

Lezione 9 - Livello di Trasporto

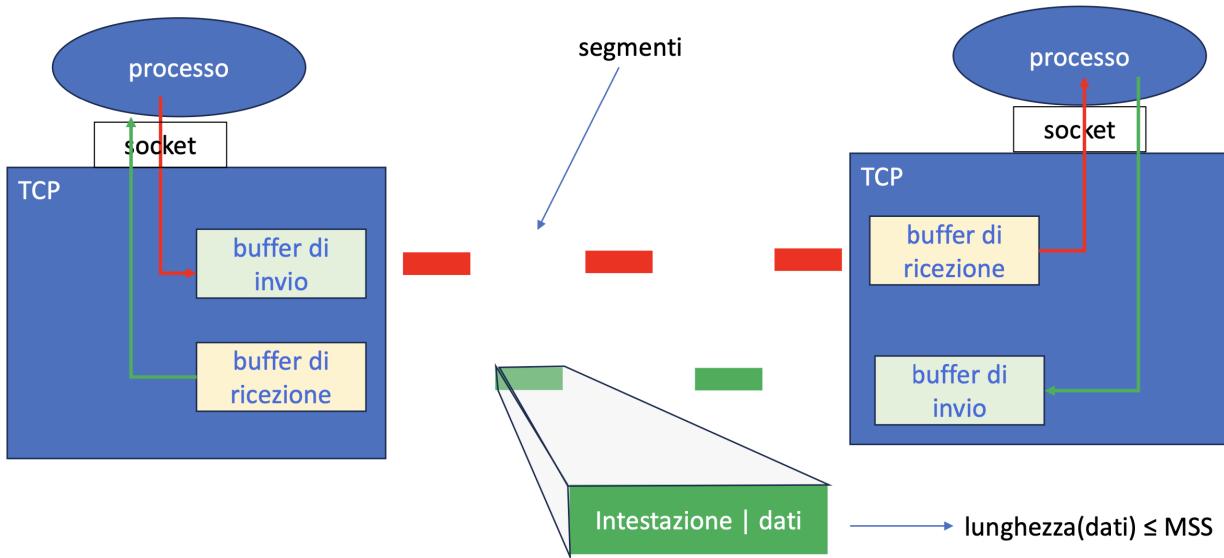
Trasporto orientato alla connessione : TCP

TCP : panoramica

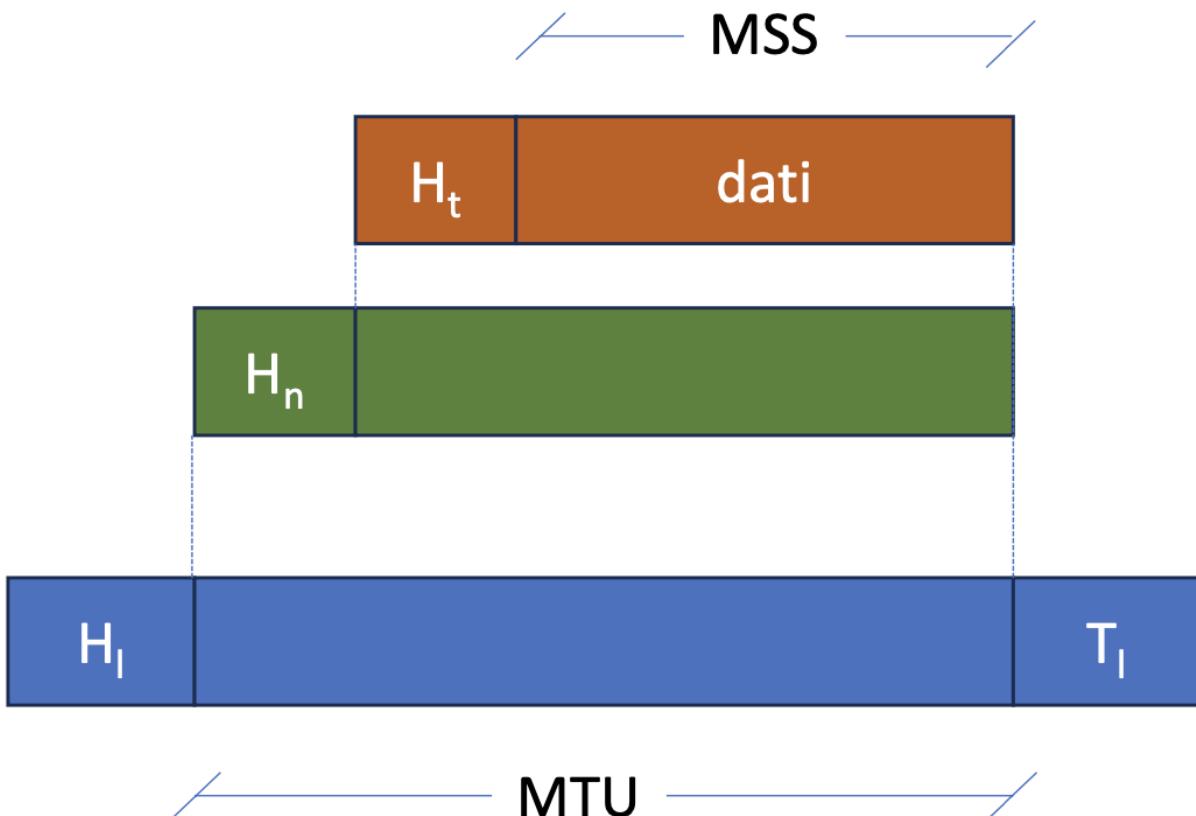
- Punto a punto (point-to-point) :
 - singolo mittente e singolo destinatario
- Flusso di byte affidabile (reliable), in sequenza (in order) :
 - nessun "confine ai messaggi"
- Full duplex data :
 - i dati possono fluire in direzioni opposte allo stesso istante nella stessa connessione
 - MSS: dimensione massima del segmento (maximum segment size); in realtà si riferisce ai dati applicativi nel segmento (intestazione esclusa)
- ACK cumulativi
- Pipeling :
 - Il controllo di flusso e della congestione definiscono la dimensione della finestra
- Orientato alla connessione :
 - L'handshaking (scambio di messaggi di controllo) inizializza lo stato del mittente e del destinatario prima di scambiare i dati
 - una connessione TCP non definisce un circuito end-to-end del tipo che caratterizza le reti a commutazione di circuito. Infatti , le connessioni TCP sono interamente implementate nei sistemi periferici , mentre i commutatori di pacchetto vedono solo i pacchetti di rete in transito.
- Flusso di controllo :
 - Il mittente non sovraccarica il destinatario

- Controllo della congestione :
 - Il mittente riduce la velocità di invio in funzione della congestione della rete.

TCP: Buffer di invio e ricezione TCP.



TCP : MMS



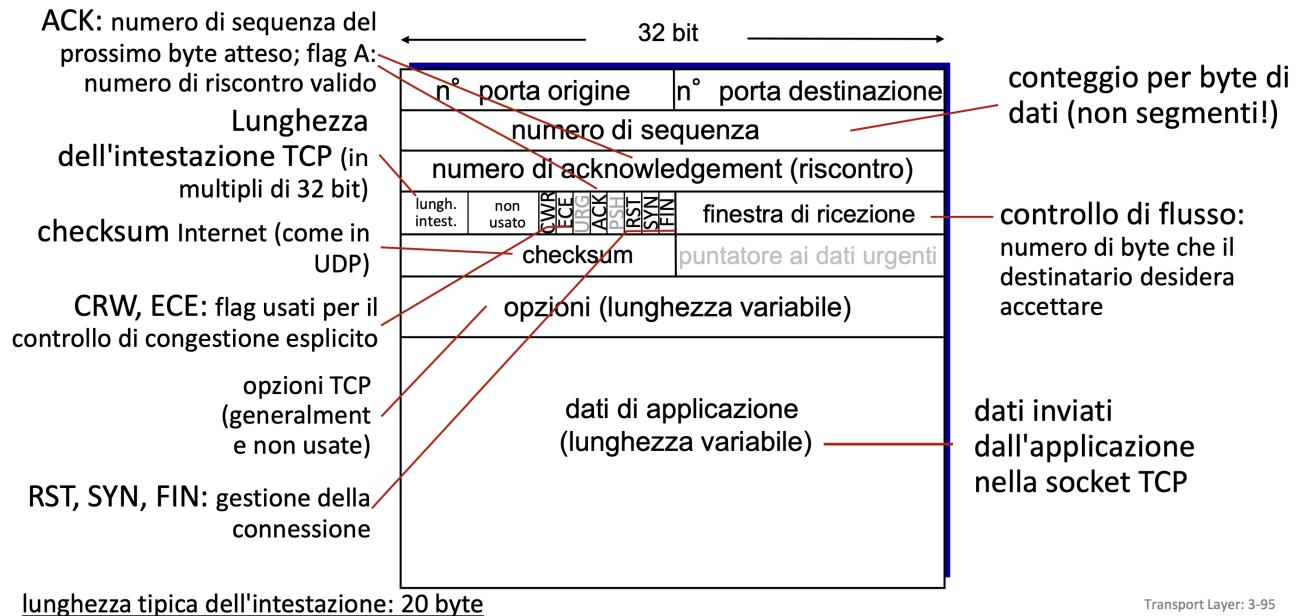
\swarrow **MSS** \searrow

La dimensione massima del segmento (MSS) TCP predefinita per IPv4 è di 536. Per IPv6 è 1220. Se un host desidera impostare MSS su un valore diverso da quello predefinito, la dimensione massima del segmento viene specificata come opzione TCP, inizialmente nel pacchetto TCP SYN durante l'handshake TCP .

