

LIVELLO DI TRASPORTO → fornire servizi di comunicazione direttamente ai processi applicativi in esecuzione su host differenti.

un protocollo mette a disposizione una comunicazione logica tra processi applicativi differenti.

||  
tutto procede come se gli host che eseguono i processi fossero direttamente connessi

LATO MITTENTE: converte i messaggi in **SEGMENTI** che vengono passati al network layer

LATO RICEVENTE: il network layer estraе i segmenti e li passa al livello superiore (trasporto).  
Quest'ultimo li riassembra e passa i messaggi all'application layer.

### TCP TRANSMISSION CONTROL PROTOCOL

- affidabile
- orientato alla connessione
- consegna in ordine
- controllo di congestione

### UDP USER DATAGRAM PROTOCOL

- non affidabile
- non orientato alla connessione
- consegna non ordinata ⇒ miglior performance
- dimensioni ridotte dell'intestazione (assenza di controlli)
- può funzionare quando il servizio di rete è compromesso

No garanzie di ritardo e di bandwidth

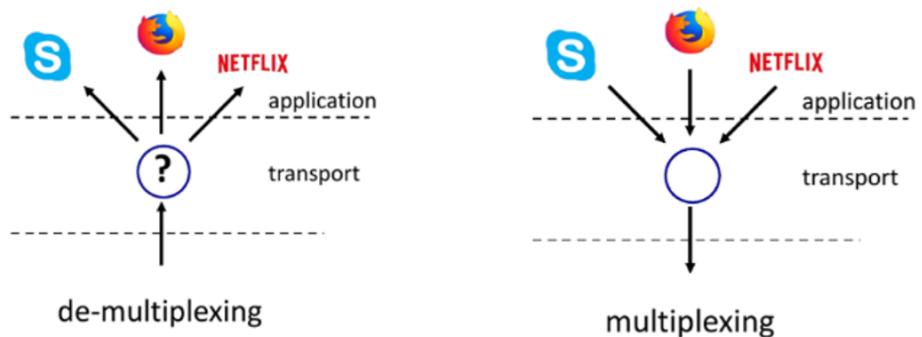
## MULTIPLEXING E DEMULTIPLEXING

= come il servizio di trasporto da host a host fornito dal network layer possa diventare un servizio di trasporto da processo a processo per le applicazioni in esecuzione sugli host.

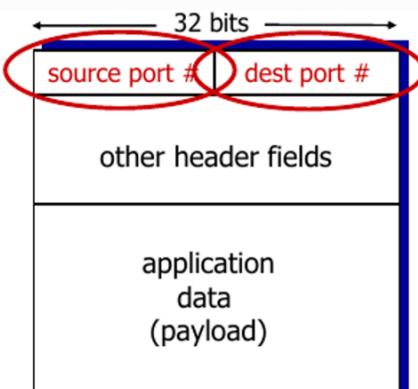
LATO RICEVENTE : il transport layer identifica la socket di ricezione e vi dirige il segmento.

DEMULITPLEXING = compito di trasportare i dati dei segmenti verso la giusta socket.

MULTIPLEXING = compito di radunare i frammenti di dati da diverse socket sull'host di origine e incapsulare ognuno con intestazioni per creare dei segmenti e passarli al network layer.



DEMULITPLEXING CON UDP :



TCP/UDP segment format

- ogni riga di intestazione ha 32 bit
  - campi : numero di porta + numero di porta d'origine
- sulla prima riga

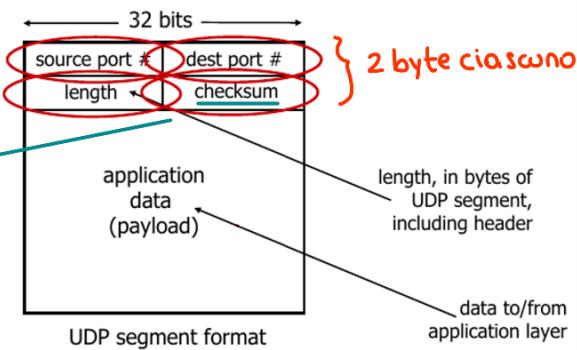
Quando viene definita una socket , bisogna specificare il numero della porta

DEMULITPLEXING CON TCP : identificata da 4 parametri

[ IP mittente, IP destinazione , Porta mittente, Porta destinazione ]

TRASPORTO UDP : utilizzato per streaming, DNS, SNMP, HTTP/3

STRUTTURA DEI SEGMENTI :



CHECKSUM

||

identifica errori nel segmento

determina se i bit del segmento  
UDP sono stati alterati durante  
il trasferimento

↓

LATO MITTENTE : effettua il complemento a 1 di somma di tutte le parole da 16 bit nel segmento, e l'eventuale riporto finale viene sommato al primo bit

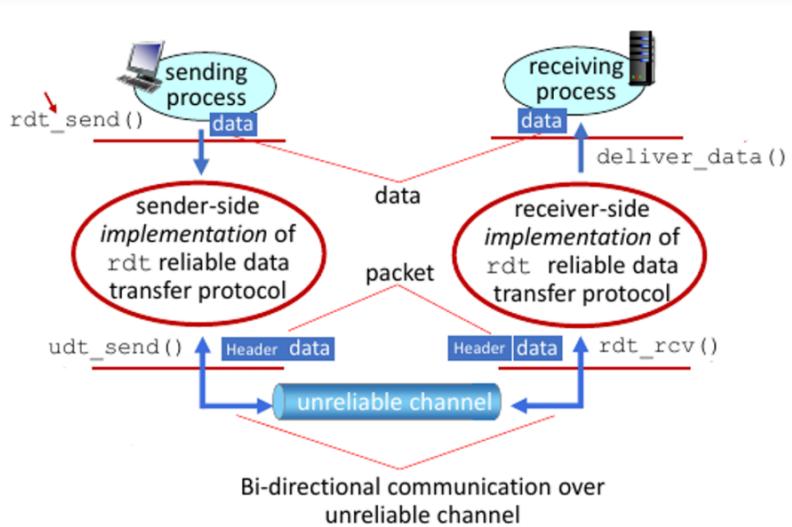
LATO RICEVENTE : si sommano le parole iniziali e il checksum.

