



Università
degli Studi
della Campania
Luigi Vanvitelli

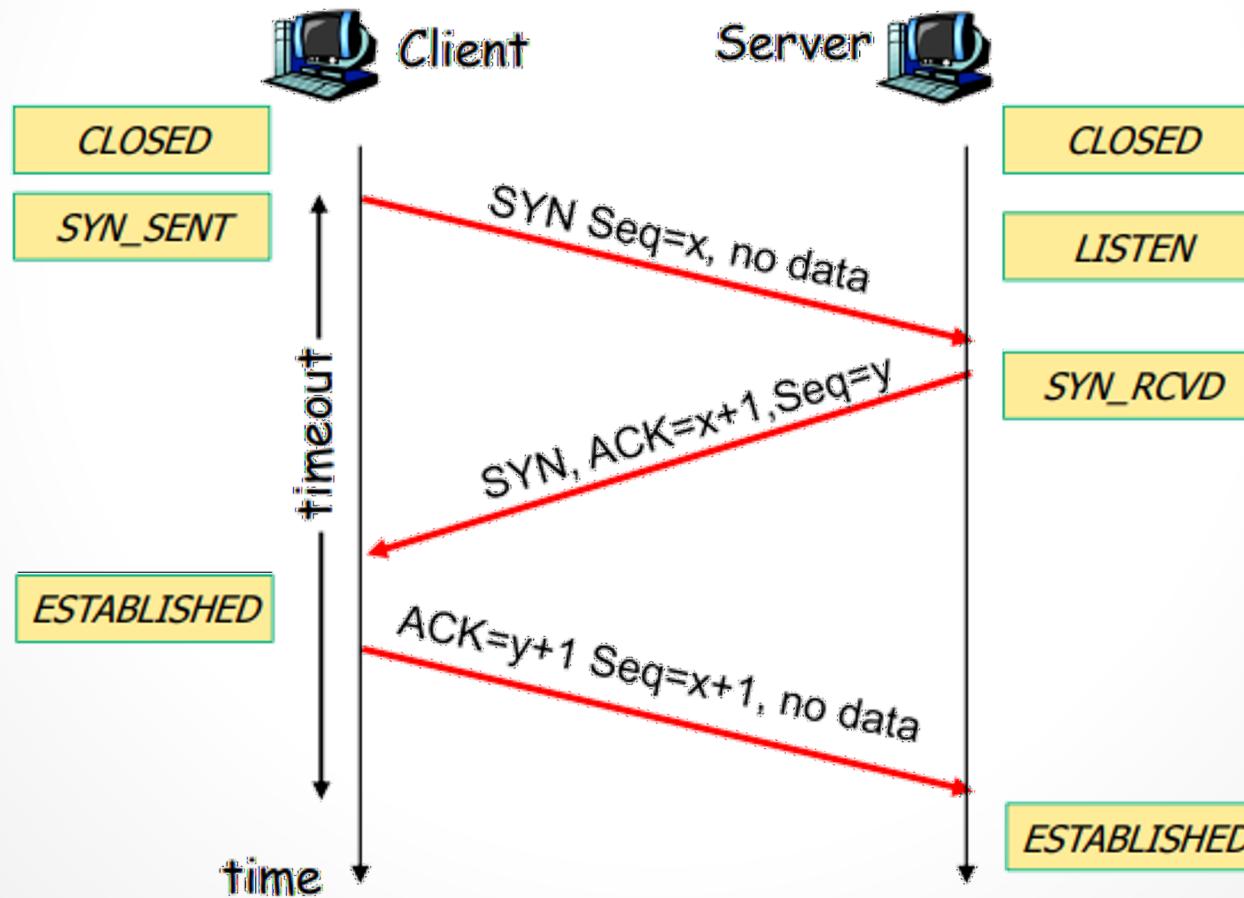
Reti di Calcolatori e Cybersecurity

TCP – Controllo di flusso e congestione

Ing. Vincenzo Abate

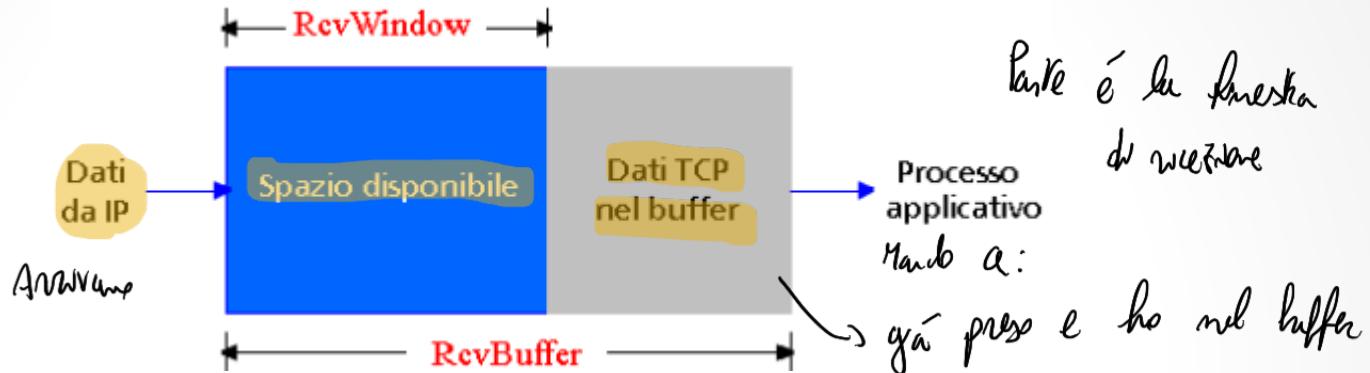
TCP avvio connessione

- **3 way-handshake**
- Il **client prende l'iniziativa inviando il primo segmento**



TCP - flow control

Il lato ricevente della connessione TCP ha un buffer di ricezione:



Il processo applicativo potrebbe essere rallentato dalla lettura nel buffer

- 1 Velocità con cui arrivano dati
- 2 VS Velocità con cui esce.
- ① non più veloce di ② allora non perdo pacchetti



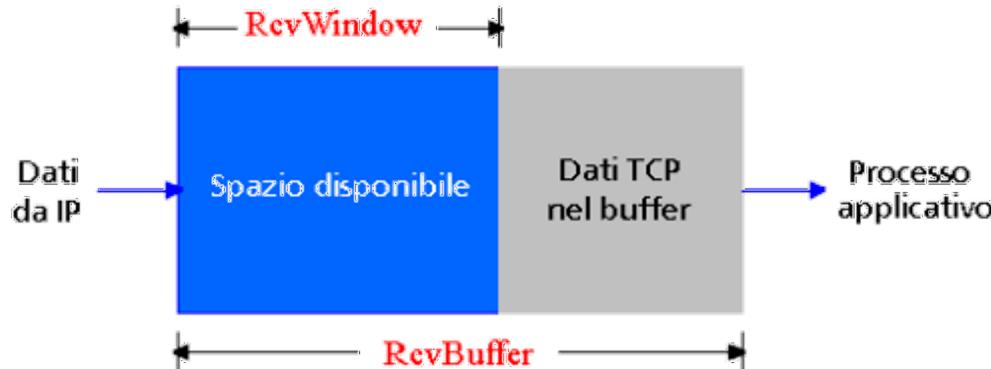
Hp: se buffer si satura, scatta pacchetto fuori sequenza.

