TCP

Controllo di flusso

TCP: controllo di flusso

- Ogni host imposta buffer di invio e di ricezione
- Il processo applicativo destinatario legge i dati dal buffer di ricezione (non necessariamente nell'istante in cui arrivano)

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

TCP: controllo di flusso

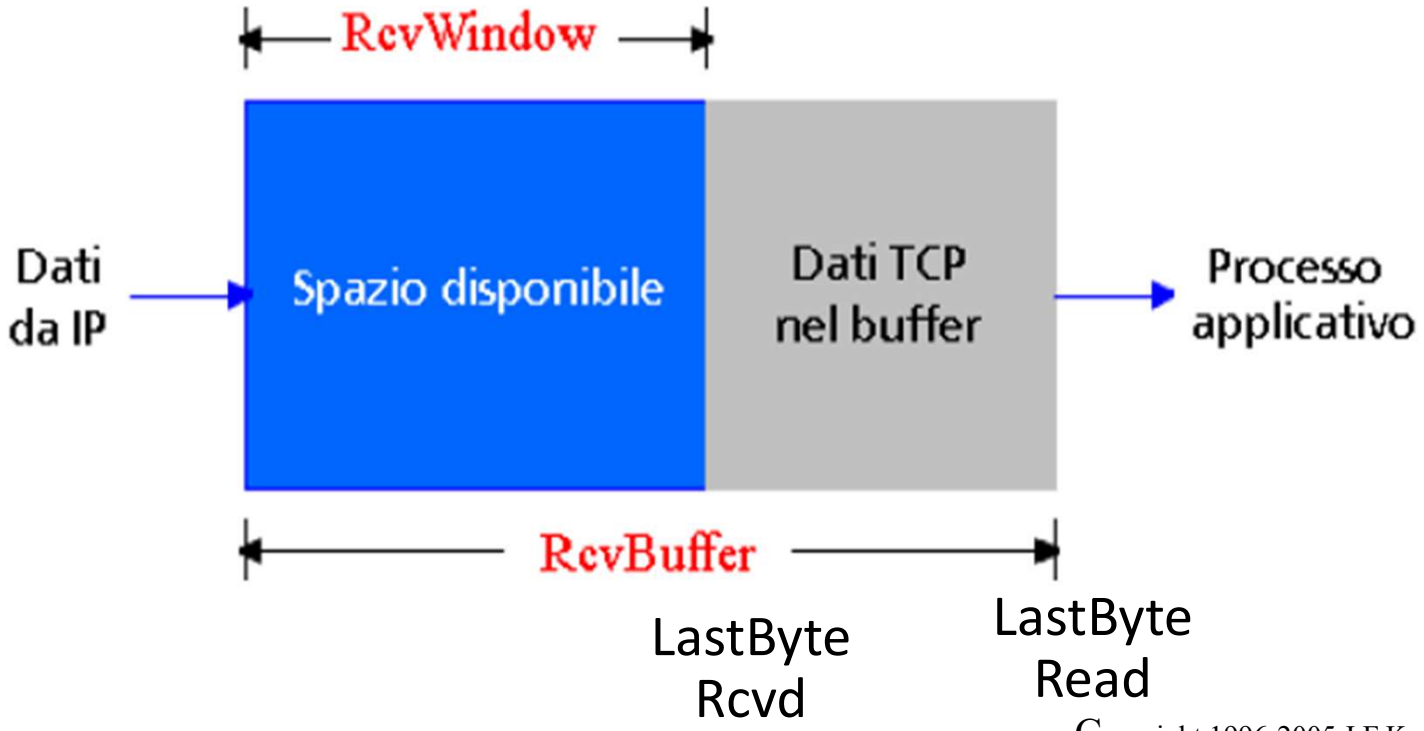
- Ogni host imposta buffer di invio e di ricezione
- Il processo applicativo destinatario legge i dati dal buffer di ricezione (non necessariamente nell'istante in cui arrivano)

Si intende con controllo di flusso la capacità del mittente di evitare la possibilità di saturare il buffer del ricevitore.

Il controllo di flusso mette in relazione la frequenza di invio del mittente con la frequenza di lettura dell'applicazione ricevente

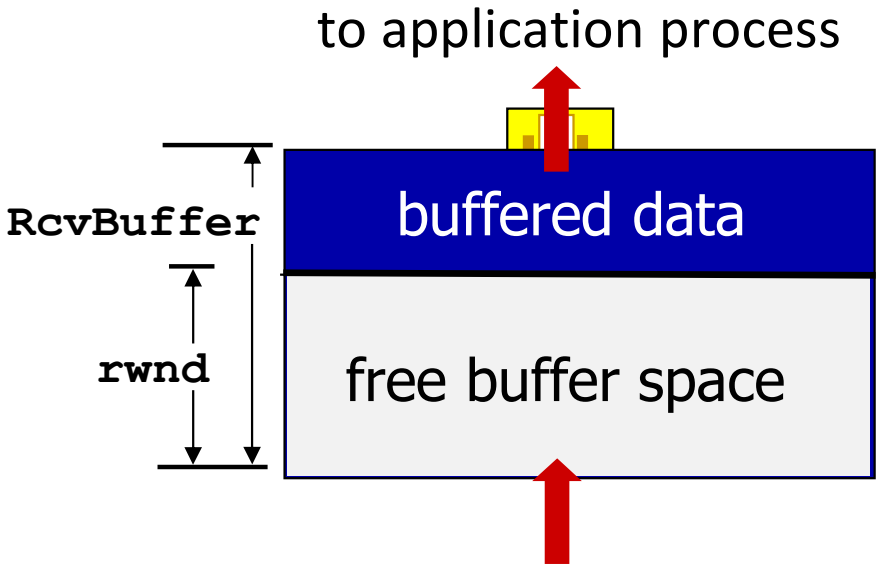
TCP: controllo di flusso (2)

- TCP implementa questa caratteristica tramite una variabile detta finestra di ricezione (*rwnd)* mantenuta nel mittente: questa variabile fornisce un'idea di quanto spazio è ancora a disposizione nel buffer del ricevitore.



TCP: controllo di flusso (3)

- Il destinario TCP comunica la capacità residua nel buffer di recezione tramite il campo **`rwnd`** field nell'header TCP
  - **`Dimensione di RcvBuffer`** configurabile tramite socket options (tipicamente il valore di default è 4096 bytes)
- Il mittente limita la quantità di dati unACKed ("in-flight") al valore ricevuto di **`rwnd`**



TCP receiver-side buffering

TCP: controllo di flusso – meccanismi di base

- Spazio disponibile nel buffer del destinatario

  `Rwnd= RcvBuffer-(LastByteReceived-LastByteRead)`

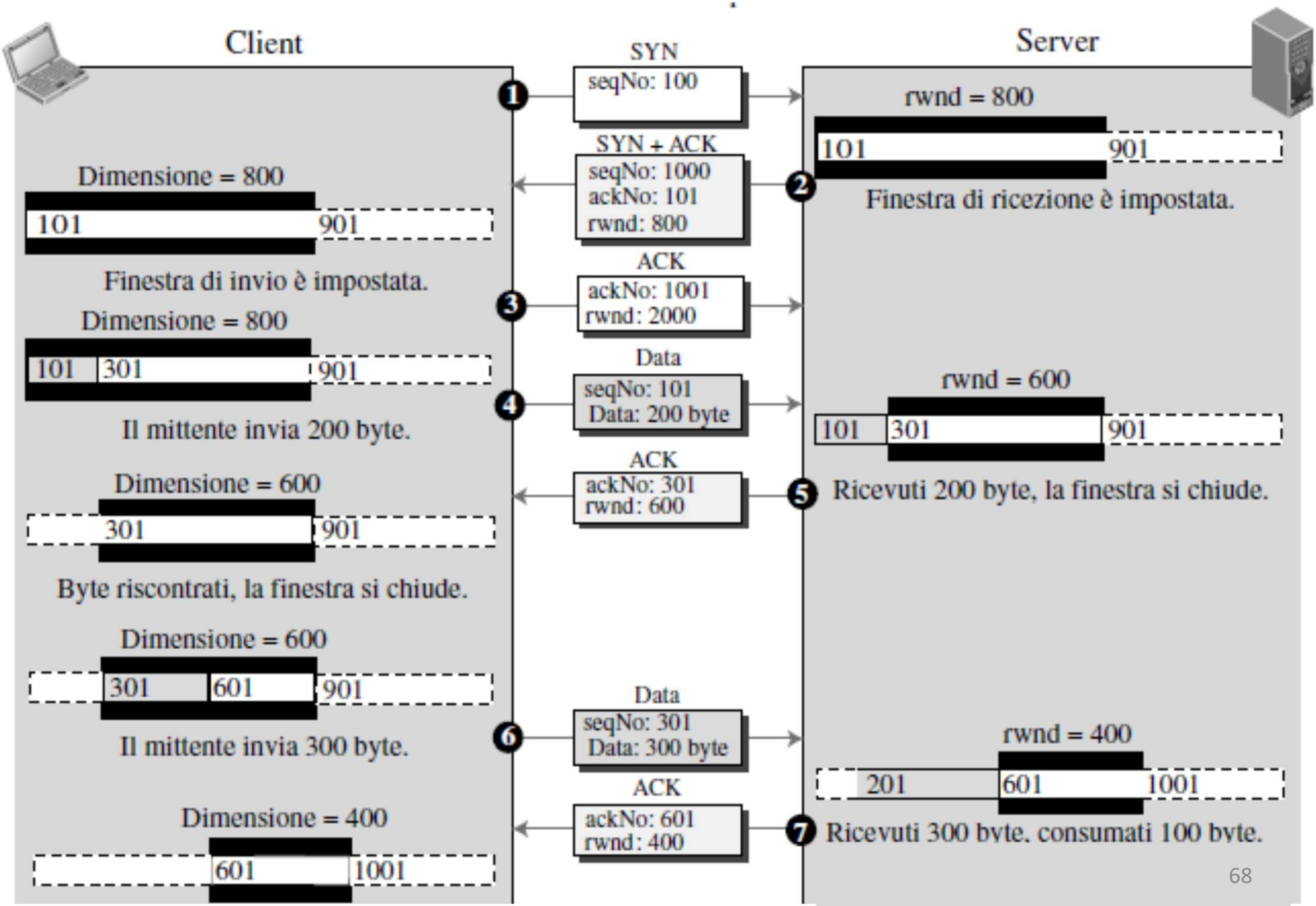
- Rwnd è dinamica
- L'host destinatario comunica la dimensione di rwnd al mittente
- Il mittente si assicura che

