



# **7 – Il Livello di Trasporto**

## **Parte I**

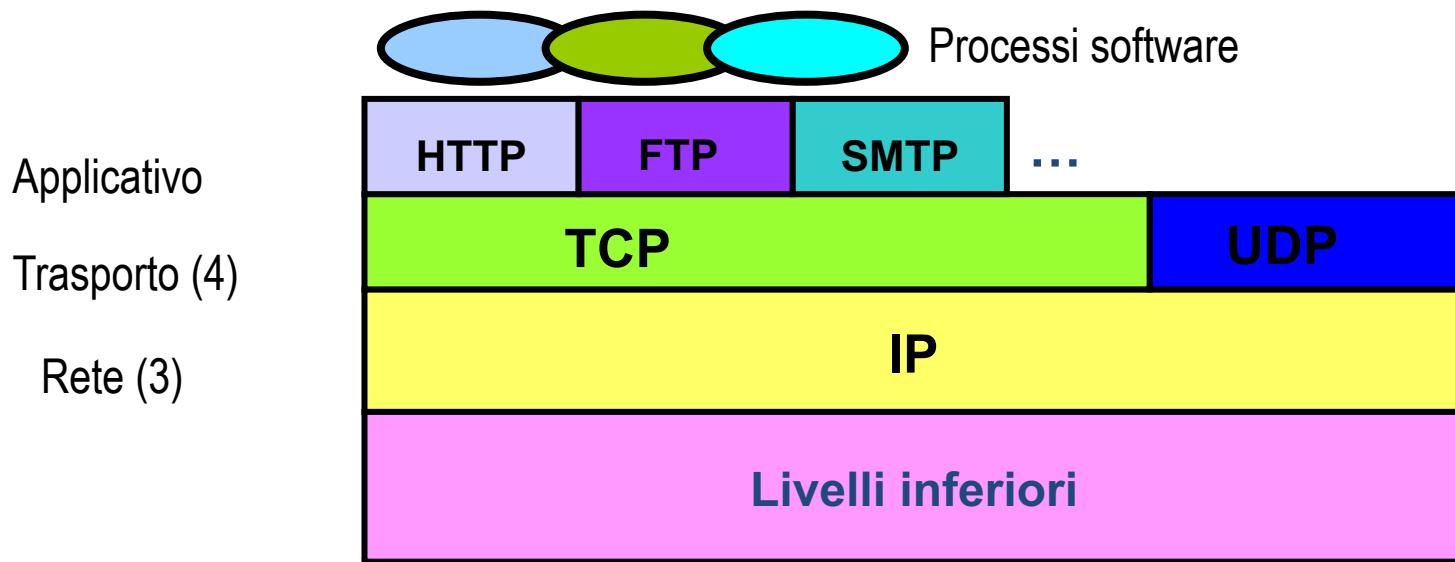
# Livello di Trasporto

- **Introduzione**
- **Protocollo UDP**
- **Trasporto affidabile**
  - Protocolli di ritrasmissione
  - Controllo di flusso a finestra mobile
- **Protocollo TCP**
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



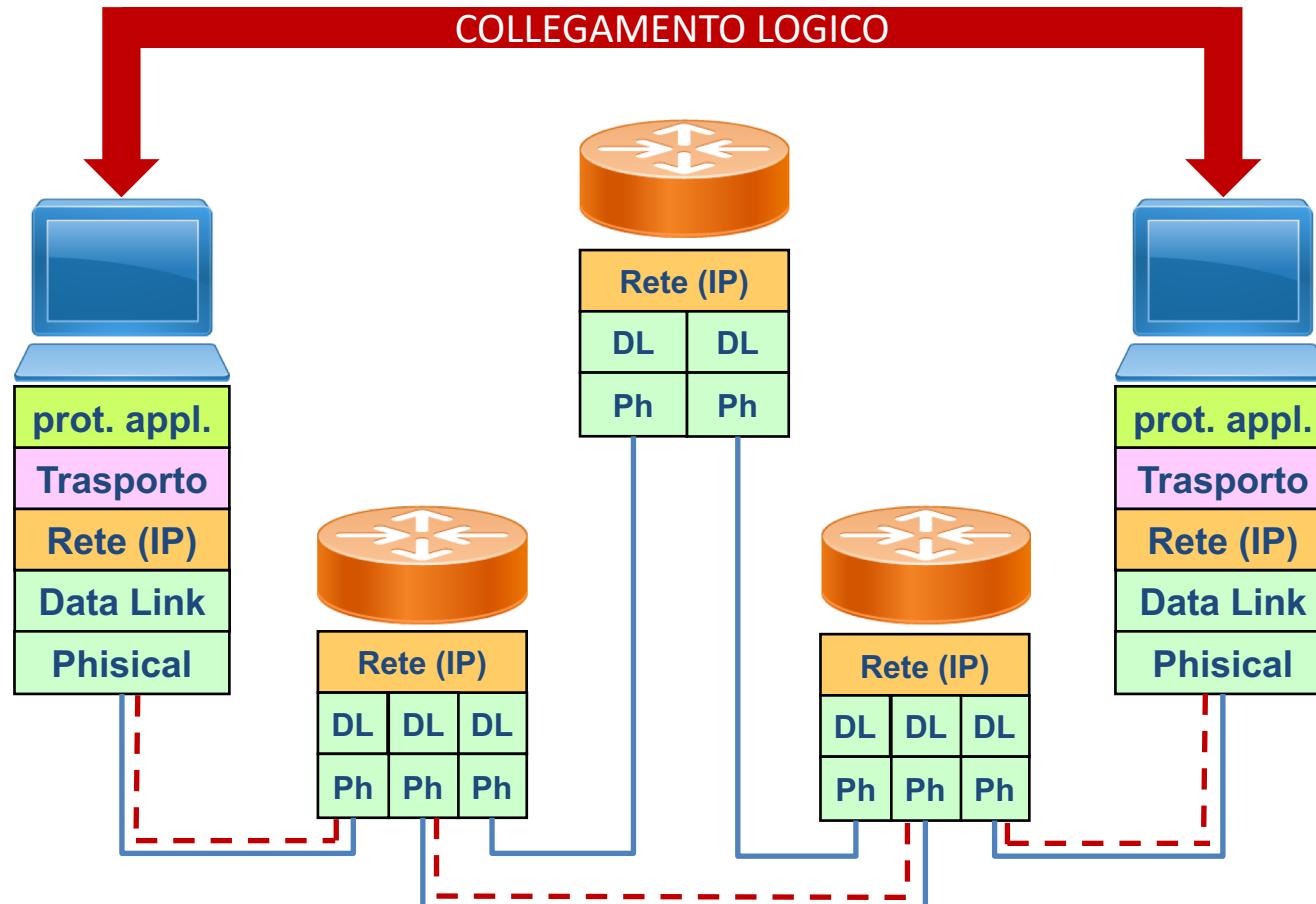
# Servizio di trasporto

- Il livello di trasporto ha il compito di instaurare un collegamento logico tra le applicazioni residenti su host remoti
- Il livello di trasporto rende trasparente il trasporto fisico (attraverso la rete) dei messaggi alle applicazioni



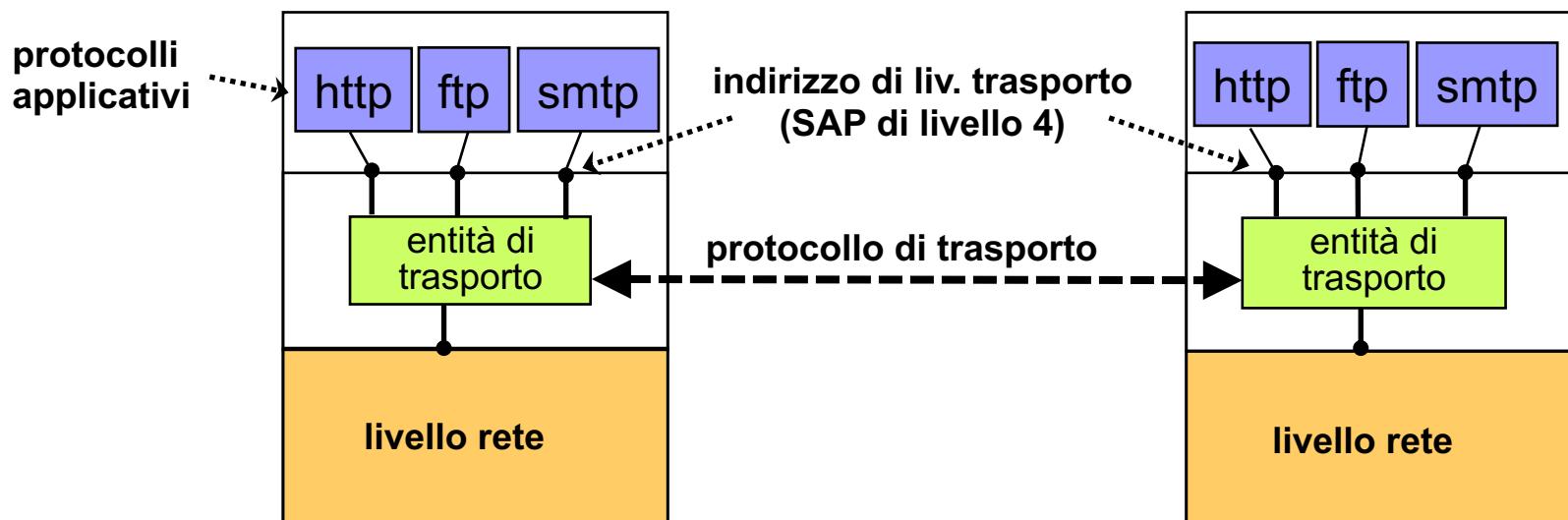
# Servizio di trasporto

- Il livello di trasporto è presente solo negli *end systems (host)*
- Esso consente il collegamento logico tra processi applicativi



# Servizio di trasporto

- Più applicazioni possono essere attive su un *end system*
  - Il livello di trasporto svolge funzioni di *multiplexing/demultiplexing* per applicazioni
  - Ciascun collegamento logico tra applicazioni è indirizzato dal livello di trasporto



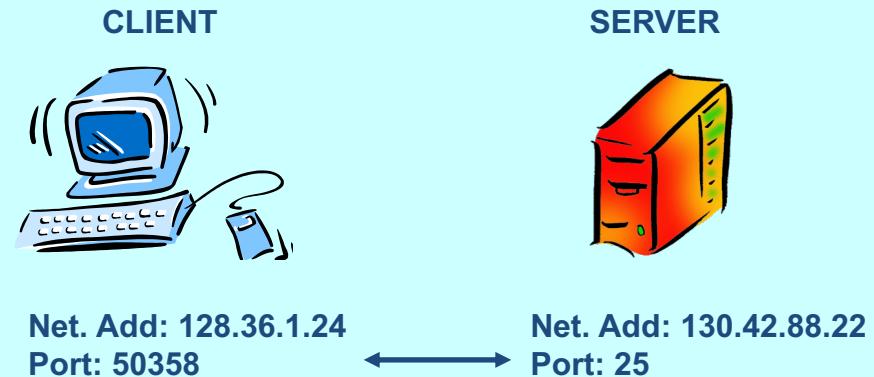
# Indirizzamento: le *porte*

- In Internet le funzioni di *multiplexing/demultiplexing* vengono gestite mediante indirizzi contenuti nelle PDU di livello di trasporto
- Tali indirizzi sono lunghi 16 bit e prendono il nome di *porte*
- I numeri di porta possono assumere valori compresi tra 0 e 65535
- I **numeri noti** sono assegnati ad importanti applicativi dal lato *server* (HTTP, FTP, SMTP, DNS, ecc.)
- I **numeri dinamici** sono assegnati dinamicamente ai processi applicativi lato *client*
- I **numeri registrati** sono assegnati a specifiche applicazioni da chi ne faccia richiesta (tipicamente protocolli proprietari)



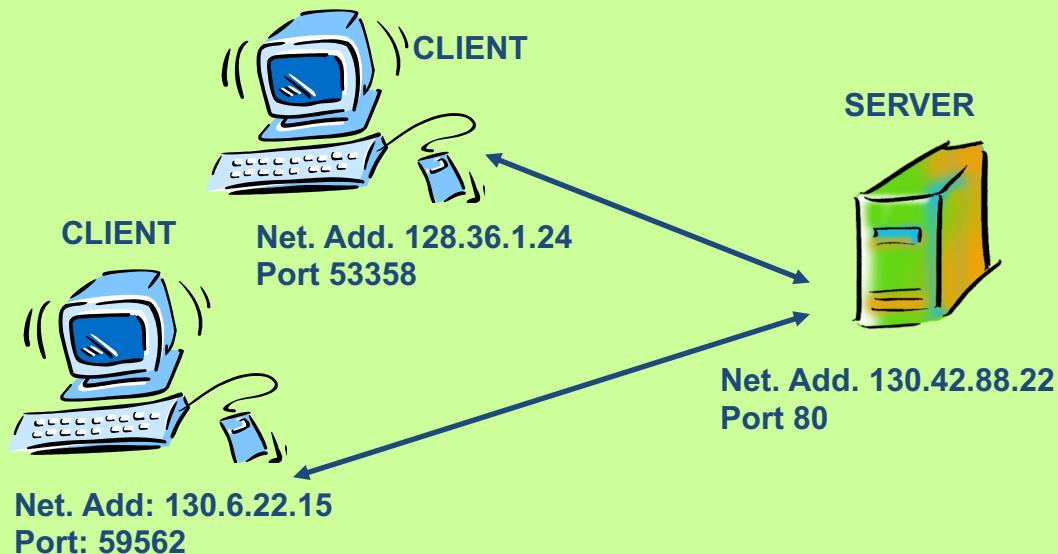
# Socket e multiplazione

Un *client* trasmette segmenti verso la porta di un server SMTP remoto



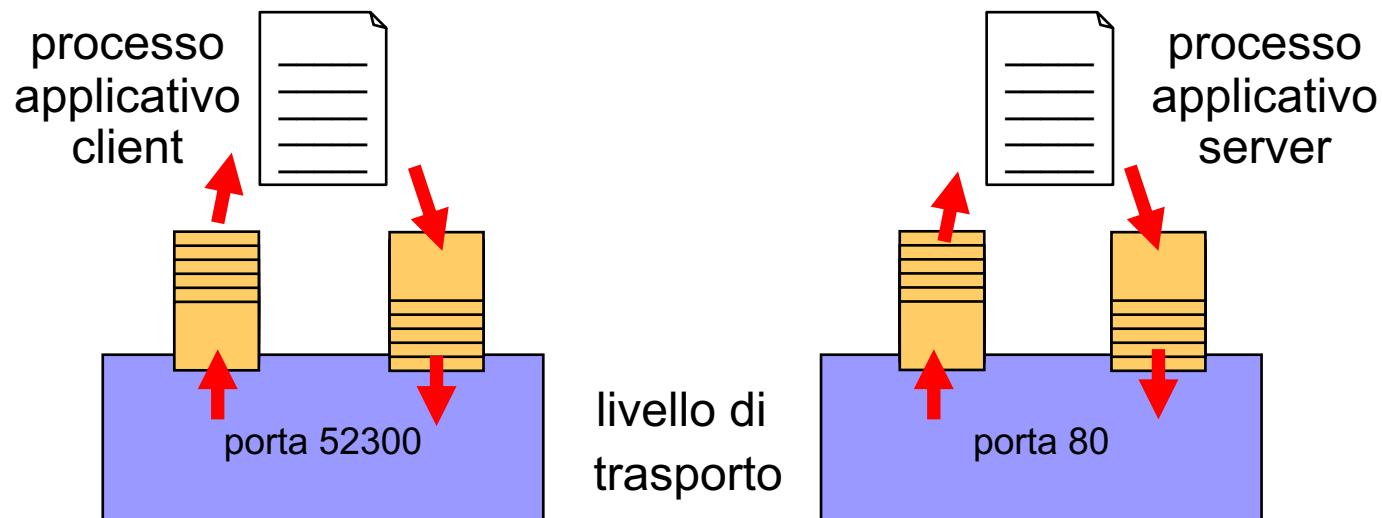
Due client accedono alla stessa porta di un server HTTP.

Non c'è comunque ambiguità, perché la coppia di socket è diversa



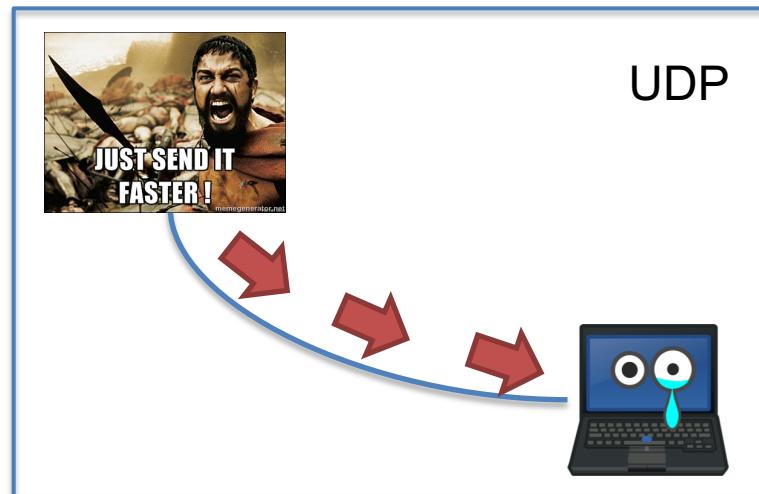
# Servizio di Buffering

- I protocolli di trasporto sono implementati nei più diffusi **sistemi operativi**
- Quando un processo viene associato ad una porta (lato *client* o lato *server*) viene associato dal sistema operativo a due code, una d'ingresso e una d'uscita
- Funzionalità di *buffering* dei dati



# Servizio di trasporto

- Il servizio di rete è non affidabile
  - Fa del proprio meglio per consegnare i singoli messaggi indipendentemente a destinazione
- Il servizio di trasporto fornito può essere di vari tipi
  - Trasporto affidabile (garanzia di consegna dei messaggi nel corretto ordine)
  - Trasporto non affidabile (solo funzionalità di multiplexing)
  - Trasporto orientato alla connessione
  - Trasporto senza connessione
- Sono definiti due tipi di trasporto
  - **TCP (Transmission Control Protocol)**, orientato alla connessione e affidabile
  - **UDP (User Datagram Protocol)**, senza connessione e non affidabile



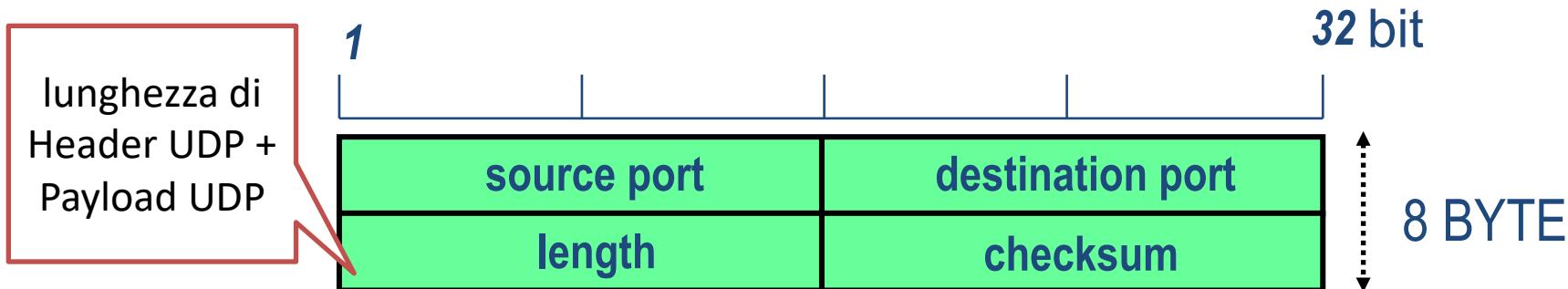
# Livello di Trasporto

- **Introduzione**
- **Protocollo UDP**
- **Trasporto affidabile**
  - Protocolli di ritrasmissione
  - Meccanismo a finestra mobile
- **Protocollo TCP**
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



# User Datagram Protocol (UDP) – RFC 768

- E' il modo più semplice di usare le funzionalità di IP
- Non aggiunge nulla a IP se non:
  - Indirizzamento delle applicazioni (mux/demux)
  - Blando controllo d'errore sull'header (senza correzione)
- ... e quindi
  - E' un protocollo datagram
  - Non garantisce la consegna
  - Non esercita nessun controllo (né di flusso, né di errore)



# Esempio di pacchetto UDP

**Numero porta sorgente (o destinatario): 2 Byte (16 bit)**

esempio: 4336 in sistema decimale

*0001 0000 1111 0000* in sistema binario

*1 0 f 0* in sistema esadecimale

esempio: 161 in sistema decimale

*0000 0000 1010 0001* in sistema binario

*0 0 a 1* in sistema esadecimale

<b>0<sub>hex</sub></b> = 0 <sub>dec</sub>	0	0	0	0
<b>1<sub>hex</sub></b> = 1 <sub>dec</sub>	0	0	0	1
<b>2<sub>hex</sub></b> = 2 <sub>dec</sub>	0	0	1	0
<b>3<sub>hex</sub></b> = 3 <sub>dec</sub>	0	0	1	1
<b>4<sub>hex</sub></b> = 4 <sub>dec</sub>	0	1	0	0
<b>5<sub>hex</sub></b> = 5 <sub>dec</sub>	0	1	0	1
<b>6<sub>hex</sub></b> = 6 <sub>dec</sub>	0	1	1	0
<b>7<sub>hex</sub></b> = 7 <sub>dec</sub>	0	1	1	1
<b>8<sub>hex</sub></b> = 8 <sub>dec</sub>	1	0	0	0
<b>9<sub>hex</sub></b> = 9 <sub>dec</sub>	1	0	0	1
<b>A<sub>hex</sub></b> = 10 <sub>dec</sub>	1	0	1	0
<b>B<sub>hex</sub></b> = 11 <sub>dec</sub>	1	0	1	1
<b>C<sub>hex</sub></b> = 12 <sub>dec</sub>	1	1	0	0
<b>D<sub>hex</sub></b> = 13 <sub>dec</sub>	1	1	0	1
<b>E<sub>hex</sub></b> = 14 <sub>dec</sub>	1	1	1	0
<b>F<sub>hex</sub></b> = 15 <sub>dec</sub>	1	1	1	1

**Numero massimo di porta:** si hanno al massimo 2 byte a disposizione

$$2^{16} - 1 = \underline{\underline{65535}} \quad [\text{in binario tutti e 16 bit a "1"}]$$

**Lunghezza massima possibile per il payload UDP:**

si hanno al massimo 2 byte a disposizione nel campo “length”

$$2^{16} - 1 = \underline{\underline{65535}} \text{ byte di lunghezza TOTALE del pacchetto UDP}$$

**Si devono togliere gli 8 byte dell'header UDP:**

$$\underline{\underline{65527}} \text{ byte disponibile per il payload UDP}$$

Se il payload è più lungo lo si trasmette in più segmenti.



