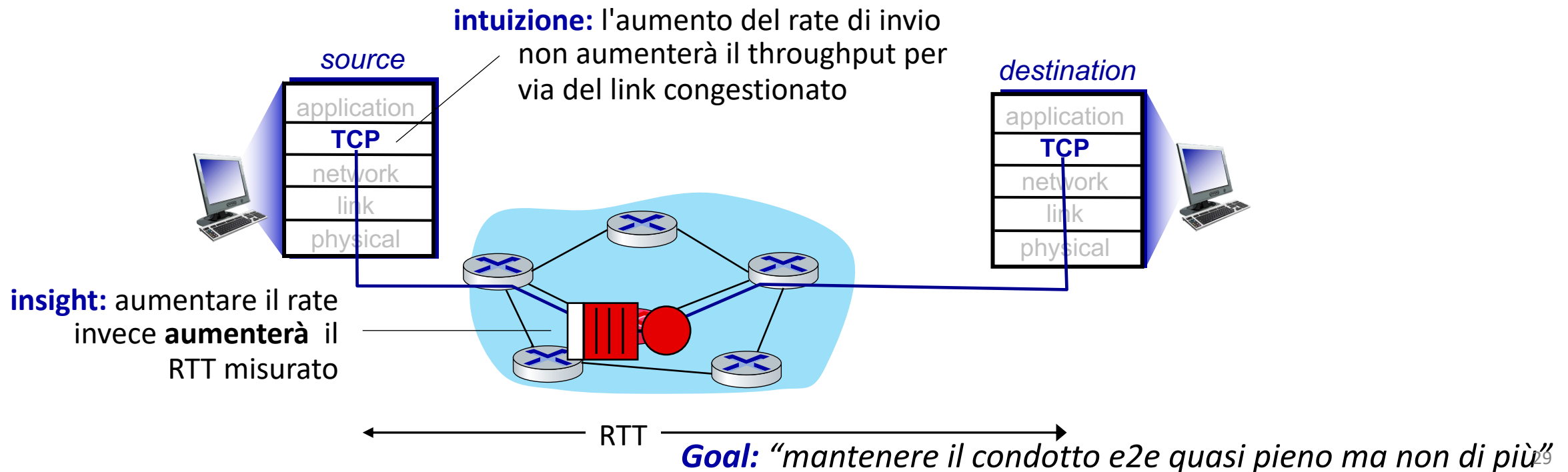
# TCP e il "link collo di bottiglia" congestionato

- TCP (classico e CUBIC) aumenta la il rate del mittente finché non si verifica la perdita di pacchetti all'uscita di un router: il *bottleneck link*



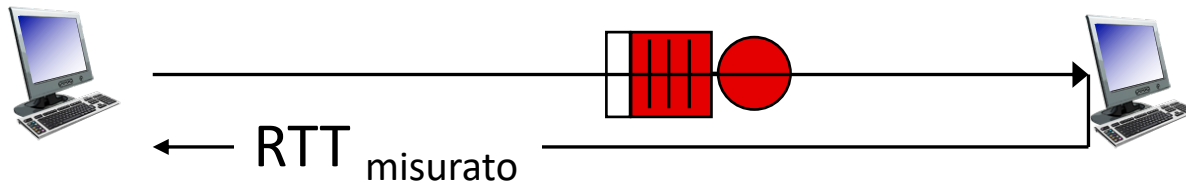
# TCP e il "link collo di bottiglia" congestionato

- TCP (classico, CUBIC) aumenta la il rate del mittente finché non si verifica la perdita di pacchetti all'uscita di un router: il *bottleneck link*
- concentriamoci sul link congestionato




# Delay-based TCP congestion control

Mantenere il condotto "quasi pieno": mantenere il bottleneck link occupato evitando ritardi/buffering elevati



$$\text{throughput misurato} = \frac{\text{\# byte inviati nell'ultimo RTT}}{\text{RTT}_{\text{misurato}}}$$