Controllo di flusso

Servizio di corrispondenza delle velocità: la frequenza d'invio deve corrispondere alla frequenza di lettura dell'applicazione ricevente

TCP - flow control

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



RcvBuffer = dimensione buffer in ricezione

RcvWindow = spazio disponibile nel Buffer

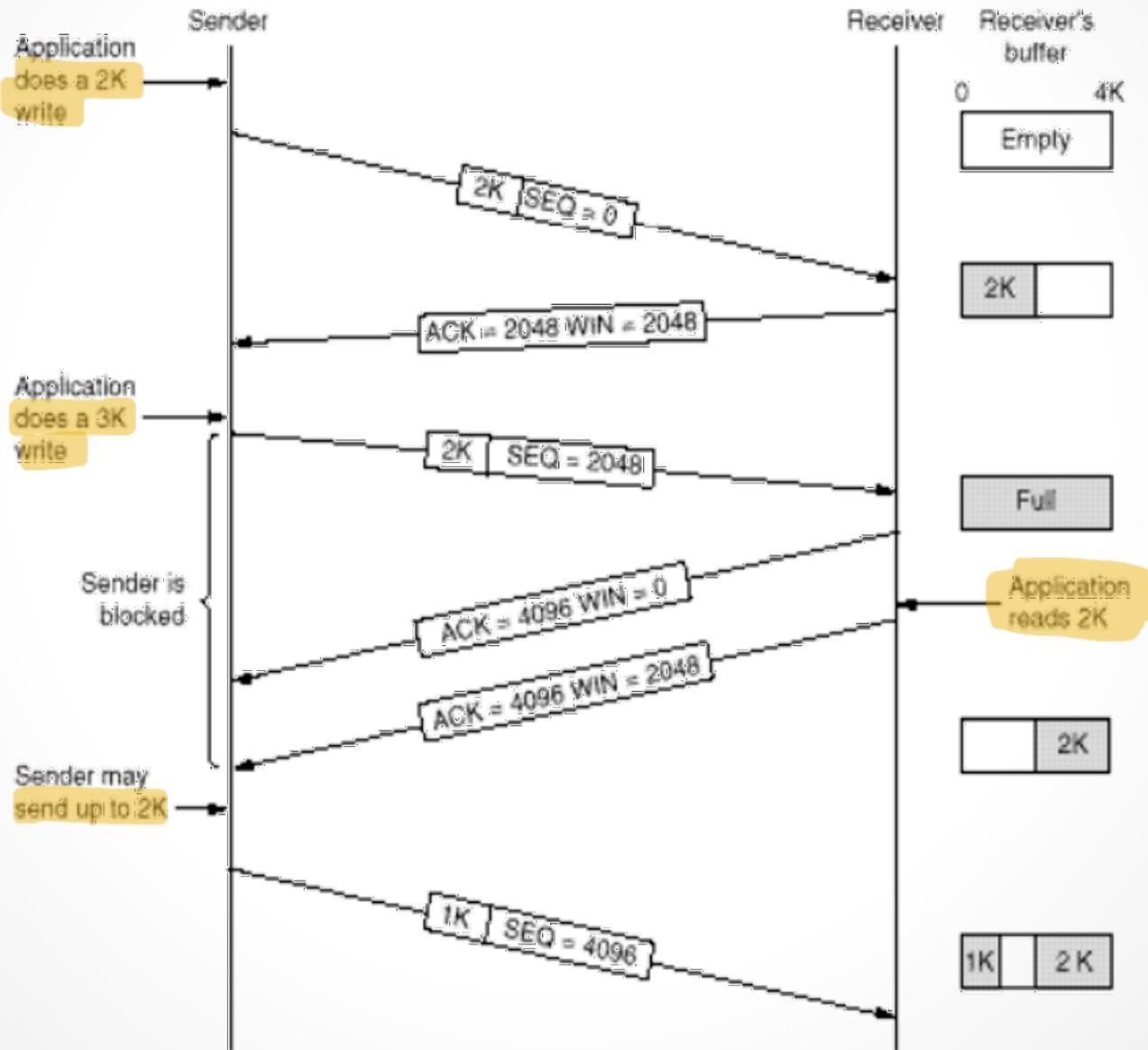
Ricevente: comunica dinamicamente al mittente la dimensione corrente del buffer nel campo RcvWindow nel segmento TCP:

$RcvWindow = RcvBuffer - [LastByteRcvd - LastByteRead]$

Mittente: conserva i dati già trasmessi ma non riscontrati e limita tale quantità all'ultima RcvWindow ricevuta:

$LastByteSent - LastByteAcked \leq RcvWindow$

TCP – transmission policy



Sindrome Silly Window

Silly Window Syndrome (ricevitore): il ricevitore svuota lentamente il buffer di ricezione e invia segmenti di ack con dimensione della finestra molto piccola, quindi il trasmettitore invia segmenti corti con molto overhead.

Soluzione con l'**algoritmo di Clark**: il ricevitore indica una finestra nulla finchè il buffer di ricezione non si è svuotato per metà o per una porzione uguale a MSS.

→ Overhead: troppa info di controllo rispetto ai dati.

Silly Window Syndrome (trasmettitore): l'applicazione genera dati lentamente, invia segmenti molto piccoli così come vengono prodotti.

Soluzione **algoritmo di Nagle**: il TCP sorgente invia la prima porzione di dati anche se corta e gli altri vengono inviati solo se o il buffer di uscita contiene almeno MSS byte, oppure se si riceve un ack per il segmento precedente.

↓ Utilizzo al meglio la rete: o quando ho i MSS byte
quando ho i MSS byte

TCP – algoritmo di Nagle

Per applicazioni che inviano dati un byte alla volta (es: TELNET):

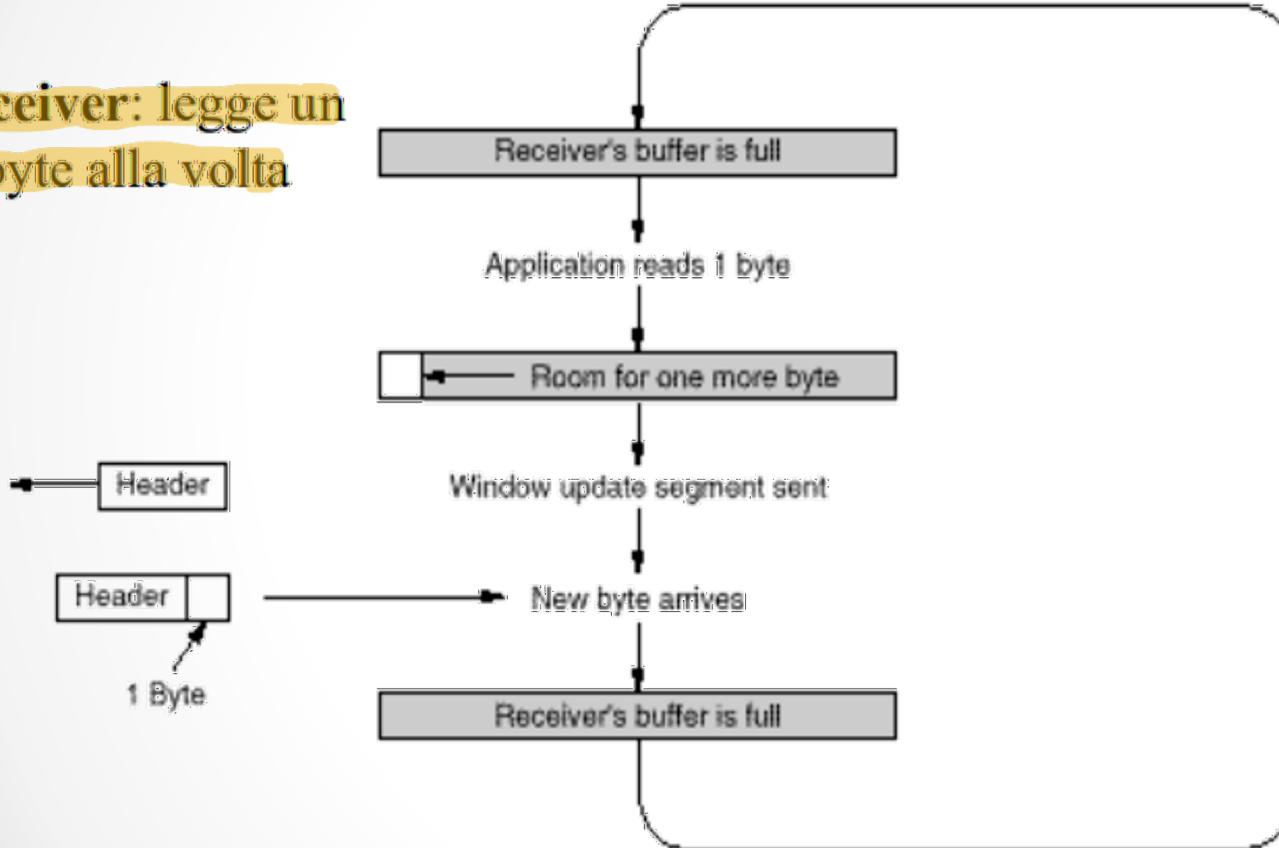
- invia il primo byte e bufferizza il resto finché non giunge l'ACK in seguito:
- invia, in un unico segmento, tutti i caratteri bufferizzati
- ricomincia a bufferizzare finché non giunge l'ACK per ognuno di essi
- Elimina la sindrome della Silly Window al trasmettitore

```
if there is new data to send
    if window size >= MSS and available data is >= MSS
        send complete MSS segment now
    else
        if there is unconfirmed data still in the pipe
            enqueue data in the buffer until an ack is received
        else
            send data immediately
        end if
    end if
end if
```