Se non ci sono errori nel pacchetto, l'addizione darà 11...1.

RELIABLE DATA TRANSFER (rdt) = TRASFERIMENTO DATI AFFIDABILE

→ protocollo banale : usiamo FSM (macchine a stati finiti)

→ mittente e ricevente non conoscono lo stato dell'altro



- **rdt 1.0** - trasferimento dati affidabile su un canale perfettamente affidabile

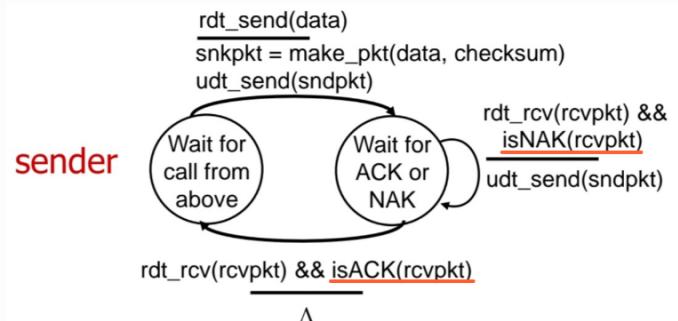
→ nessun bit dei dati trasferiti è corrotto o va perduto e tutti i bit sono consegnati nell'ordine di invio.



→ due FSM separate, una per mittente e una per destinatario

- **rdt 2.0** - trasferimento dati affidabile su un canale con errori di bit

→ usa ACKNOWLEDGMENT (ACKs)  
= notifiche  
e negative ACKNOWLEDGE (NAKs)  
che consentono al destinatario di  
far sapere al mittente l'esito del  
trasferimento, chiedendone eventualmente la ripetizione



→ Quando il mittente è nello stato di attesa di ACK e NAK, non può inviare nuovi dati (infatti rdt 2.0 == stop-and-wait)

→ **PROBLEMA** : possibile che ACK e NAK vengano alterati !

→ possibili metodi per risolvere :

- aggiunta di bit di checksum sufficienti per consentire al mittente di trovare e correggere gli errori sui bit. → **NON RISOLVO IL PROBLEMA DELLA PERDITA DI BIT**
- il mittente riinvia il pacchetto a seguito della ricezione di ACK o NAK alterato. → **PACCHETTI DUPLICATI**  
⇒ il destinatario non sa se l'ultimo ACK/NAK inviato sia stato ricevuto dal mittente

**Soluzione:** numerare i pacchetti con un **NUMERO DI SEQUENZA**

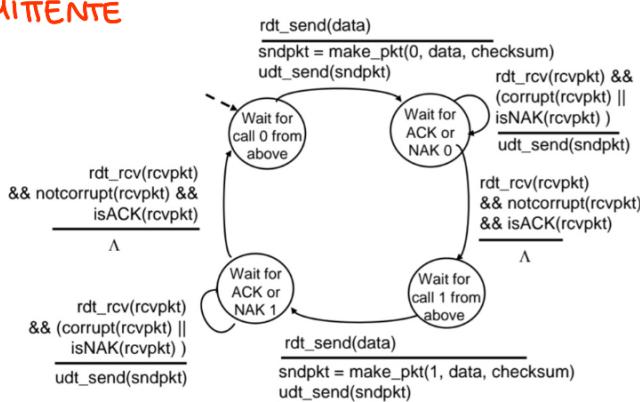
## • rdt 2.1 – HANDLING GARBET ACK/NAK

↳ usa ACK e NAK dal destinatario verso il mittente:

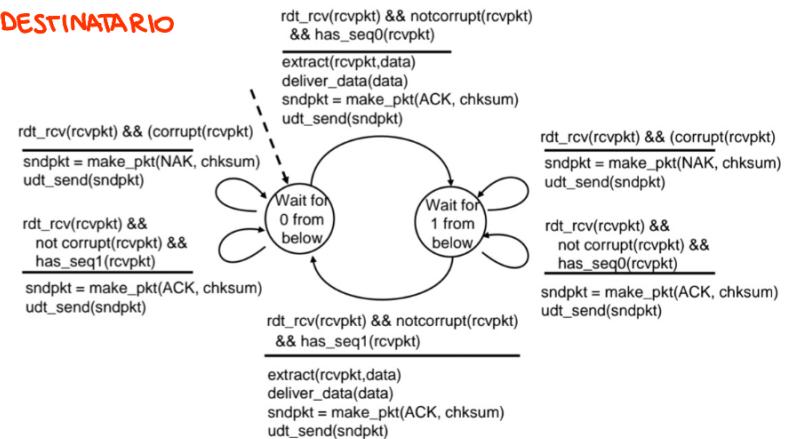
- ACK quando riceve un pacchetto fuori sequenza
- NAK quando riceve un pacchetto alterato

N.B. un mittente che riceve due ACK per lo stesso pacchetto sa che il destinatario non ha ricevuto correttamente il pacchetto successivo a quello confermato due volte.

