

**Corso di Laurea in Ingegneria delle Telecomunicazioni**



**Corso di Reti di Calcolatori**

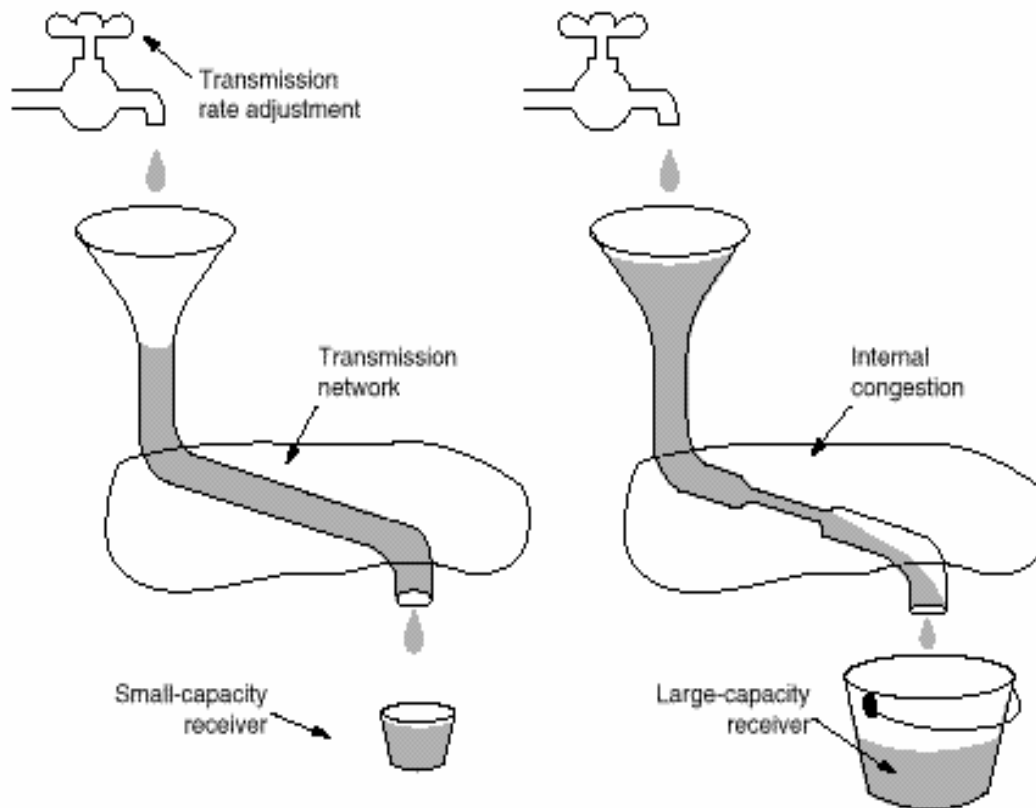
**Docente: Simon Pietro Romano**  
**spromano@unina.it**

---

**Il livello trasporto:**

**TCP: controllo di flusso e controllo della congestione**

# TCP: Controllo di Flusso e di Congestione



## Come gestire entrambi i tipi di controllo?



- *Receiver window*: dipende dalla dimensione del buffer di ricezione
- *Congestion window*: basata su una stima della capacità della rete



I byte trasmessi corrispondono alla dimensione della finestra più piccola



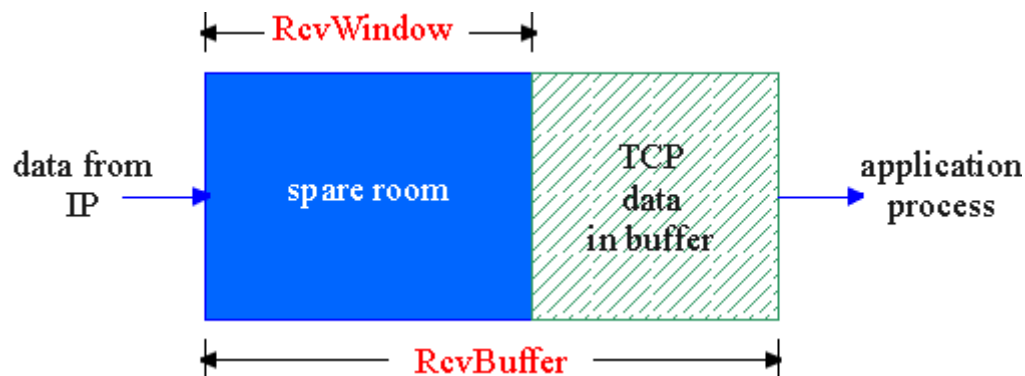
# TCP Flow Control

## flow control

Il mittente non dovrà sovraccaricare il ricevente inviando dati ad una velocità troppo elevata

`RcvBuffer` = size of TCP Receive Buffer

`RcvWindow` = amount of spare room in Buffer



receiver buffering

**ricevente:** comunica dinamicamente al mittente la dimensione corrente del buffer

- campo `RcvWindow` nel segmento TCP:

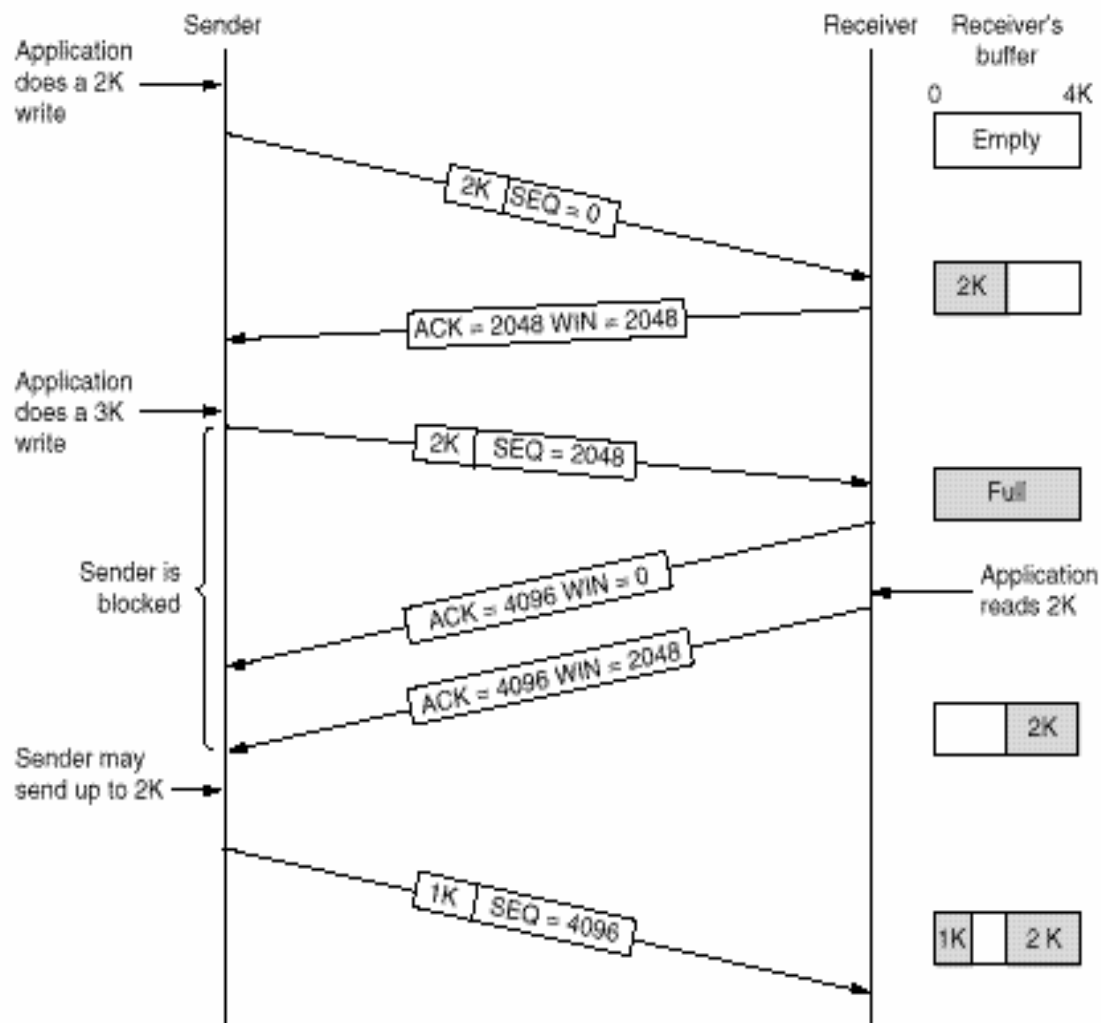
$$\text{RcvWindow} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$$

**mittente:** conserva i dati già trasmessi, ma non riscontrati, meno la quantità pari all'ultima `RcvWindow` ricevuta:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{RcvWindow}$$