Sindrome 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

Il ricevitore indica una finestra nulla finché il buffer di ricezione non si è svuotato per metà o per una porzione uguale a MSS.

Congestione

Tecnicamente dovuta a:

- un numero elevato di sorgenti di traffico *tante sorgenti che contribuiscono*
- 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:

Router non possono salvare troppi pacchetti

- perdita di pacchetti: » buffer overflow nei router
- ritardi nell'inoltro dei pacchetti: » accodamenti nei buffer dei router *Lo vuole più tempo per uscire dal buffer*
- scarso utilizzo delle risorse di rete

Congestione - Scenario 1

2 mittenti 2 riceventi

1 router con buffer ∞ :

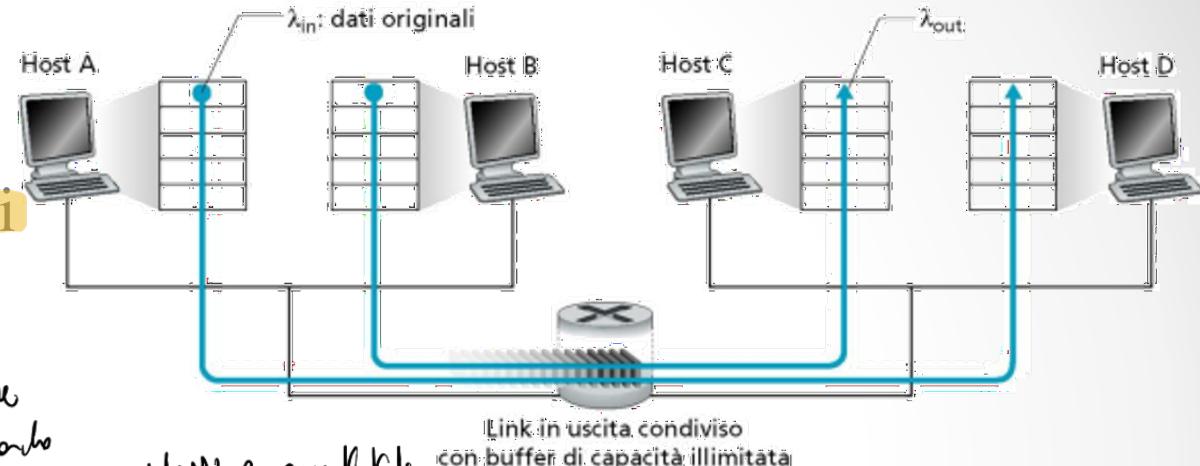
non ci sono ritrasmissioni

R = capacità link

condiviso *Quanti dati puoi mandare al secondo*

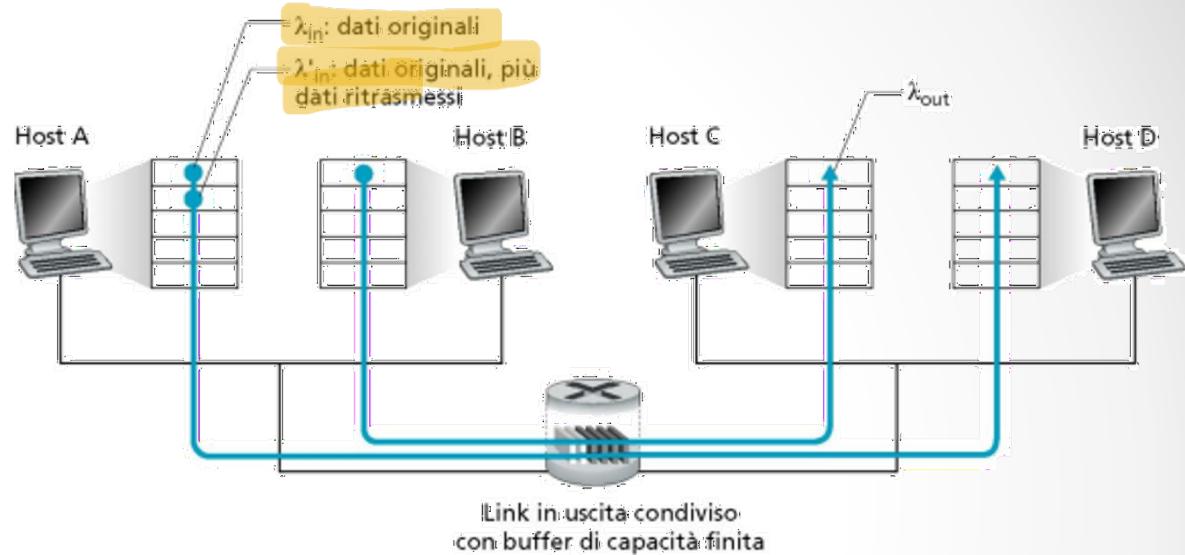
I ritardi aumentano
all'avvicinarsi del limite
di capacità del canale

Non si può superare il
max throughput ($R/2$) a
causa della condivisione
del collegamento tra le
due connessioni



Congestione – Scenario 2

- 2 mittenti/riceventi
- 1 router con buffer limitato → I pacchetti che giungono in un buffer pieno vengono scartati
- Connessione affidabile ✓ (ritrasmissione)



Poiché i pacchetti possono essere ritrasmessi occorre prestare attenzione all'uso del termine ‘tasso di invio’ ↗ dim + qualcosa: segmenti ritrasmessi

Indichiamo con λ_{in} byte/s il tasso di trasmissione e con λ'_{in} byte/s il tasso al quale il livello trasporto invia segmenti (contenenti dati originali e ritrasmissioni), spesso detta **carico offerto alla rete**