  `LastByteSent-LastByteAcked ≤ Rwnd`

  Quantità di dati trasmessi e non ancora riscontrati

  N.B. se rwnd=0, il mittente manda segmenti «sonda» di 1 byte per ricevere l'aggiornamento sulla dimensione di rwnd

Esempio


Client
Server
SYN
seqNo: 100
rwnd = 800
101
901
SYN + ACK
seqNo: 1000
ackNo: 101
rwnd: 800
Dimensione = 800
101
901
Finestra di invio è impostata.
Finestra di ricezione è impostata.
ACK
ackNo: 1001
rwnd: 2000
Dimensione = 800
101
301
901
Data
seqNo: 101
Data: 200 byte
rwnd = 600
101
301
901
Il mittente invia 200 byte.
ACK
ackNo: 301
rwnd: 600
Dimensione = 600
301
901
Ricevuti 200 byte, la finestra si chiude.
Byte riscontrati, la finestra si chiude.
Dimensione = 600
301
601
901
Data
seqNo: 301
Data: 300 byte
rwnd = 400
201
601
1001
Il mittente invia 300 byte.
ACK
ackNo: 601
rwnd: 400
Dimensione = 400
601
1001
Ricevuti 300 byte, consumati 100 byte.

Risorse online

- Controllo di flusso

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/flow-control/index.html

TCP

Controllo di congestione

TCP: controllo della congestione(RFC 2581)

congestion control:
too many senders,
sending too fast

- Il fenomeno della congestione è originato dal tentativo della/e sorgenti di richiedere più banda di quella disponibile sul percorso.
- Troppe sorgenti che trasmettono troppi dati a una velocità elevata per cui la rete non può gestirli.

- Il traffico eccessivo nella rete può provocare:
  - Lunghi ritardi (accodamenti nei buffer dei router)
  - Perdita di pacchetti (overflow nei buffer dei router)

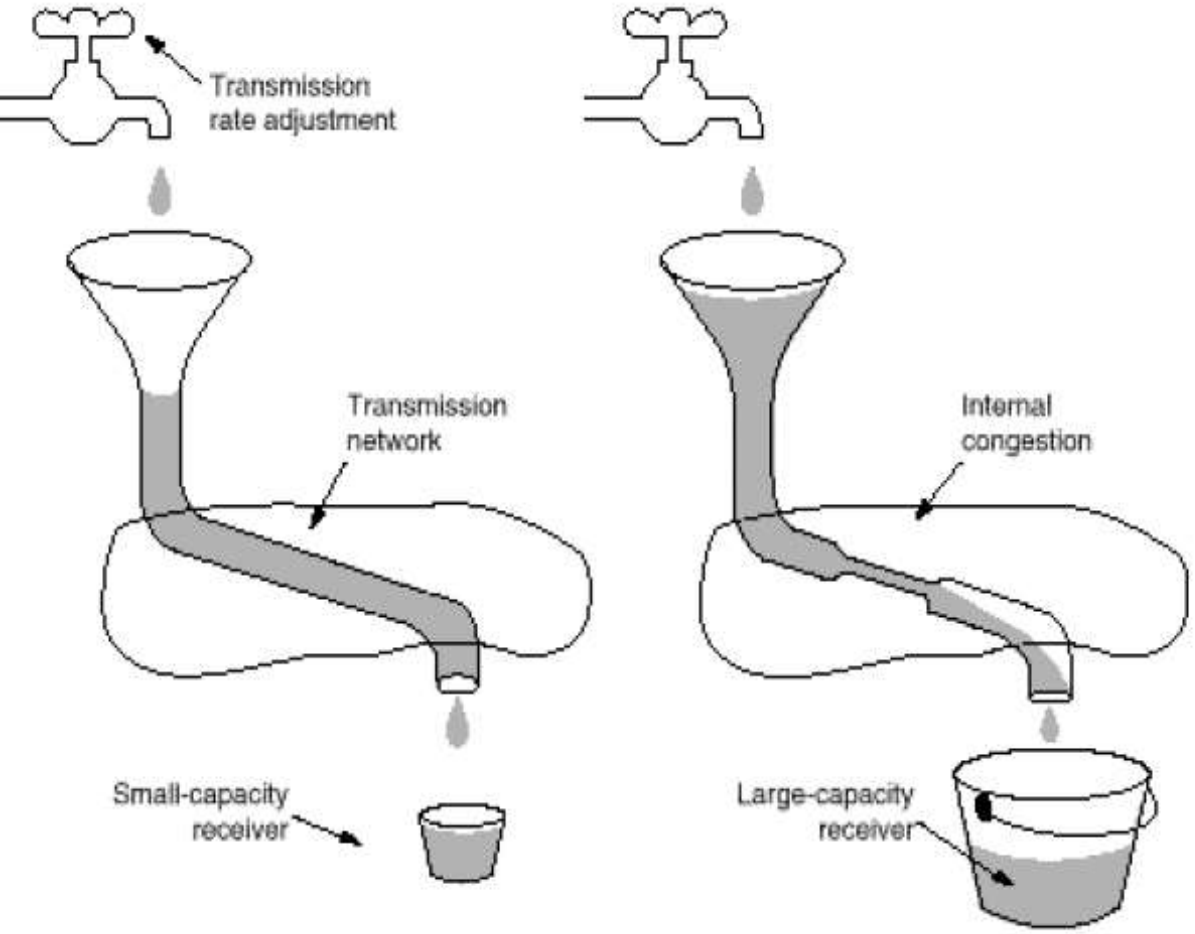
TCP: controllo della congestione

- Meccanismo di base: controllo di congestione punto-punto:
  - Nessun supporto esplicito della rete
  - congestione dedotta dai sistemi terminali

- Come fa il TCP a implementare un controllo di congestione se risiede solo sui dispositivi terminali?

- Il controllo di congestione impone a ciascun mittente di limitare la frequenza di invio di pacchetti sulla connessione, in funzione della congestione percepita
- capacità di TCP di adattarsi alla velocità della rete
  - Se TCP percepisce scarso traffico, aumenta la frequenza di invio altrimenti la diminuisce

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

https://www.docenti.unina.it/downloadPub.do?tipoFile=md&id=90675

NB. dimensione finestra di invio = ***min (rwnd,cwnd)***

Quindi rate di invio non può superare min(rwnd,cwnd)/RTT

Algoritmo per il controllo della congestione

L'algoritmo che il mittente TCP utilizza per regolare la propria frequenza di invio in funzione della congestione rilevata, è costituito da tre passi:

- Partenza lenta (slow start)
- Incremento additivo e decremento moltiplicativo (AIMD)
- Ripresa veloce *(*fast recovery*)*
- Reazione ai time-out

- La finestra di congestione (*cwnd*) impone un vincolo alla frequenza di immissione del traffico sulla rete in base alla congestione percepita.
  - frequenza di invio dei dati non supera *cwnd/RTT*

Congestion Window

- Si misura tipicamente in MSS
- 1 MSS (Maximum Segment Size) è la quantità massima di dati trasportabili da un segmento.
  - Determinato in base alla MTU (unità trasmissiva massima) – lunghezza massima del payload del frame di collegamento inviabile dall'host mittente
  - MSS scelto in modo tale che il segmento TCP, incapsulato in pacchetto IP, stia dentro un singolo di frame di collegamento
  - Esempi tipici di MSS 1460, 536, 512 byte (tolgo dall'MTU 20 byte header TCP + 20 byte header IP)
- RTT (Round Trip Time) è il tempo impiegato da un segmento per effettuare il percorso di andata e ritorno.

