Università degli Studi di Roma "Tor Vergata"
Laurea in Informatica

Sistemi Operativi e Reti
(modulo Reti)
a.a. 2024/2025

Livello di trasporto (parte4)

dr. Manuel Fiorelli

manuel.fiorelli@uniroma2.it

https://art.uniroma2.it/fiorelli

Basate sulle slide del libro di testo:
https://gaia.cs.umass.edu/kurose_ross/ppt.php

Capitolo 3: tabella di marcia

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

Principi del controllo della congestione

Congestione:

- informalmente: “troppe sorgenti inviano troppi dati troppo velocemente perché la *rete* li gestisca”
- sintomi:
  - lunghi ritardi (accodamento nei buffer dei router)
  - pacchetti persi (overflow nei buffer dei router)
- differisce dal controllo di flusso!
- tra i dieci problem più important del networking

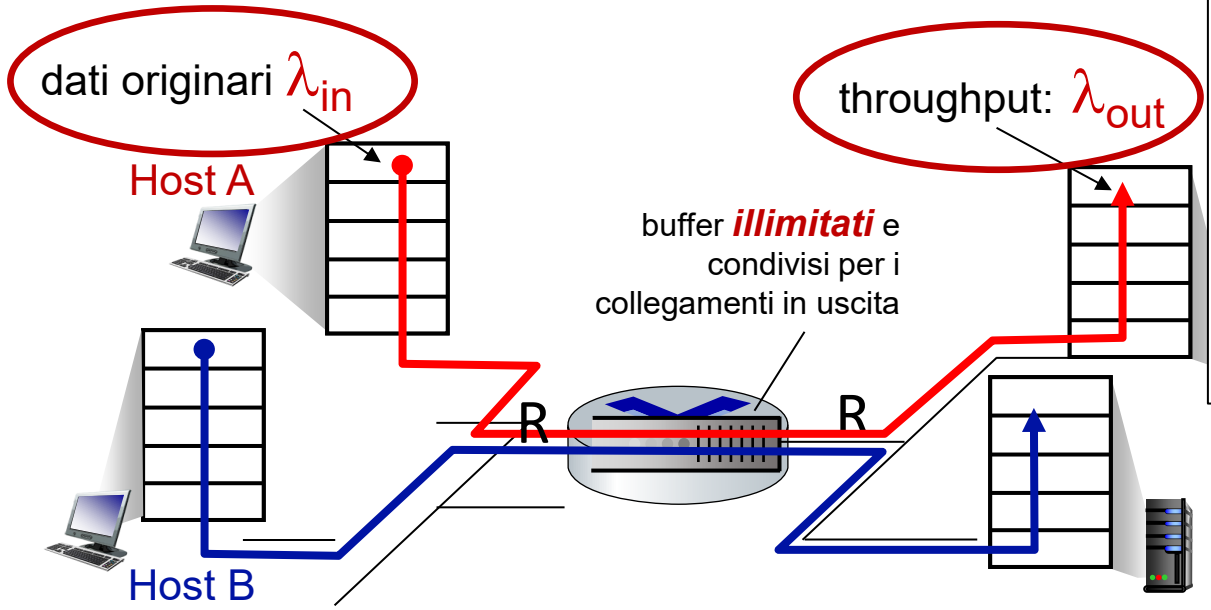
controllo della congestione: troppi mittenti, che trasmettono troppo velocemente

controllo di flusso: un mittente troppo veloce per il destinatario

Cause/costi della congestione: scenario 1

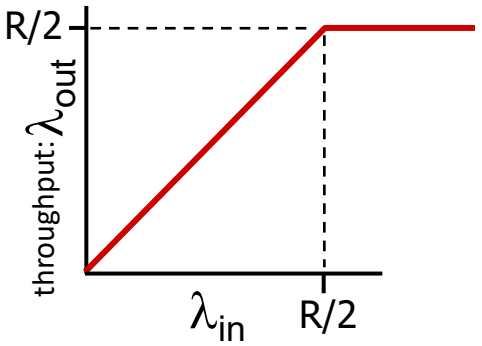
Caso più semplice:

- un router con buffer illimitati
- capacità dei collegamenti di ingresso e uscita: R
- due flussi
- nessuna ritrasmissione

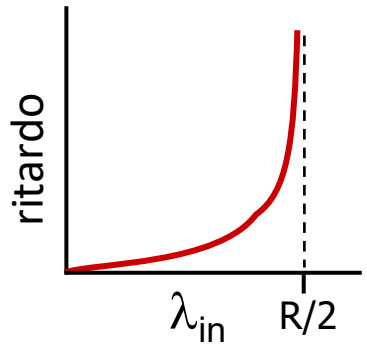


throughput a livello di applicazione (bit di informazione **utili** consegnati all'applicazione nell'unità di tempo), detto *goodput*

D: Cosa accade quando il tasso di arrivo λ_{in} si avvicina a R/2?



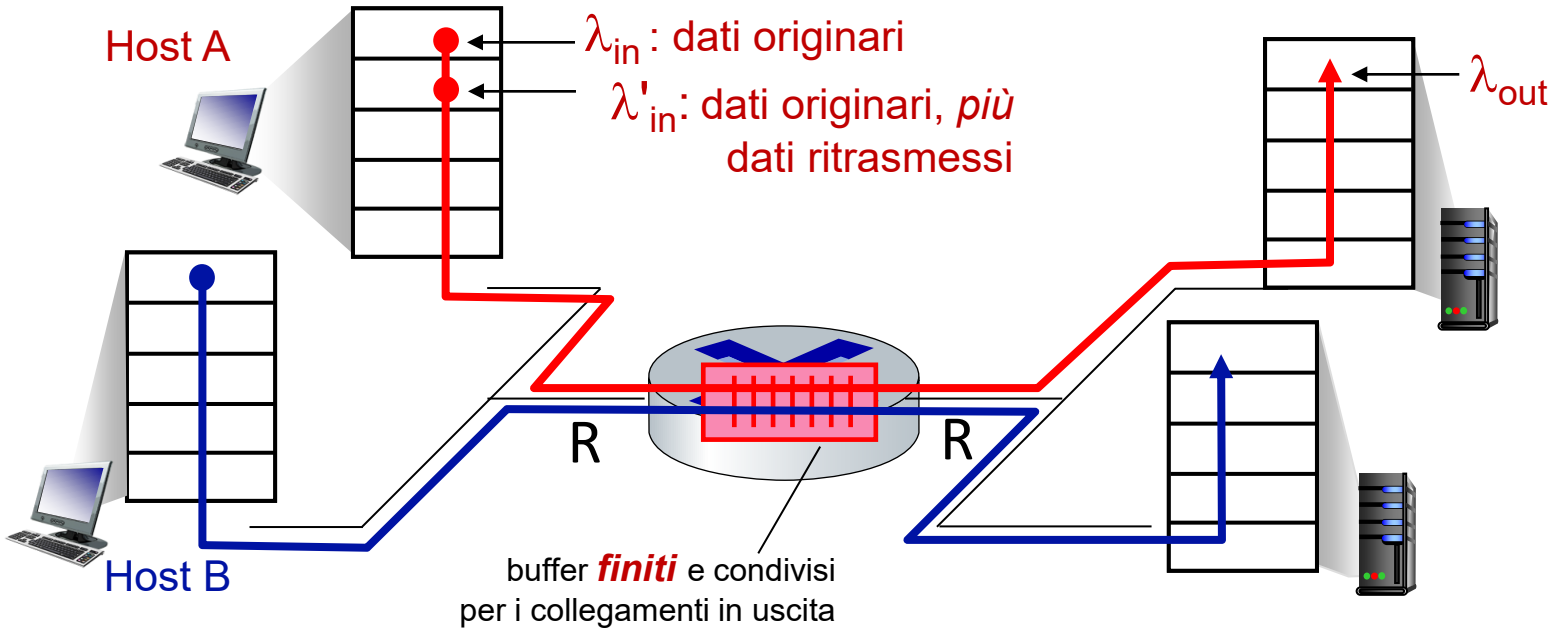
throughput massimo per connessione: R/2



grandi ritardi quando il tasso di arrivo si avvicina alla capacità del collegamento

Cause/costi della congestione: scenario 2

- un router, buffer *finiti*
- il mittente ritrasmette pacchetti perduti (scartati perché il buffer era pieno)
  - input del livello di applicazione (tasso di trasmissione verso la socket): λ_{in}
  - input del livello di traporto (tasso di trasmissione verso la rete) include le *ritrasmissioni* : $\lambda'_{in} \geq \lambda_{in}$

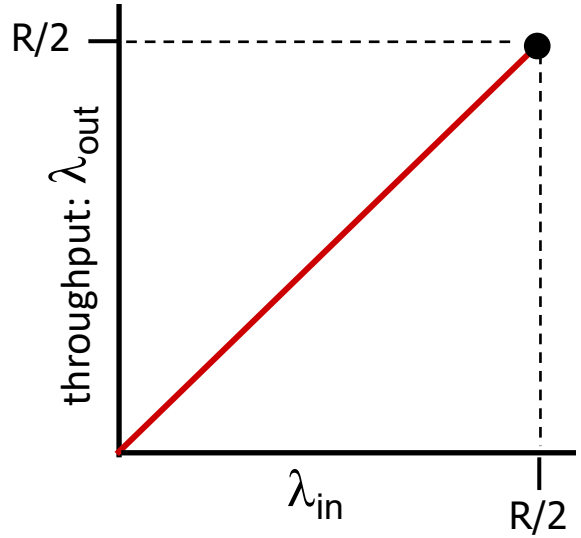


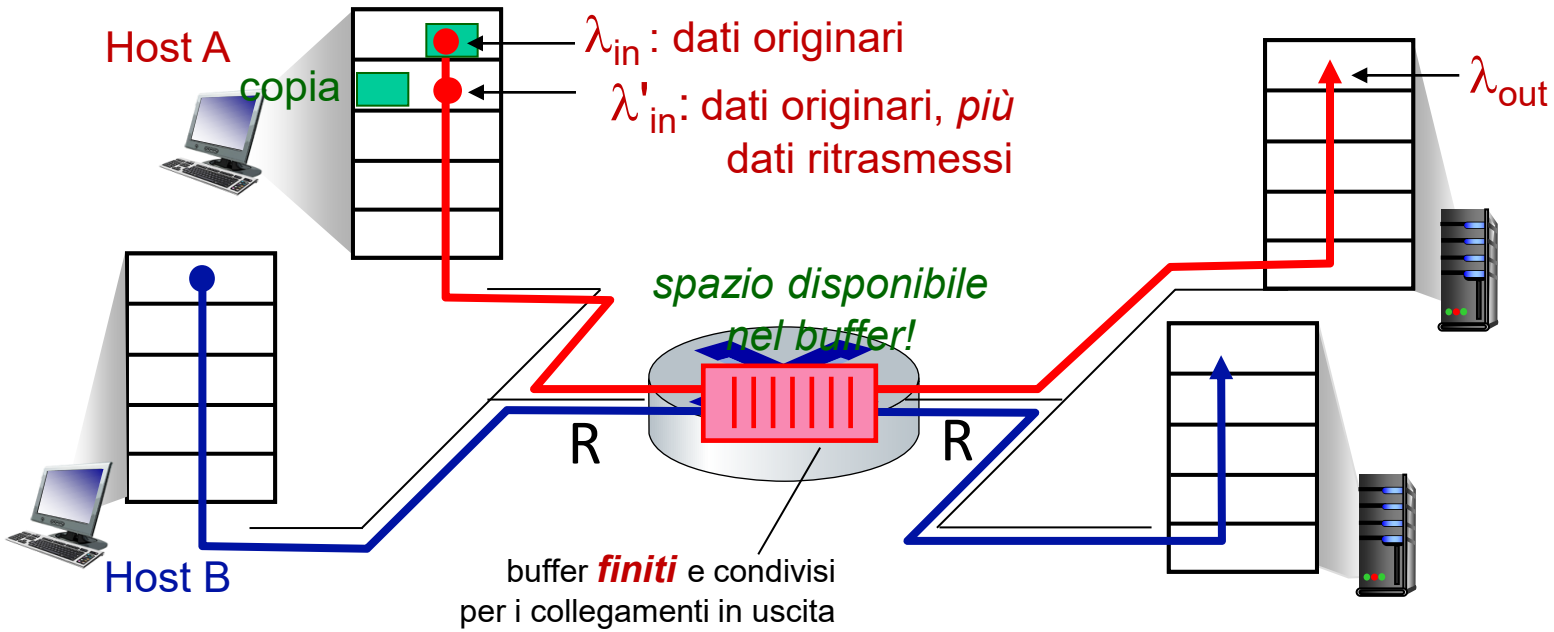
λ'_{in} detto anche carico offerto (offered load) alla rete