In questo scenario le prestazioni dipendono fortemente da come si effettua la ritrasmissione

Congestione - effetti

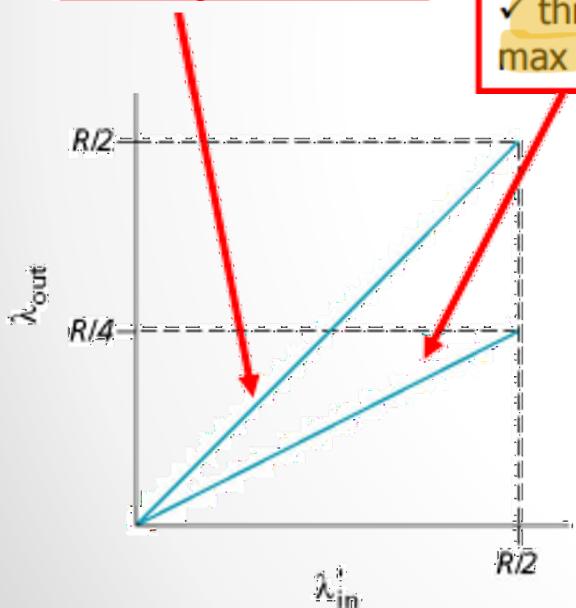
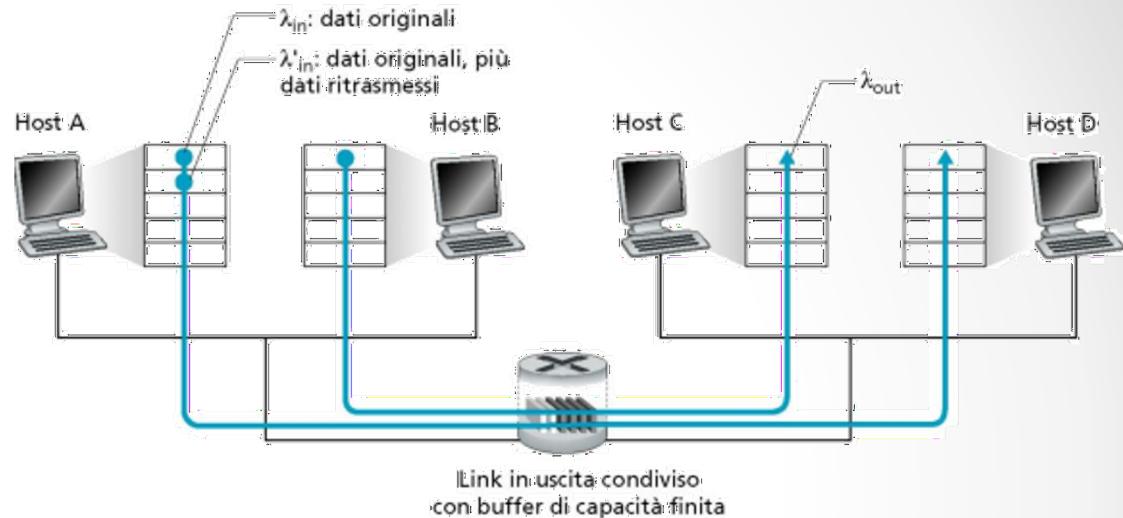
Il mittente invia dati solo quando il buffer non è pieno:

- caso ideale

✓ no ritrasmissioni
✓ throughput max = $R/2$

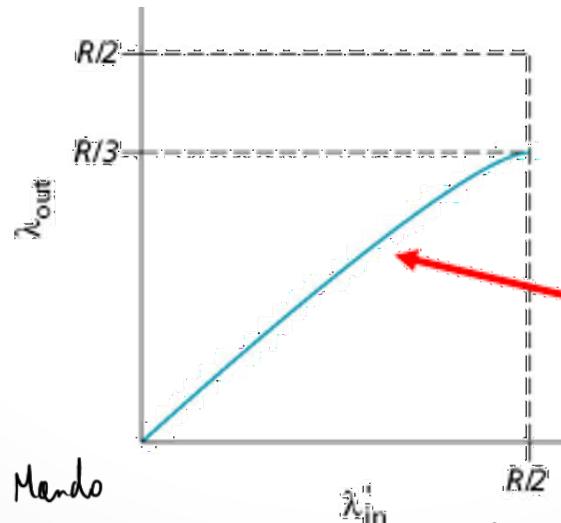
Scadenza prematura del timer del mittente:

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



medio 2 volte: Mendo

2 volte \rightarrow doppio che $\lambda_{in} < \lambda_{out} \Rightarrow$ $\lambda_{out} = \frac{R}{2}$ per max rate $\frac{R}{2}$



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

Devo compiere caso buffer pieno

avviciniamo a $R/3$

Approcci al Controllo di Congestione

Cdc end-to-end

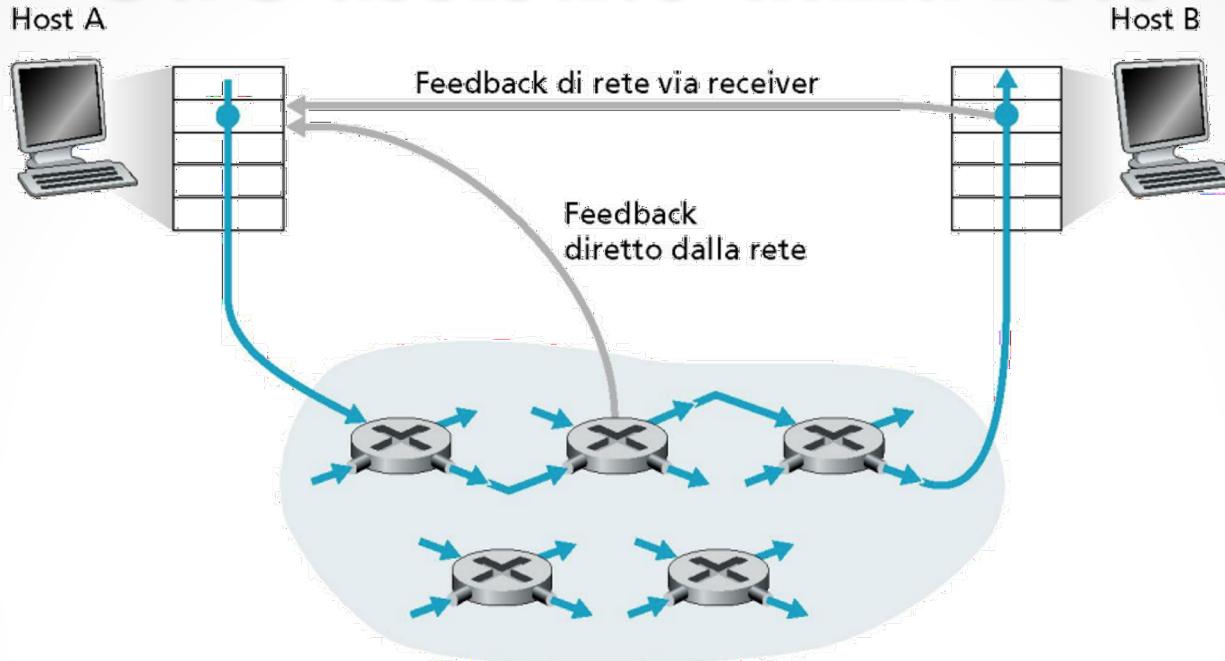
- Nessuna segnalazione esplicita dalla rete
- A partire dall'osservazione di ritardi e perdite di pacchetti gli end-system deducono uno stato di congestione nella rete
- Approccio utilizzato da TCP