### MITTENTE

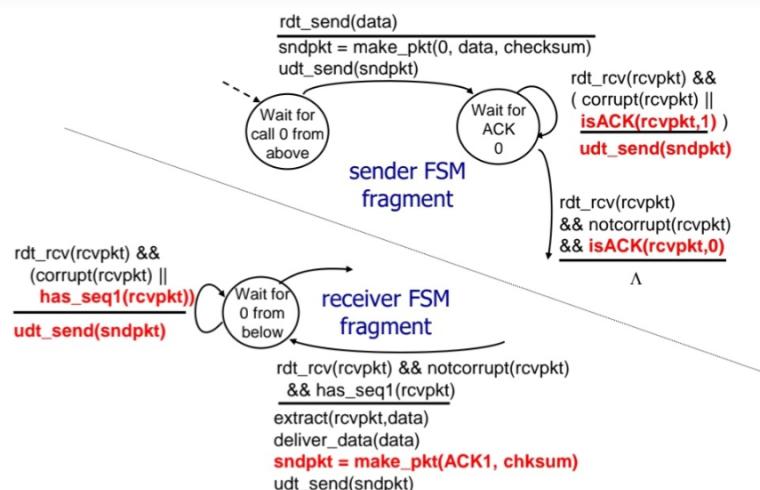


### DESTINATARIO

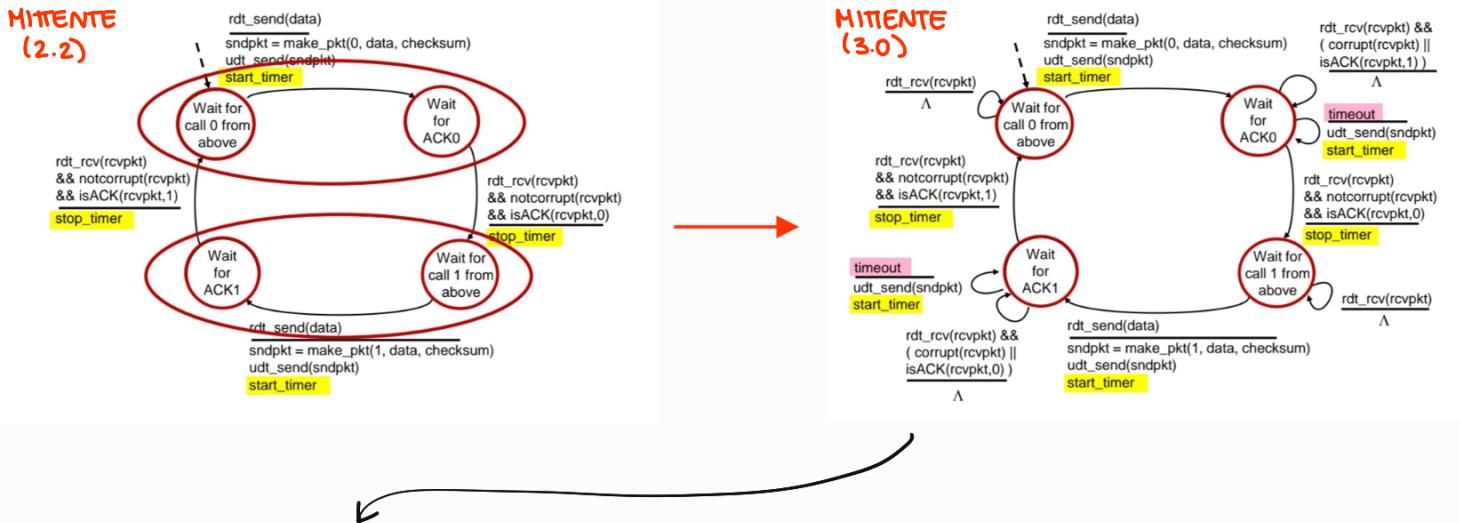


## • rdt 2.2 – NAK-FREE PROTOCOL

↳ il destinatario deve includere il numero di sequenza del pacchetto di cui invia l'ACK all'interno di quest'ultimo e il mittente deve controllare il numero di sequenza del pacchetto confermato da un ACK ricevuto. → non usa più isNAK



- rdt 3.0 – trasferimento dati affidabile su un canale con perdite ed errori su bit



il mittente attende : minimo ritardo di andata e ritorno  
+

tempo richiesto per l'elaborazione di un pacchetto  
da parte del destinatario.

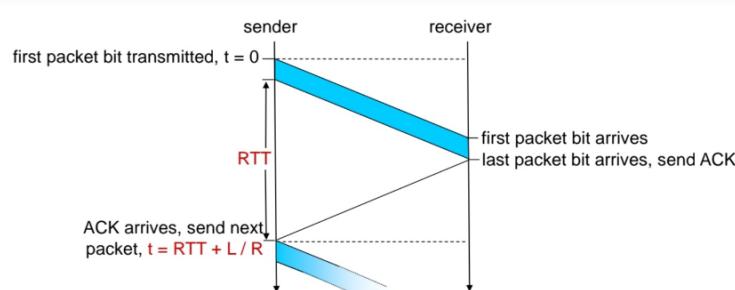
se in questo lasso di tempo non riceve ACK  $\Rightarrow$  ritrasmette il pacchetto  
 $\rightarrow$  possibilità di duplicati

rdt 3.0 anche detto **protocollo ad alternanza di bit**

### PROBLEMI :

#### ANDATA E RITORNO

- Ritardo di propagazione RTT
- $U_{\text{sender}}$  : **utilization** = frazione di tempo in cui il mittente è stato effettivamente occupato nell'invio di bit sul canale



$$U_{\text{sender}} = \frac{L/R}{\text{RTT} + L/R}$$

$\Rightarrow$  può limitare il rendimento dell'hardware di rete

SOLUZIONE : NO stop-and-wait  $\Rightarrow$  il mittente puo' inviare pacchetti senza attendere ACK

