Reti di calcolatori e Internet: Un approccio top-down

7ª edizione
Jim Kurose, Keith Ross

Pearson Paravia Bruno Mondadori Spa

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 del controllo di congestione
- 3.7 Controllo di congestione TCP

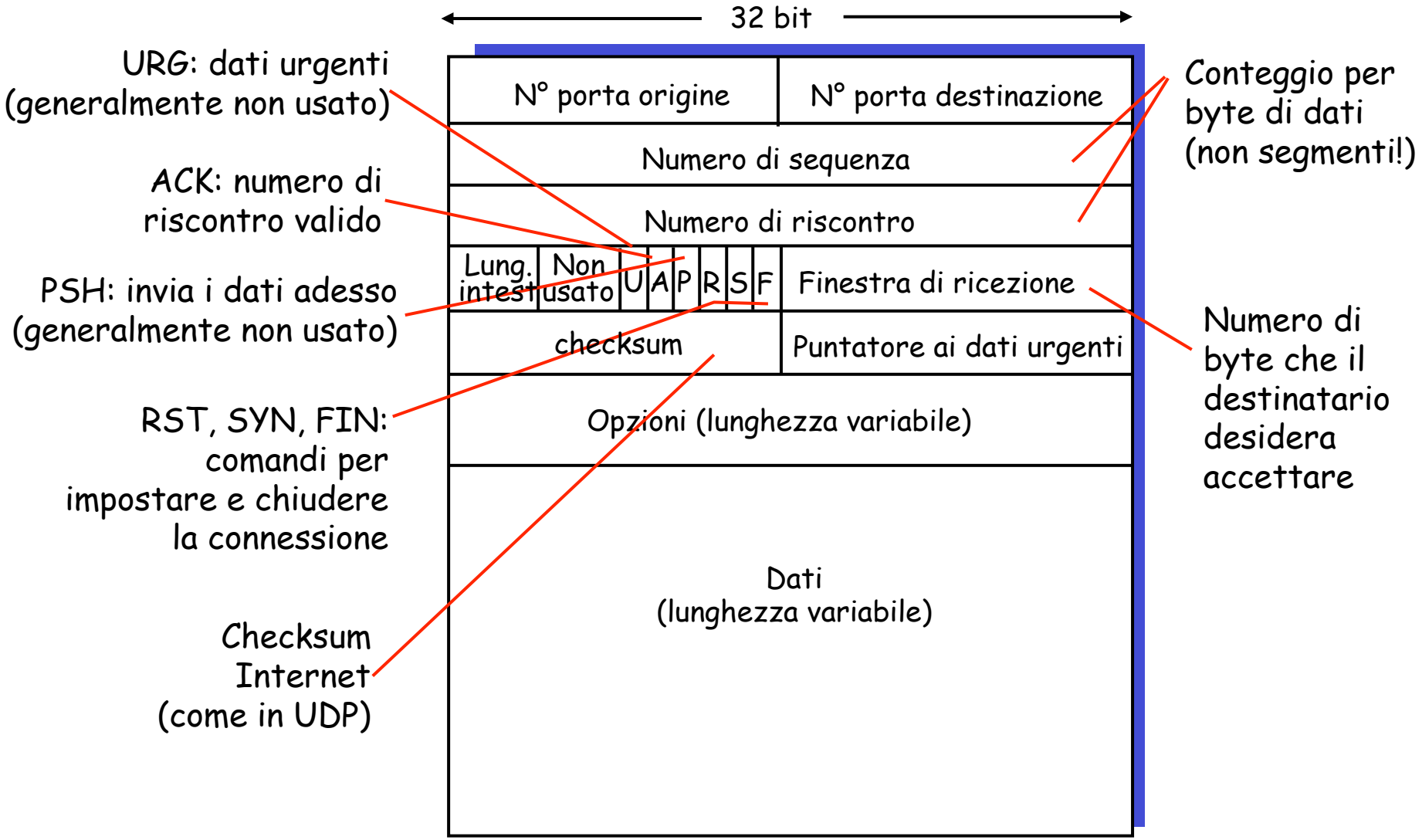
TCP: Panoramica

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 **[handshake a tre vie]**

Struttura dei segmenti TCP



TCP seq. numbers, ACKs

sequence numbers:

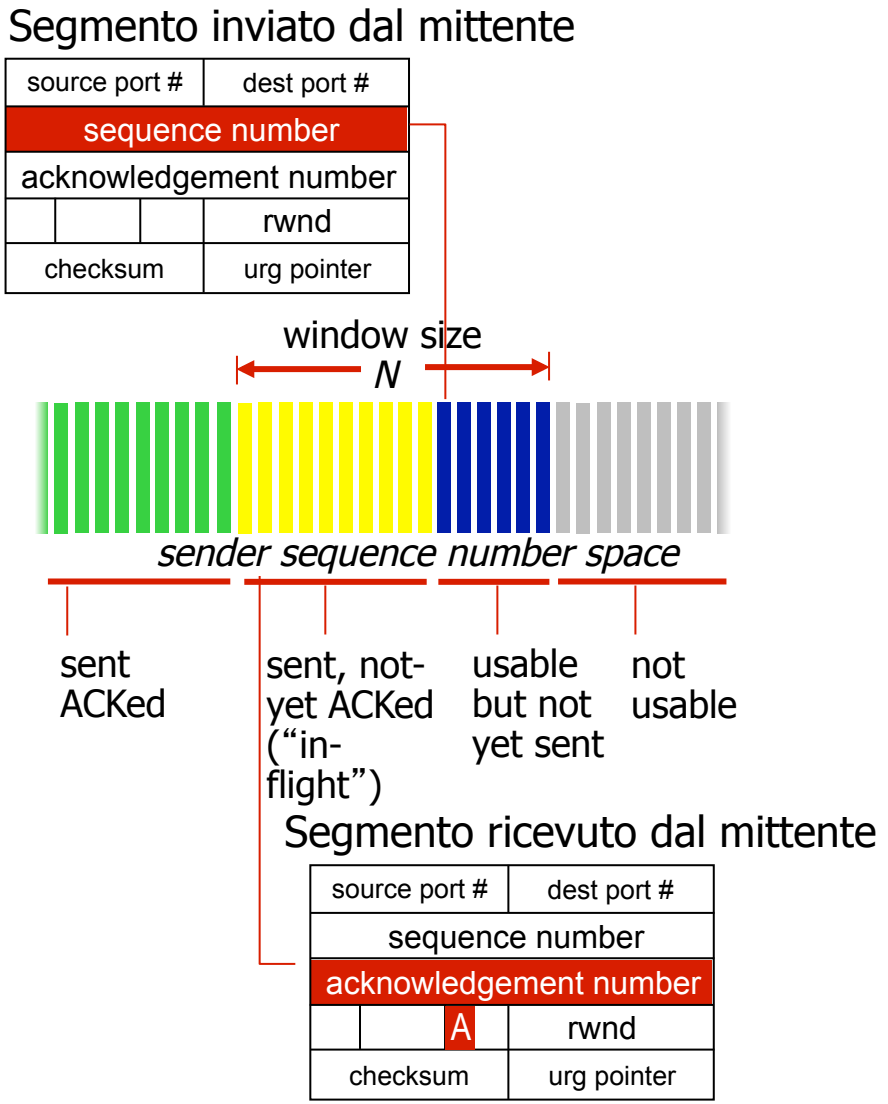
- "numero" del primo byte del segmento nel flusso di byte

