



*Corso di*  
***SISTEMI TELEMATICI***  
*a.a. 2011-2012*

**Lo strato di Trasporto: instaurazione  
della connessione, controllo del flusso  
e correzione degli errori**



# TCP: Instaurazione di una connessione

- TCP è un protocollo orientato alla connessione; sono presenti le fasi di instaurazione della connessione, trasferimento dati ed abbattimento della connessione
- Prima di iniziare il trasferimento di informazioni verso un utente remoto, deve essere instaurata una connessione, tramite un meccanismo di sincronizzazione noto come three-way handshake (stretta di mano in 3 fasi)



# Instaurazione di una connessione

La sincronizzazione serve a risolvere potenziali situazioni anomale, dovute al fatto che IP non è affidabile e quindi i datagrammi possono essere persi, ritardati, duplicati o consegnati fuori sequenza, e che TCP ritrasmette i segmenti persi dopo un certo timeout:

- ✗ un segmento appartenente ad una “vecchia” connessione può arrivare ad un host dopo che tra i processi relativi a quel segmento è stata instaurata una nuova connessione
- ✗ un host “cade” e perde traccia delle connessioni ancora in atto



# Instaurazione di una connessione

## Obiettivo

- Evitare la presenza di segmenti duplicati ritardati (stesso numero di sequenza); assicurarsi che un pacchetto sia “morto” e che siano “morti” anche tutti i riscontri relativi a quel pacchetto dopo un tempo  $T$   
( $\rightarrow$ MSL, maximum segment lifetime)

## Soluzione

- Ogni host usa un clock (non sincronizzato con gli altri) che incrementa un contatore binario a passi regolari (la cadenza dipende dalla velocità della rete; a 2 Mbit/s è di 4 ms, cioè  $1/(2M/8)$ )
- Il valore del clock è usato per numerare i byte generati
- Il numero di bit del contatore deve essere maggiore o uguale del numero di bit dei numeri di sequenza (32 bit)
- Il clock deve continuare a funzionare anche quando l'host va in “crash”



# Instaurazione di una connessione

## Requisiti

- Ogni volta che si instaura una nuova connessione l'host trasmittente sceglie in modo pseudo-casuale numero di sequenza iniziale (ISN) di 32 bit da cui partire, ponendolo uguale al valore del suo contatore binario, sincronizzato al clock locale
  - ✗ Il numero di sequenza si sceglie in modo pseudo-casuale tra 1 e  $2^{32} = 4.294.967.296$
- Gli ottetti successivi sono numerati sequenzialmente a partire dal valore di ISN scelto durante la procedura di 3-way handshake
- Lo spazio dei numeri di sequenza ( $2^{32}$ ) deve essere abbastanza ampio in modo che quando i numeri di sequenza si ripetono i vecchi datagrammi con lo stesso numero di sequenza siano scomparsi dalla rete



# Instaurazione di una connessione

- Dato che si assume che i segmenti restino in rete per non più del Maximum Segment Lifetime (MSL), si vuole che MSL sia minore del tempo di ciclo del clock
- La cadenza del clock dipende dalla velocità della rete; a 2 Mbit/s è di 4 ms, cioè  $1/(2M/8)$ . Sono necessarie circa 4 ore ( $4\text{ms} \times 2^{32} = 4.7$  ore) per compiere un ciclo completo di valori di ISN
- Dato che MSL è tipicamente minore di 4.7 ore, si può ragionevolmente assumere che il valore di ISN sia unico
  - anche a velocità di 100Mbit/s, il tempo di ciclo diventa 5,4 min, cioè ancora maggiore dei valori del MSL che è dell'ordine di decine di sec (120sec)



# Instaurazione di una connessione

E' necessario evitare che i numeri di sequenza vengano usati (cioè assegnati a nuovi segmenti) per un tempo  $T$  dopo il loro potenziale utilizzo come ISN

- Riassumendo: ogni segmento occupa uno o più numeri di sequenza nello spazio da 1 a  $2^{32}$ ; i numeri usati da un segmento sono considerati “occupati” finché passano MSL sec (quite time) dopo un crash per accertarsi che quei segmenti siano scomparsi dalla rete



# Instaurazione di una connessione

- Un meccanismo di tipo three way handshake è necessario perché i numeri di sequenza non sono collegati a un clock globale nella rete
- Le due entità TCP interagenti si sincronizzano scambiandosi il proprio numero di sequenza iniziale (ISN), che rappresenta il numero a partire dal quale tutti i byte trasmessi, una volta instaurata la connessione, saranno sequenzialmente numerati





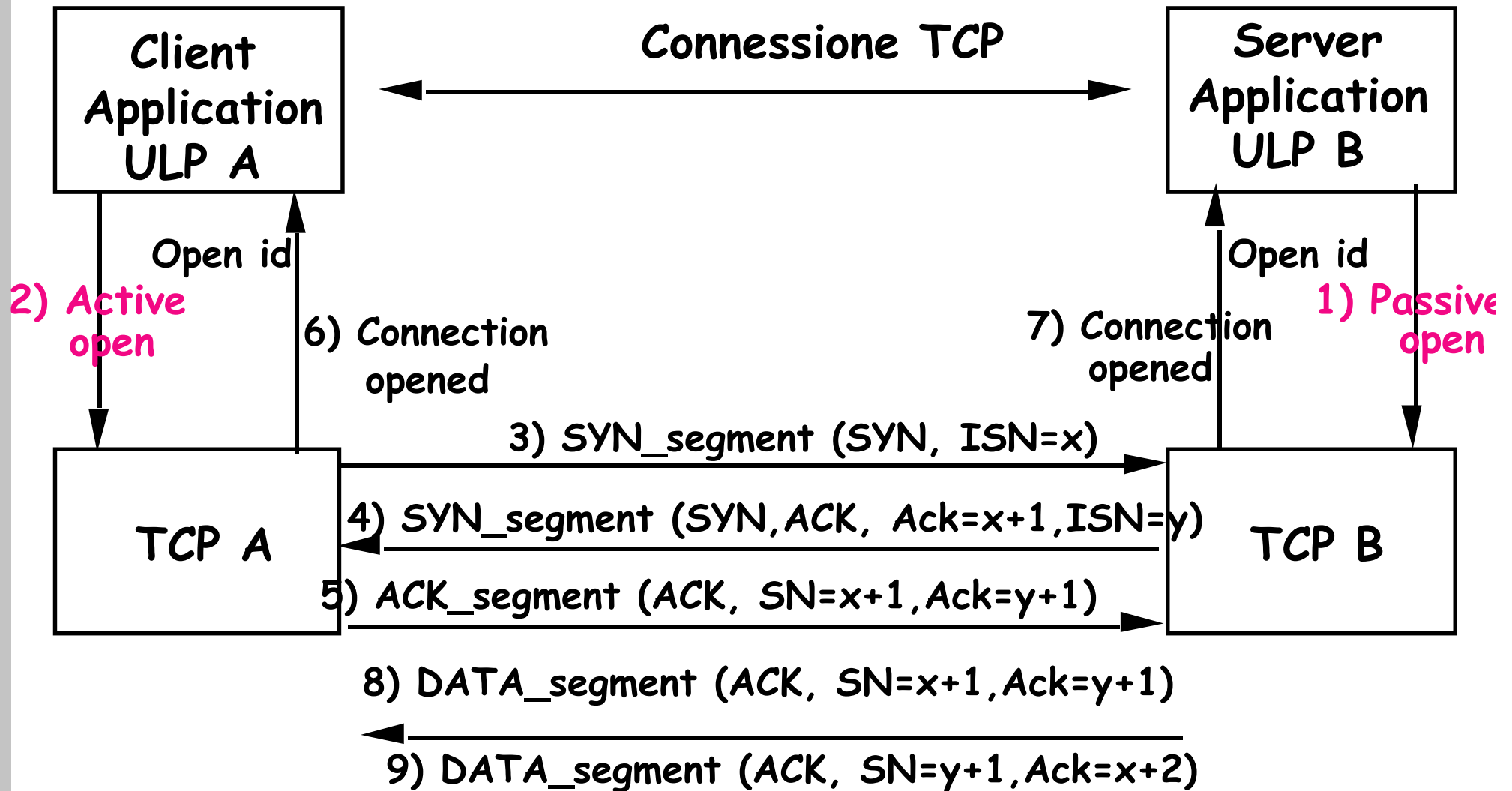
# Instaurazione di una connessione

- La sincronizzazione consiste nell'invio, da parte di ogni sistema coinvolto nella connessione, del proprio ISN e nella ricezione di una conferma dall'altra parte in un messaggio di acknowledgment:
  - 1) A --> B SYN (Synchronize) my sequence number is X
  - 2) A <-- B ACK your sequence number is X
  - 3) A <-- B SYN (Synchronize) my sequence number is Y
  - 4) A --> B ACK your sequence number is Y
- I passi 2 e 3 sono combinati in un unico messaggio, perciò la procedura è nota come three-way (o three-message) handshake



