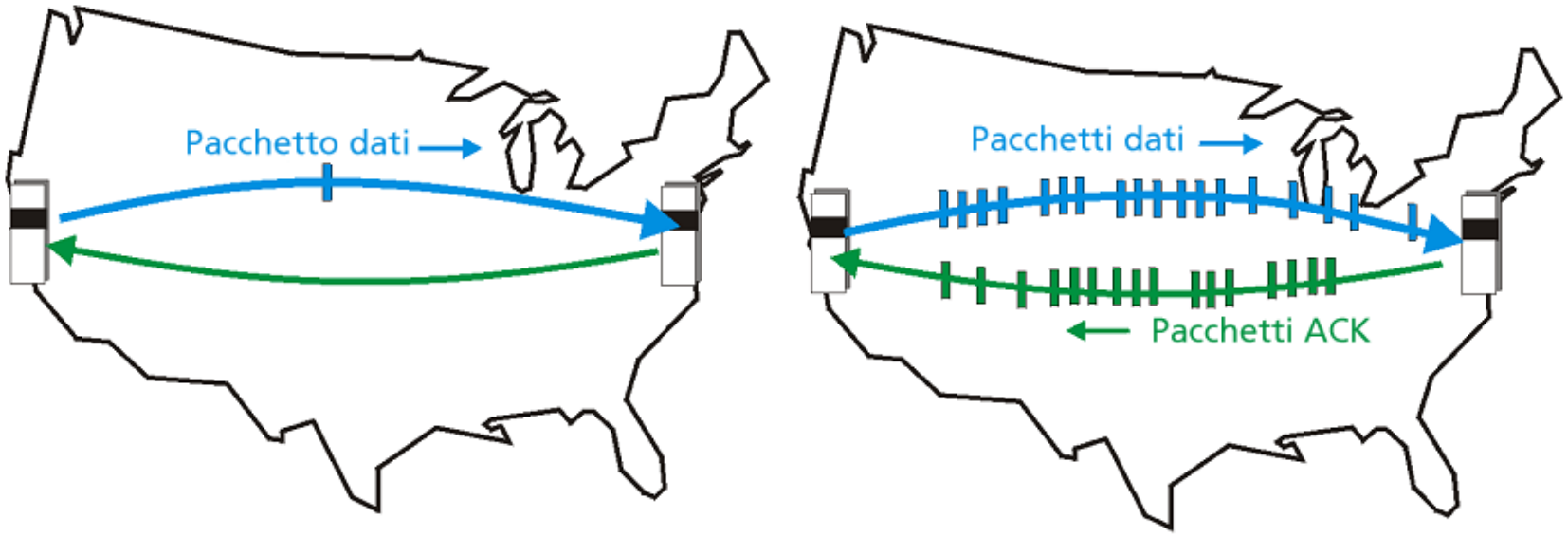
Livello di trasporto: meccanismi trasferimento dati affidabile (2), TCP

Gaia Maselli
maselli@di.uniroma1.it

Protocolli con pipeline

Pipelining: il mittente ammette più pacchetti in transito, ancora da notificare

- l'intervallo dei numeri di sequenza deve essere incrementato
- buffering dei pacchetti presso il mittente e/o ricevente



a) Protocollo stop-and-wait all'opera

b) Protocollo con pipeline all'opera

- Due forme generiche di protocolli con pipeline: *Go-Back-N* e *ripetizione selettiva*

Protocolli con pipeline

Go-back-N:

- Il mittente può avere fino a *N* pacchetti senza ACK in pipeline
- Il ricevente invia solo ACK cumulativi
  - Non dà l'ACK di un pacchetto se c'è un gap
- Il mittente ha un timer per il più vecchio pacchetto senza ACK
  - Se il timer scade, ritrasmette tutti i pacchetti senza ACK

Ripetizione selettiva

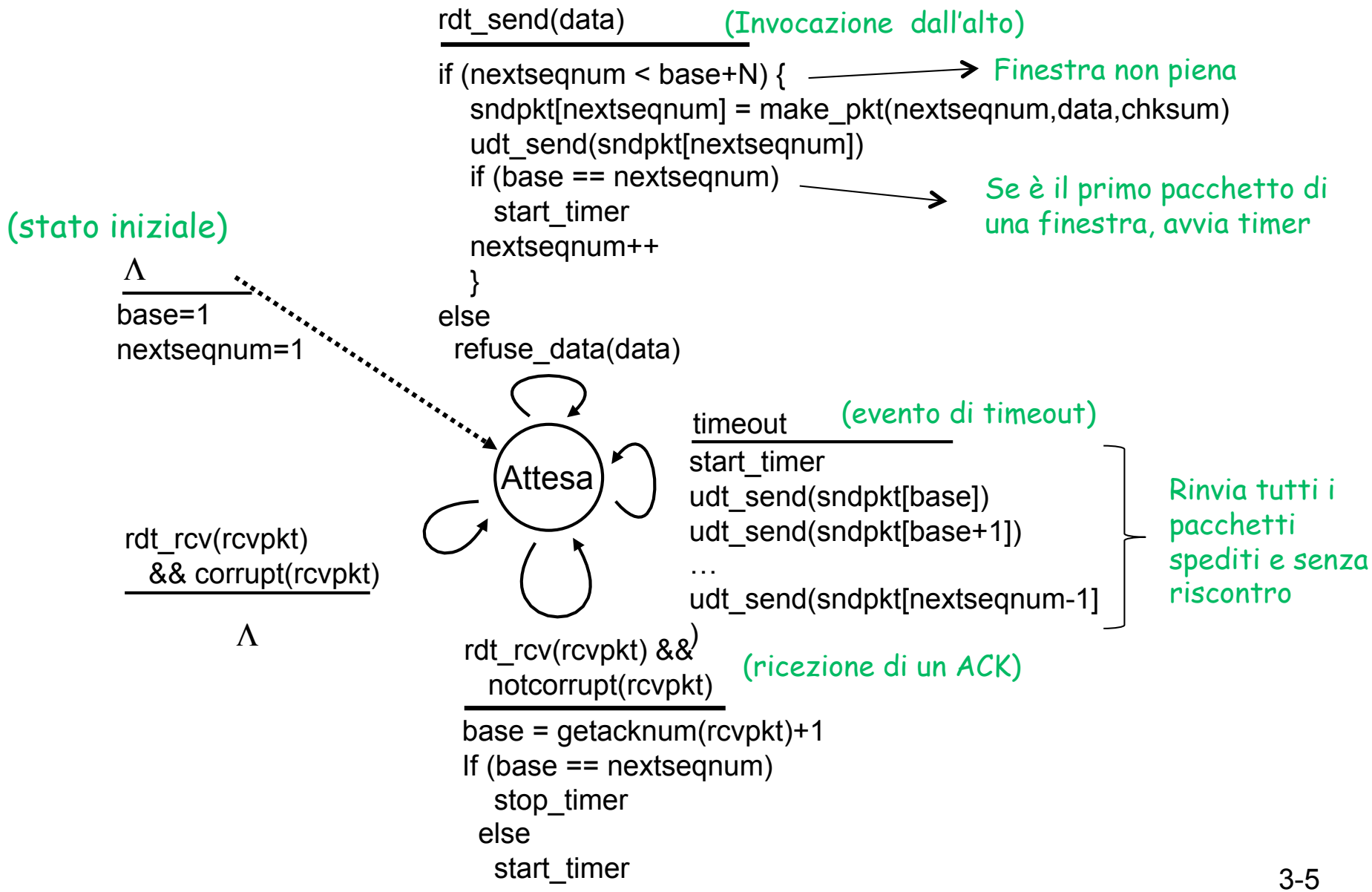
- Il mittente può avere fino a *N* pacchetti senza ACK in pipeline
- Il ricevente invia ACK solo ai singoli pacchetti
- Il mittente mantiene un timer per ciascun pacchetto che non ha ancora ricevuto ACK
  - Quando il timer scade, ritrasmette solo i pacchetti che non hanno avuto ACK