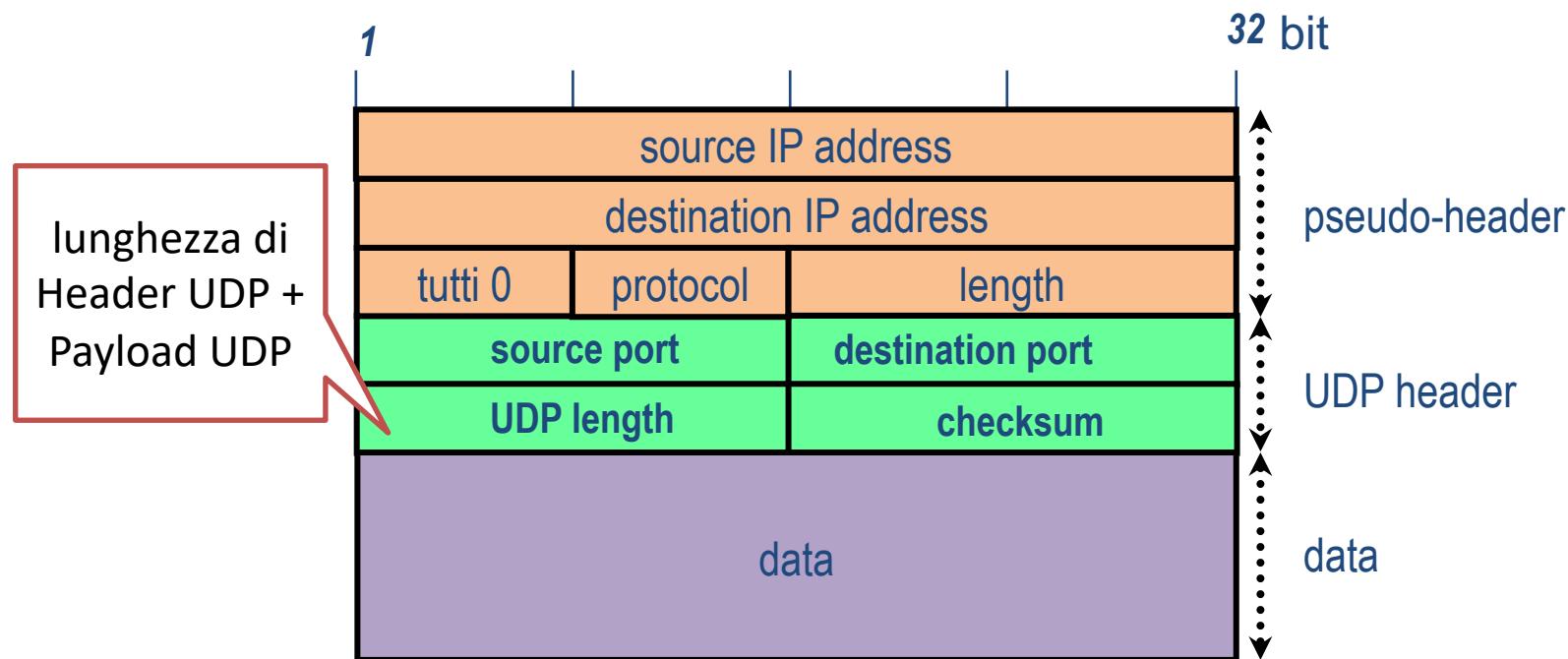
# Perchè usare UDP e non TCP?

- **Minore latenza**
  - Non occorre stabilire una connessione
- **Maggiore semplicità**
  - Non occorre tenere traccia dello stato della connessione
  - Poche regole da implementare
- **Minore overhead**
  - Header UDP è minore dell'header TCP



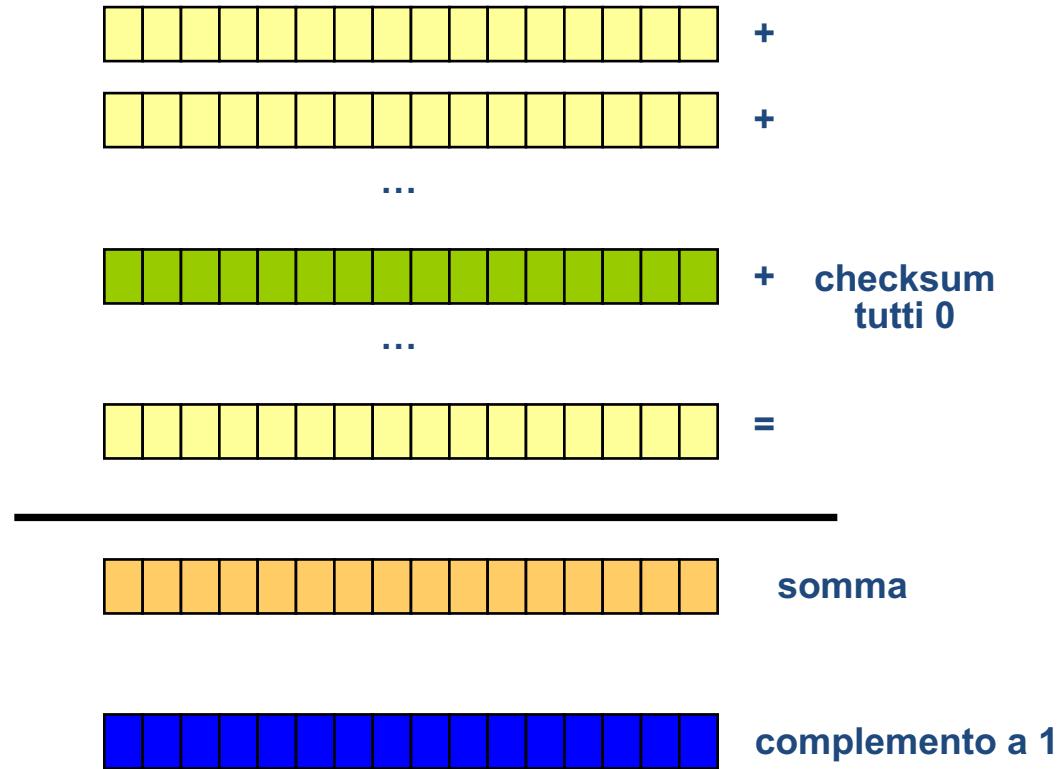
# Il campo *Checksum*: controllo di integrità

- Informazione ridondante inserita nell'header del segmento UDP per controllo d'errore
- Il campo di *checksum* (16 bit) è calcolato dal trasmettitore ed inserito nell'header
- Il ricevitore ripete lo stesso calcolo sul segmento ricevuto (comprensivo di *checksum*)
- Se il risultato è soddisfacente accetta il segmento, altrimenti lo scarta
- Viene calcolato considerando l'**header** UDP, uno **pseudo-header** IP ed i **dati**



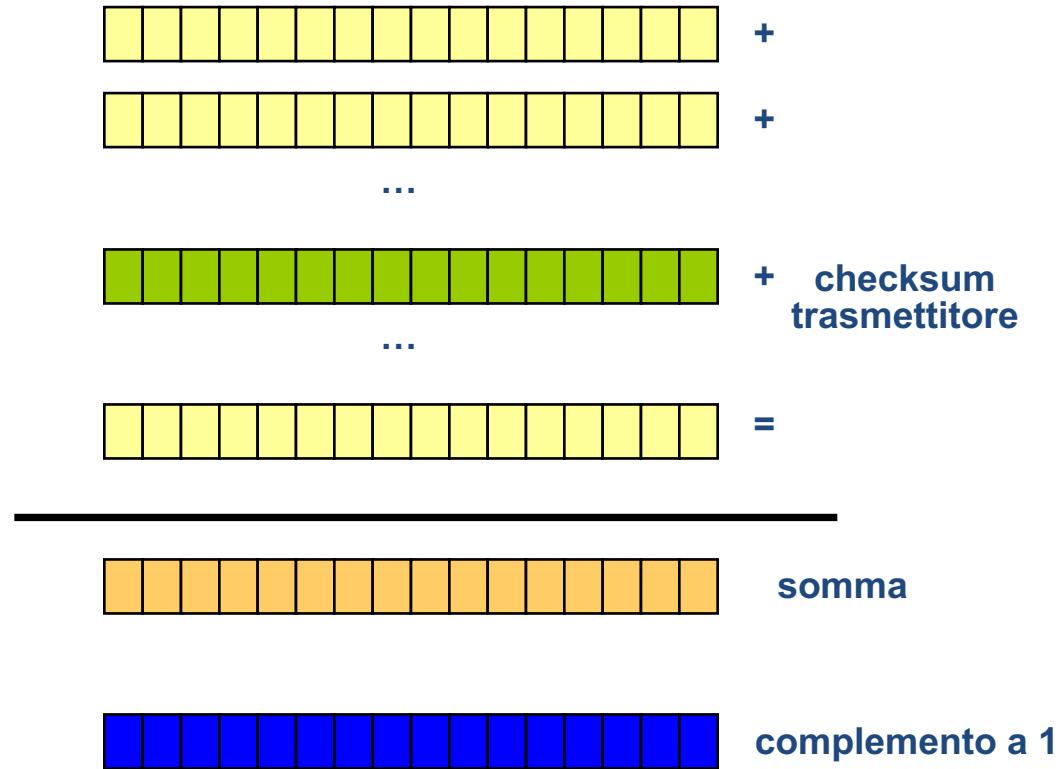
# Calcolo del Checksum *lato trasmettitore*

- L'insieme di bit è diviso in blocchi da 16 bit
- Il campo *Checksum* è inizializzato a 0
- Tutti i blocchi vengono sommati modulo 2
- Il risultato è complementato ed inserito nel campo di *checksum* del segmento inviato



# Calcolo del Checksum *lato ricevitore*

- L'insieme di bit è diviso ancora in blocchi da 16 bit
- Tutti i blocchi vengono sommati modulo 2
- Il risultato è complementato
  - Se sono tutti 0 il pacchetto è accettato
  - Altrimenti è scartato



# Livello di Trasporto

- Introduzione
- Protocollo UDP
- **Trasporto affidabile**
  - Protocolli di ritrasmissione
  - Controllo di flusso a finestra mobile
- **Protocollo TCP**
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



# Collegamento ideale

- Collegamento ideale
  - Tutto ciò che viene trasmesso arriva nello stesso ordine e viene correttamente interpretato a destinazione
- Es., come essere sicuri che una sequenza di ordini impartiti sia stata compresa con certezza?
- E' possibile oppure è uno sforzo velleitario?
- E' necessario esserne certi?



# Recupero d'errore

- Ingredienti di un **Protocollo di Ritrasmissione**
  - Ciascuna trama ricevuta correttamente viene riscontrata positivamente con un messaggio di (Acknowledgment o ACK)
  - A volte l'errore può essere segnalato da un messaggio detto di NACK (Not ACK)
  - La mancanza di ACK o la presenza di NACK segnala la necessità di ritrasmettere
  - La procedura si ripete finché la trama viene ricevuta corretta
- Necessita di canale di ritorno e di messaggi di servizio (ACK, NACK)
- Nota: anche i messaggi di servizio possono essere corrotti da errori!



# Controllo d'integrità e recupero d'errore

- Queste procedure possono essere attivate a qualunque livello
- Storicamente sono state sempre presenti a livello di linea sui collegamenti fisici a causa delle cattive linee fisiche del passato
- Presente a livello di trasporto per recupero *end-to-end*
  - Proteggere i collegamenti fisici non basta, i pacchetti possono andare persi nei buffer dei router
- Nei sistemi moderni il recupero d'errore a livello di linea può essere assente



# Protocolli di ritrasmissione

- Obiettivo:
  - integrità del messaggio/pacchetto
  - ordine della sequenza di pacchetti
  - no duplicazione
- Usando i messaggi:
  - ACK: riscontro positivo
  - NACK: riscontro negativo
- Tre tipologie:
  - *Stop & Wait*
  - *Go-Back-N*
  - *Selective Repeat*



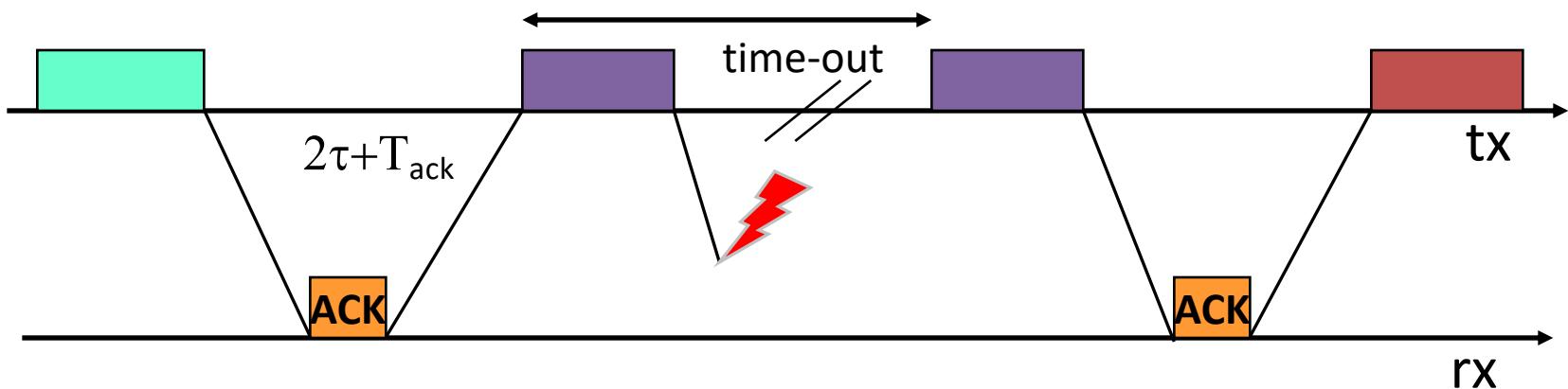
# Protocollo Stop and Wait

- Utilizza il solo ACK
- e un contatore di *time out*
- Ogni messaggio ricevuto correttamente è riscontrato dal ricevitore (ACK)



# Protocollo Stop and Wait

- Il trasmettitore trasmette un pacchetto e inizializza un contatore (*time-out*)
- Se il primo evento successivo è
  - l'ACK, trasmette il pacchetto successivo
  - il time-out, ritrasmette il pacchetto corrente

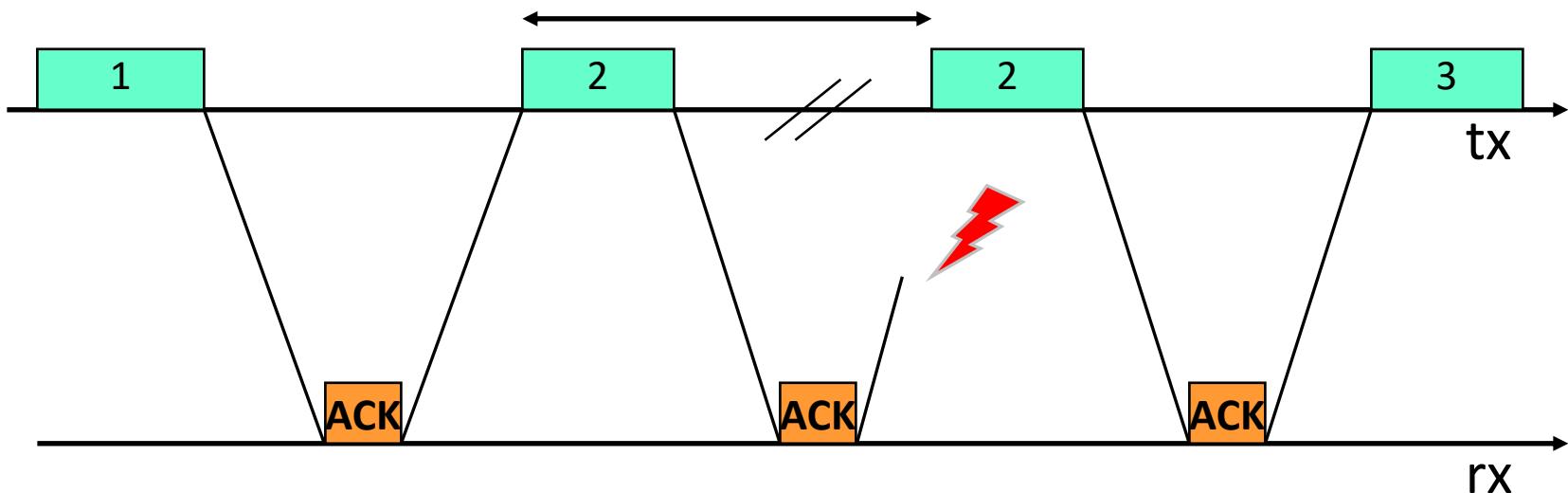


Funzionamento corretto se **time-out  $\geq 2\tau + T_{ack}$**



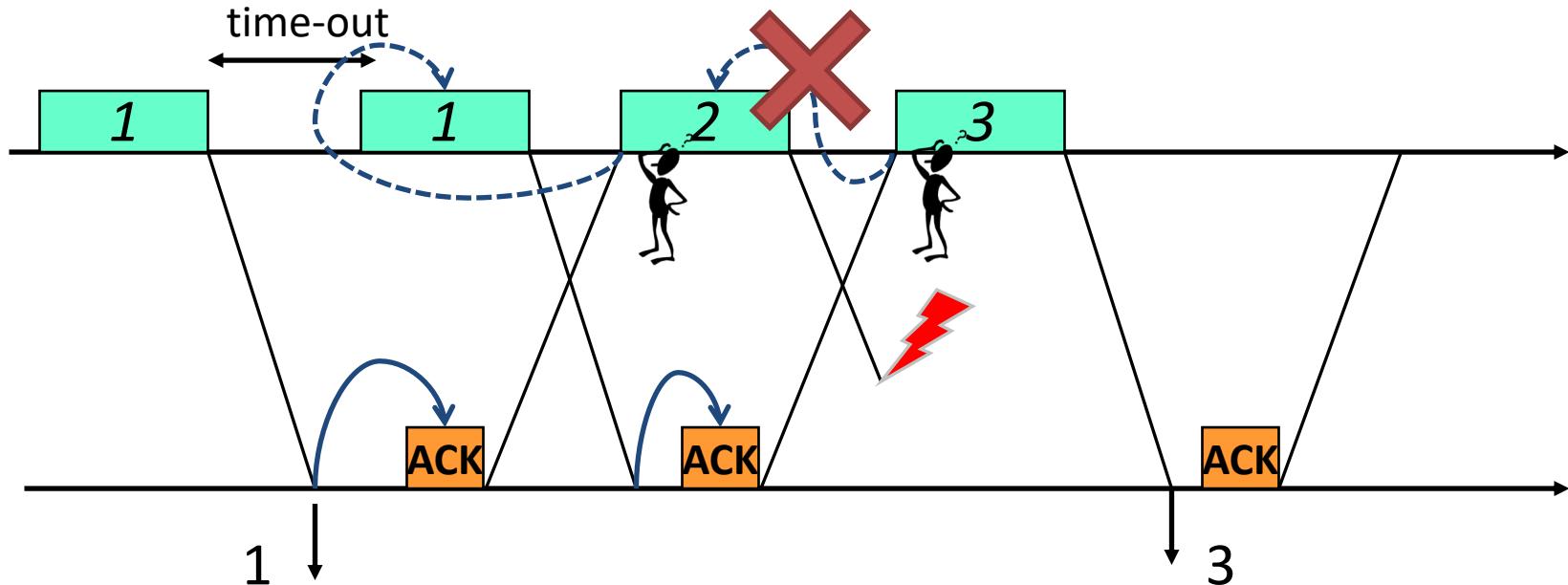
# Protocollo Stop and Wait

- Problema: se si perde l'ACK il pacchetto viene ritrasmesso (e ricevuto) di nuovo (duplicazione di pacchetti)
- Rimedio: **numerazione dei pacchetti (SN)**
- Il ricevitore riconosce che il nuovo pacchetto è un duplicato perché ha lo stesso numero dell'ultimo ricevuto



# Protocollo Stop and Wait

- Problema: errata interpretazione di un ACK. Il ricevitore crede un pacchetto ricevuto (il 2) mentre non lo è
- Rimedio: **numerazione anche degli ACK (RN)**