# TCP: Transmission Policy





## L'algoritmo di Nagle

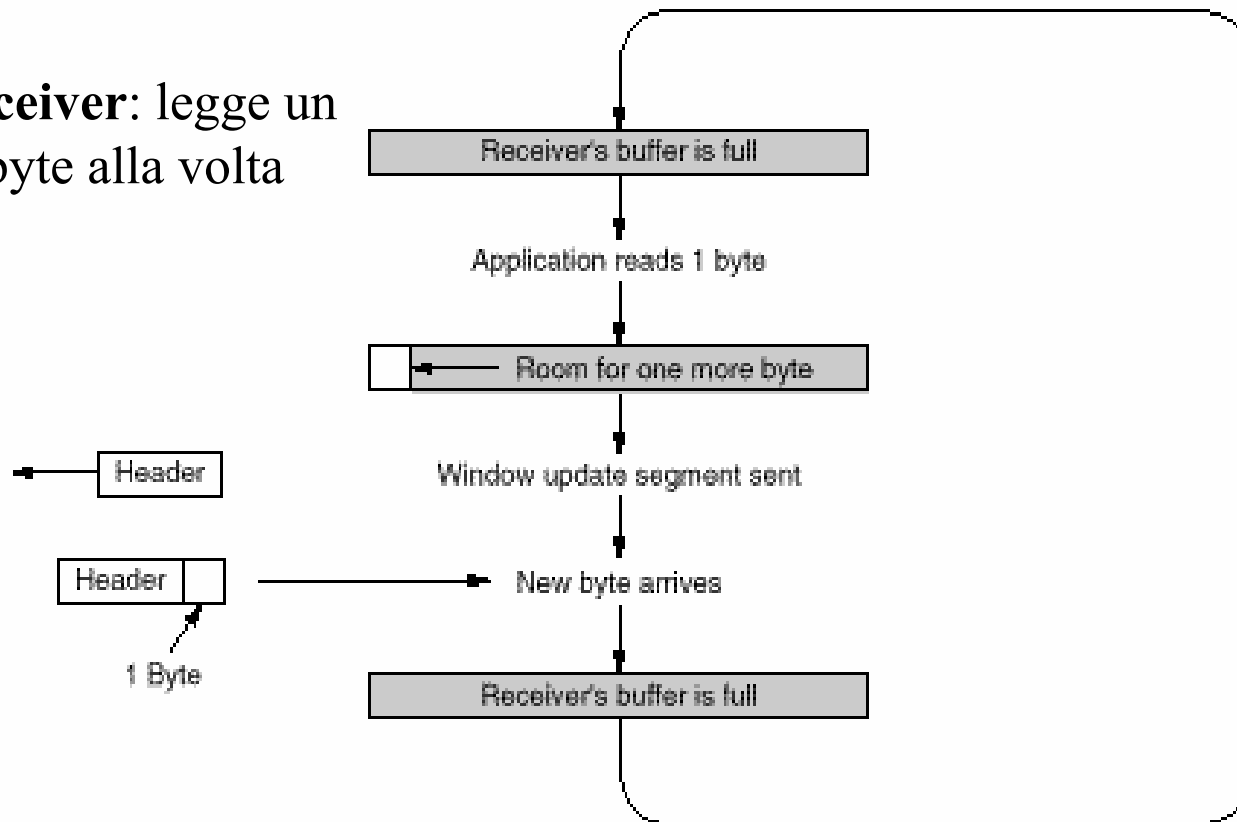
- Per applicazioni che inviano dati un byte alla volta (es: TELNET):
  - invia il primo byte e bufferizza il resto finché non giunge l'ACK
  - in seguito:
    - invia, in un unico segmento, tutti i caratteri bufferizzati
    - ricomincia a bufferizzare finché non giunge l'ACK per ognuno di essi



# La sindrome della Silly Window

**Sender:** invia blocchi grandi

**Receiver:** legge un byte alla volta



**Soluzione di Clark:**  
Impedisce al receiver di aggiornare la finestra un byte alla volta



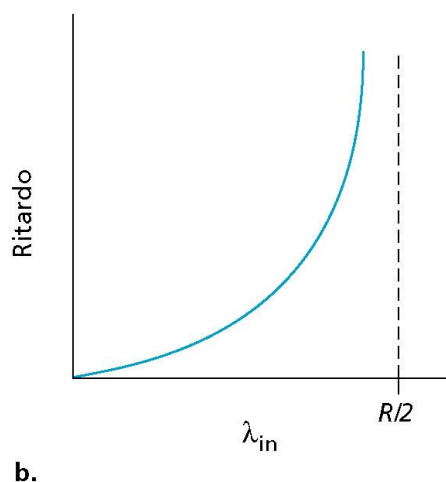
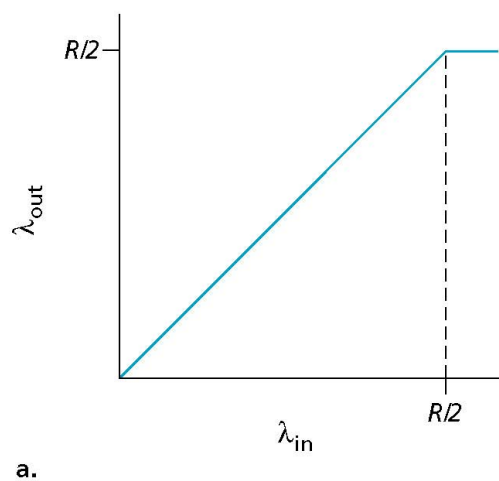
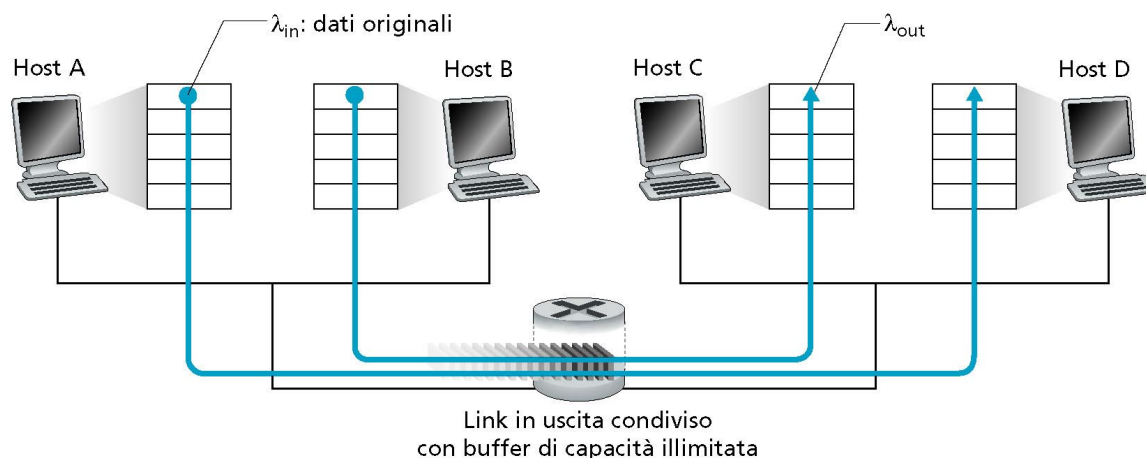
# Controllo della congestione

