# CORSO DI RETI
# SSIS

Lezione n.3

9 novembre 2005

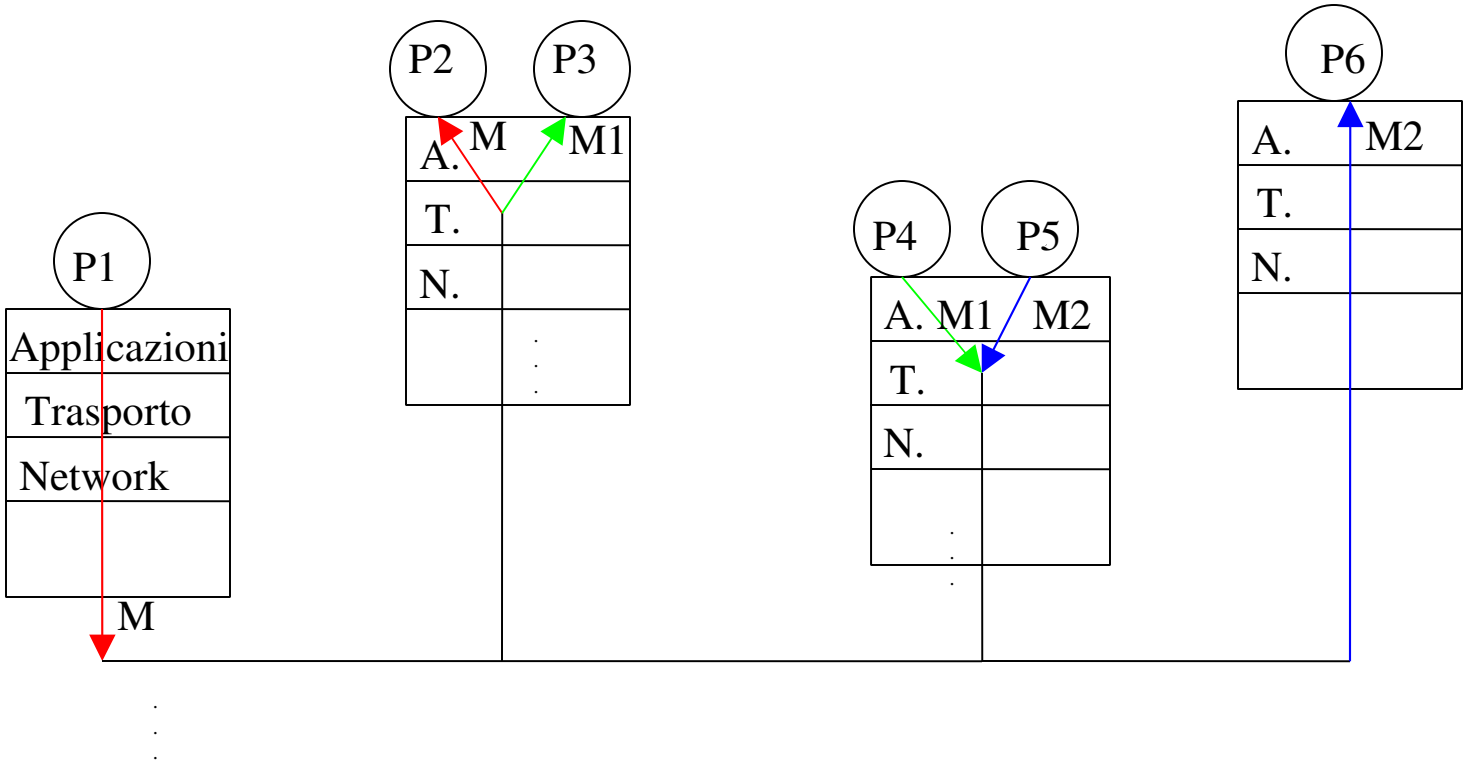
*Laura Ricci*

# IL LIVELLO TRASPORTO

- realizza un supporto per la *comunicazione logica* tra *processi distribuiti*
- *comunicazione logica* = astrazione che consente a una coppia di processi di scambiarsi messaggi, anche quando essi non risiedono su hosts direttamente connessi a livello fisico
- sfrutta i servizi dei livelli inferiori di TCP-IP (livello rete, livello link,…) per la *trasmissione dei messaggi* tra *hosts remoti*
- definisce servizi di multiplexing/demultiplexing e trasmissione sicura dei messaggi, controllo della congestione della rete
- può definire più protocolli. Internet: protocollo UDP + protocollo TCP

# IL LIVELLO TRASPORTO

*multiplexing/demultiplexing*: instradamento dei messaggi da/ verso i processi.
(minimo servizio offerto dal questo livello)



# IL LIVELLO TRASPORTO

- processi individuati da *porte* = numeri interi compresi tra 0 e 65535

- alcuni servizi associati a *porte note*
  - web porta 80
  - ftp porta 22 ……

- *multiplexing* = raccoglie i dati provenienti da processi diversi in esecuzione sullo stesso host, crea i segmenti (crea il TCP header) e li passa al livello network

- *demultiplexing* =
  - raccoglie i segmenti provenienti dal livello IP,
  - riconosce tramite le informazioni contenute nell'header il processo destinatario P,
  - instrada il segmento verso P

# IL LIVELLO TRASPORTO: PROTOCOLLO UDP

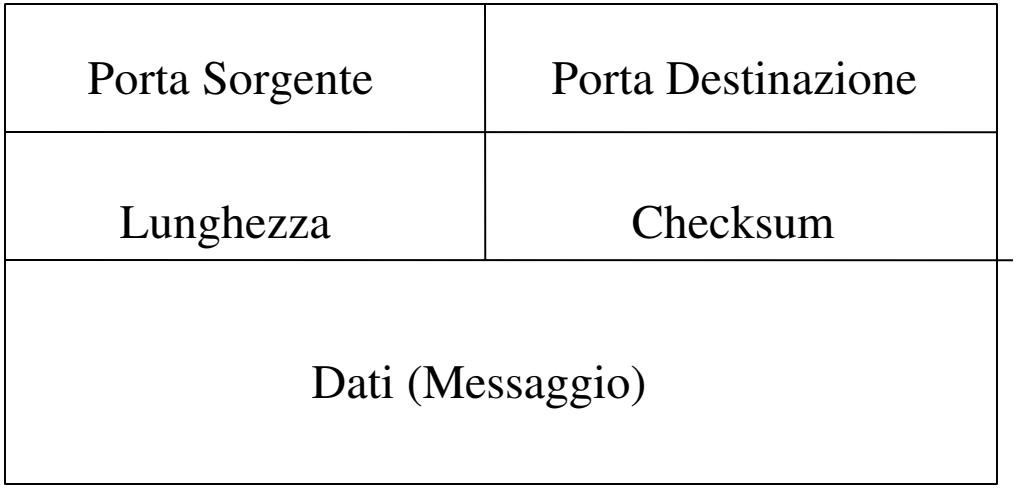
- UDP (User Datagram Protocol) : no frills protocol
- offre funzionalità  minima di multiplexing/demultiplexing + semplice controllo degli errori
- *svantaggi:* servizi minimi
- *vantaggi:* efficienza
  non è richiesto di stabilire alcuna connessione
  risparmio strutture dati necessarie per mantenere lo stato della trasmissione
  pacchetti piccoli (header più piccolo)
  nessun vincolo sulla velocità di trasmissione