# Three-way handshake: instaurazione

ULP=Upper Layer Protocol





# Instaurazione di una connessione

- 1-2: Affinché dal lato server di un servizio applicativo vi sia un processo in ascolto su una porta deve avvenire l'attivazione del TCP mediante una primitiva detta di **PASSIVE\_OPEN**, che serve a predisporre il server alla ricezione di richieste di connessione; dal lato client, quando si vuole effettivamente aprire una connessione deve essere passata al TCP locale una primitiva di **ACTIVE\_OPEN**, che contiene il socket di destinazione
3. La prima trama inviata dal TCP del lato client è una trama di **SYN** (synchronize) caratterizzata dal bit di **SYN** posto a 1. Il numero di sequenza contenuto nel campo **ISN** è il numero iniziale scelto dal TCP del client per la nuova connessione



# Instaurazione di una connessione

4. In risposta, il TCP del server nell'host destinatario invia una trama di SYN-ACK con i bit di SYN e ACK posti a 1. ISN contiene il numero di sequenza iniziale scelto dal TCP del server per la nuova connessione dal valore del suo contatore. Il campo Ack Number contiene il riscontro della corretta ricezione della trama precedente (valore di ISN ricevuto più 1)
5. L'apertura della connessione viene completata con l'invio di un segmento vuoto con l'ACK finale contenente il riscontro del TCP client della corretta ricezione della trama di SYN-ACK del TCP server; con il numero di sequenza riscontrato pari al numero di sequenza in trasmissione dell'host destinatario + 1



# Instaurazione di una connessione

- 6, 8. Quindi il TCP sender comincia a trasmettere dati, dopo aver inviato la primitiva di Connection Opened al livello applicativo. Si noti che il SN dei messaggi 5 e 8 è lo stesso perché l'invio degli ACK non contribuisce a incrementare il valore di SN
- 7, 9. L'host destinatario comincia a trasmettere solo dopo aver ricevuto quest'ultimo terzo segmento e aver inviato al proprio livello applicativo la primitiva di Connection Opened



# Maximum Segment Size

- Quando il TCP trasmittente invia il primo segmento (SYN) per instaurare una connessione, può inserire nel segmento un'informazione sulla massima dimensione del campo dati di utente di un segmento (MSS) che è in grado di trattare
- Il TCP ricevente risponde comunicando la propria MSS; con questo scambio di informazioni le due entità TCP stabiliscono il valore di MSS comune (il minore tra i due)



# Maximum Segment Size

- La scelta della MSS dipende da due fattori:
  - ✗ la dimensione della memoria a disposizione delle entità TCP
  - ✗ la dimensione della Maximum Transfer Unit (MTU) della sottorete a cui è connesso l'end-system; l'MTU è resa nota all'entità IP e TCP dal software che interfaccia TCP/IP alla sottorete (detto driver di rete)
- Il calcolo dell'MSS avviene sottraendo all'MTU la dimensione degli header IP e TCP (in genere le opzioni non si usano e gli header hanno dimensione fissa di 20 byte)



# Maximum Segment Size

- Se il TCP trasmittente non specifica nel primo segmento (SYN) il valore di MSS, si usa il valore di default di MSS pari a 536 byte
- Un valore di MSS di 536 byte, sommata ai 40 byte di header IP + TCP, dà luogo a un datagramma IP di dimensione pari a 576 byte, ovvero un datagramma che tutti i sistemi di Internet devono necessariamente accettare e gestire
- Oggi è diffusa la tendenza a usare valori molto più grandi dell'MSS per aumentare l'efficienza e la velocità del collegamento (con IPv6 si arriva ad una MTU di 1280 byte)





# Rilascio di una connessione

- Il rilascio di una connessione avviene mediante un meccanismo analogo di three-way handshake modificato
- Una connessione TCP è bi-direzionale e può essere vista come due flussi di dati indipendenti per ciascuna direzione; nel rilascio, le 2 vie sono chiuse indipendentemente
- Quando un programma applicativo non ha più dati da trasmettere ordina al TCP di chiudere la connessione “in una direzione”. Il TCP finisce di trasmettere gli eventuali dati rimasti nel buffer e ne aspetta il riscontro, quindi invia un messaggio di FIN (bit di FIN settato a 1) al TCP ricevente



# Rilascio di una connessione

- Il TCP ricevente invia un messaggio ACK di avvenuta ricezione al TCP trasmittente e poi avvisa il suo livello applicativo che non ha più dati da trasferirgli (questa fase può richiedere tempo per l'interazione umana)
- Intanto i dati possono continuare a fluire nella direzione ancora aperta della connessione, e i riscontri devono continuare a essere inviati dal TCP che aveva chiesto la chiusura della connessione. Questo finché anche l'altra direzione della connessione verrà chiusa
- Normalmente per il rilascio di una connessione servono 4 segmenti: 2 FIN e 2 ACK, però può succedere che il primo ACK e il secondo FIN siano contenuti nello stesso segmento e il numero totale di segmenti necessari si riduce così a 3



# Rilascio di una connessione

- La chiusura di una connessione avviene attraverso i seguenti passi:

1) A -----> B    FIN   (ho finito, non ho più dati)

2) A <----- B    ACK   (OK)

3) A <----- B    FIN   (ho finito anch'io)

4) A -----> B    ACK   (OK)

- oppure

1) A -----> B    FIN   (ho finito, non ho più dati)

