Università di Genova
Facoltà di Ingegneria

Telematica

9. TCP/IP - UDP/TCP

Prof. Raffaele Bolla

dist

Recupero di errore

- Alcuni protocolli di trasporto (TCP) applicano tecniche di recupero dell'errore con ritrasmissione, che in questo caso operano *end-to-end*.
- Queste tecniche sono applicate principalmente a livello di linea, fra due nodi adiacenti (scelta efficace se le linee sono soggette a tassi di errori significativi).
- Vediamo alcune delle tecniche più diffuse, considerando per il momento il contesto punto-punto del livello di linea.

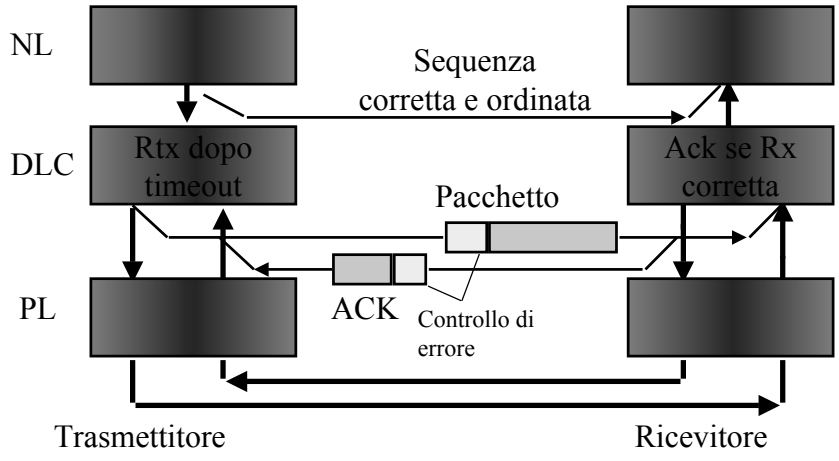
8.2

Ack e *timeout* (Cont.)

- Negli schemi ARQ (***Automatic Retransmission ReQuest***), il trasmettitore mantiene una copia del pacchetto finchè non è certo che sia stato ricevuto correttamente.
- Il ricevitore, dopo aver controllato il codice di rivelazione di errore, informa il trasmettitore, in caso di ricezione corretta, inviando un ***acknowledgement*** (*ack*).
- Il trasmettitore associa a ciascun pacchetto un *timer* e ritrasmette il pacchetto se non riceve un ack corretto entro un tempo prefissato (*timeout*).

8.3

Ack e *timeout* (Cont.)



8.4

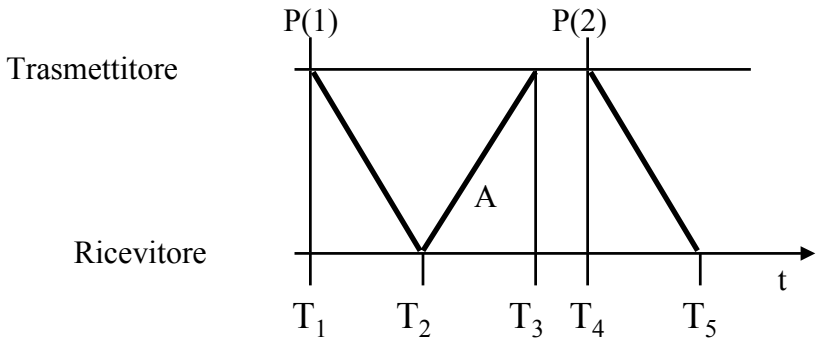
Numeri di sequenza (Cont.)

- Poiché i pacchetti in uno schema ARQ possono essere trasmessi più volte, è necessaria una loro *numerazione*, per distinguere in ricezione tra loro copie e nuovi pacchetti.
- Anche i riscontri (*acknowledgment*, più brevemente *ack*) devono essere numerati, per evitare confusione al trasmettitore.

8.5

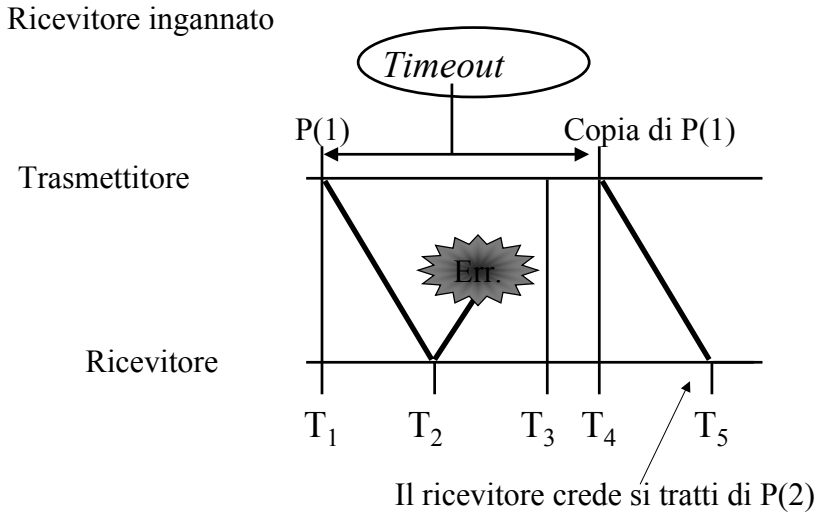
Numeri di sequenza (Cont.)

Situazione con funzionamento corretto



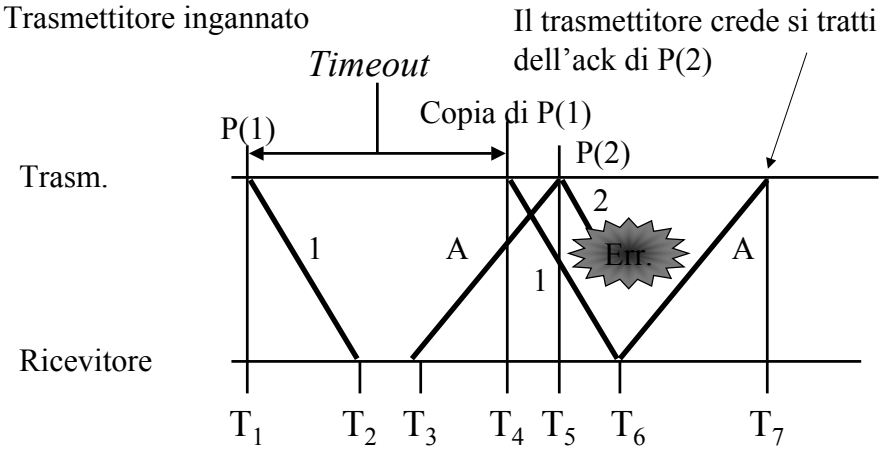
8.6

Numeri di sequenza (Cont.)



8.7

Numeri di sequenza (Fine)



8.8

Stop-and-wait (ABP) (Cont.)

Lo schema di funzionamento dei meccanismi *Stop-and-wait* o *Alternating Bit Protocol* (*ABP*) è il seguente

- Il trasmettitore (Tx) invia il primo pacchetto col numero di sequenza 0 (tenendone copia);
- Il trasmettitore attende un ACK0;
- Se ACK0 non arriva entro un *timeout*, Tx ritrasmette un'altra copia, sempre col numero 0;
- Se ACK0 arriva entro un *timeout*, Tx cancella la copia ed è pronto a trasmettere col numero 1.

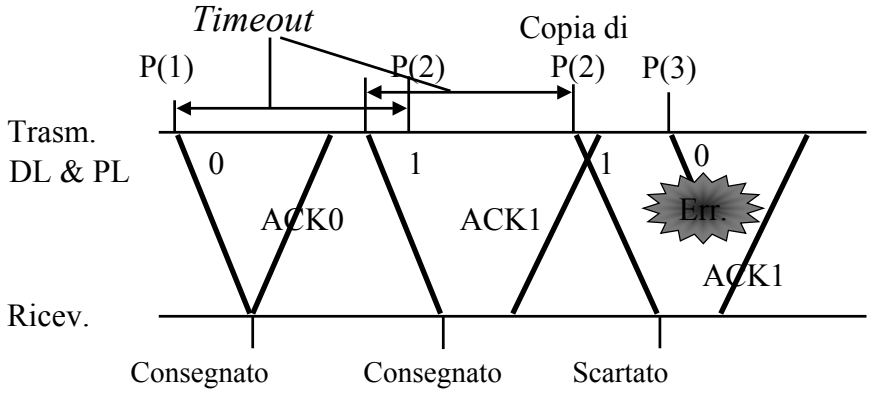
8.9

Stop-and-wait (ABP) (Cont.)