Cause/costi della congestione: scenario 2

idealizzazione: conoscenza perfetta

- il mittente invia solo quando c'è spazio disponibile nei buffer del router (nessuna perdita, $\lambda_{in} = \lambda'_{in}$)

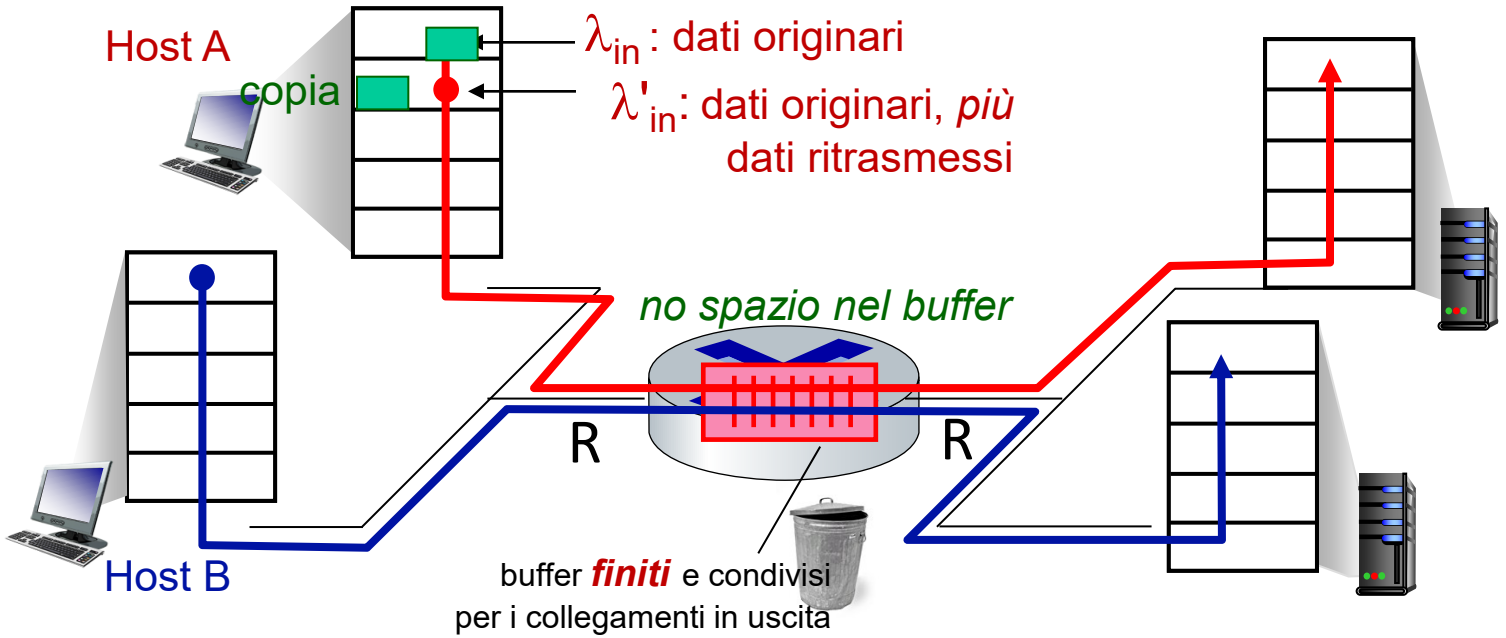




Cause/costi della congestione: scenario 2

idealizzazione: *un po' di conoscenza perfetta*

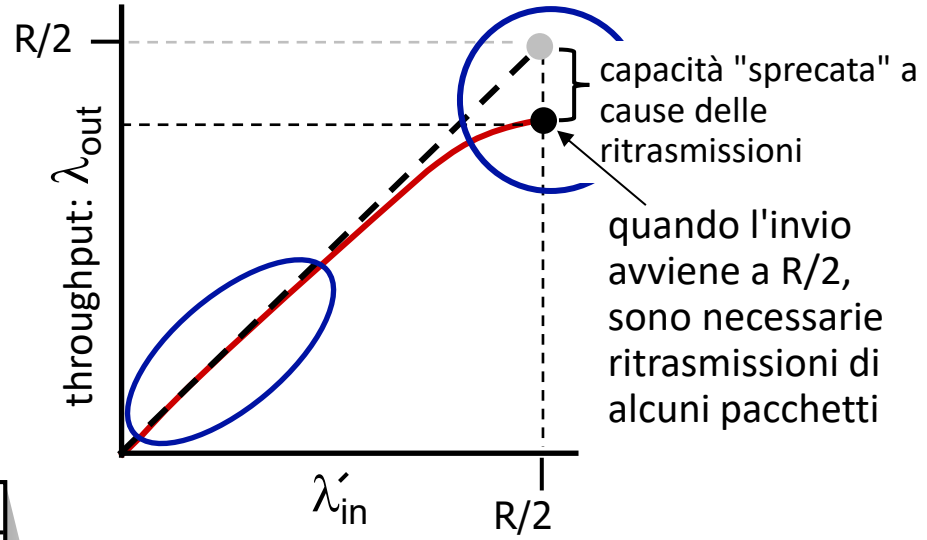
- i pacchetti possono essere persi (scartati nel router) a causa di buffer pieni
- il mittente sa quando un pacchetto è stato scartato: ritrasmette un pacchetto solo se *sa* che è stato perso

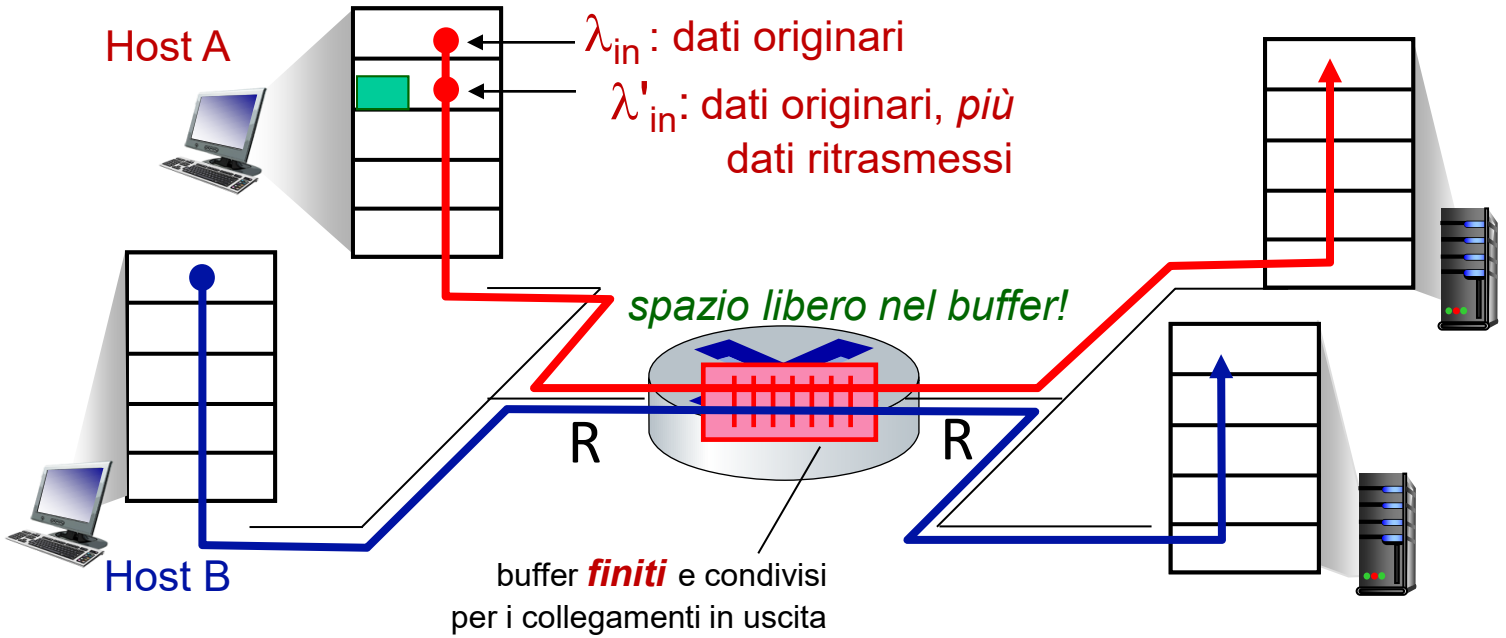


Cause/costi della congestione: scenario 2

idealizzazione: *un po' di conoscenza perfetta*

- i pacchetti possono essere persi (scartati nel router) a causa di buffer pieni
- il mittente sa quando un pacchetto è stato scartato: ritrasmette un pacchetto solo se *sa* che è stato perso