acknowledgements:

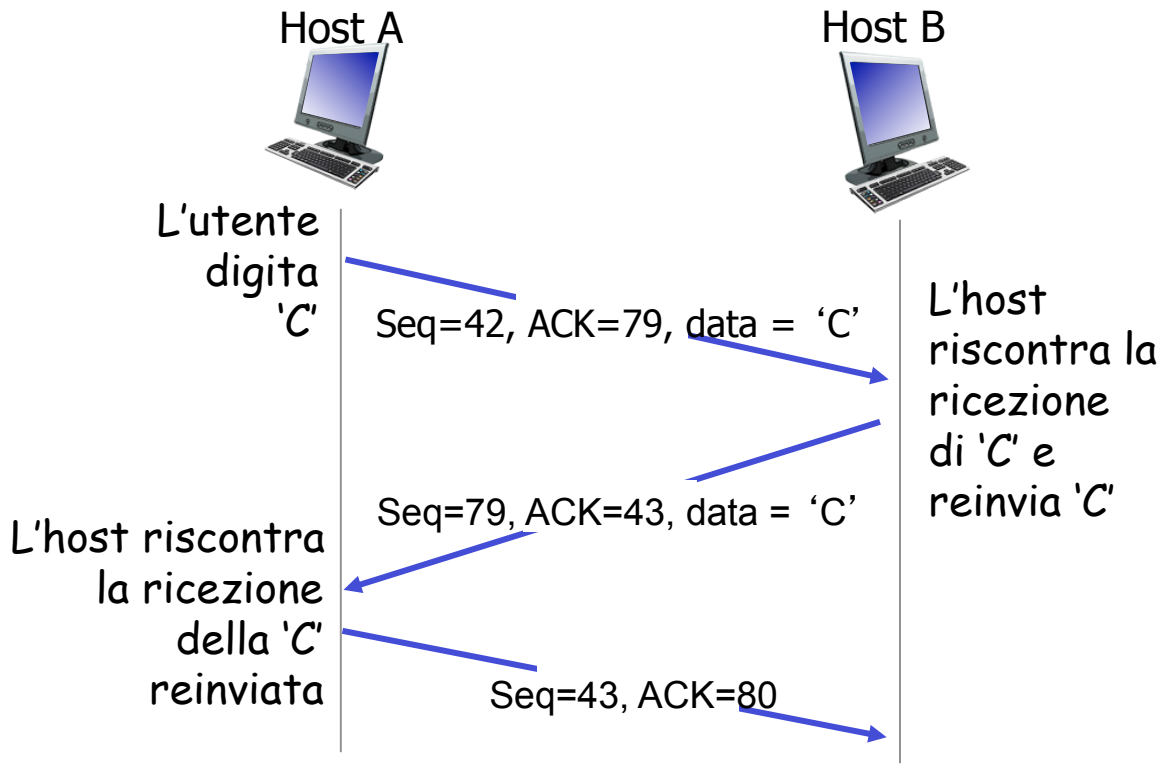
- numero di sequenza del prossimo byte atteso dall'altro lato
- ACK cumulativo

D: come gestisce il destinatario i segmenti fuori sequenza?

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



TCP seq. numbers, ACKs



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

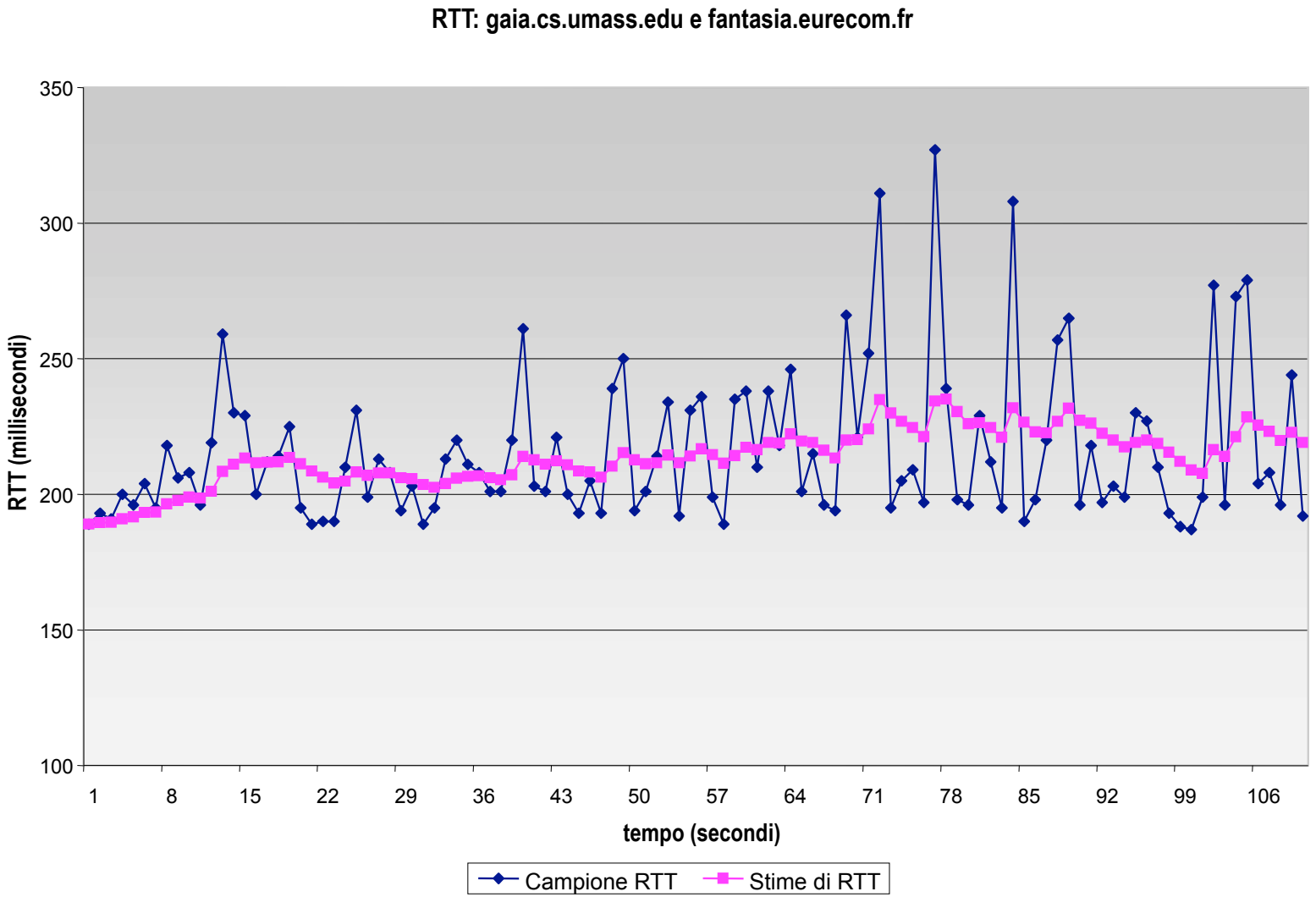
$$\text{EstimatedRTT}(t) = (1 - \alpha)*\text{EstimatedRTT}(t-1) + \alpha*\text{SampleRTT}(t)$$