- **Congestione nella rete**
  - Tecnicamente dovuta a:
    - un numero elevato di sorgenti di traffico
    - sorgenti di traffico che inviano troppi dati
    - traffico inviato ad una frequenza troppo elevata
  - In presenza di questi fenomeni, singoli o concomitanti, la rete è sovraccarica
    - effetti:
      - perdita di pacchetti:
        - » buffer overflow nei router
      - ritardi nell'inoltro dei pacchetti:
        - » accodamenti nei buffer dei router



# Effetti della congestione: esempi (1/2)

- 2 mittenti
- 2 riceventi
- 1 router con  
buffer (coda)  $\infty$ :
  - non ci sono  
ritrasmissioni



- I ritardi aumentano all'avvicinarsi del limite di capacità del canale
- Non si può superare il max throughput



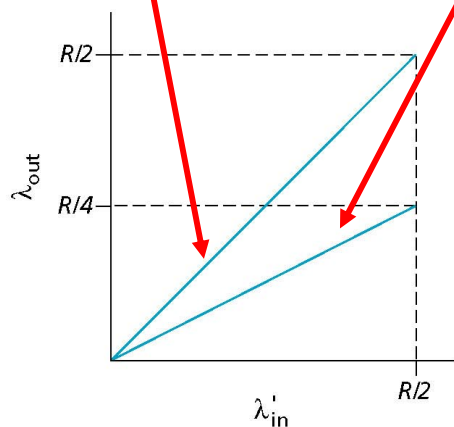
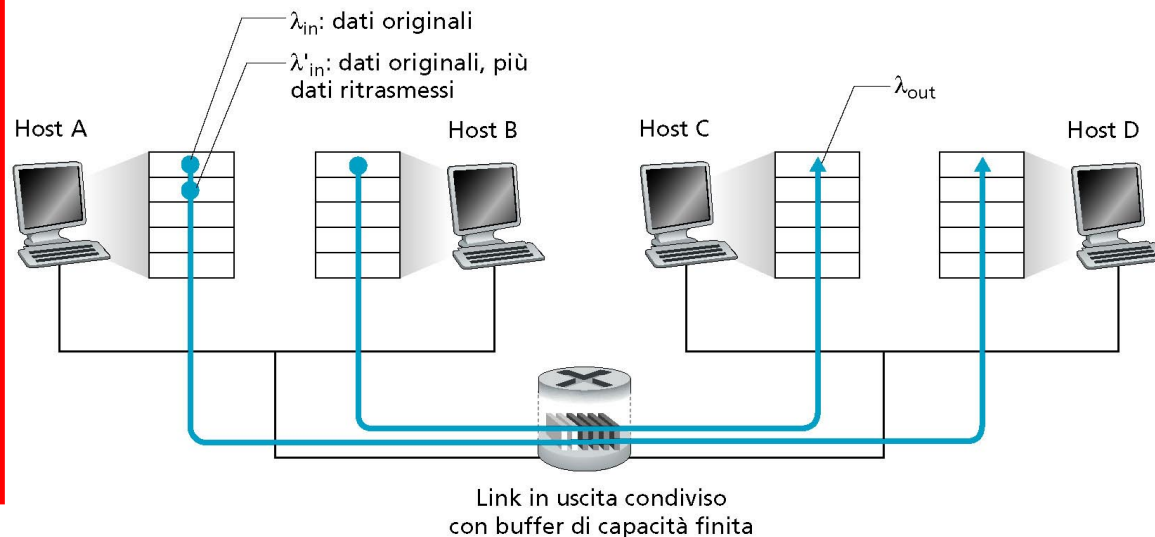
# Effetti della congestione: esempi (2/2)

Il mittente invia dati solo quando il buffer non è pieno:

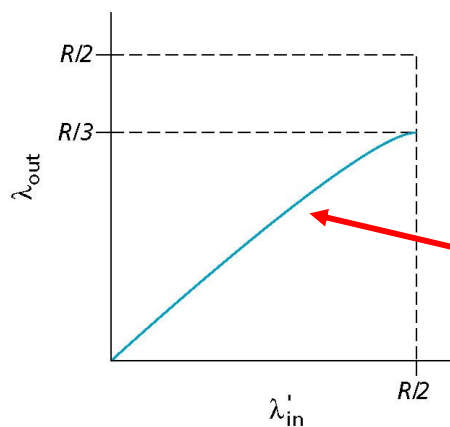
- caso ideale
  - ✓ no ritrasmissioni
  - ✓ throughput max =  $R/2$

Scadenza prematura del timer del mittente:

- ✓ es: ogni segmento è spedito, in media, due volte
- ✓ throughput max =  $R/4$



a.



b.