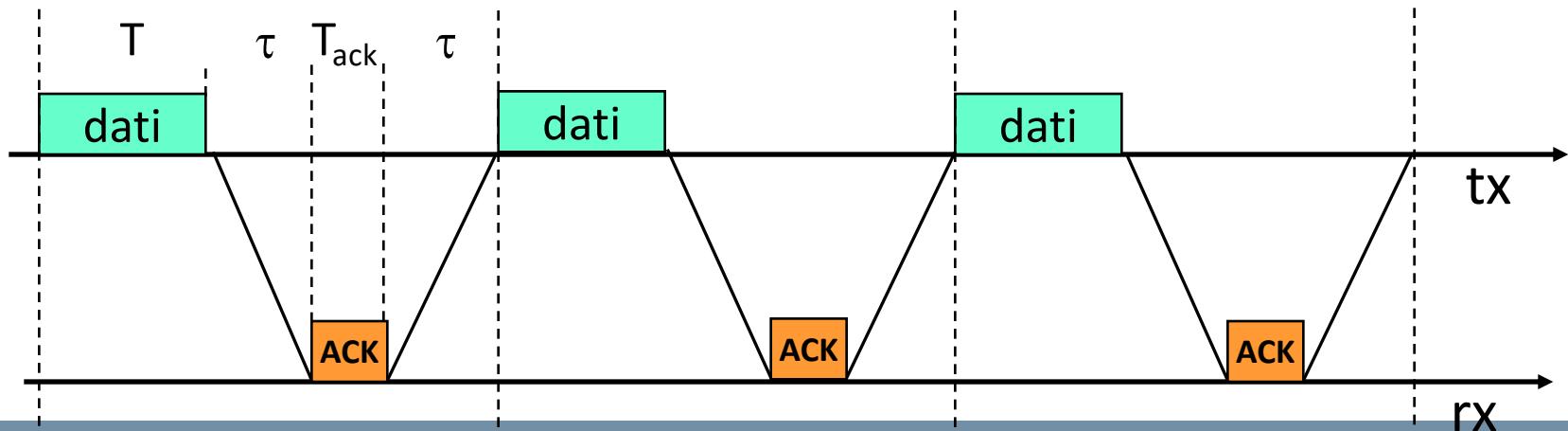
# Protocollo Stop and Wait

- Efficienza del protocollo:
  - frazione di tempo in cui il canale è usato per trasmettere informazione utile in assenza di errori

$$\eta = \frac{T}{T + T_{ack} + 2\tau}$$

$T$  = tempo di trasmissione

$\tau$  = tempo di propagazione



# Protocollo Stop and Wait

- Efficienza bassa se  $T \ll \tau$
- Protocollo non adatto a situazioni con elevato ritardo di propagazione e/o elevato ritmo di trasmissione
- Utilizzato spesso in modalità *half-duplex*

$$\eta = \frac{T}{T + T_{ack} + 2\tau} = \frac{1}{1 + \frac{T_{ack}}{T} + \frac{2\tau}{T}}$$



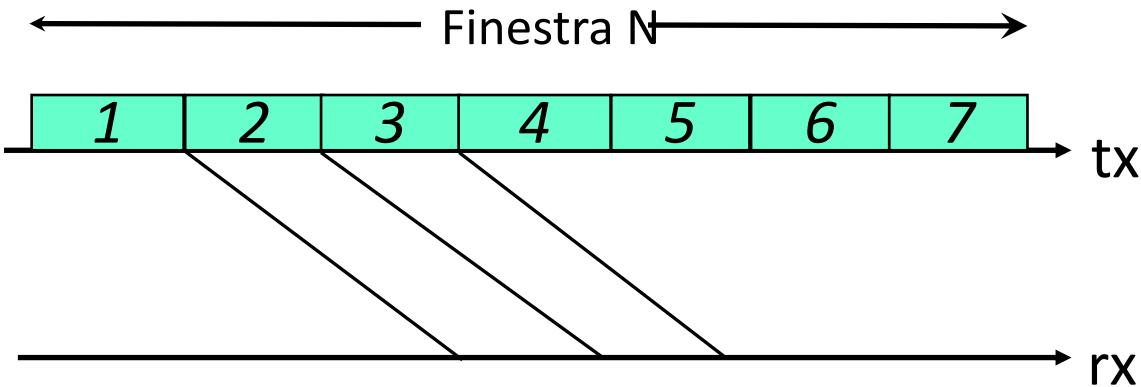


# **7 – Il Livello di Trasporto**

## **Parte II**

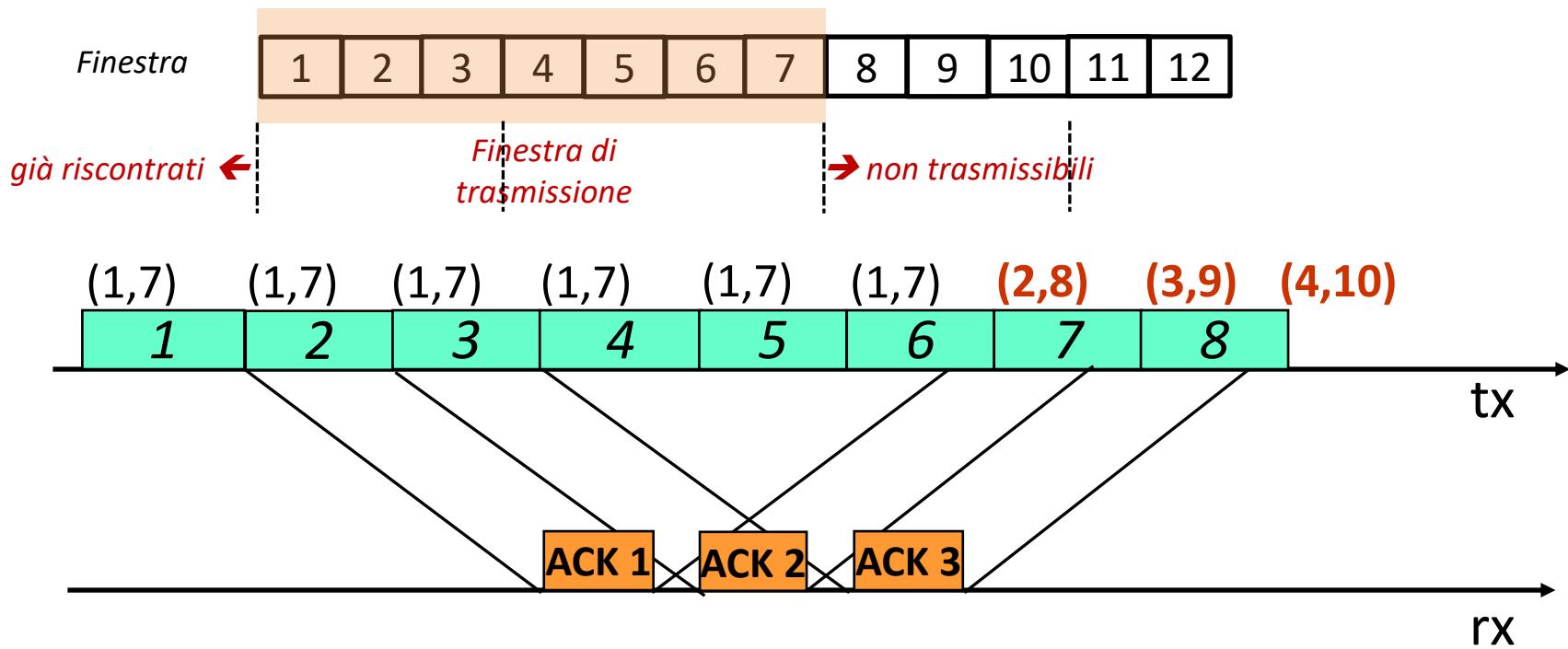
# Protocollo Go-back-N

- Variante rispetto allo *Stop and Wait*:
  - Si possono trasmettere fino a  $N$  pacchetti (finestra) senza aver avuto il riscontro



# Finestra Go-back-N

- Se il riscontro del primo pacchetto arriva prima della fine della finestra, la finestra viene fatta scorrere di una posizione (**sliding window**)

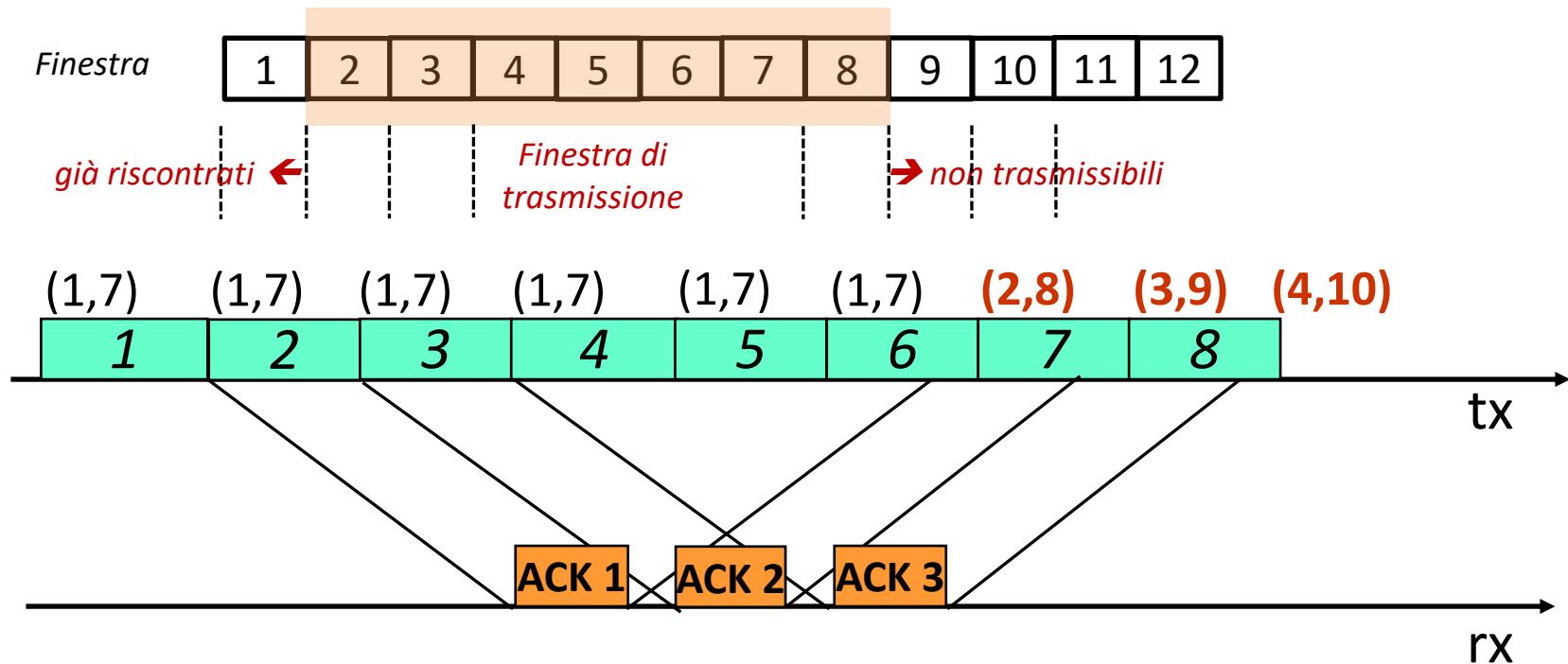


Se non ci sono errori la trasmissione non si ferma mai (efficienza 100%)



# Finestra Go-back-N

- Se il riscontro del primo pacchetto arriva prima della fine della finestra, la finestra viene fatta scorrere di una posizione (**sliding window**)

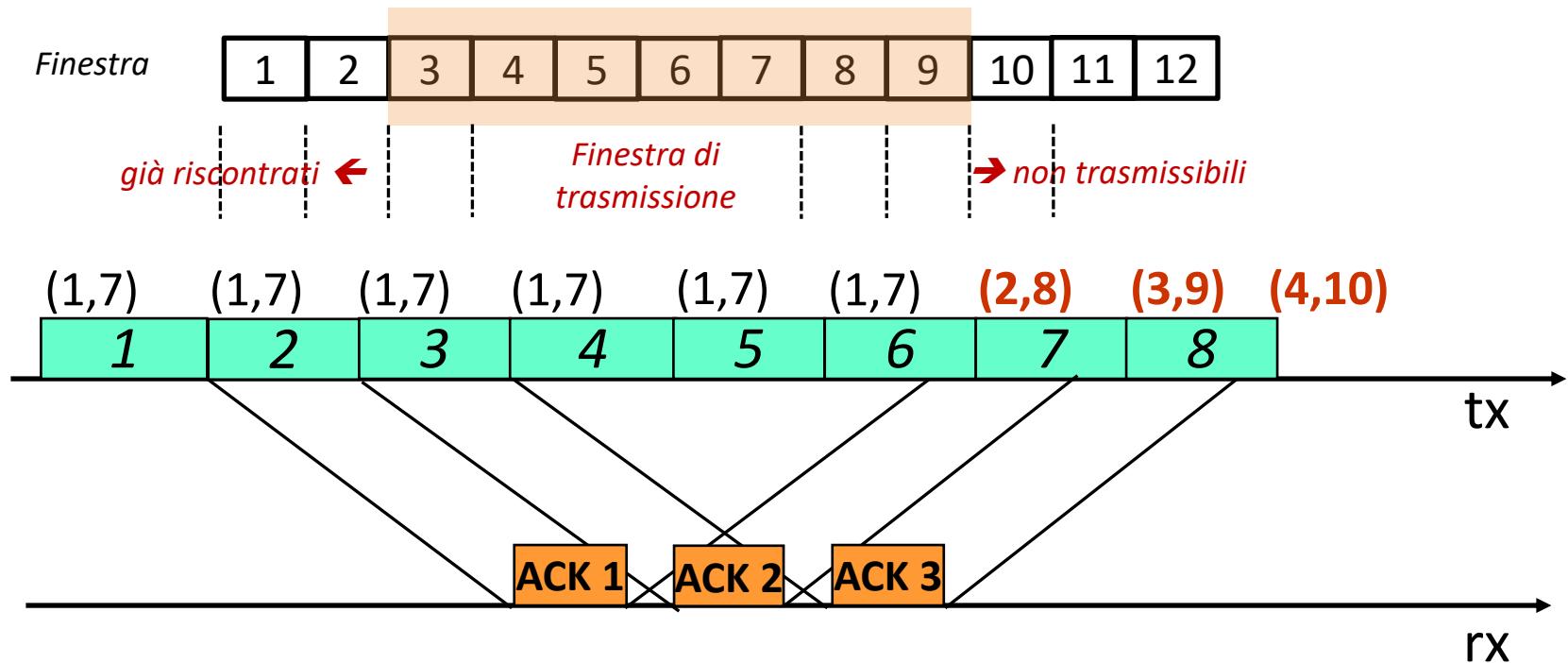


Se non ci sono errori la trasmissione non si ferma mai (efficienza 100%)



# Finestra Go-back-N

- Se il riscontro del primo pacchetto arriva prima della fine della finestra, la finestra viene fatta scorrere di una posizione (**sliding window**)

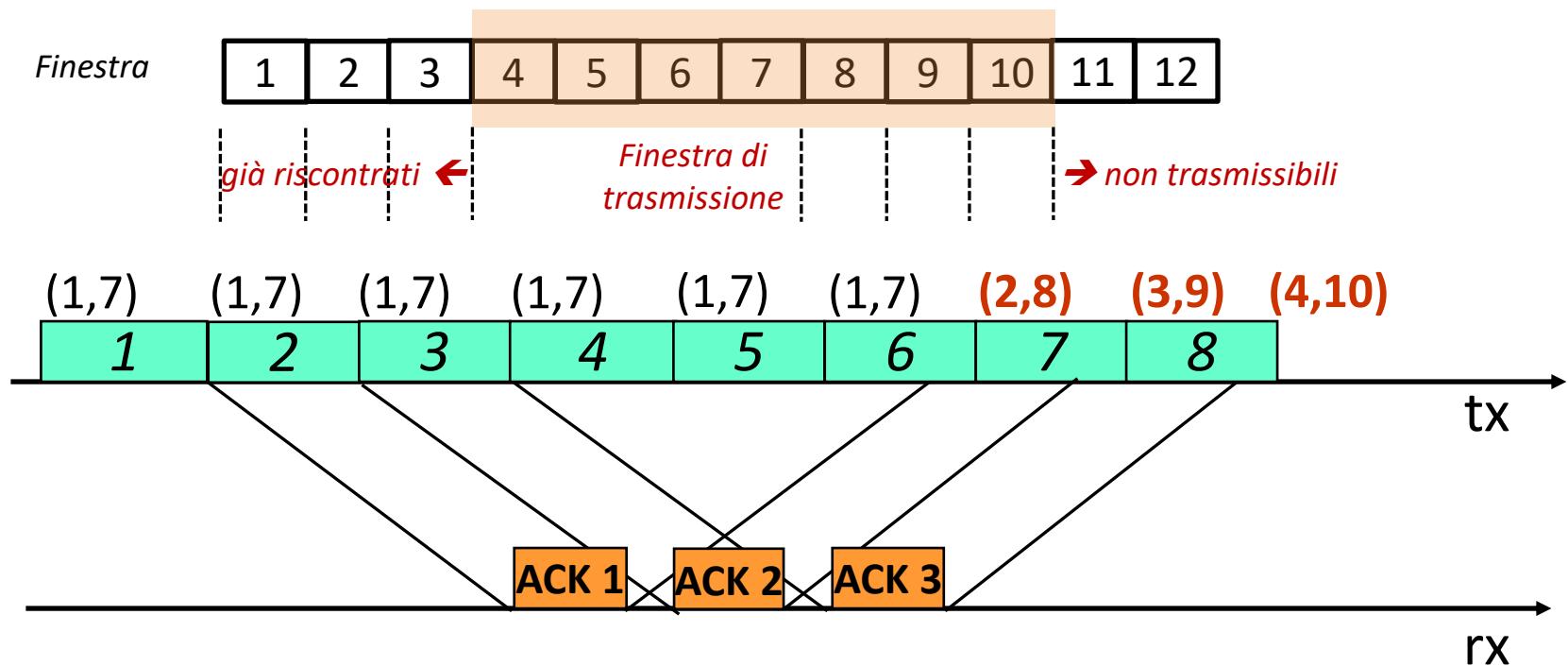


Se non ci sono errori la trasmissione non si ferma mai (efficienza 100%)



# Finestra Go-back-N

- Se il riscontro del primo pacchetto arriva prima della fine della finestra, la finestra viene fatta scorrere di una posizione (**sliding window**)

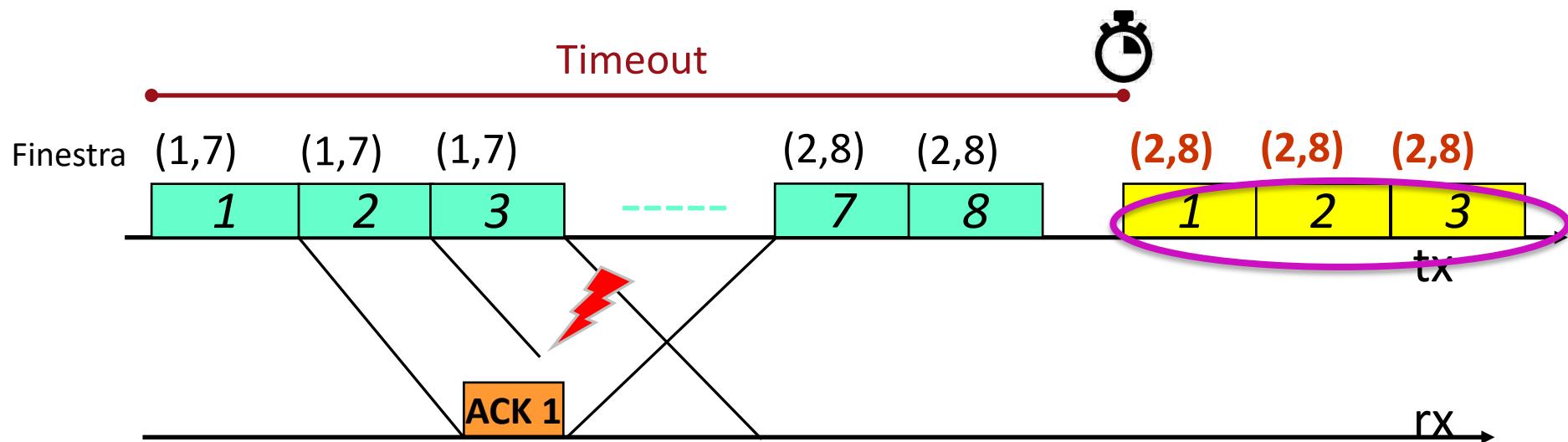


Se non ci sono errori la trasmissione non si ferma mai (efficienza 100%)



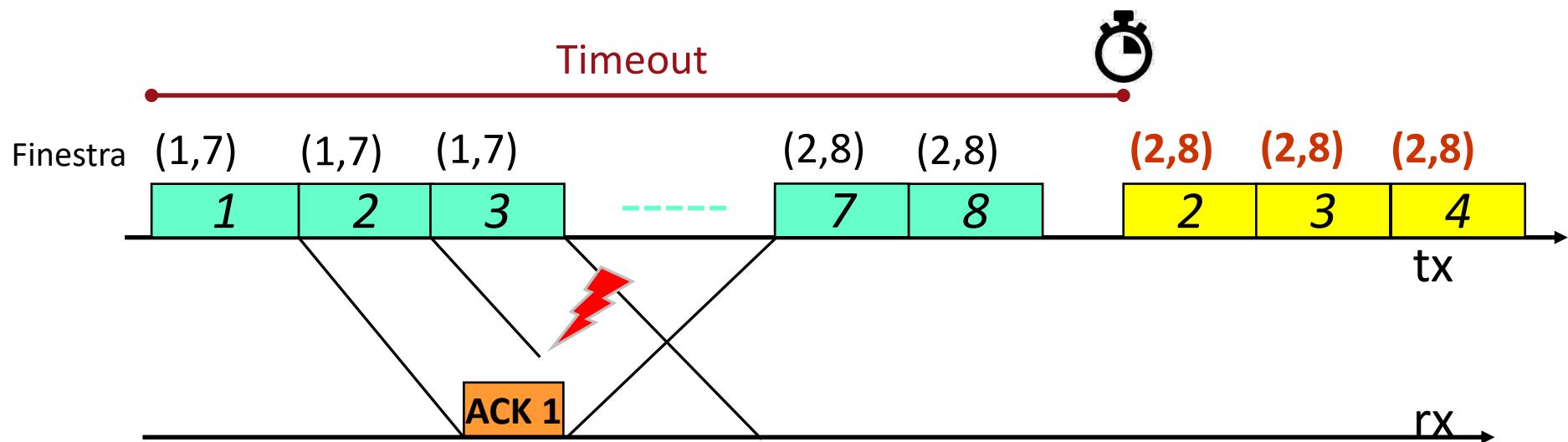
# Finestra Go-back-N

- ... altrimenti (quando si verifica un errore): **si ricomincia a trasmettere la finestra dal primo pacchetto non riscontrato**
  - “Torna indietro di N pacchetti”
- il *time out* ha lo stesso significato dello Stop&Wait
  - Raggiunto l’ultimo pacchetto della finestra, la trasmissione si blocca in attesa di un nuovo ACK o della scadenza del timeout
  - La ritrasmissione del primo pacchetto non riscontrato inizia allo scadere del timeout