# UDP: USER DATAGRAM PROTOCOL

*Struttura del segmento UDP*

*Lunghezza* = Lunghezza in byte del segmento UDP (compreso l'header)

*Checksum* = controllo degli errori



# UDP : CHECKSUM

*Mittente*

- calcola la somma di tutte le parole contenute nel messaggio
- calcola il complemento ad 1 della somma
- invia il risultato nel segmento UDP

0110011001100110 +
0101010101010101 +
0000111100001111
1100101011001010

*Complemento ad 1*
*= checksum* 0011010100110101

# UDP: CHECKSUM

*Destinatario*

- calcola la somma di tutte le parole contenute nel segmento + la checksum

  1100101011001010 +
  0011010100110101
  1111111111111111

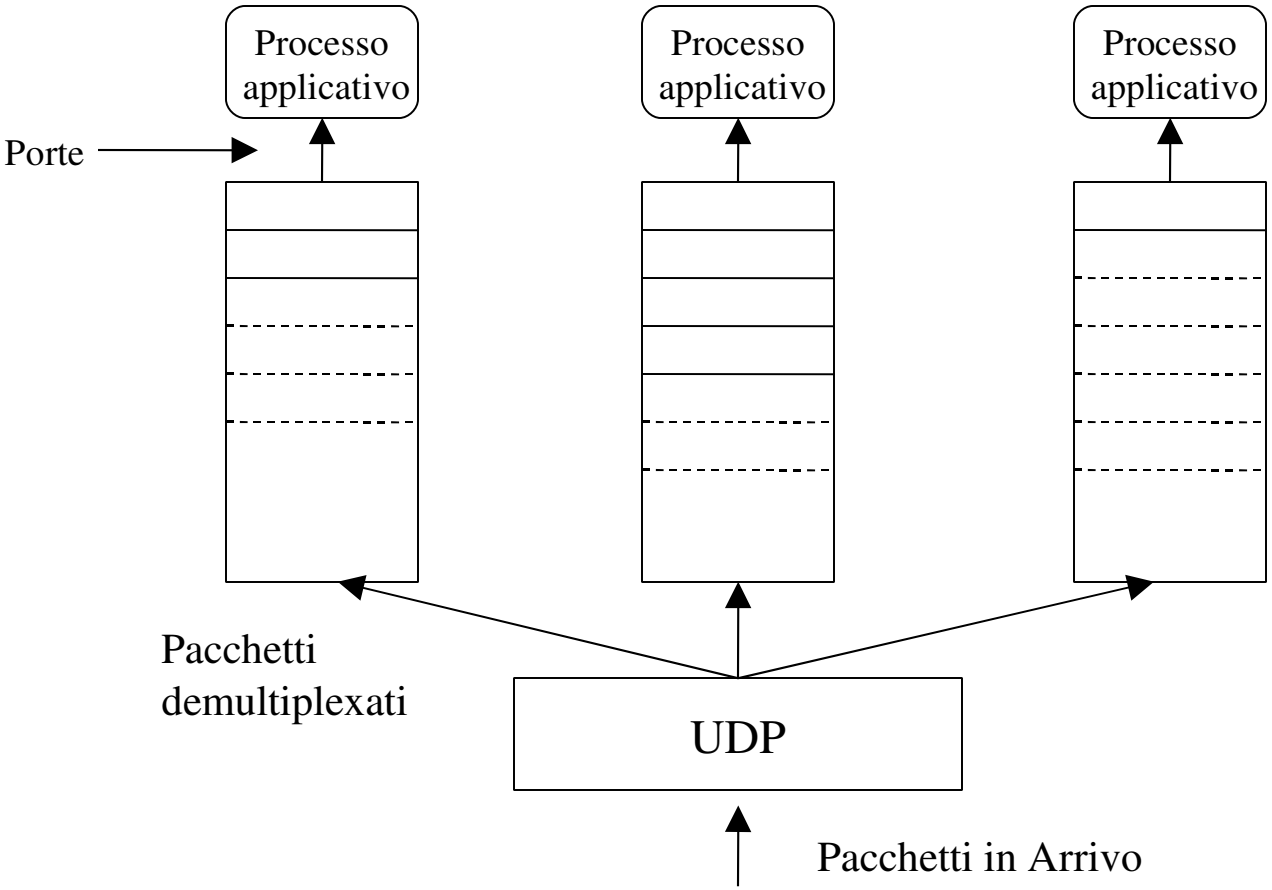
- se non si sono verificati errori in trasmissione il risultato deve essere uguale ad una sequenza di 1.

- Se si e' verificato un errore = il segmento viene scartato

Controllo degli errori = Effettuto a livello link+ livello TCP

Ridondanza dei controlli motivata dal fatto che alcuni links non effettuano controlli

# UDP: DEMULTIPLEXING DEI MESSAGGI



# UDP: DEMULTIPLEXING DEI MESSAGGI

- ad ogni porta viene associata una *coda di messaggi*
- UDP accoda i messaggi in arrivo al termine della coda opportuna
- se la coda è piena
  - il messaggio viene ignorato (eliminato)
  - non esiste alcun meccanismo di *controllo del flusso* che consenta di rallentare il flusso dei messaggi del mittente quando la coda si riempie
- quando un processo ( a livello applicazione) vuole ricevere un messaggio, ne viene prelevato uno dalla testa della coda. Se la coda è vuota il processo si blocca

# PROTOCOLLO CONNECTION ORIENTED : TRASMISSION CONTROL PROTOCOL (TCP)

Servizi offerti dal livello TCP

- multiplexing/demultiplexing dei messaggi (come UDP)
- controllo degli errori (checksum)
- trasmissione *connection oriented* che garantisce
  - consegna affidabile dei segmenti
  - mantenimento dell'ordine dei messaggi spediti
- Controllo del *flusso*
- Controllo della *congestione*

# PROTOCOLLO CONNECTION ORIENTED : TRASMISSION CONTROL PROTOCOL (TCP)

- controllo del flusso e controllo della congestione: entrambi consentono di controllare la velocità di spedizione dei segmenti da parte del mittente, con obiettivi diversi

- *controllo del flusso:*

  evita di *sovraccaricare il buffer di ricezione* del destinatario. (controllo end to end)

- *controllo della congestione*:

  evita di di sovraccaricare i routers o le linee della rete (controllo della rete)

# TCP : TRASMISSION CONTROL PROTOCOL

*Connection Oriented Protocol* =i processi comunicanti, prima di iniziare la comunicazione vera e propria eseguono *un handshake.*

*Handshake*= scambio di un insieme di segmenti di controllo. Consente ai partner della comunicazione di *inizializzare un insieme di variabili* per il controllare la trasmissione.

Connessione TCP = l'informazione per il supporto della connessione è memorizzata esclusivamente negli end systems.