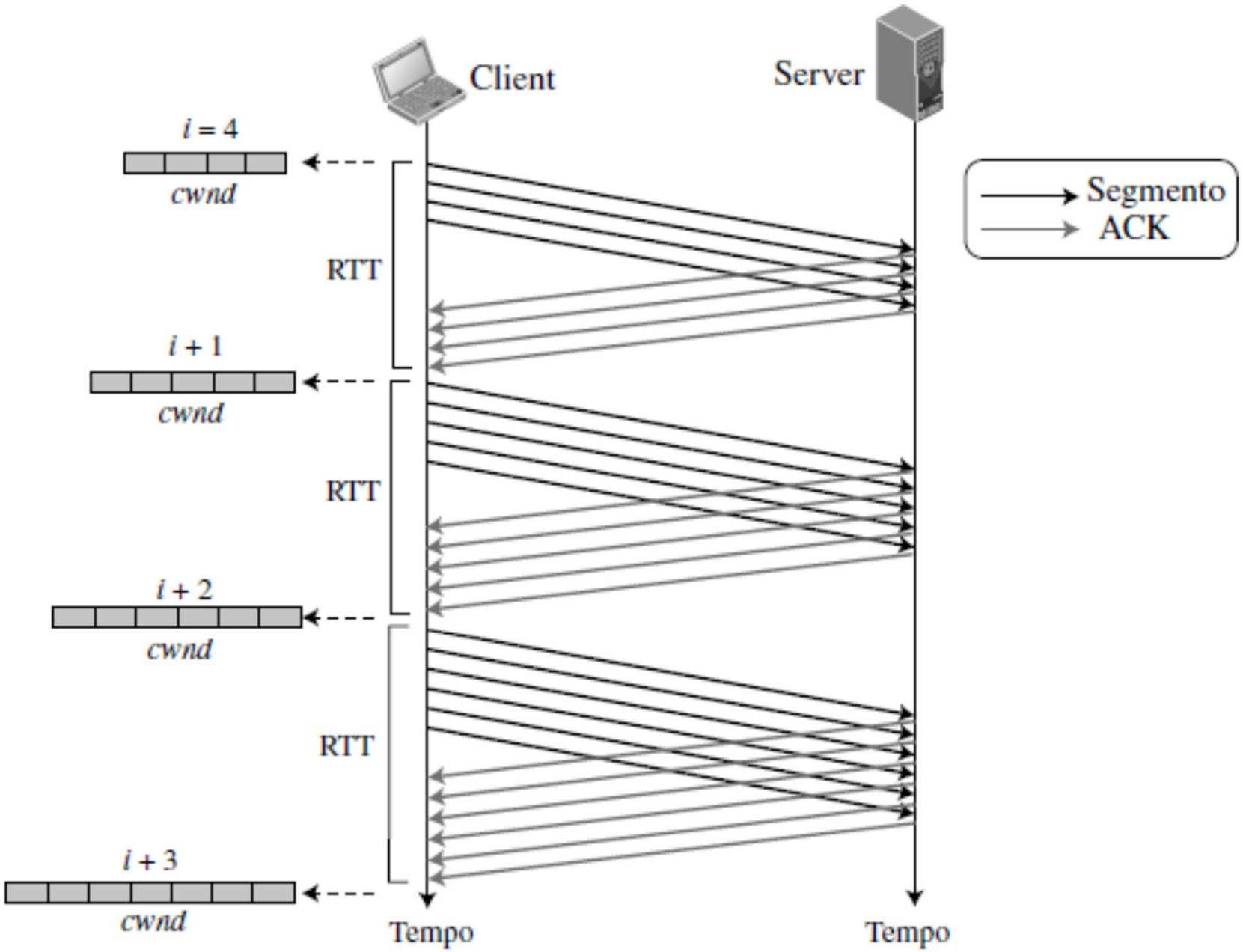
TCP: AIMD

Additive Increase Multiplicative Decrease

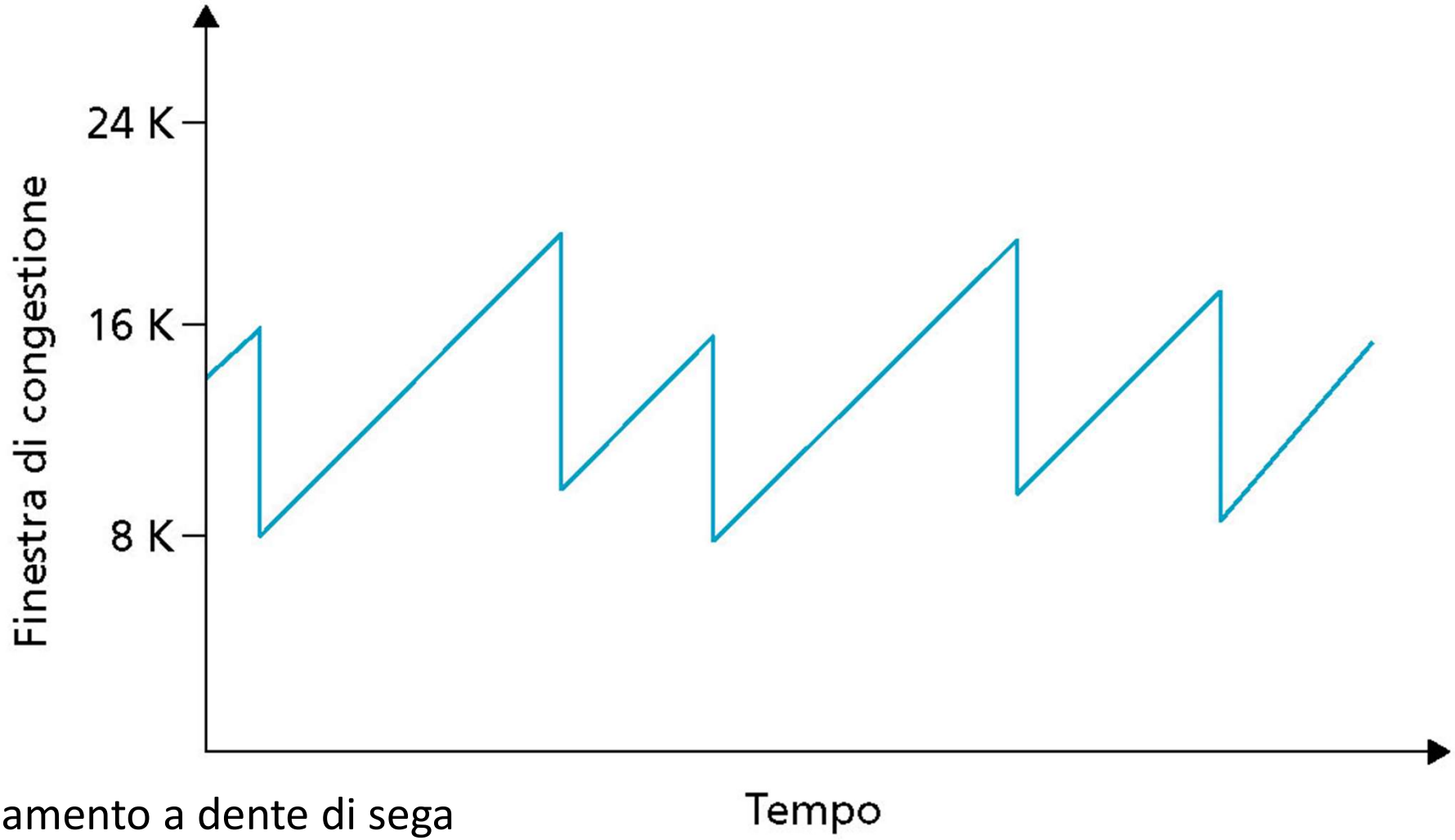
- TCP del mittente aumenta proporzionalmente la propria finestra di congestione ad ogni ACK ricevuto.
  - Di quanto viene incrementata la finestra?
  - Ad ogni ACK la finestra di congestione viene incrementata in modo che si abbia un crescita pari ad 1 MSS per ogni RTT (Congestion avoidance).
    - Ad ogni ACK cwnd= cwnd+1/cwnd
    - Es cwnd = 4 MSS -> incremento di MSS/4
- TCP del mittente dimezza la propria CongWin ad ogni evento di perdita

Vedi implementazioni TCP Reno e Tahoe per reazione a eventi di perdita

TCP AIMD


Client
Server
Segmento
ACK
i = 4
cwnd
RTT
i + 1
cwnd
RTT
i + 2
cwnd
RTT
i + 3
cwnd
Tempo
Tempo