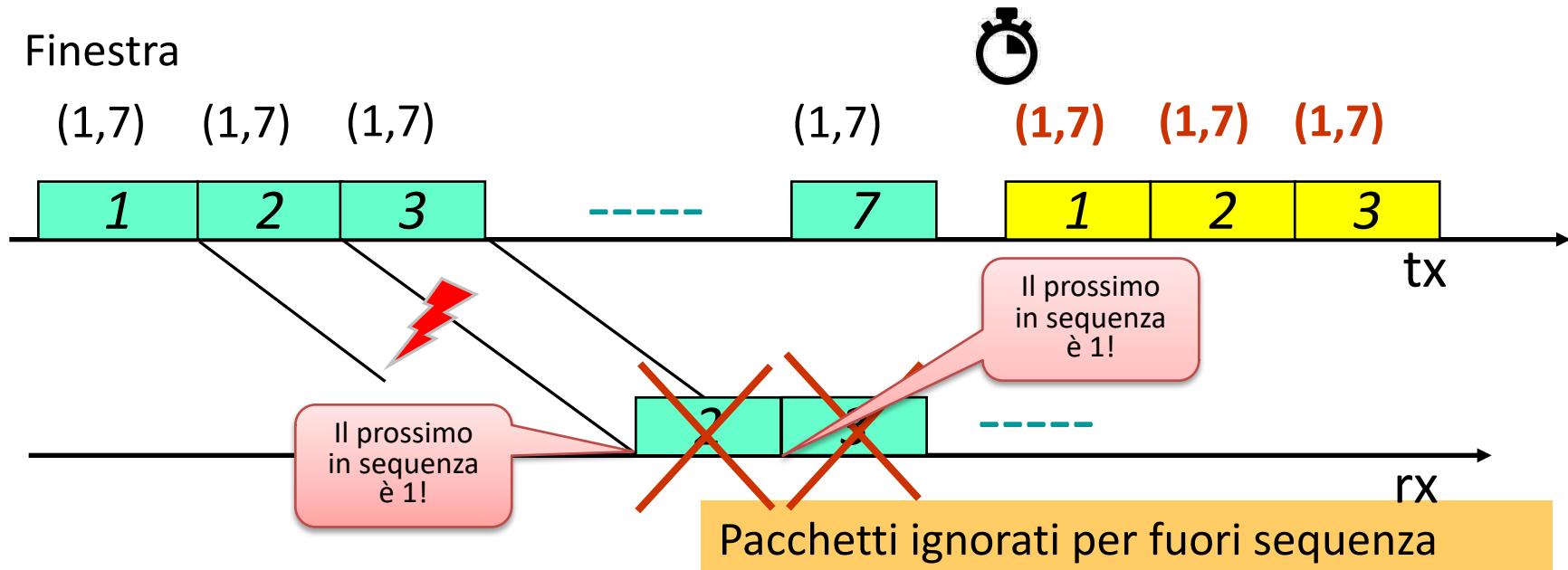
# Finestra Go-back-N

- ... altrimenti (quando si verifica un errore): **si ricomincia a trasmettere la finestra dal primo pacchetto non riscontrato**
  - “Torna indietro di N pacchetti”
- il *time out* ha lo stesso significato dello Stop&Wait
  - Raggiunto l’ultimo pacchetto della finestra, la trasmissione si blocca in attesa di un nuovo ACK o della scadenza del timeout
  - La ritrasmissione del primo pacchetto non riscontrato inizia allo scadere del timeout



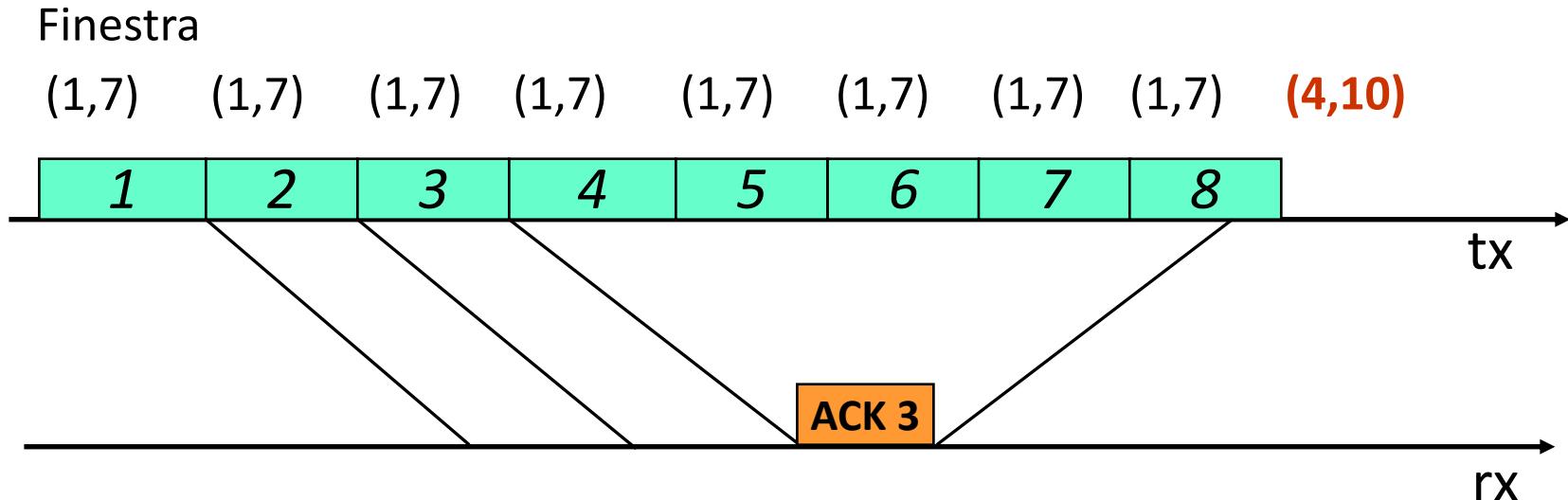
# Finestra Go-back-N

- Ciò può causare la ritrasmissione di pacchetti corretti, ma semplifica il funzionamento perché
- .. permette al ricevitore di ignorare le ricezioni fuori sequenza (l'ordine è mantenuto automaticamente)



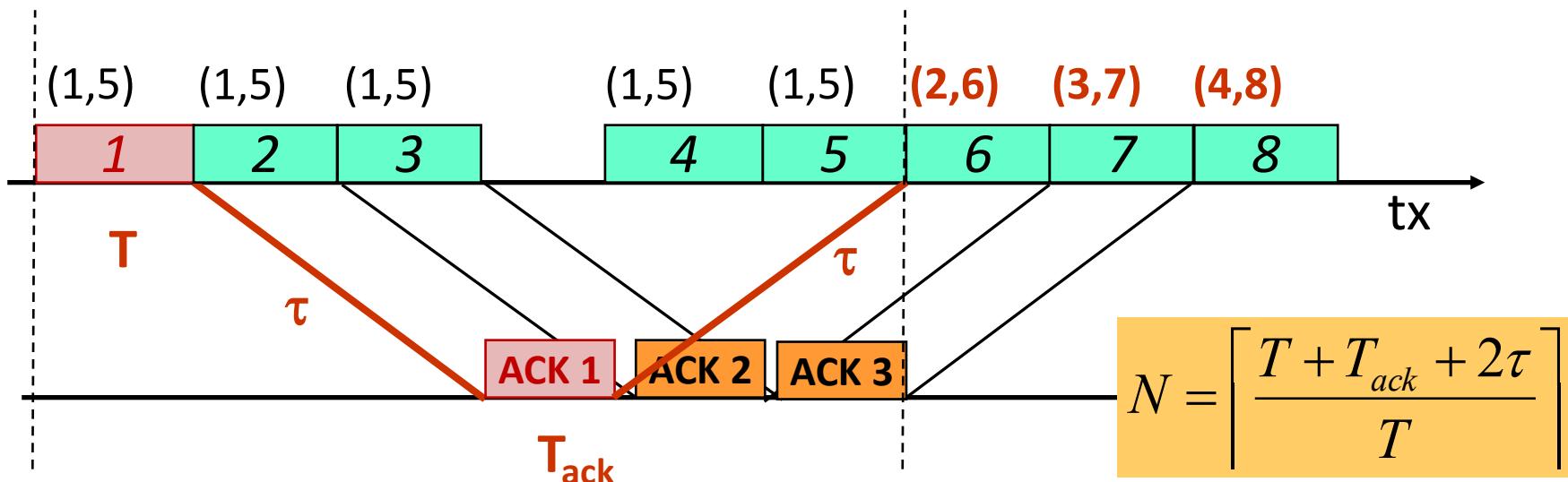
# Go-back-N: ACK collettivo

- Escludendo il fuori sequenza, il riscontro (ACK) può essere collettivo
- Questo, entro certi limiti, rimedia alla perdita di ACK



# Dimensionamento finestra

- La finestra ottima coincide con il **Round Trip Time**
  - Trasmissione pacchetto, propagazione tx-rx,  
Trasmissione ACK e propagazione rx-tx
  - La finestra aumenta di un pacchetto alla volta e la trasmisione non si interrompe mai



# Dimensionamento finestra

- La finestra può anche essere dimensionata in tempo, in byte, .....
- Il dimensionamento si complica se i tempi di propagazione non sono noti
  - es. con Go-back-N a livello di trasporto i tempi di attraversamento della rete ( $\tau$ ) possono variare col tempo..
  - e/o i pacchetti sono di lunghezza variabile
- Rimedi
  - Fare la finestra grande
    - Non pregiudica il funzionamento in assenza d'errore
    - Ma in caso d'errore, aumenta il ritardo con cui si scopre ed il numero di ritrasmissioni inutili
    - Uso del NACK
  - Stimare il tempo di RTT e adattare la finestra o il timeout

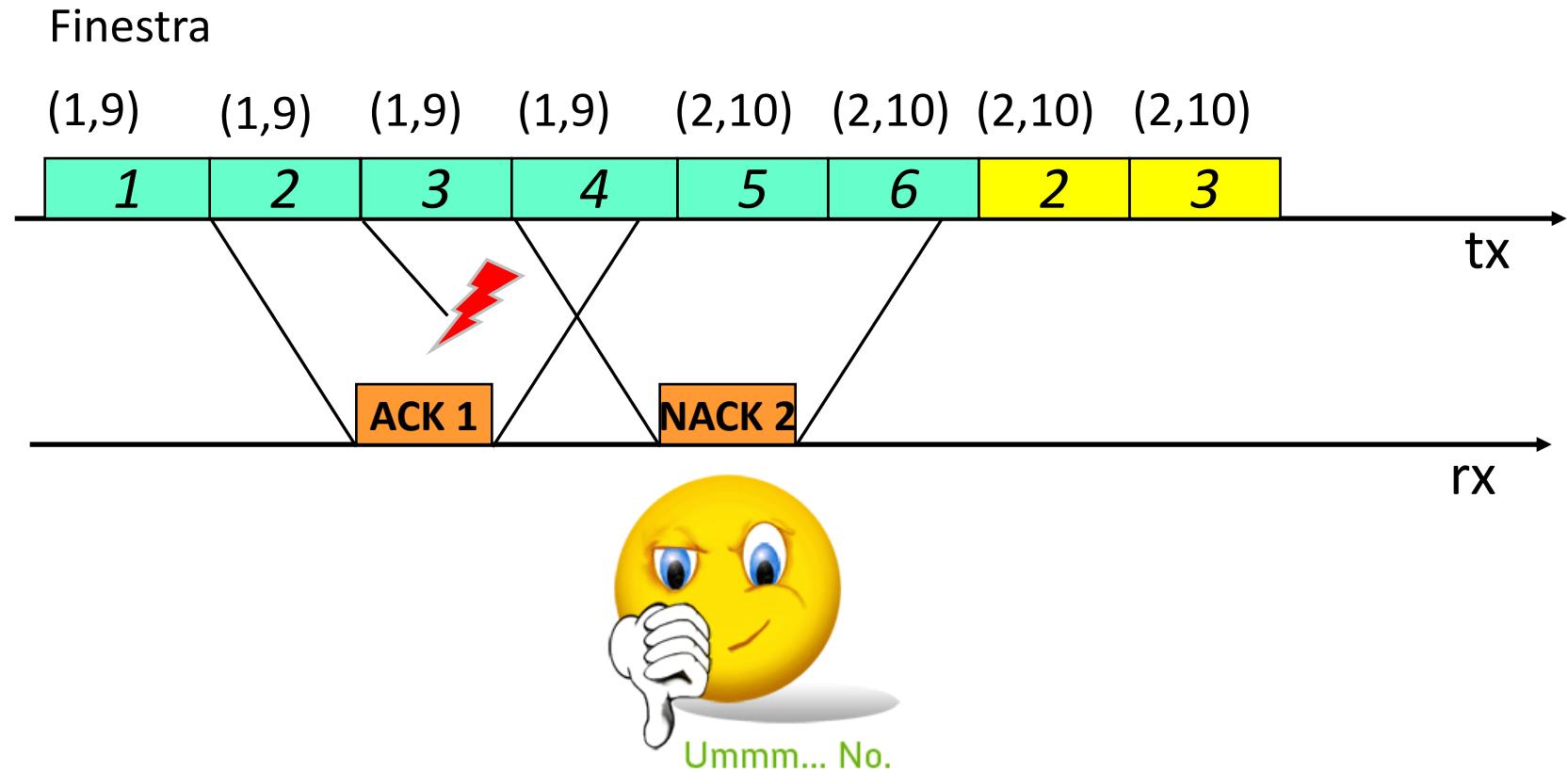


# Uso del NACK

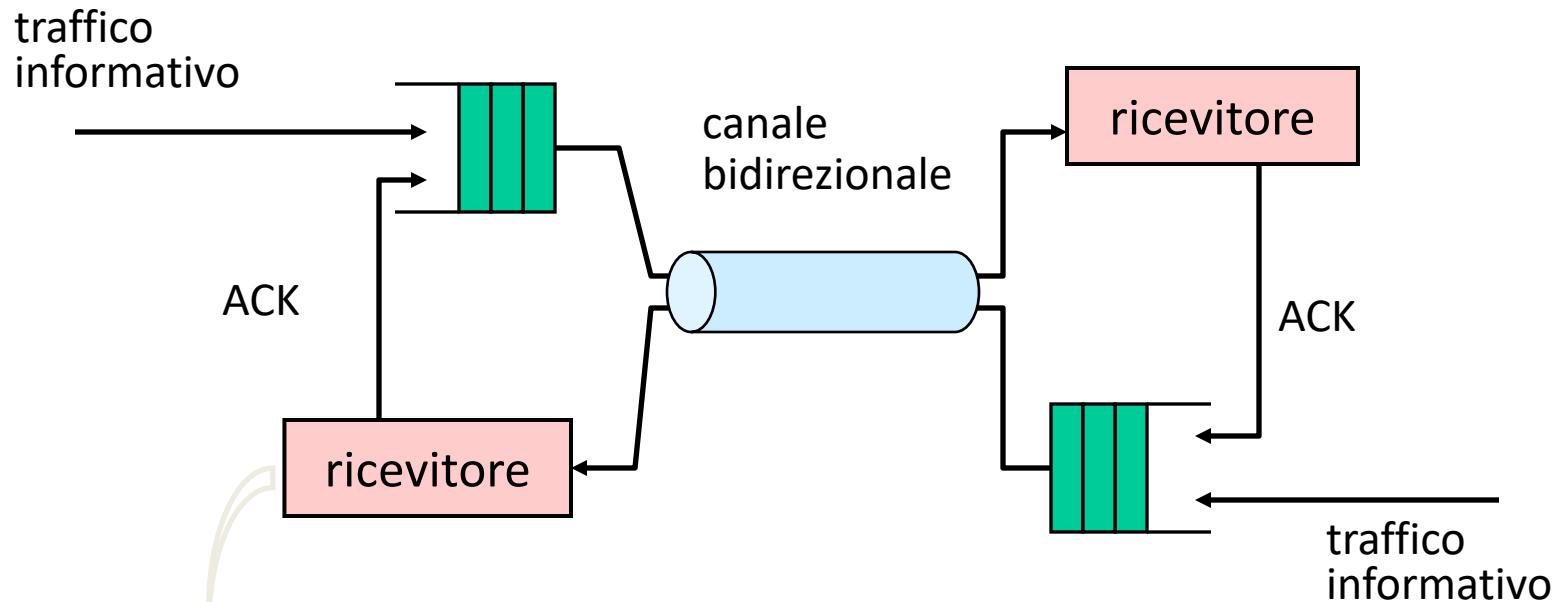
- L'uso del NACK può abbreviare i tempi di ritrasmissione in caso d'errore, evitando di aspettare la fine della finestra
- Non è possibile inviare immediatamente il NACK perché il ricevitore dovrebbe conoscere il SN del pacchetto errato, ma se e' errata ...
- Se arriva un pacchetto fuori sequenza, posso ipotizzare che sia andata perso il pacchetto precedente
- Non è possibile applicarlo nei meccanismi in cui non è garantita la consegna in ordine (livelli superiori al livello di link)



# Uso del NACK



# Go-back-N full-duplex



- Analisi pacchetti ricevuti
- Inoltro a livello superiore dei pacchetti ricevuti
- Generazione degli ACK



# **Go-back-N e piggy backing**

- Gli ACK possono anche essere inseriti negli *header* dei pacchetti che viaggiano in direzione opposta (*Piggy-backing*)



- *SN*: numero di sequenza del pacchetto trasmesso (canale diretto)
- *RN*: numero di sequenza delle pacchette atteso in direzione opposta, vale come riscontro cumulativo dei pacchetti fino a *RN-1*



# Regole Go-back-N

- Trasmettitore:
  - $N$ : dimensione finestra
  - $N_{last}$ : ultimo riscontro ricevuto
  - $N_C$ : numero corrente disponibile per pacchetto in trasmissione
  - **Regole:**
    - Ogni nuovo pacchetto viene accettato per la trasmissione solo se  $N_C < N_{last} + N$ , altrimenti viene messo in attesa; se  $N_C < N_{last} + N$  il pacchetto viene trasmesso con SN pari a  $N_C$ , viene inizializzato il timer di timeout e il valore di  $N_C$  viene incrementato di uno ( $N_C := N_C + 1$ );
    - Ad ogni riscontro RN ricevuto, si pone  $N_{last} = RN$ ;
    - I pacchetti nella finestra vengono trasmessi senza vincoli di temporizzazione;
    - In caso di scadenza di *timeout* la ritrasmissione deve ripartire dal pacchetto  $N_{last}$
  - Intervallo  $N_{last}$  e  $(N_{last} + N - 1)$ : finestra di trasmissione