CdC assistito dalla 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, DECnet, TCP/IP ECN, ATM)
- E' anche possibile utilizzare feedback di rete più sofisticati (ATM ABR – *available bit rate*): il sender è informato circa la massima frequenza alla quale può trasmettere



CdC assistito dalla rete



L'informazione di congestione viene fornita dalla rete in due modi:

- Avviso diretto del router tramite un **chokepacket** che riferisce: ‘sono congestionato!’
- Il router imposta un campo in un pacchetto che fluisce da mittente a destinatario. Alla ricezione il destinatario notifica al mittente l’indicazione di congestione (**richiede almeno un RTT**)

Per evitare di chiedere un PPT

CdC in TCP

- TCP offre un servizio affidabile di trasporto tra due processi in esecuzione su host diversi e presenta il meccanismo di controllo di congestione.
- TCP utilizza un controllo di congestione end-to-end poiché il livello IP non fornisce nessun feedback esplicito relativo alla congestione della rete. ^{→ Best effort}
- TCP impone a ciascun mittente un limite al tasso di invio sulla propria connessione in funzione della congestione della rete percepita.
- Se il mittente TCP rileva situazioni di scarso traffico sul percorso che porta alla destinazione incrementa il tasso di trasmissione, se invece rileva traffico, lo riduce.

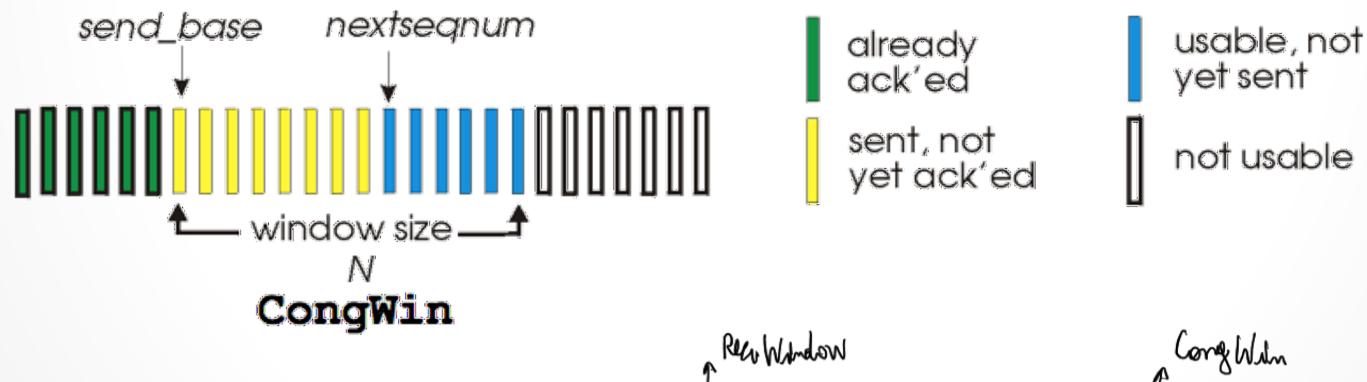
Tre domande:

1. Come limita il mittente TCP il tasso di invio?
2. Come percepisce la congestione?
3. Quale algoritmo usare per variare il tasso di invio in funzione della congestione end-to-end?

CdC in TCP

Gli estremi di una connessione TCP gestiscono un buffer in ricezione, uno in invio e diverse variabili.

Il controllo di congestione di TCP fa tener traccia agli estremi anche della finestra di congestione (Congwin) Da' la possibilità di limitare la velocità di invio. Che impone un vincolo alla velocità di immissione in rete di traffico da parte del mittente



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

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

Determinantes da Revolução e Suportes de apoio dão destaque para a marcha da Contra

CdC in TCP

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 finché non si verifica la perdita di un segmento (interpretata come il sopraggiungere dello stato di congestione)

In seguito alla perdita di un segmento:

CdC in TCP: fasi

Slow Start *incremento esponenziale, fase 1*

Partenza lenta (per modo di dire!)

Congestion Avoidance: *incremento - decremento*

Additive Increase, Multiplicative Decrease (AIMD)

incremento additivo, decremento moltiplicativo

Fast recovery

- Slow start e Congestion Avoidance sono componenti obbligatorie di TCP e differiscono per il modo in cui aumentano CongWin in risposta agli ACK ricevuti (Il primo più rapidamente)
- Fast recovery è suggerita ma non obbligatoria per i mittenti TCP

CdC in TCP: Slowstart

Quando la connessione inizia, CongWin = 1 MSS

Esempio: MSS = 500 byte e RTT = 200 msec

velocità iniziale = $\frac{500 \cdot 8}{200}$ kbps

larghezza di banda disponibile può essere >> MSS/RTT

Poiché la banda disponibile può essere molto più grande di MSS/RTT
è auspicabile raggiungere rapidamente il valore ottimale (banda disponibile)

Quando la connessione inizia, si aumenta il tasso esponenzialmente fino al primo evento di perdita

CdC in TCP: Slowstart

Crescita esponenziale della dimensione della finestra (ogni RTT)

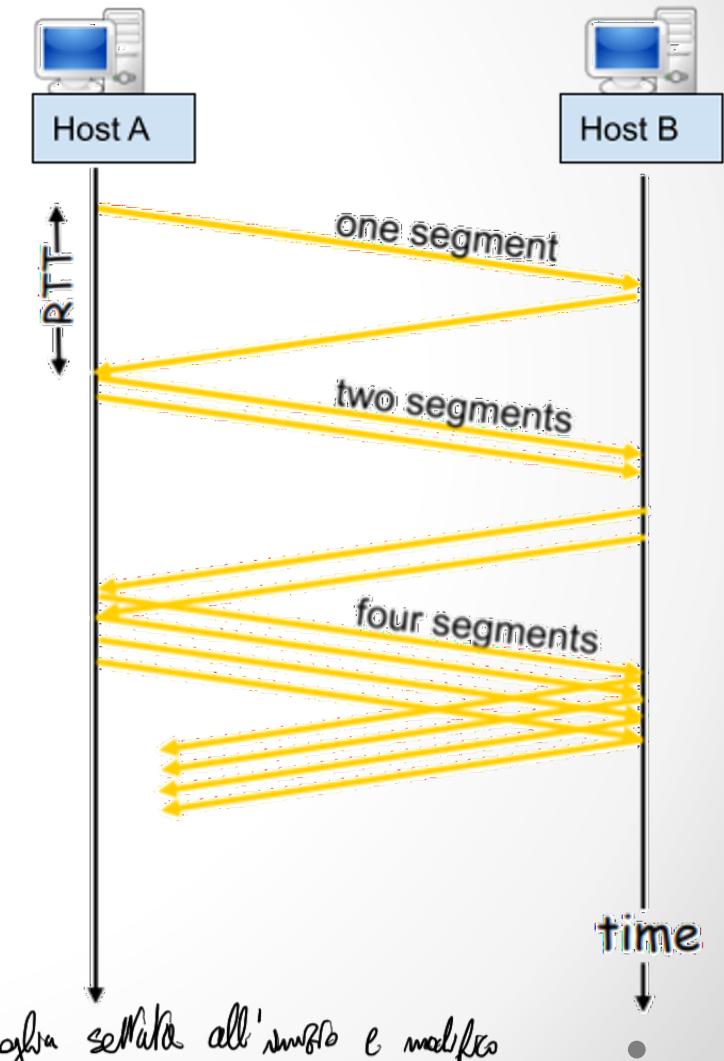
- “Slow start” → termine improprio!