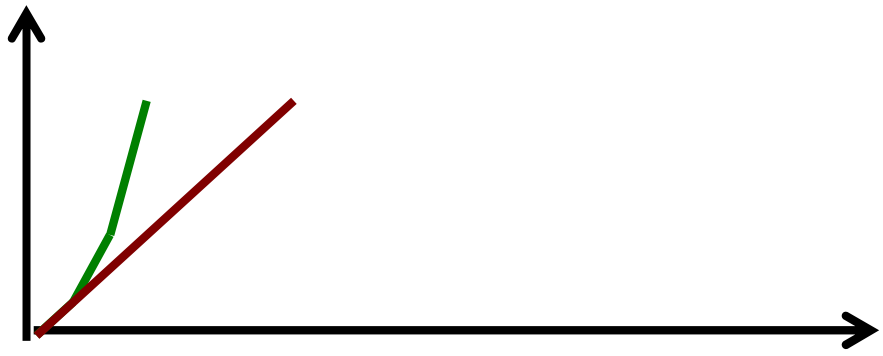
Algoritmo di controllo della congestione AIMD



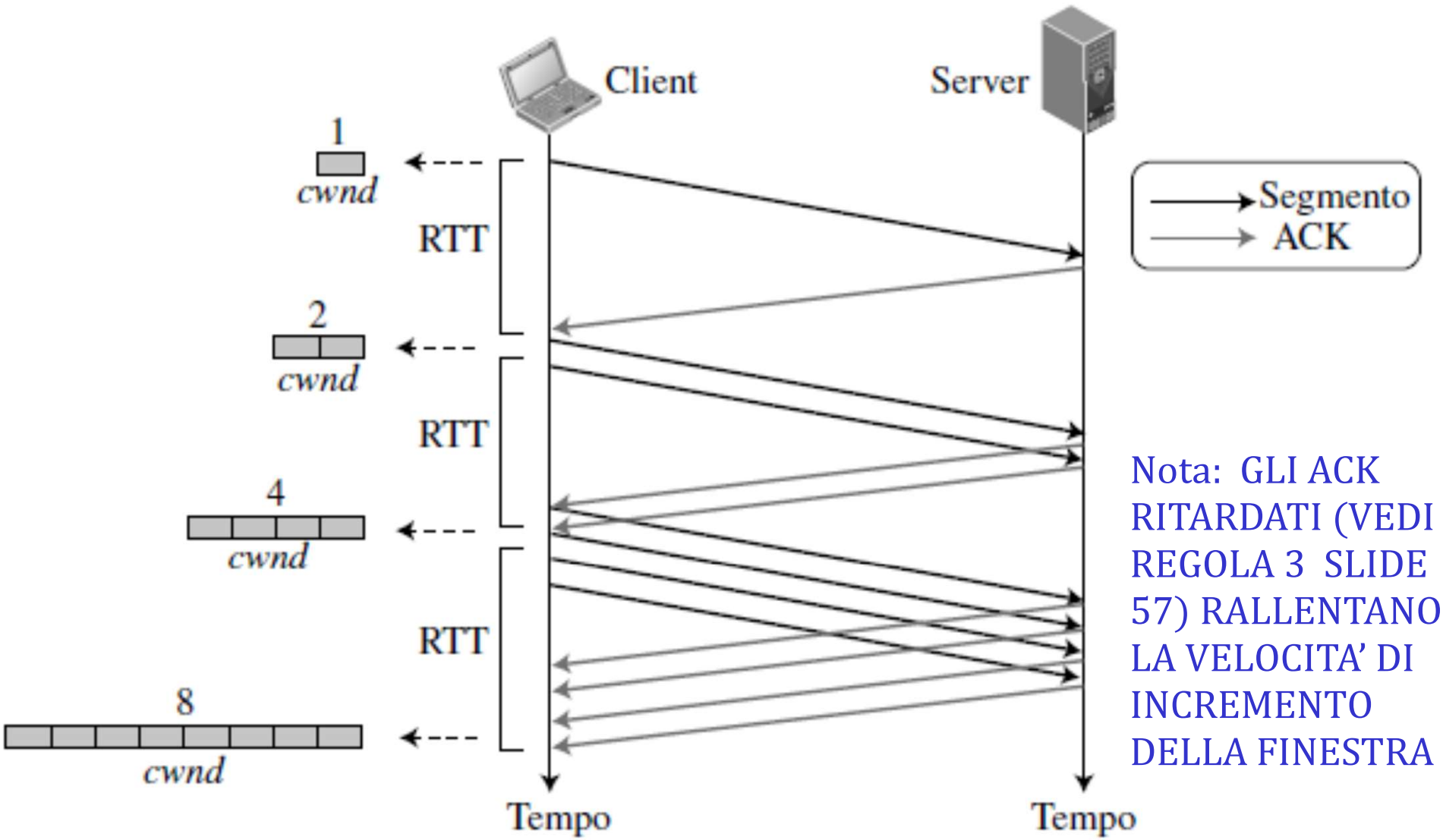
Andamento a dente di sega

TCP: Slow start

- All'inizio la finestra di congestione, CongWin, è posta pari a 1 MSS (frequenza di invio 1 MSS/RTT)
  - Se MSS=500 byte e RTT=200ms si ha una frequenza di invio di circa 20kb/s: se ho 1Mb/s di banda, impiego molto tempo ad arrivarci con incremento lineare (vedi AIMD).
- Incremento Congwin di 1MSS ad ogni ACK (non duplicato)
- L'effetto è che CongWin raddoppia ad ogni RTT -> crescita esponenziale



TCP: Slow start (2)