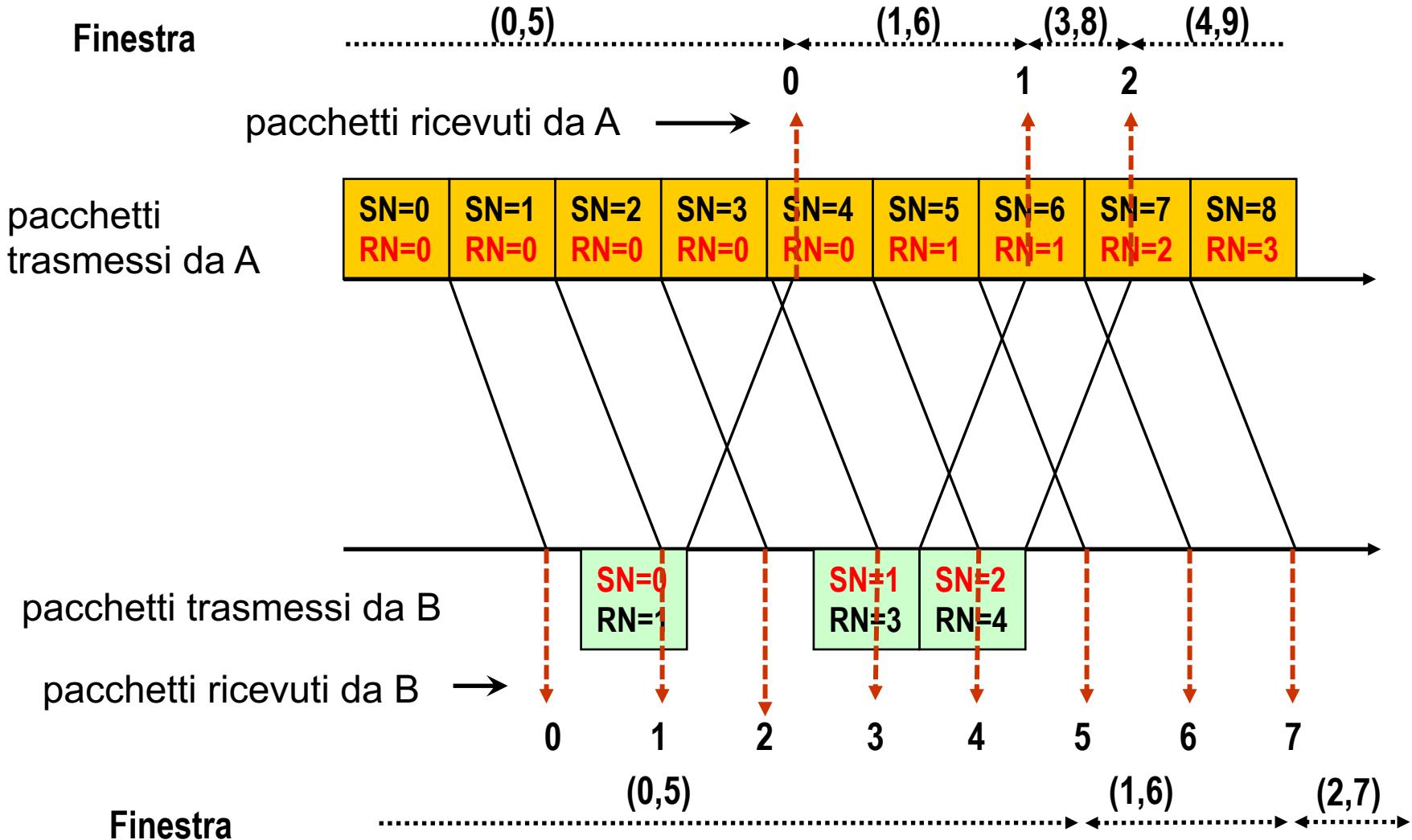


# Regole Go-back-N

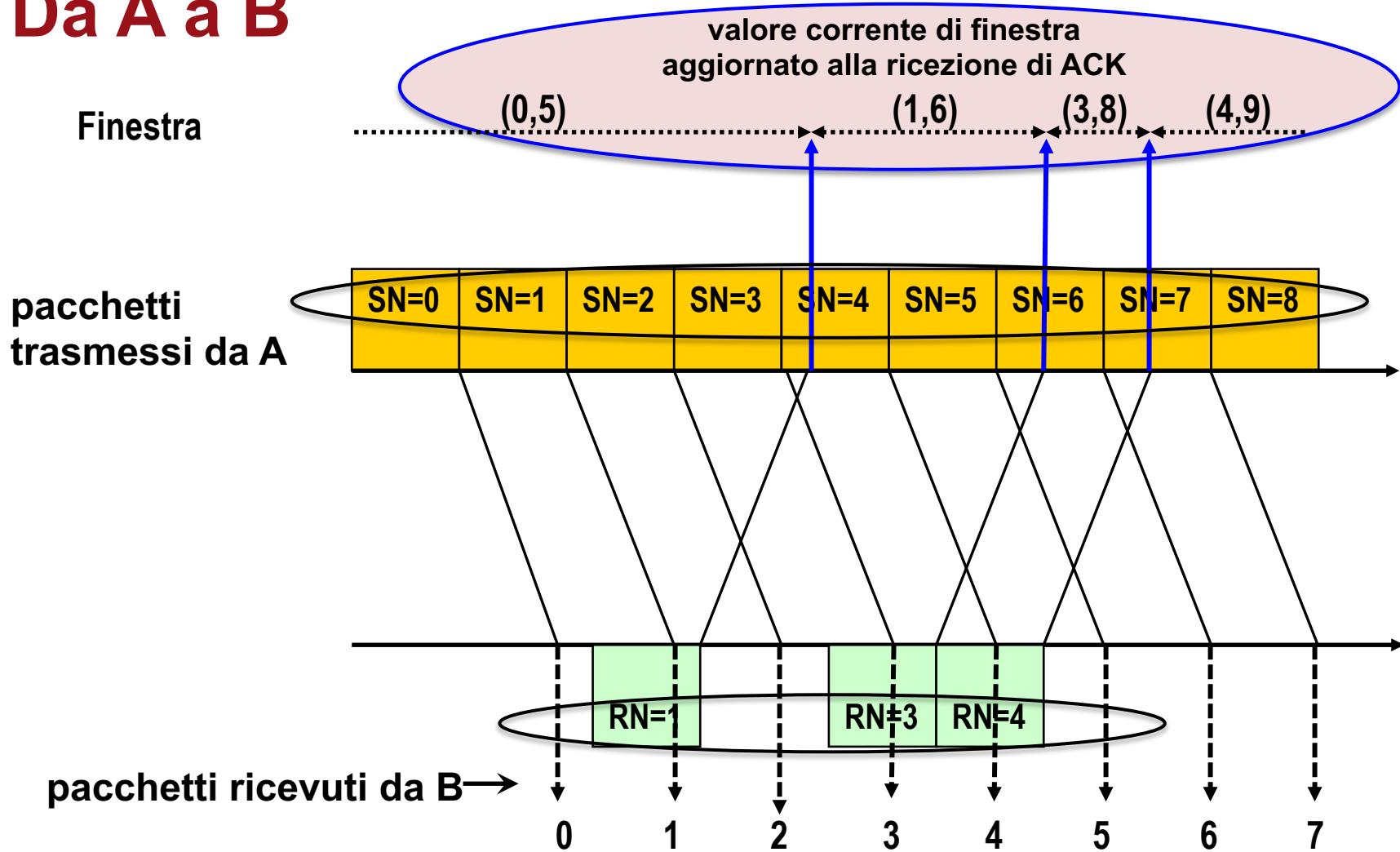
- Ricevitore:
  - *RN: stato dei riscontri corrente (SN che il ricevitore si aspetta di ricevere)*
  - **Regole:**
    - Se viene ricevuto correttamente un pacchetto con  $SN=RN$ , questo viene inoltrato ai livelli superiori e si pone  $RN:=RN+1$ ;
    - Ad istanti arbitrari ma con ritardo finito,  $RN$  viene trasmesso al mittente utilizzando i pacchetti in direzione opposta.



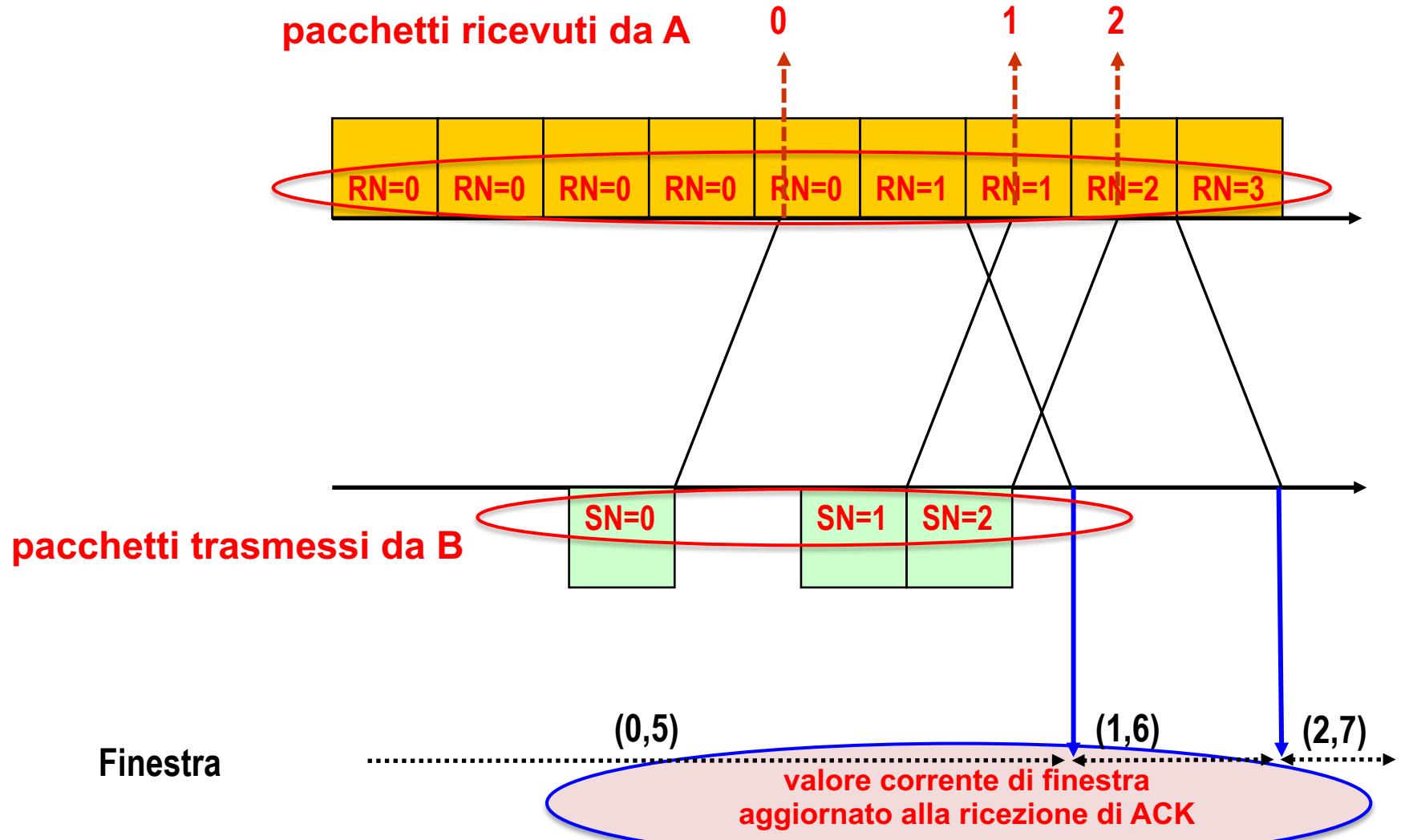
# Go-back-N full duplex senza errori



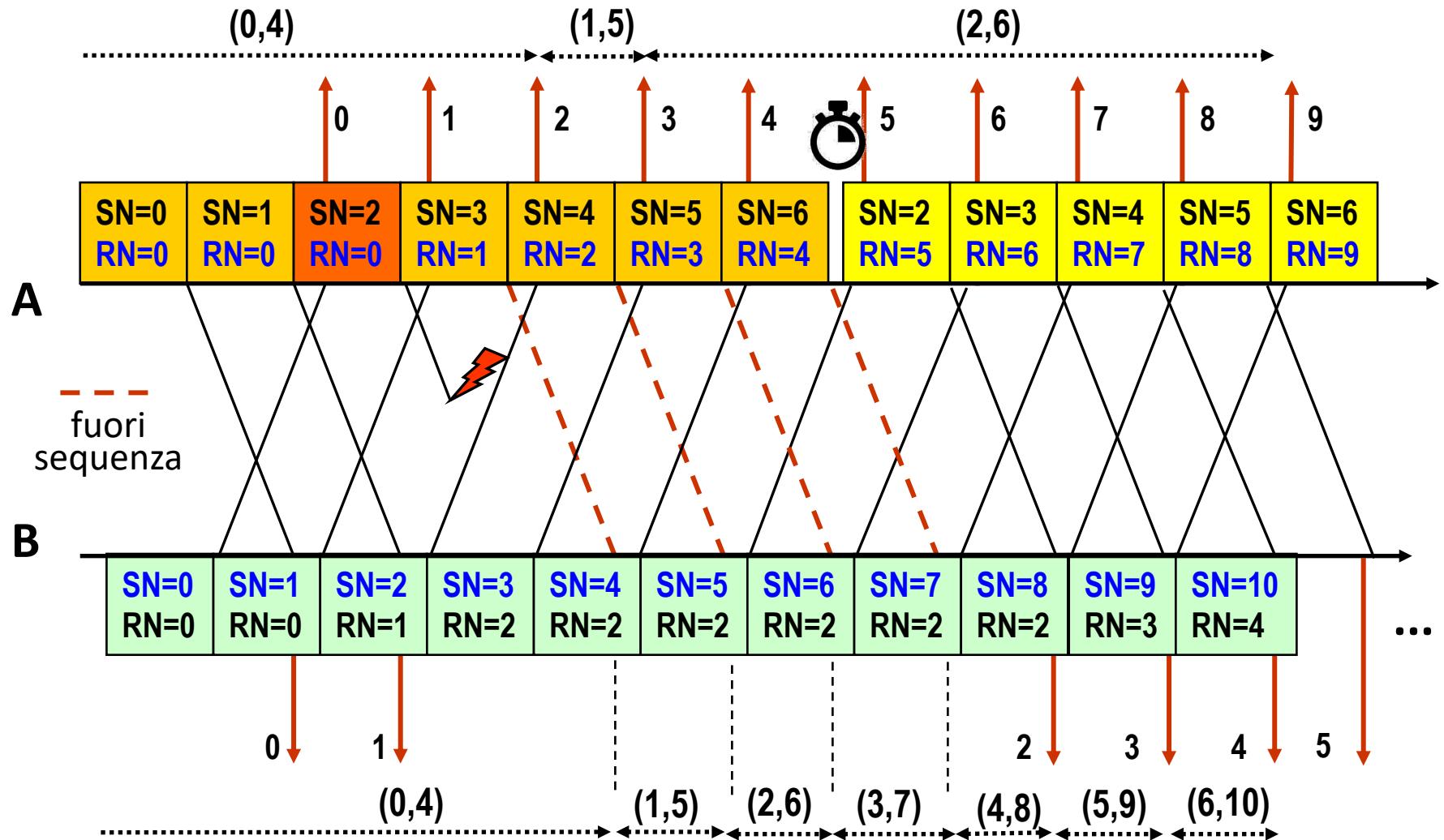
# Da A a B



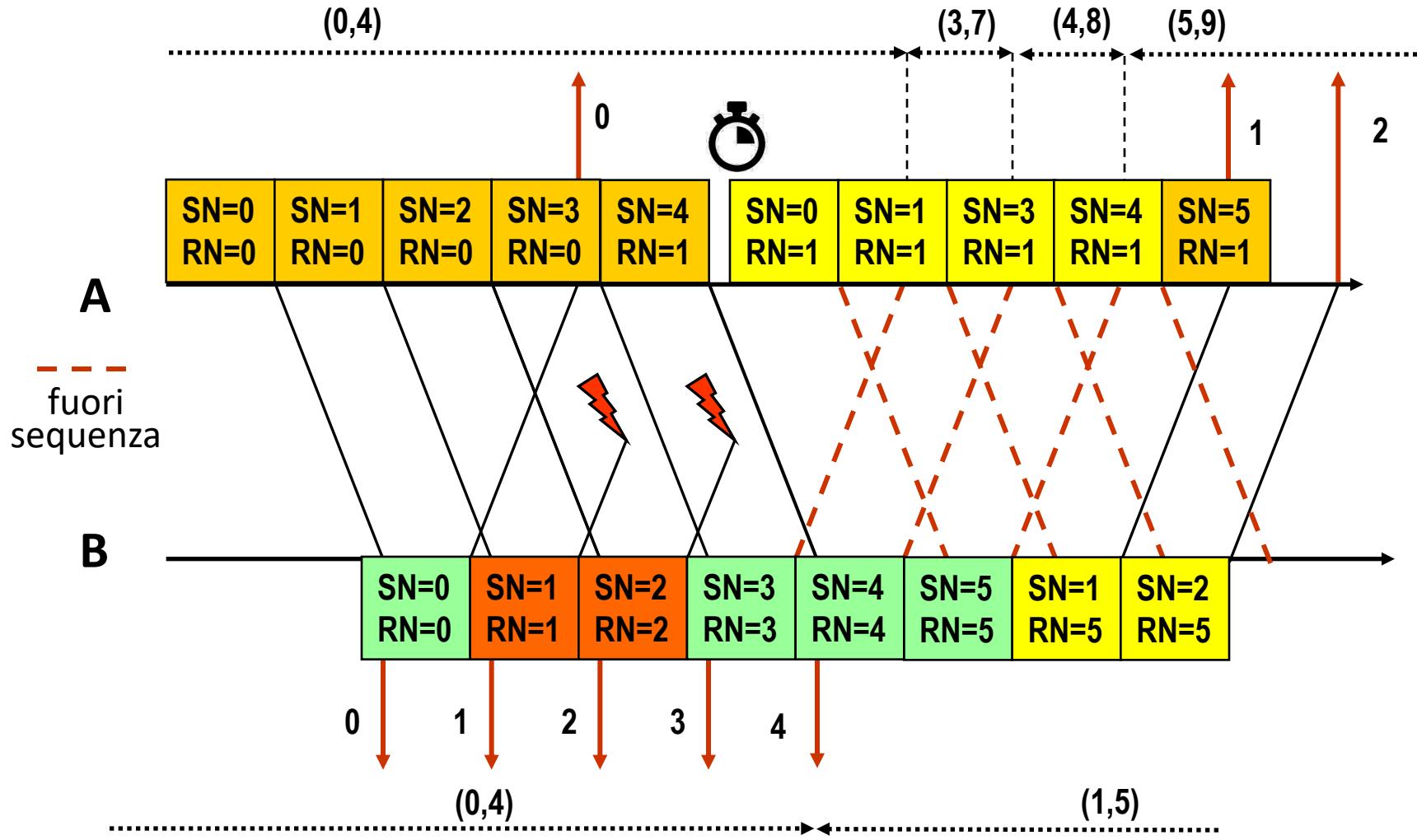
# Da B a A



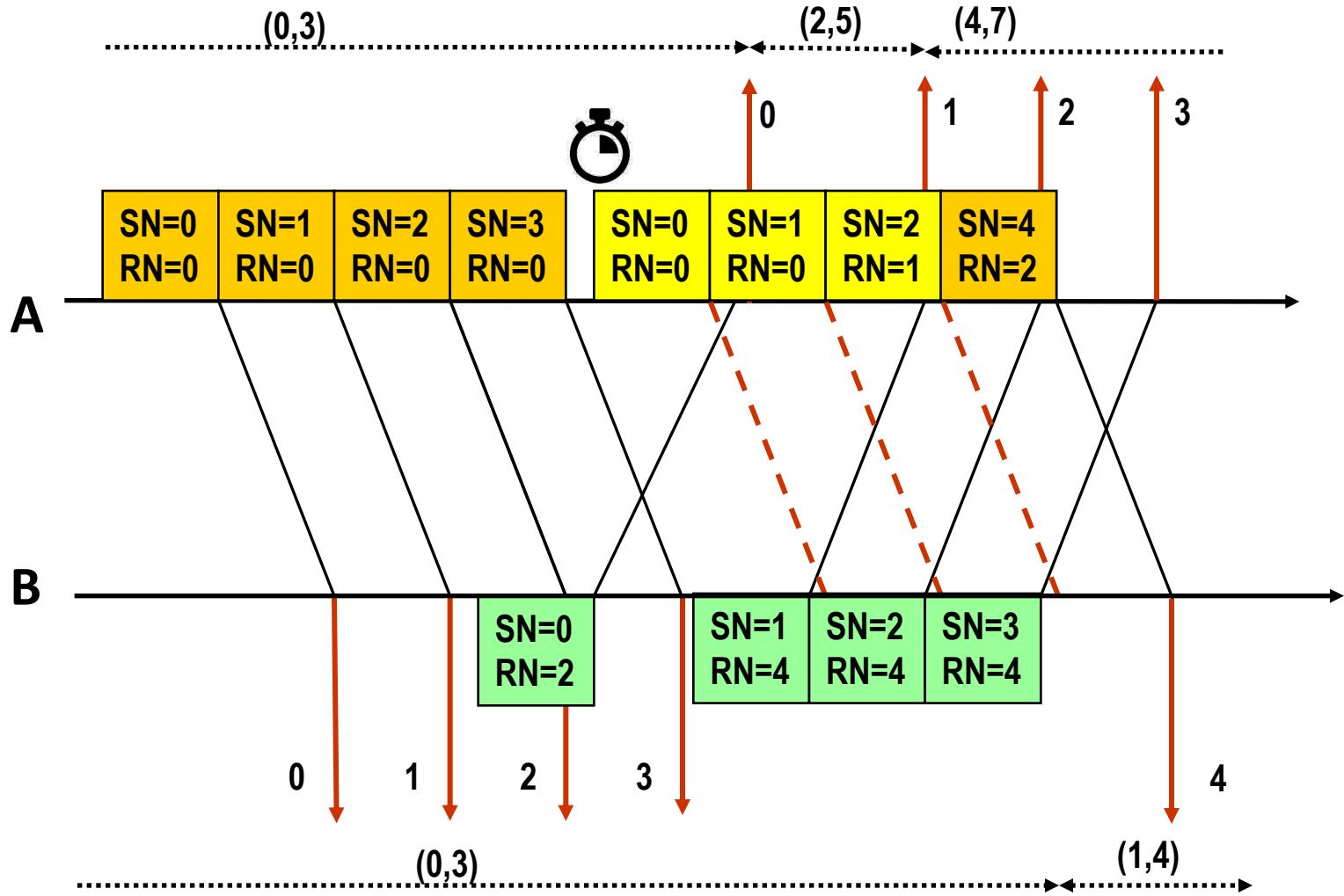
# Go-back-N full duplex con errore



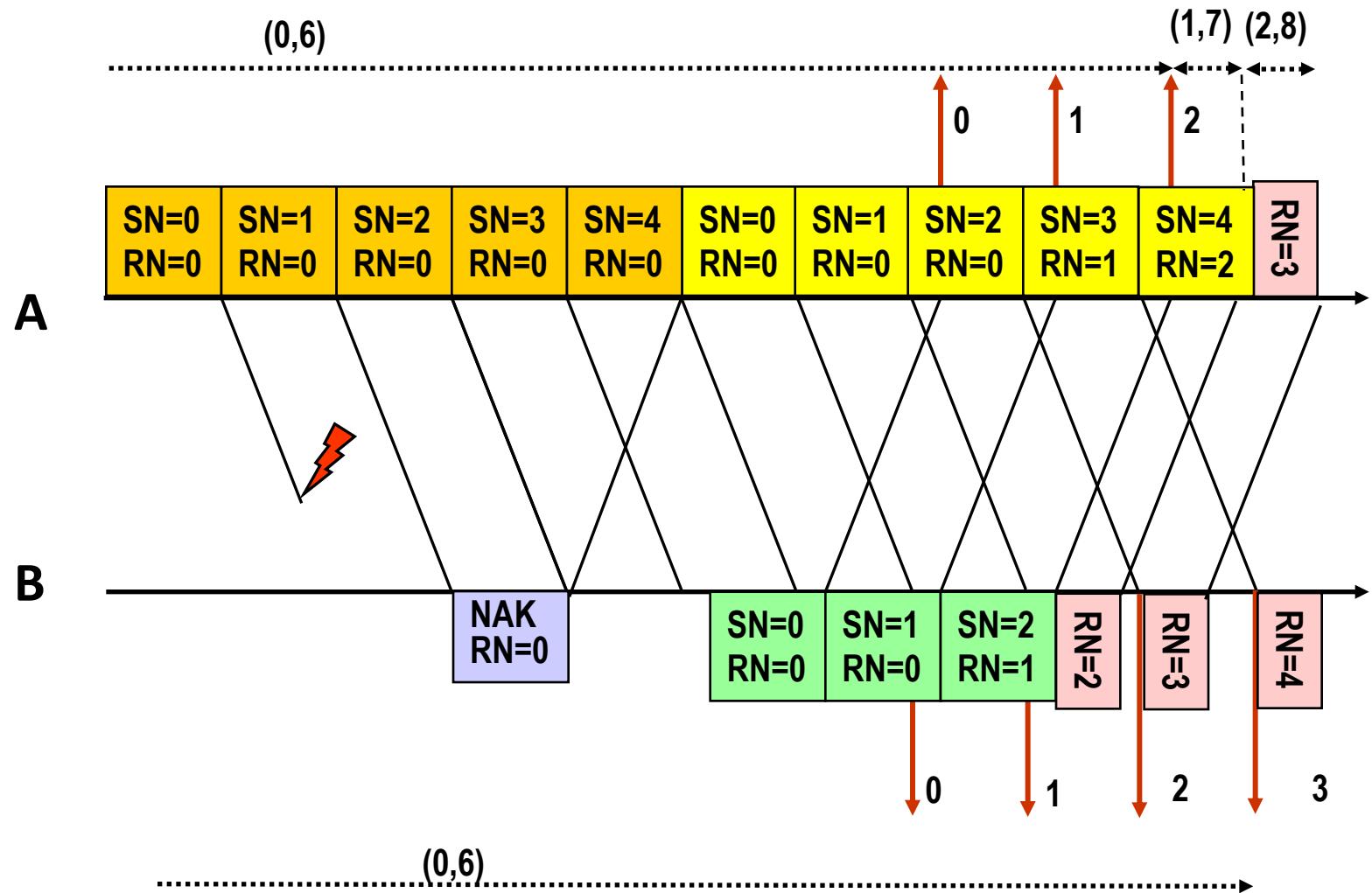
# Go-back-N full duplex con errore



# Go-back-N full con ACK ritardati



# Go-back-N con NAK



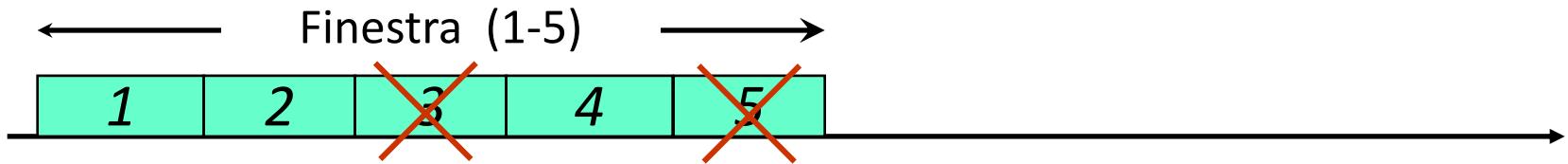
# Protocollo Selective Repeat

- Il Go-Back-N impone la ritrasmissione anche di pacchetti in realtà ricevuti correttamente
- Può essere troppo pesante se gli errori sono molti
- Come alternativa è possibile pensare di ritrasmettere i soli pacchetti errati: è questo il funzionamento *Selective Repeat* (SR)
- Modifica apparentemente banale, ma implica
  - Riordino dei pacchetti fuori sequenza
    - Buffer per immagazzinare pacchetti fuori ordine e gestione ordinamento
  - Informazione di ACK per ogni pacchetto
    - Gli ACK non sono più cumulativi