## Approccio basato sul delay:

- $\text{RTT}_{\min}$ : RTT minimo osservato (= quando percorso non congestionato) 
- throughput **non congestionato** dovrebbe essere  $\text{cwnd} / \text{RTT}_{\min}$

if measured throughput “very close” to uncongested throughput  
    increase  $\text{cwnd}$  linearly                      /\* since path not congested \*/  
else if measured throughput “far below” uncongested throughput  
    decrease  $\text{cwnd}$  linearly                      /\* since path is congested \*/

# Delay-based TCP congestion control

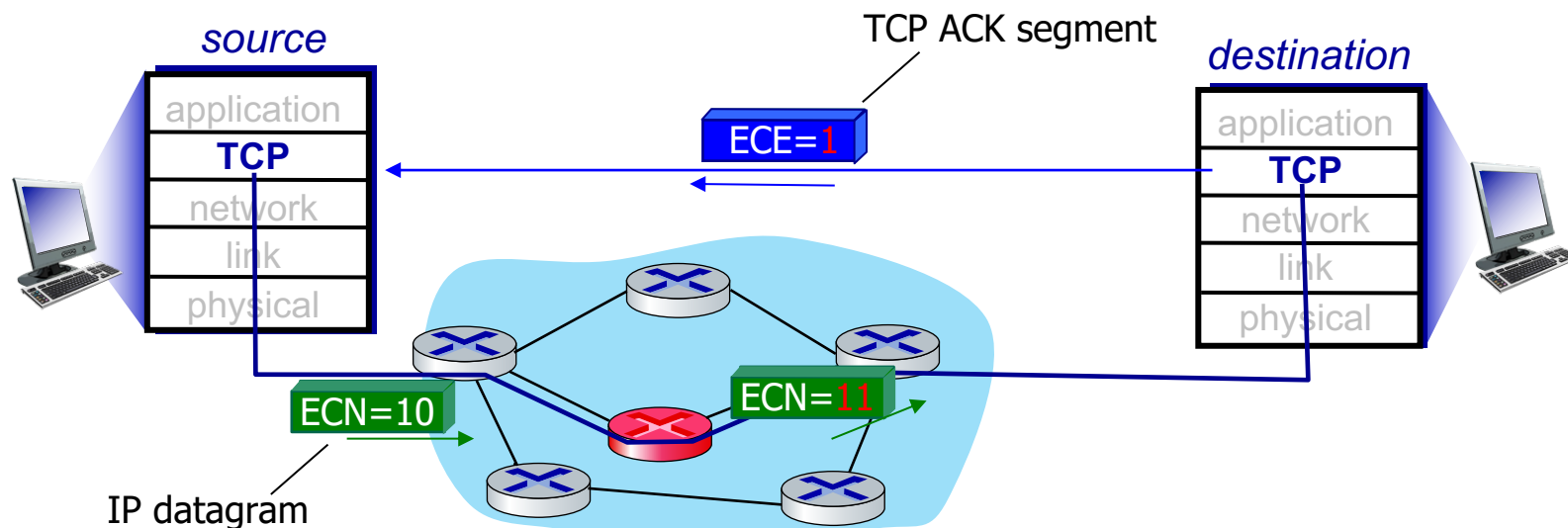
- Innovazione: controllo della congestione senza indurre/forzare perdite!
- Massimizzare il throughput ("condotto quasi pieno...") mantenendo basso il ritardo ("...ma non troppo pieno")
- Alcune versioni di TCP usano questo approccio
  - BBR implementato sulla rete backbone (interna) di Google



# Notifica esplicita di congestione (ECN)

distribuzioni TCP spesso implementano il controllo della congestione *assistito dalla rete (network-assisted)*:

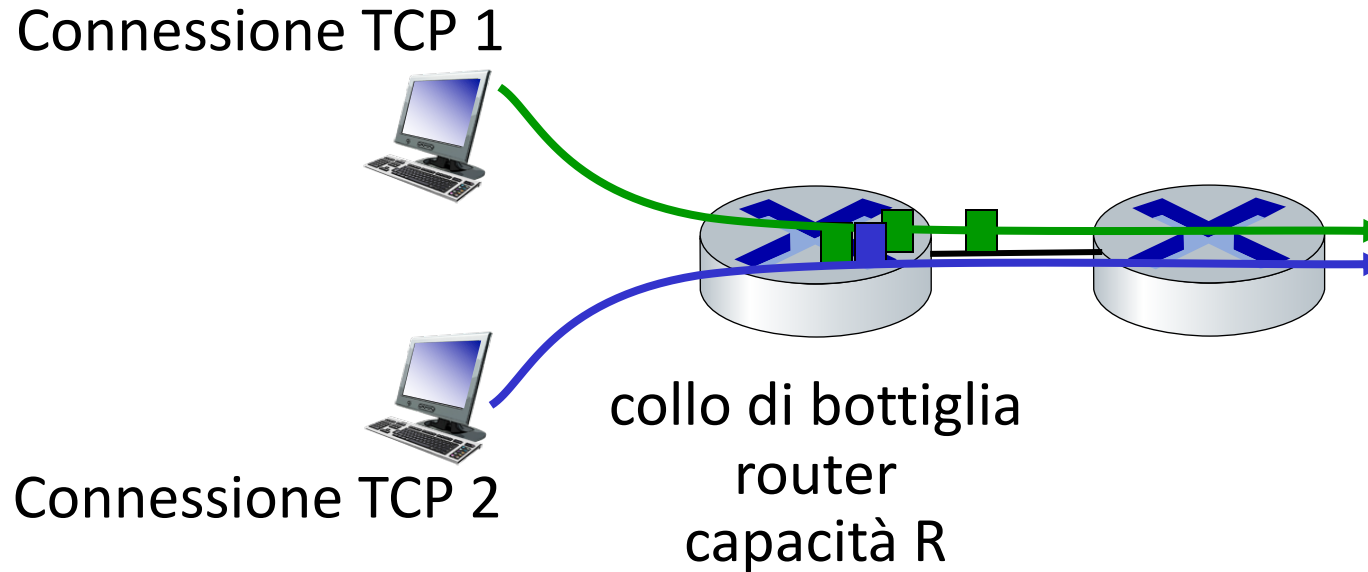
- due bit nell'intestazione IP ( campo type of service ToS ) contrassegnati *dal router di rete* per indicare la congestione
  - *policy interna* per determinare la marcatura scelta dall'operatore di rete e la politica di scarto dei pacchetti
- indicazione di congestione trasportata al destinatario
- il destinatario imposta il bit ECE sul segmento ACK per notificare al mittente la congestione
- coinvolge sia IP (marcatura bit ECN dell'intestazione IP) che TCP (marcatura bit C,E dell'intestazione TCP)