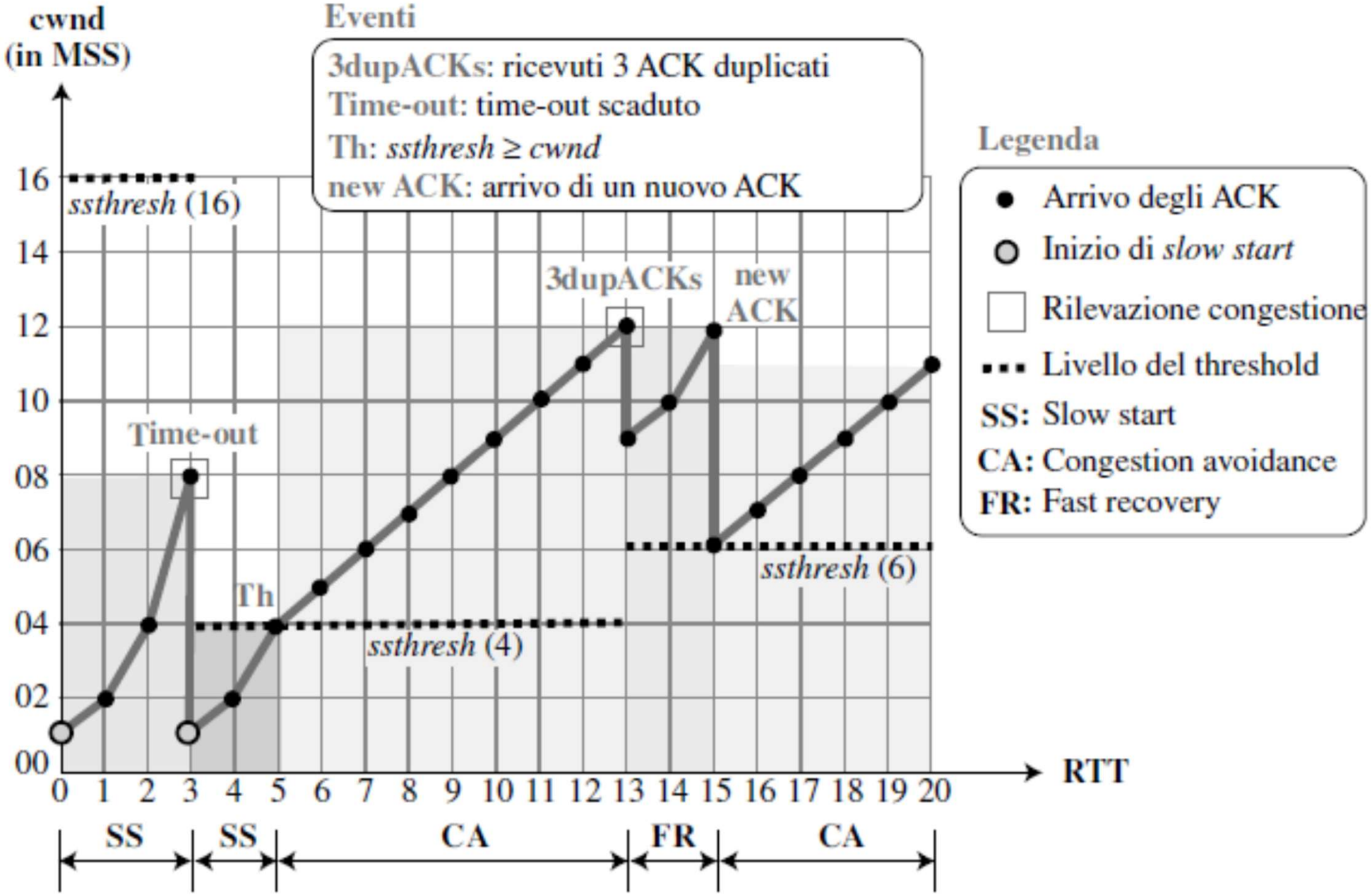


Nota: GLI ACK RITARDATI (VEDI REGOLA 3 SLIDE 57) RALLENTANO LA VELOCITA' DI INCREMENTO DELLA FINESTRA

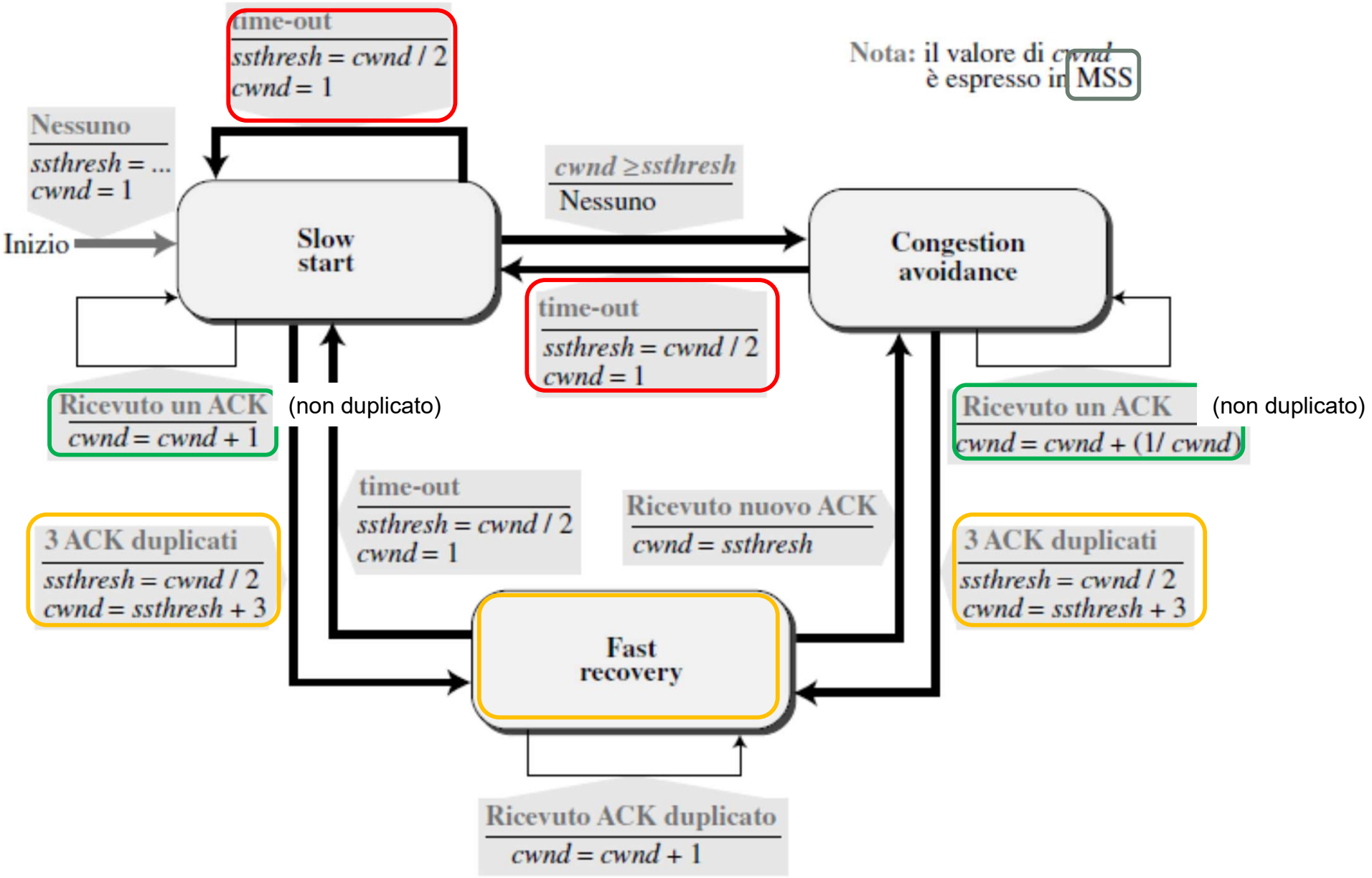
TCP RENO

- Inizialmente viene definita una variabile “soglia”, alla quale è assegnato un valore alto (es. 64 KB).
  - La soglia determina quando termina la partenza lenta (slow start) ed inizia la fase di congestion avoidance (AI)
- Se cwnd < soglia, cwnd aumenta esponenzialmente (slow start)
- Se cwnd > soglia, cwnd aumenta linearmente (Additive Increase)
- Evento di perdita
  - Se ho **3 ACK duplicati** pongo prima la soglia a metà di cwnd e poi cwnd = soglia + 3 MSS (fast recovery)
  - Se ho un ACK perso per **timeout** pongo la soglia a metà di cwnd e pongo cwnd = 1 MSS (slow start)
- Nella fase fast recovery:
  - se avviene un timeout si va in slow start
  - finché continuano ad arrivare ACK duplicati cwnd=cwnd+1
  - se arriva un nuovo ack (non duplicato) si va in congestion avoidance e cwnd=soglia

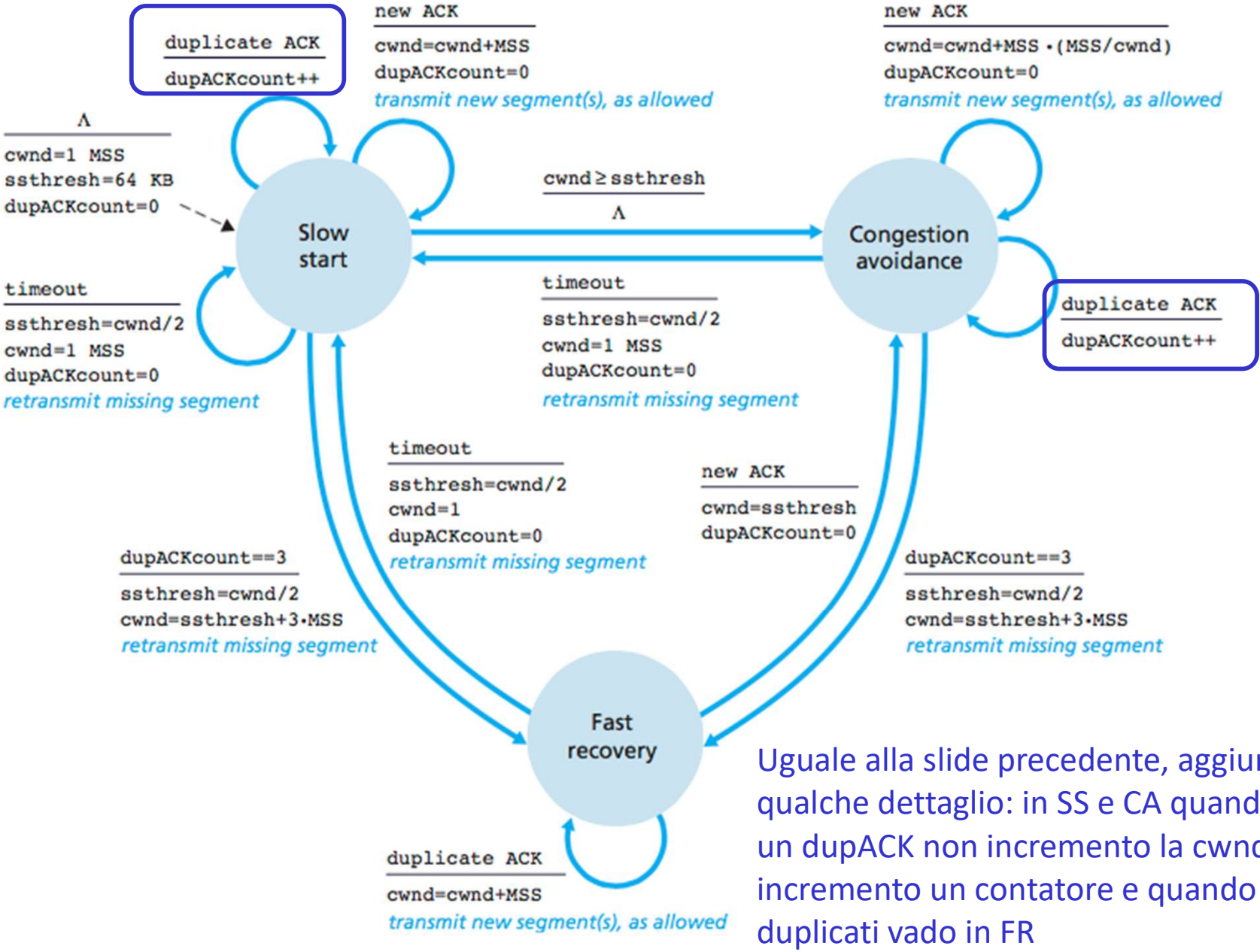
Esempio di controllo della congestione con TCP Reno