- Il ricevitore (Rx) inizialmente attende un pacchetto con il numero 0 e scarta i pacchetti errati;
- Se riceve un pacchetto corretto con il numero 1, invia un ACK1 e scarta il pacchetto;
- Quando riceve un pacchetto corretto con il numero 0, invia un ACK0 e passa il pacchetto al livello superiore.

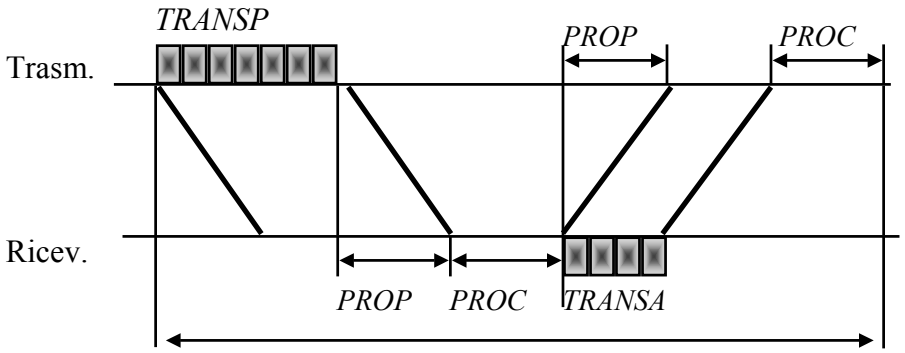
8.10

Stop-and-wait (ABP) (Cont.)



8.11

Stop-and-wait (ABP) (Cont.)



S = tempo min. fra trasmissioni successive di pacchetti

8.12

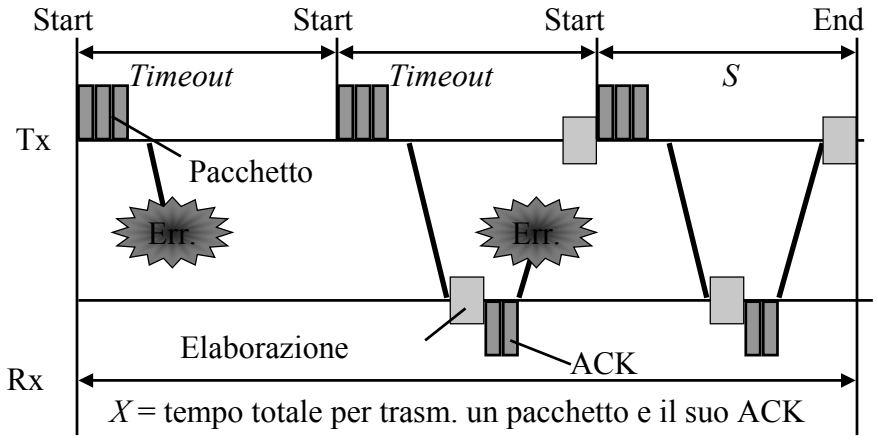
Stop-and-wait (ABP) (Cont.)

Abbreviazioni:

- *TRANSP* = tempo (tipico) di trasmissione di un pacchetto
- *PROP* = tempo di propagazione
- *TRANSA* = tempo di trasmissione di un ACK
- *PROC* = tempo di elaborazione di un ACK o di un pacchetto
  $S = TRANSP + 2xPROP + 2xPROC + TRANSA$
- *S*, a livello di trasporto diventa in Round Trip Time (RTT), tempo di andata e ritorno.

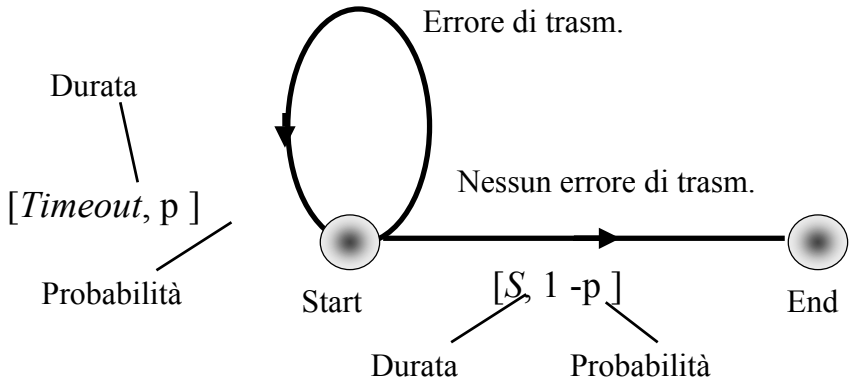
8.13

Stop-and-wait (ABP) (Cont.)



8.14

Stop-and-wait (ABP) (Cont.)



8.15

Stop-and-wait (ABP) (Fine)

- L'efficienza dell'ABP nel caso *Half-Duplex* (*HD*) e *Full-Duplex* (*FD*) risulta essere

$$\eta_{FD}(\mathrm{ABP},p) = \frac{(1\text{-}p)\mathrm{x}TRANSP}{(1\text{-}p)\mathrm{x}S + p\mathrm{x}TRANSP}$$

$$\eta_{HD}(\mathrm{ABP},p) = \frac{(1\text{-}p)\mathrm{x}TRANSP}{S}$$

8.16

Recupero di errore tramite *Sliding-Window*
(Cont.)

- Nei casi in cui il rapporto *S/TRANSP* è elevato, sarebbe conveniente poter trasmettere più pacchetti di seguito, senza doversi fermare in attesa dell'*ACK*.
- Per far questo, occorrono *buffer* più lunghi di un solo pacchetto (in trasmissione e, a seconda dei casi, anche in ricezione) ed uno schema di numerazione con modulo > 2.

8.17

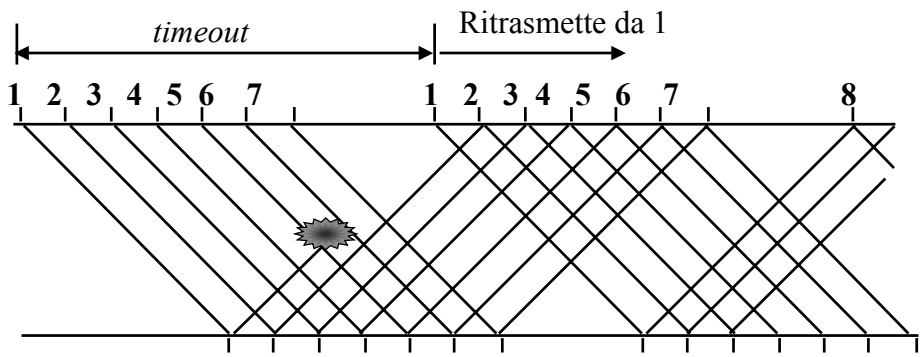
Recupero di errore tramite *Sliding-Window*
(Cont.)

Vi sono due metodi diversi che appartengono a questa categoria, denominati rispettivamente

Go-Back-N e *Selective Repeat*

8.18

Go-Back-N (Cont.)



W = ampiezza della "finestra" = 7

8.19

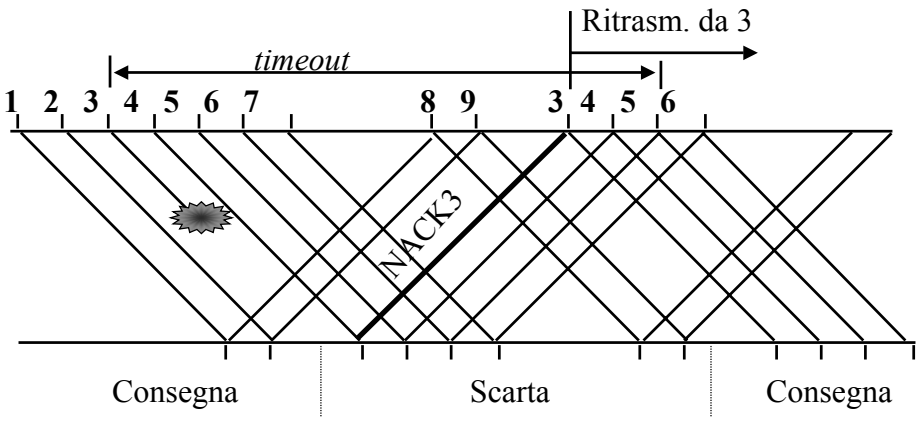
Go-Back-N (Cont.)