# Fairness di TCP

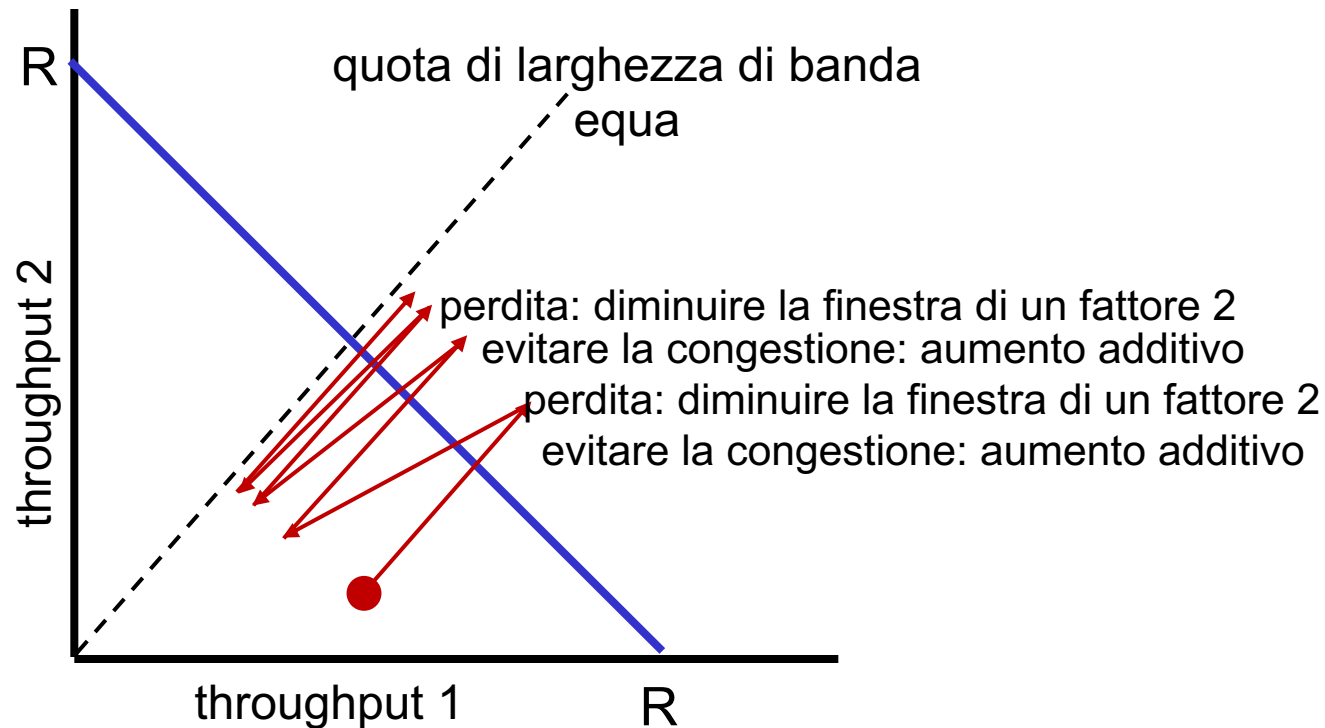
**Obiettivo di equità:** se  $K$  sessioni TCP condividono lo stesso collegamento a collo di bottiglia della larghezza di banda  $R$ , ciascuna dovrebbe avere una velocità media di  $R/K$



# D: TCP è equo?

Esempio: due sessioni TCP concorrenti:

- l'aumento additivo dà una pendenza di 1
- la diminuzione moltiplicativa riduce proporzionalmente il throughput



*TCP è equo?*

*R:* Sì, sotto ipotesi idealizzate:

- stesso RTT
- numero fisso di sessioni
- ...

# Fairness: tutte le app di rete devono essere "corrette"?

## Fairness e UDP

- le app multimediali spesso non utilizzano il protocollo TCP
  - non voglio che la velocità venga limitata dal controllo della congestione
- usa invece UDP:
  - invia audio/video a velocità costante, tollera la perdita di pacchetti, potenzialmente soffoca TCP
- non vi è alcuna "polizia di Internet" per imporre l'uso del controllo della congestione



## Fairness, connessioni TCP parallele

- l'applicazione può aprire *più* connessioni parallele tra due host
- i browser Web lo fanno, ad esempio, collegamento con rate  $R$  con 9 connessioni esistenti:
  - nuova app richiede 1 TCP, ottiene  $R/10$  vs.
  - nuova app richiede 11 TCP, ottiene  $R/2$

# Domande?

## ■ Argomenti importanti

- Formato dei protocolli principali visti finora (**DNS**, HTTP, UDP, TCP, HTTP, FTP)
- saper prevedere per i protocolli principali come un client e un server rispondono in varie situazioni
- Calcolo di ritardi di trasmissione, throughput e accodamenti