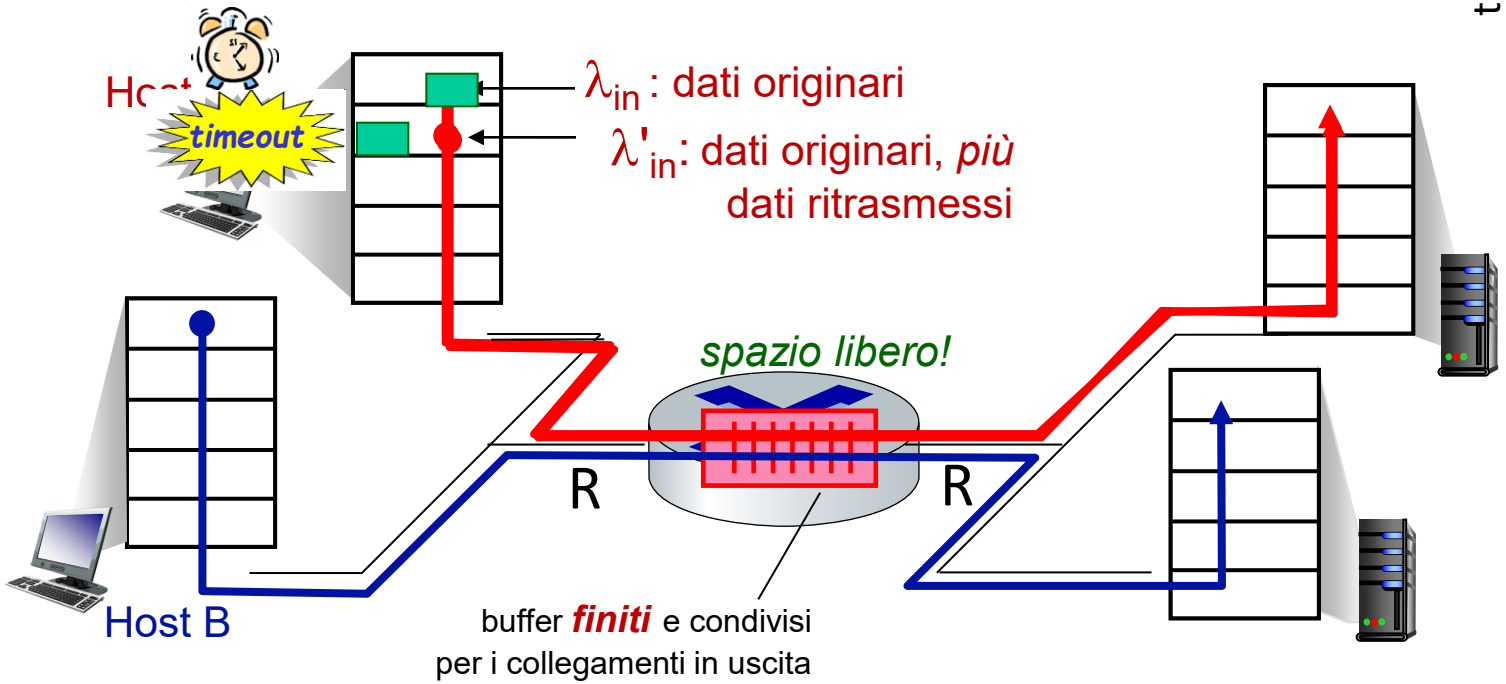


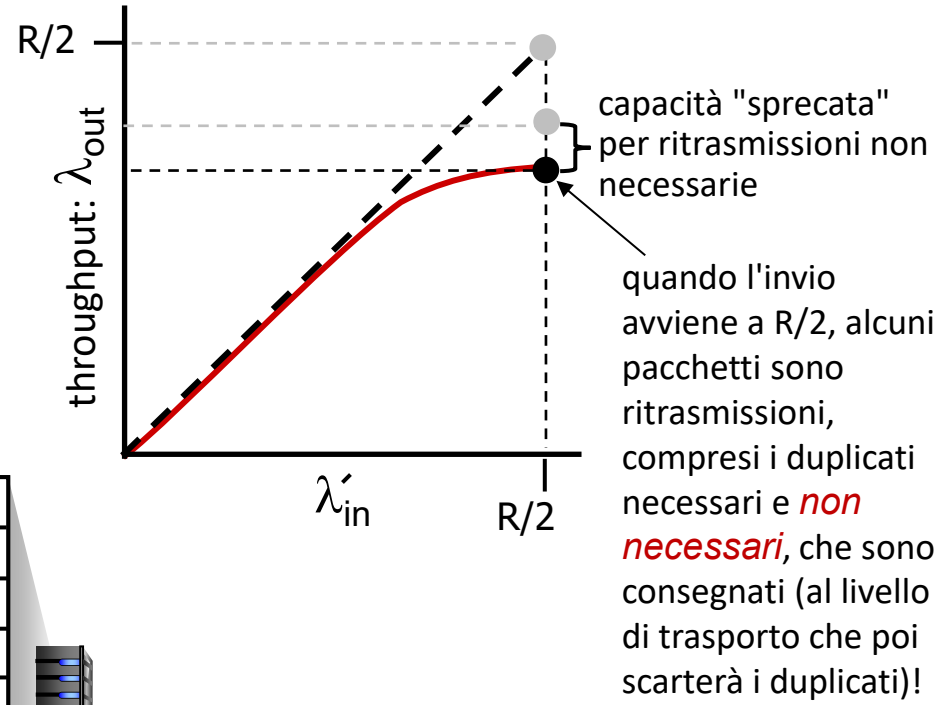


Cause/costi della congestione: scenario 2

scenario realistico: *duplicati non necessari*

- i pacchetti possono essere persi, scartati dal router a causa dei buffer pieni, richiedendo ritrasmissioni
- ma il mittente può andare in timeout prematuramente, inviando *due* copie, che vengono *entrambe* consegnate

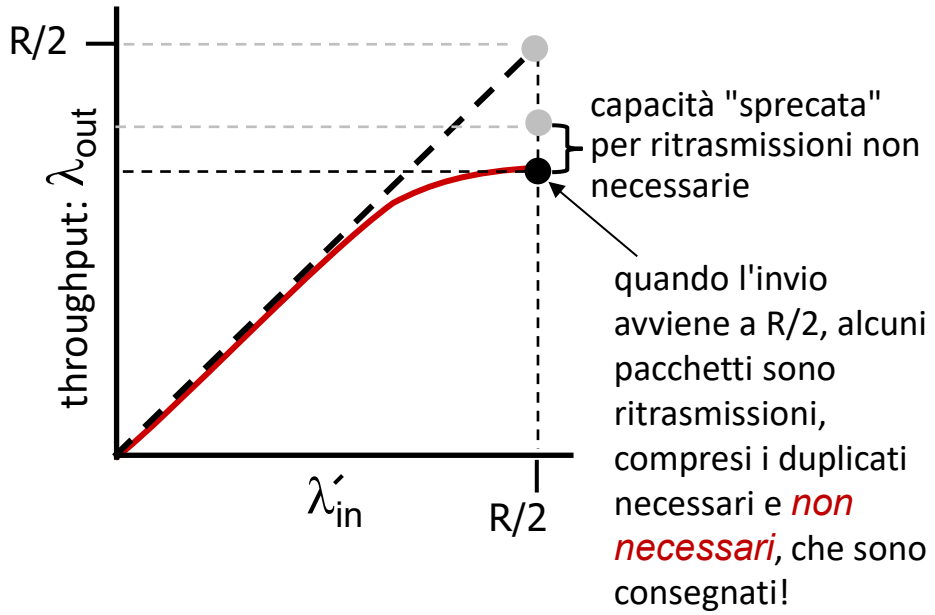




Cause/costi della congestione: scenario 2

scenario realistico: *duplicati non necessari*

- i pacchetti possono essere persi, scartati dal router a causa dei buffer pieni, richiedendo ritrasmissioni
- ma il mittente può andare in timeout prematuramente, inviando *due* copie, che vengono *entrambe* consegnate



"costi" della congestione;

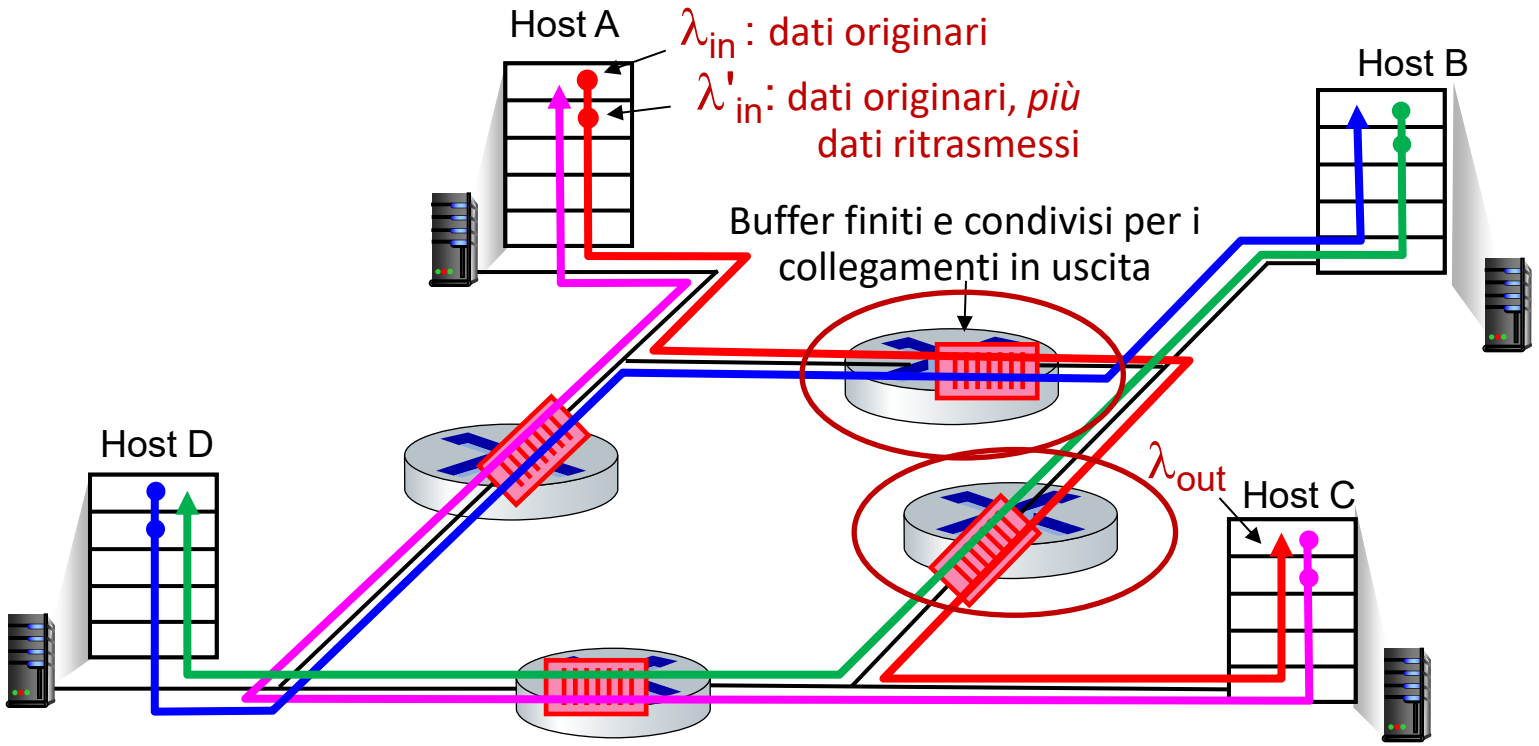
- più lavoro (ritrasmissioni) per un dato throughput di ricezione
- ritrasmissioni non necessarie: il collegamento trasporta più copie del pacchetto
  - diminuzione del throughput massimo raggiungibile

Cause/costi della congestione: scenario 3

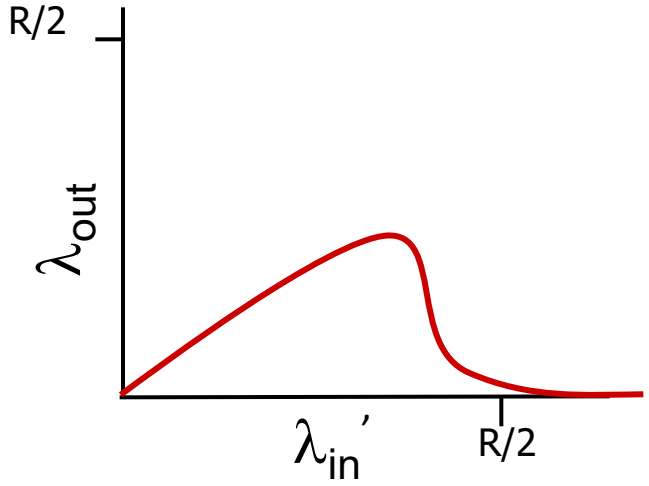
- *quattro mittenti*
- percorsi *multi-hop*
- timeout/ritrasmissione

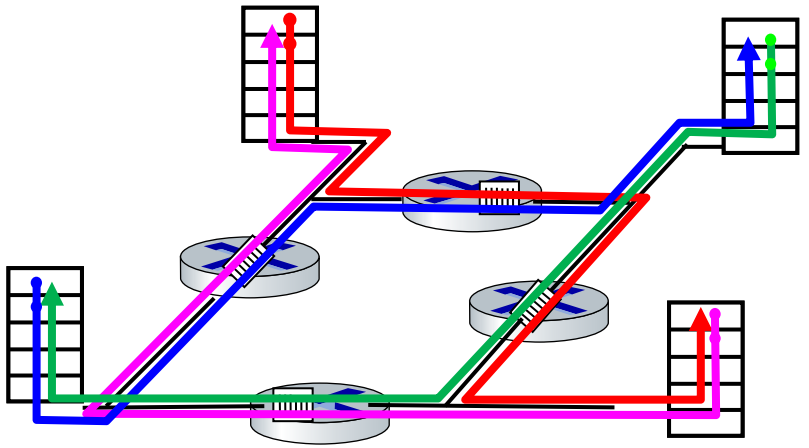
D: che cosa accade quando λ_{in} e λ_{in}' aumentano?

R: quando λ_{in}' aumenta, tutti i pacchetti blu in arrivo alla coda in alto sono scartati, throughput blu → 0



Cause/costi della congestione: scenario 3



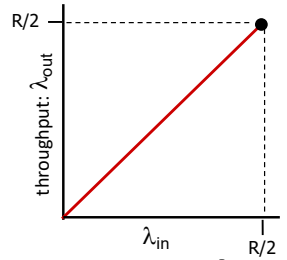


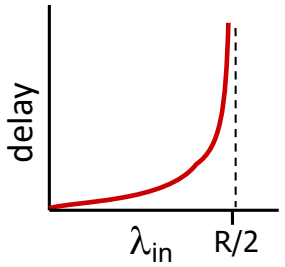
Un altro "costo" della congestione:

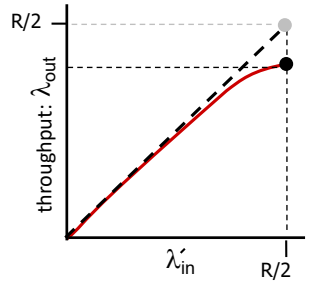
- Quando il pacchetto viene scartato, la capacità trasmissiva utilizzata sui collegamenti di upstream per instradare il pacchetto risulta sprecata!