- Se *W* è l'ampiezza della finestra, la numerazione può essere modulo *W+1* (da 0 a *W*; il numero del pacchetto n-esimo è il resto della divisione di *n-1* per *W+1*).
- Il trasmettitore, se non riceve *ACK*, si deve fermare dopo la trasmissione di *W* pacchetti consecutivi.
- Il *buffer* del trasmettitore è *W*; quello del ricevitore è unitario. Il ricevitore scarta i pacchetti fuori sequenza, anche se corretti.

8.20

Go-Back-N **(Cont.)**

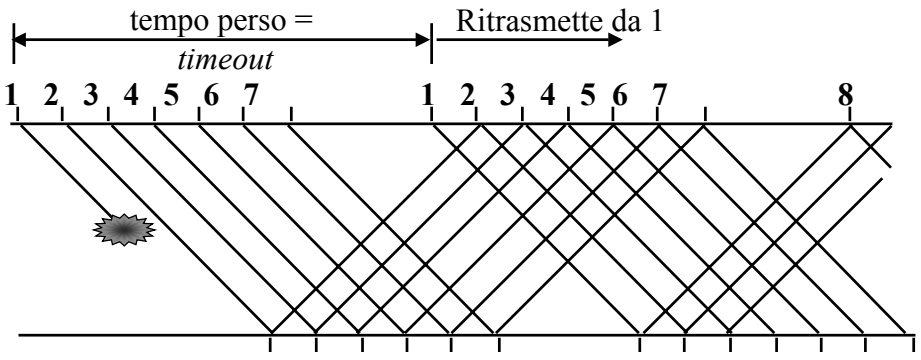
NACK



8.21

Go-Back-N **(Cont.)**

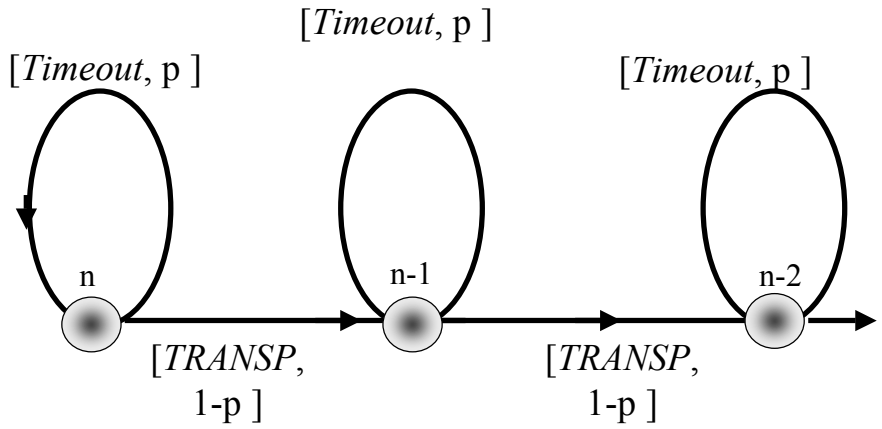
Errori



8.22

Go-Back-N **(Cont.)**

Evoluzione



8.23

Go-Back-N **(Fine)**

Efficienza

- L'efficienza del *Go-Back-N*, nell'ipotesi (*round-trip assumption*) che sia

timeout = $S = W \times TRANSP$

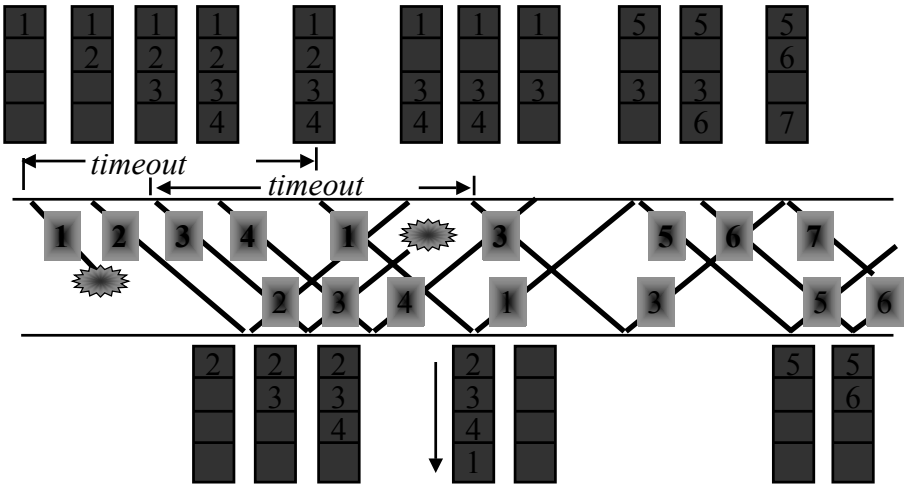
risulta essere

$$\eta_{RT}(\text{GBN, p}) = \frac{1}{1 + \frac{p}{1-p} \times W}$$

8.24

Selective Repeat Protocol

La sequenza degli eventi



8.25

Selective Repeat Protocol

Le regole

- I pacchetti sono numerati in modo sequenziale.
- I numeri dei pacchetti non ancora confermati (***unACKnowledged***) devono differire per meno di *W,* se *W* è la dimensione del *buffer.*
- Il trasmettitore memorizza i pacchetti non confermati e li ritrasmette dopo *timeout.*
- Il ricevitore memorizza i pacchetti ricevuti fuori ordine e conferma (***ACKnowledge***) quelli ricevuti corretti; consegna (al livello superiore) i pacchetti nell'ordine corretto.

8.26

Selective Repeat Protocol

Le regole (Cont.)

- I ***buffer*** del trasmettitore e del ricevitore sono uguali.
- Se i numeri di sequenza dei pacchetti in attesa di riscontro potessero differire tra loro per *W* o più, il ricevitore potrebbe trovarsi nella situazione di dover memorizzare un numero arbitrariamente grande di pacchetti.

8.27

Selective Repeat Protocol

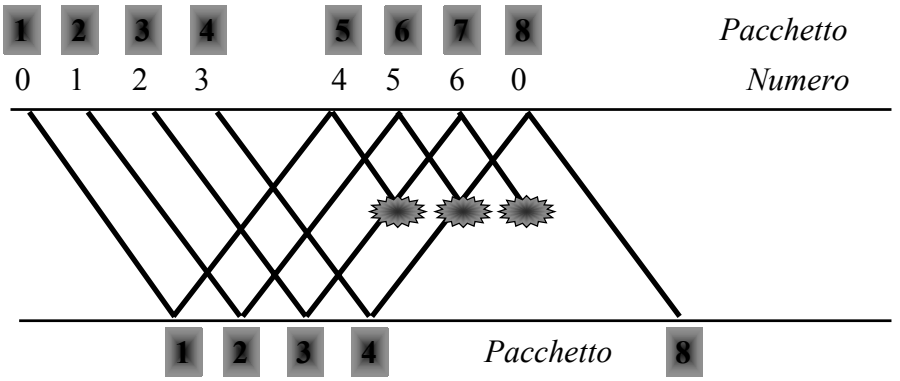
La Numerazione (Cont.)

- Come nel *Go-Back-N*, non è possibile la numerazione sequenziale progressiva senza limite.
- Se *W* è la dimensione della finestra, lo schema di numerazione deve essere modulo *2W* (il pacchetto *n*-esimo è numerato col resto della divisione di *n-1* per *2W*).

8.28

Selective Repeat Protocol

La Numerazione (Cont.)