TCP Reno


time-out
ssthresh = cwnd / 2
cwnd = 1
Nota: il valore di cwnd è espresso in MSS
Nessuno
ssthresh = ...
cwnd = 1
Inizio
Slow start
cwnd ≥ ssthresh
Nessuno
Congestion avoidance
time-out
ssthresh = cwnd / 2
cwnd = 1
Ricevuto un ACK
cwnd = cwnd + 1
(non duplicato)
Ricevuto un ACK
cwnd = cwnd + (1/ cwnd)
(non duplicato)
time-out
ssthresh = cwnd / 2
cwnd = 1
Ricevuto nuovo ACK
cwnd = ssthresh
3 ACK duplicati
ssthresh = cwnd / 2
cwnd = ssthresh + 3
3 ACK duplicati
ssthresh = cwnd / 2
cwnd = ssthresh + 3
Fast recovery
Ricevuto ACK duplicato
cwnd = cwnd + 1

TCP Reno



Uguale alla slide precedente, aggiunge qualche dettaglio: in SS e CA quando ricevo un dupACK non incremento la cwnd, incremento un contatore e quando ho 3 ack duplicati vado in FR

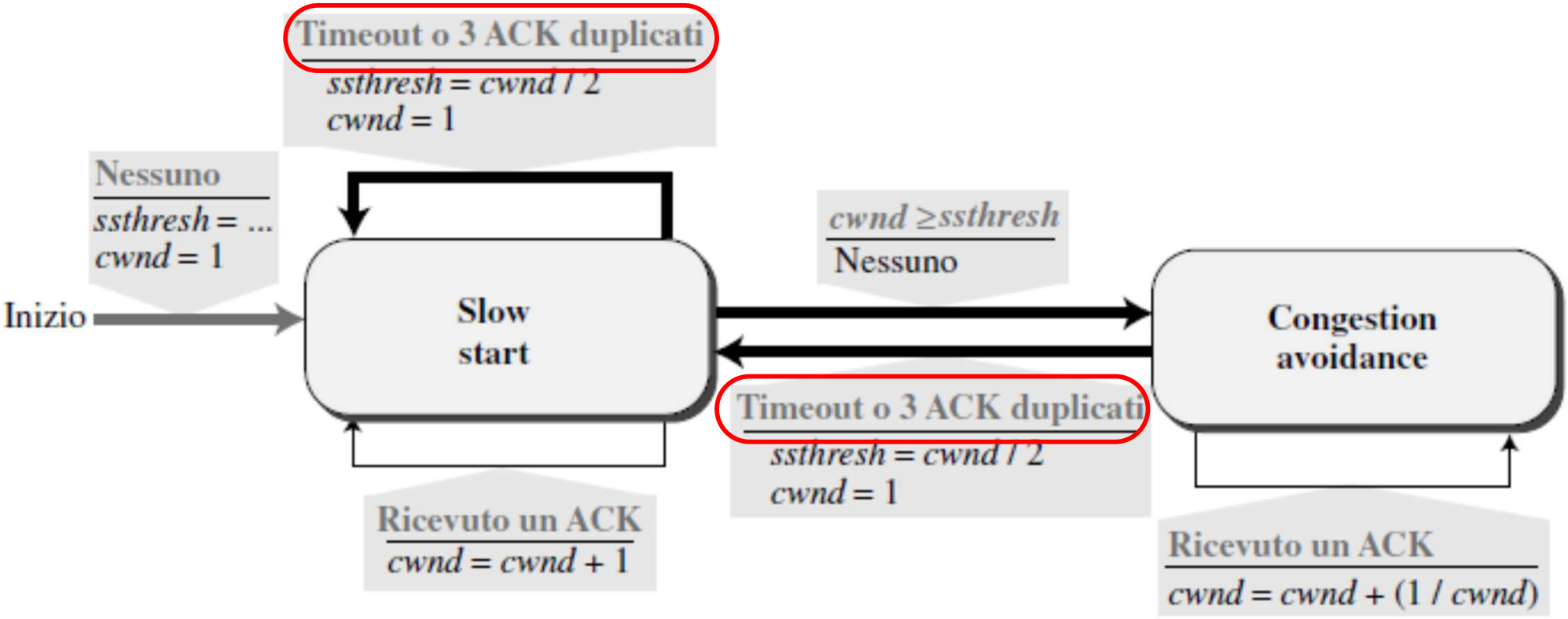
Kurose Ross

Ancora su controllo di congestione

- Abbiamo visto TCP Reno
- In realtà ci sono varie implementazione del controllo di congestione…

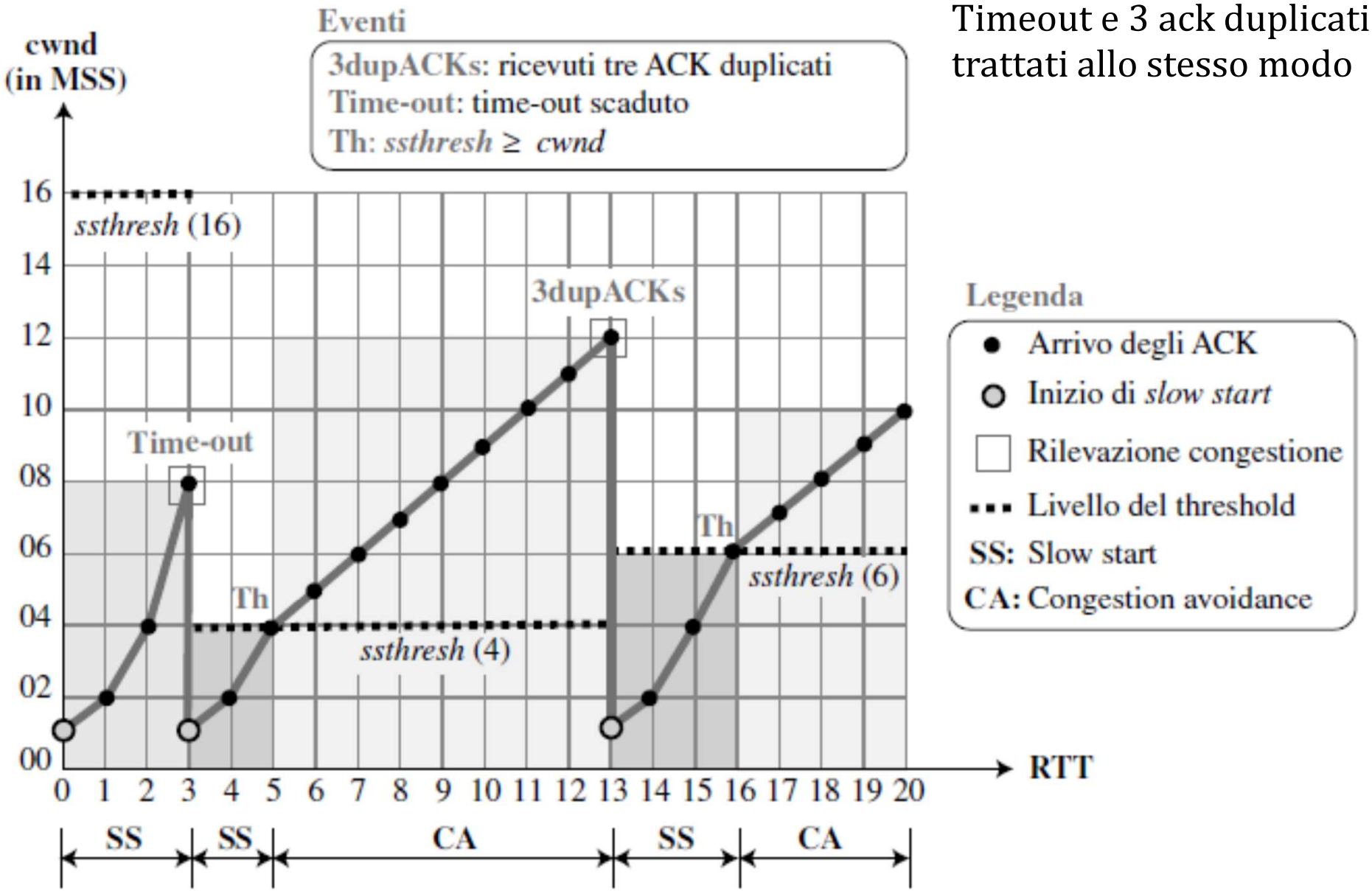
- TCP Tahoe (meno recente di Reno)
  - Slow start e congestion avoidance
  - Timeout e 3 ack duplicati gestiti allo stesso modo:
    - Soglia = cwnd/2 e cwnd =1 (si va in slow start)