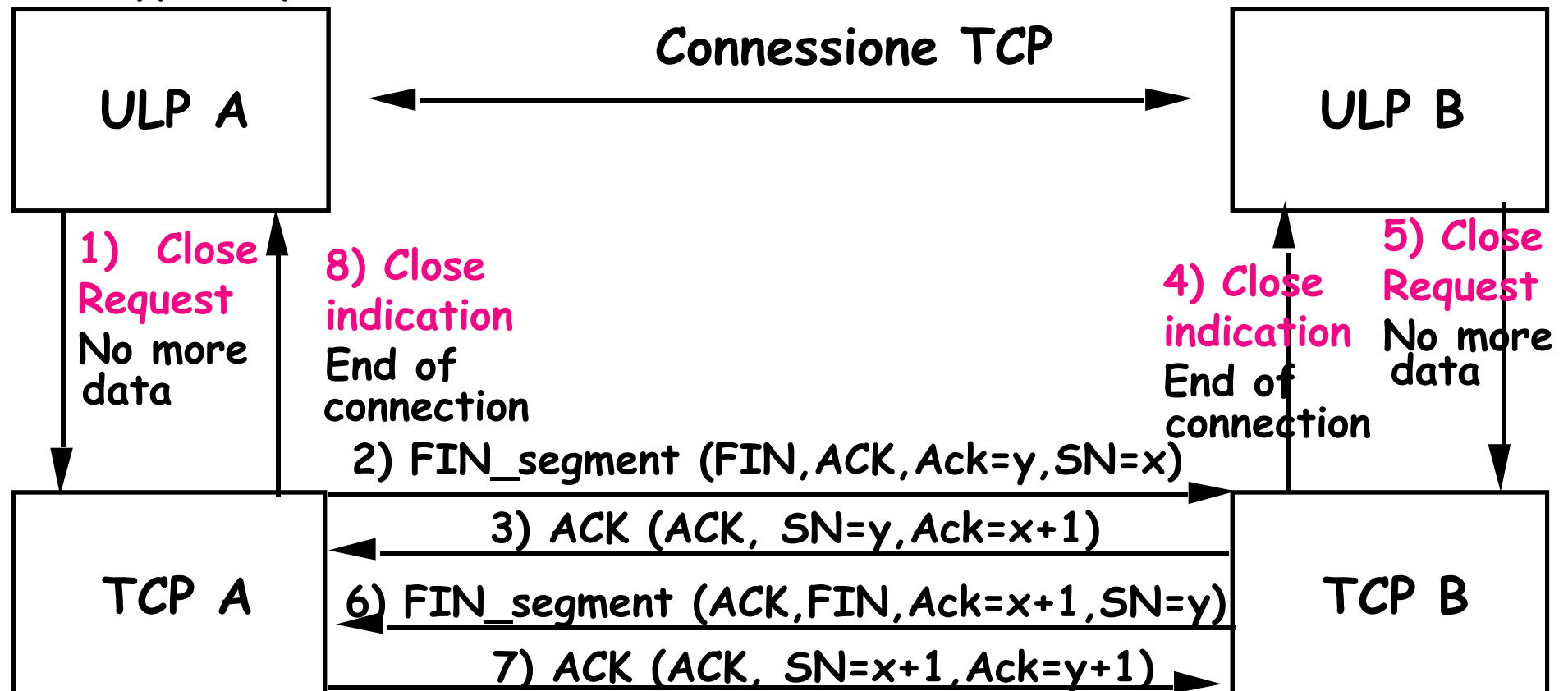
2) A <----- B    ACK   (ho ancora dei dati da trasmettere)

3) A <----- B    FIN   (ho finito anch'io)

4) A -----> B    ACK   (OK)

# Three-way handshake: rilascio

ULP=Upper Layer Protocol





# Rilascio di una connessione

- Il rilascio di una connessione è più complesso della sua instaurazione; infatti informare un'applicazione che è pervenuta una richiesta di chiudere una direzione della connessione ed ottenere una risposta potrebbe richiedere un tempo considerevole (se implica, ad es., un'interazione umana)
- Il meccanismo usato non garantisce la corretta chiusura in ogni caso e il TCP rimedia con dei timeout (pari a 2 MSL). Se il pacchetto di FIN non è seguito da un ACK, dopo un tempo sufficiente la connessione full duplex viene chiusa. L'altra parte può non accorgersi della chiusura immediatamente ma prima o poi si renderà conto che nessuno risponde e a sua volta chiude.



# Rilascio di una connessione

- Se una connessione non può essere chiusa secondo la procedura normale a causa di situazioni anomale, o se un programma applicativo è forzato a chiudere immediatamente una connessione, TCP prevede una procedura di reset
- Tale procedura consiste nell'inviare un segmento con il bit RST "settato". Alla ricezione di tale segmento la connessione è immediatamente terminata senza scambio di ulteriori messaggi e TCP ne informa i programmi applicativi

# Trasferimento dati



- La strategia utilizzata dal TCP per il trasferimento dati usa riscontri solo positivi (PAR, Positive Acknowledge with Retransmission) ed è basata sull'uso di finestre in trasmissione ed in ricezione
  - ✗ rispetto a meccanismi come Stop & Wait che prevedono un riscontro per ogni unità dati, i meccanismi a finestra permettono di incrementare la portata del collegamento
- TCP vede il flusso di dati in trasmissione come una sequenza di byte e quindi la finestra in trasmissione opera a livello di byte, invece che a livello di trama o pacchetto come in X.25

# Trasferimento dati



- La finestra in trasmissione specifica il numero di byte che possono essere inviati senza ricevere un riscontro
- Analogamente, la finestra in ricezione specifica il numero di byte che possono essere accettati fuori sequenza
- In TCP, la dimensione della finestra in ricezione coincide con quella in trasmissione; quindi la dimensione di finestra specifica sia il numero di byte che un'entità TCP può trasmettere senza ricevere riscontri, sia il numero di byte che un'entità TCP può ricevere fuori sequenza



# Trasferimento dati



- La dimensione della finestra di trasmissione non è decisa dal mittente, ma è comunicata al mittente dal destinatario (nel campo Window di 16 bit) “ogni volta” che questo emette un segmento; quindi la dimensione della finestra varia dinamicamente nel tempo
- Per questo motivo la finestra è chiamata in TCP Advertised Window (finestra “pubblicizzata” dal destinatario al mittente)

# Trasferimento dati



- All'inizio di una connessione, quando il mittente non ha ancora trasmesso dei dati, emette un numero di byte pari alla dimensione  $W$  della finestra; quindi si interrompe in attesa di un riscontro
- Il TCP prevede esclusivamente riscontri (ACK) positivi
- I riscontri possono essere “cumulativi”, cioè il destinatario conferma la ricezione dell'ultimo byte di una sequenza di dati ricevuta “completamente” in modo corretto, ovvero senza errori e senza elementi mancanti

# Trasferimento dati



- In particolare, il destinatario comunica al mittente il “prossimo” byte che si aspetta di ricevere (nel campo Acknowledgement Number), significando così che “tutti” i byte precedenti sono stati ricevuti correttamente, e la nuova dimensione della finestra  $W$
- Alla ricezione di un riscontro (con Ack Number pari a  $x$ ) il mittente sposta in avanti la finestra ed è autorizzato a inviare, senza attesa di riscontro, i byte con numero di sequenza compreso tra  $x$  e  $W+x$

# Trasferimento dati



- Il sender tiene traccia del successivo numero di sequenza da usare nella variabile **SND.NXT** e del più vecchio numero di sequenza non riscontrato in **SND.UNA**; il receiver tiene traccia del successivo numero di sequenza da accettare nella variabile **RCV.NXT**
- Quando il sender crea un segmento e lo trasmette, fa avanzare **SND.NXT**; quando il ricevitore accetta un segmento fa avanzare **RCV.NXT** e invia un riscontro; quando il sender riceve un riscontro, fa avanzare **SND.UNA**
- Se il flusso di dati è momentaneamente “idle” e tutti i dati inviati sono stati riscontrati le 3 variabili sono uguali

# Trasferimento dati



- La differenza tra i valori di queste variabili è una misura del ritardo nella comunicazione; la quantità di cui le variabili sono fatte avanzare è la lunghezza dei dati nel segmento
- Si noti che una volta che la connessione è stabilita tutti i segmenti devono trasportare informazioni di riscontro

# Controllo di flusso



- Il controllo di flusso è una procedura attuata in modo coordinato tra due entità TCP, sorgente e destinatario, intesa a limitare il flusso dei dati trasmessi, in funzione delle risorse a disposizione “nei sistemi terminali” e prescindendo dal traffico presente in rete
  - ✗ tale meccanismo è indispensabile in Internet dove calcolatori di dimensione e capacità di calcolo diverse comunicano tra loro; il più lento dei due deve poter rallentare l’emissione di informazione dell’altro
- Lo scopo del controllo di flusso è di evitare che un mittente invii dei segmenti ad un destinatario che, “in quel momento”, non è in grado di riceverli, a causa di indisponibilità di risorse sia di elaborazione che di memorizzazione

# Controllo di flusso



- Il controllo di flusso nel TCP è implementato mediante lo stesso meccanismo usato per il controllo di errore, “a finestra scorrevole” (sliding window) di tipo “credit allocation”
  - ✗ l’ampiezza della finestra è variabile
  - ✗ il meccanismo è “orientato al byte”: la finestra rappresenta, istante per istante, il numero massimo di byte che possono essere trasmessi verso il destinatario
- Lo schema è basato sul campo Window (16 bit) in cui il destinatario scrive la larghezza (in byte) della finestra di trasmissione  $W$  che il mittente dovrà usare dal quel momento in poi
- Il mittente userà tale valore  $W$  finché non riceve dal destinatario un successivo segmento con una dimensione di finestra diversa