Circuiti virtuali = supporto a livello di end systems + routes

# TCP : TRASMISSION CONTROL PROTOCOL

Caratteristiche del protocollo TCP

- *punto a punto*:

  è possibile la trasmissione tra una sola coppia di host A,B (unicast). Multicast supportato solo da UDP

- *full duplex*

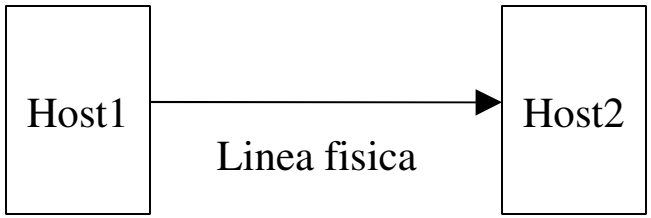
  una connessione tra una coppia H1, H2 di host supporta l'invio dei messaggi in entrambe le direzioni (da H1 ad H" e viceversa)

# TRASMISSIONE AFFIDABILE: LIVELLO LINK

- gli algoritmi classici per la trasmissione affidabile sono stati definiti a livello link

- studiamo il problema della trasmissione affidabile a livello link e poi estendiamo la soluzione per il livello trasporto

- Problema a livello link: dati 2 host connessi da una *linea fisica* (cavo o fibra) inaffidabile




come rendere affidabile il collegamento?

# TRASMISSIONE AFFIDABILE: LIVELLO LINK

meccanismi per la *trasmissione affidabile*

- uso di codici per la rilevazione e la correzione di errori
- se l'errore non può essere corretto, alcuni frames vengono comunque scartati dal destinatario

  $\Rightarrow$

  necessari ulteriori meccanismi per rendere la trasmissione affidabile basati su
  - uso di *acknowledgments* (ACK, conferme) per segnalare la corretta ricezione di un frame
  - ritrasmissione di frames in mancanza di ricezione di ACK dopo un certo intervallo di tempo (*timeout*)

# TRASMISSIONE AFFIDABILE: LIVELLO LINK

Approcci basati su ack + timeouts

- *stop and wait*

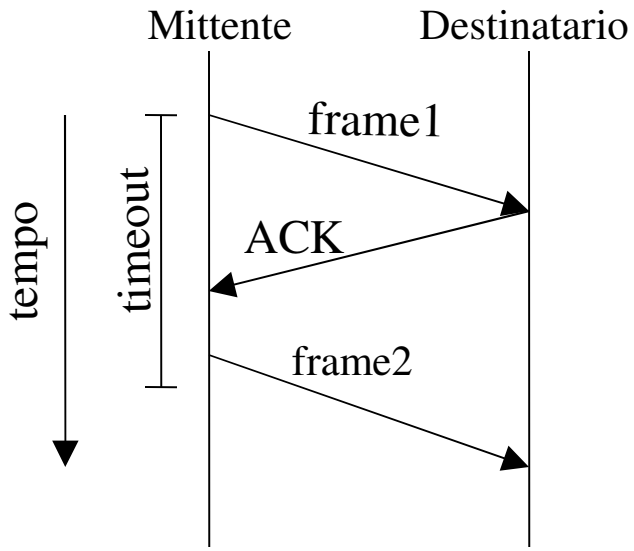
  il mittente spedisce un frame ed aspetta l'*acknowledgment (conferma)* per quel frame prima di spedire il frame successivo. Se l'acknoledgment non arriva entro un certo intervallo di tempo *(timeout),* il mittente *ritrasmette* il frame originario

- *sliding window*

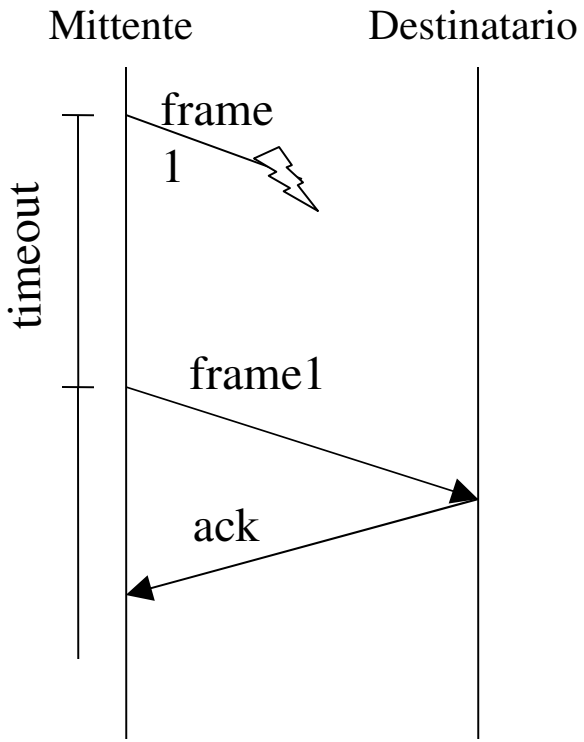
  il mittente può inviare una sequenza di pacchetti senza dover attendere l'ack.

  *Vincolo:* Esiste un limite L prefissato al numero di pacchetti non confermati (per cui non si è ricevuto l'ACK) che il cliente può inviare sulla rete

# TRASMISSIONE AFFIDABILE: STOP AND WAIT

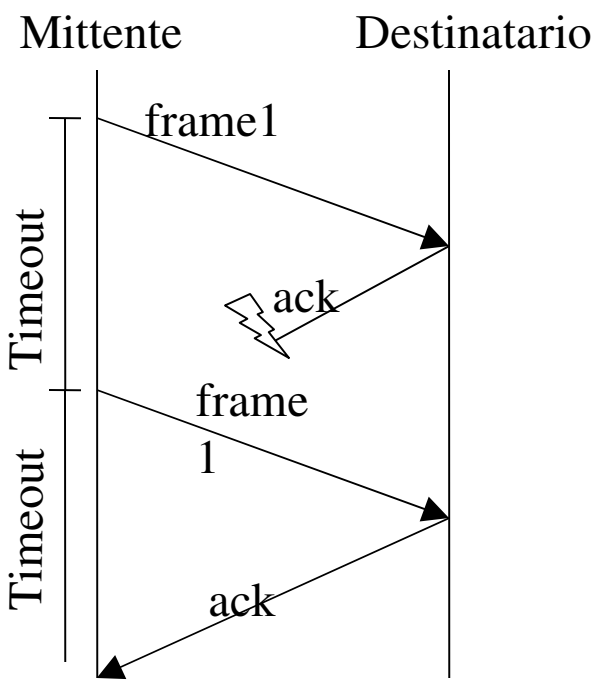



*caso1:* ack ricevuto prima del timeout.