TCP Tahoe


Timeout o 3 ACK duplicati
ssthresh = cwnd / 2
cwnd = 1
Nessuno
ssthresh = ...
cwnd = 1
Inizio
Slow start
cwnd ≥ssthresh
Nessuno
Congestion avoidance
Timeout o 3 ACK duplicati
ssthresh = cwnd / 2
cwnd = 1
Ricevuto un ACK
cwnd = cwnd + 1
Ricevuto un ACK
cwnd = cwnd + (1 / cwnd)

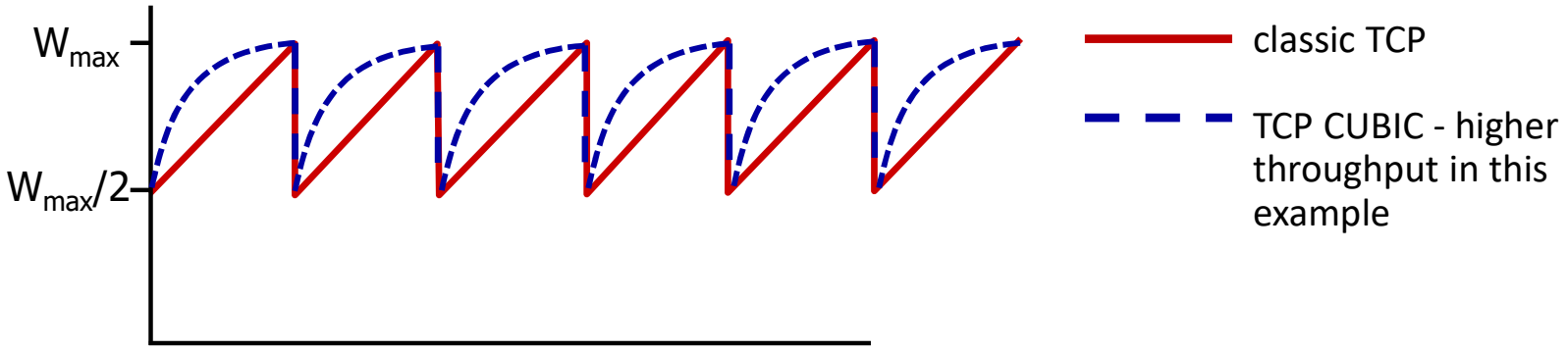
TCP Tahoe

Versione precedente a Reno
Timeout e 3 ack duplicati sono trattati allo stesso modo



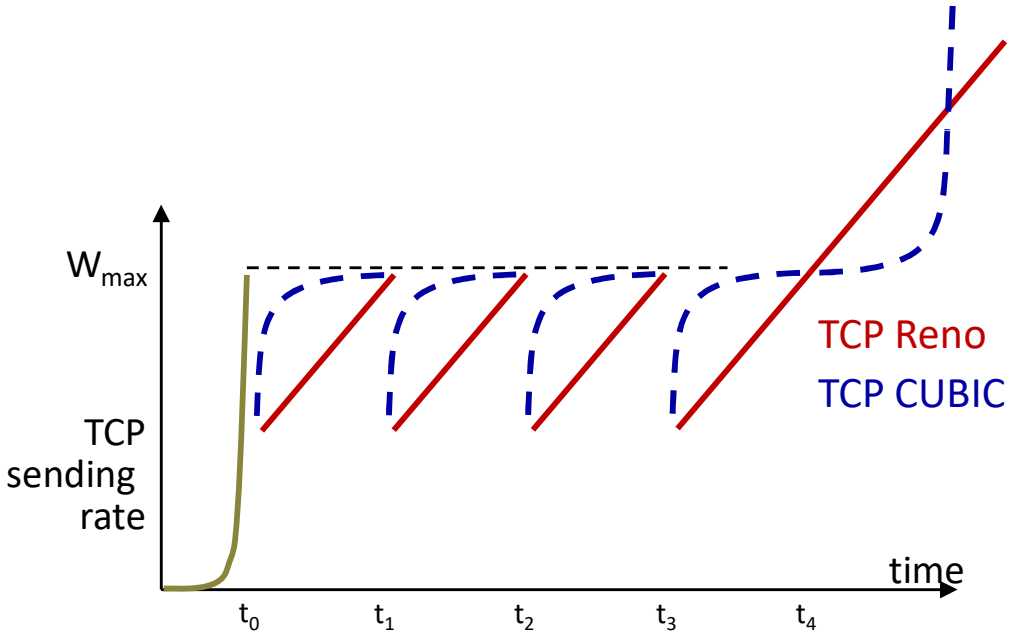
TCP CUBIC

- C'è un modo migliore di AIMD per "testare" la banda utilizzabile?
- Idea:
  - W_{max}: rate di invio quando è stata rilevata congestione
  - Lo stato di congestione del collegamento "collo di bottiglia" probabilmente (?) non è cambiato molto
  - Dopo aver dimezzato la finestra in seguito ad un evento di perdita, prima incremento verso W_{max} *più veloce*, ma poi avvicinamento a W_{max} più *lento*



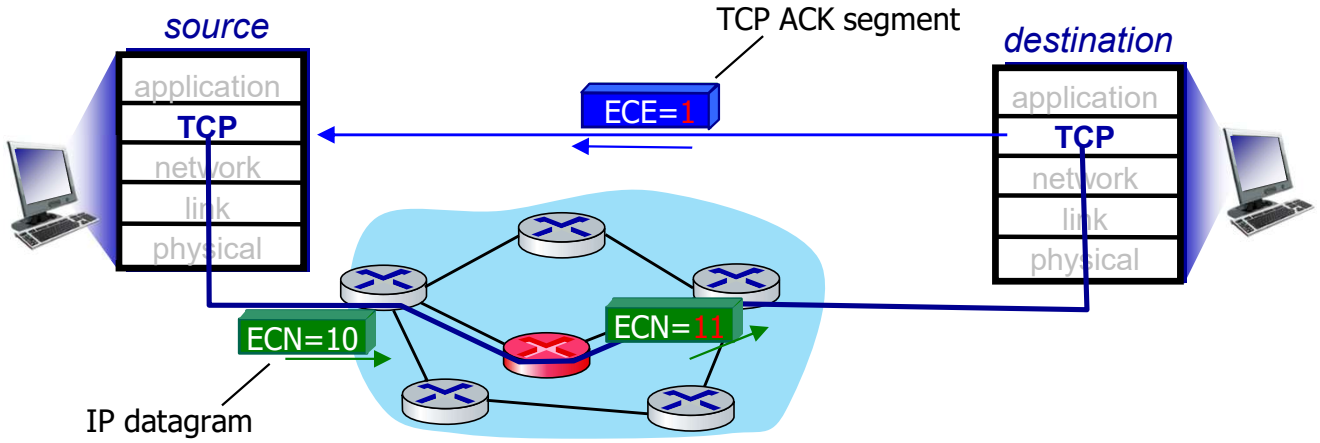
TCP CUBIC

- K: punto nel tempo in cui la cwnd raggiungerà W_{max}
  - K stesso è configurabile
- Aumento di W come una funzione del *cubo* della distanza tra l'istante di tempo attuale e k
  - Incrementi maggiori quando siamo lontani da K
  - Incrementi più piccoli (cauti) quando siamo più vicini a K
- TCP CUBIC default in Linux, implementazione molto diffusa per i Web servers



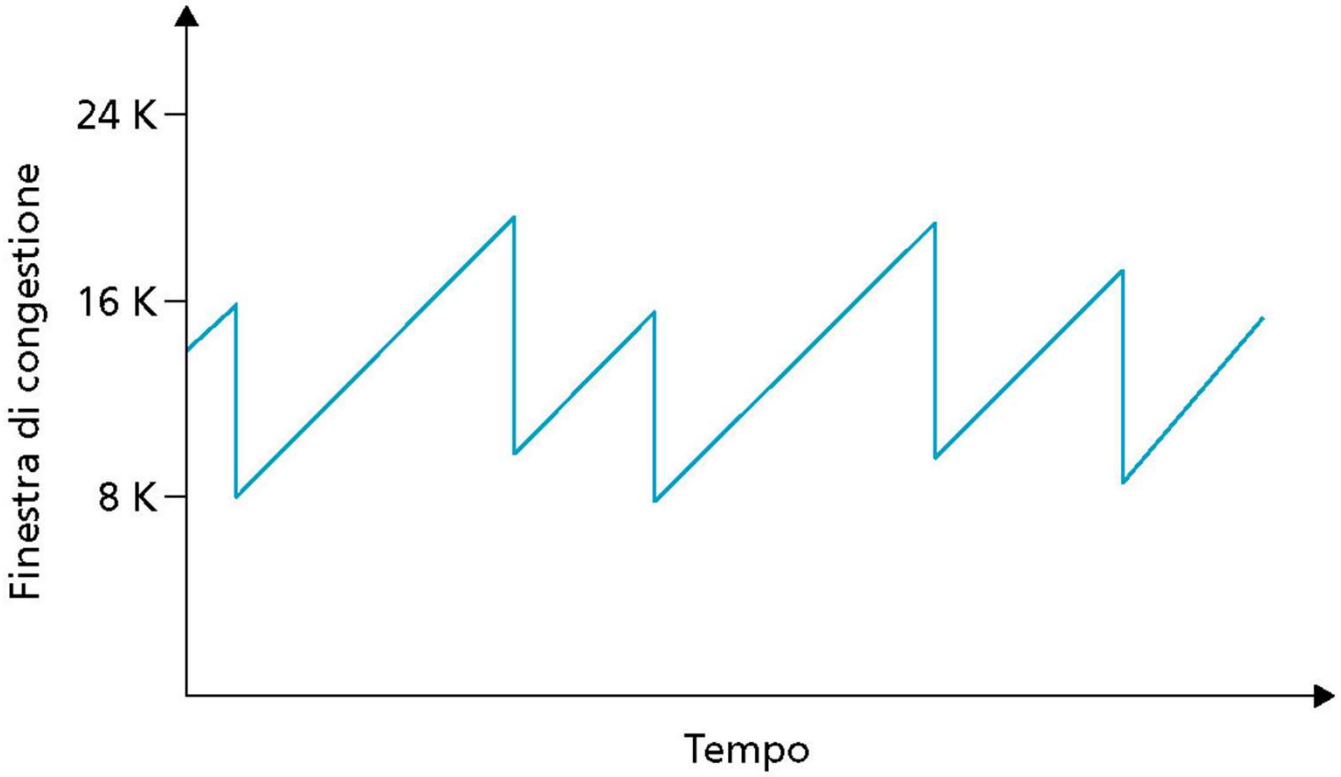
Explicit congestion notification (ECN)