# Controllo di flusso

- Un riscontro ( $\text{ACK Number}=x$  e  $\text{Window}=W$ ) significa che:
  - ✗ sono riscontrati tutti gli ottetti ricevuti fino a quello numerato con  $X-1$
  - ✗ il trasmittente è autorizzato a trasmettere, senza attendere altri riscontri, fino a ulteriori  $W$  ottetti, ovvero fino all'ottetto numerato con  $x+W-1$
- Solitamente la  $RW$  ha un'ampiezza pari a  $n \cdot \text{MSS}$ , dove  $n$  è un numero intero per evitare che il TCP trasmittente esegua una segmentazione poco efficiente
  - se la finestra di ricezione è pari a 301 byte e la MSS 100 byte, il TCP trasmittente sarebbe indotto a formare segmenti da 1 byte, a cui aggiungere l'header del TCP e di IP (tipicamente di 20 byte ciascuno). Se a ciò si aggiunge l'header di livello 2 (8 byte nel caso di PPP), per inviare un'informazione di 1 byte è necessario inviarne 49



# Controllo di flusso



- La finestra comunicata dal ricevitore, Advertised Window, rappresenta dunque anche la quantità di dati che l'entità destinataria è disposta a ricevere (funzione della capacità di calcolo e della memoria in ricezione)
- Il ricevitore può variare “dinamicamente” la dimensione della finestra in funzione delle sue esigenze e limitare il ritmo di trasmissione qualora non sia in grado di gestirlo
- Quindi il ricevitore informa il trasmettitore, avendo presente però che quest'ultimo la modificherà solo dopo aver ricevuto dati, corretti ed in sequenza, che abbiano "riempito" le finestre precedentemente offerte



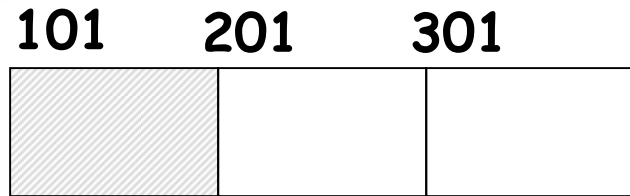
# Controllo di flusso: esempio 1

## Esempio

- Si abbia una MSS di 100 byte e una finestra  $RW=3MSS=300$  byte. In un certo istante, il ricevitore accetta i byte da 101 a 400.
- Il trasmettitore invia i byte dal 101 al 200 che non vengono mandati subito all'applicazione (magari perché il TCP è uno dei tanti processi concorrenti che gira su un PC o su una workstation, tra i quali il S.O. cicla); allora la RW effettiva si è ridotta a 200 byte.
- Il trasmettitore invia un altro segmento che contiene i byte dal 201 al 300; il TCP ricevente invia tutti i byte al livello superiore e svuota la sua finestra, che torna alla sua dimensione originale di 300 byte, accettando i byte da 301 a 600.

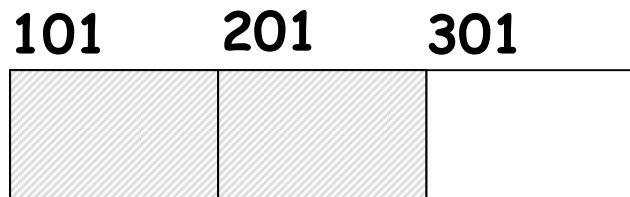
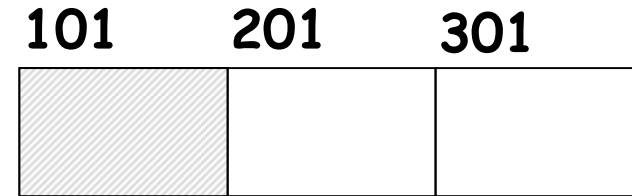


# Controllo di flusso: esempio 1



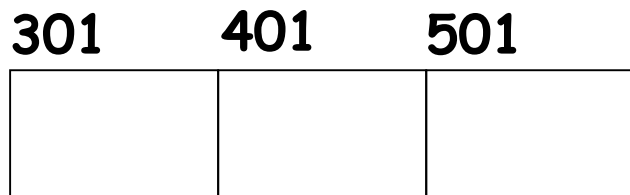
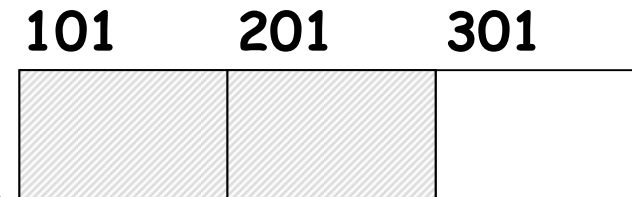
unacked