## ■ Esonero

- Una parte V/F
- Una parte di esercizi
- Una parte risposte aperte

# Livello di trasporto: sommario

- Servizi a livello di trasporto
- Multiplexing e demultiplexing
- Trasporto senza connessione: UDP
- Principi di trasferimento affidabile dei dati
- Trasporto orientato alla connessione: TCP
  - struttura del segmento
  - trasferimento affidabile dei dati
  - controllo del flusso
  - gestione della connessione
- Principi di controllo della congestione
- Controllo della congestione TCP
- **Evoluzione della funzionalità del livello di trasporto**

# Funzionalità a livello di trasporto in evoluzione

- TCP, UDP: principali protocolli di trasporto da 40 anni
- diverse varianti di TCP sviluppate, per scenari specifici:

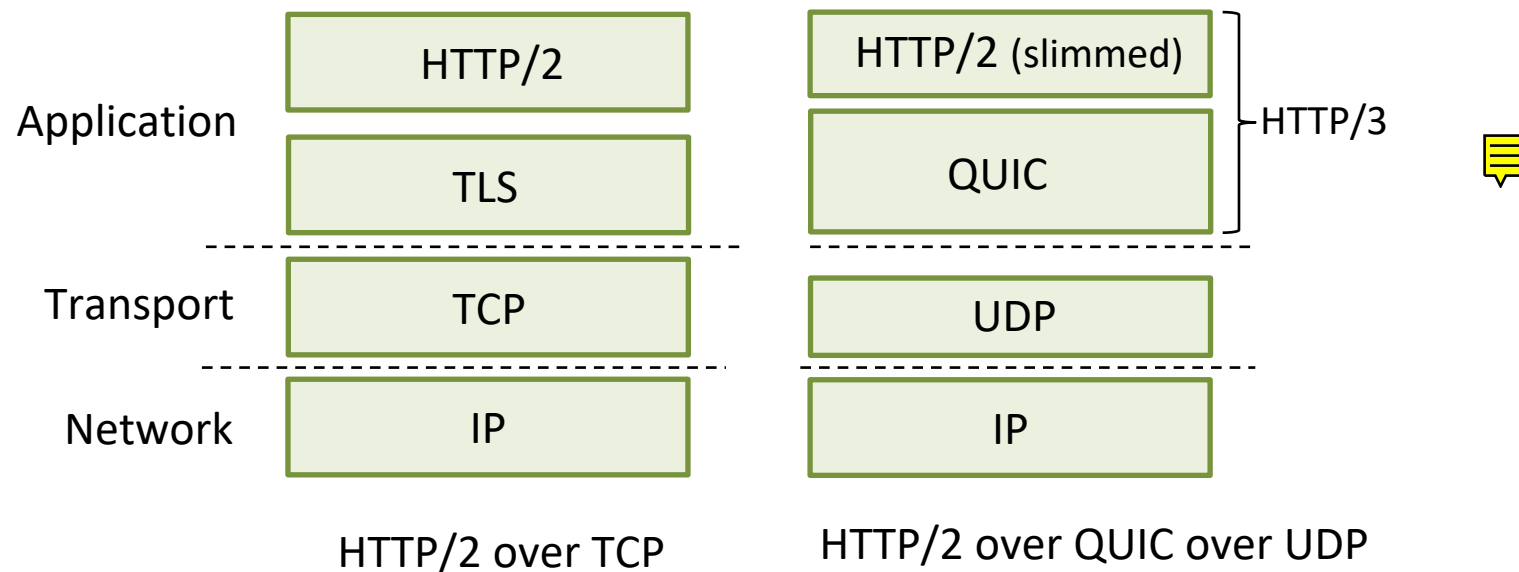
Scenario	Challenges
Long, fat pipes (large data transfers)	Many packets “in flight”; loss shuts down pipeline
Wireless networks	Loss due to noisy wireless links, mobility; TCP treat this as congestion loss
Long-delay links	Extremely long RTTs
Data center networks	Latency sensitive
Background traffic flows	Low priority, “background” TCP flows