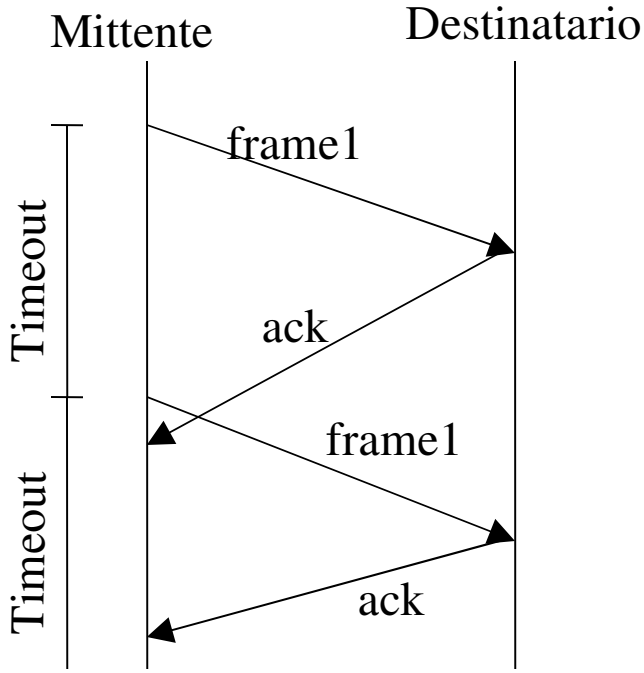



*caso2:* frame1 rinviato allo scadere del timeout (prima spedizione fallita)

# TRASMISSIONE AFFIDABILE: STOP AND WAIT




*caso 3:* frame1 rispedito allo scadere del timeout (spedizione dell'ack fallita)




*caso 4:* timeout scaduto prima della ricezione dell'ack

# TRASMISSIONE AFFIDABILE: STOP AND WAIT

- nei casi 3 e 4 il frame è ricevuto correttamente, ma a causa della perdita dell'ack o dell'arrivo ritardato dall'ack, esso viene ritrasmesso

  $\Rightarrow$

  il destinatario deve utilizzare un meccanismo per capire se il frame ricevuto è già stato ricevuto (ritrasmissione) e va scartato

- *Alternating Bit Protocol*

  - si associa un bit ad ogni frame
  - si alternano il valore del bit 0-1-0-1-0-1
  - lo stesso bit viene associato ad un frame ed alle (eventuali) copie rispedite
  - il destinatario può individuare i frames da scartare confrontando i bit associati

# TRASMISSIONE AFFIDABILE: STOP AND WAIT

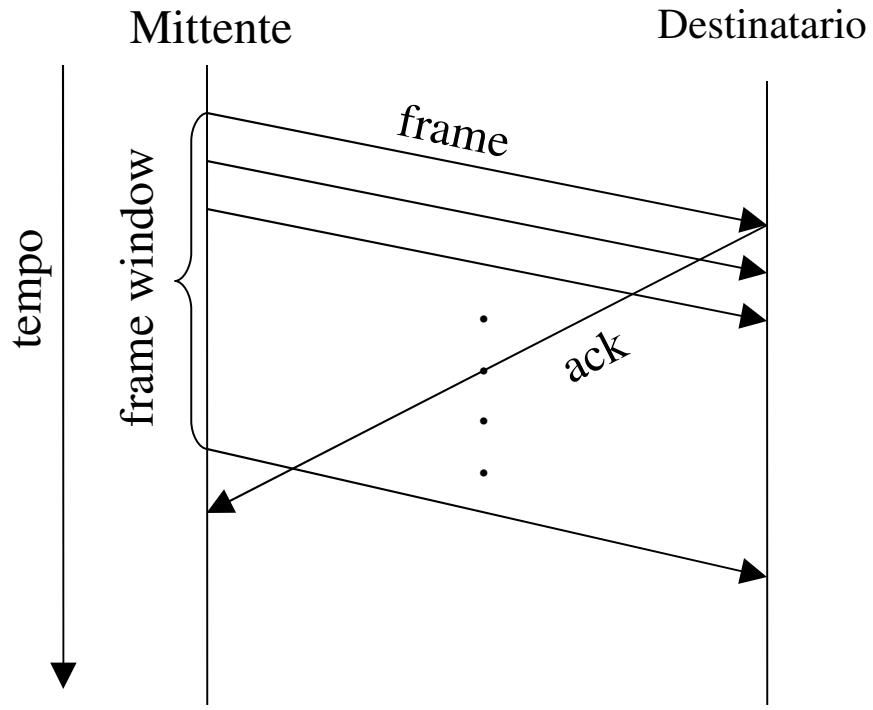
*Stop and wait*

- *vantaggi:* semplicità. E' richiesta la bufferizzazione di un solo frame nel mittente (quello per cui non si è ancora ricevuto l'ack)

- *svantaggio principale:* bassa utilizzazione della banda di trasmissione

- un solo frame sulla linea per volta, linea inutilizzata in attesa dell'ack.

# TRASMISSIONE AFFIDABILE: SLIDING WINDOW



# TRASMISSIONE AFFIDABILE: SLIDING WINDOW

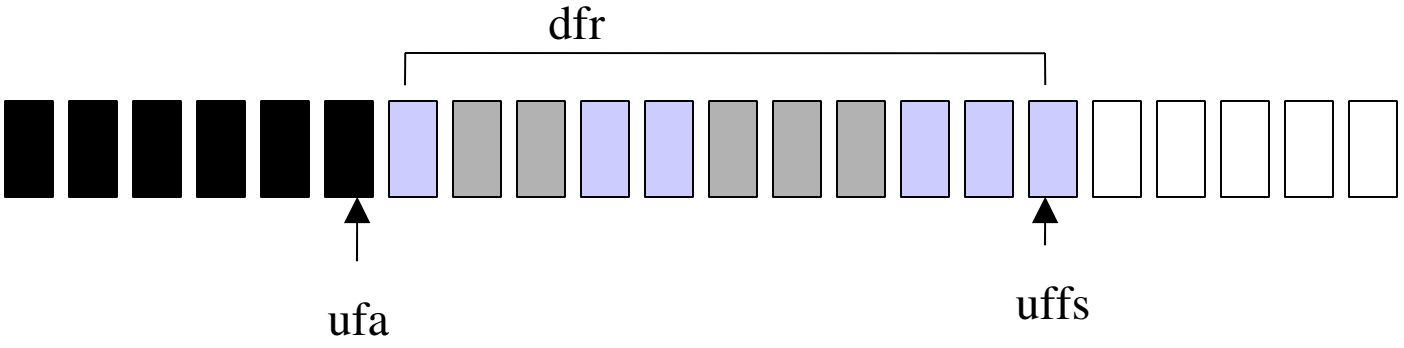
Sliding window caratteristiche:

- a ciascun frame viene assegnato un *numero di sequenza*
- richiede la bufferizzazione di diversi frames in spedizione/ricezione
  - il mittente *bufferizza* i segmenti spediti per cui non ha ancora ricevuto l'ack, in modo da *rispedirli*, se necessario
  - il destinatario *bufferizza* i segmenti ricevuti *fuori ordine*

    *esempio:*

    il destinatario riceve in successione i farmes con numeri di sequenza uguali a 7 e 9. Passa il 7 al livello superiore, bufferizza il 9 finchè non riceve l'8. Quando riceve l'8, passa l'8 ed il 9 al livello superiore