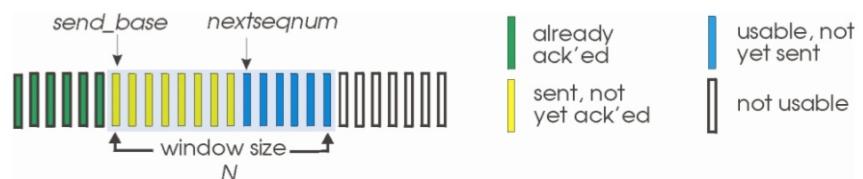
### tecnica PIPELING :

- l'intervallo dei numeri di sequenza disponibili va incrementato
  - I lati di invio e ricezione devono poter memorizzare in un buffer piu' di un pacchetto
- $\hookrightarrow$  due approcci di base : Go-back-N e ripetizione selettiva

### GO-BACK-N (GBN o protocollo a finestra scorrevole)

Il mittente puo' trasmettere piu' pacchetti senza dover attendere alcun acknowledgement, ma non puo' avere piu' di un dato numero massimo consentito di pacchetti  $N$  in attesa di ACK nella pipeline.

#### Lato mittente :

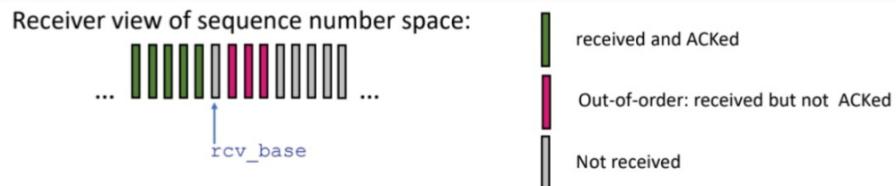


- CUMULATIVE ACK : indica che tutti i pacchetti con un numero di sequenza minore o uguale a  $n$  sono stati correttamente ricevuti dal destinatario.

$\rightarrow$  quando riceve ACK( $n$ ) sposta la finestra a  $n+1$

- timeout for older : alla ricezione di timeout( $n$ ) il mittente ri-invia tutti i pacchetti spediti che non hanno ancora ricevuto un ACK

#### Lato destinatario :

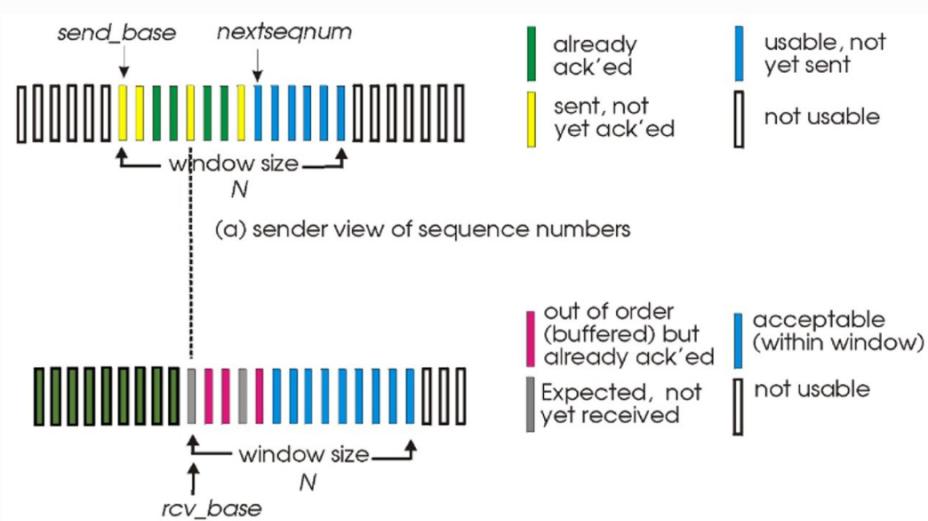


- ACK-only : manda un ACK per il pacchetto ricevuto correttamente e consegna i suoi dati al livello superiore e in ordine!

- Scarta i pacchetti fuori sequenza (=corretti ma non in ordine) e rimanda un ACK per il pacchetto in ordine ricevuto piu' di recente

## RIPETIZIONE SELETTIVA (SR)

→ evita le ritrasmissioni non necessarie



Lato destinatario :

- manda ACK specifici per i pacchetti ricevuti in modo corretto.
  - ↳ invia un riscontro sia per quelli in ordine che fuori sequenza
  - ↳ memorizzati in un buffer finche' non sono stati ricevuti i pacchetti mancanti (= con num di sequenza piu' bassi)

Lato mittente :

- timer per ogni pacchetto per cui non e' stato ricevuto ACK
- Se si riceve un ACK, etichetta il pacchetto come ricevuto (se e' nella finestra)

# Trasporto TCP

- orientato alla connessione → prima di scambiarsi i dati, i processi effettuano l'handshake
- lo stato della connessione risiede completamente nei due sistemi periferici  
↳ i router intermedi sono ignari delle connessioni.
- offre un servizio **full-duplex**: i dati fluiscano in maniera bidirezionale
- c'è **PUNTO A PUNTO**: ha luogo tra un singolo mittente e un singolo destinatario.
- consegna affidabile e in ordine
- controllo del flusso e di congestione
- offre **ACKNOWLEDGE CUMULATIVI** e **PIPELINING**
  - perche' TCP effettua ack solo dei byte fino al primo byte mancante nel flusso

## STRUTTURA DEI SEGMENTI TCP :

consiste di campi intestazione e di un campo contenente un blocco di dati provenienti dall'applicazione.