Evento di perdita:

- timeout
- tre ACK duplicati consecutivi
- Ritorna a CongWin = 1 MSS

Algoritmo Slowstart

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



CdC in TCP: AIMD

Una volta raggiunta la soglia, si entra in un comportamento detto Congestion Avoidance:

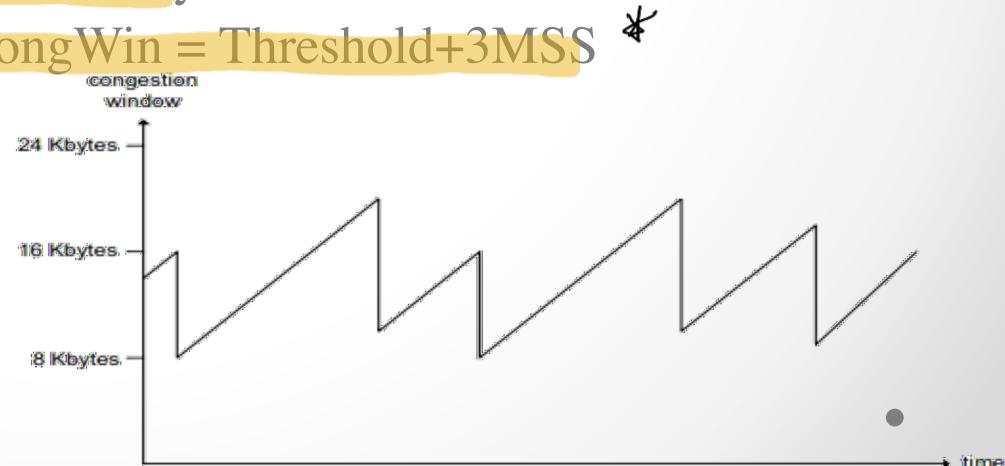
- ci si avvicina con cautela al valore della banda disponibile tra le due estremità della connessione (Additive Increase, AI)
 - incremento di CongWin di $MSS \cdot (\text{MSS/Congwin})$ alla ricezione di un ACK

Al sopraggiungere della congestione (Multiplicative Decrease):

- scadenza di un timeout → Slow Start
 - Threshold = CongWin/2; CongWin = 1 MSS
- ricezione di tre ACK duplicati consecutivi
 - Se TCP versione Tahoe, stessa cosa di timeout → Slow Start
 - Se TCP versione Reno → Fast Recovery
 - Threshold = CongWin/2; CongWin = Threshold+3MSS *



Se ricevo 3 ACK non è
congestione totale, puoi così avvia



TCP Reno

TCP Reno elimina la fase di Slow Start dopo un evento di perdita dedotto dalla ricezione di tre ACK duplicati:

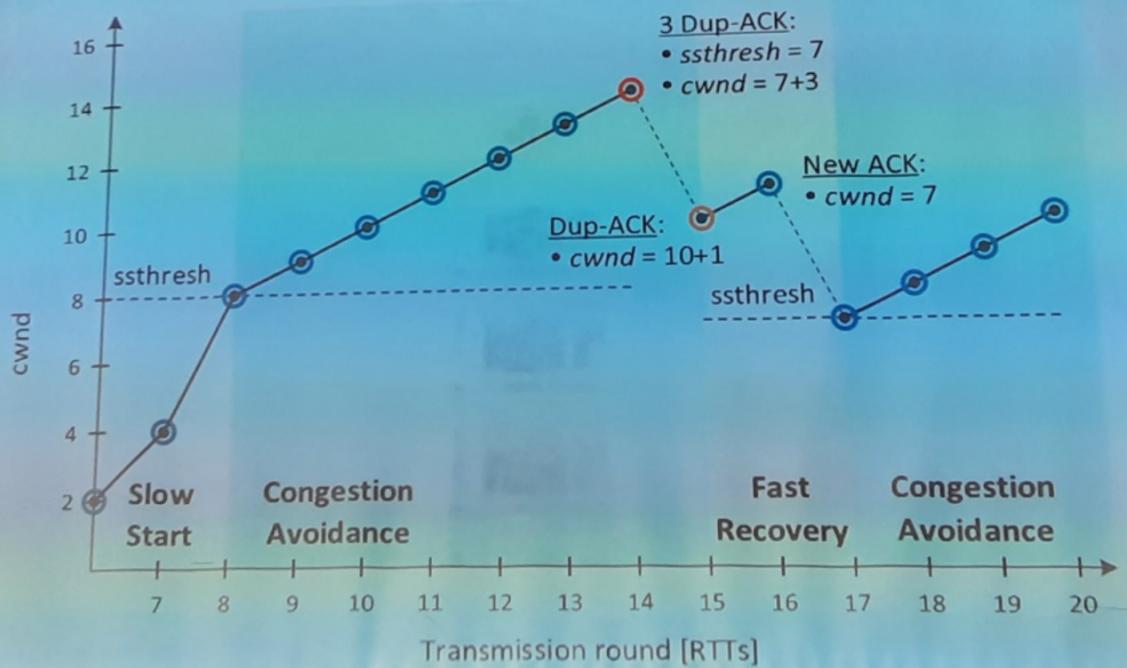
- tale evento indica che, nonostante si sia perso un pacchetto, almeno 3 segmenti successivi sono stati ricevuti dal destinatario:
 - a differenza del caso di timeout, la rete mostra di essere in grado di consegnare una certa quantità di dati
 - è possibile, quindi, evitare una nuova partenza lenta, ricominciando direttamente dalla fase di fast recovery

* Per ogni ACK duplicato la CWND aumenta di 1 segmento.
(permette la trasmissione di nuovi dati) → (permette trasmissione)

- Quando la sorgente riceve l'ACK che conferma ricezione del segmento ritrasmesso.
- Si pone $CWND = SS + ARSH$
 - Si procede la trasmissione con nel congestion avoidance