- $MSS + \text{lunghezza}(H_t) + \text{lunghezza}(H_n) \leq MTU$
- Valori tipici :
 - Lunghezza (H_t) = 20 B
 - Lunghezza (H_n) = 20 B
 - $MTU_{ETHERNET} = 1500 \rightarrow MSS = 1460$
- Un host può determinare il MSS guardando la MTU del collegamento locale , ma ciò non offre garanzie circa altri collegamenti intermedi
- Può essere negoziato durante la connessione con l'opzione MSS
- Path MTU Discovery : permette di scoprire il valore più piccolo della MTU lungo il percorso da mittente a destinatario

Se un pacchetto IP eccede la MTU su un collegamento di uscita , il router dovrà frammentarlo (in IPv4) oppure scartarlo (in IPv6).

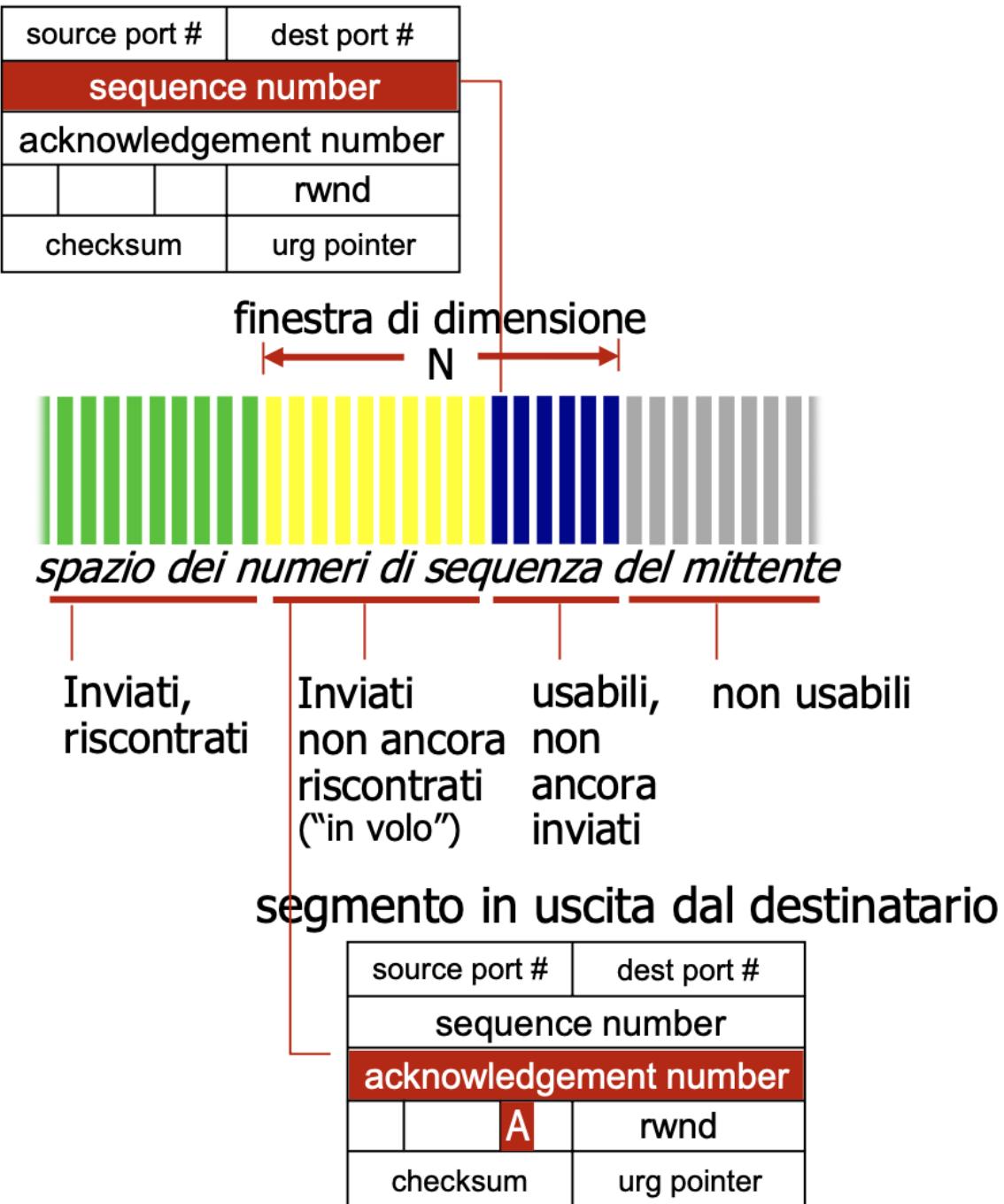
Struttura dei segmenti TCP



Transport Layer: 3-95

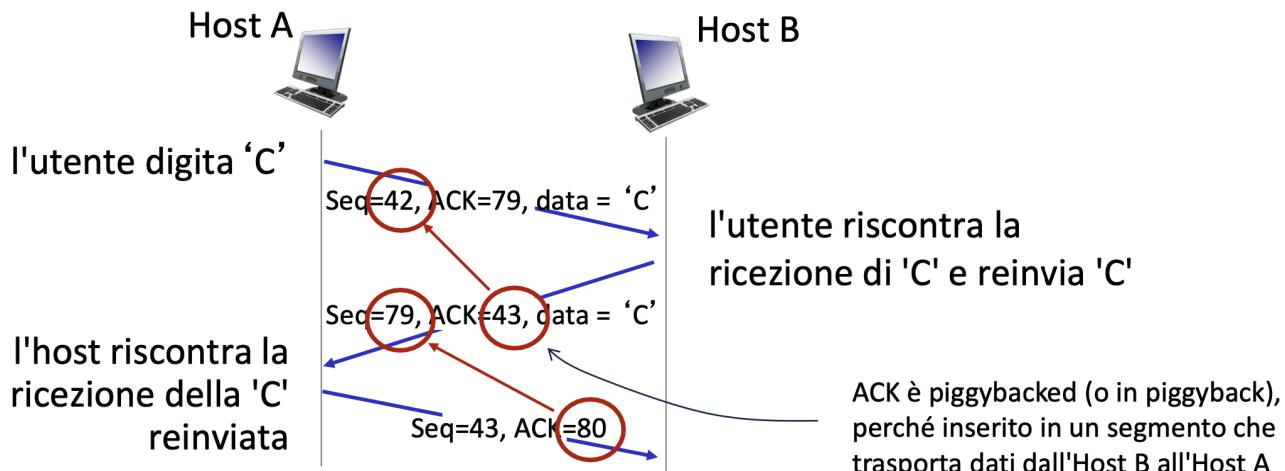
Numeri di sequenza e ACK di TCP

segmento in uscita dal mittente



- **Numero di sequenza :**
 - "numero" del primo byte nel segmento nel flusso di byte
- **ACK :**
 - Numero di sequenza del prossimo byte atteso dall'altro lato
 - ACK cumulativo
 - RFC 2018 : Acknowledgment Selettivo

- **D** : come gestisce il destinatario i segmenti fuori sequenza?
 - **R** : la specifica TCP non lo dice – dipende dall'implementatore



un semplice serve echo

TCP: tempo di andata e ritorno (round trip time) e timeout

D : come impostare il valore del timeout di TCP?

- più grande di RTT (tempo trascorso da quando si invia un segmento a quando se ne riceve l'acknowledgment), ma RTT non è noto varia!
- **Troppo piccolo** : timeout prematuro, ritrasmissioni non necessarie
- **Troppo grande** : reazione lenta alla perdita di segmenti

D : Come stimare RTT?

- **SampleRTT** : tempo misurato dalla trasmissione del segmento fino alla ricezione di ACK
 - Ignora le trasmissioni
- **SampleRTT** varia, quindi occorre una stima più "livellata" di RTT
 - *media di più misure recenti*, non semplicemente il calore corrente di SampleRTT
- Intervallo di timeout : **EstimatedRTT** più un "margin di sicurezza"

- grande variazione di **EstimatedRTT** : margine di sicurezza maggiore

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$
- **DevRTT** : EWMA della deviazione di **SampleRTT** da **EstimatedRTT** :

$$\text{DevRTT} = (1 - \beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

Mittente TCP (semplificato)

evento: ricevuti dati dall'applicazione

- crea un segmento con il numero di sequenza
- il numero di sequenza è il numero del primo byte del segmento nel flusso di byte
- avvia il timer , se non già in funzione
 - pensare al timer come se fosse associato al più vecchio segmento non ancora riscontrato
 - intervallo di scadenza : **TimeOutInterval**

evento: timeout

- ritrasmette il segmento che ha causato il timeout (cioè il segmento in attesa di ACK con il più piccolo numero di sequenza)
- riavvia il timer

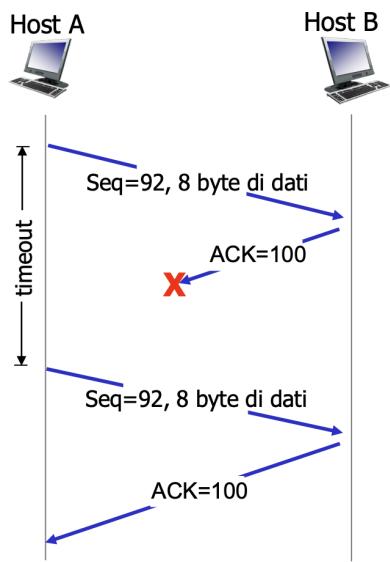
evento: ACK ricevuto

- se riscontra segmenti precedentemente non riscontrati (ACK y dove $y > \text{SendBase}$)
 - aggiorna ciò che si sa essere stato riscontrato ($\text{SendBase} = y$)
 - avvia il timer se ci sono altri segmenti ancora non riscontrati