- Media mobile esponenziale ponderata
- L'influenza dei vecchi campioni decresce esponenzialmente
- Valore tipico: $\alpha = 0{,}125$

D: Perché si parla di media mobile esponenziale?

Suggerimento: Espandere la formula rispetto a t-1, poi t-2, e così via, analizzando il peso dato ad ogni addendo

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:

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

(tipicamente, $\beta = 0{,}25$)

Poi impostare l'intervallo di timeout:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4*\text{DevRTT}$$

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 del controllo di congestione
- 3.7 Controllo di congestione TCP

TCP: trasferimento dati affidabile

- TCP crea un servizio di trasferimento dati affidabile sul servizio inaffidabile di IP
- Pipeline dei segmenti
- ACK cumulativi
- TCP usa un solo timer di ritrasmissione
- Le ritrasmissioni sono avviate da:
  - eventi di timeout
  - ACK duplicati
- Inizialmente consideriamo un mittente TCP semplificato:
  - ignoriamo gli ACK duplicati
  - ignoriamo il controllo di flusso e il controllo di congestione

TCP: eventi del mittente

Dati ricevuti 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 è già in funzione (pensate al timer come se fosse associato al più vecchio segmento non riscontrato)

Timeout:

- Ritrasmette il segmento che ha causato il timeout
- Riavvia il timer

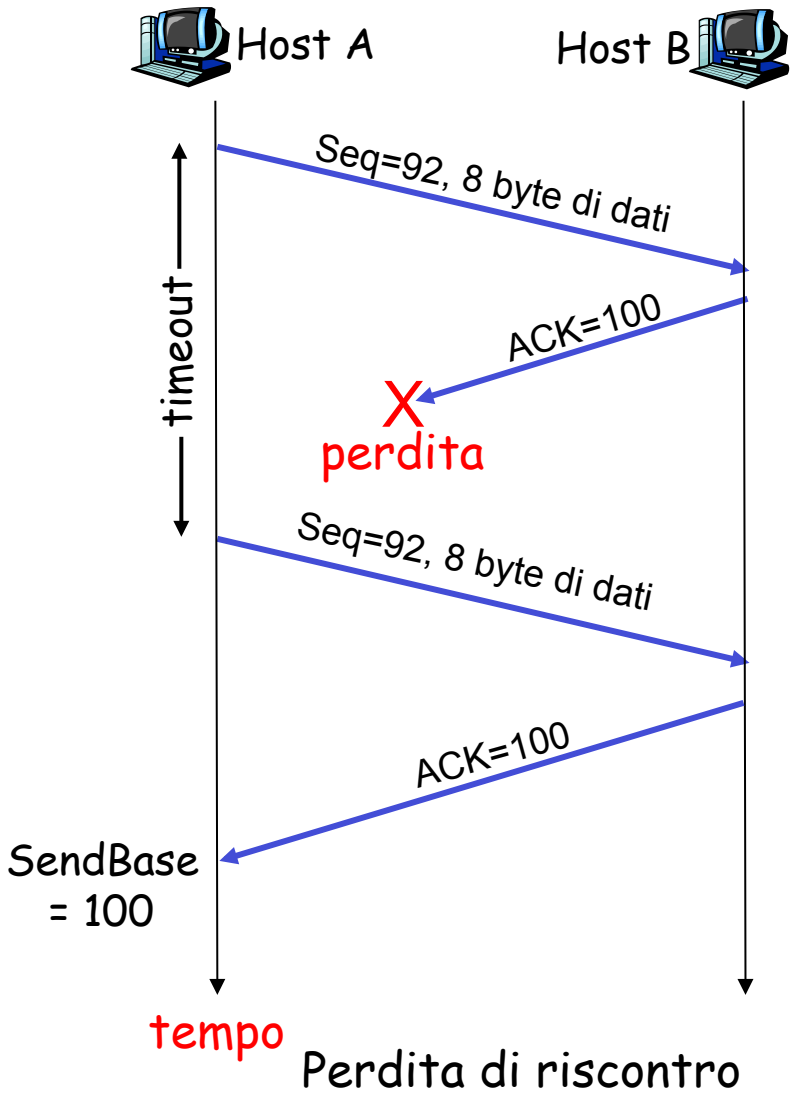
ACK ricevuti:

- Se riscontra segmenti precedentemente non riscontrati
  - aggiorna ciò che è stato completamente riscontrato
  - avvia il timer se ci sono altri segmenti da completare