Go-Back-N

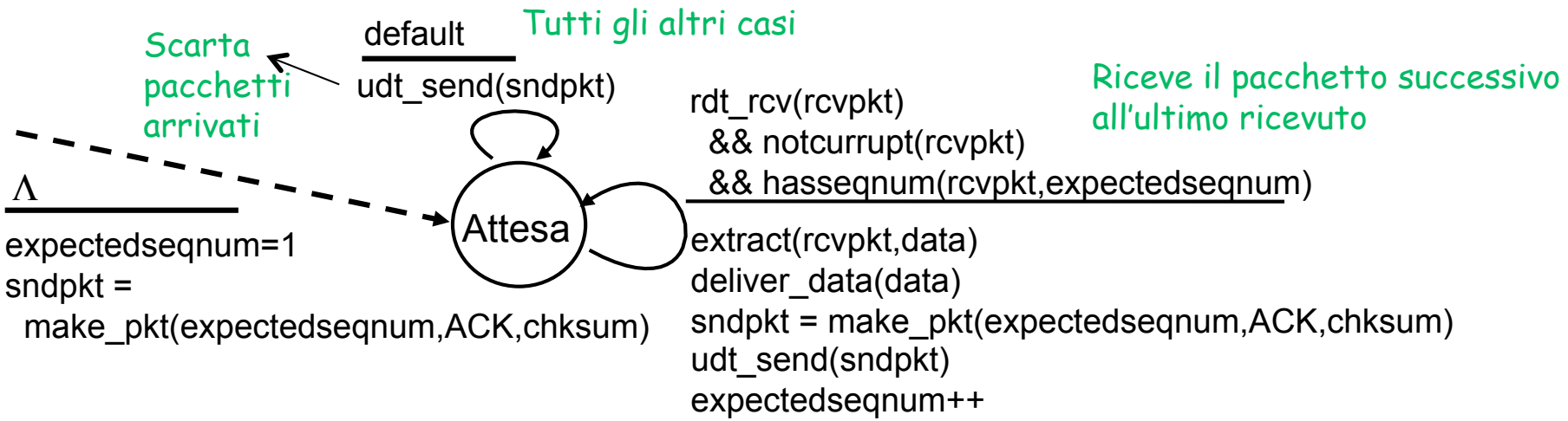
- Mittente:
- Numero di sequenza a k bit nell'intestazione del pacchetto
- "Finestra" contenente fino a N pacchetti consecutivi non riscontrati

- ACK(n): riscontro di tutti i pacchetti con numero di sequenza minore o uguale a n - "riscontri cumulativi"
  - pacchetti duplicati potrebbero essere scartati (vedere il ricevente)
- timer per il più vecchio pacchetto senza ack
- *timeout(n):* ritrasmette il pacchetto *n* e tutti i pacchetti con i numeri di sequenza più grandi nella finestra

GBN: automa esteso del mittente



GBN: automa esteso del ricevente

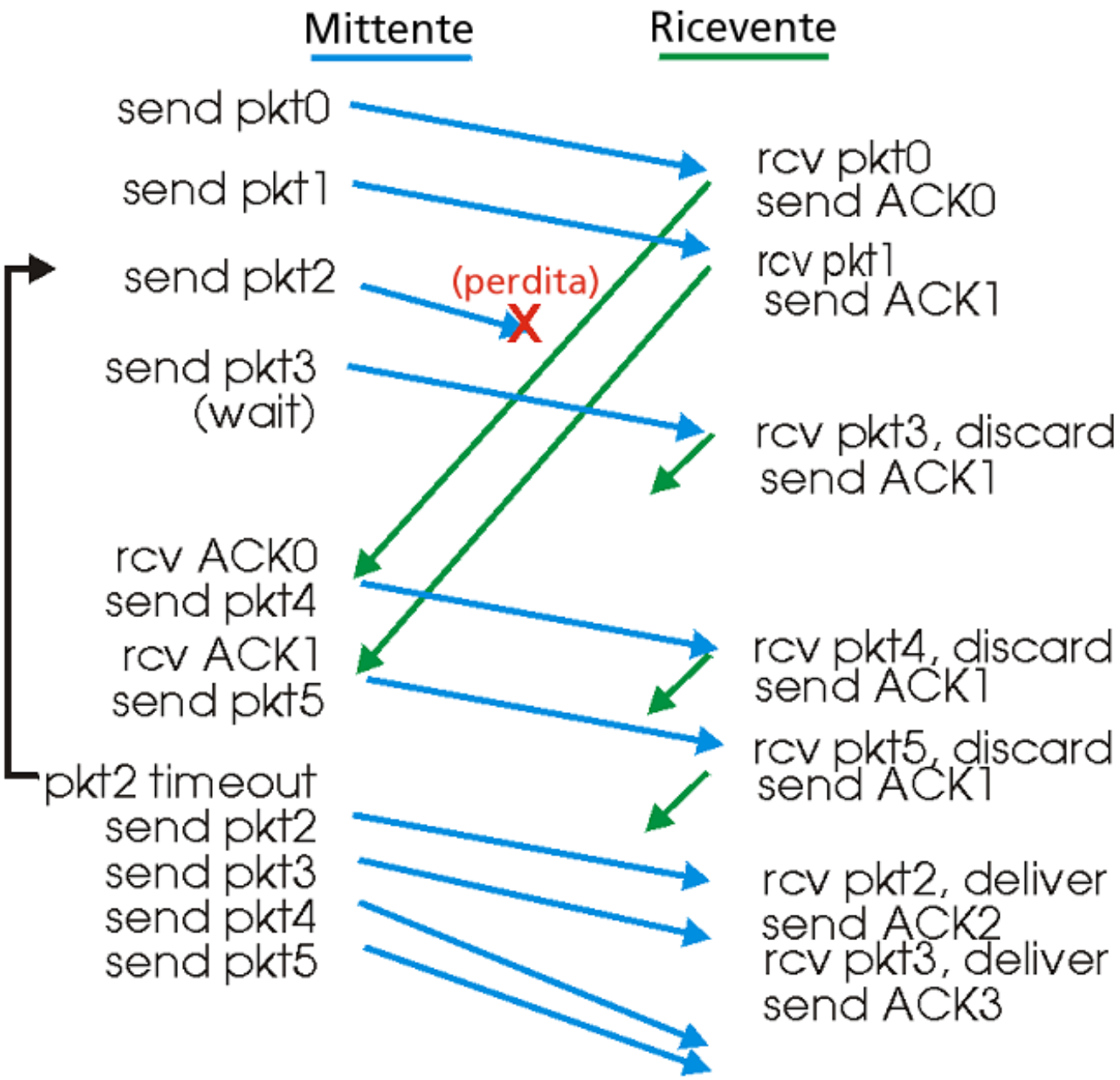


ACK-soltanto: invia sempre un ACK per un pacchetto ricevuto correttamente con il numero di sequenza più alto *in sequenza*