Data #seq 101



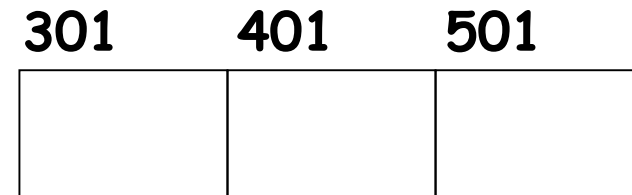
unackedunacked

Data #seq 201



SW del TX

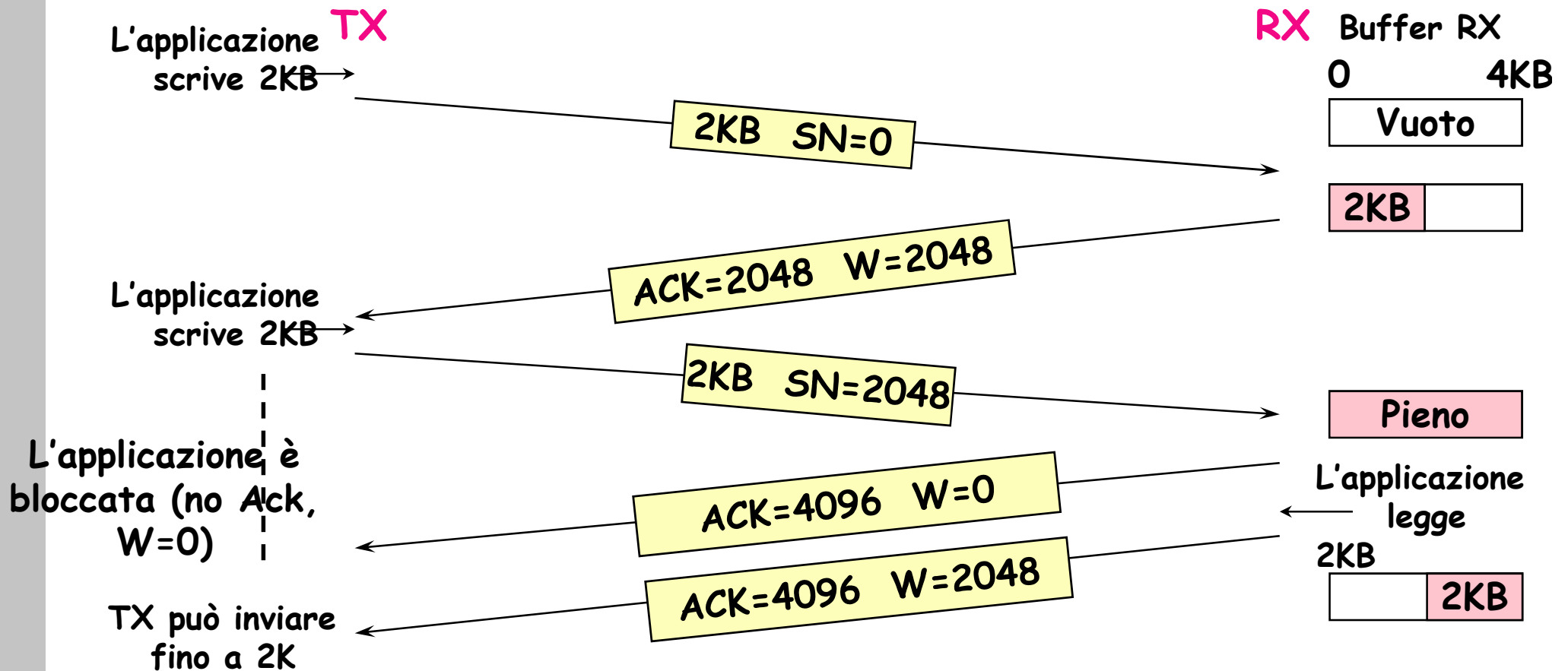
Ack #ack 301



RW del RX



# Controllo di flusso: esempio 2



# Controllo di flusso: esempio 3



Finestra iniziale di 1400 byte

1000	1001	2400	2401
------	------	------	------

1000	1601	2400	2401
------	------	------	------

1000	2001	2400	2401
------	------	------	------

Finestra incrementata di 200 byte

1600	1601	2001	2600	2601
------	------	------	------	------

Finestra esaurita

1600	1601	2600	2601
------	------	------	------

Finestra incrementata di 1400 byte

2600	2601	4000	4001
------	------	------	------

Pronto a ricevere 1400 byte

1000	1001	2400	2401
------	------	------	------

Ricevuti 600 byte  
disponibilità per altri 200 byte

1600	1601	2600	2601
------	------	------	------

Ricevuti ulteriori 400 byte

1600	1601	2001	2600	2601
------	------	------	------	------

Ricevuti ulteriori 600 byte  
disponibilità per altri 1400 byte

2600	2601	4000	4001
------	------	------	------

SN=1001

SN=1201

SN=1401

SN=1601

SN=1801

A=1601  
W=1000

SN=2001

SN=2201

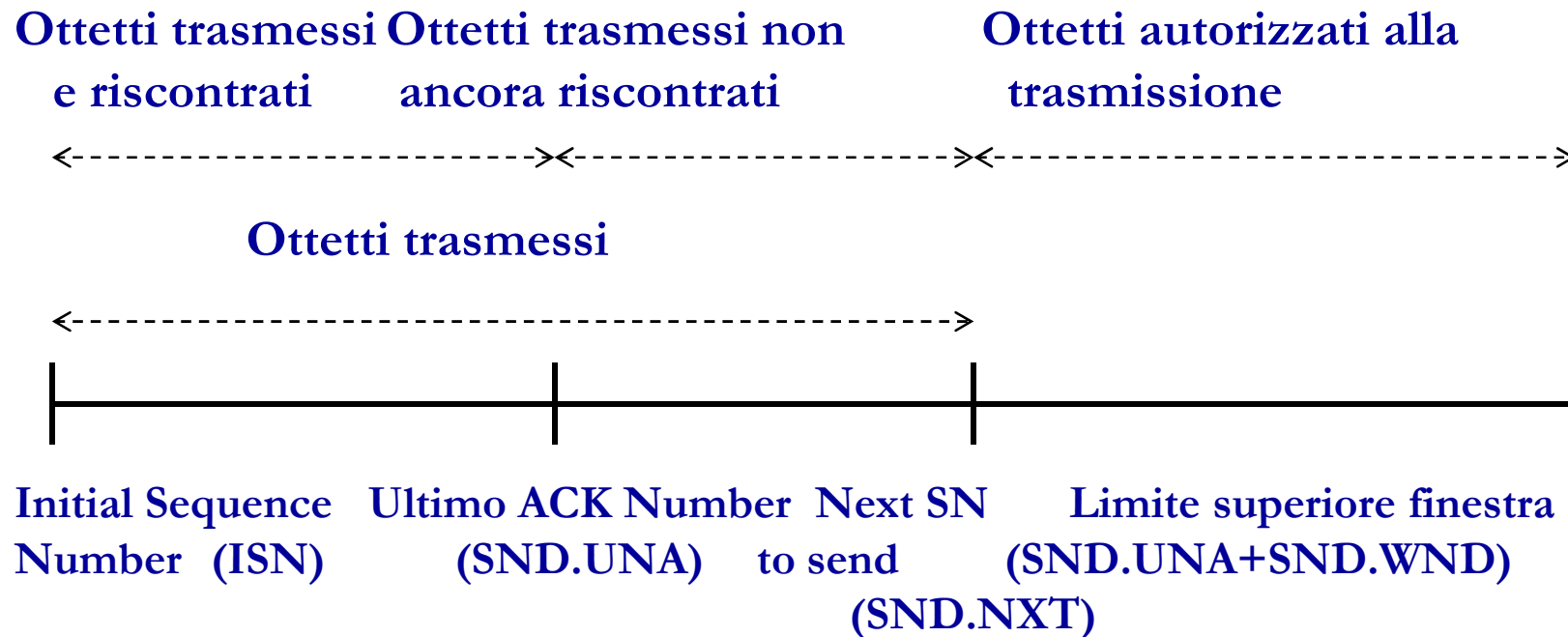
SN=2401

A=2601  
W=1400

# Controllo di flusso



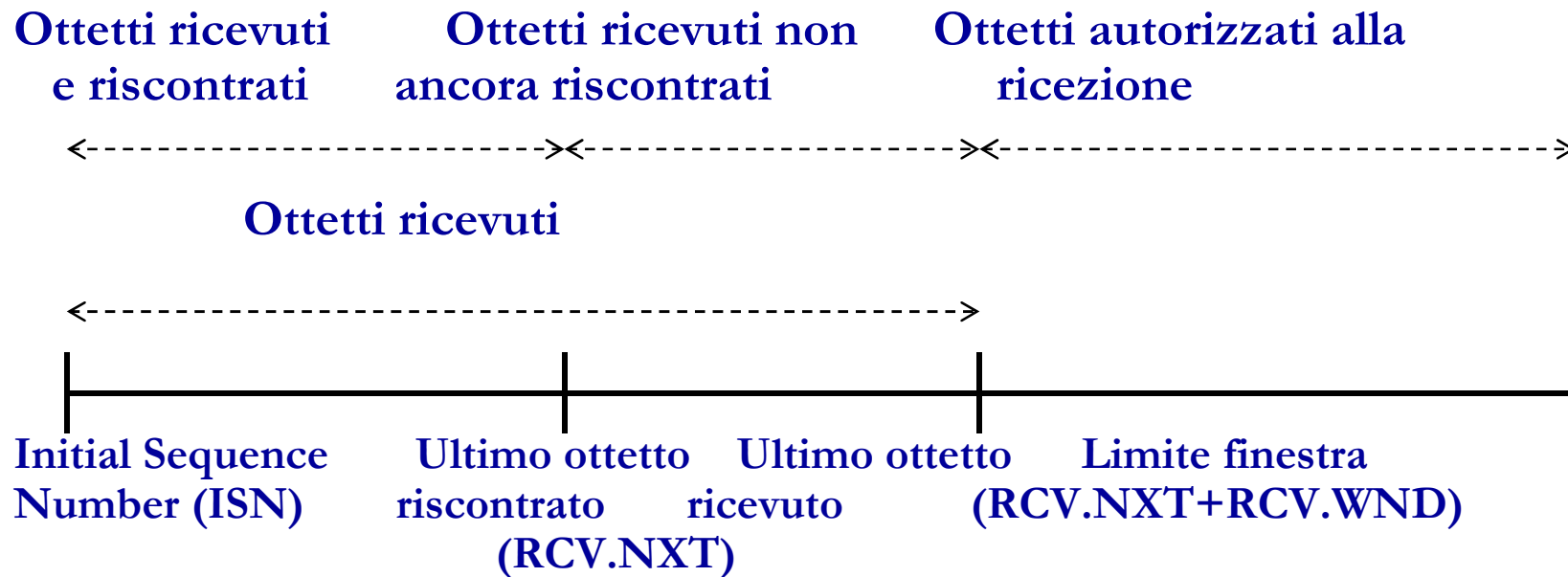
- Puntatori del controllo a finestra lato emittente