L' INTESTAZIONE include:

- numeri di porta di origine e numeri di porta di destinazione
- un campo checksum
- **numero di sequenza** e numero di acknowledgement (entrambi 32 bit)
- finestra di ricezione (16 bit) per controllo del flusso
- lunghezza dell'intestazione (4 bit)
- campo options (facoltativo e di lunghezza variabile)
- campo flag (6 bit) → il bit **ACK** usato per indicare se il valore trasportato nel campo di ack è valido.

N.B.

e' il numero nel flusso di byte del primo byte del segmento

(in quanto TCP vede i dati come un flusso di byte ordinati)

## TIMEOUT e STIMA DEL TEMPO A/R

Il Timeout dovrebbe essere maggiore del tempo di andata e ritorno sulla connessione (RTT)

bisogna settarlo correttamente

- ( troppo corto = timeout prematuro )
- ( troppo lungo = reazione lenta alle perdite )

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

media ponderata dei valori di SampleRTT

valore raccomandato 0,125

calcolato per i segmenti trasmessi una sola volta

in statistica è detta **MEDIA MOBILE ESPONENZIALE PONDERATA** (EWMA)

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

NON PUO' ESSERE INFERIORE A EstimatedRTT

variazione RTT = stima di quanto SampleRTT si discosta da EstimatedRTT

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

## Come si instaura la connessione ?

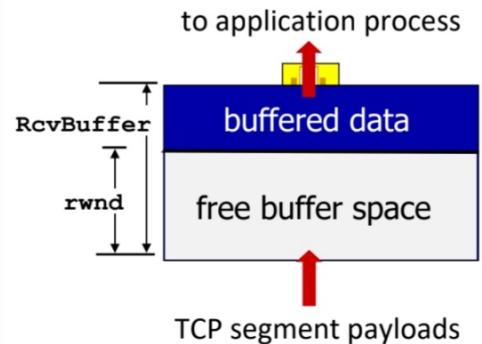
- 1) Il client invia per primo uno speciale segmento TCP ;
  - 2) Il server risponde con un secondo segmento speciale TCP ;
  - 3) Il client risponde con un terzo segmento speciale .
- ① e ② non trasportano payload (NO dati a livello app)
- ③ può trasportare informazioni utili

**HANDSHAKE A TRE VIE**  
(three-way handshake)

**FLOW CONTROL** : il ricevente controlla il mittente, in modo che questo non mandi in overflow il buffer del destinatario trasmettendo troppi dati e troppo velocemente.

impostato nel campo TCP  
**receive windows (rwnd)**

che indica lo spazio libero nel buffer del destinatario.



## CONGESTION CONTROL

CONGESTIONE = troppe fonti inviano troppi dati troppo velocemente perche' la rete possa gestirli.  $\Rightarrow$  **lunghi ritardi e perdite.**

CAUSE :

scenario 1: due host con una connessione che condivide un singolo router intermedio.

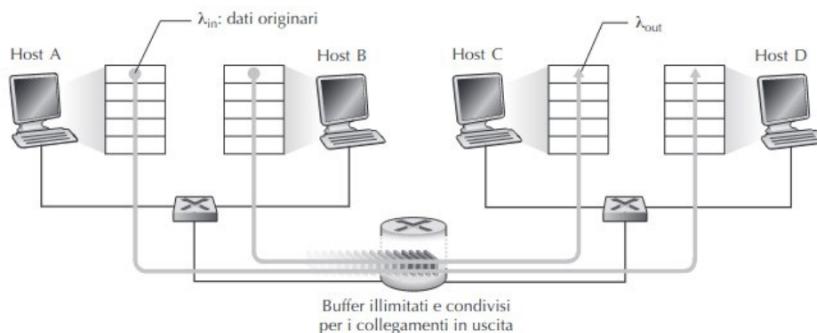


Figura 3.43 Scenario di congestione 1: due connessioni che condividono un hop con buffer illimitato.

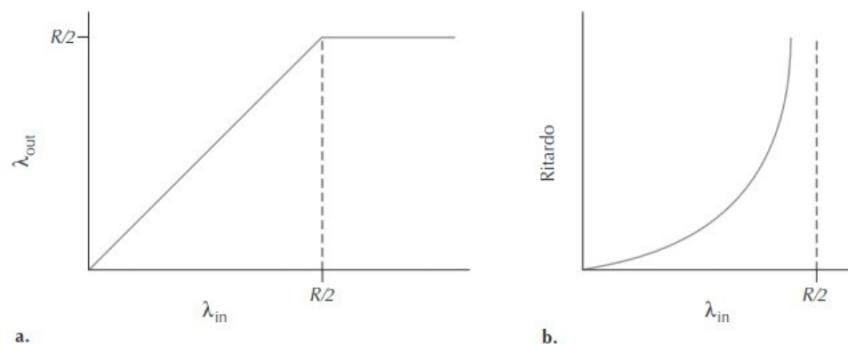


Figura 3.44 Scenario di congestione 1: throughput e ritardi in funzione della frequenza trasmittiva dell'host.