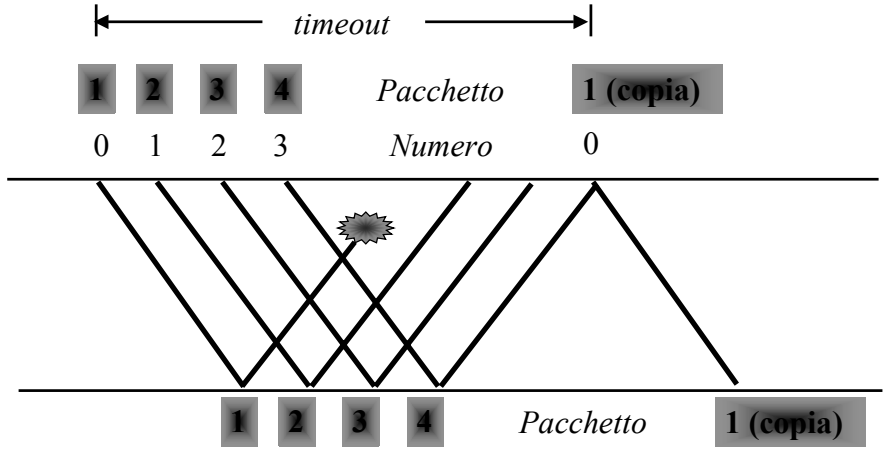


W=4 e numerazione modulo 7 confonde il ricevitore

8.29

Selective Repeat Protocol

La Numerazione



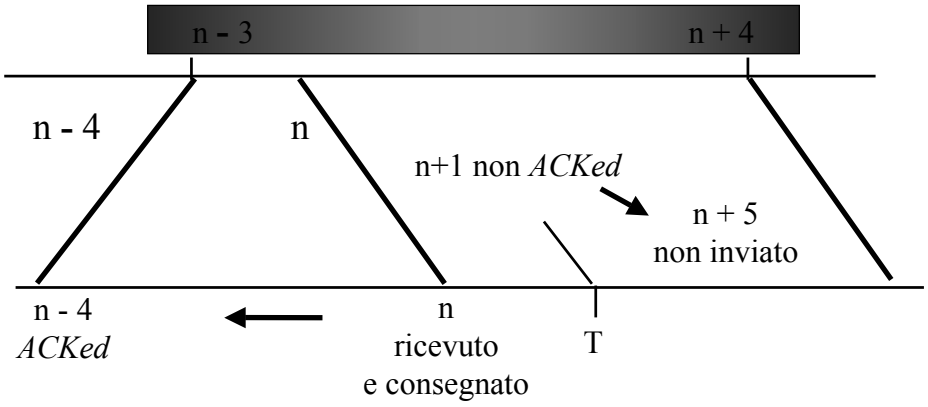
W=4 e numerazione modulo 7 confonde il Ricevitore

8.30

Selective Repeat Protocol

La Numerazione (Cont.)

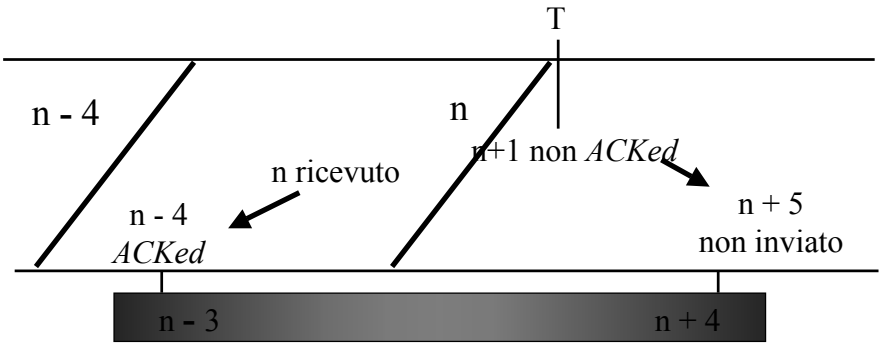
Il prossimo pacchetto al ricev. dopo il tempo T deve essere in questo insieme



8.31

Selective Repeat Protocol

La Numerazione (Fine)

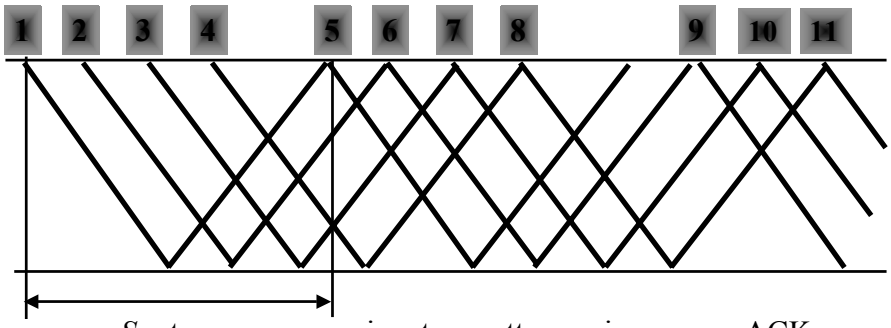


Il prossimo ACK al trasm. dopo il tempo T deve essere in questo insieme

8.32

Selective Repeat Protocol

Trasmissione senza errori



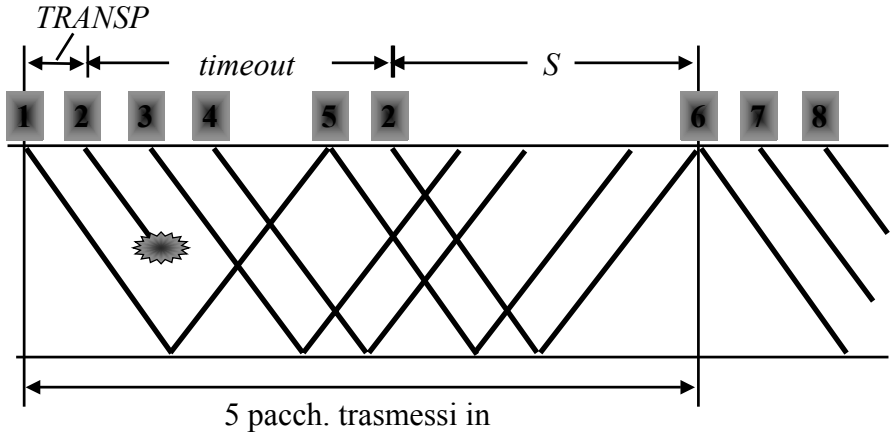
S = tempo necessario a trasmettere e ricevere un ACK

$W \times TRANSP < S$

8.33

Selective Repeat Protocol

Evoluzione con 1 errore



5 pacch. trasmessi in
(*S* + *TRANSP* + *timeout*) secondi

8.34

Protocolli di trasporto

- Si consideri per prima una rete affidabile, ossia che consegni i pacchetti in ordine e senza errori (per esempio grazie ad un livello di linea affidabile che realizzi il recupero dell'errore).
- I compiti che devono essere svolti dal livello di trasporto in questo caso sono:
  - Indirizzamento
  - Multiplexing
  - Controllo di flusso
  - Apertura e chiusura delle connessioni

8.35

Protocolli di Trasporto
Indirizzamento

- Per individuare l'entità di applicazione a cui inviare l'informazione, l'entità di trasporto (ET) di sorgente ha bisogno:
  - Identificatore dell'utente (porta/SAP)
  - Identificatore entità di trasporto (di destinazione)
  - Indirizzo dell'host (rete)
- In genere l'utente è individuato dalla coppia (porta, host) di cui la porta viene inserita nell'intestazione dell'ET, mentre l'indirizzo di host viene passato al livello di rete.

8.36

Protocolli di Trasporto

Multiplexing

- Il livello di trasporto può operare un multiplexing diretto o inverso nei confronti del livello di rete:
  - Diretto: inviando più flussi contemporaneamente su un unico servizio del livello di rete (per diminuire l'*overhead*)
  - Inverso: suddividendo più flussi su servizi del livello di rete (per migliorare le prestazioni).