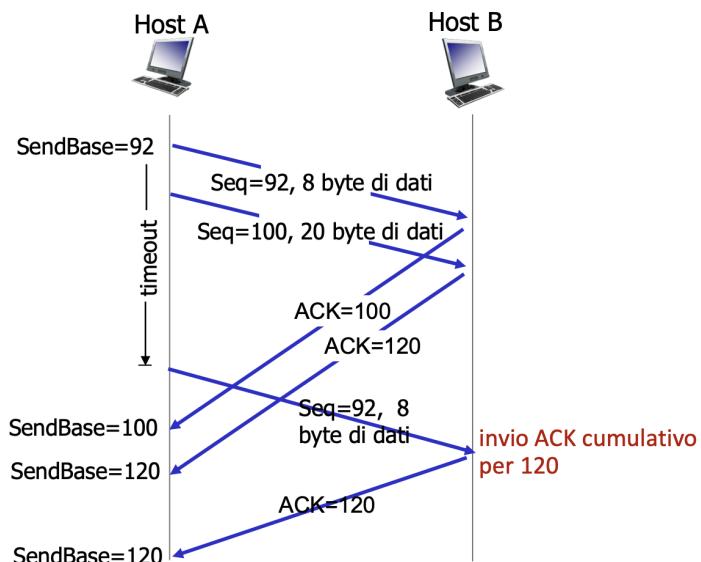
Ricevente TCP : Generazione degli ACK

Evento presso il destinatario	Azione del ricevente
Arrivo ordinato di segmento con numero di sequenza atteso. Tutti i dati fino al numero di sequenza attivo sono già stati riscontrati.	ACK ritardato. Attende fino a 500 millisecondi per l'arrivo ordinato di un altro segmento. Se in questo intervallo non arriva il successivo segmento, invia un ACK.
Arrivo ordinato di segmento con numero di sequenza atteso. Un altro segmento ordinato è in attesa di trasmissione dell'ACK.	Invia immediatamente un singolo ACK cumulativo, riscontrando entrambi i segmenti ordinati.
Arrivo non ordinato di segmento con numero di sequenza superiore a quello atteso. Viene rilevato un buco.	Invia immediatamente un ACK duplicato, indicando il numero di sequenza del prossimo byte atteso (che è l'estremità inferiore del buco)
Arrivo di segmento che colma parzialmente o completamente il buco nei dati ricevuti.	Invia immediatamente un ACK, ammesso che il segmento cominci all'estremità inferiore del buco

TCP : Scenari di Ritrasmissione



scenario con ACK perso



timeout prematuro

Host A



Host B



Seq=92, 8 byte di dati

Seq=100, 20 byte di dati

ACK=100



ACK=120

Seq=120, 15 byte di dati

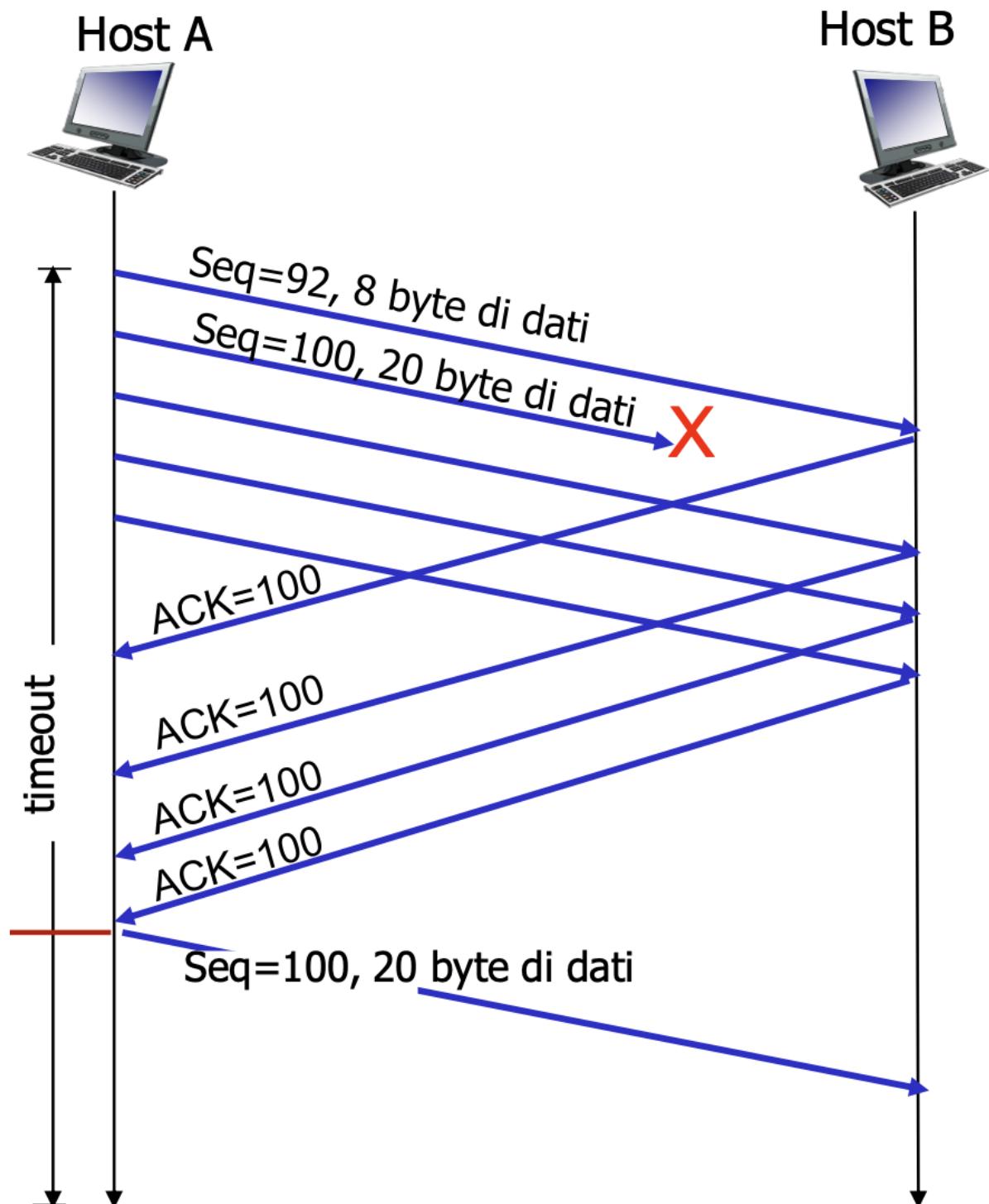
I'ACK cumulative copre
... a ...

I'ACK precedente perso

Ritrasmissione rapida

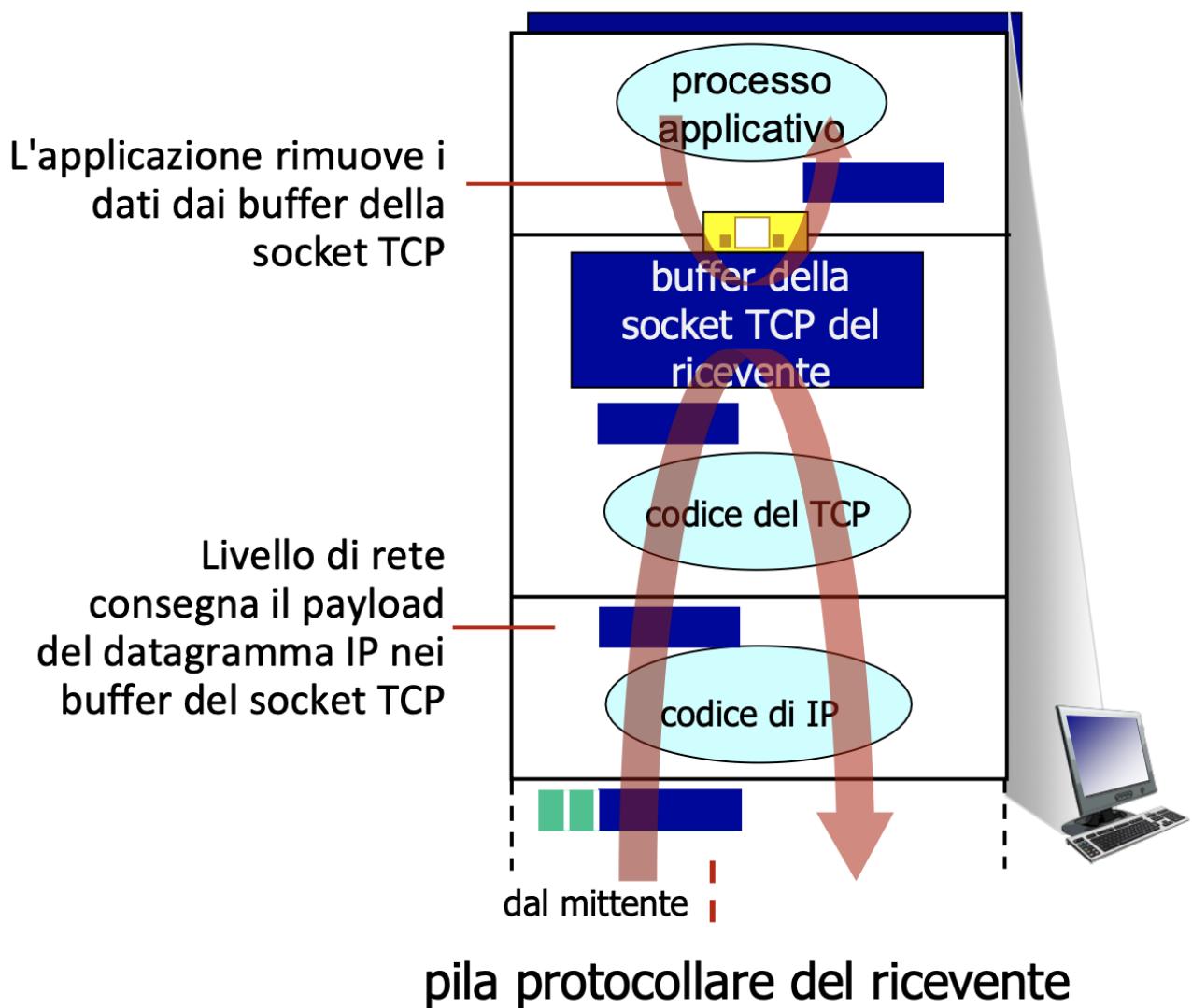
Se il mittente riceve 3 ACK addizionali per gli stessi dati (“3 ACK duplicati”), rispedisce il segmento non riscontrato con il più piccolo numero di sequenza