# SLIDING WINDOW: LATO DESTINATARIO




ufa - ultimo frame per cui si è inviato l'ack

uffs - ultimo frame che può essere accettato fuori sequenza

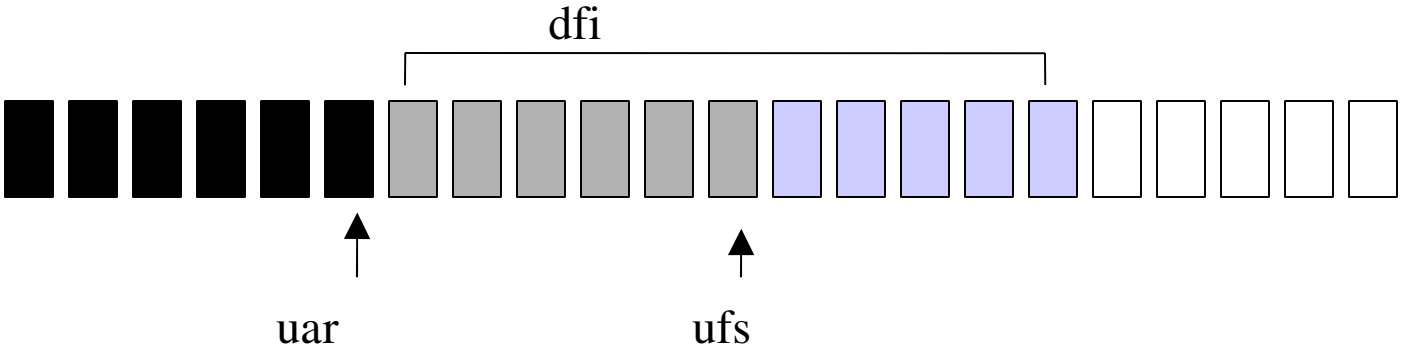
dfr - dimensione finestra di ricezione

frame ricevuti per cui si è inviato l'ack

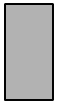
frame ricevuti fuori ordine

frame che si attende di ricevere per colmare i gaps

# SLIDING WINDOW: LATO MITTENTE




*Mittente*

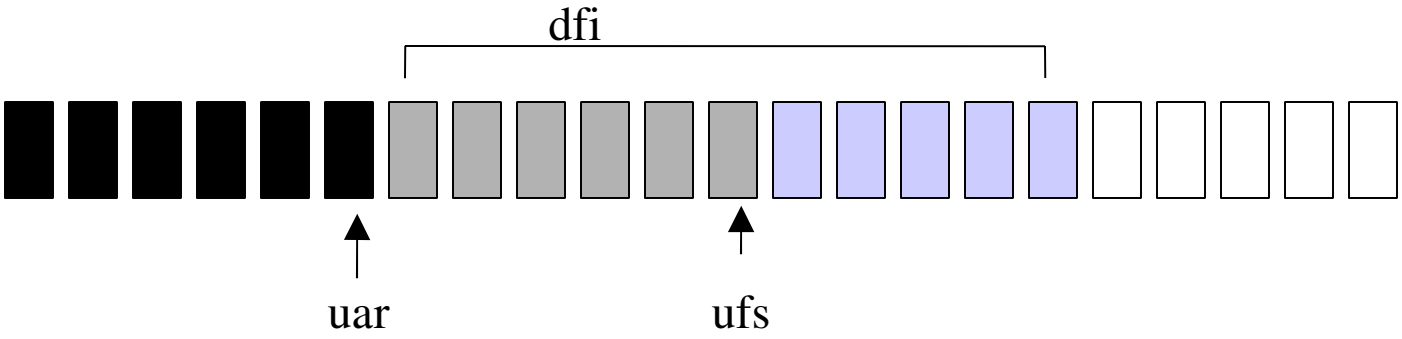
- bufferizza ed associa un timeout ad ogni frame  spedito

- se il timeout scade prima di aver ricevuto l'ack *rispedisce il frame*

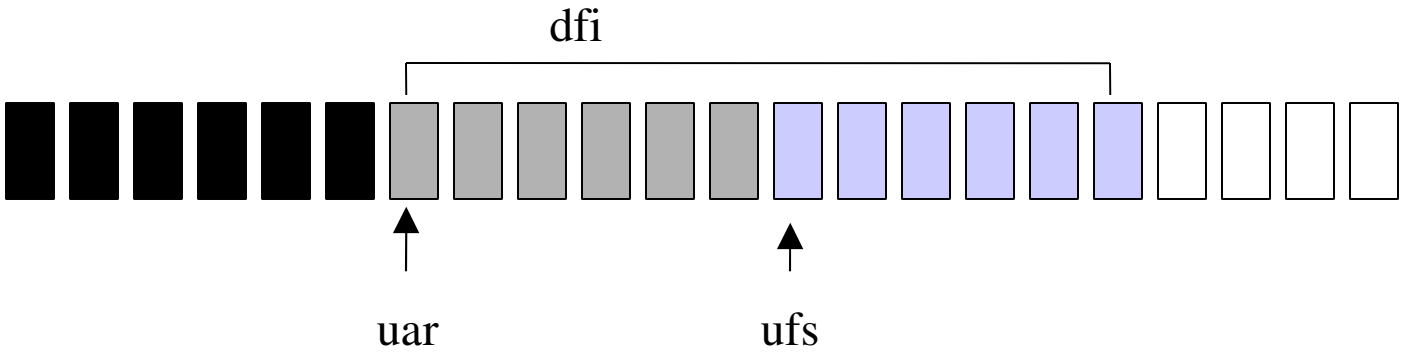
- poichè il destinatario invia gli ack in ordine, secondo i numeri di frame, quando il mittente riceve un ack spostato verso destra uar di una o più posizioni

- è in grado di bufferizzare al massimo dfi frames

# SLIDING WINDOW: LATO MITTENTE

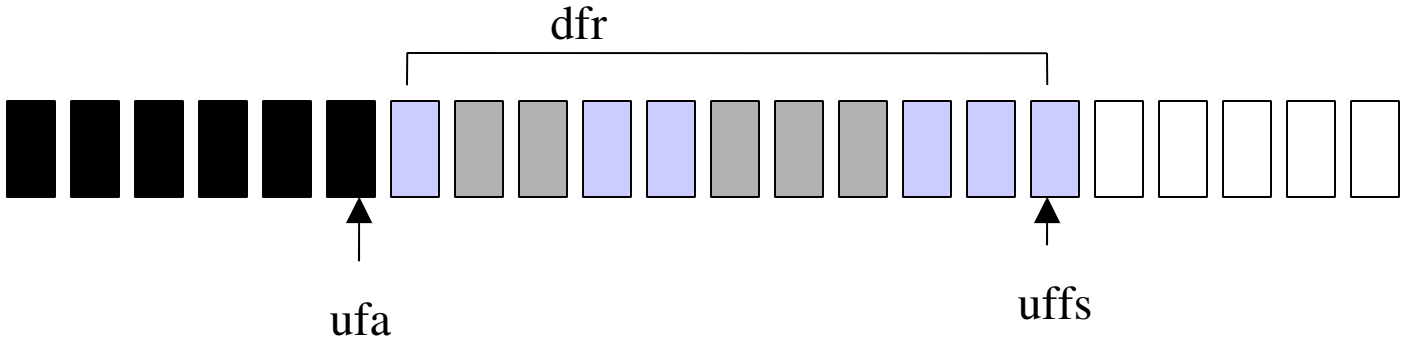



ricezione di un ack per un frames =
scorrimento della finestra di invio (sliding window)




*Osservazione:* nel caso di timeout per il frame puntato da uar, non riesco a far avanzare la finestra $\Rightarrow$ diminuisce la quantità di dati sulla linea

# SLIDING WINDOW: LATO DESTINATARIO




*Destinatario*:

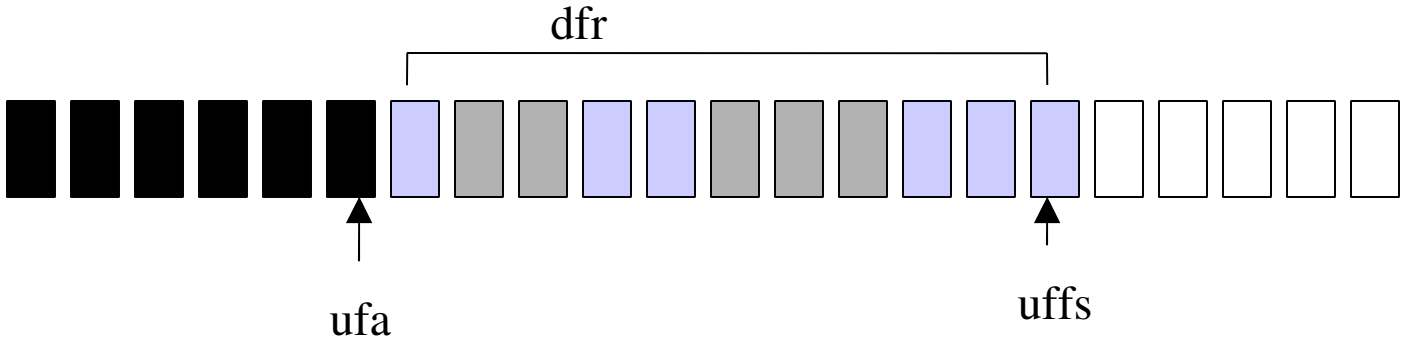
- scarta eventuali frames con numero di sequenza maggiore o uguale a uffs o minore o uguale a ufa (frames duplicati)

- ricezione di un frame f con numero di sequenza

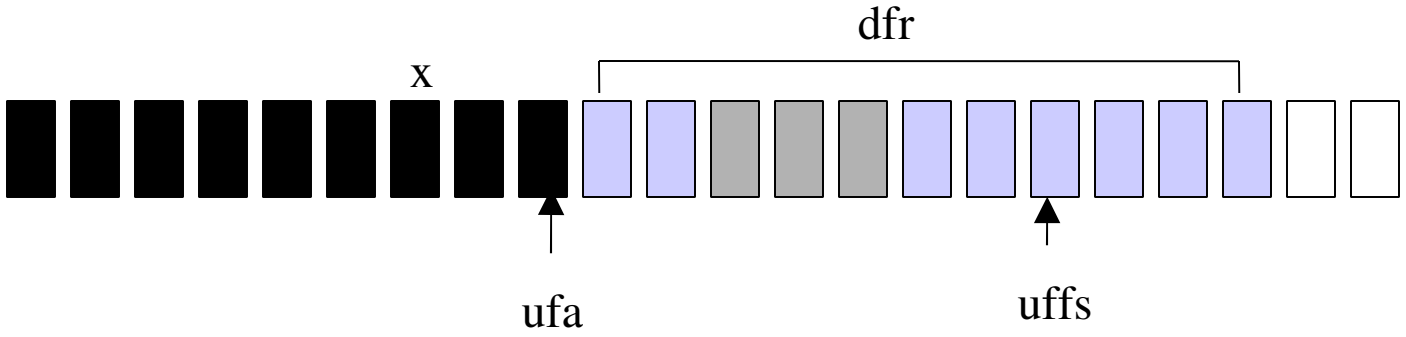
  *se* $n \neq (ufa + 1)$ il frame ricevuto è fuori ordine e viene bufferizzato (non si invia l'ack)

  *se* $n = (ufa + 1)$ si invia un *ack cumulativo* $\Rightarrow$ si conferma la ricezione di f e di tutti i frames bufferizzati successi a f, fino al primo gap.

# SLIDING WINDOW: LATO DESTINATARIO

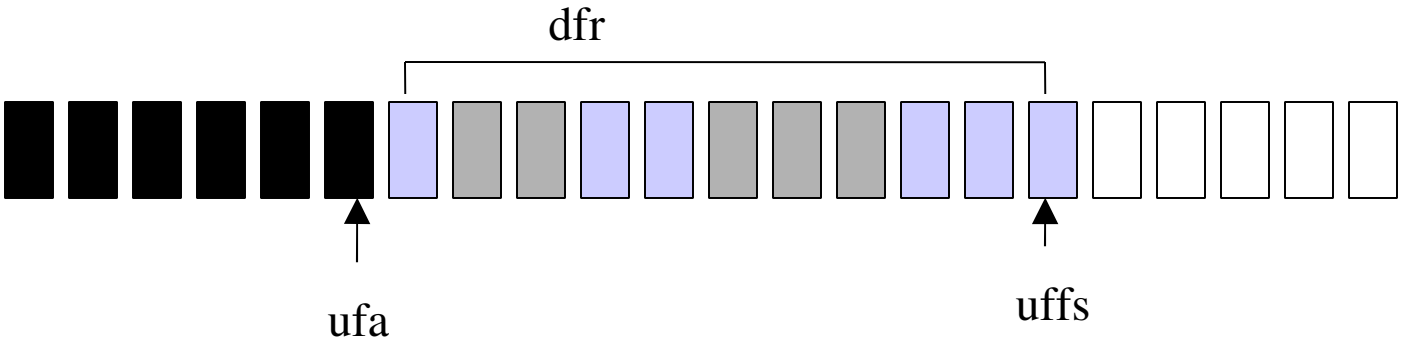



*Destnatario:* ricezione del frame con *numero di sequenza = ufa+1*

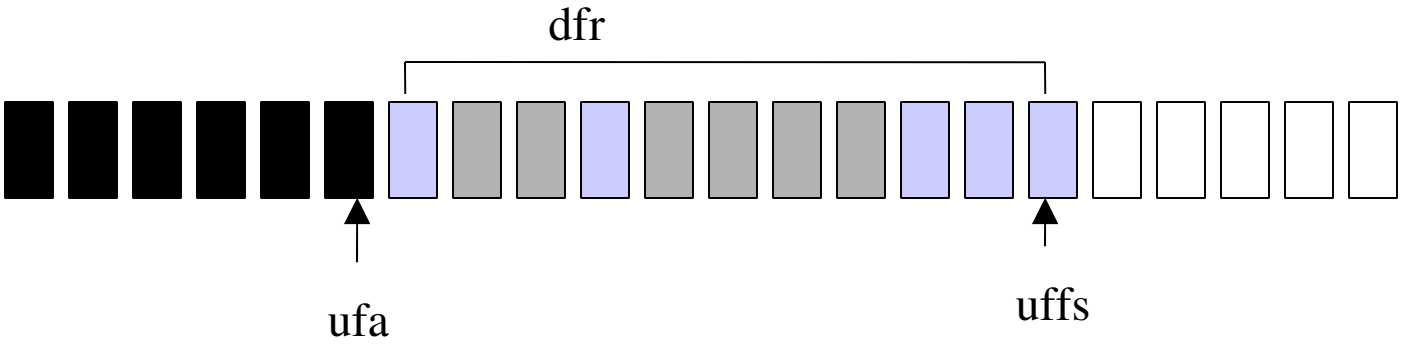



Invia **una ack** che conferma la ricezione dei pacchetti con numeri di sequenza x, x+1, x+2. L'ack contiene il numero di sequenza x+2

# SLIDING WINDOW: LATO DESTINATARIO




*Mittente:* ricezione di un frame fuori ordine




Il frame viene bufferizzato. Non viene inviato alcun ack

# SLIDING WINDOW: MODIFICHE

- Ricezione di pacchetti *fuori ordine*: è possibile mandare *conferme negative (NACK)* per i pacchetti mancanti.