8.37

Protocolli di Trasporto

Controllo di flusso

- Controllo di flusso significa regolare l'immissione di dati nella rete da parte della sorgente.
- La ragione per la quale può essere necessaria questa azione è (in questo caso):
  - L'utente dell'ET ricevente non è in grado di accettare il flusso di dati attuale
  - L'entità stessa di trasporto non è in grado di accettare il flusso (per es. buffer esauriti)
- La situazione è diversa da quella presente a livello di linea perché il ritardo di andata e ritorno (*Round Trip Delay*) è
  - Molto più lungo
  - Potenzialmente molto variabile

8.38

Protocolli di Trasporto

Controllo di flusso

- Quattro possibili strategie:
  - Non fare nulla (scartare i pacchetti)
  - Rifiutare ulteriori pacchetti dal livello di rete
  - Usare un protocollo a *sliding window* (finestra scivolante) fissa
  - Usare un meccanismo a credito
- Il secondo caso significa demandare il controllo di flusso al livello di rete o comunque ai livelli inferiori

8.39

Protocolli di Trasporto

Controllo di flusso - *Sliding Window*

- Il controllo di flusso *sliding window* è in sostanza basato sul mancato invio delle conferme.
- Il trasmettitore non conferma l'ultimo (o gli ultimi) pacchetto arrivato che gli satura il buffer
- Siccome la rete è affidabile, il trasmettitore può interpretare il mancato invio di una conferma come indicazione di eccessivo invio e quindi (sapendo che i pacchetti devono essere arrivati) non ritrasmetterli fino a che non riceve le conferme mancanti.
- Non funziona correttamente in caso di rete non affidabile (non è in grado di distinguere fra perdite reali e indicazioni di rallentare).

8.40

Protocolli di Trasporto

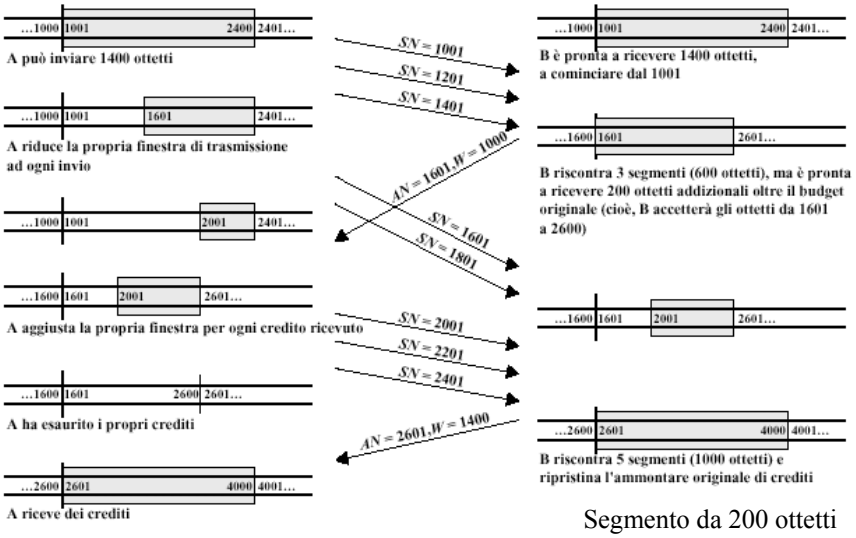
Controllo di flusso - Meccanismi a credito

- Separano la conferma dalla dimensione della finestra.
- Quando il trasporto invia dati inserisce un SN (*Sequence Number*) che indica il numero del primo ottetto inviato
- Ogni entità ricevente fornisce una conferma con le seguenti informazioni
  - AN (*Ack Number*) che indica che tutti gli ottetti con SN = AN-1 sono arrivati correttamente
  - W che indica che è possibile trasmettere ancora W ottetti, quelli da AN a AN+W-1.

8.41

Protocolli di Trasporto

Controllo di flusso - Meccanismi a credito



SN = Serial Number; AN = Acknowledge Number; W = dimensione della finestra

8.42

Protocolli di Trasporto

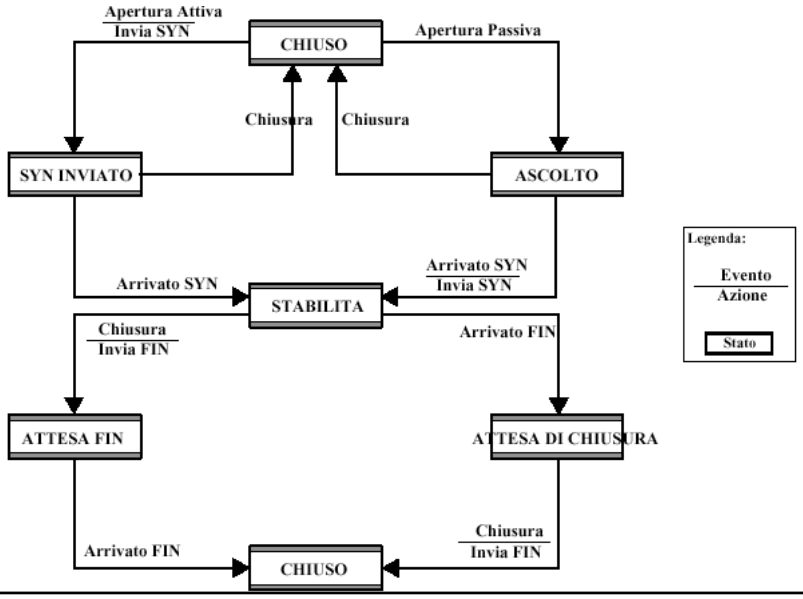
Apertura e chiusura della connessione

- L'apertura della connessione serve a tre scopi:
  - Assicurarsi che la destinazione esiste ed è consenziente alla comunicazione;
  - Permettere la negoziazione di parametri;
  - Attivare l'allocazione di risorse (spazio nei buffer);

8.43

Protocolli di Trasporto

Apertura e chiusura della connessione



8.44

Protocolli di Trasporto

Rete non affidabile

- Nel caso la rete non sia affidabile, ossia possano avvenire perdite/errori ed i pacchetti possano arrivare fuori sequenza, allora bisogna affrontare o riaffrontare i seguenti aspetti:
  - Consegna ordinata (numeri di sequenza)
  - Strategia di ritrasmissione
  - Identificazione dei duplicati
  - Controllo di flusso
  - Apertura della connessione
  - Chiusura della connessione

8.45

Protocolli di Trasporto

Strategia di ritrasmissione

- Si possono utilizzare i sistemi ARQ.
- Il problema principale e stimare la durata del *timeout*, ossia la durata del ***Round Trip Time*** (RTT, tempo di andata e ritorno).
- In genere si tenta di usare una stima adattativa di tale parametro, per esempio facendo la media dei ritardi con cui giungono le conferme; questo modo di procedere può non funzionare perché:
  - Il ricevitore potrebbe non inviare l'Ack immediatamente
  - Se il segmento è stato ritrasmesso, il ricevitore non può sapere se l'Ack è riferito alla prima o alla seconda trasmissione
  - Le condizioni delle rete possono cambiare molto velocemente

8.46

Protocolli di Trasporto

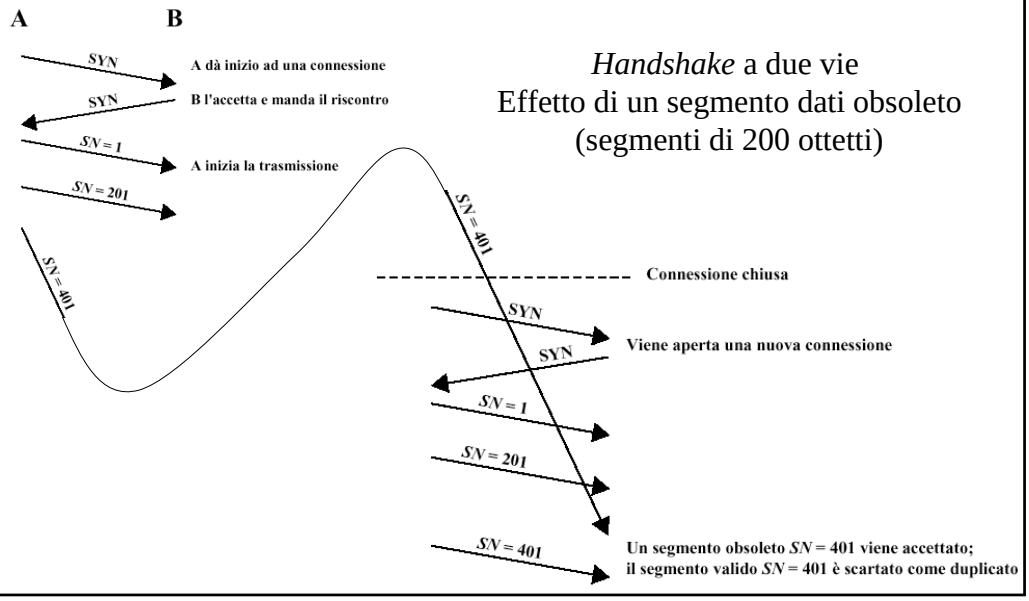
Controllo di flusso

- Il controllo di flusso a credito funziona bene anche nelle reti non affidabili.
- Anche se va persa una conferma che stabilisce un nuovo valore di finestra più piccolo, il risultato sarà che il trasmettitore ri-invierà dei pacchetti e le nuove conferme ripristineranno la situazione giusta.
- L'unica situazione di *deadlock* può avvenire se fra due stazioni A e B, B invia (AckNumber =AN = i, Window = W = 0) e poco dopo (AN = i, W = j) ma quest'ultimo pacchetto va perso. In questo caso A non invia più nulla perché ha la finestra chiusa e B neanche perché attende nuovi pacchetti. Soluzione: Timer legato alla finestra.