Scenario 2 : due host e un router con buffer di dimensione limitata.

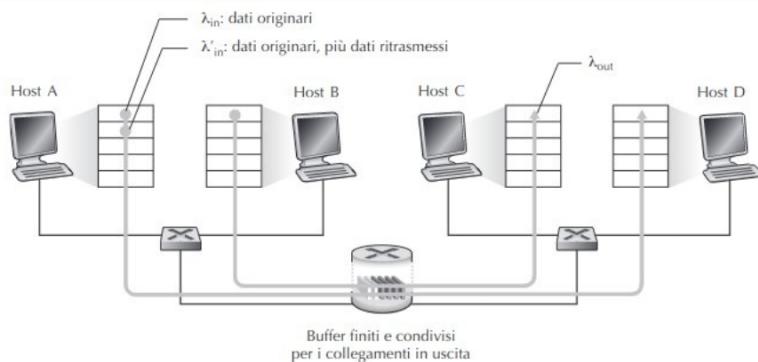


Figura 3.45 Scenario 2: due host (con ritrasmissioni) e un router con buffer di dimensione finita.

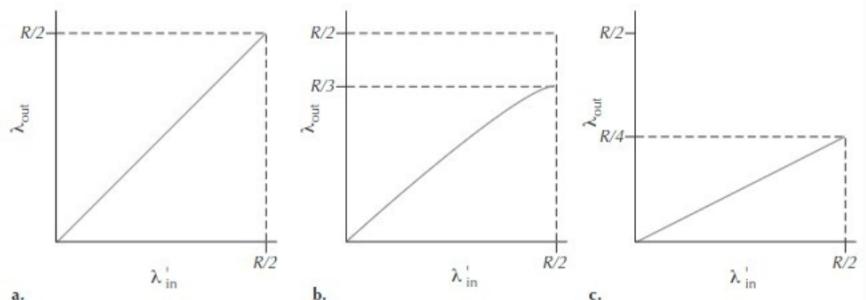


Figura 3.46 Prestazioni dello Scenario 2 con buffer di dimensione finita.

Scenario 3: quattro host mittenti, ciascuno su percorsi composti da due collegamenti sovrapposti tra loro ; ciascun host utilizza un meccanismo di timeout e ritrasmissione ; router con buffer di dimensione finita.

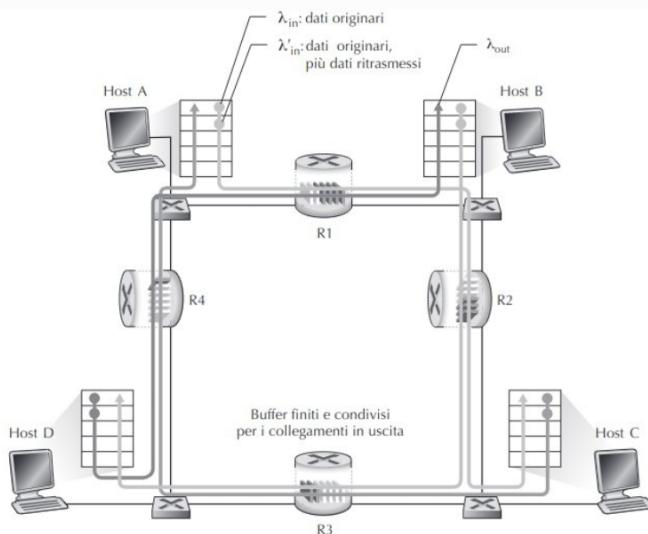


Figura 3.47 Scenario 3: quattro mittenti, router con buffer di dimensione finita e percorsi multihop.

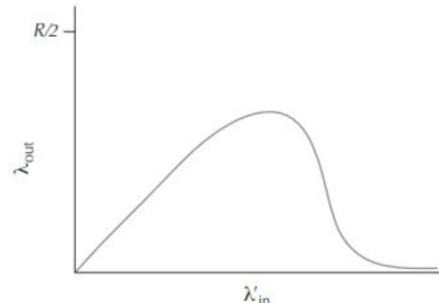


Figura 3.48 Prestazioni dello Scenario 3 (buffer di dimensione finita e percorsi multihop).

DUE APPROCCI :

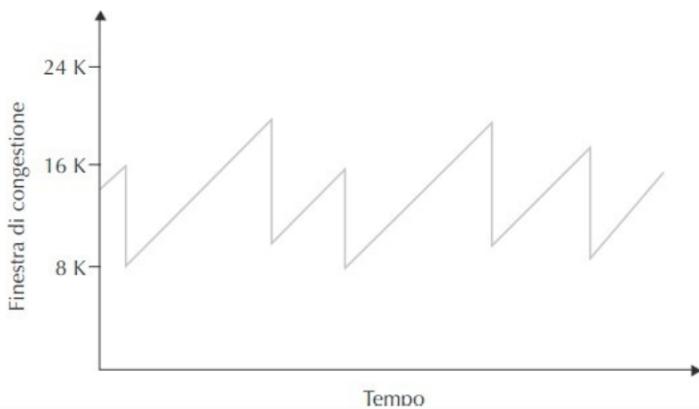
- Controllo di congestione **end-to-end** : nessun feedback esplicito; la congestione e' dedotta dalla perdita e dal ritardo.
- Controllo di congestione **network-assisted** : i router forniscono un feedback diretto agli host mittente / ricevente.