- potrebbe generare ACK duplicati
- deve memorizzare soltanto **`expectedseqnum`**

- Pacchetto fuori sequenza:
  - scartato (non è salvato) -> senza buffering del ricevente!
  - rimanda un ACK per il pacchetto con il numero di sequenza più alto *in sequenza*

GBN in azione



Ripetizione selettiva

- In GBN per un solo pacchetto perso si ritrasmettono tutti i successivi già inviati nel pipeline
- Nella *ripetizione selettiva*, il mittente ritrasmette **soltanto** i pacchetti per i quali non ha ricevuto un ACK
  - timer del mittente per ogni pacchetto non riscontrato
- Il ricevente invia riscontri **specifici** per tutti i pacchetti ricevuti correttamente (sia in ordine, sia fuori sequenza)
  - buffer dei pacchetti, se necessario, per eventuali consegne in sequenza al livello superiore
- Finestra del mittente
  - *N* numeri di sequenza consecutivi
  - Limita ancora i numeri di sequenza dei pacchetti inviati non riscontrati

Ripetizione selettiva: finestre del mittente e del ricevente

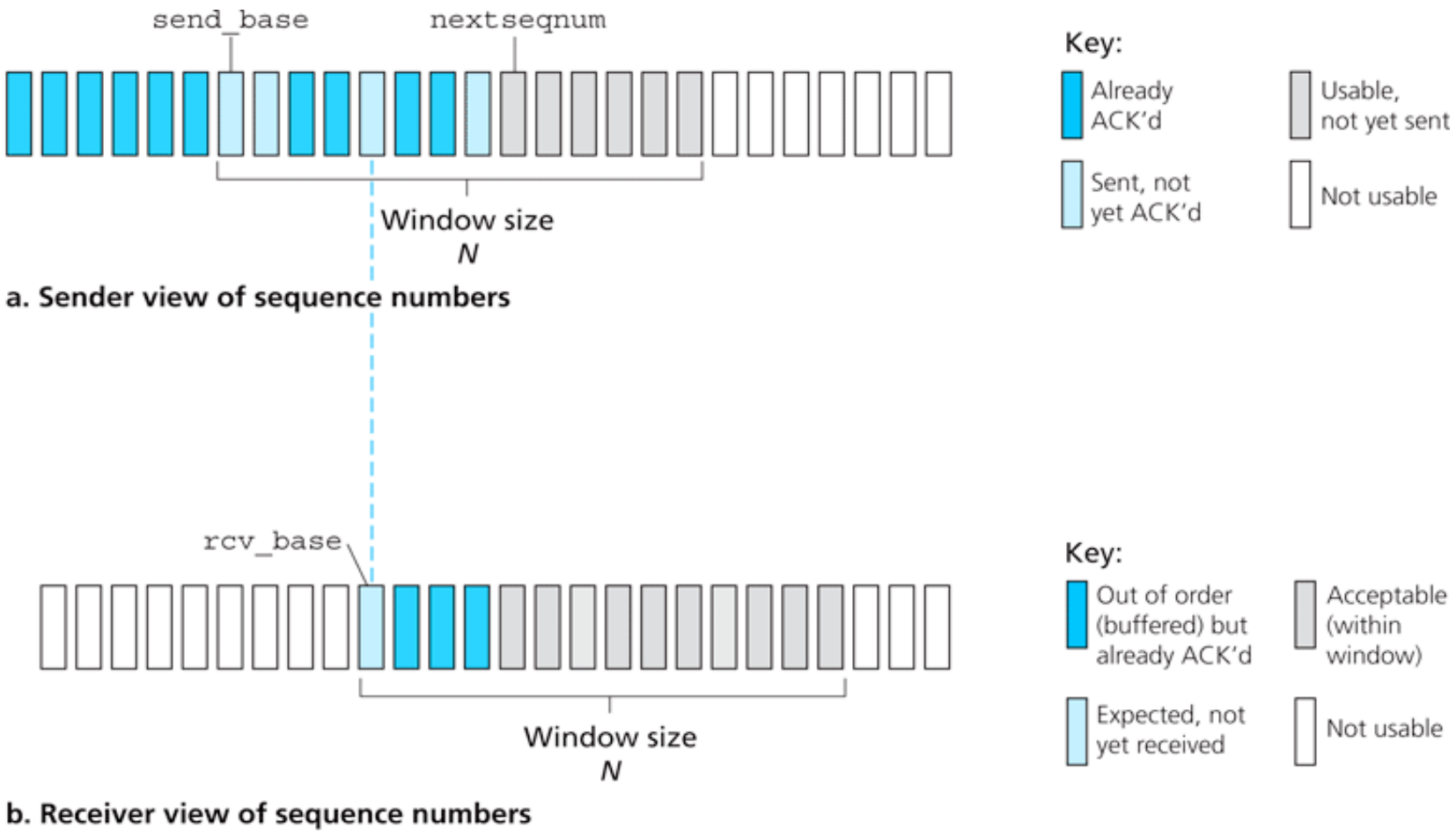


Figure 3.23 ♦ Selective-repeat (SR) sender and receiver views of sequence-number space

Ripetizione selettiva

Mittente

Dati dall'alto:

- Se nella finestra è disponibile il successivo numero di sequenza, invia il pacchetto

Timeout(n):

- Ritrasmette il pacchetto n, riparte il timer (ogni pacchetto ha il proprio timer, dato che al timeout sarà ritrasmesso un solo pacchetto)

ACK(n) in [sendbase,sendbase+N]:

- Marca il pacchetto *n* come ricevuto
- Se *n* è il numero di sequenza più piccolo, la base della finestra avanza al successivo numero di sequenza del pacchetto non riscontrato, e nuovi pacchetti possono essere trasmessi

Ricevente

Pacchetto n in [rcvbase, rcvbase +N-1]

- Invia ACK(n)
- Fuori sequenza: buffer
- In sequenza: consegna (vengono consegnati anche i pacchetti bufferizzati in sequenza); la finestra avanza al successivo pacchetto non ancora ricevuto