Il mittente rispedisce un segmento solo quando è sicuro che sia andato perso:

- ✓ il throughput effettivo è inferiore al carico offerto (trasmissioni dati originali + ritrasmissioni)
- ✓ es: curva in figura

# Tecniche di Controllo della Congestione



## Approccio end-to-end:

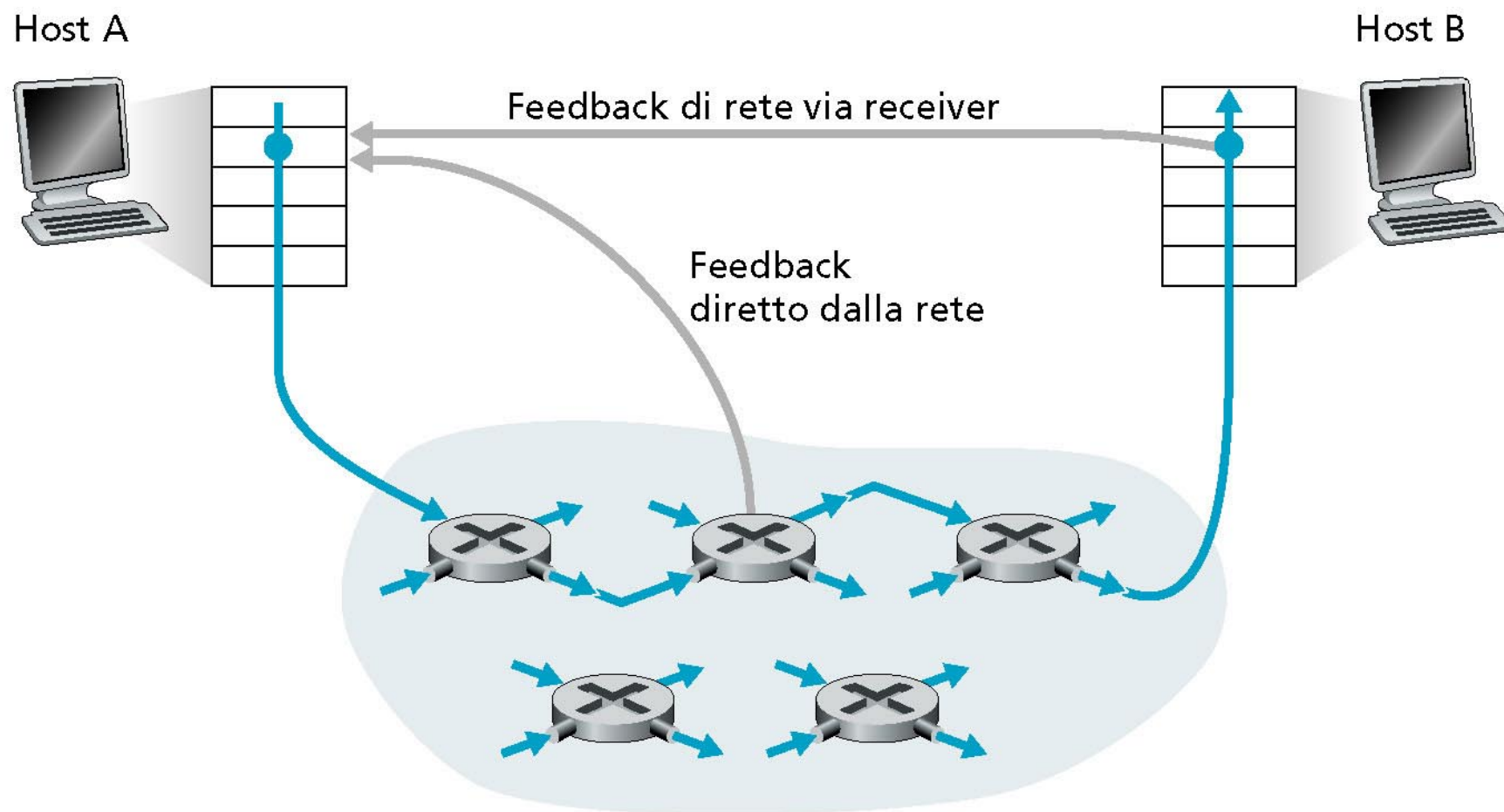
- Nessuna segnalazione esplicita dalla rete
- A partire dall'osservazione di ritardi e perdite di pacchetti gli end-system deducono uno stato di congestione nella rete
- Approccio utilizzato da TCP

## Approccio in base a segnalazione della rete:

- I router forniscono informazioni circa lo stato della rete agli end-system:
  - l'invio di un singolo bit indica lo stato di congestione
    - SNA, DECbit, TCP/IP ECN, ATM
  - in alternativa, il sender è informato circa la massima frequenza alla quale può trasmettere