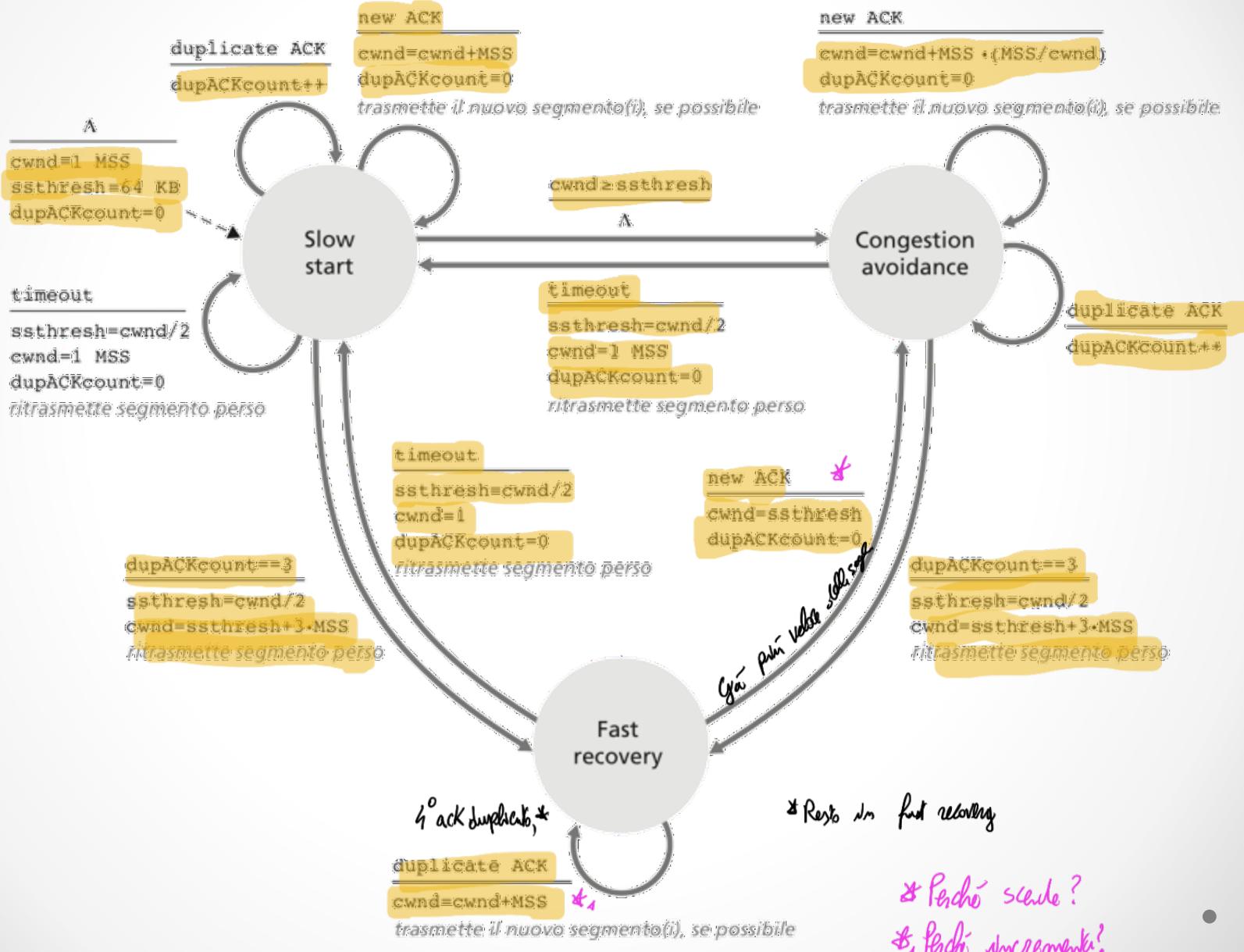
Se scade timeout si va

CdC TCP

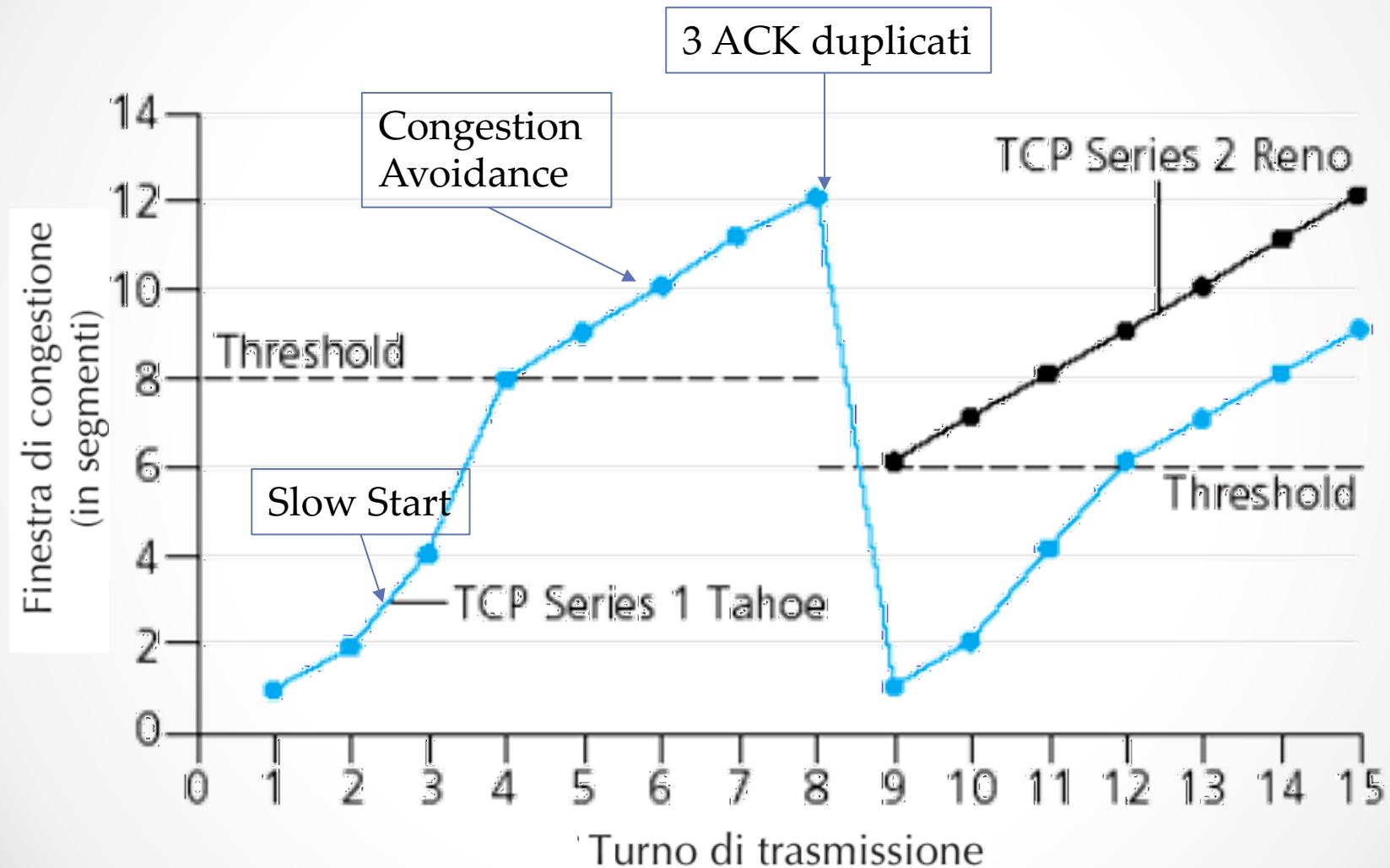


Se perde 2 pacchetti, il Rthmo si avanza, ma sicuramente valo un incremento per $\Delta 2^0$.

CdC TCP: modello a stati finiti



CdC TCP: modello a stati finiti

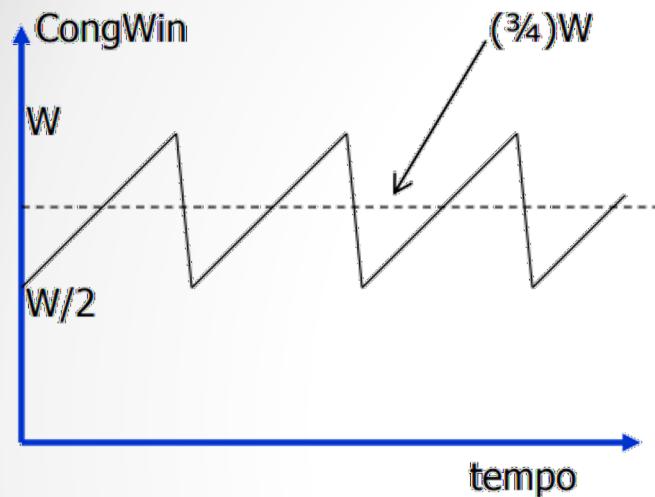


Ricapitolando

- Finestra di congestione sotto la soglia:
 - Slow start
 - Crescita esponenziale della finestra
- Finestra di congestione sopra la soglia:
 - Prevenzione della congestione Cong. avoid
 - 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 + 3 MSS
 - 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)