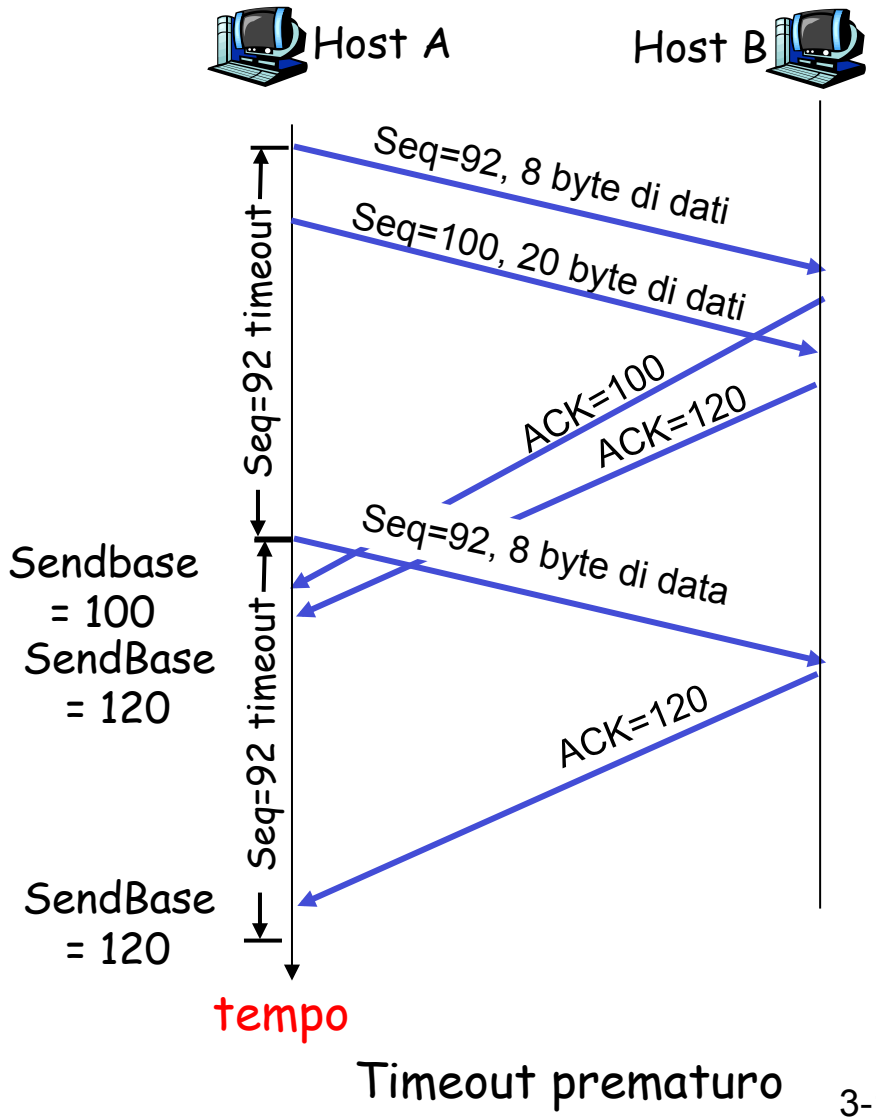
```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (sempre) {
  switch(evento)

  evento: i dati ricevuti dall'applicazione superiore
    creano il segmento TCP con numero di sequenza NextSeqNum
    if (il timer attualmente non funziona)
        avvia il timer
    passa il segmento a IP
    NextSeqNum = NextSeqNum + lunghezza(dati)

  evento: timeout del timer
    ritrasmetti il segmento non ancora riscontrato con
          il più piccolo numero di sequenza
    avvia il timer

  evento: ACK ricevuto, con valore del campo ACK pari a y
    if (y > SendBase) {
        SendBase = y
       if (esistono attualmente segmenti non ancora riscontrati)
            avvia il timer
        }

} /* fine del loop */
```

Mittente TCP (semplificato)

TCP: scenari di ritrasmissione

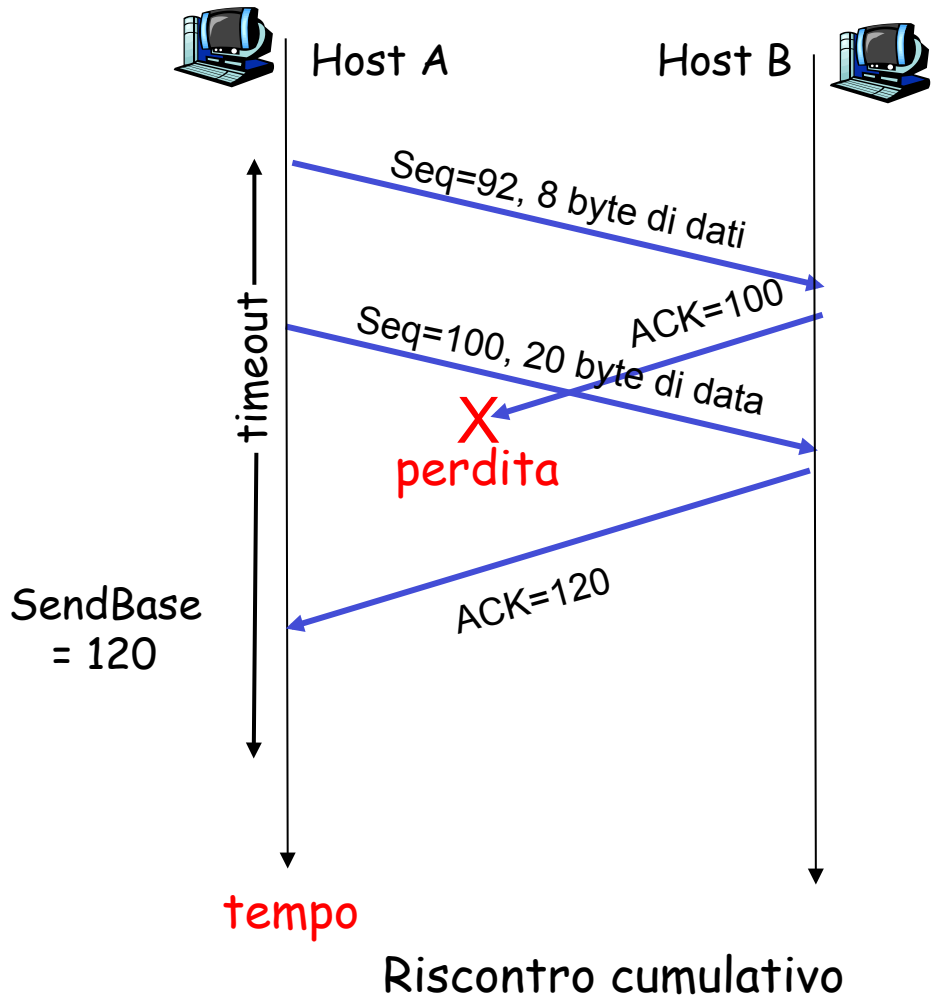


Perdita di riscontro



Timeout prematuro

TCP: scenari di ritrasmissione



Riscontro cumulativo

TCP: generazione di ACK [RFC 1122, RFC 2581]