## CONTROLLO DI CONGESTIONE TCP → adotta controllo end-to-end

Approccio: imporre a ciascun mittente un limite alla velocità di invio sulla propria connessione in funzione della congestione percepita.

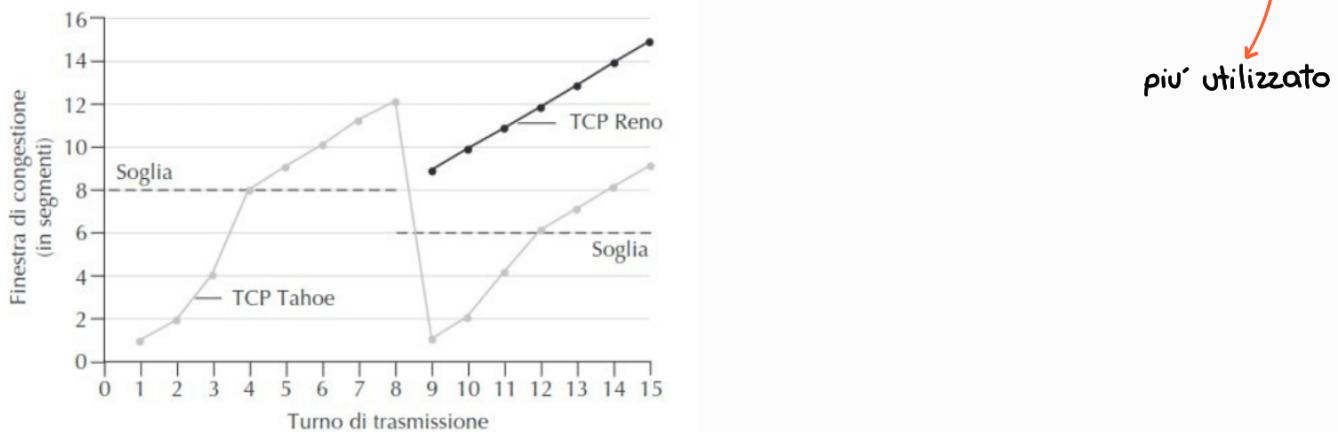
### ADDITIVE INCREASE, MULTIPLICATIVE DECREASE (AIMD) :

- avvio lento, aumenta la velocità di invio di 1 MSS ad ogni RTT finché non si rileva una perdita, poi dopo ogni perdita dimezza. maximum segment size
- + ottimizza il flusso congestionato, è stabile e presenta una divisione equa.
- è variabile e lento ad aumentare.



### TCP AIMD MULTIPLE DECREASE :

- si dimezza in caso di perdita rilevata da triplo ACK duplicato (TCP Reno)
- va a 1 MSS quando una perdita viene rilevata per timeout (TCP Tahoe)



Il mittente TCP limita la trasmissione :  $\text{LastByteSent} - \text{LastByteAcked} < \text{cwnd}$

$$\text{TCP rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

### 3 STATI :

- **SLOW START** - quando si stabilisce una connessione TCP, il valore di cwnd viene inizializzato a 1 MSS  $\Rightarrow$  velocità di invio iniziale di circa  $MSS/RTT$ .  
cwnd si incrementa di 1 MSS ad ogni ACK ricevuto  
(crescita esponenziale)

Terminazione:

- con un evento di perdita indicato da un evento di timeout  
(il mittente rimette  $cwnd = 1$  e riparte slow start)
- $cwnd == ssthresh \rightarrow$  modalità congestion avoidance

- **CONGESTION AVOIDANCE** - il valore di cwnd è circa la metà di quello che aveva l'ultima volta in cui era stata rilevata una congestione.

TCP incrementa cwnd di 1 MSS ogni RTT (incremento lineare)

Terminazione:

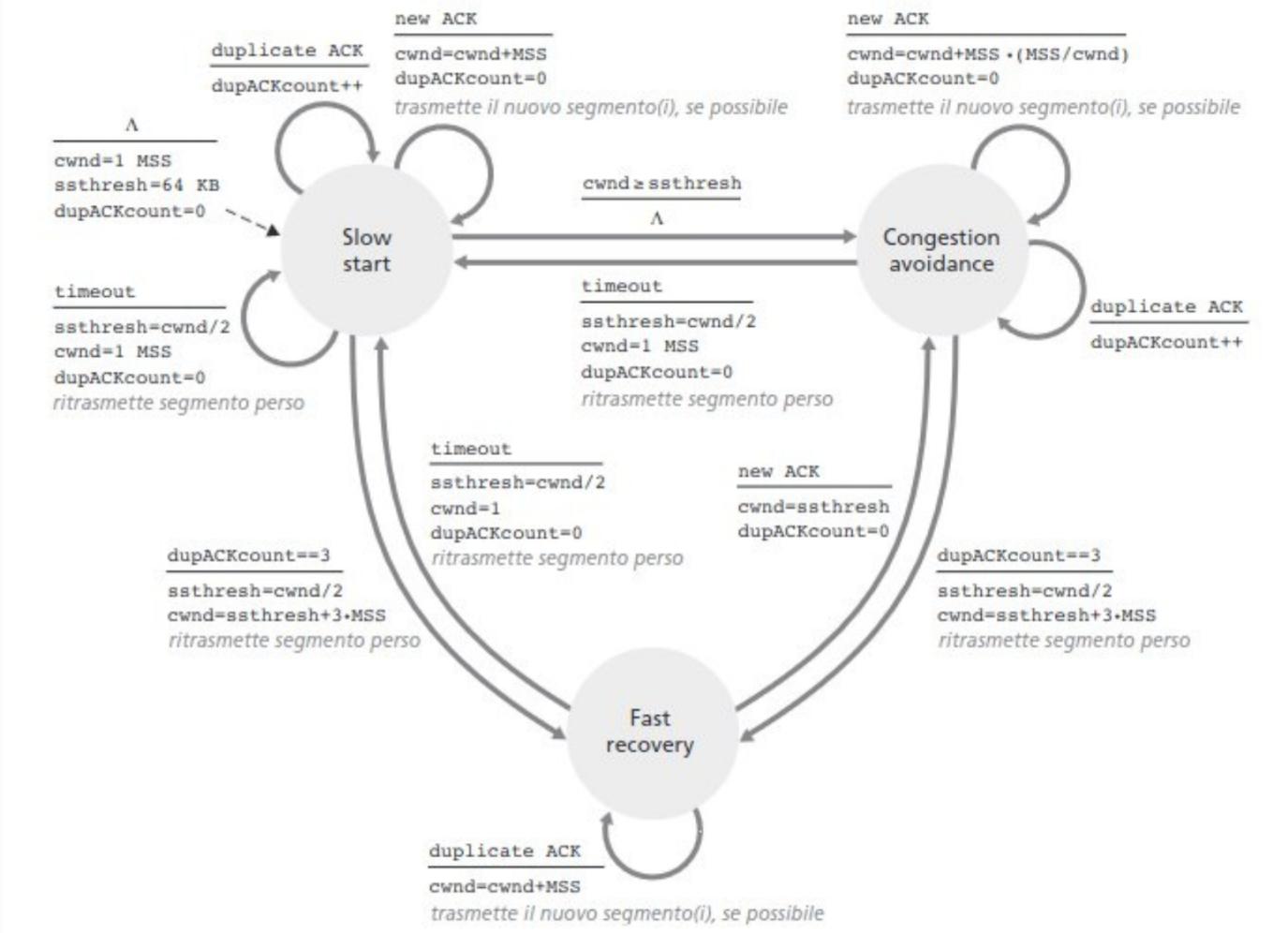
- quando si verifica un timeout fa come slow start
- se si ricevono tre ack duplicati (perdita) TCP dimezza il valore di cwnd e imposta  $ssthresh = cwnd/2$  e entra nello stato di fast recovery.

- **FAST RECOVERY** - cwnd è incrementato di 1 MSS per ogni ACK duplicato ricevuto relativamente al segmento perso che ha causato l'entrata nello stato attuale.

Quando arriva un ACK per il segmento perso, TCP entra nello stato di congestion avoidance dopo aver ridotto il valore di cwnd.

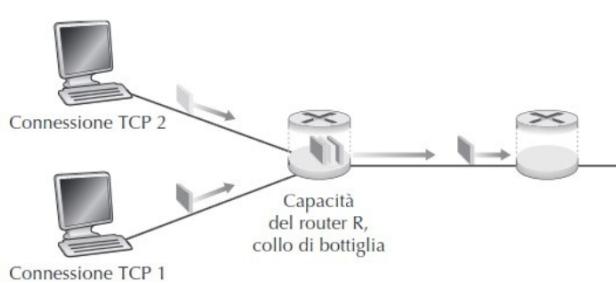
Se si verifica timeout si va allo stato di slow start dopo aver impostato  $ssthresh = cwnd/2$  e  $cwnd = 1$  MSS

FAST RECOVERY è raccomandato ma NON OBBLIGATORIO



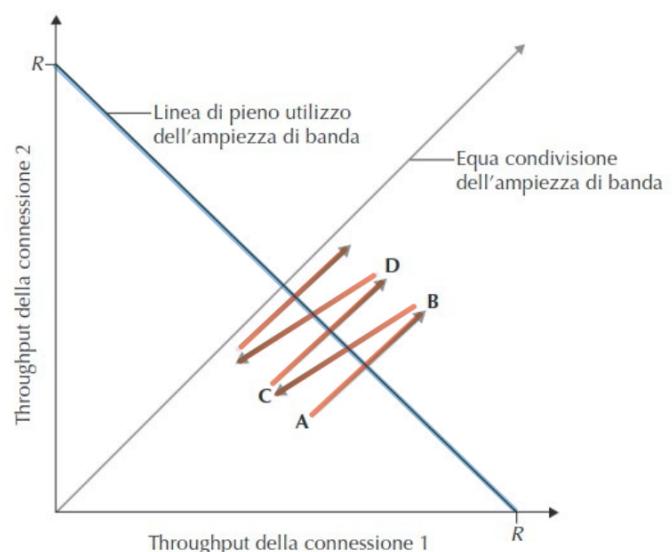
## FAIRNESS

Un controllo di congestione si dice **fair** (= equo) se la velocità media di ciascuna connessione è circa  $\frac{R}{K}$  capacità trasmissiva del router numero di connessioni TCP



Due connessioni TCP che condividono un singolo collegamento che fa da collo di bottiglia.

Se RTT è diverso, anche la crescita lo è.



**DELAY-BASED TCP (BBR):** mantiene l'RTT minimo che e' considerato migliore e senza congestione e, poiche' il ritardo e' esponenziale, cerca di mantenerlo vicino al minimo.