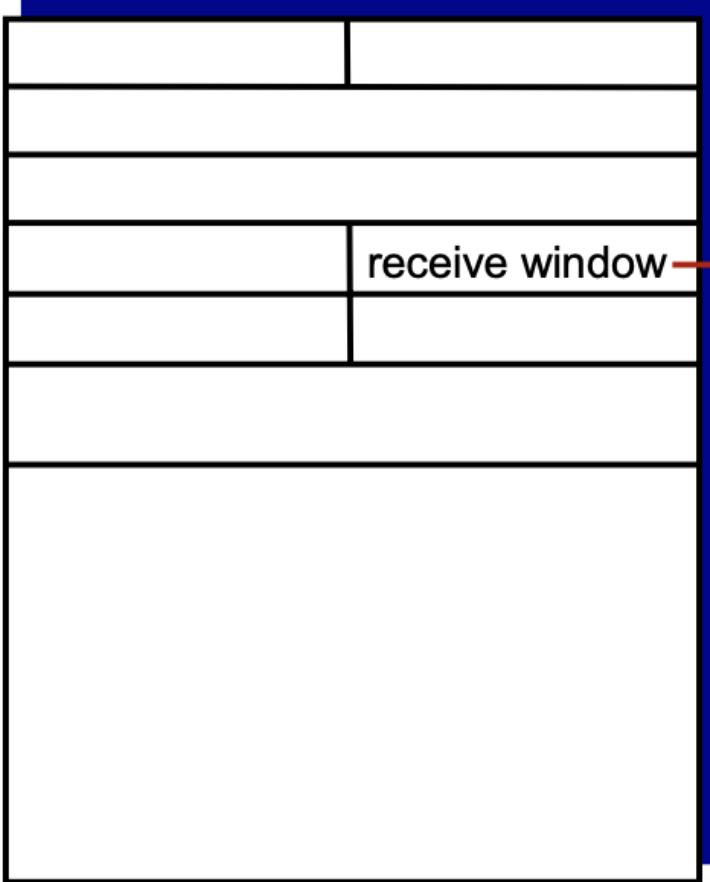
- è probabile che il segmento non riscontrato sia stato perso, quindi non aspettare il timeout.



TCP : controllo di flusso

D : Cosa succede se il livello di rete fornisce i dati più velocemente di quanto il livello applicativo rimuova i dati dai buffer delle socket?



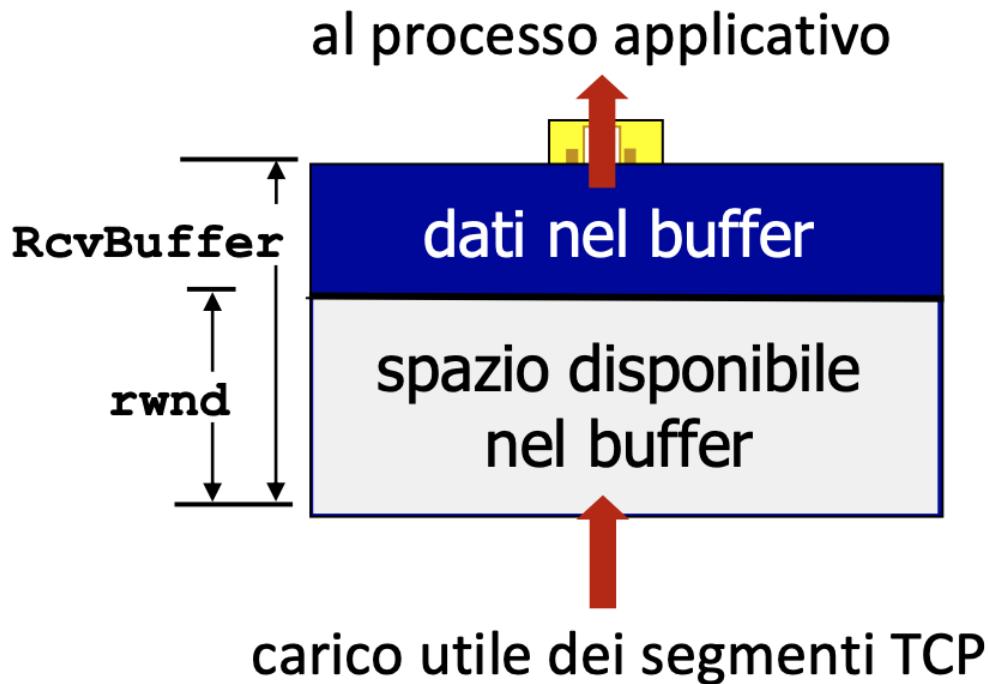


controllo di flusso: numero di byte che il ricevente è disposto ad accettare

Controllo di flusso : Il destinatario controlla il mittente, cosicché il mittente non ecceda il buffer del destinatario, inviando troppo e troppo velocemente

- Il mittente comunica lo spazio disponibile nel buffer nel campo **rwnd** (*receive window*, finestra di ricezione) nell'intestazione TCP
 - **RcvBuffer** dimensione impostata attraverso opzioni della socket (valore predefinito è tipicamente 4096 byte)
 - molti sistemi operativi regolano automaticamente **RcvBuffer**
- Il mittente limita i dati non riscontrati **arwnd**
 - garantisce che il buffer di ricezione non vada in overflow

$$\text{rwnd} = \text{RcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead})$$



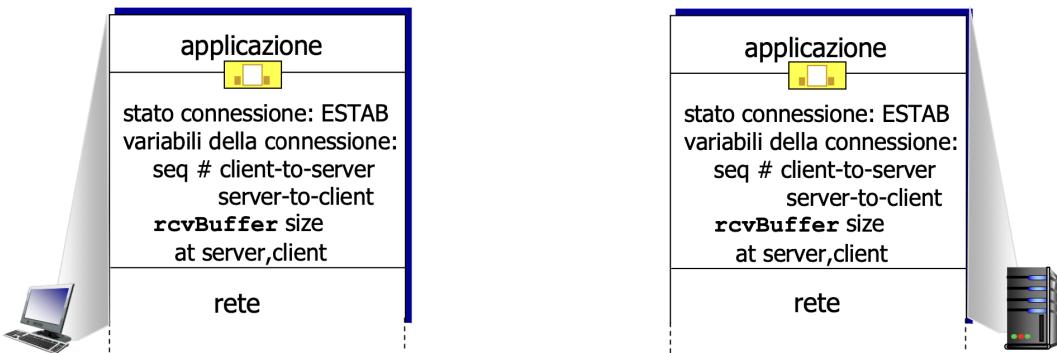
buffering dal lato del ricevente TCP

Gestione della connessione TCP

Prima di scambiare i dati, il mittente e il destinatario si "stringono la mano" ("handshake"):

- accettano di stabilire una connessione (ognuno sa che l'altro è disposto a stabilire una connessione)
- concordare i parametri di connessione (ad esempio, i numeri di

sequenza iniziali e receive buffer)

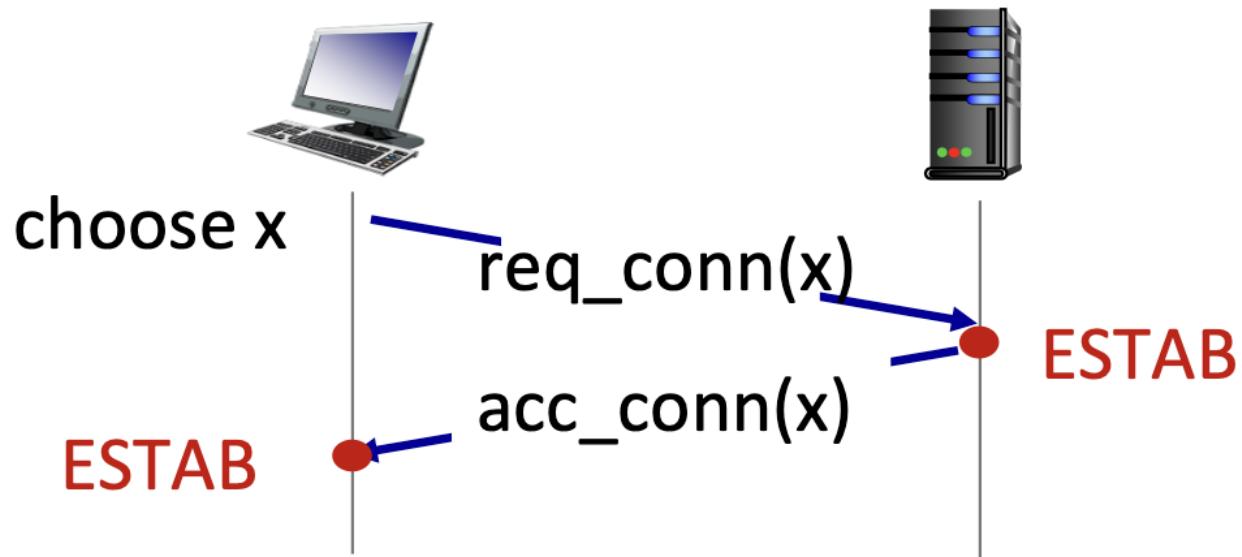
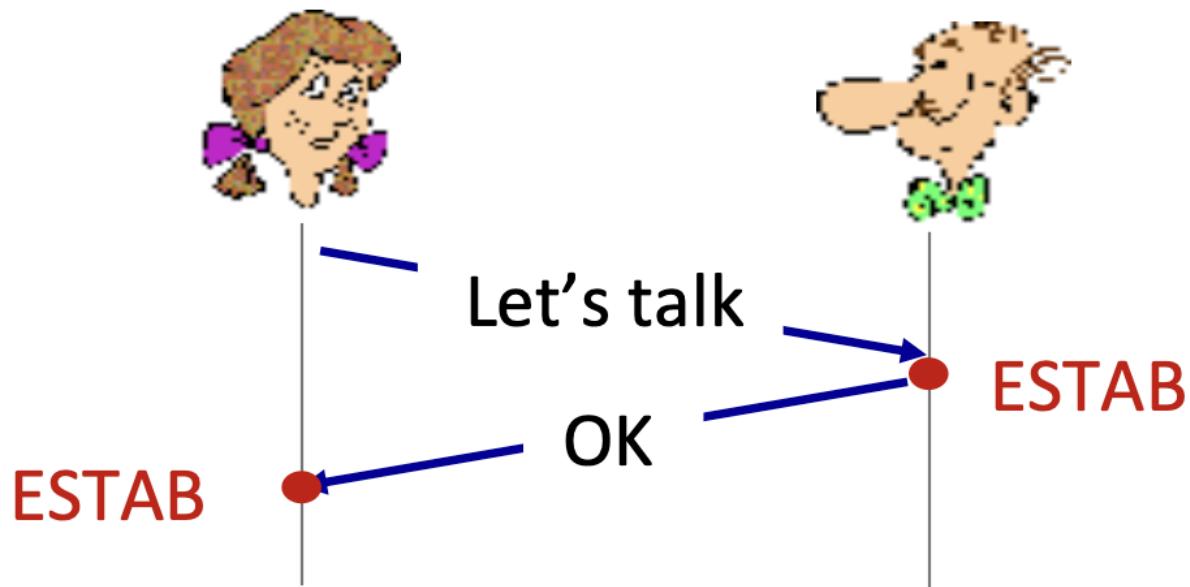


```
Socket clientSocket =  
    newSocket("hostname", "port number");
```