# Controllo di flusso



- Puntatori del controllo a finestra lato ricevente





# Sindrome della silly window

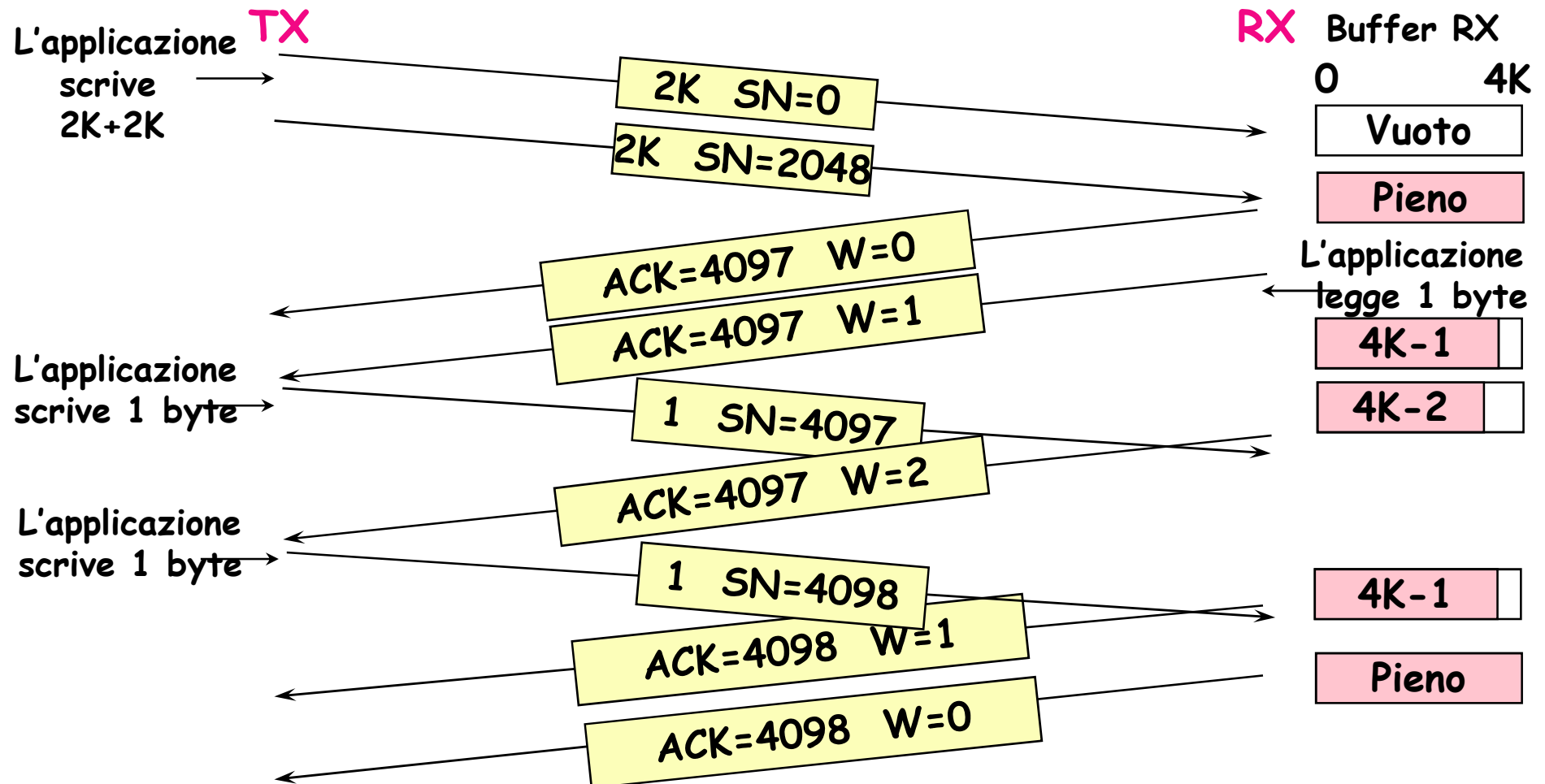
- Nelle prime implementazioni del TCP si era riscontrato un problema, noto come “Silly window syndrome”, che si riscontra quando l'applicazione sorgente passa i dati al suo TCP a grossi blocchi mentre l'applicazione ricevente assorbe i byte lentamente (es. 1 alla volta)
- In questo caso, il ricevitore vede ogni volta 1 byte libero nel buffer e con l'invio dell'ACK e del campo window sollecita la trasmissione di un segmento con un solo byte. Naturalmente, una quantità di segmenti con un solo byte porta ad uno spreco di risorse a causa dell'elevatissimo overhead





# Sindrome della silly window

Esempio:





# Sindrome della silly window

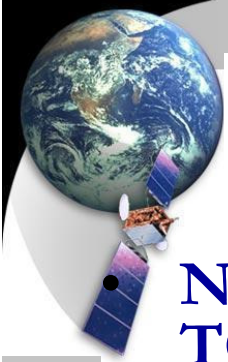
- Per superare il problema della “silly window syndrome” (Clark 1982) si sono introdotti i seguenti meccanismi:
- lato ricevitore: il ricevitore non deve inviare “window update” per piccole variazioni della finestra di ricezione, ovvero: il ricevitore dovrebbe aspettare di avere a disposizione almeno 1 MSS o la metà della finestra di ricezione prima di inviare un window update
  - ✗ quindi il ricevitore mente indicando un buffer pieno fino a che il buffer non si è svuotato per metà o per una quantità almeno pari a 1 MSS
- lato trasmettitore: il sender deve evitare di emettere segmenti troppo piccoli, ovvero tenta di creare segmenti non più piccoli di  $1/2$  MSS (se non sollecitato con primitive a fare il PUSH dei dati)



# Controllo di flusso: throughput

- Il throughput (S) (byte/s) di una connessione TCP dipende da:
  - dimensione della finestra TCP in byte (W)
  - ritardo di propagazione (D) in sec tra sorgente e destinazione TCP
  - bit rate (R) in byte/s della sorgente TCP

$$S = \begin{cases} 1 & \text{se } W > RD \\ W/RD & \text{se } W < RD \end{cases}$$