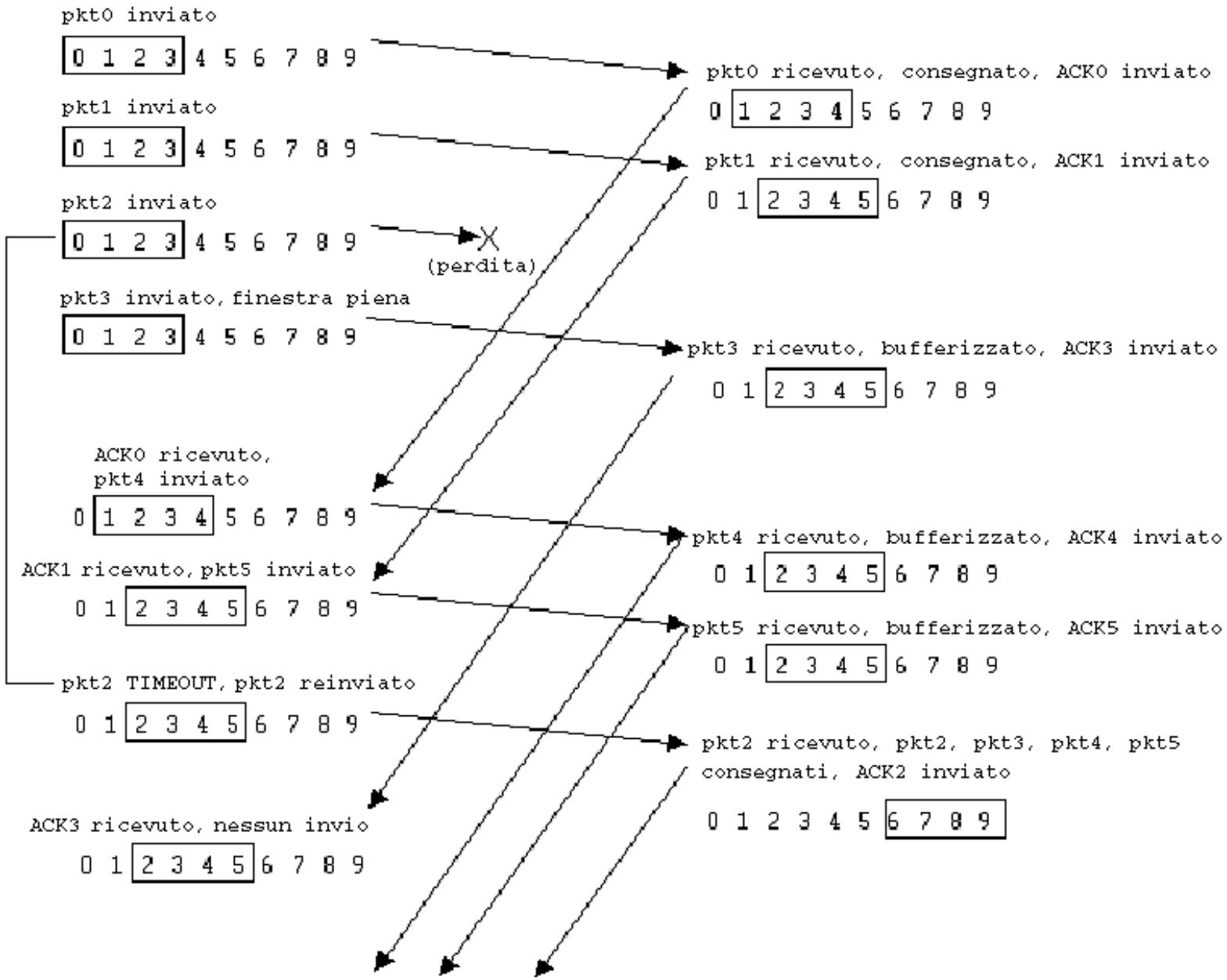
Pacchetto n in [rcvbase-N, rcvbase-1]

- Pacchetto già ricevuto e riscontrato, si manda cmq ACK(n). N.B. importante mandarlo altrimenti la finestra del mittente non avanza

altrimenti:

- ignora

Ripetizione selettiva in azione



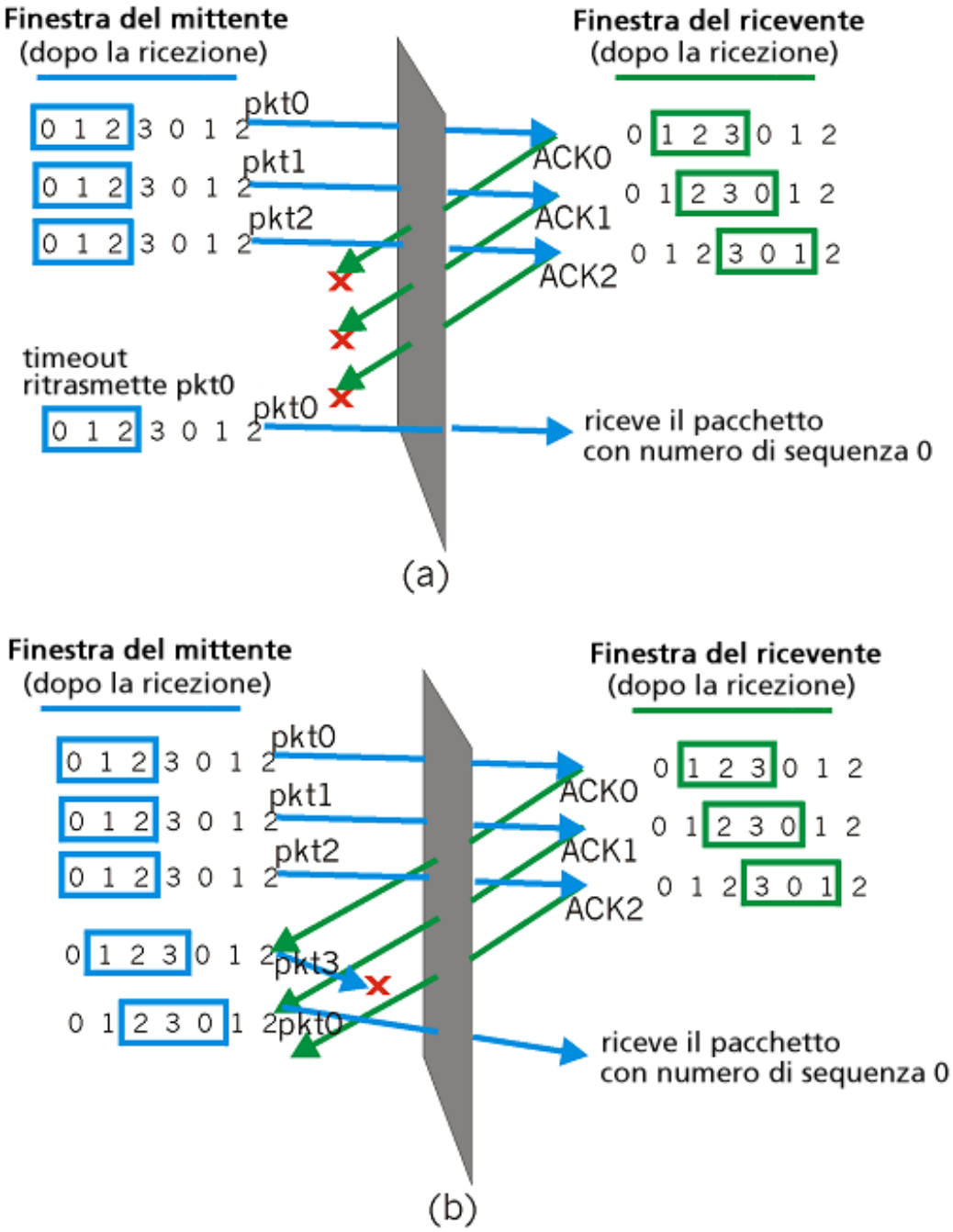
Ripetizione selettiva: problema

Esempio:

- Numeri di sequenza: 0, 1, 2, 3
- Dimensione della finestra = 3

- Il ricevente non vede alcuna differenza fra i due scenari!
- Passa erroneamente i dati duplicati come nuovi in (a)

D: Qual è la relazione fra lo spazio dei numeri di sequenza e la dimensione della finestra?



Riassunto dei meccanismi di trasferimento dati affidabile e loro utilizzo

Meccanismo	Uso
Checksum	Per gestire errori nel canale
Acknowledgment	Per gestire errori nel canale
Numero di sequenza	Ack con errori Perdita pacchetti
Timeout	Perdita pacchetti
Finestra scorrevole, pipeling	Maggior utilizzo della rete

Gestione canale inaffidabile

Miglioramento prestazioni

- Meccanismi per realizzare un trasferimento dati affidabile in un contesto generale
- TCP come e quali meccanismi usa per realizzare un trasferimento affidabile?