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

Accettare di stabilire una connessione

handshake a 2 vie:

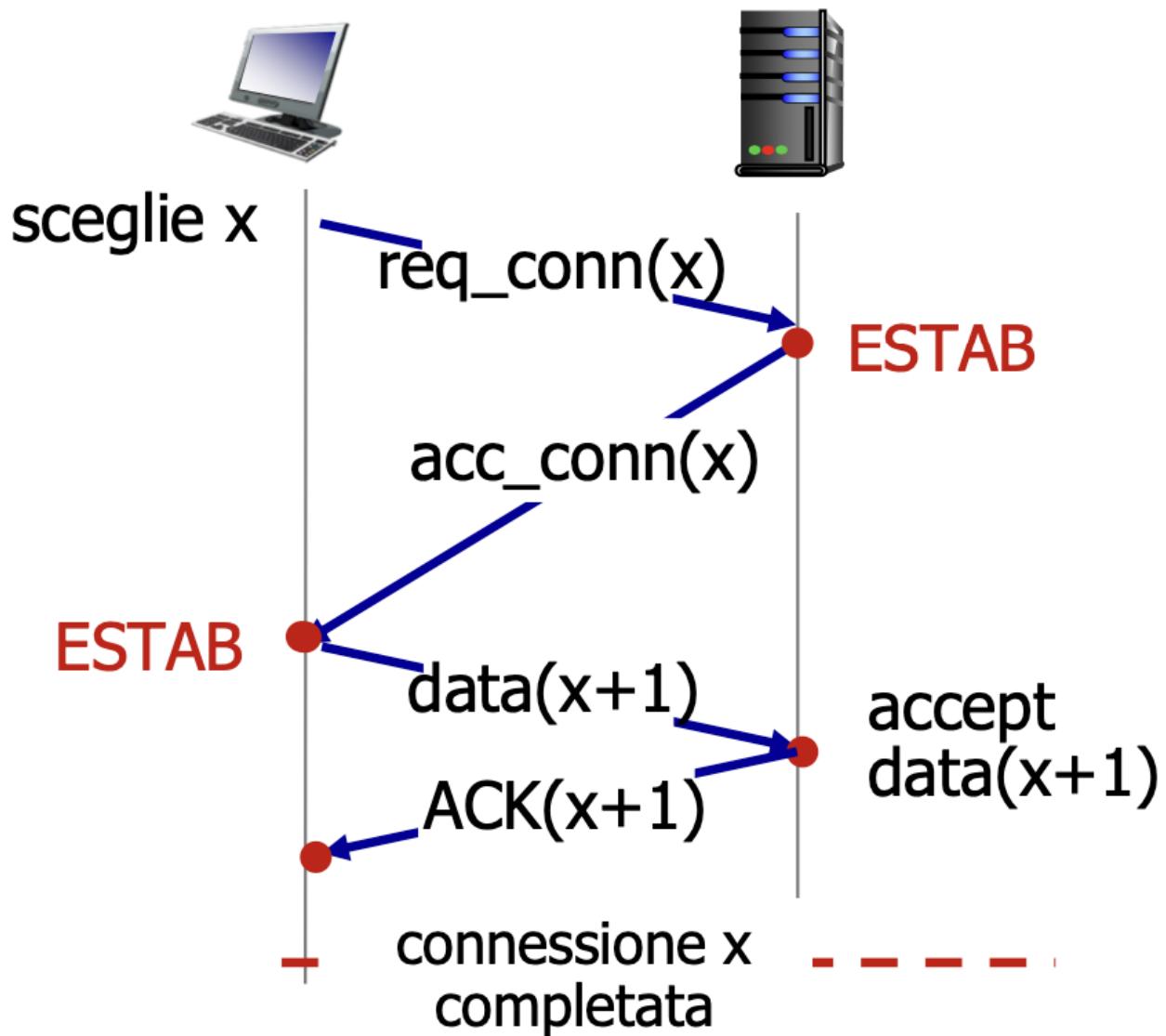


D : l'handshake a 2 vie funzionerebbe sempre?

- ritardi variabili
- messaggi ritrasmessi (ad esempio, $\text{req_conn}(x)$) a causa della perdita di messaggi
- riordino dei messaggi

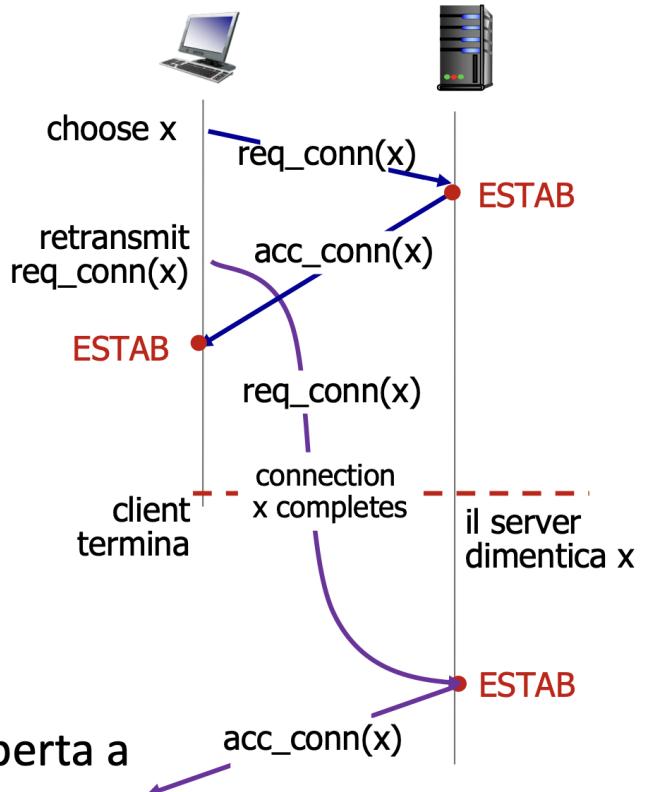
- impossibilità di "vedere" l'altro lato

Scenari per l'handshake

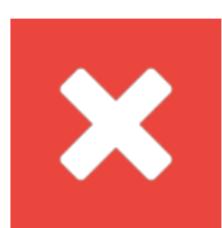
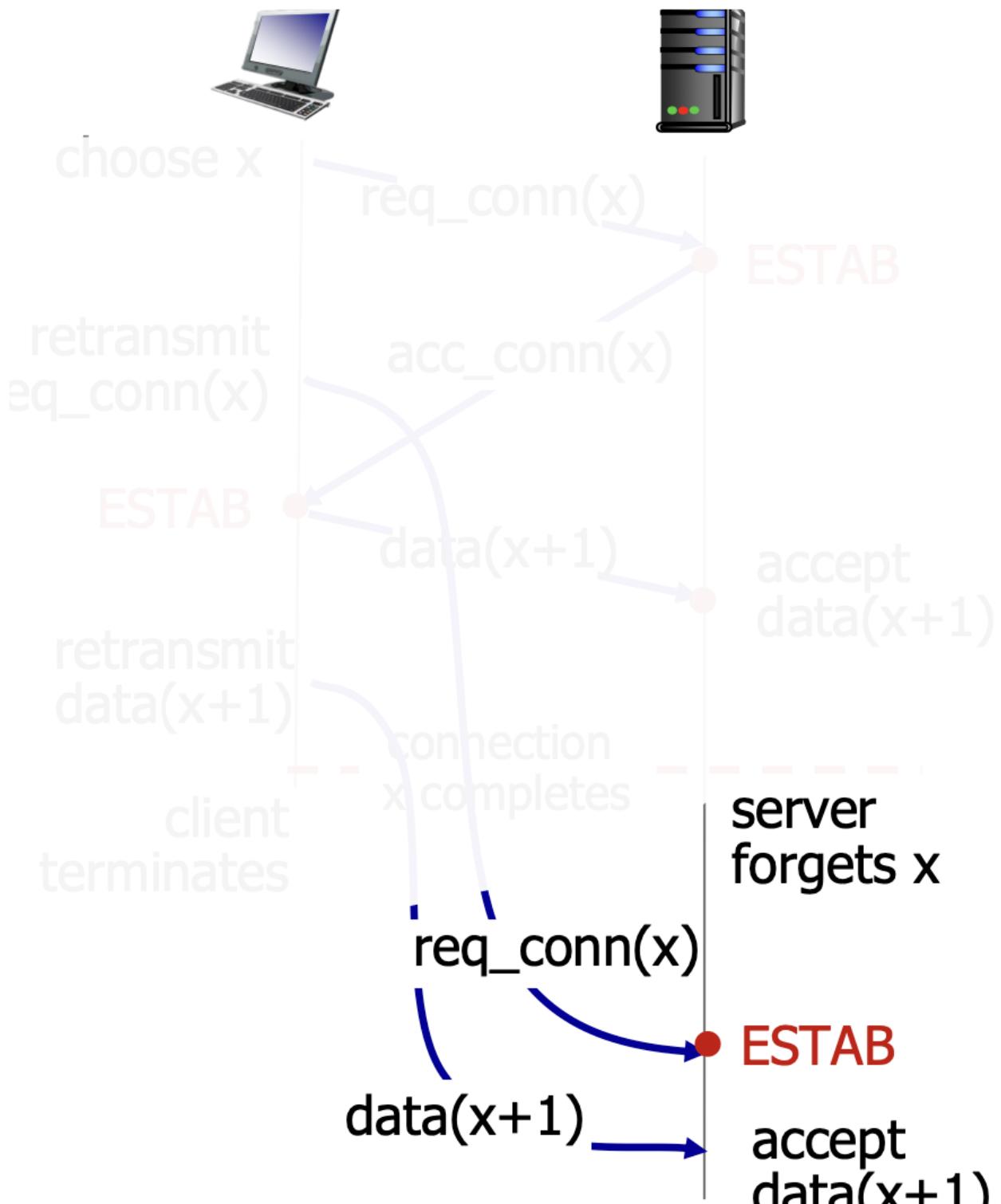