- Trend: spostare le funzioni del livello di trasporto al livello applicazione, sopra UDP
  - HTTP/3: QUIC

# QUIC: Quick UDP Internet Connections

- protocollo a livello di applicazione, sopra UDP
  - aumentare le prestazioni di HTTP
  - implementato su molti server e app di Google (Chrome, app YouTube per dispositivi mobili)



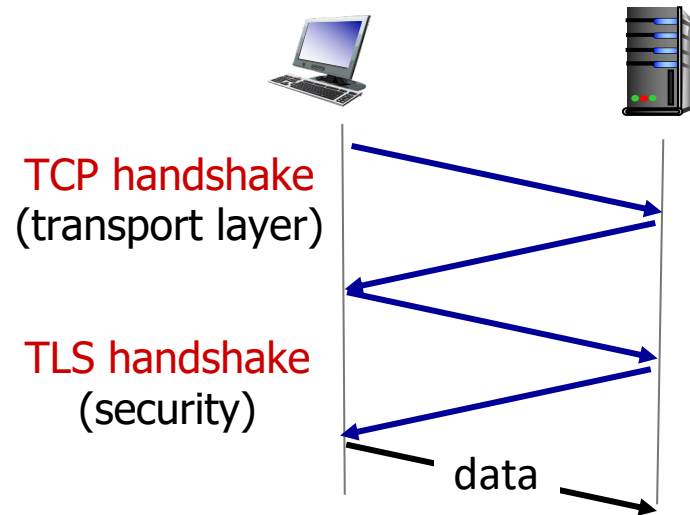
# QUIC: Quick UDP Internet Connections

adotta gli approcci che abbiamo studiato per la creazione di connessioni, il controllo degli errori, il controllo della congestione

- **controllo degli errori e della congestione:** "I lettori che hanno familiarità con il rilevamento delle perdite e il controllo della congestione del TCP troveranno qui algoritmi simili a quelli ben noti del TCP." [dalla specifica QUIC]
- **creazione della connessione:** affidabilità, controllo della congestione, autenticazione, crittografia, connessione rapida in un RTT
- più "stream" a livello di applicazione multiplexati su una singola connessione QUIC
  - trasferimento dati affidabile separato, sicurezza
  - controllo della congestione in comune

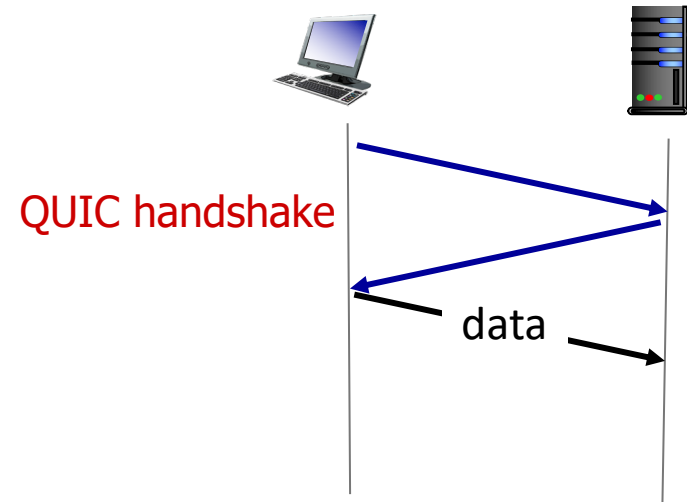


# QUIC: Stabilire la connessione



TCP (reliability, congestion control state) + TLS (authentication, crypto state)