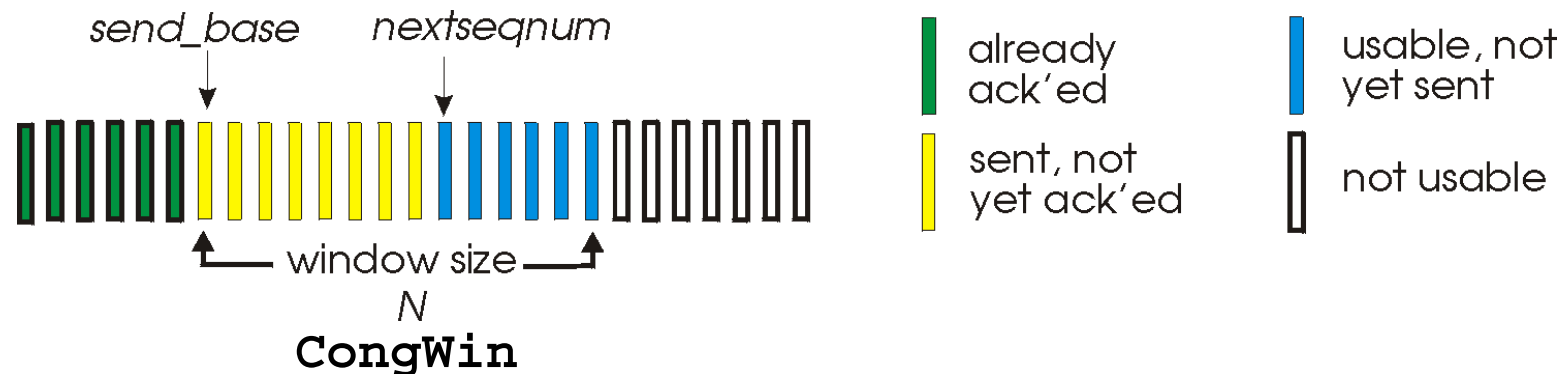
# Feedback di rete: tecniche alternative





# Controllo della congestione in TCP

- Controllo end-to-end: nessun feedback dalla rete
- Frequenza di trasmissione variabile:
  - dipendente dalla cosiddetta *finestra di congestione*



**Considerando controllo di flusso e controllo di congestione insieme, si ha, dunque:**

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{RcvWindow}, \text{CongWin}\}$$

# Controllo della congestione: idea di base



- Si procede **per tentativi**, per stabilire quanto si può trasmettere:
  - **obiettivo:**
    - trasmettere alla massima velocità possibile (`Congwin` quanto più grande possibile) senza perdite
  - **approccio utilizzato:**
    - incrementare `Congwin` finchè non si verifica la perdita di un segmento (interpretata come il sopraggiungere dello stato di congestione)
    - in seguito alla perdita di un segmento:
      - decrementare `Congwin`
      - ricominciare daccapo



# Controllo della congestione: fasi

- *Slow Start*
  - Partenza lenta (per modo di dire!)
- **Congestion Avoidance:**
  - Additive Increase, Multiplicative Decrease (AIMD)
    - incremento additivo, decremento moltiplicativo

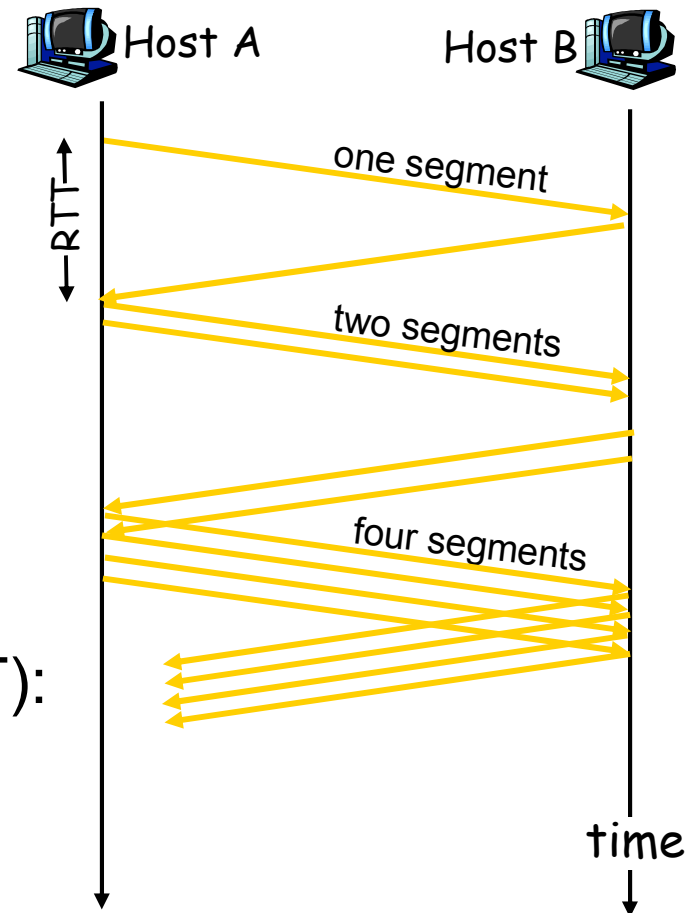


# Lo Slow Start in TCP

## Algoritmo Slowstart

```
//initialization  
Congwin = 1  
  
for (each segment ACKed)  
    Congwin++  
until (loss event OR  
      CongWin > threshold)
```