Evento presso il destinatario	Azione del destinatario TCP
Arrivo ordinato di un segmento con numero di sequenza atteso. Tutti i dati fino al numero di sequenza atteso sono già stati riscontrati.	ACK ritardato. Attende fino a 500 ms l'arrivo del prossimo segmento. Se il segmento non arriva, invia un ACK.
Arrivo ordinato di un segmento con numero di sequenza atteso. Un altro segmento è in attesa di trasmissione dell'ACK.	Invia immediatamente un singolo ACK cumulativo, riscontrando entrambi i segmenti ordinati.
Arrivo non ordinato di un 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.
Arrivo di un segmento che colma parzialmente o completamente il buco.	Invia immediatamente un ACK, ammesso che il segmento cominci all'estremità inferiore del buco.

Raddoppio dell' intervallo di timeout (variante di alcune implementazioni)

- Scadenza timeout
  - Raddoppio dell' intervallo di timeout
  - Crescita esponenziale dell' intervallo
- Riavvio timer per dati dall' applicazione o per la ricezione di ACK
  - Si ricomincia dai più recenti valori di `EstimatedRTT e DevRTT`
- Blando controllo della congestione

Ritrasmissione rapida – triplo ACK duplicato

- Il periodo di timeout spesso è relativamente lungo:
  - lungo ritardo prima di ritrasmettere il pacchetto perduto.
- Rileva i segmenti perduti tramite gli ACK duplicati.
  - Il mittente spesso invia molti segmenti.
  - Se un segmento viene smarrito, è probabile che ci saranno molti ACK duplicati.
- Se il mittente riceve 3 ACK per lo stesso dato, suppone che il segmento che segue il dato riscontrato è andato perduto:
  - ritrasmissione rapida: rispedisce il segmento prima che scada il timer.

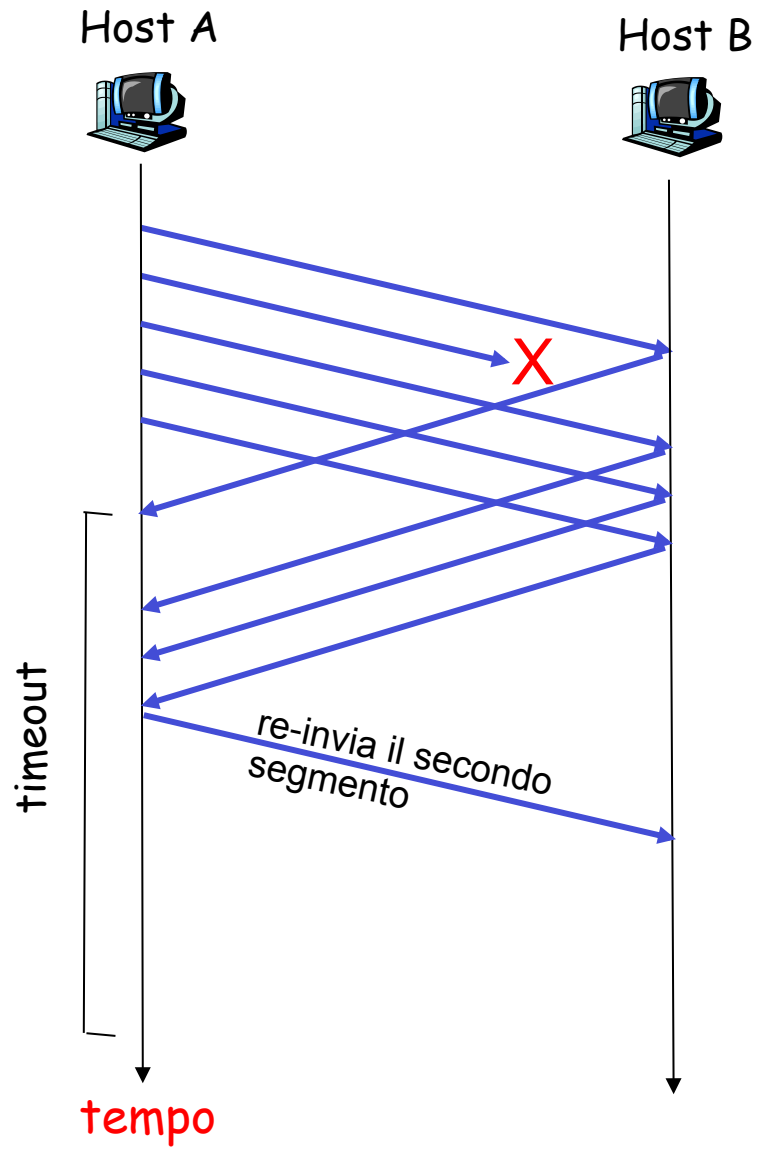


Figure 3.37 Re-invio di un segmento dopo un triplice ACK duplicato

Algoritmo della ritrasmissione rapida:

```
evento: ACK ricevuto, con valore del campo ACK pari a y
        if (y > SendBase) {
            SendBase = y
            if (esistono attualmente segmenti non ancora riscontrati)
                avvia il timer
        }
        else {
            incrementa il numero di ACK duplicati ricevuti per y
            if (numero di ACK duplicati ricevuti per y = 3) {
                rispedisci il segmento con numero di sequenza y
            }
```