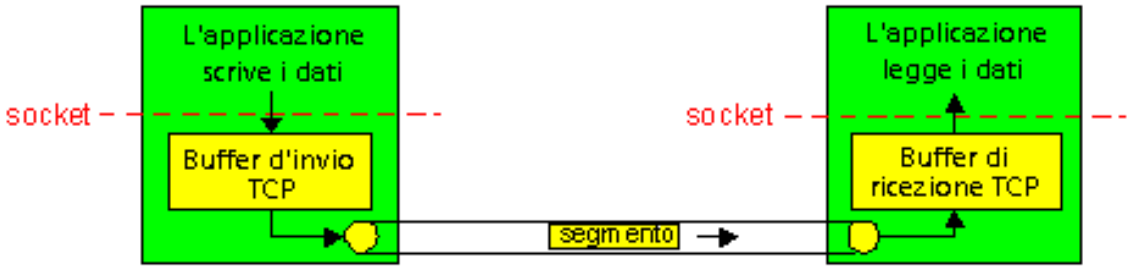
Livello di trasporto

- Principi del trasferimento dati affidabile
- Trasporto orientato alla connessione: TCP
  - struttura dei segmenti
  - trasferimento dati affidabile
  - controllo di flusso
  - gestione della connessione

TCP: Panoramica

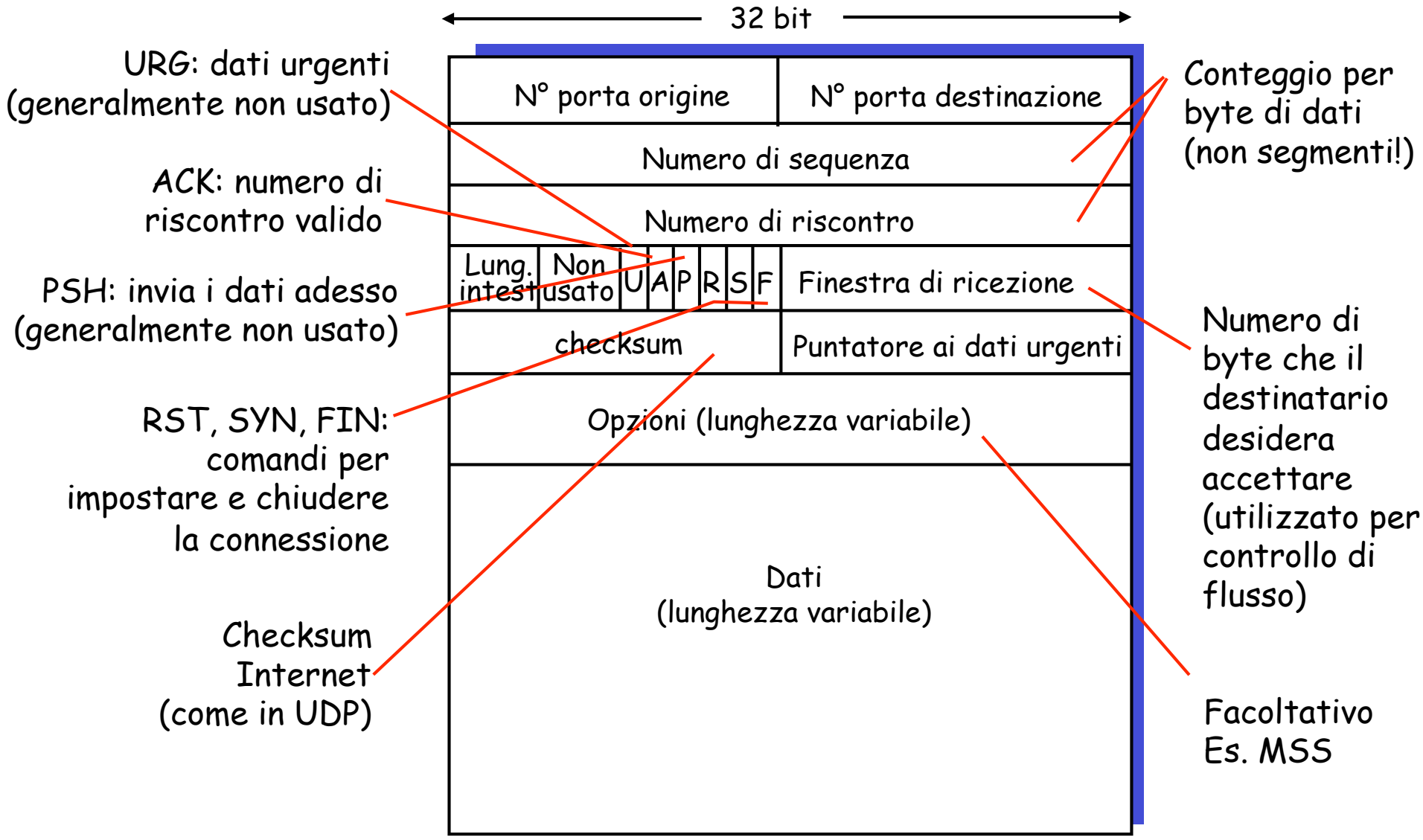
RFC: 793, 1122, 1323, 2018, 2581

- TRANSMISSION CONTROL PROTOCOL
- 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

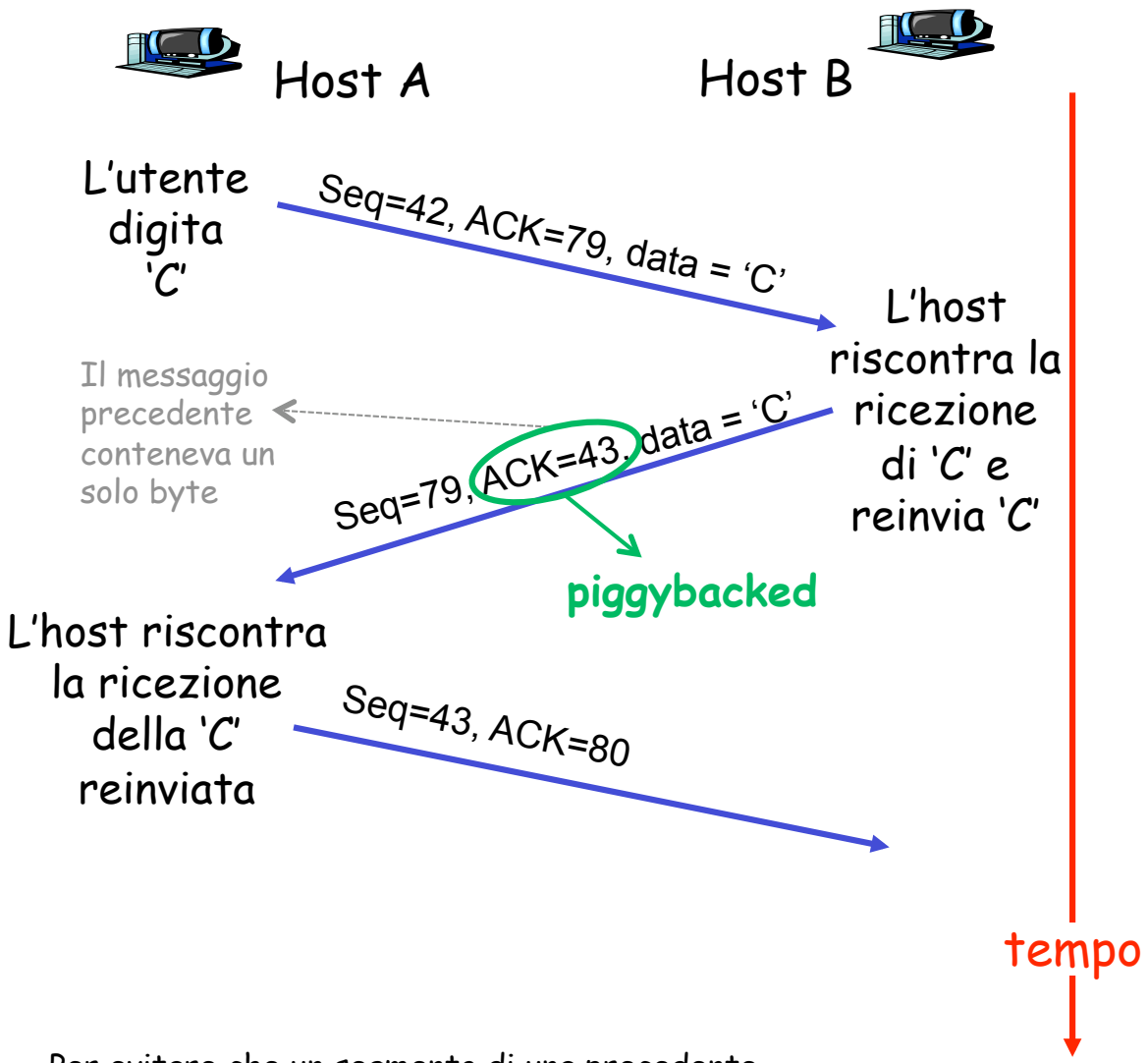
ACK:

- 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 – solitamente il destinatario mantiene i byte non ordinati

Una semplice applicazione Telnet



N.B. Il numero di sequenza iniziale è scelto a caso → Per evitare che un segmento di una precedente connessione ancora presente in rete possa essere interpretato come valido per la nuova connessione

TCP: tempo di andata e ritorno e timeout

D: come impostare il valore del timeout di TCP?

- Più grande del tempo di andata e ritorno della connessione (RTT)
  - ma RTT varia
- Troppo piccolo: timeout prematuro
  - ritrasmissioni non necessarie
- Troppo grande: reazione lenta alla perdita dei segmenti

D: come stimare RTT?

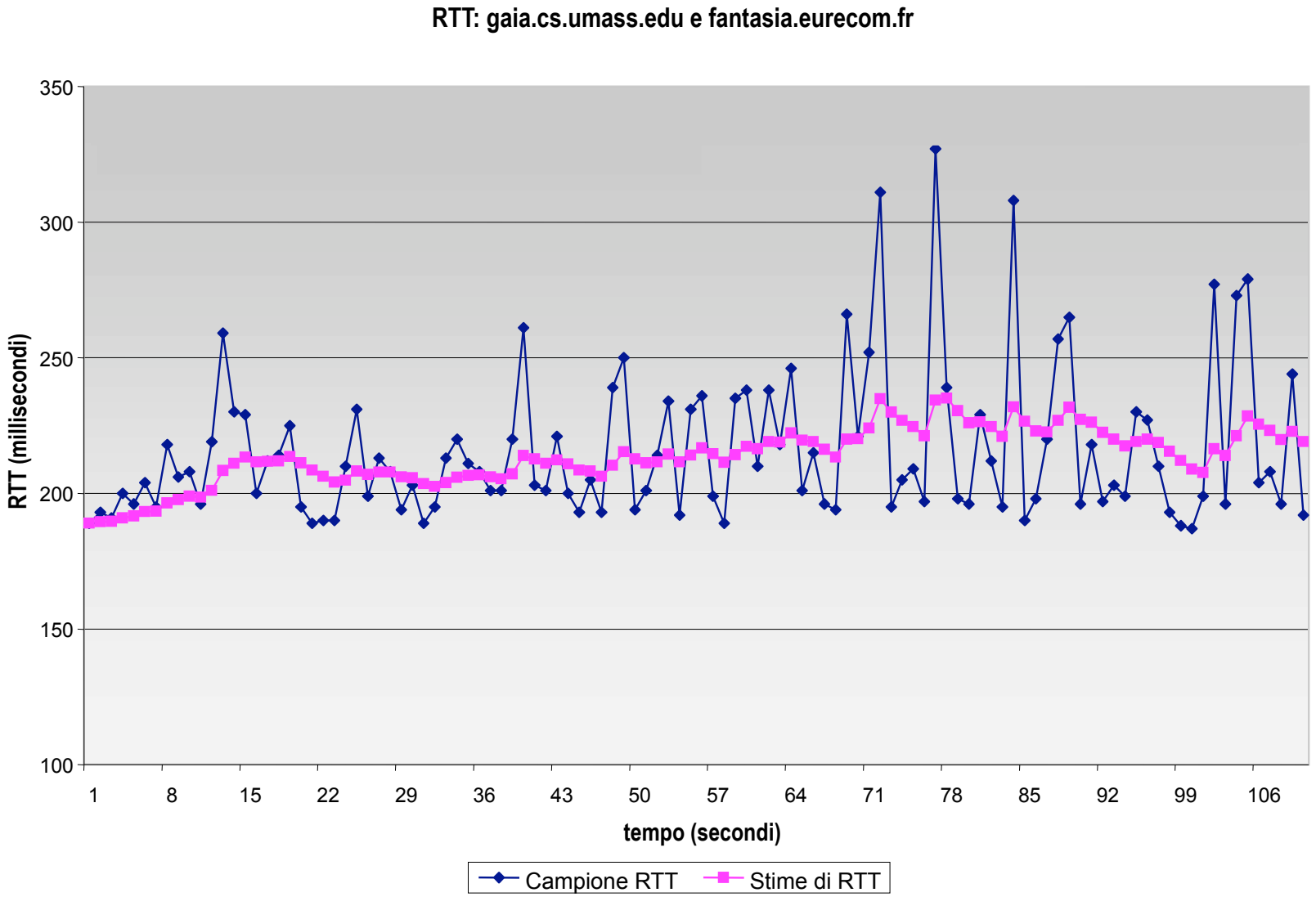
- **SampleRTT**: tempo misurato dalla trasmissione del segmento fino alla ricezione di ACK
  - ignora le ritrasmissioni
  - Un solo **SampleRTT** per più segmenti trasmessi insieme
- **SampleRTT** varia a causa di congestione nei router e carico nei sistemi terminali, quindi occorre una stima "più livellata" di RTT
  - media di più misure recenti, non semplicemente il valore corrente di **SampleRTT**

TCP: tempo di andata e ritorno e timeout

$$\mathbf{EstimatedRTT_{t+1} = (1 - \alpha)*EstimatedRTT_t + \alpha*SampleRTT_{t+1}}$$

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

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

Livello di trasporto

- Principi del trasferimento dati affidabile
- Trasporto orientato alla connessione: TCP
  - struttura dei segmenti
  - trasferimento dati affidabile
  - controllo di flusso
  - gestione della connessione

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)
- Intervallo di scadenza: `TimeOutInterval` (calcolato in termini di `EstimtedRTT` **e** `DevRTT`)