  *Esempio:* ricevo il frame con numero di sequenza 8, quando il 6 ed il 7 non sono ancora arrivati. E' probabile che 6 e 7 siano stati persi $\Rightarrow$
  invio un *nack* per 6 ed uno per 7

  *Svantaggio*: invio di ulteriori messaggi sulla rete

- *Conferme selettive:* Invio un ack per ogni singolo pacchetto ricevuto.
  *Svantaggio:* gestione più complessa del buffer di spedizione

# TCP: TECNICA DELLA SLIDING WINDOW

*Sliding windows a livello TCP su WAN. Differenze rispetto alla tecnica base*

- *valore variabile di RTT* ( Round Trip Time). Il valore di RTT *può variare* su connessioni diverse ed anche per la stessa connessione, in istanti di tempo diversi

  $\Rightarrow$ definizione del *time-out* deve essere *adattiva*

- *riordinamento dei pacchetti:* tempo di vita massimo di un segmento su internet: *120 secondi.*

  $\Rightarrow$ un segmento può arrivare a destinazione con ritardo molto alto

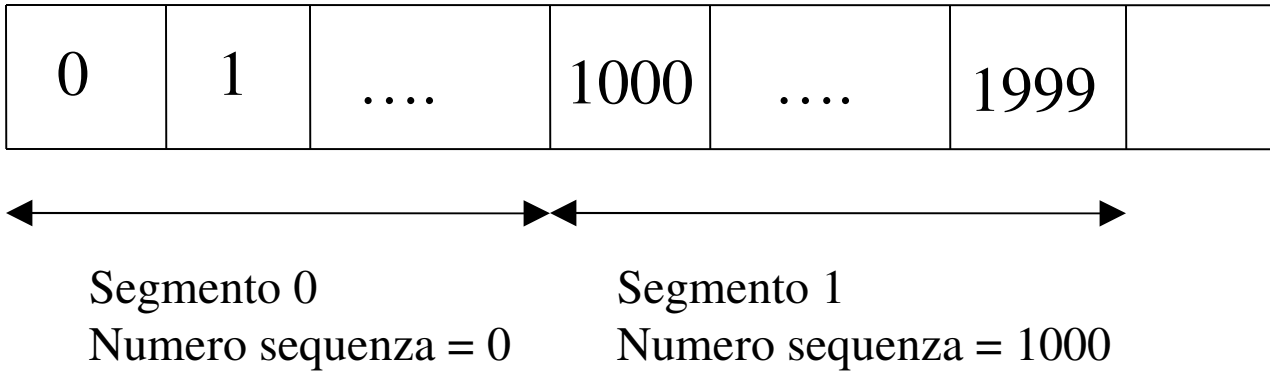
  $\Rightarrow$ un pacchetto spedito su una connessione può rimanere nella rete ed essere ricevuto mediante una connessione stabilita successivamente tra la solita coppia di hosts.

# TCP: NUMERI DI SEGMENTO

- Trasmissione TCP: considera uno stream ordinato *di bytes,* che vengono raggruppati in segmenti

- *Numero di sequenza* del segmento = numero del *primo byte* del segmento

*Esempio:* Si spedisce su una connessione TCP un file di 500.000 bytes. La dimensione di ogni segmento è di 1000 bytes



# TCP: RIVISITAZIONE SLIDING WINDOWS

Quando il destinatario riceve un segmento:

- se il segmento è *in ordine*, (presenta il numero di sequenza atteso) invio l'ack

- se il segmento è *fuori ordine* (presenta un numero di sequenza > di quello atteso) generazione di un *ack duplicato* per l'ultimo segmento ricevuto correttamente in ordine

Quando il mittente *riceve un ack* relativo al segmento con numero di sequenza k

- se l'ack non è duplicato sposto la sliding window
- se è un ack duplicato incrementa un contatore. Quando tale contatore supera un valore soglia, (in genere 3 per TCP), si suppone che il segmento k+1 sia stato perso per cui si ritrasmetta

# TCP : THREE WAY HANDSHAKE

Connessione TCP:

- client decide *attivamente* di aprire un canale di comunicazione verso un server
- server rimane in attesa (passiva) di richieste di connessioni

La richiesta di apertura di comunicazione è implementata mediante un

*three way handshake(stretta di mano)*

invio dei messaggi di controllo necessari per stabilire la trasmissione

- *client* : invia un messaggio di controllo al server (payload vuoto)
- *server* : risponde con un secondo messaggio di controllo (payload vuoto)
- *client* : invia l'ultimo messaggio di controllo (può contenere un payload)

dopo l'handshake i processi possono iniziare a scambiarsi i *segmenti*

# TCP : THREE WAY HANDSHAKE

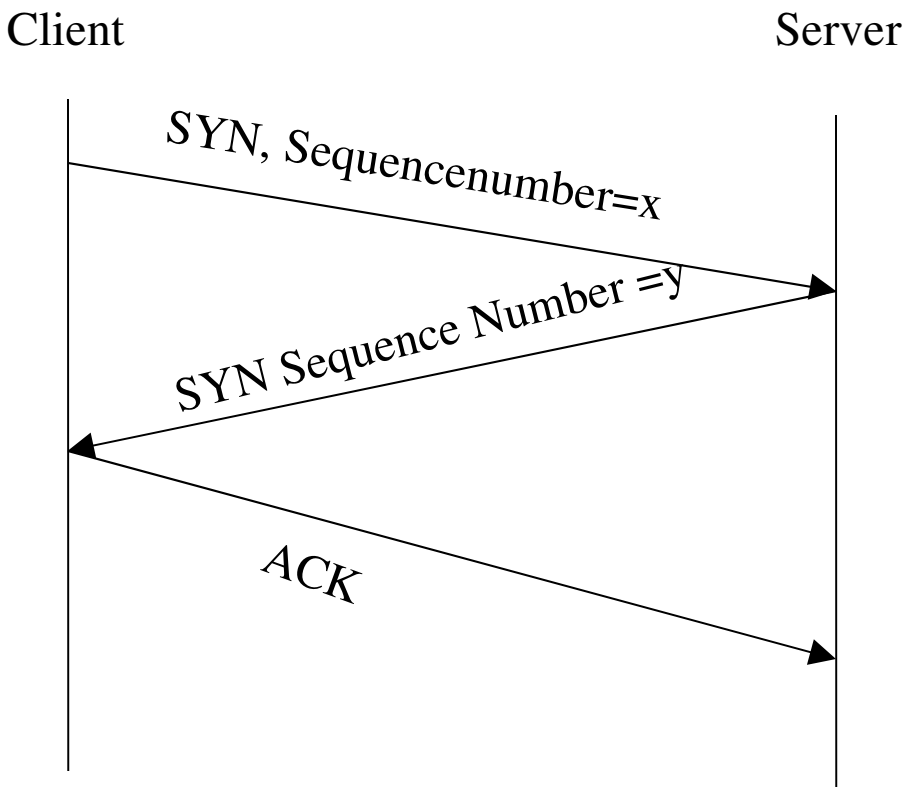
*Three-way handshake (raggiungimento di un accordo in tre fasi)*

- il client sceglie in modo casuale un numero di sequenza iniziale *n* per i segmenti che invierà al server ed invia *n* al server (SYN bit = 1)