8.47

Protocolli di Trasporto

Apertura della connessione



Handshake a due vie
Effetto di un segmento dati obsoleto
(segmenti di 200 ottetti)

Protocolli di Trasporto

Apertura della connessione

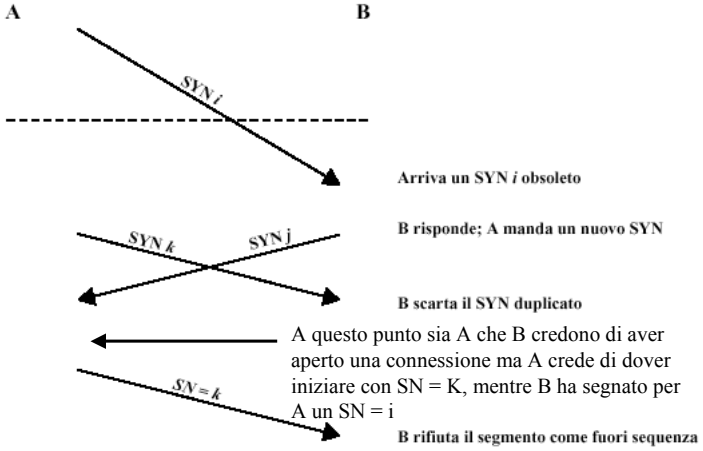
- Per evitare che pacchetti di connessioni già chiuse confondano quella attiva si può, ad esempio, usare un numero iniziale di sequenza diverso per connessione (supponendo che lo spazio dei numeri di sequenza sia grande)
- Quindi, in pacchetto di apertura SYN trasporterà anche il numero di sequenza (SYN i).

8.49

Protocolli di Trasporto

Apertura della connessione

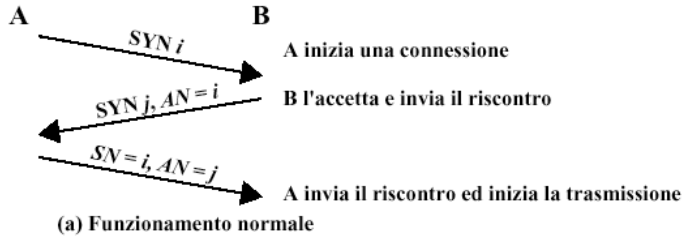
Handshake a due vie
Effetto di un segmento SYN obsoleto



8.50

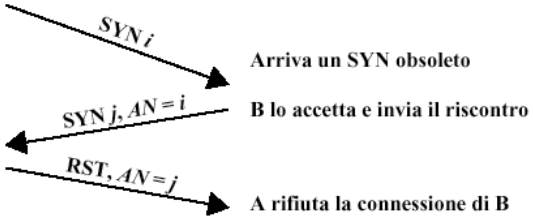
Protocolli di Trasporto

Apertura della connessione



(a) Funzionamento normale

Handshake a tre vie

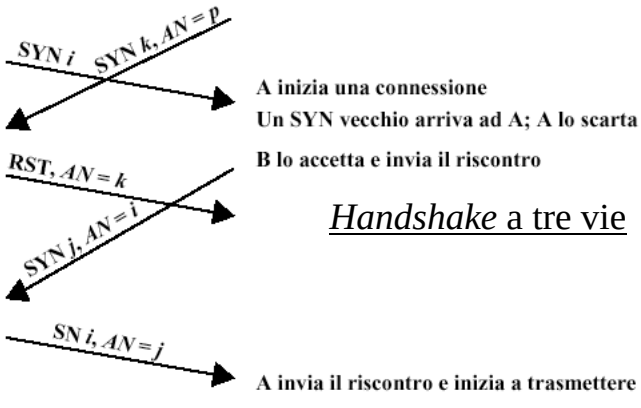


(b) SYN in ritardo

8.51

Protocolli di Trasporto

Apertura della connessione



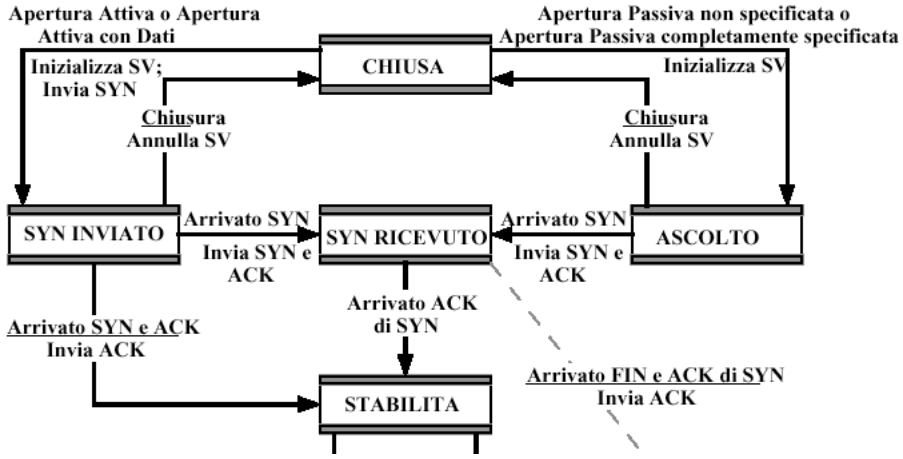
Handshake a tre vie

(c) SYN e ACK in ritardo

8.52

Protocolli di Trasporto

Apertura della connessione

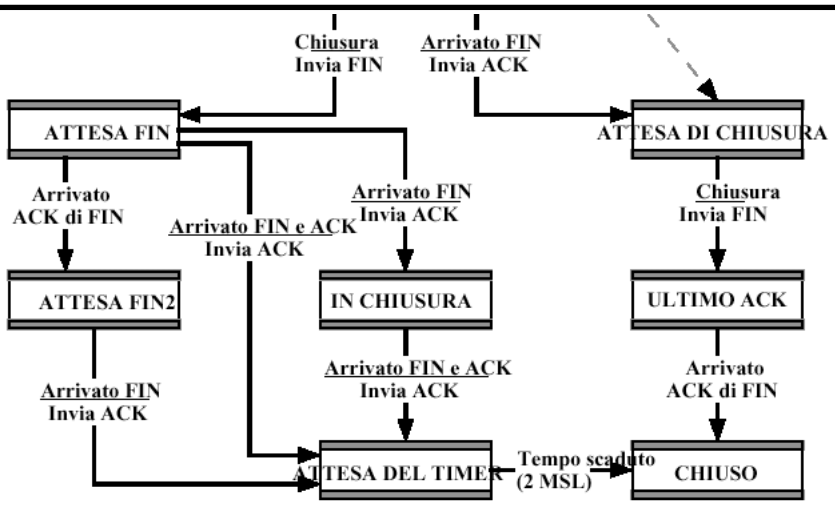


SV = State Vector (vettore di stato)

8.53

Protocolli di Trasporto

Chiusura della connessione



SV = State Vector (vettore di stato)
MSL = Maximum Segment Lifetime (tempo di vita massimo del segmento)

8.54

Protocolli di trasporto nel TCP/IP

- La suite TCP/IP propone due diversi protocolli di trasporto, che hanno delle caratteristiche opposte:
  - ***User Datagram Protocol*** (**UDP**) semplicissimo, che non fornisce quasi nessuna funzionalità (è circa come usare l'IP direttamente)
  - ***Transmission Control Protocol*** (**TCP**) che invece fornisce tutte le possibili funzionalità previste a livello di trasporto