un ACK duplicato per un segmento già riscontrato

ritrasmissione rapida

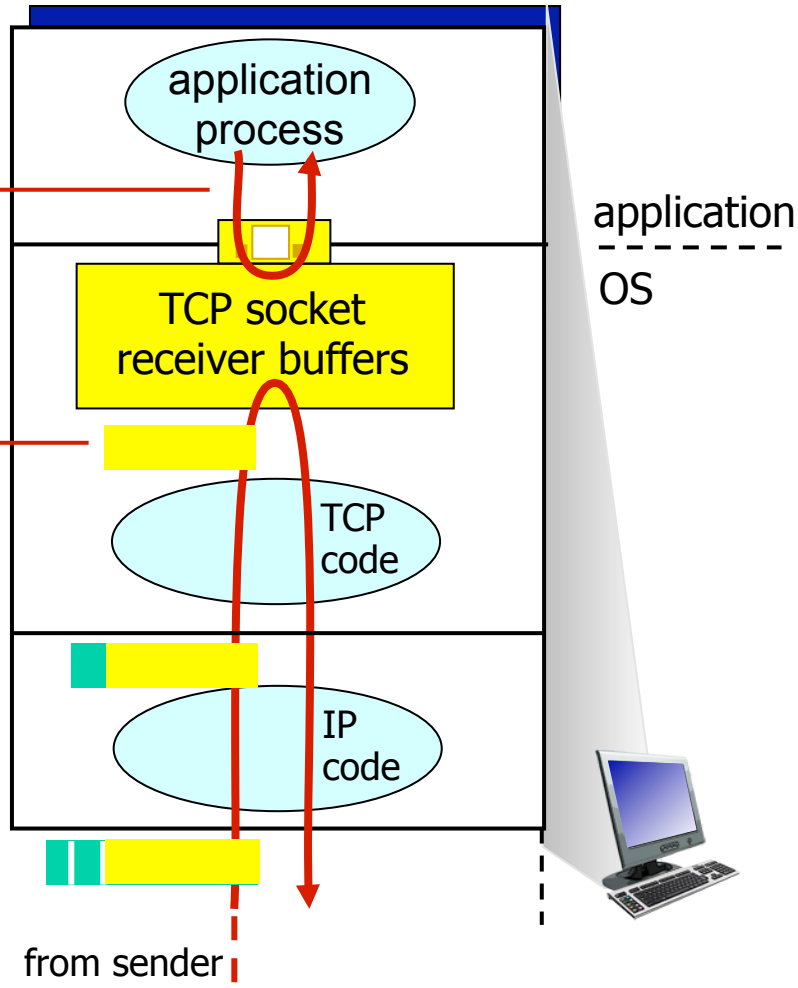
Capitolo 3: Livello di trasporto

- 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 del controllo di congestione
- 3.7 Controllo di congestione TCP

TCP controllo di flusso

Il processo applicativo potrebbe leggere i dati dal buffer della socket TCP ....

... più lentamente di quanto il TCP destinatario li stia ricevendo



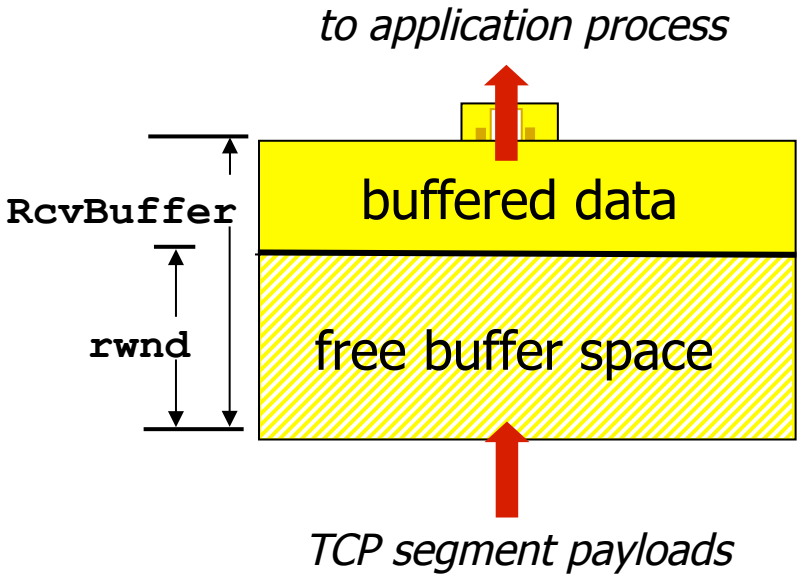
protocol stack del destinatario

Controllo di flusso

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

TCP flow control

- Il mittente comunica lo spazio disponibile includendo il valore di **`RcvWindow`** (**`rwnd`**) nei segmenti
  - **`RcvBuffer`** è una delle opzioni nella creazione della socket (il valore di default è 4096 B)
  - Molti sistemi operativi adattano **`RcvBuffer`**
- Il mittente limita i dati non riscontrati a **`RcvWindow`**
- garantisce che il buffer di ricezione non vada in overflow