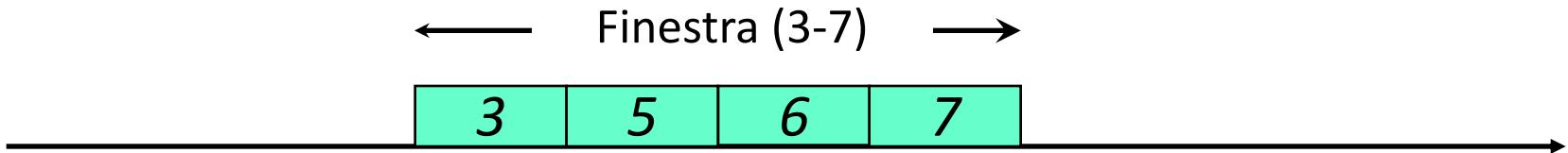


# Selective Repeat: Finestra

- E' sempre presente la finestra di N pacchetti consecutivi, però i pacchetti reali possono essere di meno
- La finestra si aggiorna scorrendo **sui pacchetti consecutivi ricevuti correttamente** (come nel GO-BACK-N)

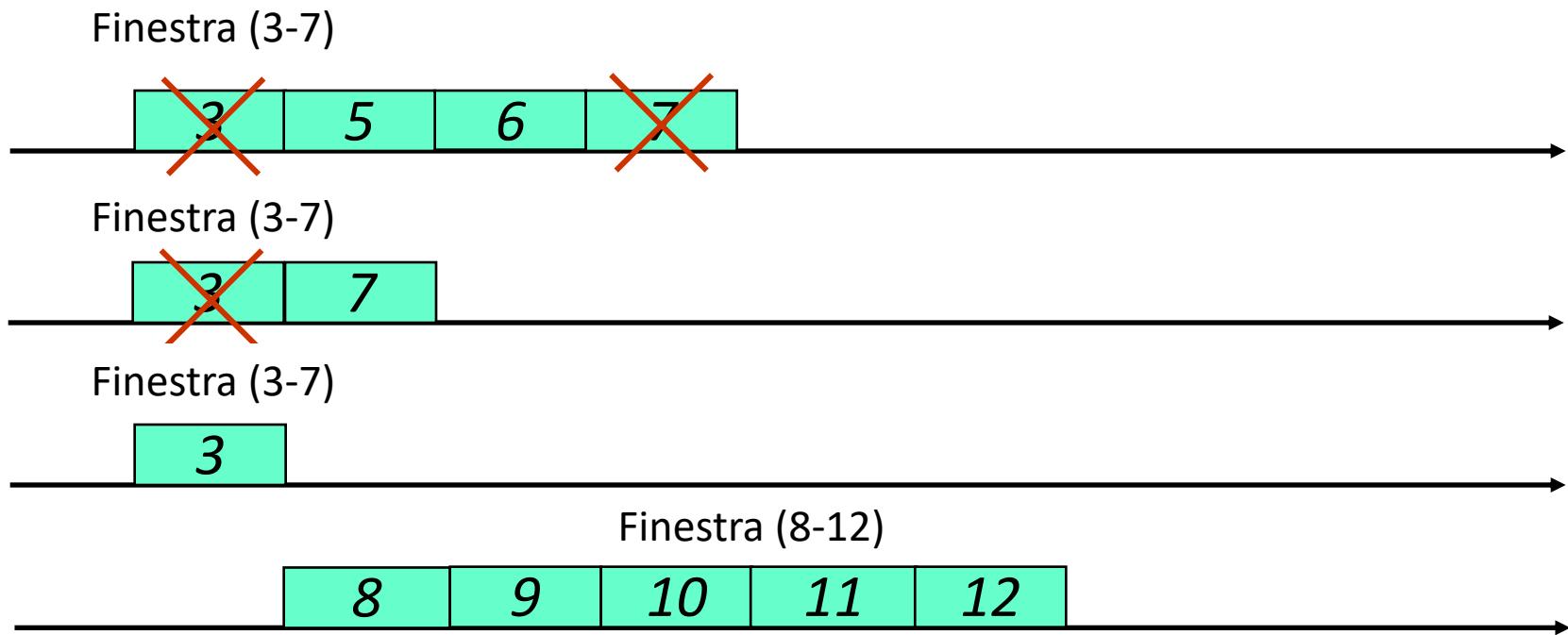


- Si arresta sul primo pacchetto senza ACK (come nel GO-BACK-N)
- Giunti a fine finestra si **trasmette la nuova finestra con la ritrasmissione dei SOLI PACCHETTI ERRATI della finestra precedente**



# Selective Repeat: Finestra

- La finestra durante le ritrasmissioni non contiene un numero fisso di pacchetti (quelli corretti non vengono ritrasmessi)



# Selective Repeat: ACK

- ACK singoli per ogni pacchetto
- ACK cumulati (RN primo pacchetto della finestra da ricevere) - più **bitmap** delle trame da ricevere nella finestra



- L'informazione necessaria è maggiore



# Osservazioni sul controllo d'errore

- Necessità di inizializzare il protocollo
  - I numeri SN e RN devono essere inizializzati
  - Deve esistere un momento di inizio non equivocabile in cui scambiare l'informazione per l'inizializzazione

Occorre un meccanismo a connessione che stabilisca l'istante  $t=0$



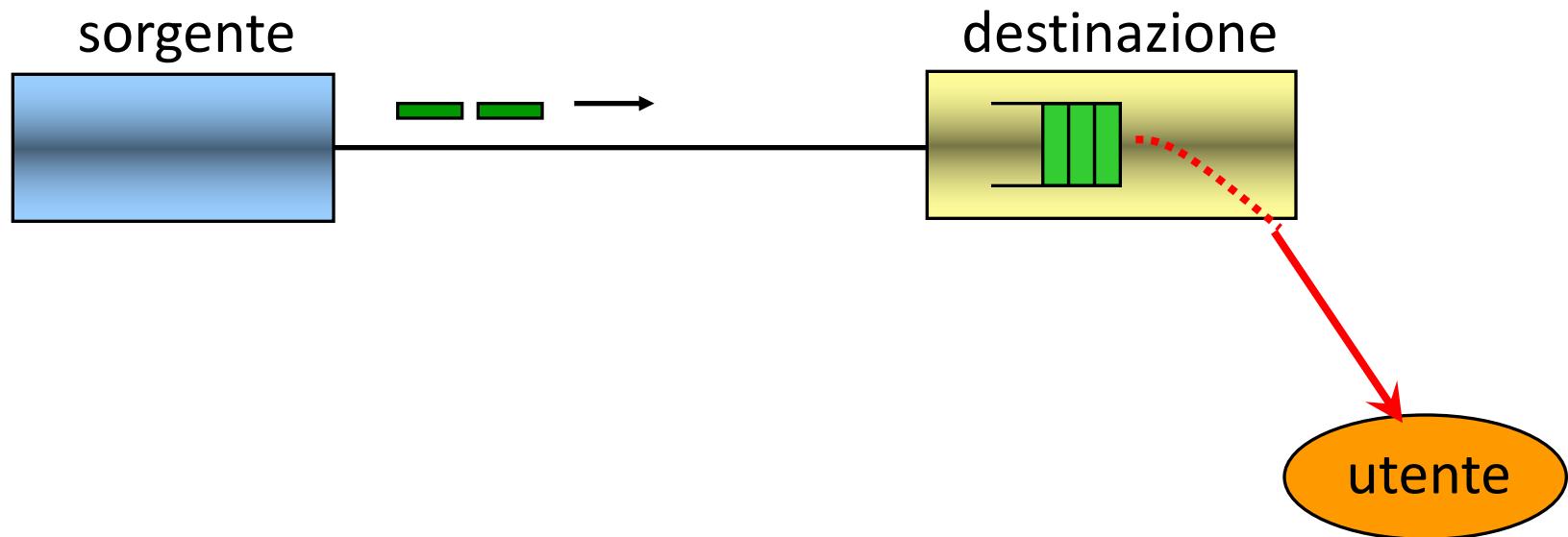
# Livello di Trasporto

- Introduzione
- Protocollo UDP
- Trasporto affidabile
  - Protocolli di ritrasmissione
  - Controllo di flusso a finestra mobile
- Protocollo TCP
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



# Controllo di flusso

- Buffer di ricezione limitato a **W** posizioni
- Ritmo di assorbimento dell'utente arbitrario
- Obiettivo: regolando il ritmo di invio, evitare che pacchetti vadano persi perché all'arrivo trovano il buffer pieno

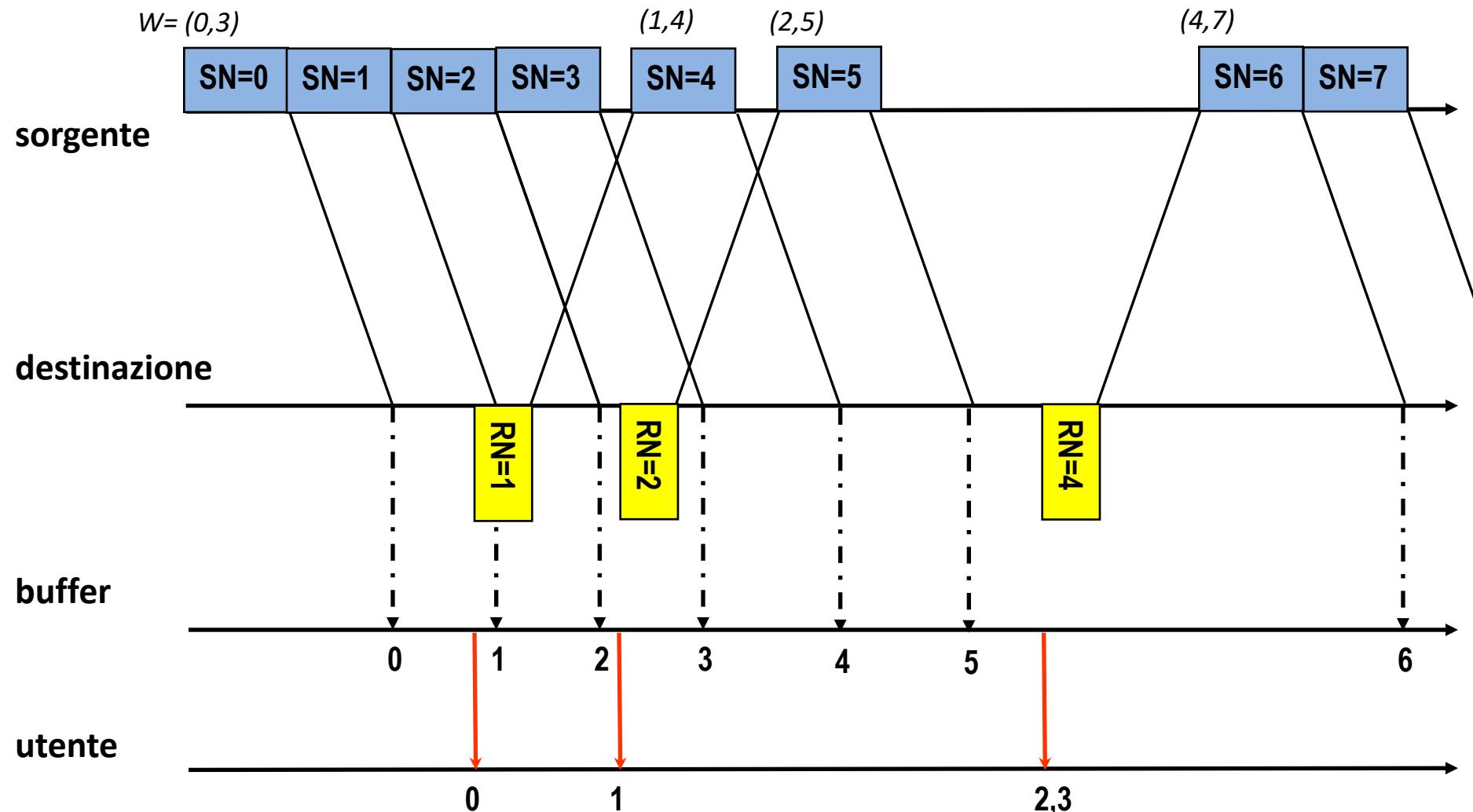


# Sliding window flow control

- Controllo di flusso a finestra mobile
- E' possibile usare un meccanismo come quello del *Go-back-N*
- La sorgente non può inviare più di  $W$  trame (stessa funzione del parametro  $N$ ) senza aver ricevuto il riscontro
- I riscontri vengono inviati dal ricevitore solo quando i pacchetti vengono letti (**tolti dal buffer**) dal livello superiore



# Sliding-window flow control



# Problema delle ritrasmissioni

- Il meccanismo di controllo di flusso descritto nelle slide precedenti è strettamente legato al meccanismo di controllo d'errore e questo può essere fonte di problemi
- Se il ricevitore ritarda molto l'invio dei riscontri a causa del livello superiore lento, il trasmettitore inizia la ritrasmissione perché scade il time-out
- Aumentare troppo il time-out non è ovviamente una soluzione in quanto l'aumento del time-out aumenta i ritardi in caso di errore

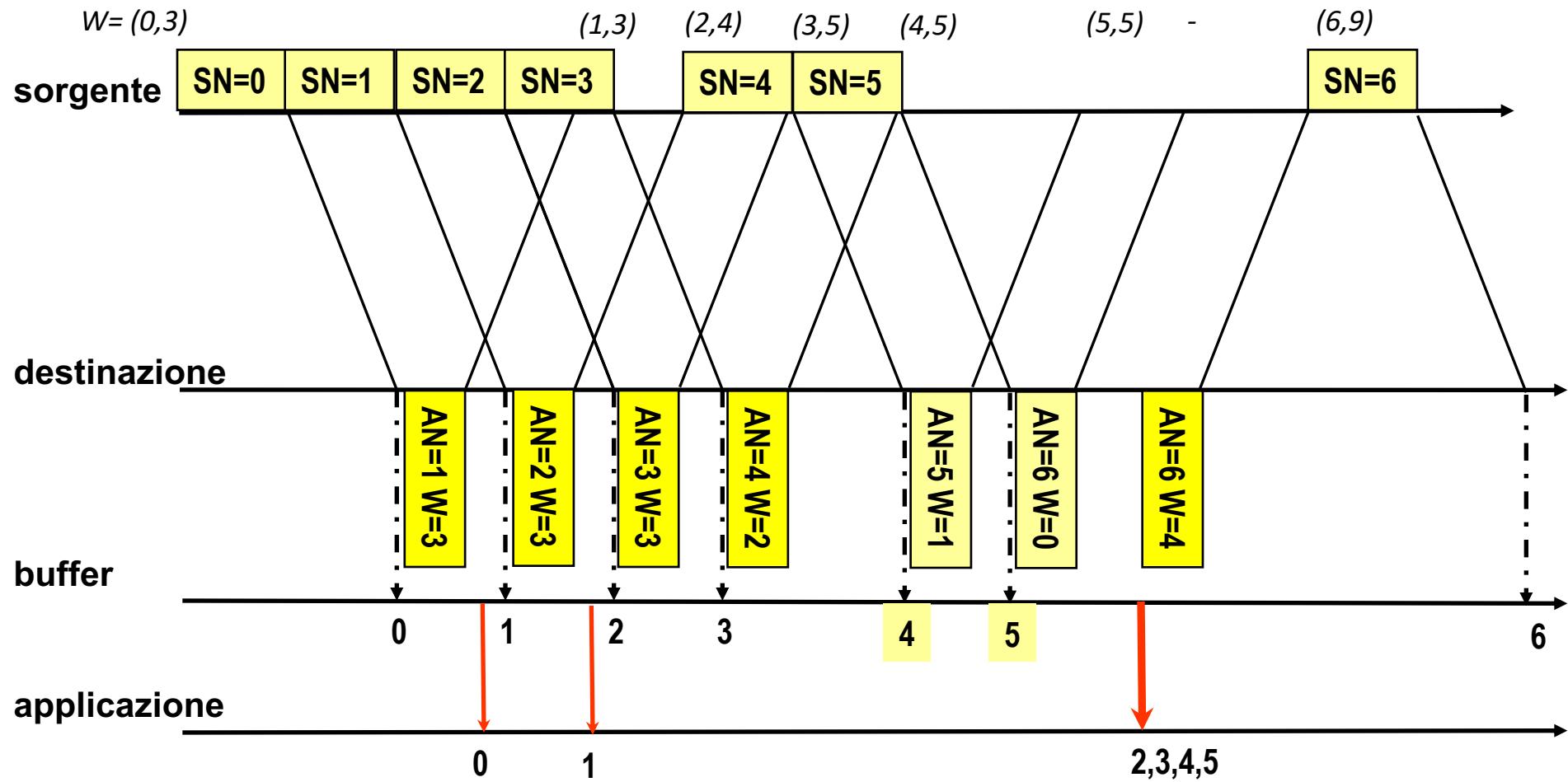


# Uso del campo W

- Il problema può essere risolto in modo radicale separando i meccanismi di controllo d'errore e di controllo di flusso a finestra
- Si inserisce nei riscontri (o nell'header delle trame in direzione opposta) un campo finestra W (insieme a quella del GBN)
  - Il ricevitore invia i riscontri sulla base dell'arrivo dei pacchetti
  - E usa il campo W per indicare lo spazio rimanente nel buffer



# Uso del campo W



# Uso del campo W

- Gestione della finestra del controllo di flusso
  - Non è necessario che il ricevitore dica la “verità” sullo spazio restante R
  - Può tenersi un margine di sicurezza ( $W=R-m$ )
  - Può aspettare che il buffer si sia svuotato per una frazione (ad es.  $W=0$  se  $R < K/2$  ed  $W=R$  altrimenti)
  - Può usare dei meccanismi adattativi





# **7 – Il Livello di Trasporto**

## **Parte III**

# Livello di Trasporto

- Introduzione
- Protocollo UDP
- Trasporto affidabile
  - Protocolli di ritrasmissione
  - Controllo di flusso a finestra mobile
- **Protocollo TCP**
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



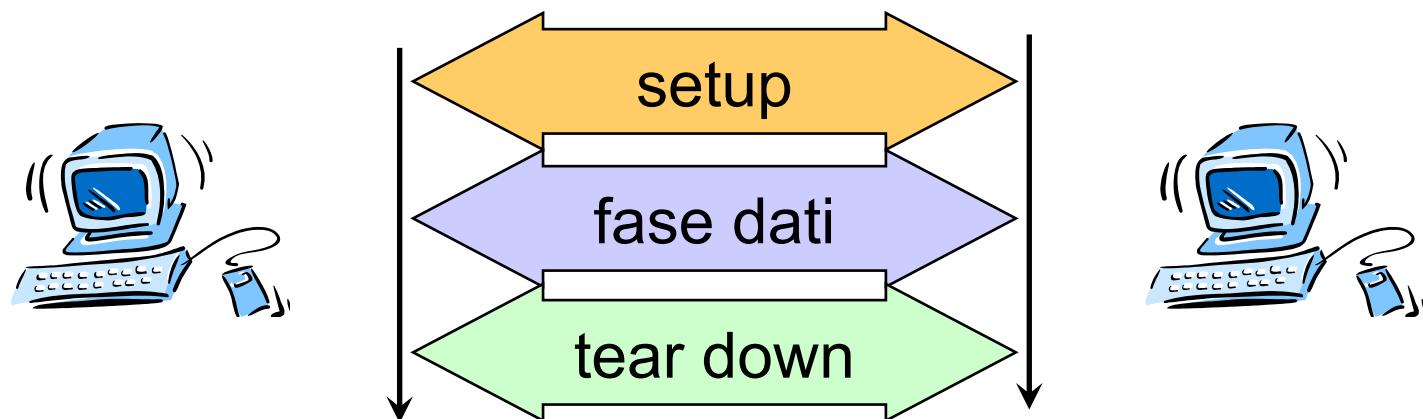
# Transmission Control Protocol (TCP)