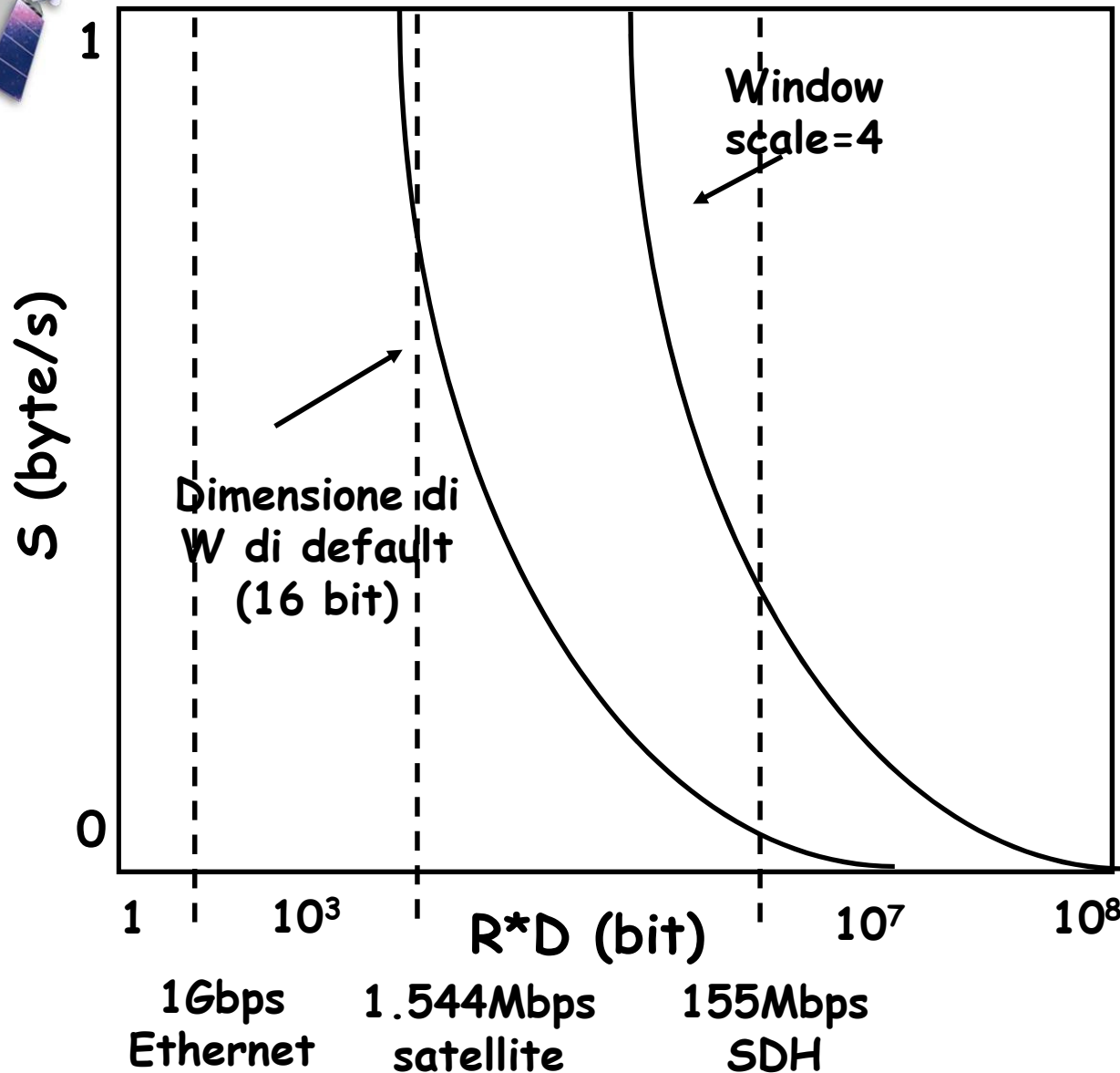


# Controllo di flusso: throughput

• Nel calcolo del throughput ( $S$ ) ignoriamo l'overhead del segmento TCP

- una sorgente TCP emetta una sequenza di byte verso una destinazione
- il primo byte impiegherà un tempo  $D$  per arrivare a destinazione, più un tempo addizionale  $D$  perché il riscontro torni alla sorgente
- durante questo tempo ( $2D$ ) la sorgente potrebbe trasmettere  $2D \cdot R$  byte
- in realtà, però, la sorgente è limitata dalla dimensione  $W$  della finestra TCP finché non riceve almeno un riscontro
- quindi se  $W > RD$  (il riscontro arriva prima che la finestra si chiuda) si ottiene il throughput max (1) sulla connessione, se invece  $W < RD$  (il sender si ferma dopo  $W$  in attesa del riscontro per riaprire la finestra) il max throughput normalizzato è dato dal rapporto  $W$  su  $RD$

# Controllo di flusso: throughput



- La max dimensione di  $W$  è  $2^{16}-1=65535$  byte
- $RD$  nel caso Ethernet da 1Gbps con estensione 100m è  $<10^3$  bit; nel caso di link satellitare a 1.544Mbps è  $<10^6$
- In entrambi i casi la dimensione di finestra di default va bene
- Nel caso link SDH a 155Mbps tra 2 punti distanti,  $RD$  è  $<10^7$  e la dimensione classica di  $W$  non va bene, allora si usa un fattore di scala che permette di allargare la finestra a  $2^{20}-1=10^6$  byte



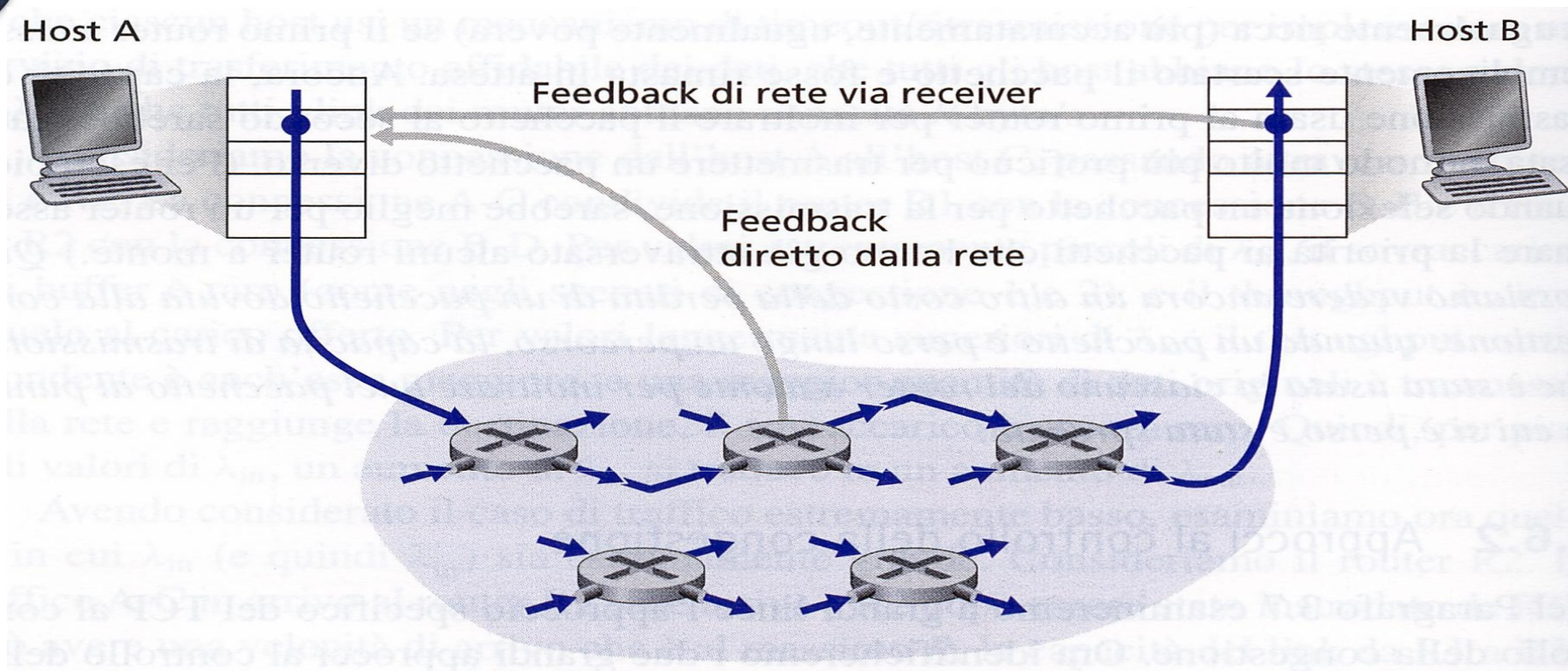
# Controllo di flusso: Window size

- La dimensione della finestra di 64KB può essere un problema per le linee con High Bandwidth o High Delay o High Bandwidth Delay Product (RD)
  - Es. connessioni satellitari
- Con alti ritardi di propagazione e elevate velocità di trasmissione, la finestra si svuota velocemente, ma il sender non può inviare altri dati finché non riceve un riscontro

## Soluzioni:

- RFC 1323 propone un “Window scale option” che permette di negoziare un fattore di scala che estende la dimensione di Window di altri 16 bit (fino a  $2^{32}$  byte)
- RFC 1106 propone l'uso del Selective Repeat al posto del Go-back N; introducendo i NAcK per chiedere la ritrasmissione di un segmento specifico





**Procedura di controllo della congestione**



# Controllo e recupero di errore

La strategia utilizzata dal TCP per il recupero di errore è simile a quella usata in X.25 (LAPB), di tipo Go-back-n con riscontri solo positivi (PAR, Positive Acknowledge with Retransmission), ed è basata sull'uso di finestre in trasmissione ed in ricezione