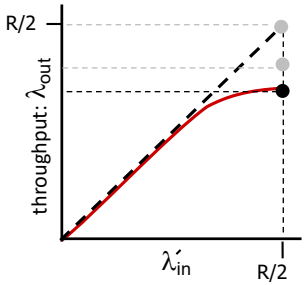
Cause/costi della congestione: intuizioni

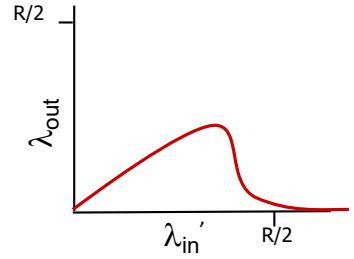
- il throughput non può mai superare la capacità
- il ritardo aumenta mentre ci si avvicina alla capacità
- la perdita/ritrasmissione diminuisce il throughput effettivo
- i duplicati non necessari diminuiscono ulteriormente il throughput effettivo
- capacità di trasmissione a monte / buffering sprecato per i pacchetti persi a valle







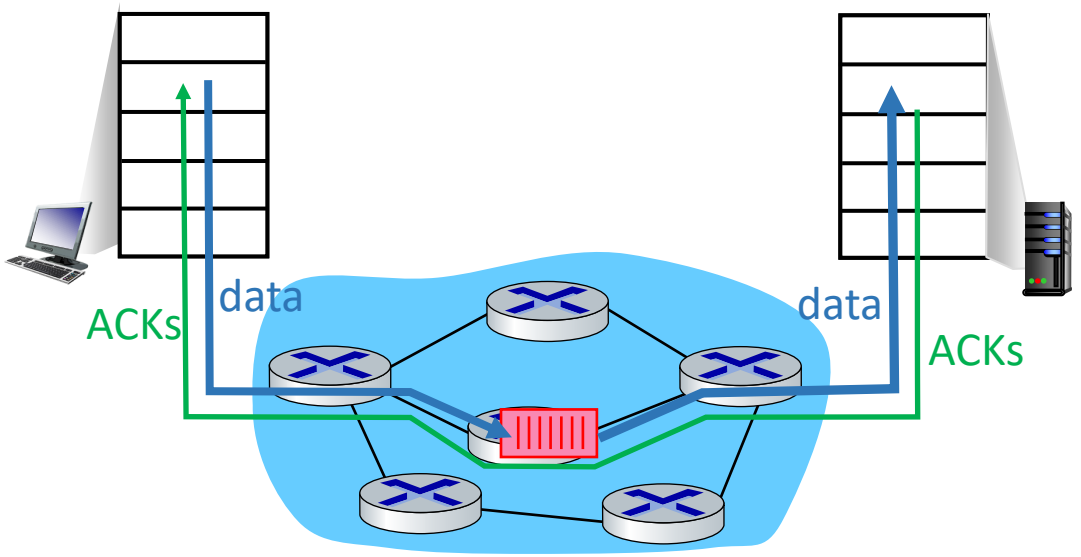




Approcci al controllo della congestione

Controllo della congestione end-to-end

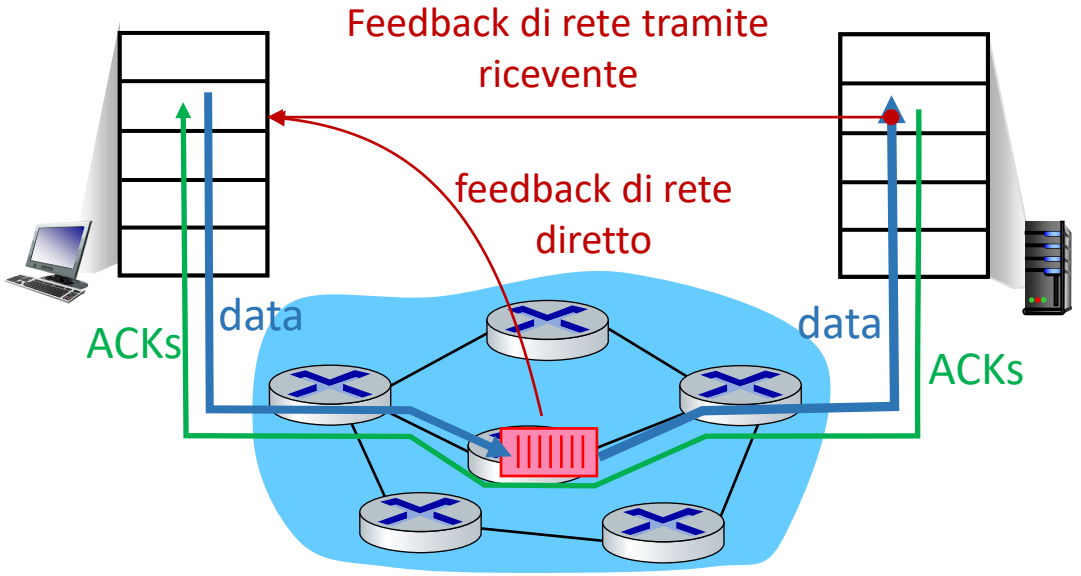
- nessun supporto esplicito dalla rete
- la congestione è *dedotta* osservando le perdite e i ritardi nei sistemi terminali
- metodo adottato da TCP



Approcci al controllo della congestione

Controllo della congestione assistito dalla rete:

- i router forniscono un feedback *diretto* all'host mittente tramite un *chokepacket* che lo avvisa dello stato di congestione
- oppure, un router può marcare i pacchetti che lo attraversano in modo tale che il destinatario alla sua ricezione informi il mittente
- possono indicare il livello di congestione o impostare esplicitamente un tasso di invio
- protocolli TCP ECN, ATM, DECbit



Capitolo 3: tabella di marcia

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

Trasmissione dati affidabile v. Controllo della congestione

- Trasmissione dati affidabile
  - reagisce alla perdita (e alla corruzione) dei pacchetti, possibilmente causata dalla congestione
  - "tratta i sintomi della congestione"
- Controllo della congestione
  - "cura la malattia"
  - evita che la malattia si aggravi fino a degenerare fino allo scenario di "collasso di congestione"

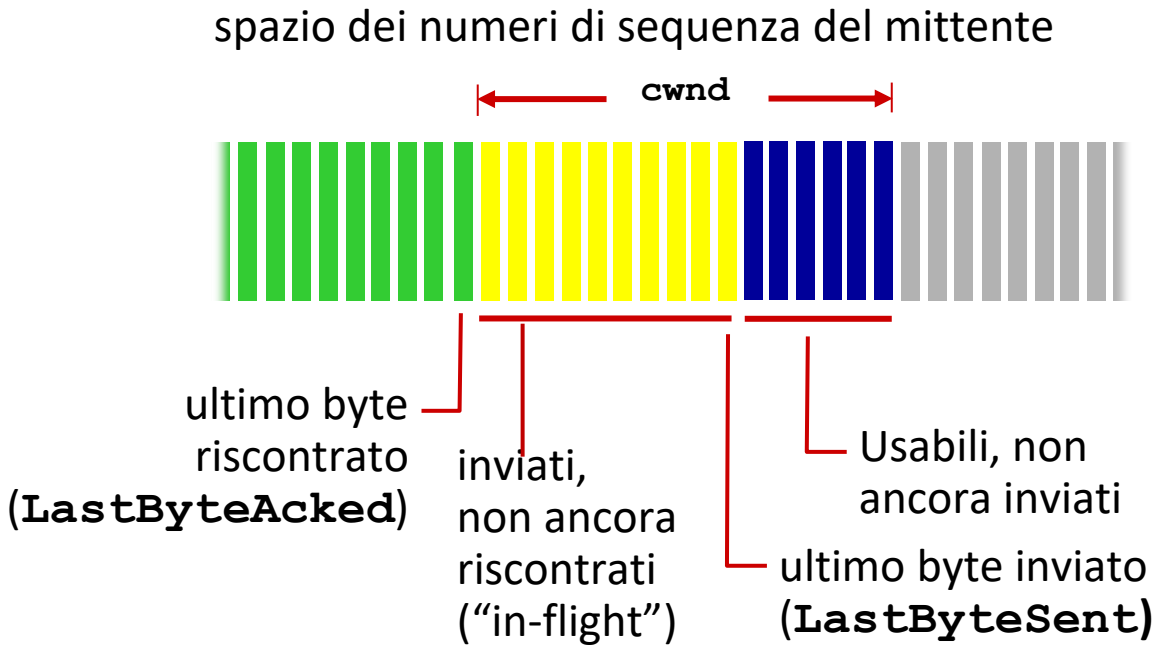
Controllo di congestione TCP

- TCP "classico" (come sviluppato da RFC 2581 e successivamente da RFC 5681)
  - controllo di congestione end-to-end
  - sviluppato in risposta alla serie di collassi di collasso di congestione subiti da Internet agli albori

    > Nell'ottobre dell'86, Internet subì il primo di una serie di "crolli di congestione". Durante questo periodo, il throughput dei dati dal LBL alla UC Berkeley ([...]) scese da 32 kbps a 40 bps.

    Van Jacobson; Michael J. Karels (November 1988), Congestion Avoidance and Control: https://ee.lbl.gov/papers/congavoid.pdf
- Evoluzioni recenti di TCP
  - indicazione di congestione esplicita fornita dalla rete
  - atre modifiche rispetto a TCP classico

Controllo della congestion TCP: dettagli



Comportamento di invio di TCP:

- *all'incirca:* invia `cwnd` byte, attende RTT per gli ACKS, quindi invia ulteriori byte

$$\text{tasso di invio} \approx \frac{\texttt{cwnd}}{\text{RTT}} \text{ byte/s}$$

- Il mittente limita la trasmissione: **`LastByteSent - LastByteAcked ≤ cwnd`**
- `cwnd` viene regolata dinamicamente in risposta alla congestione della rete osservata (implementando il controllo della congestione TCP)

Relazione col controllo di flusso

- Il controllo del flusso regola la quantità e la velocità dei dati inviati in funzione della finestra di ricezione comunicata dal destinatario **`rwnd`** (byte liberi nel buffer di ricezione del destinatario)
- Quindi,

  `LastByteSent - LastByteAcked ≤ rwnd`
- Combinando questo vincolo con quello visto in precedenza

  `LastByteSent - LastByteAcked ≤ min{rwnd, cwnd}`

Assumendo che il buffer di ricezione sia sufficiente grande, possiamo trascurare il vincolo della finestra di ricezione (che assumiamo sempre maggiore della finestra di congestione)

Controllo di congestione TCP: incremento additivo e decremento moltiplicativo (AIMD)

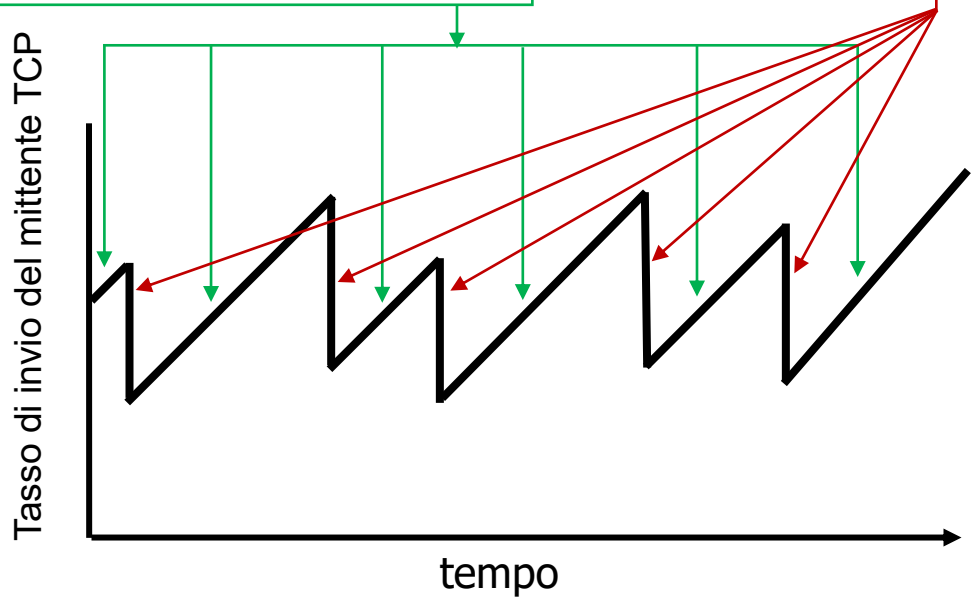
- *approccio:* i mittenti possono aumentare il tasso di invio, fino a quando non si verifica la perdita di pacchetti (congestione), quindi diminuire la velocità di invio in caso di perdita

Incremento additivo

aumentare la velocità di invio di 1 MSS ogni RTT fino a quando non viene rilevata una perdita