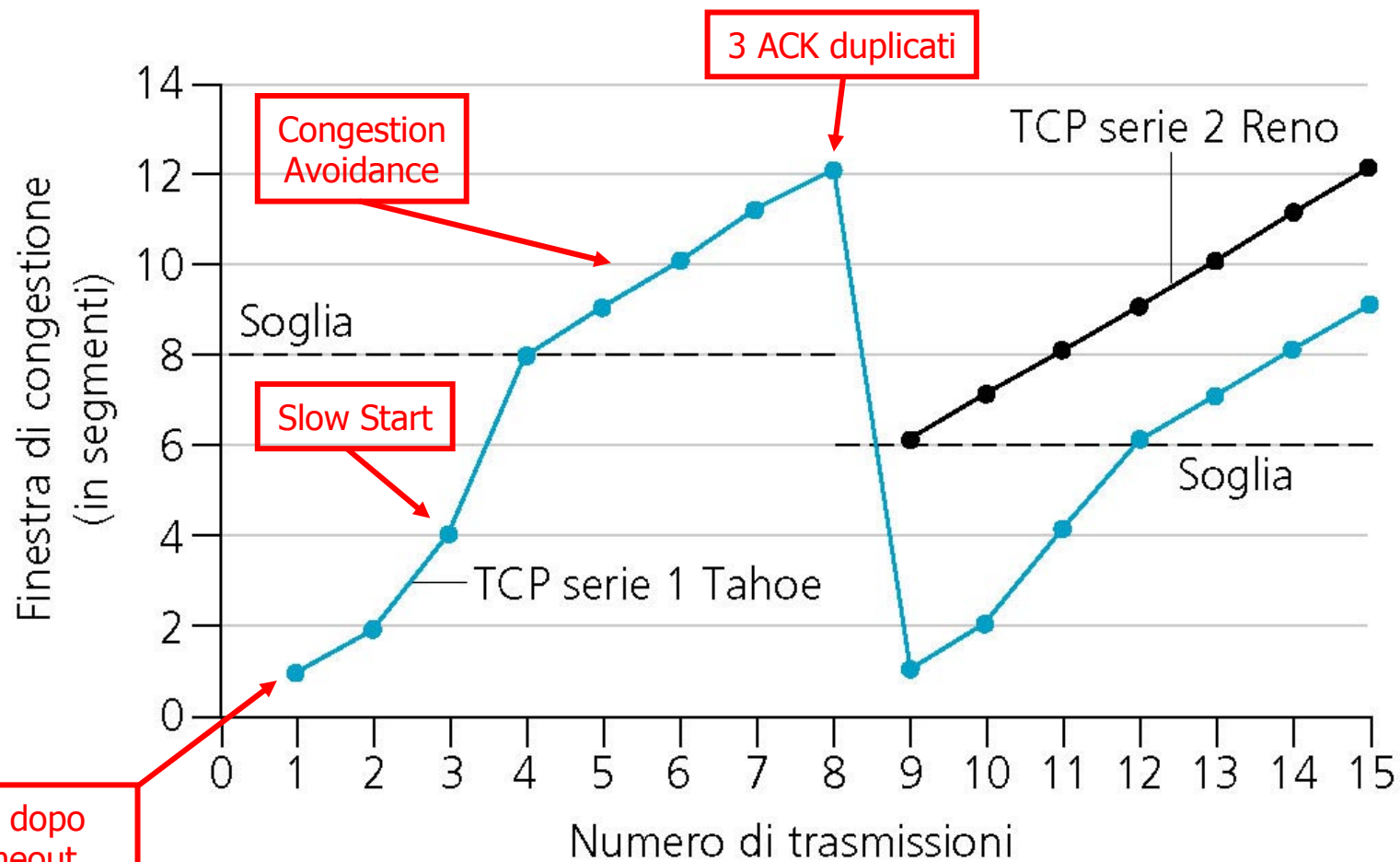
- Crescita esponenziale della dimensione della finestra (ogni RTT):
  - “Slow start” → termine improprio!
- Evento di *perdita*:
  - timeout
  - tre ACK duplicati consecutivi





# Controllo della congestione in Internet

È presente un ulteriore parametro: **soglia (threshold)**



Subito dopo un timeout (CongWin = 16)