Nessun problema!

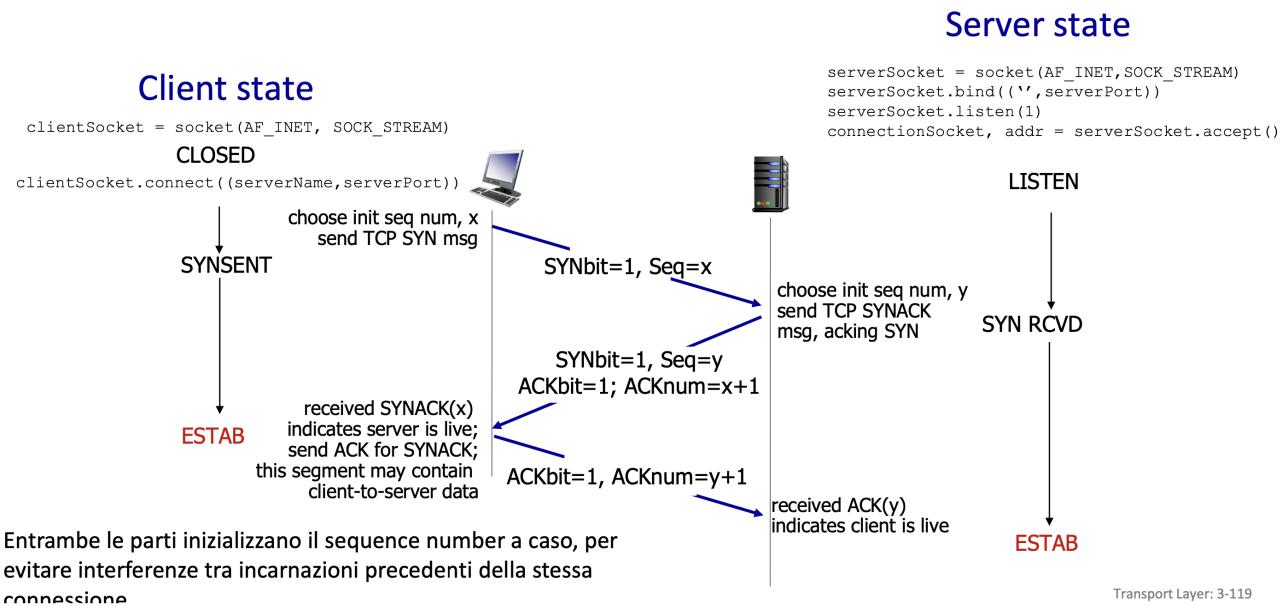




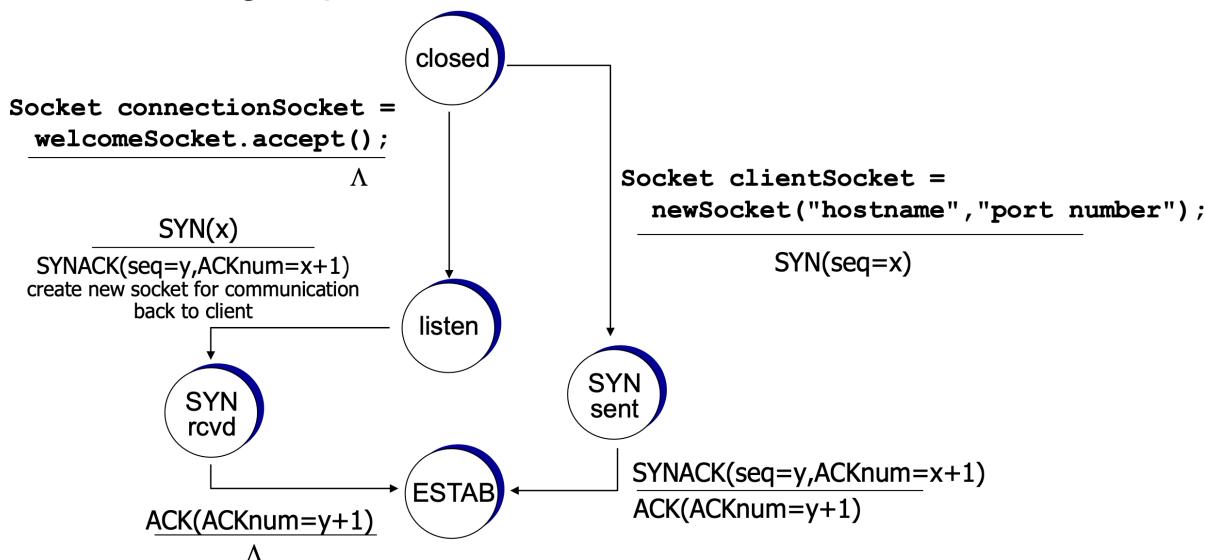
Problema: connessione aperta a metà! (nessun client)



**Problem: dati
duplicate accettati!**



- se un host riceve una richiesta di connessione per una porta su cui nessun socket è in ascolto, l'host invia al mittente un segmento con bit RST uguale a 1
- la ricezione di un segmento TCP RST informa un potenziale aggressore che la porta incriminata non è usata da alcun processo e che il segmento destinato a quella porta non è stato bloccato da un firewall lungo il percorso



Attacco SYN FLOOD

- L'aggressore invia un segmento TCP SYN ad un server con un indirizzo IP fasullo
- Il server risponde alloca e inizializza le variabili e i buffer della connessione e risponde inviando un segmento TCP SYNACK alla porta e all'indirizzo IP (fasullo) di origine e transita nello stato SYN_RCVD
- La rete tenta di consegnare il segmento TCP SYNACK all'indirizzo IP fasullo , possibilmente raggiungendo un altro host che non c'entra nulla e che non risponderà
- Il server , nello strato SYN_RCVD , non ricevendo un ACK , eventualmente (dopo un minuto o più) rilascerà tutte le risorse associate a quella connessione mezza aperta
- Nel frattempo , però , l'aggressore è riuscito a consumare delle risorse del server. Inviando numerosi segmenti SYN ad un server obiettivo , un aggressore può montare un attacco DoS.

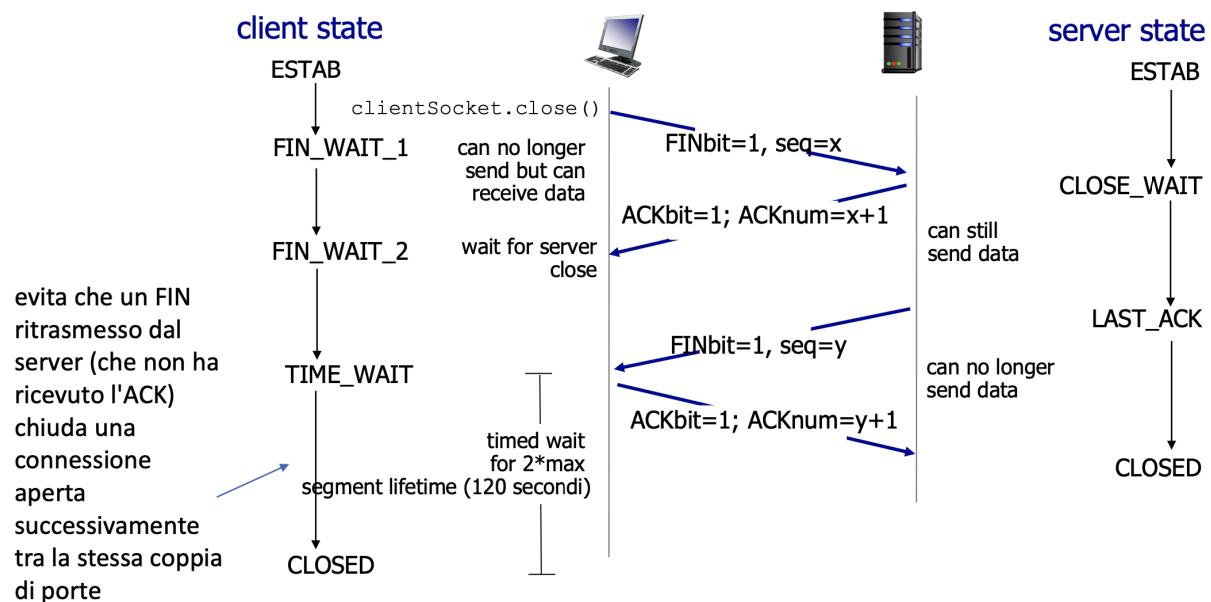
Attacco SYN FLOOD : contromisure "SYN cookie"

- Alla ricezione del segmento SYN iniziale, il server:
 - Calcola l'*hash* degli indirizzi IP e numeri di porta di origine e di destinazione e di una chiave segreta , producendo un cookie $\text{cookie}=\text{hash}(\text{IP sorgente}, \text{IP destinazione}, \text{porta sorgente}, \text{porta destinazione})$
 - usa il cookie con numero di sequenza iniziale da inserire dentro al segmento SYNACK
 - non alloca alcuna risorsa né memorizza il cookie
- Se il segmento SYN ricevuto in precedenza era legittimo, il client alla ricezione del segmento SYNACK risponde con un segmento ACK , che usa come numero di Acknowledgment cookie +1
- Alla ricezione di un segmento ACK, il server può capire che è legato al SYN precedente perché gli basta calcolare Acknowledgment - 1 per ottenere il cookie. Quindi , rieseguire il calcolo del cookie per

vedere se ottiene lo stesso valore. Si noti che l'inclusione di una chiave segreta impedisce a un aggressore di creare un cookie valido.