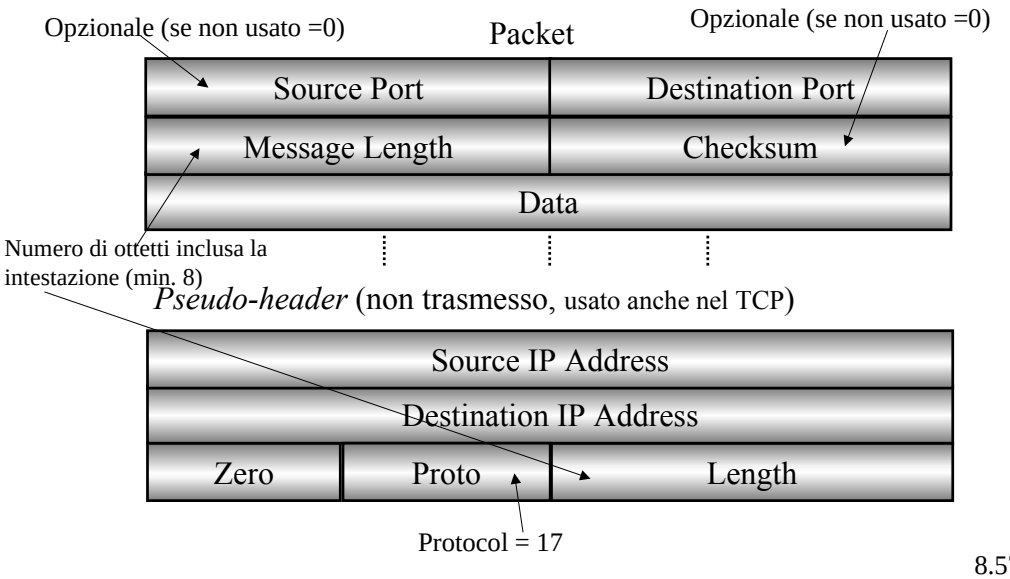
8.55

User Datagram Protocol (UDP)

- UDP realizza un protocollo di trasporto non orientato alla connessione e non affidabile (senza recupero d'errore).
- Aggiunge alle funzionalità di IP la possibilità di distinguere fra diverse destinazioni all'interno della stessa macchina, ma non garantisce l'integrità dei dati e consegna ordinata dei dati.
- E' identificato dal *Protocol Number* 17 nell'*header* IP.
- In sostanza permette solo di identificare le applicazioni sorgenti e destinazioni tramite un ***port number***.

8.56

User Datagram Protocol (UDP)



8.57

User Datagram Protocol (UDP)

- I numeri di porta sono di due tipi:
  - Preassegnati (*well-known*) in modo univoco a certi servizi, e sono i numeri più bassi;
  - Assegnati dinamicamente, tutti gli altri.
- Alcuni dei preassegnati sono:
  - 0 ECHO *echo*
  - 11 USERS Utenti attivi
  - 37 TIME Data e ora
  - 42 NAMESERVER *Domain name server*
  - 69 TFTP *Trivial File Transport*
  - 520 RIP *Routing Internet Prot.*

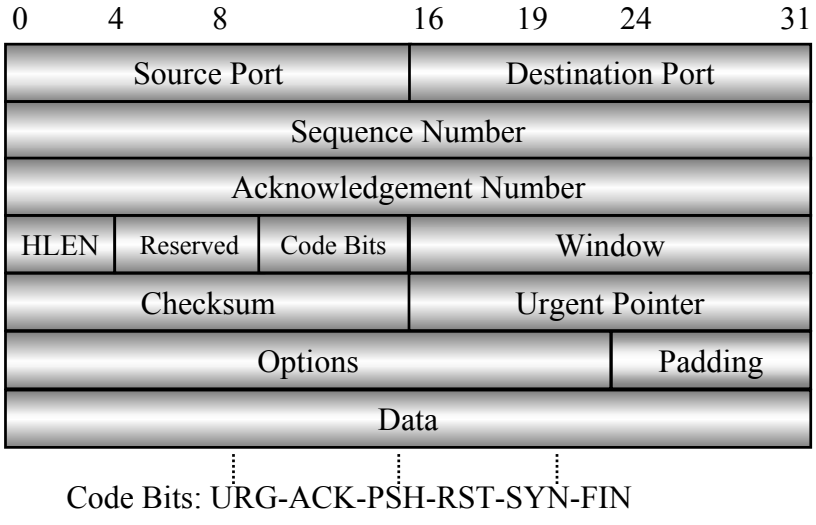
8.58

Transmission Control Protocol (TCP)

- Il TCP è definito nel RFC 793, ed ha le seguenti caratteristiche principali:
  - È orientato alla connessione.
  - Full-duplex.
  - Unicast.
  - Trasporta flussi di dati.
  - Realizza un trasmissione affidabile (recupero d'errore e consegna ordinata) su un servizio di rete datagram non affidabile (IP).
  - Applica un controllo di flusso/congestione adattativo.

8.59

Transmission Control Protocol (TCP)



8.60

Transmission Control Protocol (TCP)

- TCP usa soltanto un singolo tipo di unità dati di protocollo, chiamato segmento TCP. L'intestazione è piuttosto lunga (minima 20 ottetti), visto che è unica per tutti i meccanismi del protocollo.
  - ***Source Port*** (porta di sorgente) **(16 bit):** Utente TCP sorgente
  - ***Destination Port*** (porta di destinazione) **(16 bit):** Utente TCP destinazione
  - ***Sequence Number*** (numero di sequenza) **(32 bit):** Numero di sequenza del primo ottetto dati in questo segmento, tranne quando il flag SYN è attivo. Se il flag SYN è attivo, contiene L'*Initial Sequence Number* (ISN, numero di sequenza iniziale) ed il primo ottetto di dati è ISN +1.

8.61

Transmission Control Protocol (TCP)

  - ***Acknowledgement Number*** (numero di riscontro) **(32 bit):** Un riscontro *piggybacked* (inserito in un invio di dati in senso opposto). Contiene il numero di sequenza dell'ottetto dati successivo che l'entità TCP si aspetta di ricevere.
  - ***Data Offset*** (offset dei dati) **(4 bit):** Numero di parole di 32 bit nell'intestazione.
  - **Flag (6 bit):**
    - » URG: Campo *Urgent Pointer* significativo.
    - » ACK: Campo Acknowledgement Number significativo.
    - » PSH: Funzione push.
    - » RST: Re-inizializza la connessione.
    - » SYN: Sincronizza i numeri di sequenza.
    - » FIN: Nessun dato in più dal trasmettitore.

8.62

Transmission Control Protocol (TCP)

- ***Window*** (finestra) **(16 bit):** Allocazione di crediti del controllo di flusso, in ottetti. Contiene il numero di ottetti dati, a partire da quello indicato nel campo di Acknowledgement, che il trasmettitore è disposto ad accettare.
- **Checksum (16 bit):** Complemento ad uno della somma modulo 2^{16} – 1 di tutte le parole a 16 bit nel segmento più una pseudo-intestazione
- ***Urgent Pointer*** (puntatore d'urgenza) **(16 bit):** Indica l'ultimo ottetto nella sequenza di dati di tipo *urgent*.
- ***Option*** (opzioni) **(variabile):** Ad esempio:
  - » Dim max del segmento (MSS, Maximum Segment Size)
  - » **Window- scale**
  - » **SACK (*Selective* Ack).**

8.63

Transmission Control Protocol (TCP)

- Apertura/chiusura della connessione
  - Utilizza un meccanismo di *handshake* a tre vie, la connessione è individuata dalla coppia porta di destinazione di sorgente. Quindi può esserci una solo connessione aperta per coppia. La chiusura può anche essere "brutale" tramite la primitiva ABORT.
- Trasferimento dei dati
  - Avviene numerando gli ottetti inviati, sono previste due modalità aggiuntive
    - » **Flusso dati *push*:** TCP decide autonomamente quando sono stati accumulati un numero sufficiente di dati per formare un segmento da trasmettere. L'utente può richiedere che il TCP trasmetta tutti i dati in sospeso fino a, o inclusi quelli, etichettati con un *push flag*. Dal lato del ricevitore, il TCP inoltra questi dati all'utente in modo analogo.
    - » **Flusso dati *urgent*:** Fornisce uno strumento per informare l'utente TCP di destinazione che dei dati significativi o "urgenti" si trovano nel flusso dati in arrivo. Dipende dall'utente di destinazione determinare come comportarsi in presenza di questo tipo di dati.