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

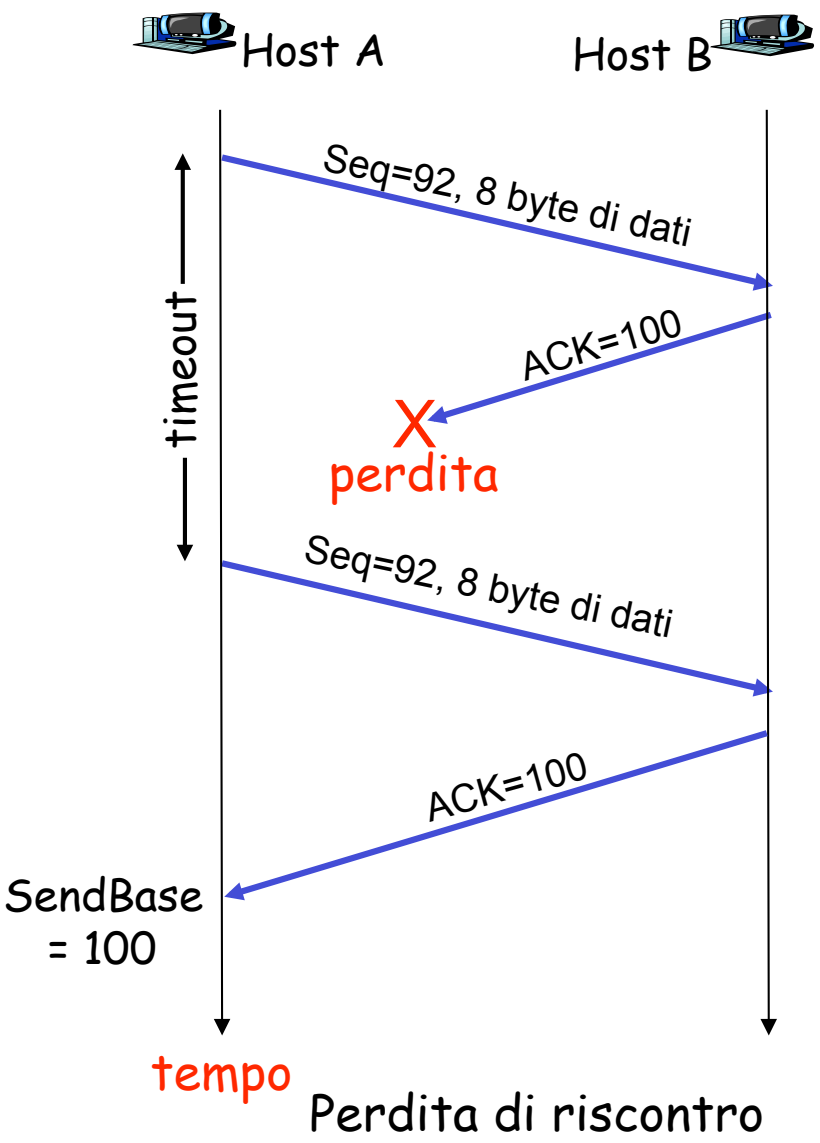
```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

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 attivo)
            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)

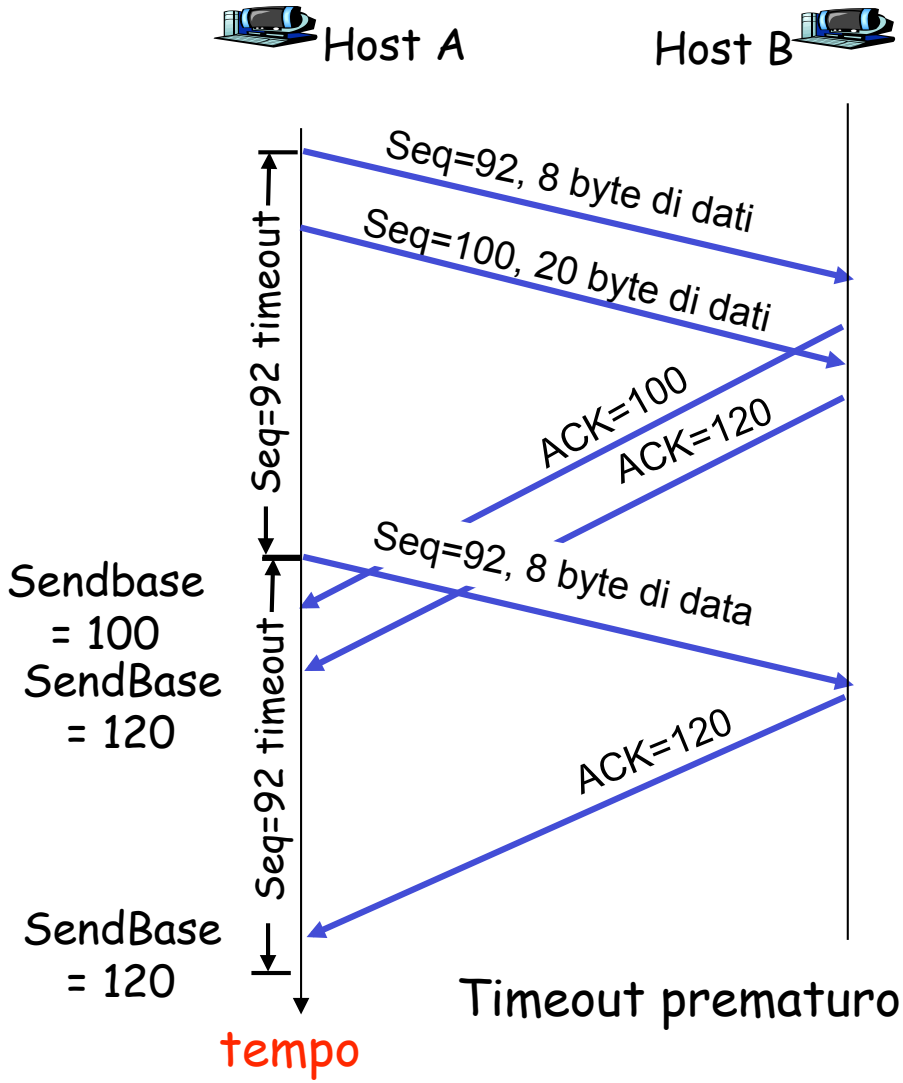
Commento:
- SendBase-1: ultimo byte cumulativamente riscontrato

Esempio:
- SendBase-1 = 71; y = 73, quindi il destinatario vuole 73+ ; y > SendBase, allora vengono riscontrati tali nuovi dati

TCP: scenari di ritrasmissione

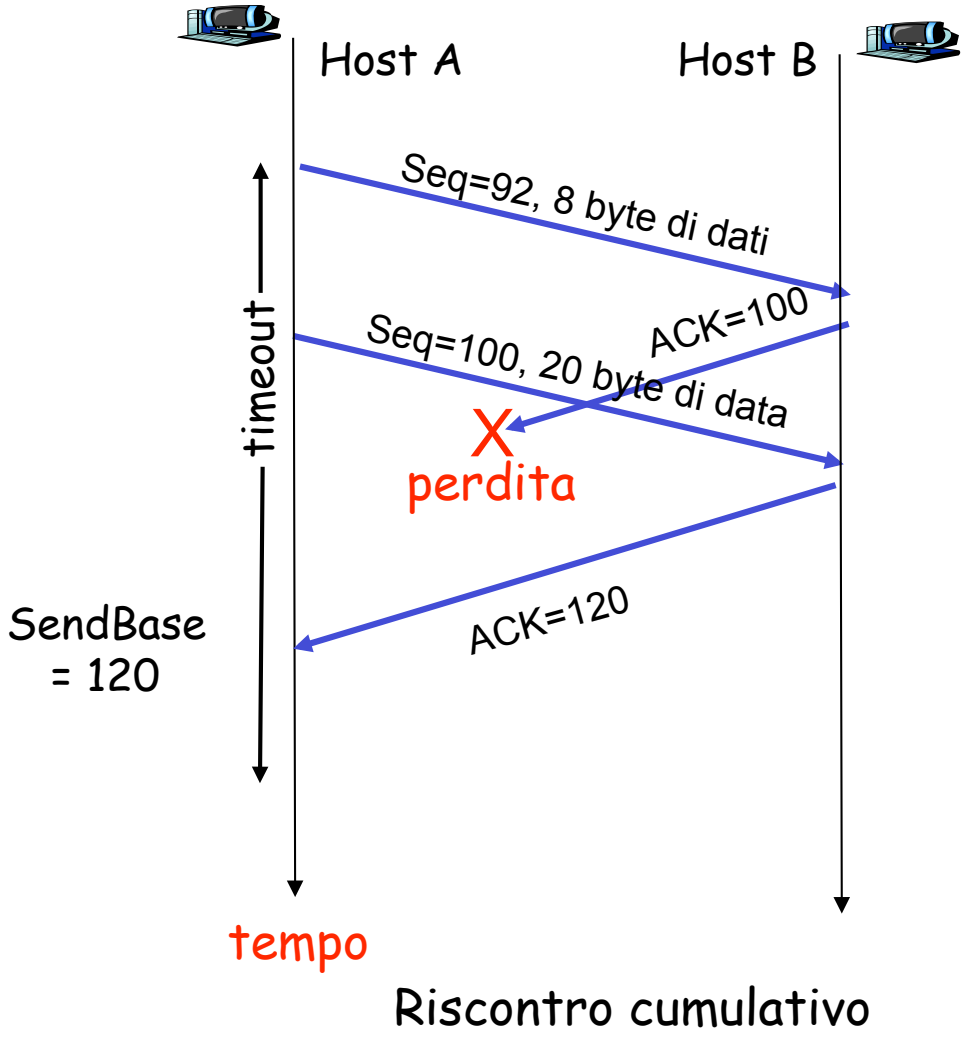


Perdita di riscontro



Timeout prematuro

TCP: scenari di ritrasmissione



Riscontro cumulativo

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

Evento presso il destinatario	Azione del ricevente 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 delayed. 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 (vedi precedente).	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.

Ritrasmissione rapida