- il server riceve *n* e lo memorizza, quindi alloca i buffers necessari per gestire la comunicazione. Sceglie a sua volta un numero iniziale di sequenza *m* per i segmenti da inviare al client ed invia *m* al client (SYN bit = 1)

- Il client riceve *m* e lo memorizza, quindi alloca i buffers necessari per gestire la comunicazione. Invia un ack al server

# TCP : THREE WAY HANDSHAKE



# TCP : THREE WAY HANDSHAKE

- Scelta casuale numeri iniziali di sequenza.
- Perché non scegliere 0 come valore iniziale dei numeri di sequanza?
- Si vuole diminuire la probabilità di utilizzare un numero di sequenza relativo ad un pacchetto ‘ in ritardo’ ancora presente sulla rete, ma relativo ad una connessione precedente

# TCP: CONTROLLO DEL FLUSSO

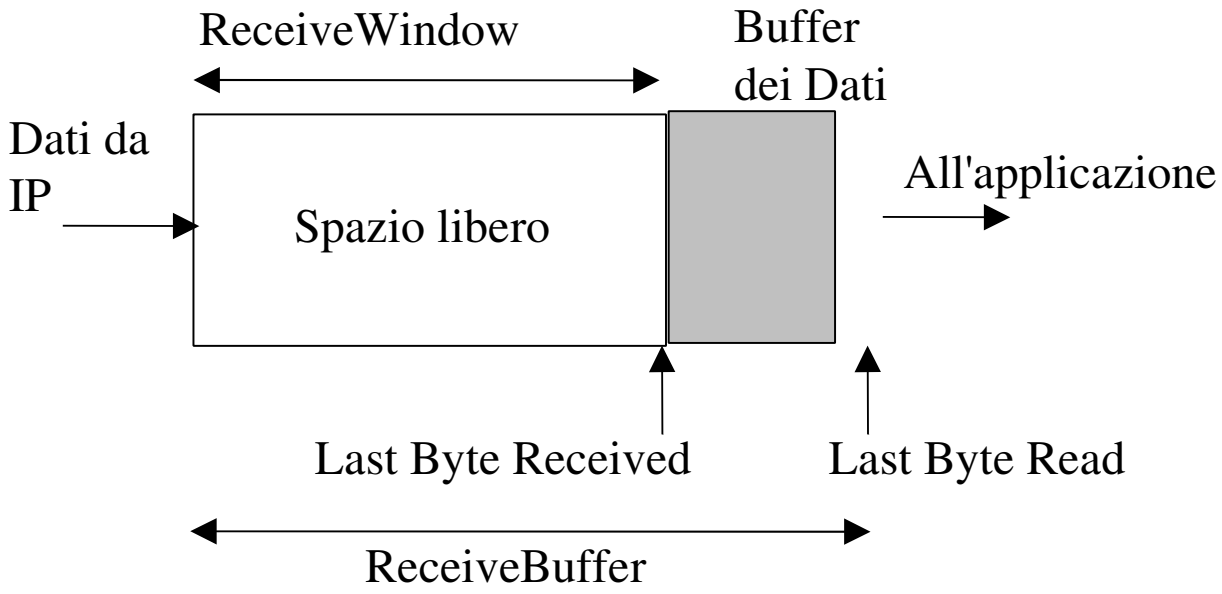
- L'host ricevente che accetta dati da una connessione IP, riceve i dati in un buffer di *dimensione finita*

- i dati vengono prelevati dal buffer da un processo a livello applicazione. Non è possibile prevedere la velocità con cui tale processo preleva i dati dal buffer

- occorre garantire che quando il buffer del destinatario si riempie, il mittente non spedisca ulteriori dati, perchè questi andrebbero persi

- *Controllo del flusso:* meccanismo che consente di regolare la velocità con cui il mittente invia i dati, a seconda della velocità con cui il destinatario preleva i dati dal buffer

# TCP: CONTROLLO DEL FLUSSO



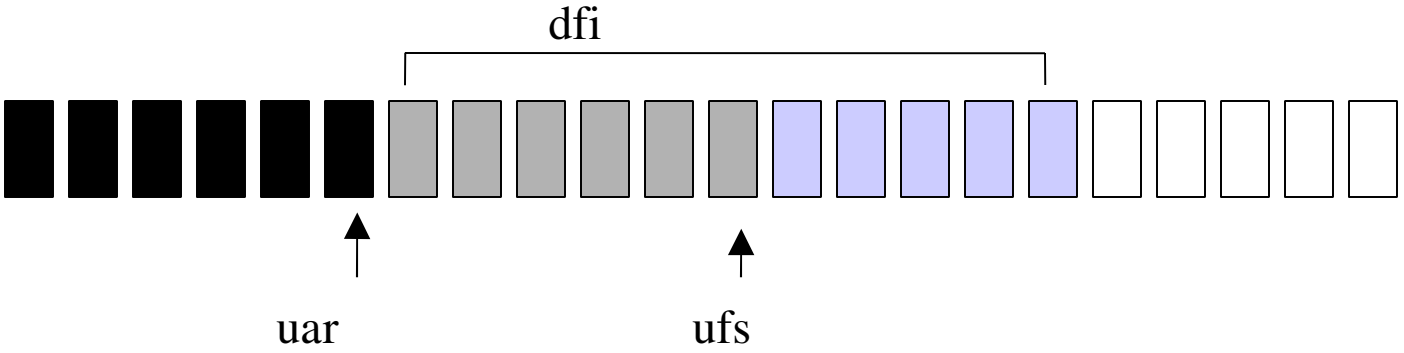

- LastByteReceived - LastByteRead <= ReceiveBuffer

- ReceiveWindow= ReceiveBuffer-[LastByteReceived-LastByteRead]

- *Controllo del flusso:* il ricevente invia la *dimensione della sua ReceiveWindow* nel segmento TCP che spedisce ad A.

All'inizio *ReceiveWindow= ReceiveBuffer*

# TCP: CONTROLLO DEL FLUSSO




*uar* ultimo acknowledgement ricevuto

*ufs* ultimo frame spedito

*dwi* dimensione finestra di invio

Il destinatario:

- riceve la dimensione della receive window del destinatario (rcws)

- deve garantire

$$ufs\text{-}uar <= rcws$$

# TCP: CONTROLLO DEL FLUSSO

- Un processo ricevente lento può provocare il blocco totale di un processo trasmittente veloce
- Il buffer di ricezione si riempie, la dimensione della receive window diventa 0
- Il mittente non può inviare altri dati, anche se i dati inviati in precedenza sono stati confermati con successo
- Il buffer di trasmissione si riempie, bloccando a sua volta il processo applicativo

# TCP: CONTROLLO DEL FLUSSO

- una volta che il ricevente ha comunicato una receive window di dimensione = 0, il mittente non può inviare altri dati.

- il ricevente non invia segmenti di risposta

- *Problema:* come può il mittente scoprire che la receive window è nuovamente disponibile?

- *Soluzione:* quando il mittente riceve la notifica che la finestra di ricezione ha dimensione = 0, continua ad inviare segmenti di 1 byte che, all'inizio non vengono accettati. Quando la dimensione della receive window diventa **diversa da 0, un segmento viene accettato ed al mittente viene notificato che la finestra e' nuovamente disponibile**