- Il TCP è un protocollo di trasporto che assicura il trasporto affidabile
  - In corretta sequenza
  - Senza errori/perdite dei dati
- Mediante TCP è possibile costruire applicazioni che si basano sul trasferimento di file senza errori tra host remoti (web, posta elettronica, ecc.)
- E' alla base della filosofia originaria di Internet: servizio di rete semplice e non affidabile, servizio di trasporto affidabile
- Il TCP effettua anche un controllo di congestione *end-to-end* che limita il traffico in rete e consente agli utenti di condividere in modo equo le risorse



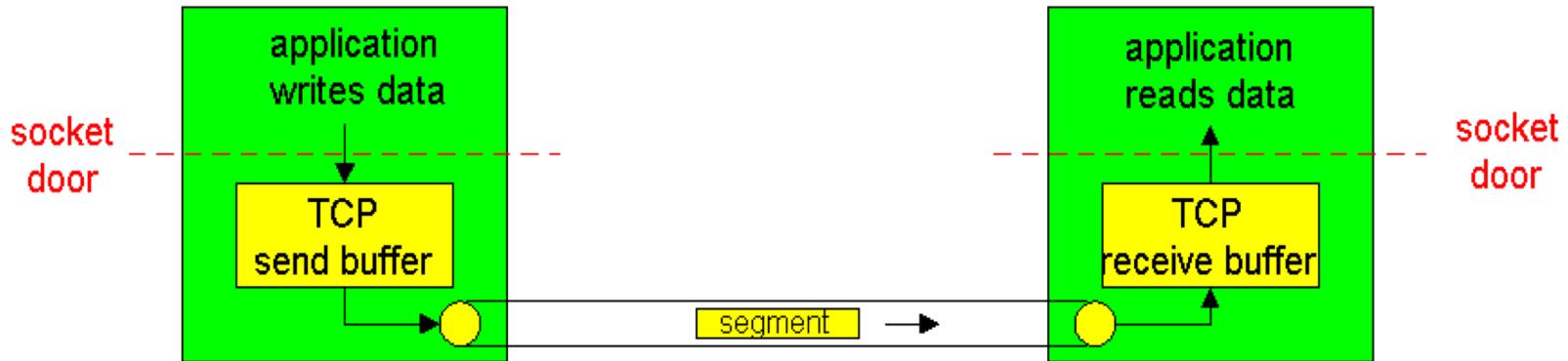
# TCP: *connection oriented*

- Il TCP è orientato alla connessione (*connection oriented*):
  - Prima del trasferimento di un flusso dati occorre instaurare una connessione mediante opportuna segnalazione
  - Le connessioni TCP si appoggiano su una rete *connectionless* (datagram)
  - Le connessioni TCP sono di tipo *full-duplex* (esiste sempre un flusso di dati in un verso e nel verso opposto, anche se questi possono essere quantitativamente diversi)



# TCP: flusso dati

- Il TCP è orientato alla trasmissione di flussi continui di dati (stream di byte)
- Il TCP converte il flusso di dati in *segmenti* che possono essere trasmessi in IP
- Le dimensioni dei segmenti sono variabili
- L'applicazione trasmittente passa i dati (byte) a TCP e TCP li accumula in un buffer.
- Periodicamente, o quando avvengono particolari condizioni, il TCP prende una parte dei dati nel buffer e forma un segmento
- La dimensione del segmento è critica per le prestazioni, per cui il TCP cerca di attendere fino a che un ammontare ragionevole di dati sia presente nel buffer di trasmissione



# TCP: numerazione byte e riscontri

- Il TCP adotta un meccanismo per il controllo delle perdite di pacchetti di tipo **Go-Back-N**
- Sistema di numerazione e di riscontro dei dati inviati
  - TCP numera ogni byte trasmesso, per cui ogni byte ha un numero di sequenza
  - Nell'*header* del segmento TCP è trasportato il numero di sequenza del primo byte nel segmento stesso
  - Il ricevitore deve riscontrare i dati ricevuti inviando il numero di sequenza del prossimo byte che ci si aspetta di ricevere.
  - Se un riscontro non arriva entro un dato *timeout*, i dati sono ritrasmessi

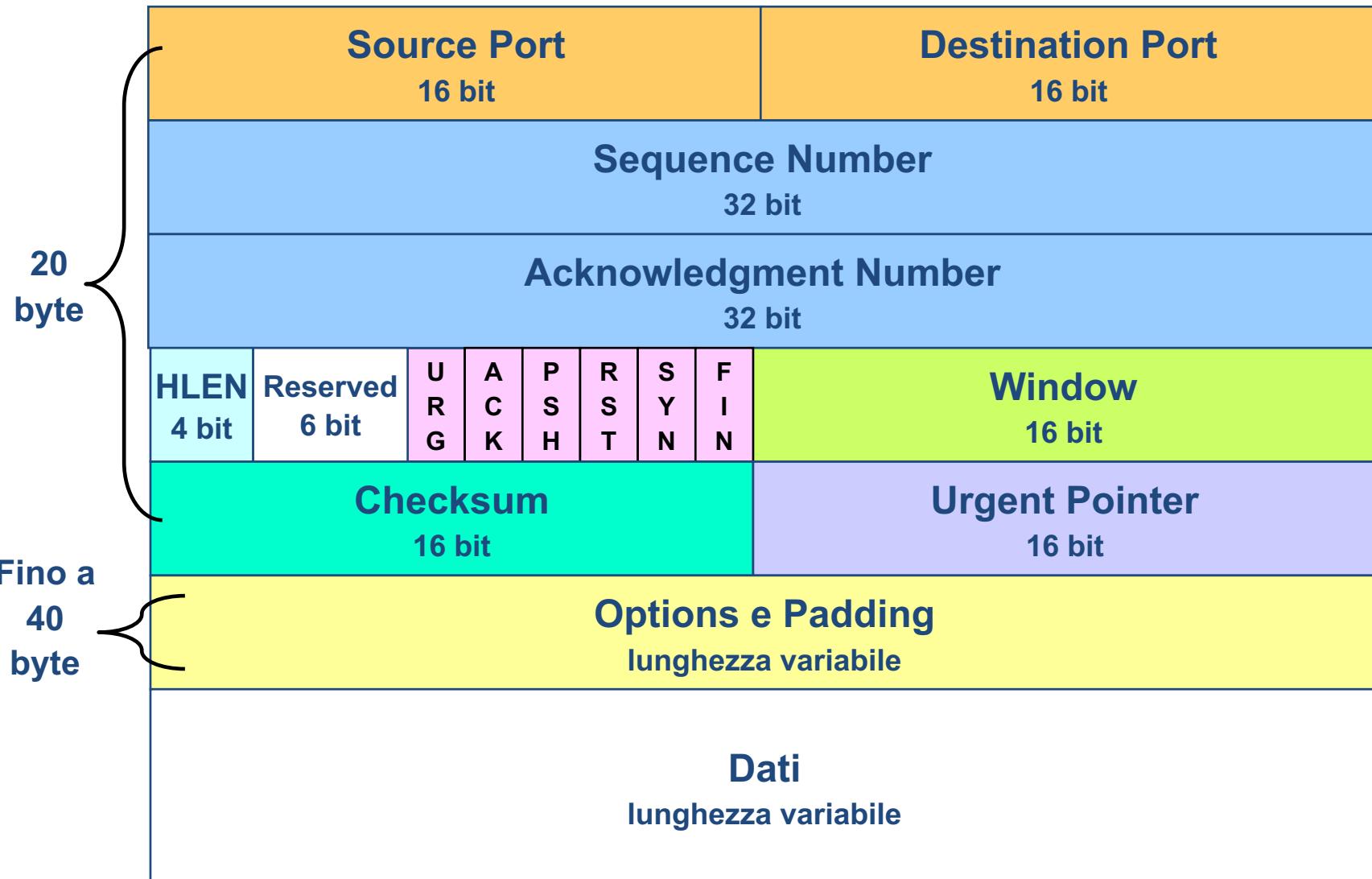


# Livello di Trasporto

- Introduzione
- Protocollo UDP
- Trasporto affidabile
  - Protocolli di ritrasmissione
  - Controllo di flusso a finestra mobile
- Protocollo TCP
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



# Segmento TCP



# Header Segmento TCP I

**Source port, Destination port:** indirizzi di porta sorgente e porta destinazione di 16 bit

**Sequence Number:** il numero di sequenza del primo byte nel payload

**Acknowledge Number:** numero di sequenza del prossimo byte che si intende ricevere (numero valido solo se flag ACK valido)

**HLEN (4 bytes words):** contiene la lunghezza complessiva dell'header TCP, che DEVE essere un multiplo intero di 32 bit

**Reserved 6 bit:** non utilizzati

6 “flag” di valore 1 o 0 per funzioni di servizio:

**URG** è 1 se ci sono dati urgenti

**ACK** è 1 se il pacchetto è un ACK valido (l'acknowledge num con numero valido)

**PSH** è 1 quando TX vuole usare il comando di PUSH

**RST** *reset*, resetta la connessione senza un *tear down*

**SYN** *synchronize*, usato durante il setup per comunicare numeri di sequenza iniziali



# Header Segmento TCP II

**Window:** contiene il valore della finestra di ricezione come comunicato dal ricevitore al trasmettitore

**Checksum:** il medesimo di UDP, calcolato in maniera uguale

**Options and Padding:** da 0 a 40 byte, da riempire (con multipli di 32 bit) per campi opzionale, per es. durante il setup per comunicare il **MAXIMUM SEGMENT SIZE (MSS)** (il valore di default è 536 byte, il valore massimo è 65535 byte).



# Servizi e porte

- La divisione tra porte note, assegnate e dinamiche è la stessa che per UDP
- Alcuni delle applicazioni più diffuse:

22	SSH
21	FTP signalling
20	FTP data
23	telnet
25	SMTP
53	DNS
80	HTTP
110	POP
143	IMAP



# Livello di Trasporto

- Introduzione
- Protocollo UDP
- Trasporto affidabile
  - Protocolli di ritrasmissione
  - Controllo di flusso a finestra mobile
- Protocollo TCP
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



# Controllo d'errore

- Il meccanismo di controllo d'errore del TCP serve a recuperare pacchetti persi in rete
- La causa principale della perdita è l'*overflow* di una delle code dei *router* attraversati a causa della congestione
- Il meccanismo di ritrasmissione è di tipo *Go-back-N* con *Timeout*
- La finestra di trasmissione (valore di N) dipende dal meccanismo di controllo di flusso e di congestione
- L'orologio per la ritrasmissione di un segmento viene inizializzato al momento della trasmissione e determina la ritrasmissione quando raggiunge il valore del *Timeout*



# Gestione del Time-Out

- Uno dei problemi è stabilire il valore ottimo del timeout:
  - Timeout troppo breve, il trasmettitore riempirà il canale di ritrasmissioni di segmenti,
  - Timeout troppo lungo impedisce il recupero veloce di reali errori
- Il valore ottimale dipende fortemente dal ritardo in rete (rete locale o collegamento satellitare?)
- Il TCP calcola dinamicamente un valore opportuno per il timeout stimando il RTT (*Round Trip Time*)



# Livello di Trasporto

- Introduzione
- Protocollo UDP
- Trasporto affidabile
  - Protocolli di ritrasmissione
  - Controllo di flusso a finestra mobile
- Protocollo TCP
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



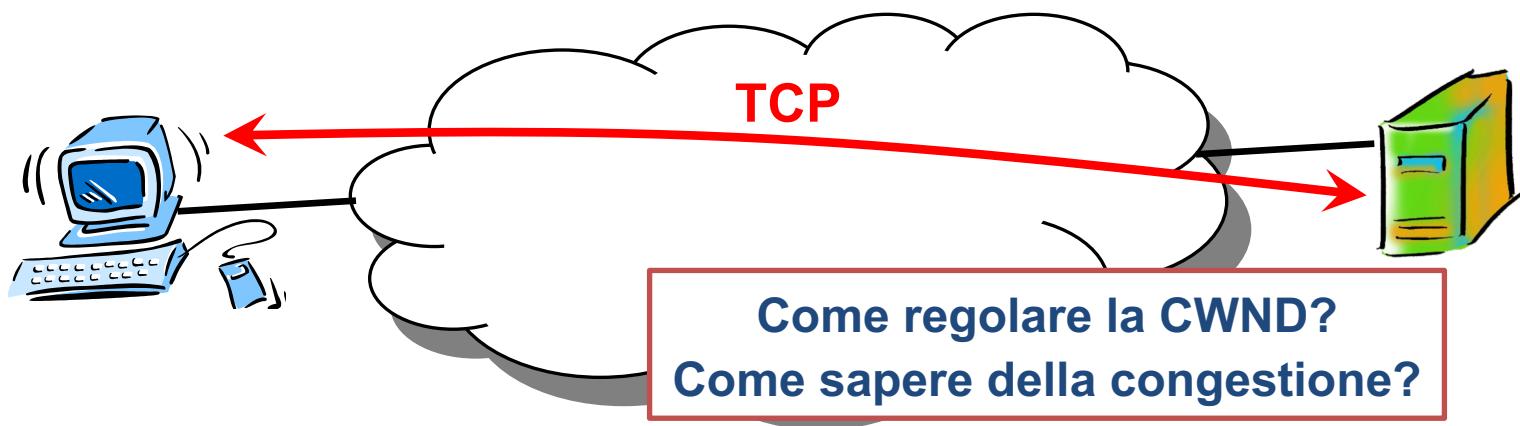
# Controllo di congestione

- Il controllo di flusso
  - Dipende **solo** dalla “capacità” del ricevitore
  - Non è sufficiente ad evitare la congestione nella rete
- Nella rete INTERNET attuale non ci sono meccanismi sofisticati di controllo di congestione a livello di rete (come ad esempio meccanismi di controllo del traffico in ingresso)
- Il controllo di congestione è delegato al TCP!!!
  - Se il traffico in rete porta a situazioni di congestione il TCP deve ridurre velocemente il traffico in ingresso
- Essendo il TCP implementato solo negli *host* il controllo di congestione è di tipo *end-to-end*



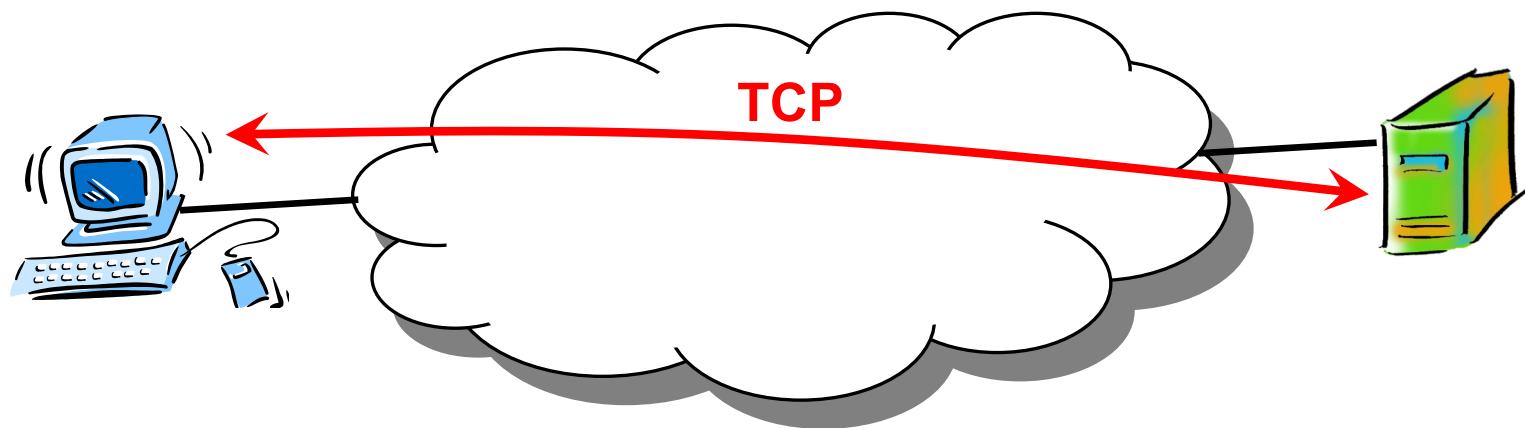
# Controllo di congestione

- Il modo più naturale per controllare il ritmo di immissione in rete dei dati per il TCP è quello di regolare la finestra di trasmissione
- Il trasmettitore mantiene una *Congestion Window* (CWND) che varia in base agli eventi che osserva (ricezione ACK, timeout)
- Il trasmettitore non può trasmettere più del minimo tra RCVWND (Receive Window – spazio del buffer in ricezione disponibile per ricevere nuovi dati) e CWND



# Controllo di congestione

- L'idea base del controllo di congestione del TCP è quella di interpretare la perdita di un segmento, segnalata dallo scadere di un timeout di ritrasmissione, come un evento di congestione
- La reazione ad un evento di congestione è quella di ridurre la finestra (*CWND*)



# Slow Start & Congestion Avoidance

- Il valore della finestra CWND viene aggiornato dal trasmettitore TCP in base ad un algoritmo
- Il modo in cui avviene l'aggiornamento dipende dalla fase (o stato) in cui si trova il trasmettitore
- Esistono due fasi fondamentali:



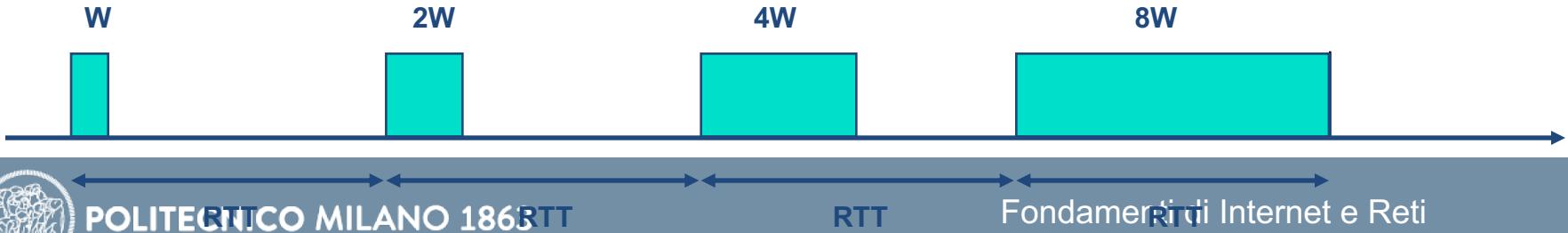
- Slow Start
- Congestion Avoidance

- La variabile STHRESH è mantenuta al trasmettitore per distinguere le due fasi:
  - ➔ se  $CWND < STHRESH$  si è in *Slow Start*
  - ➔ se  $CWND > STHRESH$  si è in *Congestion Avoidance*



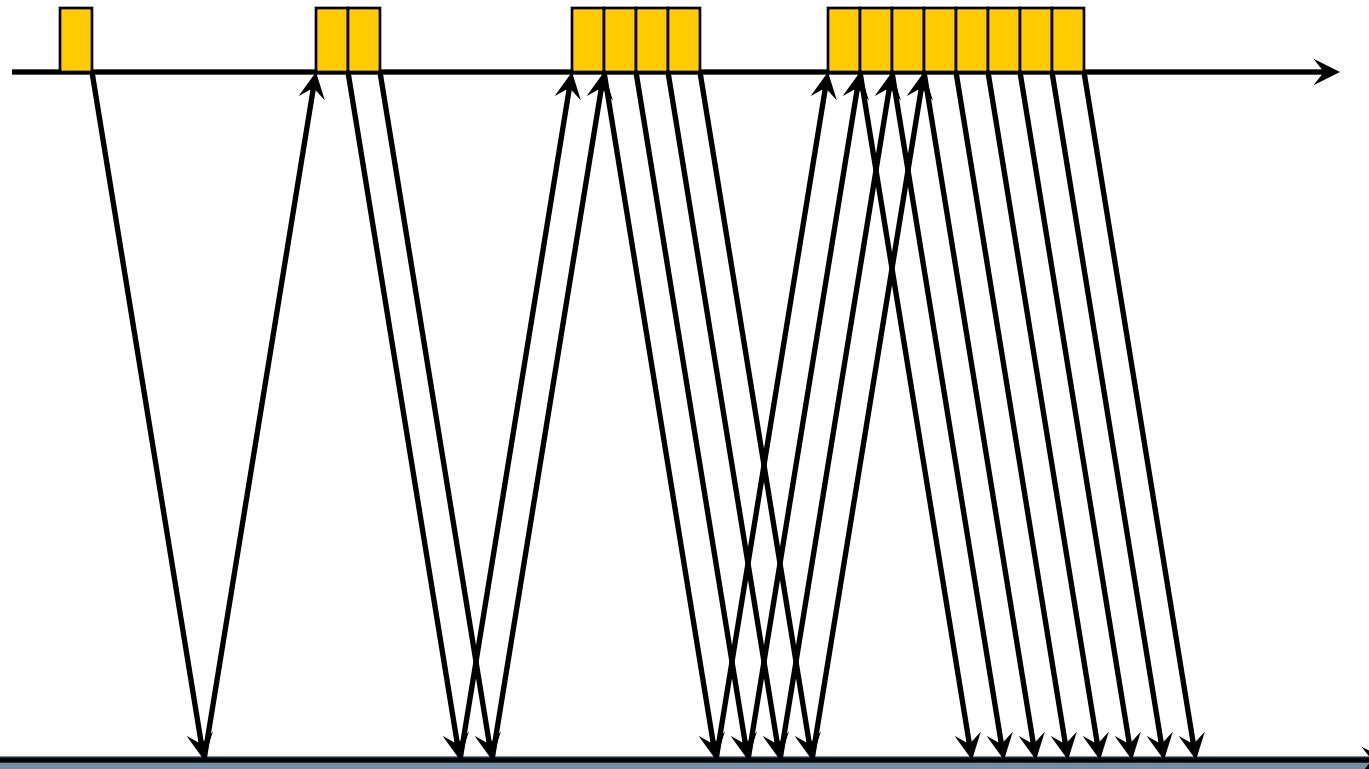
# Slow Start

- All'inizio, il trasmettitore pone la CWND a 1 segmento (MSS) e la SSTHRESH ad un valore di *default* molto elevato
- Essendo CWND < SSTHRESH si parte in *Slow Start*
- *In Slow Start:*
  - La CWND viene incrementata di 1 per ogni ACK ricevuto
- Si invia un segmento e dopo RTT si riceve l'ACK, si pone CWND a 2 e si inviano 2 segmenti, si ricevono 2 ACK, si pone CWND a 4 e si inviano 4 segmenti, ...