- Ogni segmento contiene una checksum che il ricevitore usa per verificare l'integrità dei dati. Se il segmento è corretto viene inviato un segmento di acknowledge. Se il segmento non è corretto viene semplicemente scartato, dopo un timeout il segmento verrà ritrasmesso





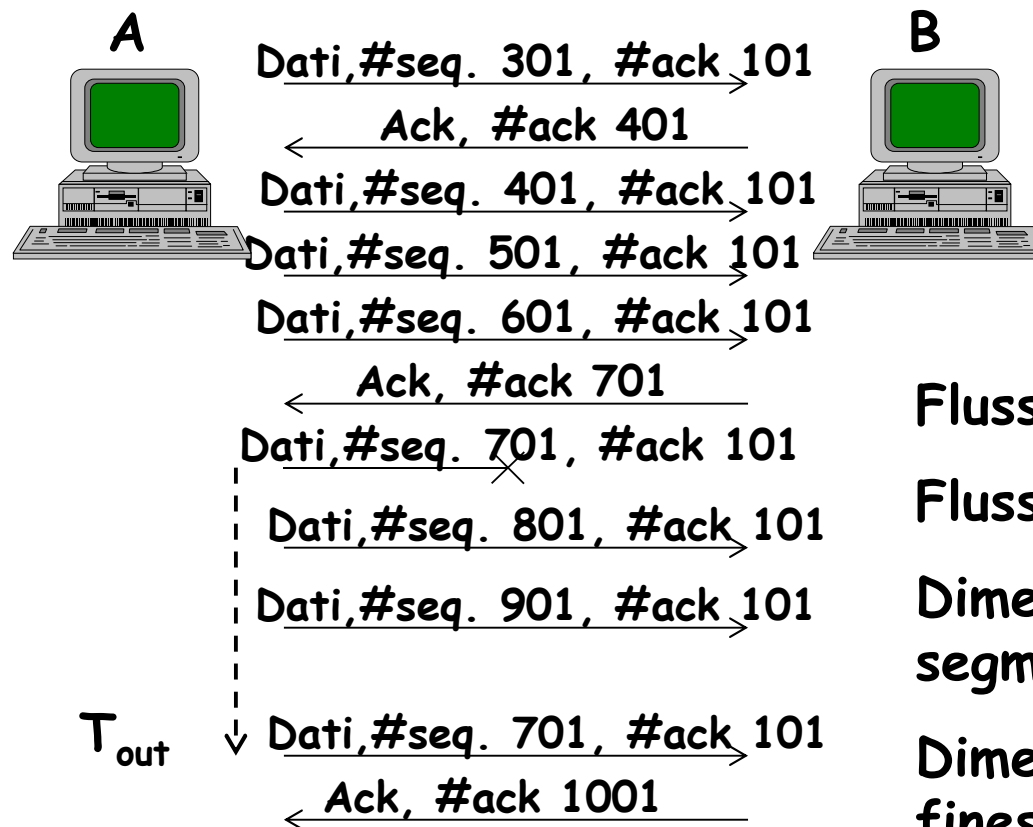
# Controllo e recupero di errore

Il meccanismo di recupero da errore è basato sull'uso di un timeout (fuori tempo massimo); ovvero la ritrasmissione di un segmento è innescata dalla mancata ricezione degli ACK entro un tempo predefinito (RTO=retransmission timeout)

- L'entità mittente, dopo avere inviato un segmento, aspetta un tempo limite prefissato (RTO) e, se non riceve un riscontro, assume che il segmento si sia perso

# Controllo e recupero di errore

- Due eventi provocano la ritrasmissione di un segmento: il segmento si è perso e non arriva a destinazione, il segmento arriva a destinazione non corretto (checksum errata)



Flusso dati: da A a B

Flusso riscontri: da B ad A

Dimensione dei  
segmenti=100 byte

Dimensione della  
finestra=300 byte

# Calcolo del RTO



- Il dimensionamento del timeout (RTO) è un aspetto critico per le prestazioni del TCP.
  - ✗ minore è il timeout, in meno tempo si interrompe la trasmissione. Però se il suo valore è troppo piccolo, i segmenti in ritardo, a causa di congestione, potrebbero essere considerati persi e quindi ri-trasmessi, ciò aumenterebbe la congestione e di conseguenza le ri-trasmissioni finché la portata tende a zero
  - ✗ se il suo valore è troppo grande, la risposta ad un evento di perdita sarebbe troppo lenta con conseguente perdita di efficienza e minore velocità di trasferimento

# Calcolo del RTO



- Il timeout è fissato con uno schema “adattativo” in base al valore stimato del round-trip delay (RTT)
- TCP misura il RTT, cioè il tempo di andata e ritorno tra sorgente e destinazione impiegato da un’unità dati, ovvero il tempo che passa tra l’invio di un segmento e la ricezione del relativo ACK
  - ✕ RTT è somma di 3 contributi: ritardo di trasferimento in un verso della comunicazione; ritardo di trasferimento nell’altro verso; tempo necessario al destinatario per rispondere (tempo di risposta)

# Calcolo del RTO



**In generale, la determinazione dei tempi di trasferimento nel RTT è complessa in Internet:**

- ✕ una connessione può attraversare una LAN ad alta velocità o più sottoreti lente in diversi continenti; perciò il ritardo di trasferimento è molto variabile ed è influenzato dallo stato di congestione momentanea delle sottoreti attraversate
- **Se si potesse effettuare il recupero di errore a strati inferiori al TCP, sottorete per sottorete, si avrebbero stime precise del ritardo di trasferimento, ma in Internet ciò non è possibile, visto che richiederebbe**
  - ✕ la cooperazione tra sottoreti diverse (impossibile)
  - ✕ il controllo di errore effettuato da ogni sistema di interconnessione intermedio (da evitare)

# Calcolo del RTO



- In base al valore di RTT misurato in maniera regolare, il TCP aggiorna “dinamicamente” il valore di RTO; TCP potrebbe scegliere un valore di RTO maggiore o uguale al “valore medio” osservato di RTT
- Tuttavia, la misura del RTT basata sulla “media” delle osservazioni può essere affetta da errori:
  - ✗ l'emissione degli ACK da parte del TCP ricevente può essere non immediata (riscontri cumulativi non riferiti a uno specifico datagramma)
  - ✗ se è stata effettuata una ritrasmissione è impossibile distinguere se l'ACK si riferisce alla trasmissione iniziale o alla ritrasmissione (l'RTT si calcola dal datagramma originale o da quello ritrasmesso?)
  - ✗ lo stato di congestione della rete può cambiare molto rapidamente

# Calcolo del RTT



- Soluzione: si usa una stima del RTT che fornisce una media pesata del RTT (media esponenziale), denominata Smoothed Round Trip Estimate (SRTT) (RFC793)
- L'entità TCP mittente inizializza un temporizzatore (timer) nell'istante in cui inizia la trasmissione di un segmento e ne memorizza l'istante di partenza in una memoria destinata a contenere le informazioni di gestione della connessione
- Alla ricezione di un riscontro valido (che include il numero di sequenza del segmento trasmesso), il mittente calcola il tempo trascorso, cioè l'RTT corrente, e aggiorna il valore stimato di SRTT

# Calcolo del RTT



- L'SRTT privilegia i valori dei campioni di RTT più recenti:

$$\text{SRTT}(k+1) = a \text{SRTT}(k) + (1-a) \text{RTT}(k+1)$$

$0 \leq a \leq 1$ ; tanto più il valore del peso (smoothing factor)  $a$  è vicino a 0, tanto maggiore sarà il peso dato all'ultima osservazione di RTT (normalmente  $0.8 \leq a \leq 0.9$ )

- Valori minori del peso corrispondono ad un aggiornamento veloce del SRTT; valori maggiori rendono il SRTT insensibile a brevi variazioni del ritardo di trasferimento