Decremento moltiplicativo

dimezzare la velocità di invio ad ogni evento di perdita



AIMD dente di sega: *sondare* la larghezza di banda

Questa è una descrizione di alto livello, che ignora la fase iniziale di slow start e assume che le perdite siano indicate da un triplo ACK duplicato in presenza di fast recovery

TCP AIMD: di più

Dettaglio decremento moltiplicativo:

In TCP Reno, il mittente riduce il tasso di invio in risposta a eventi di perdita:

- dimezzamento in caso di perdita rilevata da un triplo ACK duplicato (passando poi – per beve tempo – nella fase di fast recovery)
- taglio a 1 MSS ("maximum segment size") quando la perdita è rilevata dal timeout (ritornando poi nella fase di slow start)

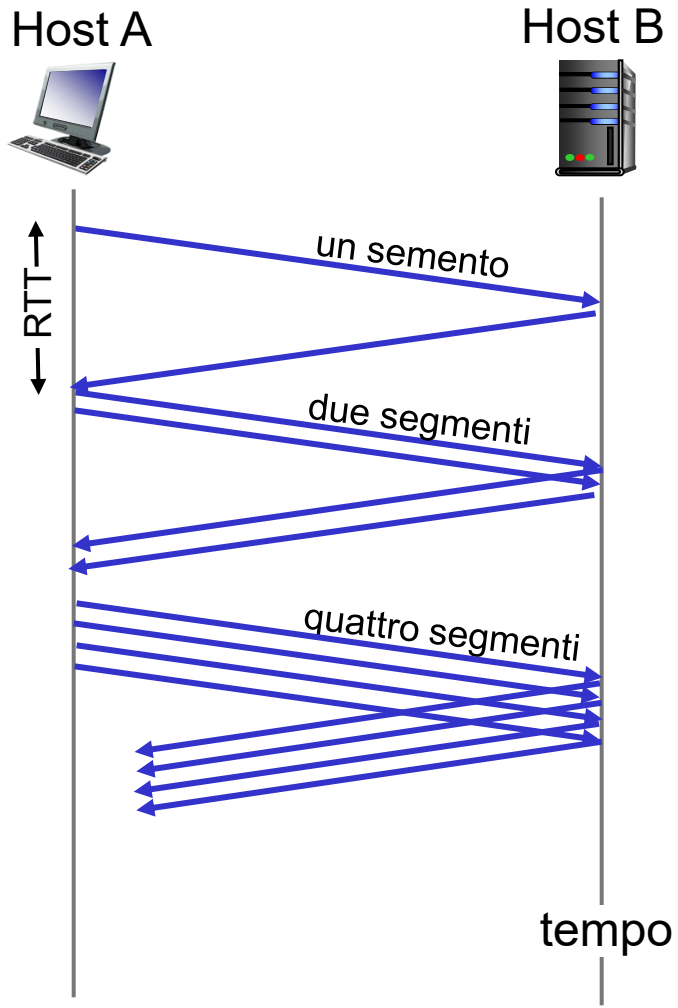
Una versione precedente, detta TCP Tahoe, la risposta a qualsiasi evento di perita era il taglio a 1 MSS e il passaggio a slow start

Perché AIMD?

- AIMD – un algoritmo asincrono, distribuito – è stato dimostrato che:
  - ottimizza i flussi congestionati in tutta la rete!
  - ha proprietà desiderabili di stabilità

Slow start (partenza lenta)

- Quando inizia la connessione, la frequenza aumenta in modo esponenziale fino a quando non si verifica un evento di perdita:
  - inizialmente **`cwnd`** = 1 MSS
  - raddoppia **`cwnd`** ogni RTT
  - fatto incrementando cwnd per ogni ACK ricevuto
- *sintesi*: il tasso iniziale è lento, ma aumenta in modo esponenziale e veloce



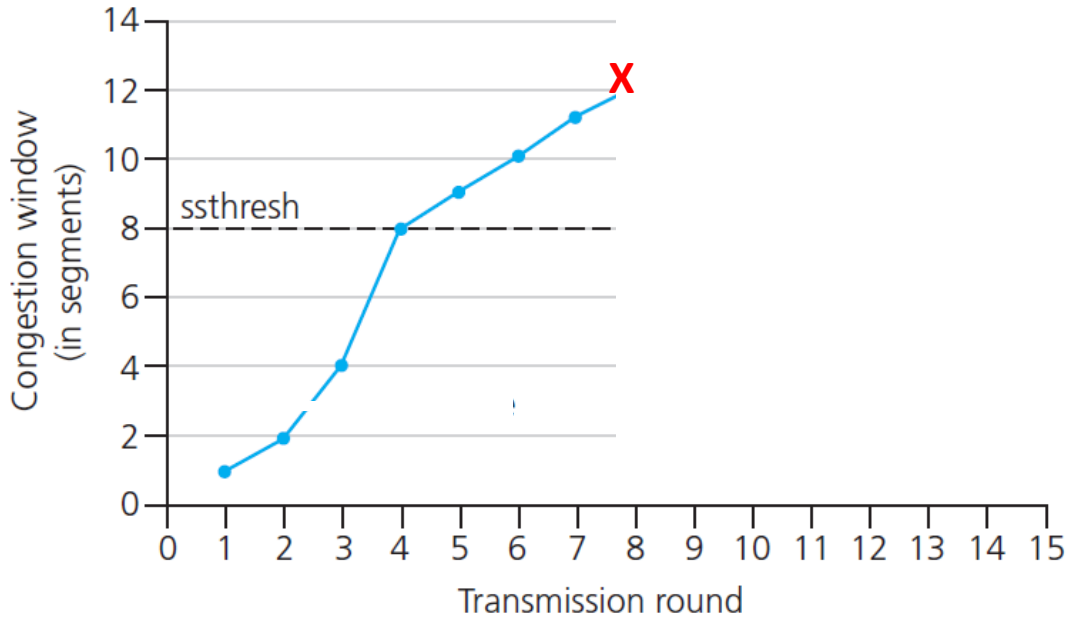
TCP: da slow start a congestion avoidance

D: quando l'aumento esponenziale dovrebbe passare a quello lineare?

R: quando **`cwnd`** raggiunge 1/2 del suo valore prima del timeout.

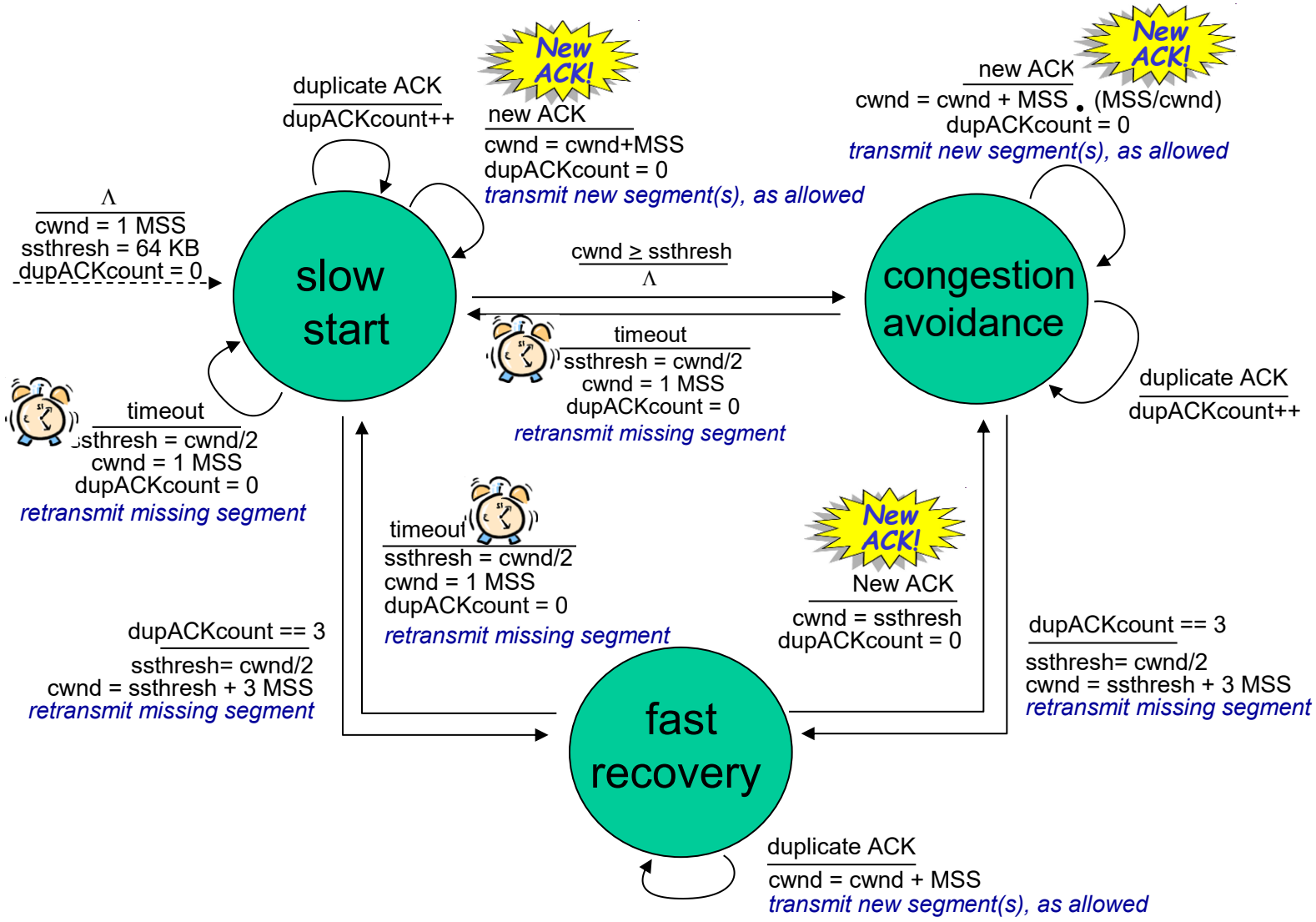
implementazione:

- **`ssthresh`** variabile (all'inizio 64 KB)
- in caso di evento di perdita, **`ssthresh`** è impostato a 1/2 **`cwnd`** giusto prima dell'evento di perdita



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Riassunto: controllo della congestione TCP

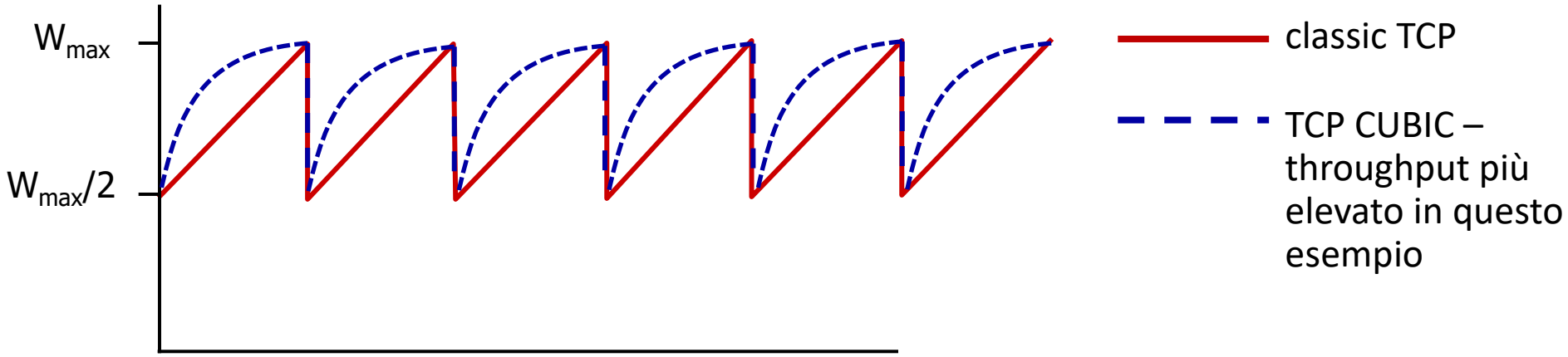


Riassunto: controllo della congestione TCP

- quando arriva un *nuovo ACK*, la finestra del mittente scorre verso destra e (a meno di recente riduzione della finestra di congestione), dovrebbe essere possibile l'invio nuovi segmenti (un segmento è uscito dalla rete e uno ne entra); quando la finestra di congestione viene incrementata sufficientemente può essere possibile l'invio di ulteriori segmenti (rivedete ciò che accade con slow start)
- nella fase di *congestion avoidance,* la finestra di congestione viene incrementata di $MSS \cdot \frac{MSS}{cwnd}$ per ogni nuovo ACK; poiché in un RTT ci si aspettano $\frac{cwnd}{MSS}$ ACK, il risultato finale (ottenuto moltiplicando queste due quantità) è un incremento di 1 MSS ogni RTT
- nella fase di *fast recovery*, in assenza di *nuovi ACK*, la finestra del mittente non avanza verso destra, bloccando *temporaneamente* l'invio di nuovi segmenti; tuttavia, la *cwnd* viene incrementata per ogni ACK duplicato (che indica l'arrivo a destinazione di un segmento successivo a quello potenzialmente perso) finché non diventa sufficientemente grande da permettere l'invio di nuovi segmenti (in media, in $\frac{1}{2} RTT$)
- la fase di *fast recovery* dura all'incirca un RTT affinché arrivi l'ACK del segmento ritrasmesso