# Slow Start

- Al contrario di quanto il nome faccia credere l'incremento della finestra avviene in modo esponenziale (raddoppia ogni RTT)



# Slow Start

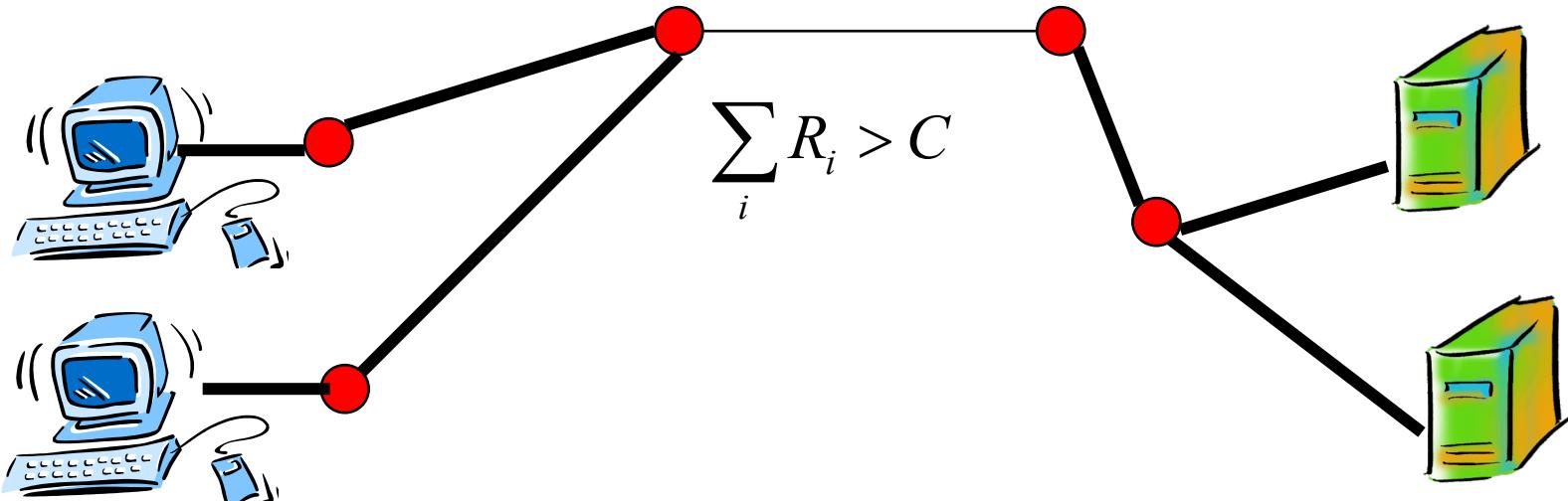
- L'incremento può andare avanti fino
  - Primo evento di congestione
  - Fino a che CWND < SSTHRESH
  - CWND < RCWND
- Insieme alla finestra aumenta il ritmo (o rate) di trasmissione che può essere stimato come:

$$R = \frac{CWND}{RTT} \text{ [bit/s]}$$



# Evento di Congestione

- Un evento di congestione si verifica quando il ritmo di trasmissione porta in congestione un link sul percorso in rete verso la destinazione
- Un link è congestionato quando la somma dei ritmi di trasmissione dei flussi che lo attraversano è maggiore della sua capacità



# Evento di congestione

- Scade un timeout di ritrasmissione
  - Il TCP reagisce ponendo SSTHRESH uguale alla metà dei “byte in volo” (byte trasmessi ma non riscontrati); più precisamente
  - E ponendo CWND a 1MSS
- Si noti che di solito i “byte in volo” sono con buona approssimazione pari a CWND



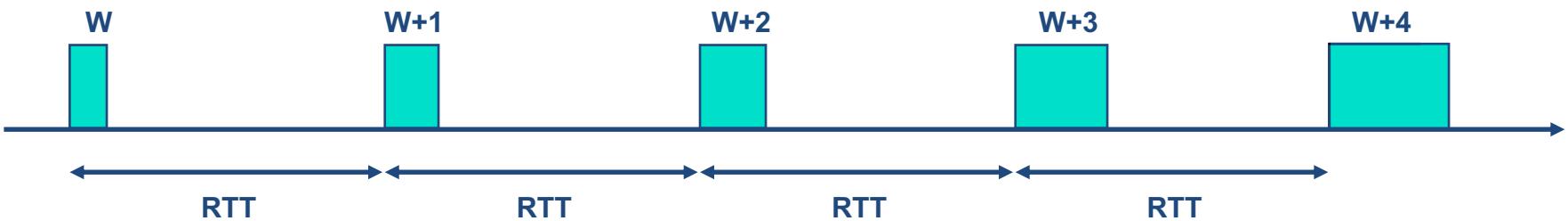
# Evento di congestione

- Come risultato:
  - CWND è minore di SSTHRESH e si entra nella fase di Slow Start
  - Il trasmettitore invia un segmento e la sua CWND è incrementata di 1 ad ogni ACK
- Il trasmettitore trasmette tutti i segmenti a partire da quello per cui il timeout è fallito (politica Go-Back-N)
- Il valore a cui è posta la SSTHRESH è una stima della finestra ottimale che eviterebbe futuri eventi di congestione



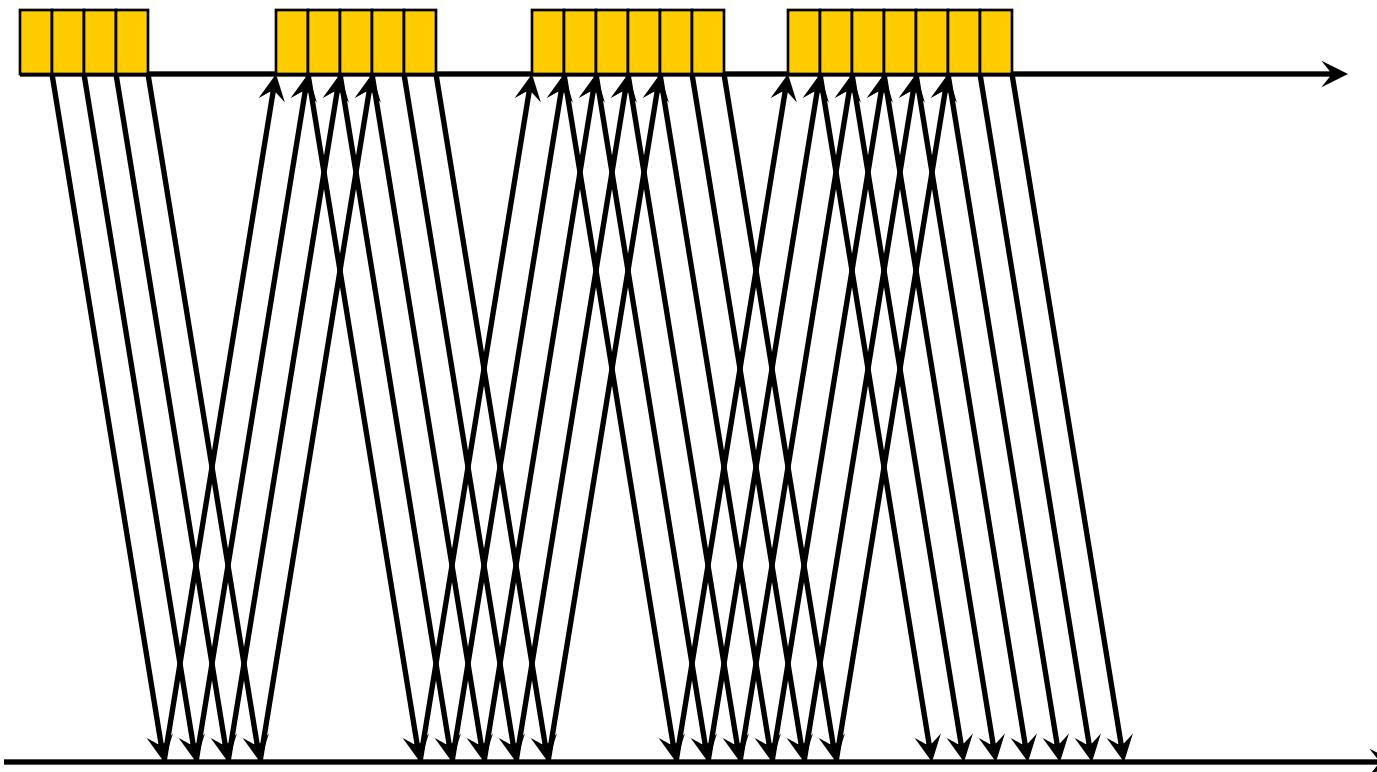
# Congestion Avoidance

- Lo slow start continua fino a che CWND diventa grande come SSTHRESH e poi parte la fase di *Congestion Avoidance*
- Durante il *Congestion Avoidance*:
  - Si incrementa la CWND di  $1/\text{CWND}$  ad ogni ACK ricevuto
- Se la CWND consente di trasmettere N segmenti, la ricezione degli ACK relativi a tutti gli N segmenti porta la CWND ad aumentare di 1 segmento
- In Congestion Avoidance si attua un incremento lineare della finestra di congestione

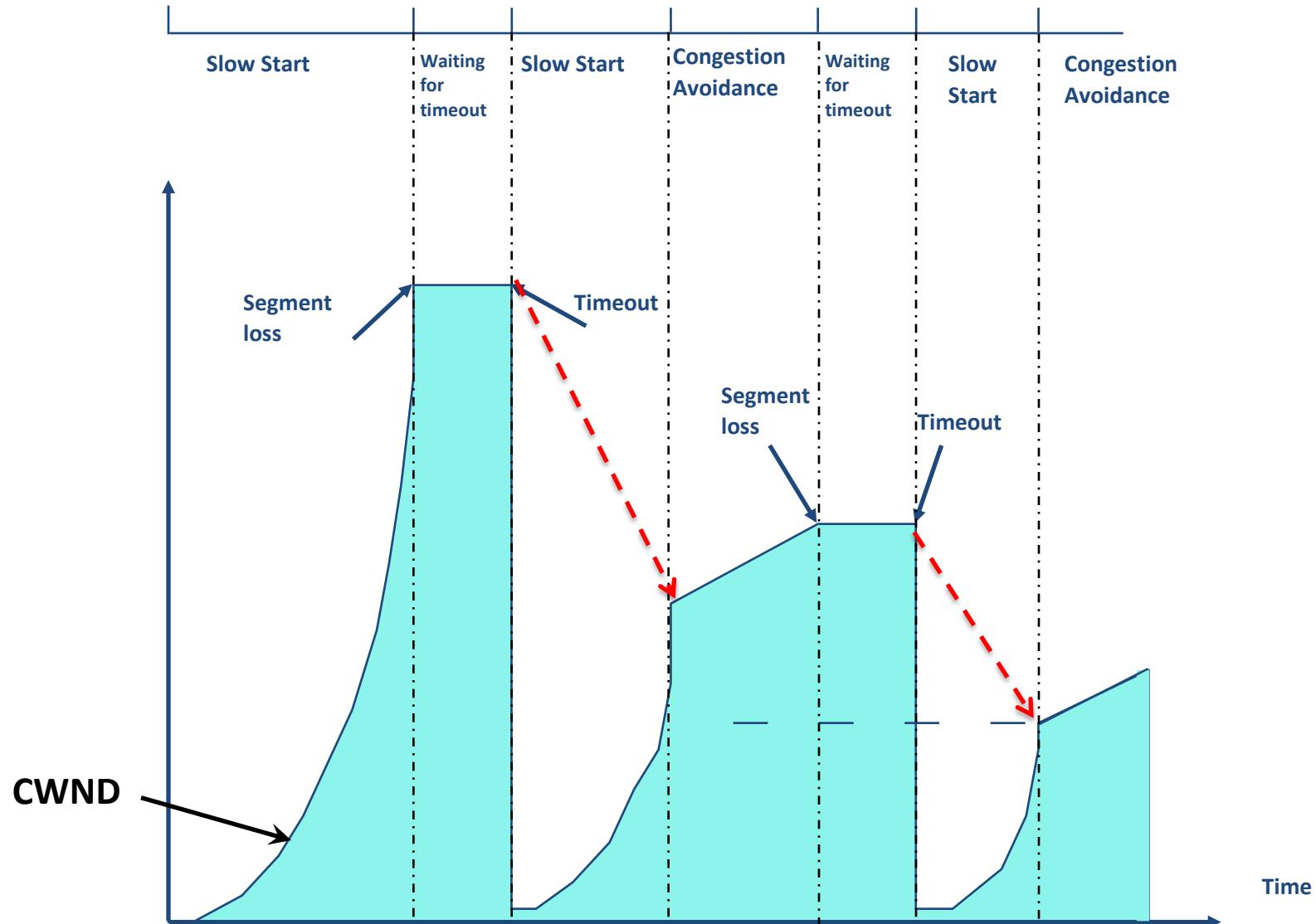


# Congestion Avoidance

- Dopo aver raggiunto STHRESH la finestra continua ad aumentare ma molto più lentamente

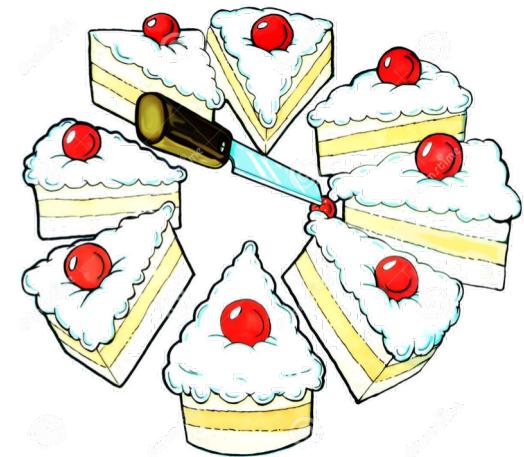


# Esempio di funzionamento



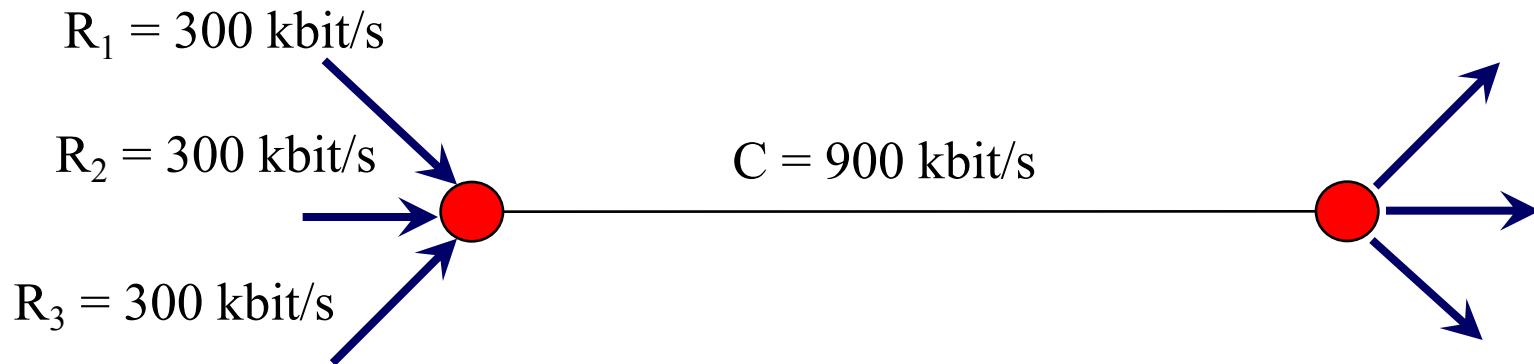
# Condivisione equa delle risorse

- In condizioni ideali il meccanismo di controllo del TCP è in grado di
  - Limitare la congestione in rete
  - Consentire di dividere in modo equo la capacità dei link tra i diversi flussi
- Le condizioni ideali sono alterate tra l'altro da
  - Differenti RTT per i diversi flussi
  - Dimensione dei Buffer

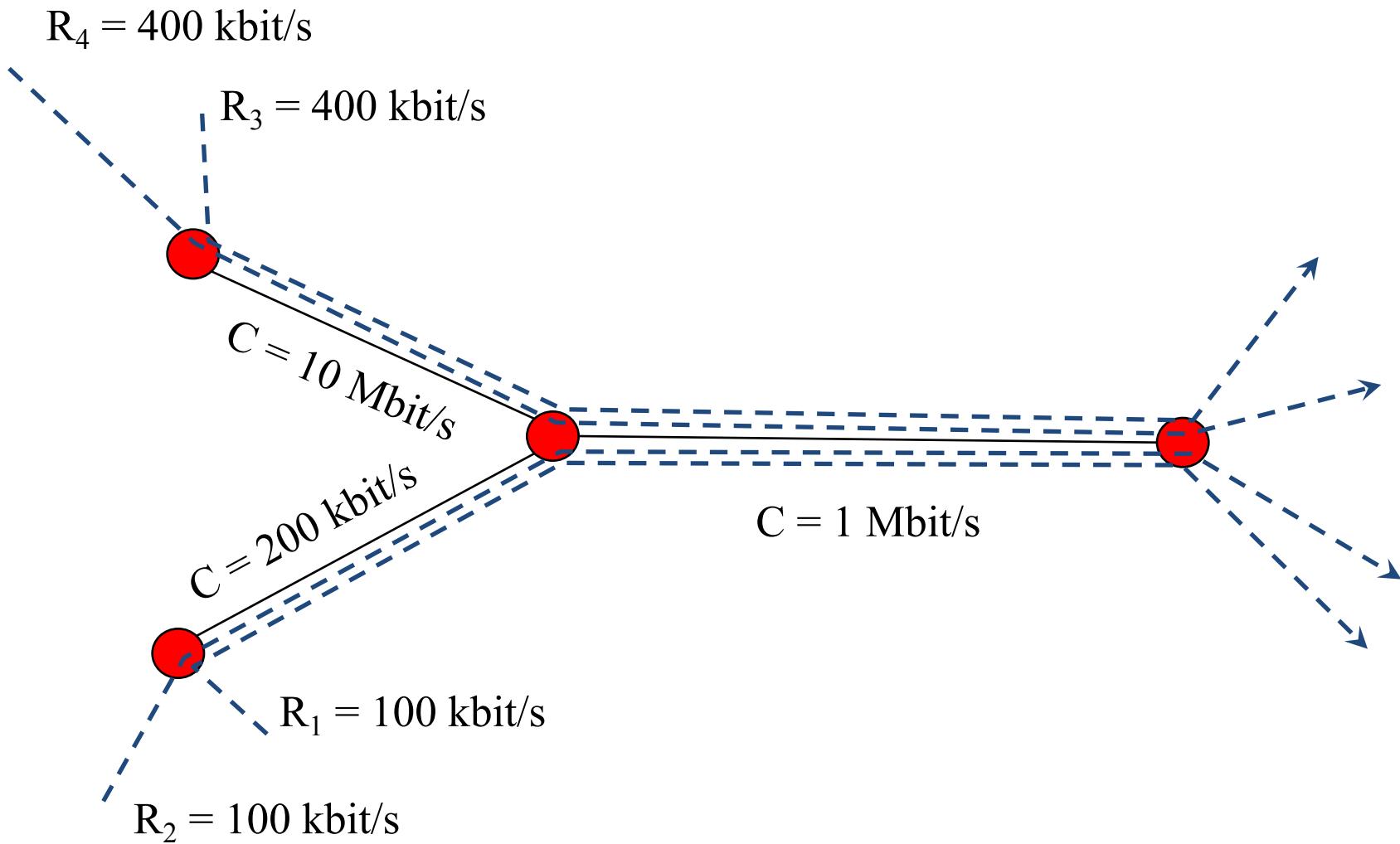


# Condivisione equa delle risorse

- I valori dei rate indicati sono solo valori medi e valgono solo in condizioni ideali
- Il ritmo di trasmissione in realtà cambia sempre e in condizioni non ideali la condivisione può non essere equa

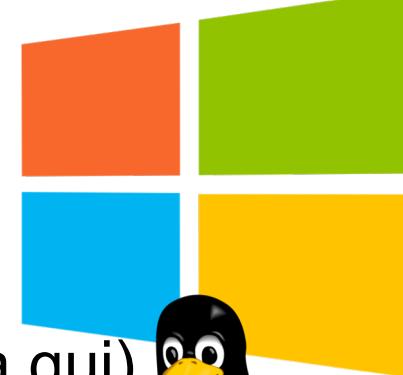


# Condivisione equa delle risorse



# TCP e sistemi operativi

- Esistono diverse versioni di protocollo TCP
- TCP è implementato nel sistema operativo
- Versione base è TCP **Tahoe** (versione presentata qui)



Famiglia Windows

per esempio **Reno / New Reno**

Famiglia Linux

Altro approccio al Livello 4 di trasporto. Per esempio **BIC** per grandi prodotti banda ritardo o **CUBIC** (non guarda gli ACK)

Famiglia MacO

prima protocolli proprietari, per esempio **MacTCP**. Poi **CUBIC**