- 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.

Esistono segmenti non riscontrati quindi avvia timer

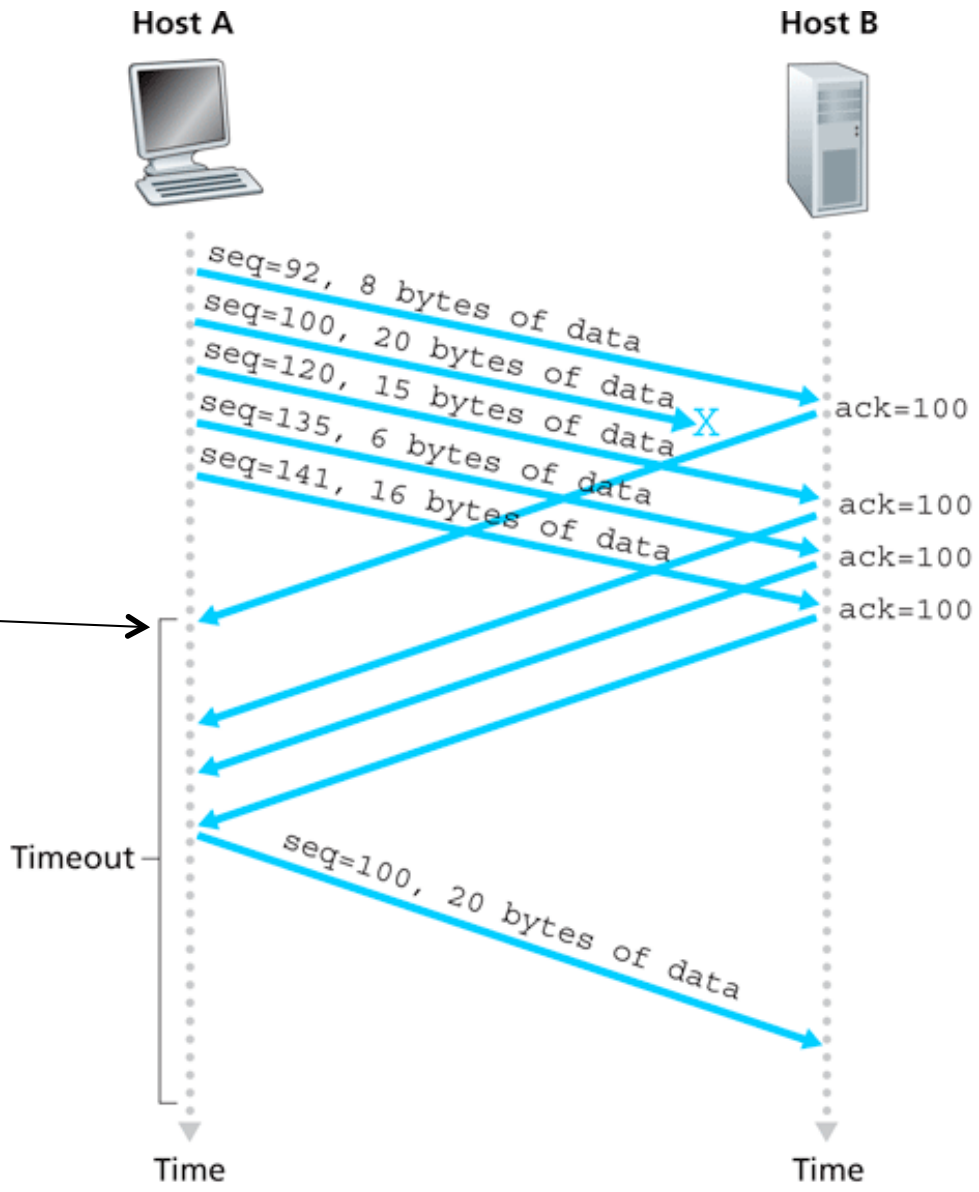


Figure 3.37 ♦ Fast retransmit: retransmitting the missing segment before the segment's timer expires.

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

Riassunto su meccanismi adottati da TCP

- **Pipeline** (Approccio ibrido tra GBN e Ripetizione Selettiva)
- **Numero di sequenza**: primo byte nel segmento
- ***ACK cumulativo*** (conferma tutti i byte precedenti a quello indicato) e **delayed** (ritardato, nel caso di arrivo di un pacchetto in sequenza, con precedenti già riscontrati)
- **Timeout** basato su RTT: **unico** timer di ritrasmissione (associato al più vecchio segmento non riscontrato). Quando arriva una notifica intermedia, si riavvia il timer sul più vecchio segmento non riscontrato
- **Ritrasmissione**
  - **Singola**: solo il segmento **non riscontrato** (non i successivi)
  - **Rapida**: al 3 ACK duplicato prima del timeout → ritrasmissione