Throughput medio



Se il massimo valore di CongWin è costante (W) ed anche RTT è costante, il throughput medio di una connessione TCP è

$$T = \frac{0,75 \cdot W}{RTT} = \frac{W^*}{RTT}$$

s/Alma

dove $W^* = \text{valore medio di CongWin}$

Se si desidera un throughput medio con deve essere:

$$T = 10 \text{ Gbps} = 10^{10} \text{ b/s} = 1250 \cdot 10^6 \text{ byte/s}$$
$$RTT = 100 \text{ ms}$$

$$W^* = T \cdot RTT = 1250 \cdot 10^6 \cdot 10^{-1} \text{ byte} =$$
$$= 125 \cdot 10^6 \text{ byte}$$

Se 1 MSS = 1500 byte, esprimendo la dimensione media in MSS, si ha:

$$W^* = (1250/1500) \cdot 10^5 \text{ MSS} = 83333 \text{ MSS}$$

Se voglio un throughput mediamente al valore medio deve essere quello di W^* come funzione di congestione

Throughput medio

- La stima del throughput di una connessione TCP è stata oggetto di diversi studi
- La formula semplificata

$$T = \frac{0.75 \cdot W}{RTT}$$

ignora l'effetto della perdita di pacchetti

- In (*) è presentato un modello leggermente più accurato che tiene in conto della perdita di pacchetti
- Detto L il tasso medio di perdita di un pacchetto, è possibile ricavare:

$$T = \frac{C \cdot W}{RTT \cdot \sqrt{L}}$$

con $C = 1.22 = \sqrt{3/2}$

- (*) Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis.

The macroscopic behavior of the TCP congestion avoidance algorithm.

ACM SIGCOMM Computer Communications Review, vol. 27, issue 3, July 1997, pp. 67-82

Equità connessioni TCP

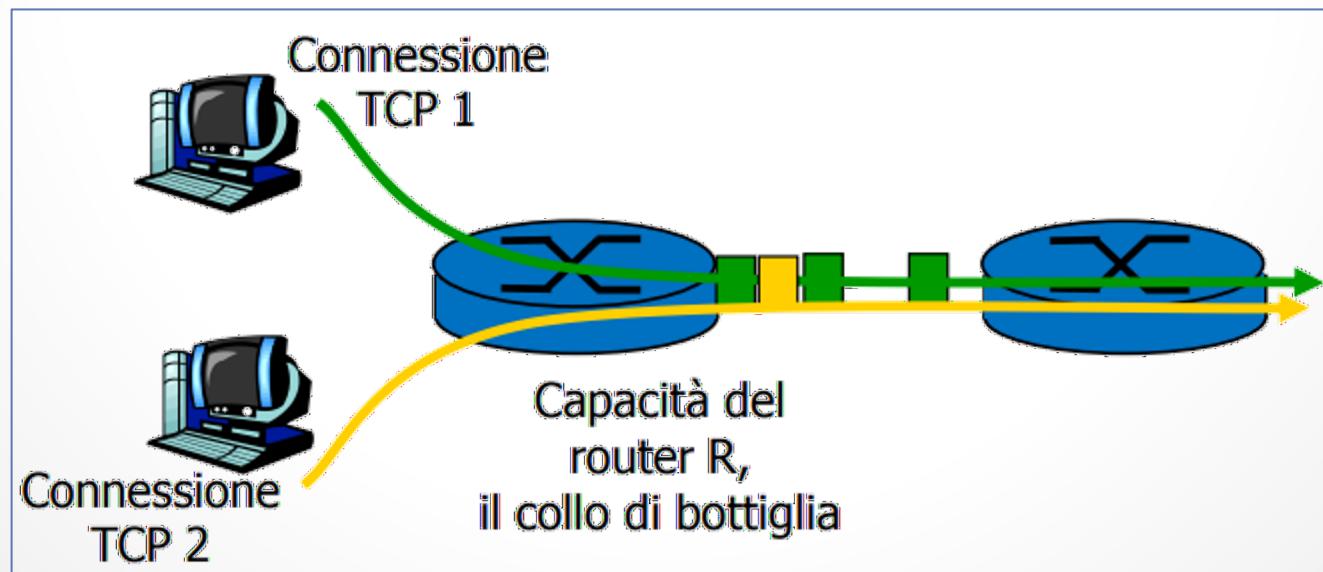
Se K sessioni TCP condividono lo stesso collegamento con ampiezza di banda R , che è un collo di bottiglia per il sistema, ogni flusso dovrebbe avere un tasso di trasmissione medio pari a $R/K \rightarrow$ equità (fairness)

TCP può portare a garantire questo?

Teoricamente sì, ma se RTT è

Lo stesso: se una ha RTT basso fa prima ad occupare banda.

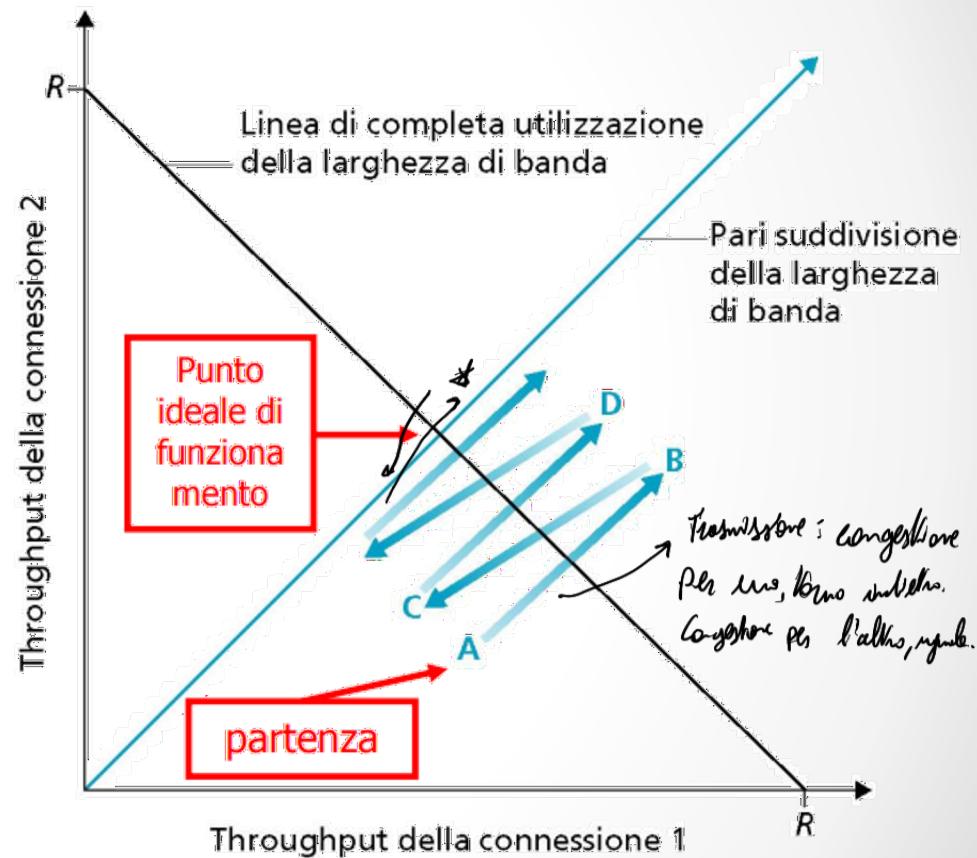
TCP è equo?



Equità connessioni TCP

Ipotesi:

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



* Qui ci spostiamo a saltare così.
Se man ha RTT segnali arriverà prima bandita.