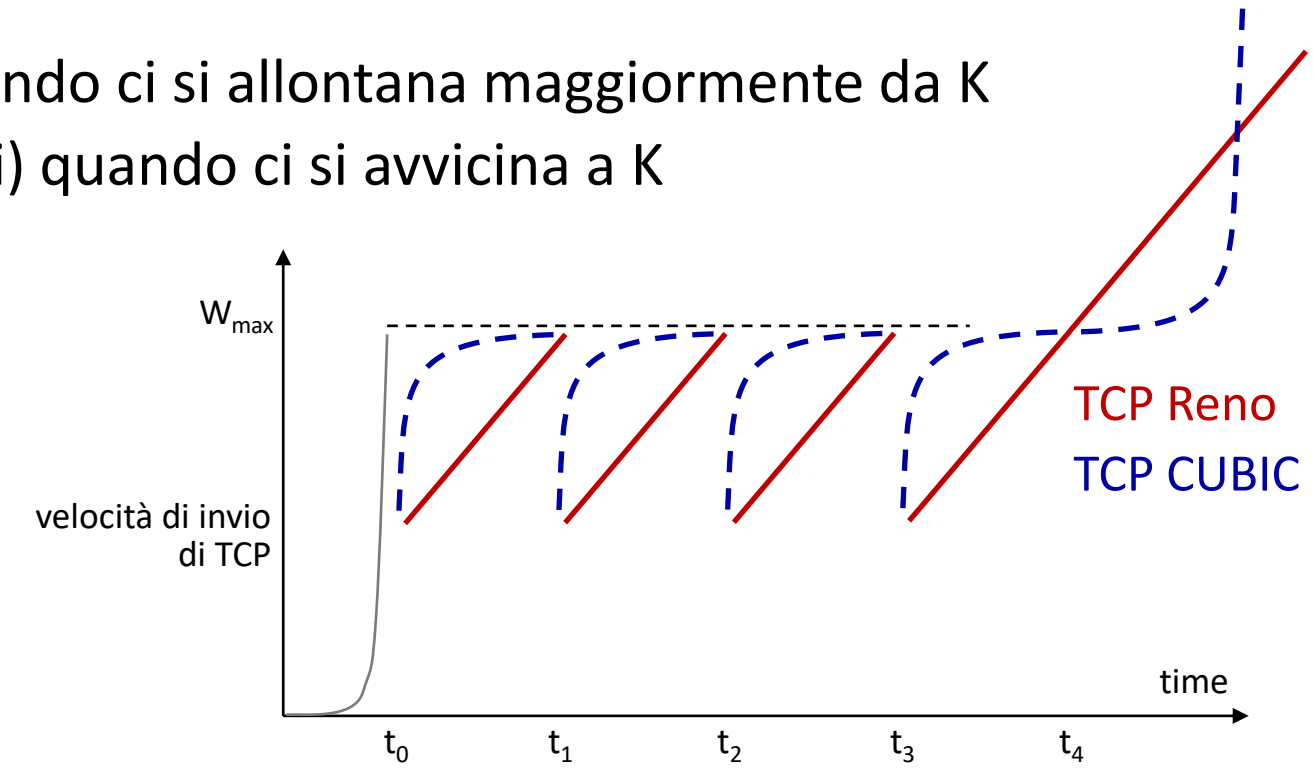
TCP CUBIC

- Esiste un modo migliore di AIMD per "sondare" la larghezza di banda utilizzabile?
- Intuizione:
  - W_{max}: la dimensione della finestra del controllo di congestione all'istante in cui viene rilevata la perdita
  - lo stato di congestione del collegamento bottleneck probabilmente (?) non è cambiato molto
  - dopo aver dimezzato la velocità/finestra in caso di perdita, inizialmente si sale verso W_{max} *più velocemente*, ma poi ci si avvicina a W_{max} *più lentamente*



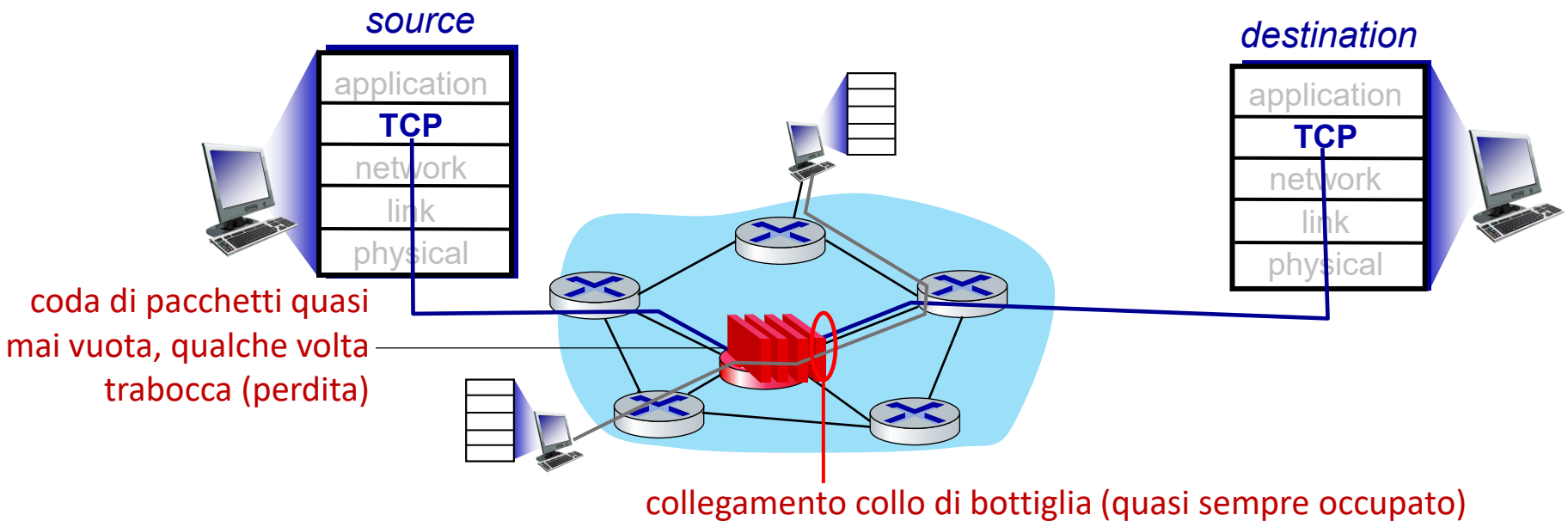
TCP CUBIC

- K: l'istante nel futuro in cui la finestra TCP raggiungerà nuovamente W_{max}
  - K è determinato da diversi parametri
- aumenta W come una funzione del *cubo* della distanza tra l'istante corrente e K
  - aumenti maggiori quando ci si allontana maggiormente da K
  - aumenti minori (cauti) quando ci si avvicina a K
- TCP CUBIC predefinito in Linux, il TCP più diffuso per i server Web più comuni



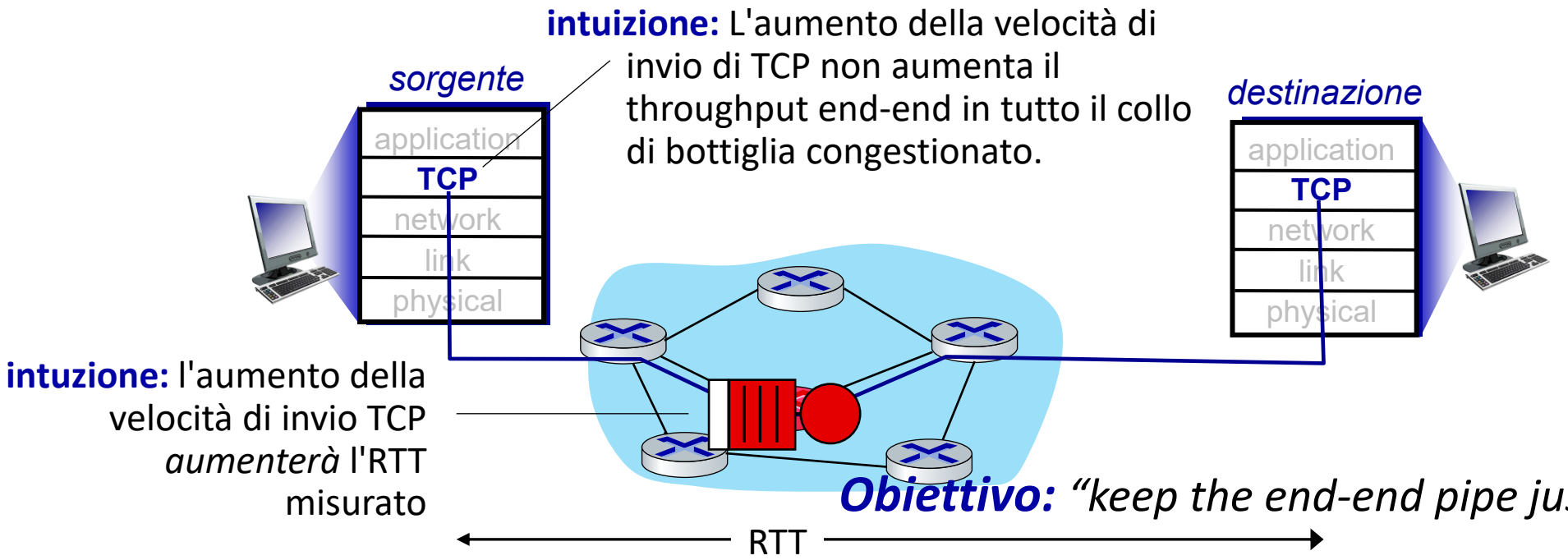
TCP e il collegamento "collo di bottiglia" congestionato

- TCP (classic, CUBIC) aumenta la velocità di invio di TCP finché non si verifica una perdita di pacchetti all'uscita di un router: il *collegamento "collo bottiglia" (bottleneck)*



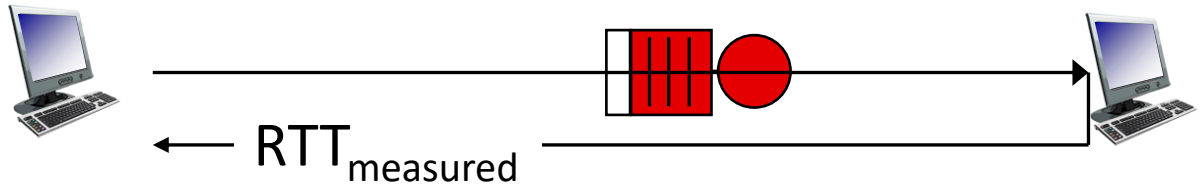
TCP e il collegamento "collo di bottiglia" congestionato

- TCP (classic, CUBIC) aumenta la velocità di invio di TCP finché non si verifica una perdita di pacchetti all'uscita di un router: il collegamento "collo bottiglia" (bottleneck)
- comprendere la congestione: utile concentrarsi sul collegamento con collo di bottiglia congestionato



Controllo di congestion basato sul ritardo

Mantenere la tubazione da mittente a destinatario "sufficientemente piena, ma non di più": mantenere il collegamento a collo di bottiglia impegnato nella trasmissione, ma evitare ritardi elevati/buffering.



$$\text{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{RTT_{measured}}$$

Approccio basato sul ritardo (TCP Vegas):

- RTT_{min} - RTT minimo osservato (percorso non congestionato)
- throughput non congestionato con finestra di congestione `cwnd` è cwnd/RTT_{min}

se il throughput misurato è "molto vicino" al throughput non congestionato
    aumentare linearmente il cwnd /* poiché il percorso non è congestionato */
altrimenti se il throughput misurato è "molto inferiore" al throughput non congestionato
    diminuire linearmente cwnd /* poiché il percorso è congestionato */