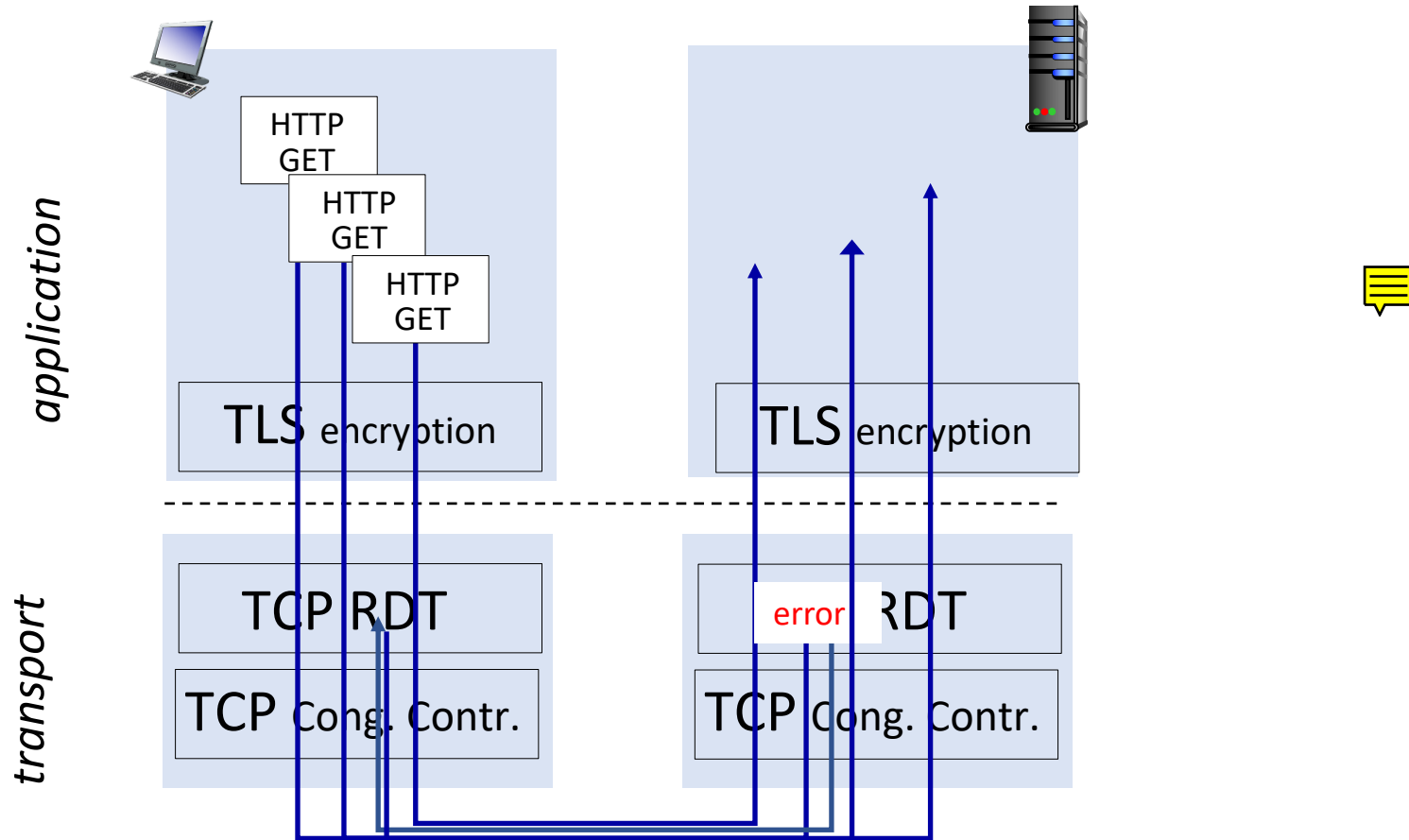
- 2 serial handshakes



QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake

# QUIC: streams: parallelism, no HOL blocking



(a) HTTP 1.1