Chiusura di una connessione TCP

- client e server chiudono ciascuno il proprio lato della connessione
 - inviano il segmento TCP con il bit FIN = 1
- rispondere al FIN ricevuto con un ACK
 - alla ricezione del FIN, l'ACK può essere combinato con il proprio FIN
- è possibile gestire scambi FIN simultanei



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 problemi più importanti del networking

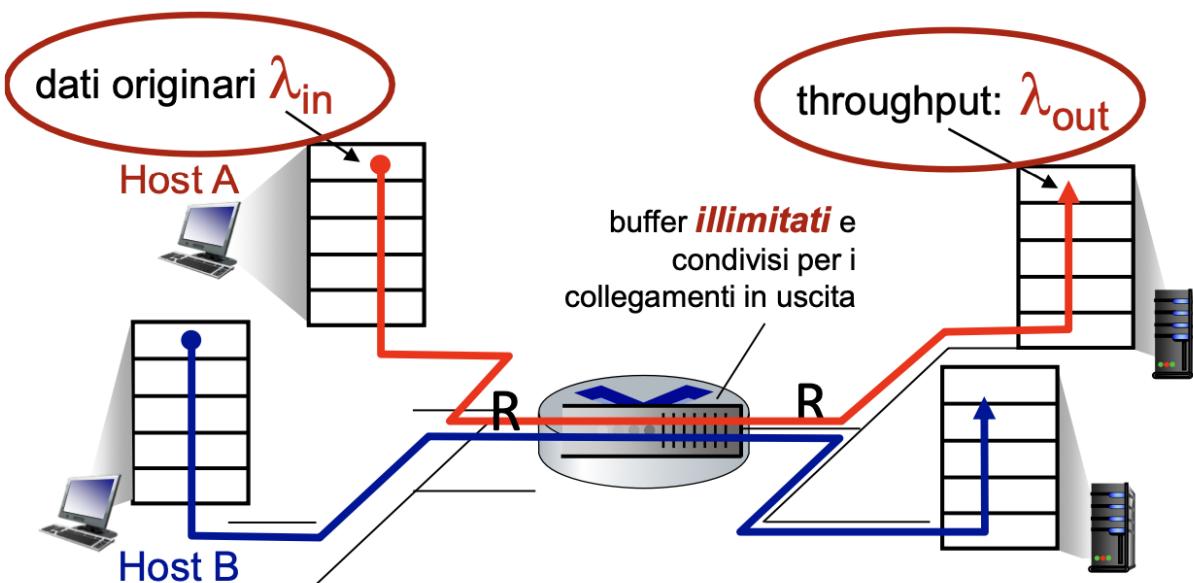
controllo della congestione : troppi mittenti , che trasmettono troppo velocemente

controllo di flusso : un mittente troppo veloce per il destinatario

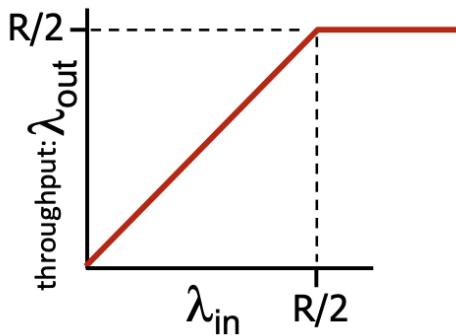
Cause/Costi della congestione : scenario 1

Caso più semplice :

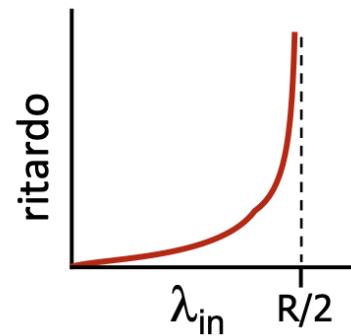
- un router con buffer illimitati
- capacità dei collegamenti di ingresso e uscita: R
- due flussi
- nessuna ritrasmissione



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



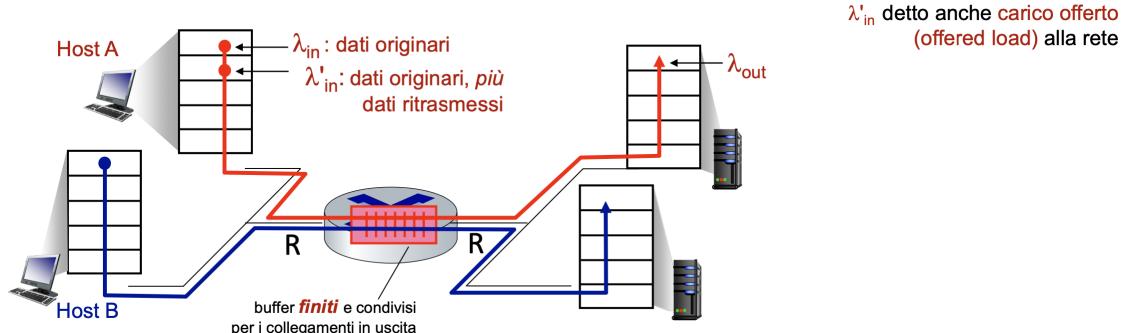
throughput massimo per connessione: $R/2$



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

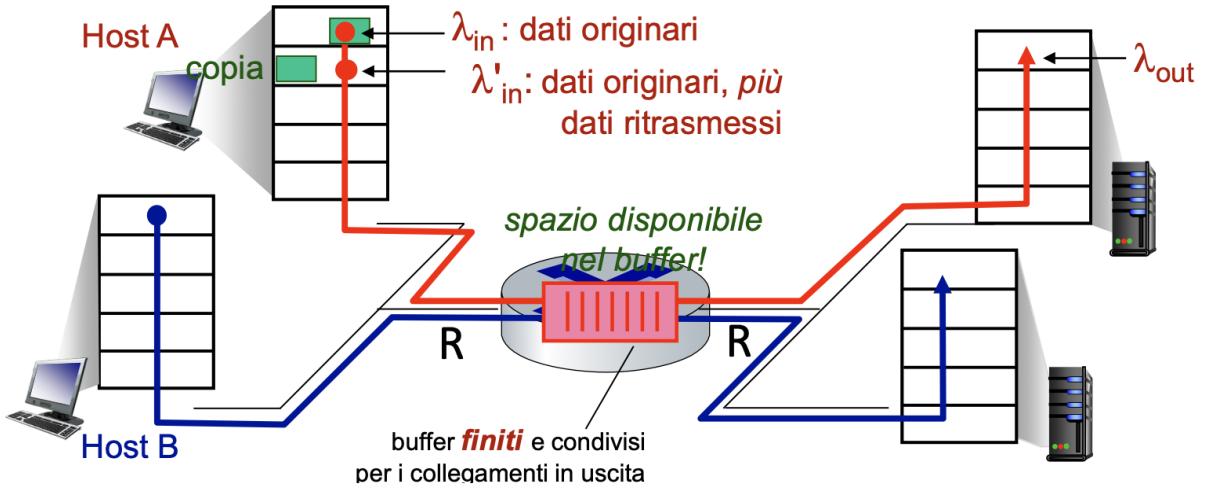
Cause/Costi della congestione : scenario 2