receiver-side buffering

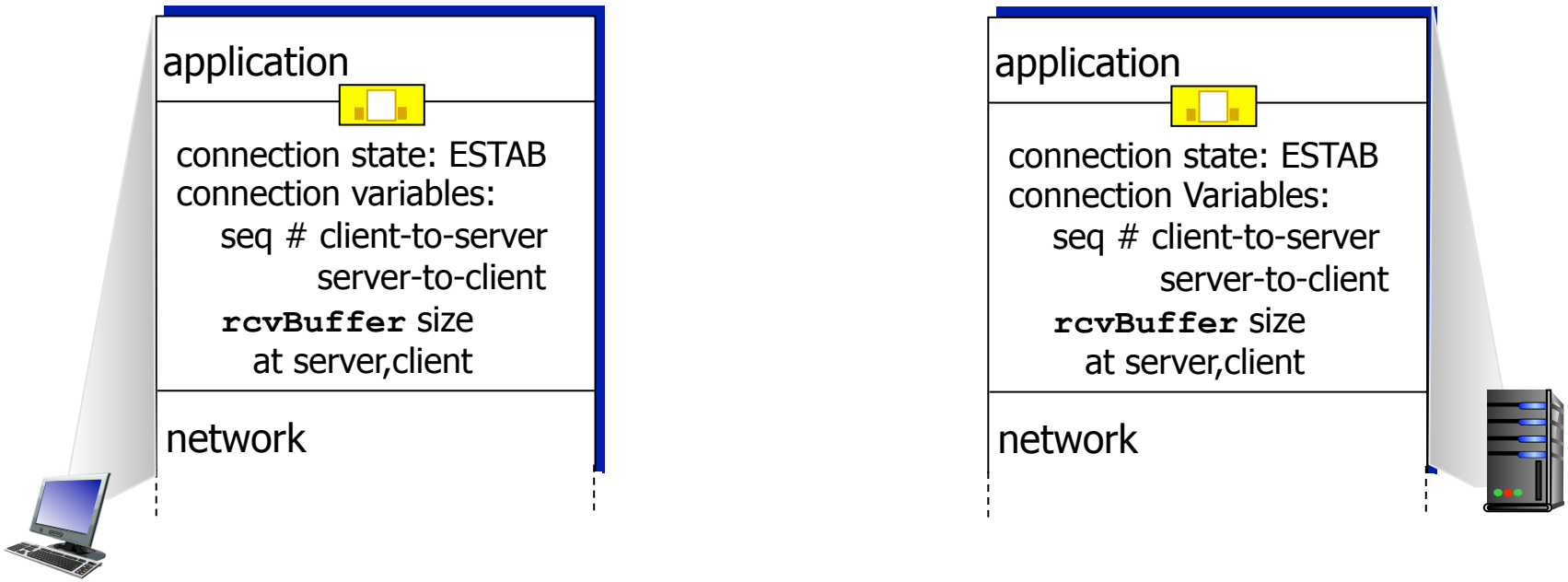
Capitolo 3: Livello di trasporto

- 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 del 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**)




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

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

Gestione della connessione TCP

Handshake a tre vie:

Passo 1: il client invia un segmento SYN al server

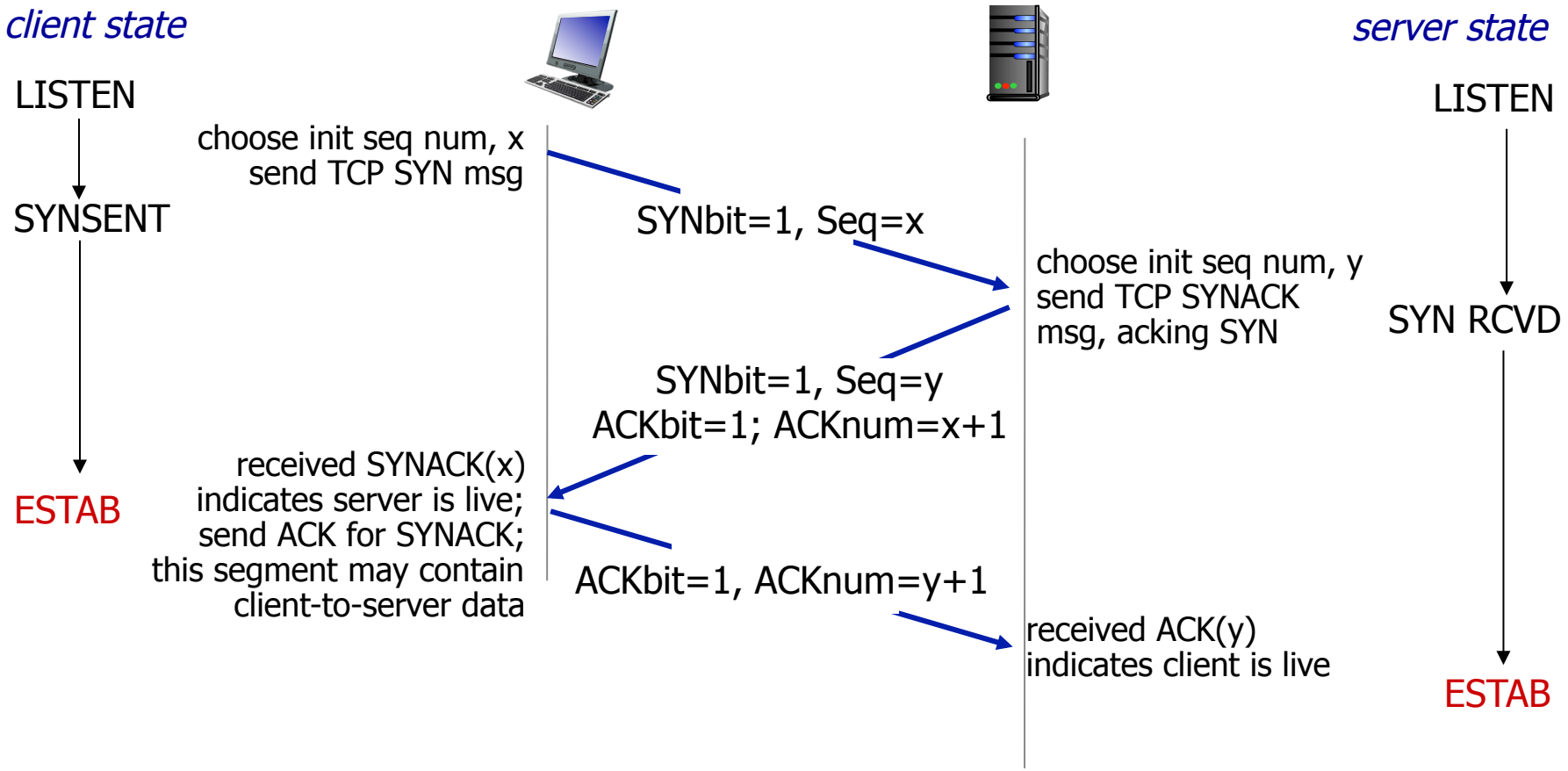
- specifica il numero di sequenza iniziale
- nessun dato

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

TCP 3-way handshake



Gestione della connessione TCP (continua)

Chiudere una connessione:

Es. il client chiude la socket:

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

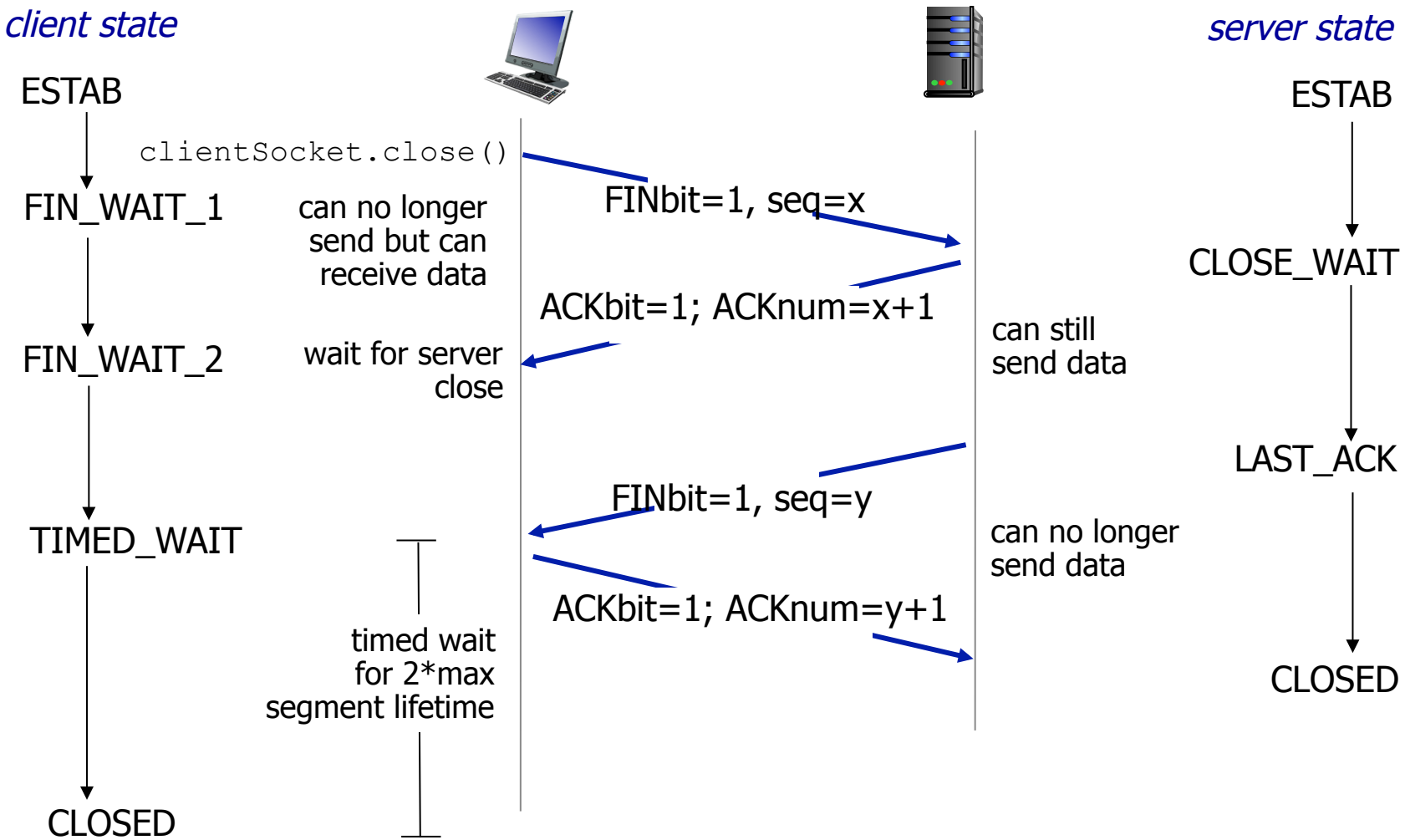
Passo 2: il server riceve il segmento FIN e risponde con un ACK e invia un FIN.

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

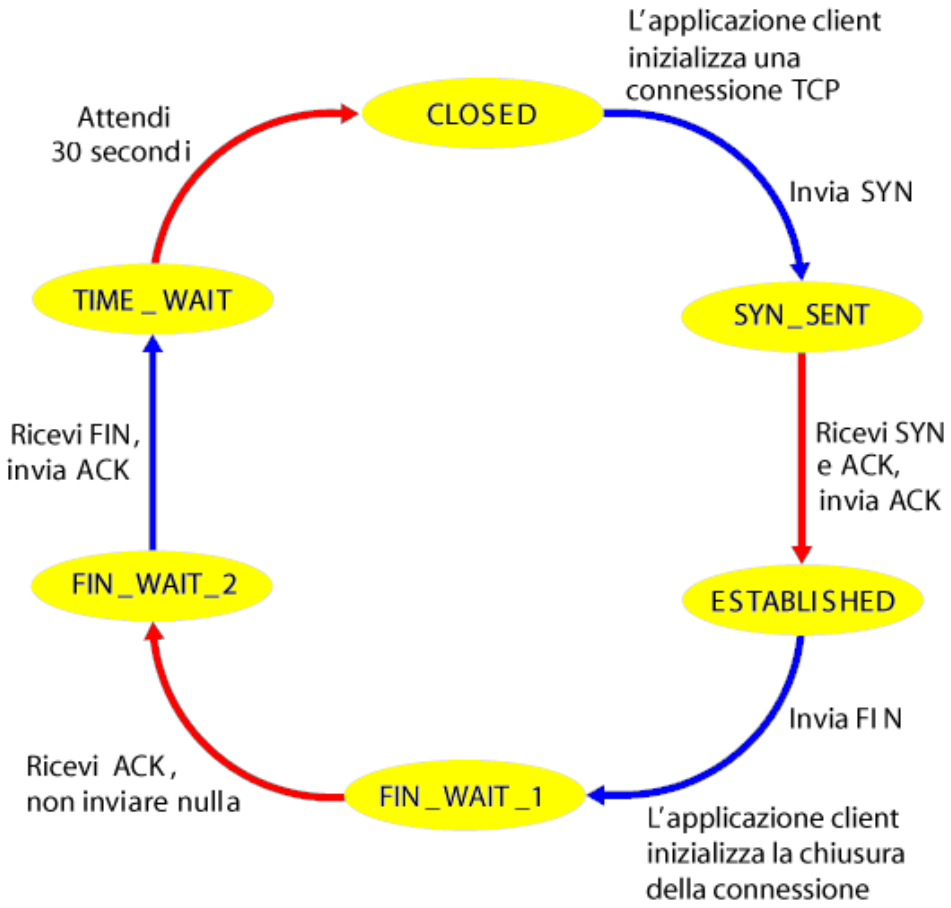
- inizia l'attesa temporizzata - risponde con un ACK ai FIN che riceve

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

TCP: closing a connection



Gestione della connessione TCP (continua)



Sequenza di stati visitati da un client TCP

Sequenza di stati visitati dal TCP sul lato server