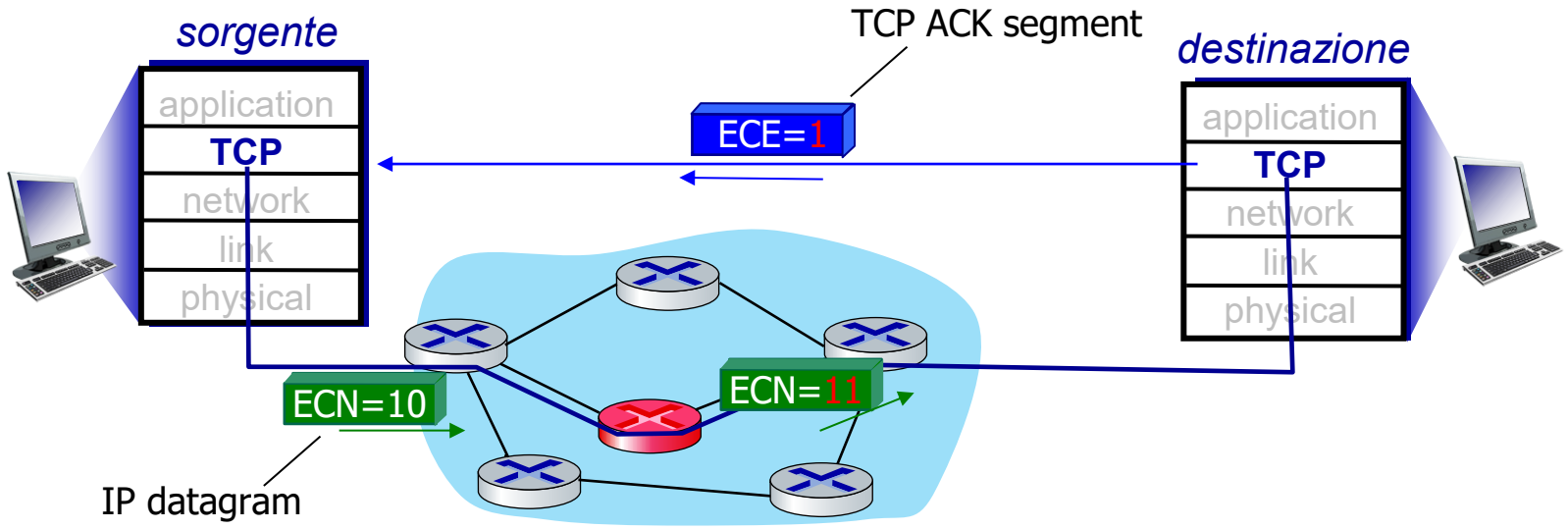
Controllo di congestion basato sul ritardo

- controllo di congestion senza indurre/forzare perdite
- massimizzare il throughput (“keeping the just pipe full… ”) ma mantenendo il ritardo basso (“…but not fuller”)
- un certo numero di TCP distribuiti adottano un approccio basato sui ritardi
  - BBR (Bottleneck Bandwidth and Round-trip propagation time) impiegato sulla rete dorsale (interna) di Google

Explicit congestion notification (ECN)

Le implementazioni di TCP spesso implementano un controllo della congestione *assistito dalla rete*:

- due bit (ECN) nell'intestazione IP (all'interno del campo ToS) impostati *da un router di rete* per indicare la congestione
  - *policy* per determinare la marcatura scelta dall'operatore di rete
- indicazione di congestione portata a destinazione
- la destinazione imposta il bit ECE sul segmento ACK per notificare al mittente la presenza di una congestione
- coinvolge sia l'IP (marcatura dei bit ToS dell'intestazione IP) che il TCP (marcatura dei bit CWR,ECE dell'intestazione TCP).

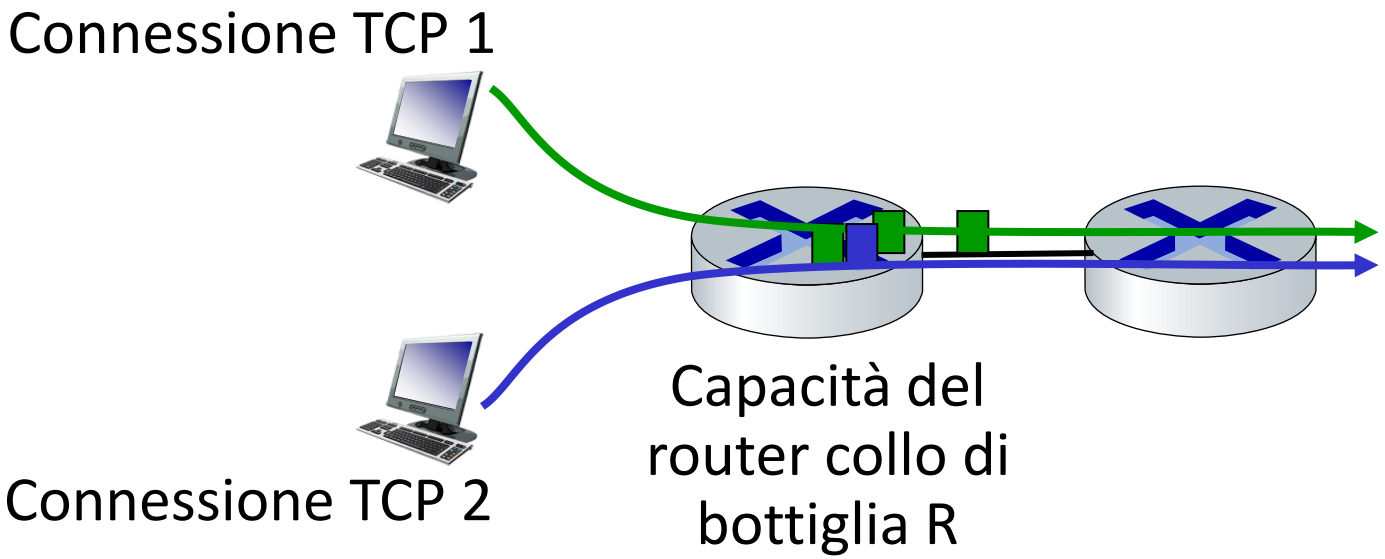


Explicit congestion notification (ECN)

- ECN viene negoziato in fase di instaurazione di una connessione TCP mediante opportune opzioni
- Il mittente imposta i bit ECN nell'intestazione IP per indicare che contengono i segmenti di una connessione in grado di gestire ECN

TCP fairness

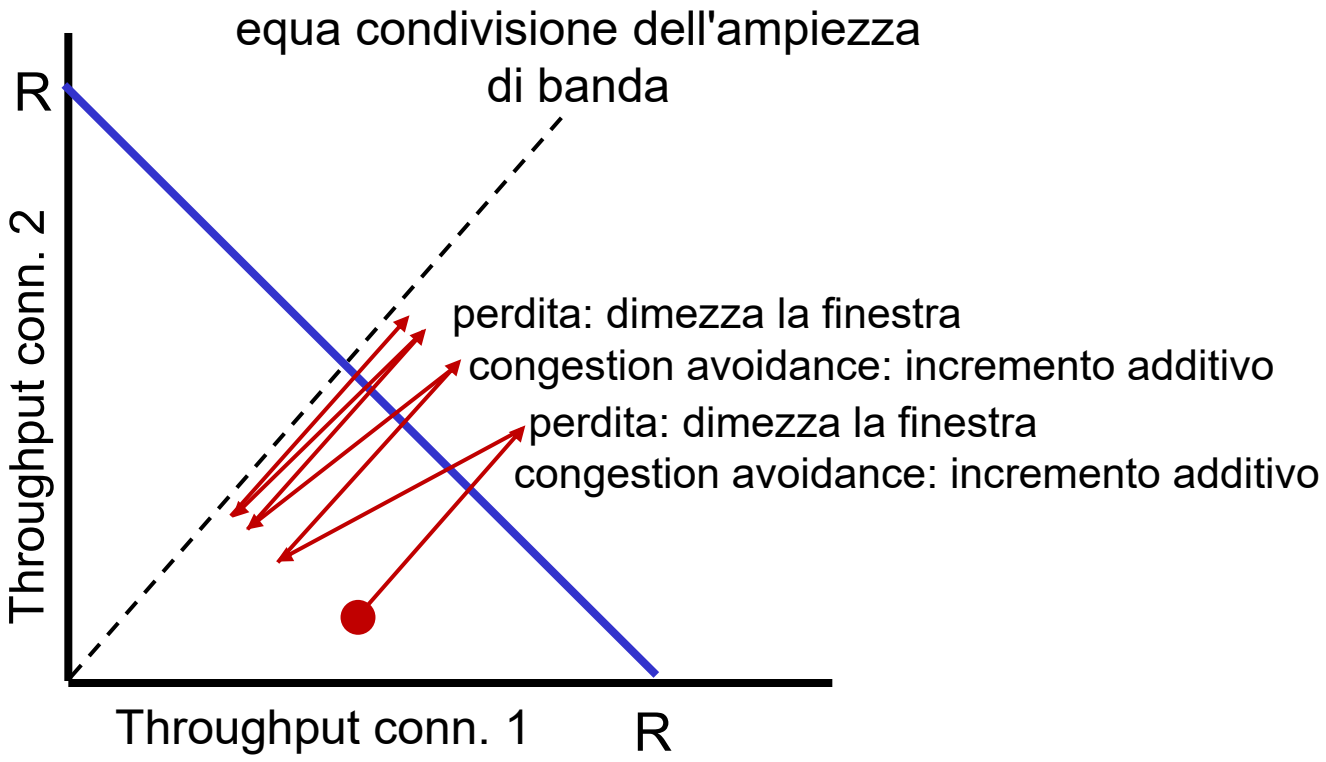
Obiettivo di equità: se *K* sessioni TCP condividono lo stesso collegamento a collo di bottiglia con larghezza di banda *R*, ciascuna dovrebbe avere una velocità media di *R/K*



D: TCP è Fair?

Esempio: due sessioni TCP in competizione:

- L'aumento additivo dà una pendenza di 1, quando il throughput aumenta
- la diminuzione moltiplicativa riduce il throughput in modo proporzionale



TCP è fair?

R: Sì, sotto assunzioni idealizzate:

- stesso RTT
- numero fisso di sessioni in congestion avoidance

Firness: tutte le app di rete devono essere "fair"?

Fairness e UDP

- Le app multimediali spesso non usano TCP
  - Non vogliono che la velocità sia ridotta dal controllo della congestione
- Usano invece UDP:
  - Inviano audio/video a velocità costante, tollerano la perdita di pacchetti
- Non esiste una "polizia di Internet" che controlli l'uso del controllo della congestione

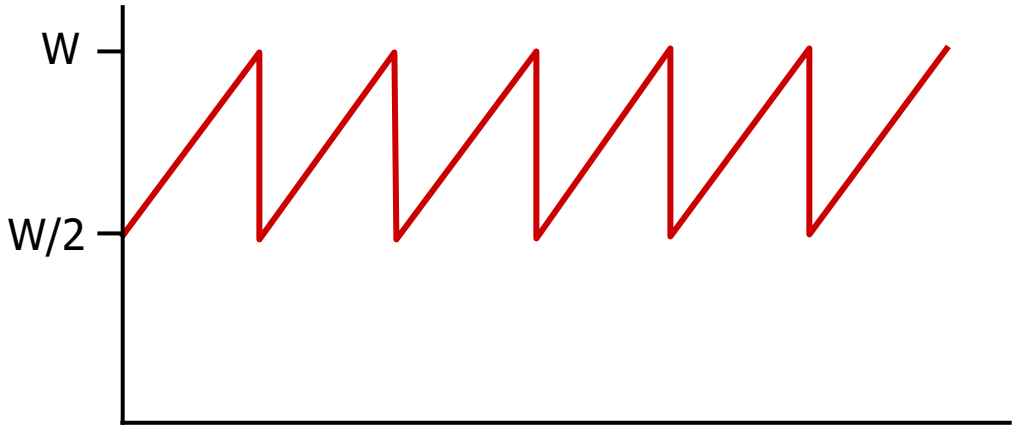
Fairness, connessioni TCP parallele

- L'applicazione può aprire *più* connessioni parallele tra due host
- I browser web lo fanno, ad esempio il link di velocità R con 9 connessioni esistenti:
  - nuova applicazione usa 1 connessione TCP, ottiene tasso R/10
  - nuova applicazione usa 11 connessioni TCP, ottiene tasso maggiore di R/2