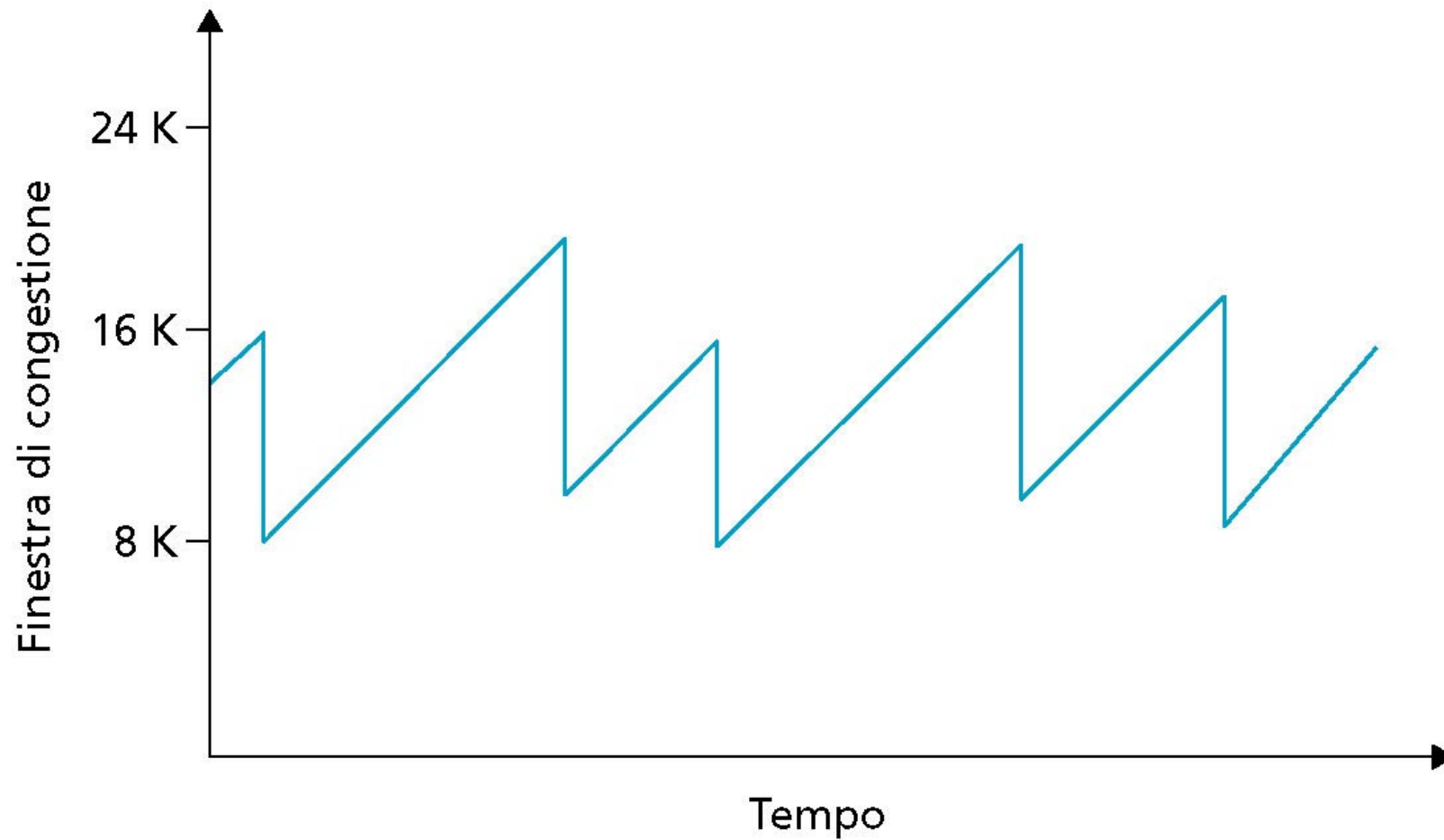


## Dopo lo slow start: AIMD

- **Additive Increase**
  - Una volta raggiunta la soglia:
    - ci si avvicina con cautela al valore della banda disponibile tra le due estremità della connessione
    - meccanismo adottato:
      - incremento di  $CongWin$  alla ricezione di un ACK
    - questa fase è nota come fase di *congestion avoidance*
- **Multiplicative Decrease:**
  - Al sopraggiungere della congestione (scadenza di un timeout o ricezione di tre ACK duplicati consecutivi):
    - la finestra di congestione viene dimezzata



# AIMD: andamento a “dente di sega”





# Ricapitolando...

- Finestra di congestione sotto la soglia:
  - Slow start
  - Crescita esponenziale della finestra
- Finestra di congestione sopra la soglia:
  - Prevenzione della congestione
  - Crescita lineare della finestra
- Evento di perdita dedotto da ACK duplicato 3 volte:
  - Soglia posta alla metà del valore attuale della finestra
  - **TCP Reno:**
    - Finestra posta pari alla soglia
  - **TCP Tahoe:**
    - Finestra posta pari ad un segmento (MSS -- Maximum Segment Size)
- Evento di perdita dedotto da timeout:
  - Soglia posta alla metà del valore attuale della finestra
  - Finestra posta pari ad un segmento (MSS -- Maximum Segment Size)



## TCP Reno: “*fast recovery*”

- TCP Reno elimina la fase di partenza lenta dopo un evento di perdita dedotto dalla ricezione di tre ACK duplicati:
  - tale evento indica che, nonostante si sia perso un pacchetto, alcuni segmenti sono stati ricevuti dal destinatario:
    - a differenza del caso di timeout, la rete mostra di essere in grado di consegnare una certa quantità di dati
    - è possibile, quindi, evitare una nuova partenza lenta, ricominciando direttamente dalla fase di prevenzione della congestione



# Equità tra le connessioni TCP

- Ipotesi:
  - MSS e RTT uguali per le due connessioni:
    - a parità di dimensioni della finestra, il throughput è lo stesso
  - Entrambe le connessioni si trovano oltre lo slow start:
    - fase di prevenzione della congestione:
      - incremento additivo
      - decremento moltiplicativo