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



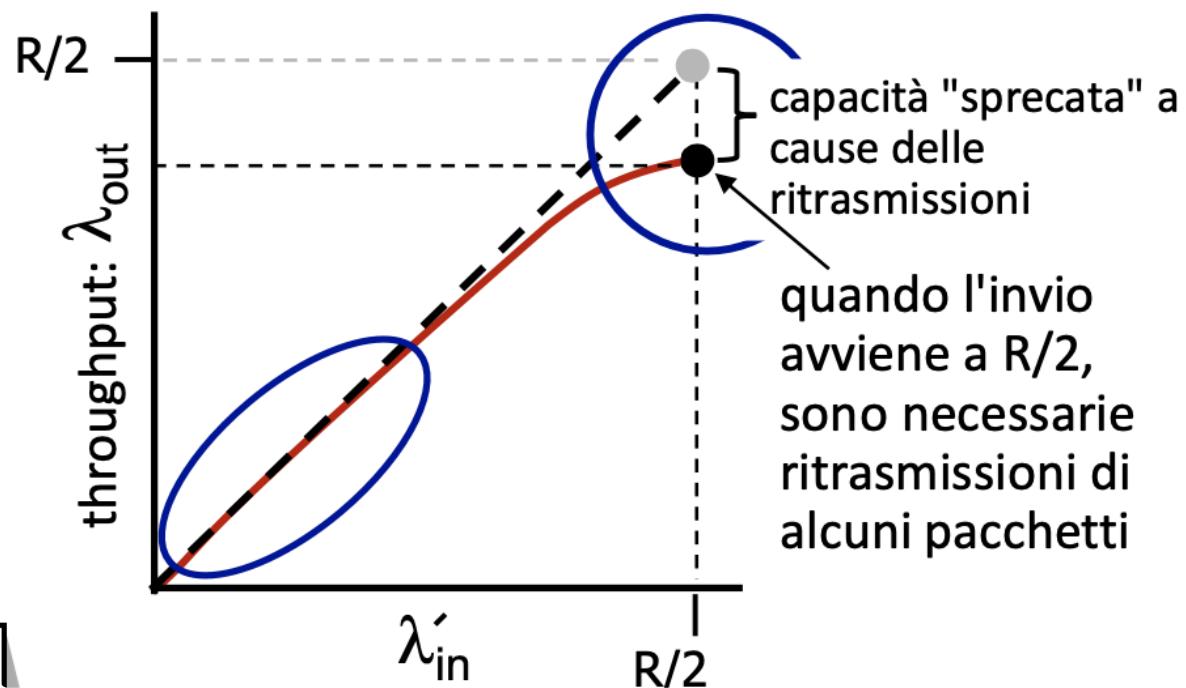
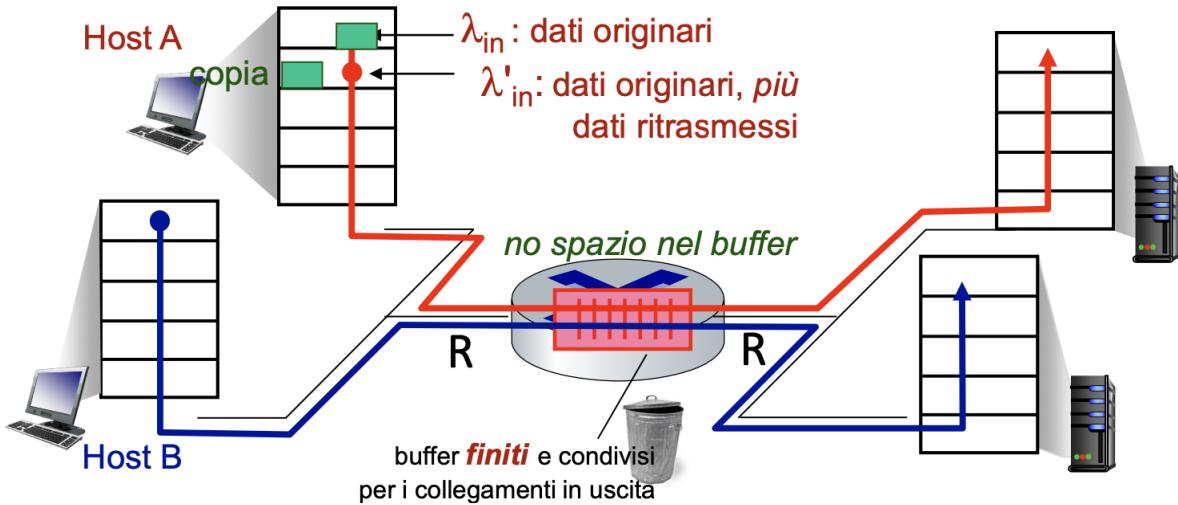
idealizzazione: conoscenza perfetta

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



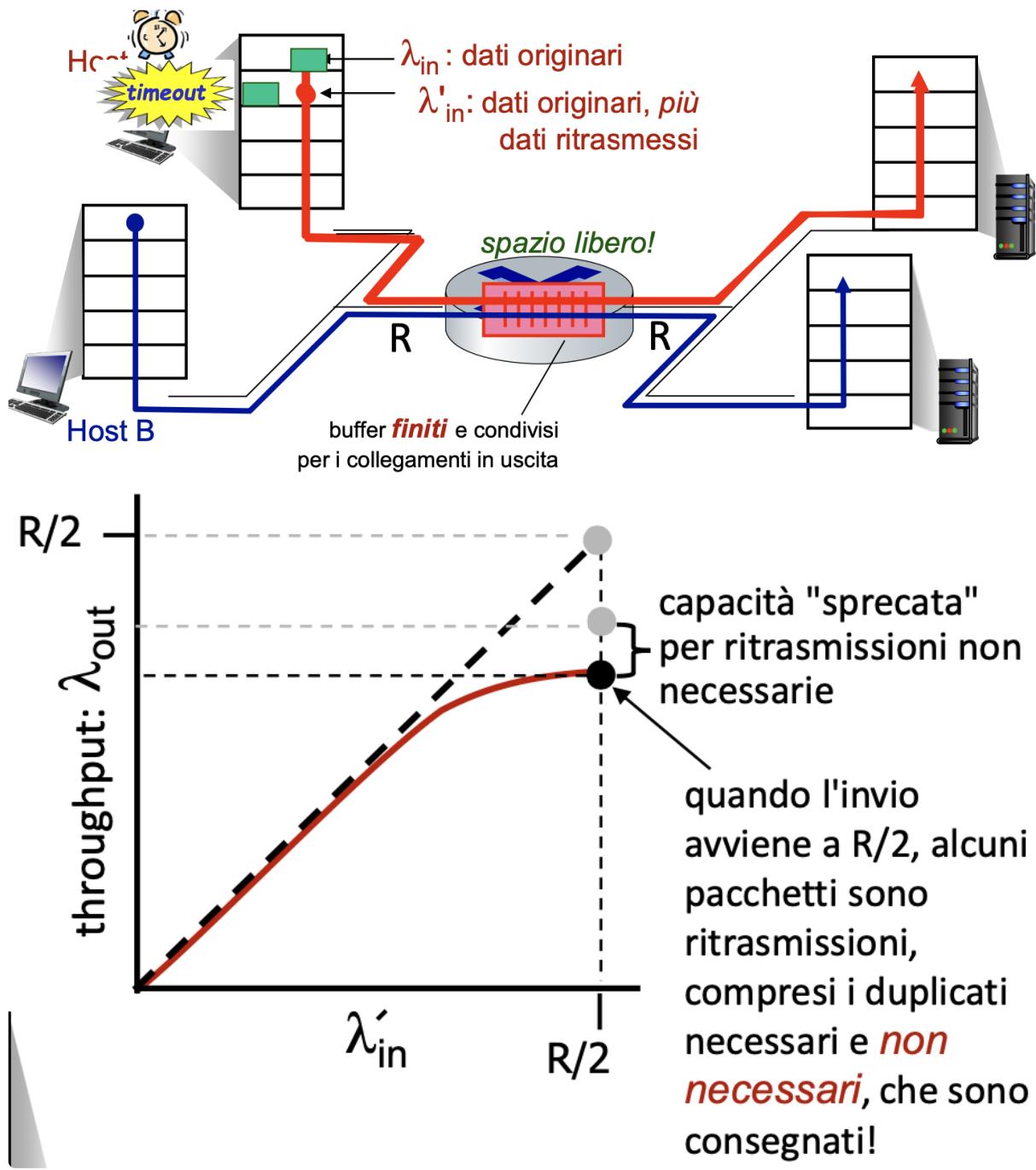
idealizzazione: un pò di conoscenza perfetta

- 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



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 :

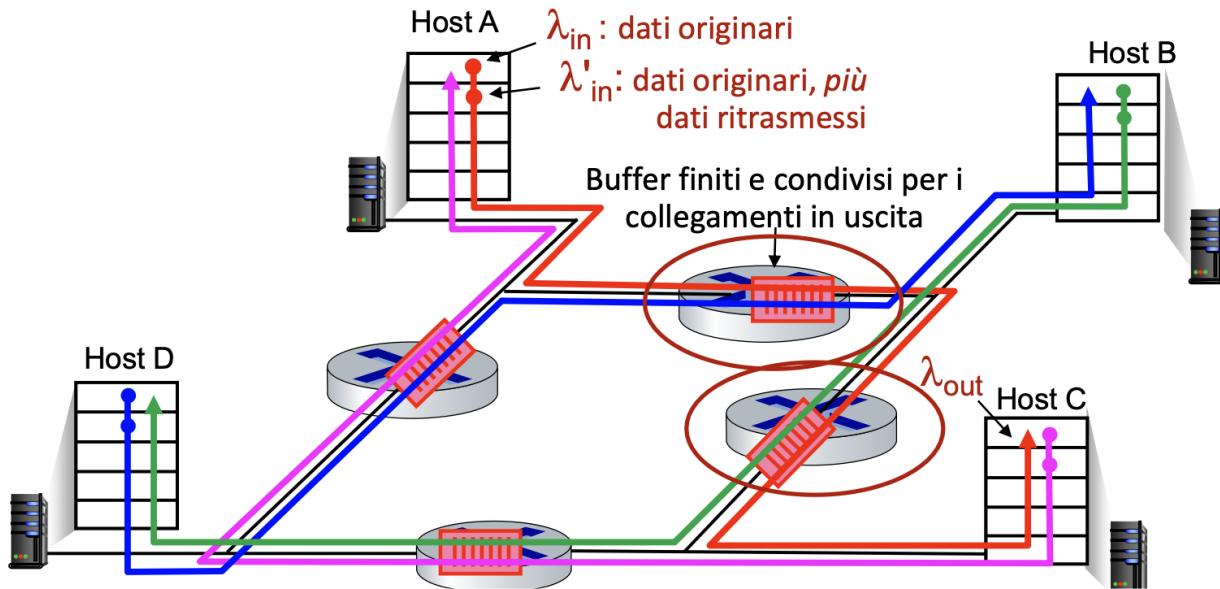
- più lavoro (ritrasmissioni) per un dato throughput di ricezione
- ritrasmissioni non necessarie: il collegamento trasporta più copie del pacchetto
 - diminuzione del throughput massimo raggiungibile

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 $\rightarrow 0$



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

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
- protocollo TCP ECN , ATM , DECbit