Descrizione macroscopica del throughput di TCP

- Valore medio del throughput come funzione della dimensione della finestra e di RTT?
  - ignoriamo slow start, assumiamo che ci siano sempre dati da inviare
- W: dimensione della finestra (misurata in byte) quando si verifica una perdita
  - dimensione media della finestra (numero di byte "in volo") è ¾ W
  - throughput medio è 3/4W ogni RTT

$$\text{throughput TCP medio} = \frac{3}{4}\frac{W}{RTT} \text{ byte/s}$$



Livello di trasporto: tabella di marcia

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

Evoluzione della funzionalità del livello di trasporto

- TCP, UDP: i principali protocolli di trasporto per 40 anni
- Sono stati sviluppate diversi "varianti" (*flavor*)di TCP, per scenari specifici:

Scenario	Sfide
Long, fat pipes (trasferimenti di dati di grandi dimensioni)	Molti pacchetti "in volo"; la perdita interrompe la pipeline
Reti wireless	Perdita dovuta a collegamenti wireless rumorosi, mobilità; il TCP la tratta come perdita di congestione
Long-delay links	RTT estremamente elevato
Reti dei data center	Sensibili alla latenza
Background traffic flows	Flussi TCP a bassa priorità, "in background"

- Spostamento delle funzioni del livello di trasporto al livello di applicazione, in cima a UDP
  - HTTP/3: QUIC

TCP su “long, fat pipes”

- esempio: segmenti di 1500 byte, RTT di 100ms, vogliamo un throughput di 10 Gbps
- richiede W = 83˙333 segmenti in volo
- throughput in termini della probabilità di perdita dei pacchetti, L [Mathis 1997]:

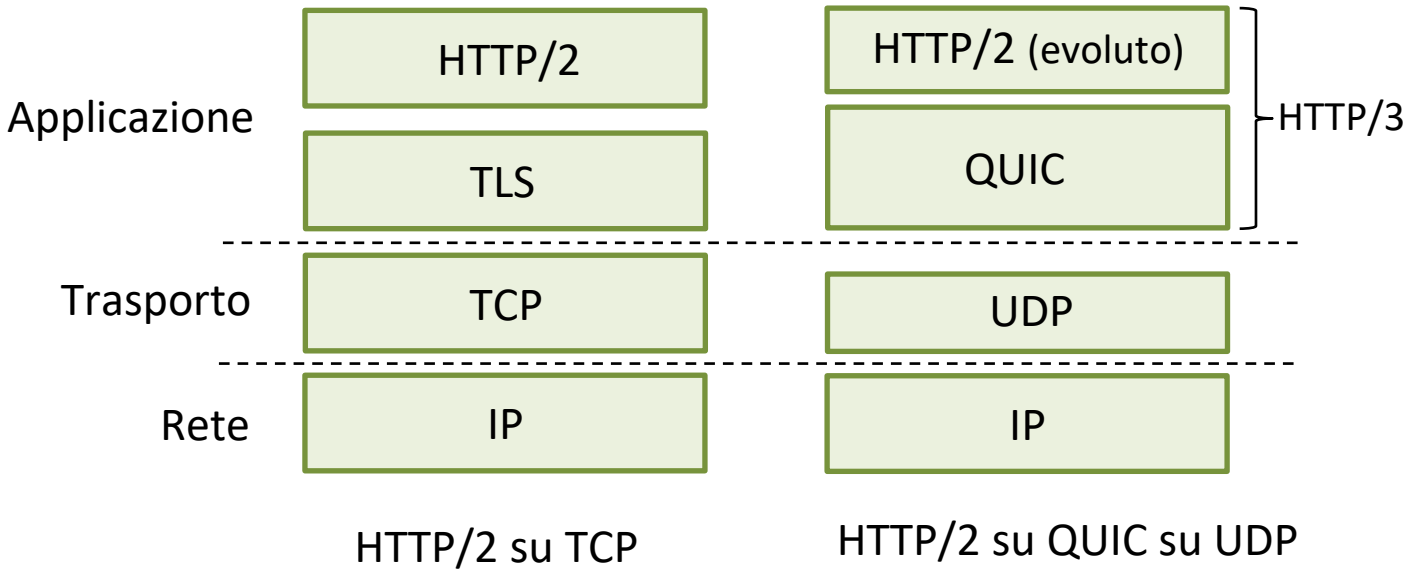
$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT}\sqrt{L}}$$

➜ per raggiungere un throughput di 10 Gbps, occorre un tasso di perdita $L = 2 \cdot 10^{-10}$ *– un tasso di perdita davvero piccolo!*

- versioni di TCP per scenari lungi, ad alta velocità

QUIC: Quick UDP Internet Connections

- protocollo di livello applicazione, sopra a UDP
  - aumenta le prestazioni di HTTP
  - impiegati in molti server e app di Google (Chrome, app mobile di YouTube)



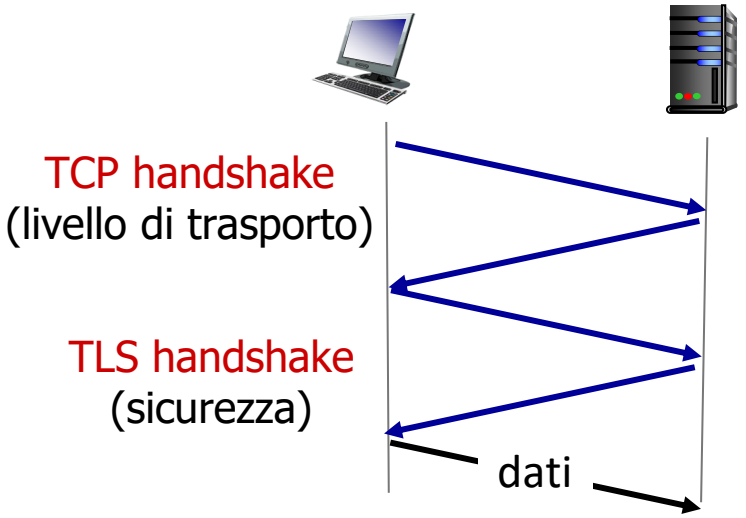
QUIC: Quick UDP Internet Connections

Adotta gli approcci che abbiamo studiato in queste lezioni per l'instaurazione della connessione, il controllo degli errori e il controllo della congestione

- **Controllo degli errori e della congestione:** “Readers familiar with TCP’s loss detection and congestion control will find algorithms here that parallel well-known TCP ones.” [from QUIC specification]
- **Instaurazione della connessione:** affidabilità, controllo della congestione, autenticazione, cifratura, stato stability in un solo RTT

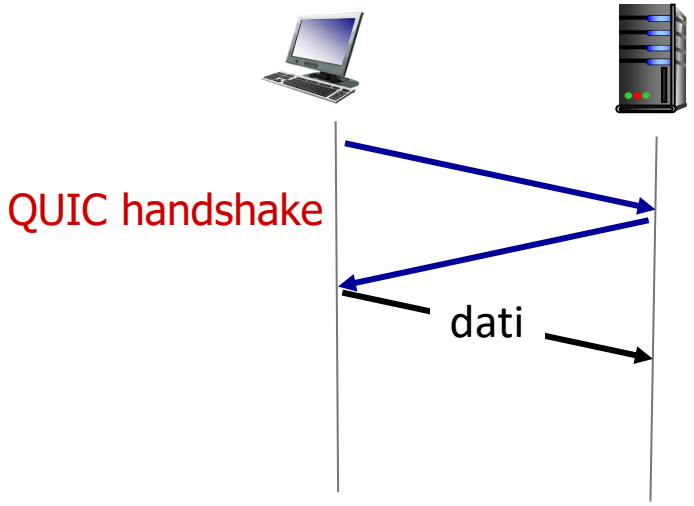
- multiplexing di molteplici "flussi" (*stream*) a livello di applicazione su una singola connessione QUIC
  - stato del trasferimento dati affidabile e della sicurezza separati
  - stato del controllo della congestione condiviso

QUIC: Instaurazione della connessione



TCP (stato per il trasferimento affidabile e per il controllo della congestione) +
TLS (stato per l'autenticazione autenticazione e per la cifratura)

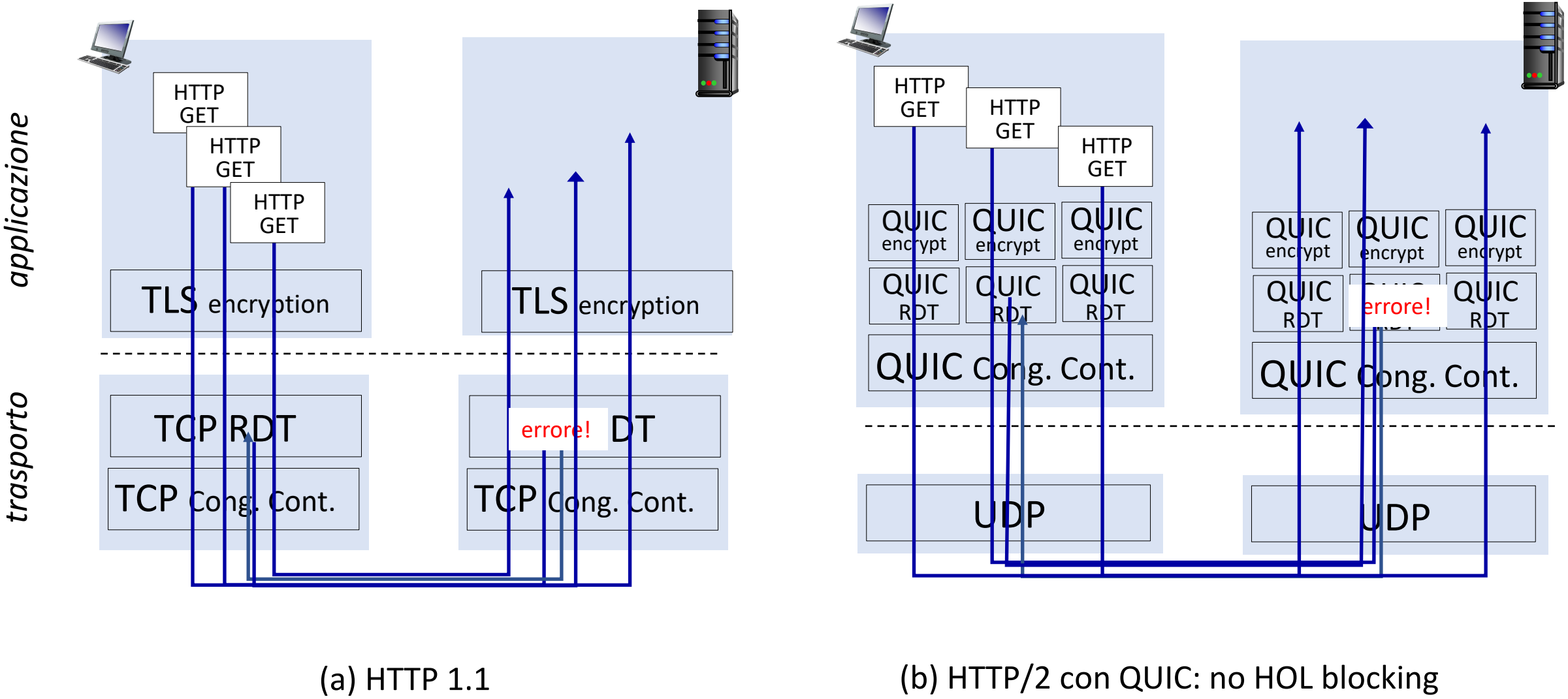
- 2 handshake in successione



QUIC: stato della affidabilità, controllo della congestion, autenticazione e cifratura

- 1 handshake

QUIC: flussi: parallelismo, no blocco HOL



(a) HTTP 1.1

(b) HTTP/2 con QUIC: no HOL blocking

Capitolo 3: riassunto

- Principi alla base dei servizi del livello di trasporto:
  - multiplexing, demultiplexing
  - trasferimento dati affidabile
  - controllo del flusso
  - controllo della congestione
- Installazione, implementazione in Internet
  - UDP
  - TCP

Prossimamente :

- abbandonare la “periferia” (livelli di applicazione e di trasporto)
- per addentrarci nel “nucleo” della rete
  - piano dei dati
  - piano di controllo