- Le implementazioni TCP spesso implementano un meccanismo di controllo di congestione network-assisted:
  - due bit in header IP (campo ToS) settati dai router per indicare la congestione
  - Informazione di congestione inviata a destinazione
  - La destinazione imposta il bit ECE (ECN-Echo) bit nel segmento di riscontro per notificare la congestione al mittente
  - Il mittente setta il bit CWR (Congestion Window Reduced) per indicare che ha ricevuto la notifica di congestione
- Coinvolge sia IP che TCP



Andamento macroscopico della finestra di congestione

- Nella maggior parte dei casi la congestione viene rilevata tramite 3 ack duplicati
- Trascurando slow start e Fast recovery -> incremento additivo (finestra aumenta di 1/cwnd ad ogni riscontro non duplicato) e decremento moltiplicativo (cwnd dimezza)



Andamento a dente di sega

TCP: throughput

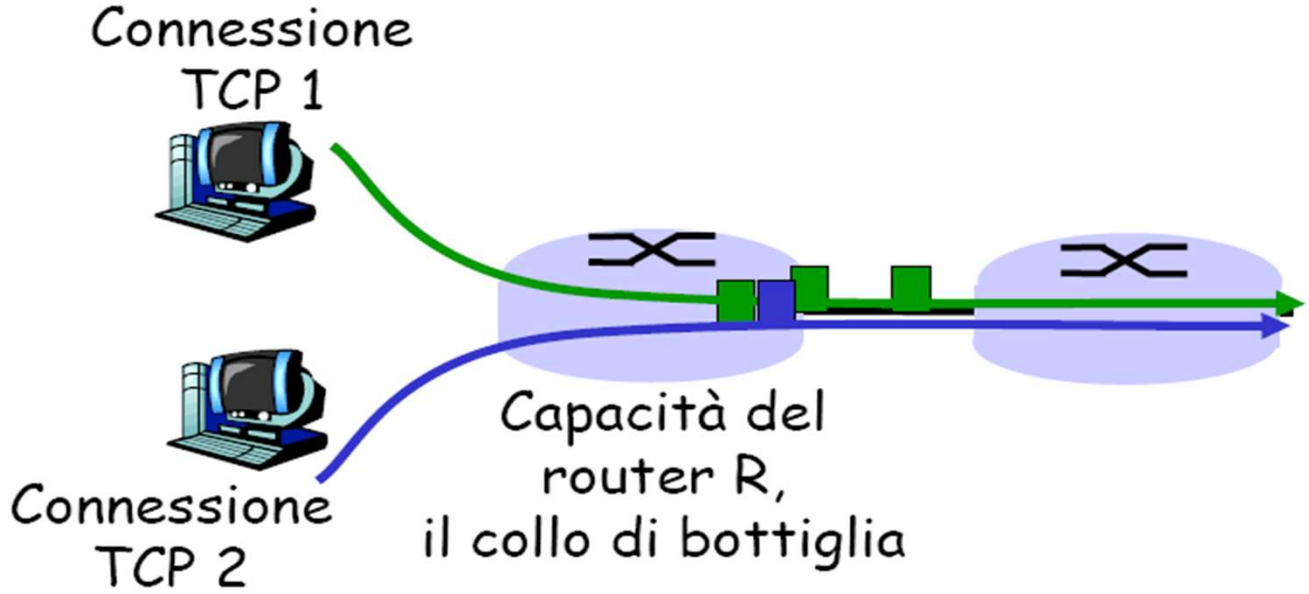
- Partendo dalle medesime considerazioni della slide precedente su andamento macroscopico della finestra
- Indicando con W il valore massimo in byte della finestra (ovvero quando si verifica l'errore), TCP in regime stazionario offre il seguente throughput medio (frequenza di invio media)

$$Throughput = \frac{0{,}75 \cdot W}{RTT}$$

Quando la finestra è W il throughput è W/RTT
Dopo l'evento di perdita la finestra va a W/2, quindi il throughput è W/2RTT
Throughput medio: 0,75 W/RTT

Equità (Fairness)

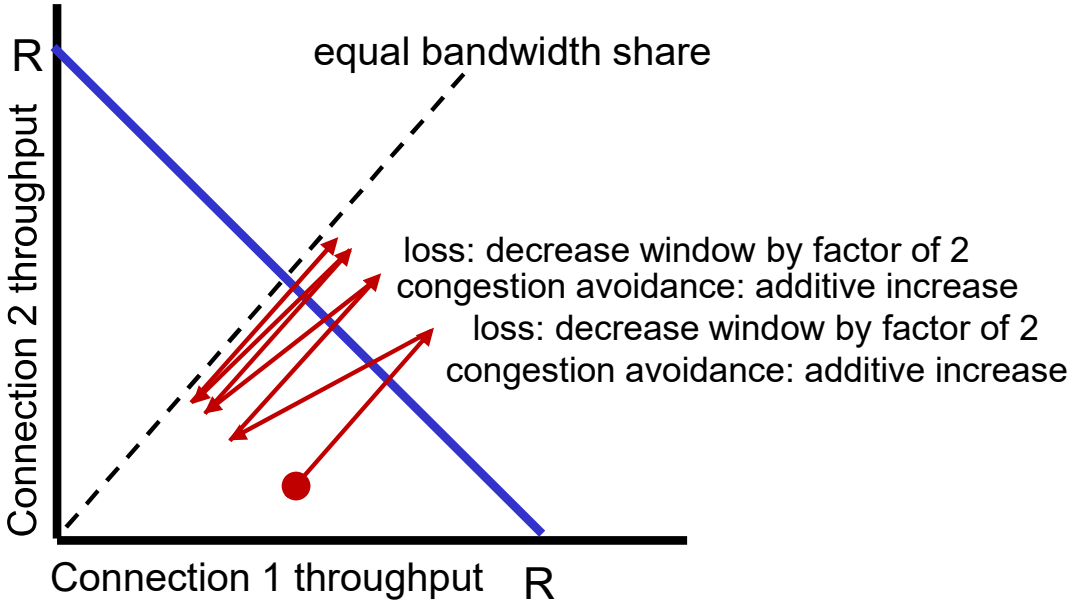
- Ipotesi:
  - *K* connessioni TCP insistono su un unico link di capacità *R* bit/s
  - Le connessioni hanno gli stessi valori di MSS e RTT
  - Non ci sono altri protocolli che insistono sullo stesso link
- Risultato:
  - Ognuna delle connessioni TCP tende a trasmettere *R/K* bit/s



TCP è equo?

esempio: due sessioni TCP in competizione:

- additive increase
- multiplicative decrease



TCP è equo ?

A: Si, nelle seguenti ipotesi

- stesso RTT e MSS
- Numero costante di sessione solo in CA

Throughput realizzati dalle connessioni

- Ipotesi: le due connessioni hanno stesso valore di MSS e RTT
- NOTA: nella pratica connessioni con RTT più piccolo variano più velocemente congwin e raggiungono throughput superiori a connessioni con RTT maggiore

Connessioni TCP parallele

- Un'applicazione può aprire connessioni multiple in parallel tra due host
- Tipico per i web browsers e.g.,
  - Collegamento con rate R con 9 connessioni esistenti connections:
  - new app apre 1 sessione TCP, ottiene un rate R/10
  - new app apre 11 sessioni TCP , ottiene R/2

Risorse online

- Controllo di congestione e fairness

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/tcp-congestion/index.html