8.64

Transmission Control Protocol (TCP)

Realizzazioni

- Il protocollo lascia alcune scelte sostanzialmente libere, ossia permette diverse realizzazioni tra loro comunque interoperabili. Questo in particolare per quanto concerne
  - Strategia di trasmissione (immediato o ritardato)
  - Strategia di inoltro
  - Strategia di accettazione (in ordine, o nella finestra (invia Ack solo per dati in ordine))
  - Strategia di ritrasmissione
  - Strategia di riscontro (singolo o cumulativo) per i bit arrivati nell'ordine corretto

8.65

Transmission Control Protocol (TCP)

Strategia di ritrasmissioni

- Il TCP mantiene una coda di segmenti già trasmessi ma non riscontrati. La specifica TCP stabilisce che si ritrasmetta un segmento se non si riceve un riscontro entro un certo lasso di tempo. Una realizzazione di TCP può impiegare una di tre diverse strategie di ritrasmissione.
  - **Solo-il-primo**
  - **A lotto**
  - **Individuale**

8.66

Transmission Control Protocol (TCP)

Strategia di ritrasmissioni

- **Solo-il-primo:**
  - » Mantiene un solo timer di ritrasmissione per l'intera coda
  - » Se riceve un riscontro, rimuove il relativo segmento o segmenti dalla coda ed ri-inizializza il timer.
  - » Se il tempo si esaurisce, ritrasmette il segmento in cima alla coda e ri-inizializza il timer.
  - – E' efficiente in termini di traffico generato, in quanto vengono ritrasmessi solo i segmenti persi (o i segmenti il cui ACK è stato perso).
  - – Tuttavia, poiché il timer nel secondo segmento della coda non viene attivato fino a quando non arriva il riscontro del primo segmento, si possono verificare ritardi considerevoli

8.67

Transmission Control Protocol (TCP)

Strategia di ritrasmissioni

- **A lotto:**
  - » Mantiene un solo timer di ritrasmissione per l'intera coda.
  - » Se riceve un riscontro, rimuove il relativo segmento o segmenti dalla coda e ri-inizializza il timer.
  - » Se il tempo si esaurisce, ritrasmette tutti i segmenti nella coda e ri-inizializza il timer.
  - – Riduce i ritardi, ma può dare origine a ritrasmissioni inutili (va bene per implementare un GO-BACK-N)
- **Individuale (di supporto al Sack):**
  - » Mantiene un timer differente per ogni segmento nella coda
  - » Se riceve un riscontro, rimuove il relativo segmento o segmenti dalla coda ed elimina il timer o i timer corrispondenti.
  - » Se il tempo di un qualunque timer si esaurisce, ritrasmette il segmento corrispondente e rinizializza quel timer.

8.68

Transmission Control Protocol (TCP)
Controllo di congestione

- Il meccanismo TCP di controllo di flusso basato sui crediti è stato esteso per realizzare anche un controllo di congestione sorgente -destinazione.
- La congestione ha due effetti principali:
  - come inizia la congestione, il tempo di transito lungo la rete aumenta.
  - come la congestione diventa pesante, vengono persi pacchetti o segmenti
- Il meccanismo di controllo di flusso del TCP può servire per riconoscere l'inizio della congestione (osservando l'incremento dei ritardi e dei segmenti persi) e per reagire riducendo il flusso di dati.

8.69

Transmission Control Protocol (TCP)
Controllo di congestione

- La prima cosa da fare è stimare il RRT (*round-trip-time*)
  - Media semplice

Segmento i-esimo

$$\mathrm{ARTT}(K+1) = \frac{1}{K+1}\sum_{i=1}^{K+1}\mathrm{RTT}(i)$$

RRT medio (*Average*)

$$\mathrm{ARTT}(K+1) = \frac{K}{K+1}\mathrm{ARTT}(K) + \frac{1}{K+1}\mathrm{RTT}(K+1)$$

Si noti che ad ogni termine nella sommatoria viene dato lo stesso peso; ovvero, ogni termine è moltiplicato per la stessa costante $1/(K + 1)$.

8.70

Transmission Control Protocol (TCP)
Controllo di congestione

- Media esponenziale: sarebbe meglio dar maggior peso ai valori più recenti, in quanto rappresentano con più probabilità il comportamento futuro. Una tecnica comune di predizione, basata su un certo numero di valori passati, è la media esponenziale:

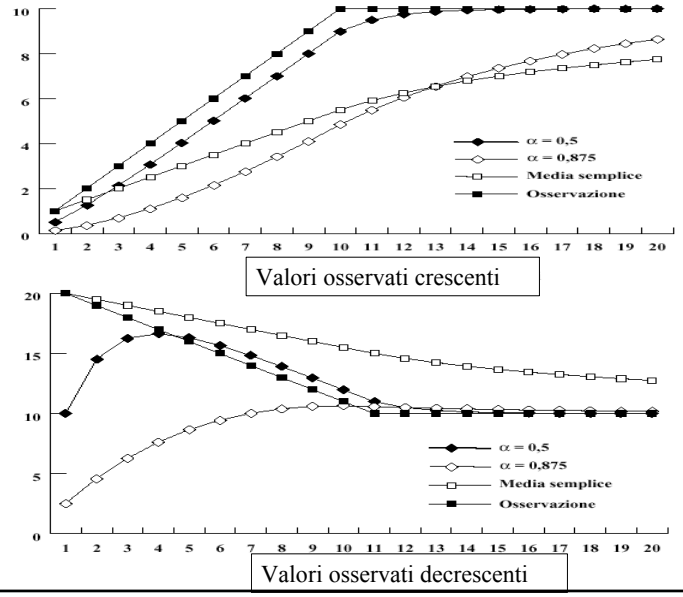
$$\mathrm{SRTT}(K+1) = \alpha \times \mathrm{SRTT}(K) + (1-\alpha) \times \mathrm{RTT}(K+1)$$

oppure

$$\mathrm{SRTT}(K+1) = (1-\alpha) \times \mathrm{RTT}(K+1) + \alpha(1-\alpha) \times \mathrm{RTT}(K) +$$
$$\alpha^2(1-\alpha) \times \mathrm{RTT}(K-1) + \ldots + \alpha^K(1-\alpha) \times \mathrm{RTT}(1)$$

8.71

Transmission Control Protocol (TCP)
Controllo di congestione



8.72

Transmission Control Protocol (TCP)
Controllo di congestione

- Il timeout (***Retrasmission TimeOut***, RTO) può essere calcolato come:

Costante additiva

$$\mathrm{RTO}(K+1) = \mathrm{SRTT}(K+1) + \Delta$$

- In questo caso però Δ non è legato a SRTT, quindi una formulazione più opportuna (usata nella versione originale del TCP) potrebbe essere

$$\mathrm{RTO}(K+1) = \mathrm{MIN}(\mathrm{UBOUND}, \mathrm{MAX}(\mathrm{LBOUND}, \beta \times \mathrm{SRTT}(K+1)))$$

>1

8.73

Transmission Control Protocol (TCP)
Controllo di congestione

- La varianza nella rete può essere elevata, ma specialmente può variare nel tempo.
- Quando la varianza è bassa, un valore alto di β sovrastima RTO e quindi in caso di pacchetto perso si attende più tempo del dovuto
- Ma quando la varianza è alta il valore β = 2 può non essere comunque sufficiente ad evitare ritrasmissioni inutili.
- Per cui è stato proposta una stima della varianza dell'RTT (algoritmo di Jacobson)

8.74

Transmission Control Protocol (TCP)

Controllo di congestione

- La stima della deviazione standard viene fatta in modo analogo a quello della media:

$$\mathrm{SRTT}(K+1) = (1-g)\times \mathrm{SRTT}(K) + g \times \mathrm{RTT}(K+1) \qquad g = 1/8$$

$$\mathrm{SERR}(K+1) = \mathrm{RTT}(K+1) - \mathrm{SRTT}(K)$$

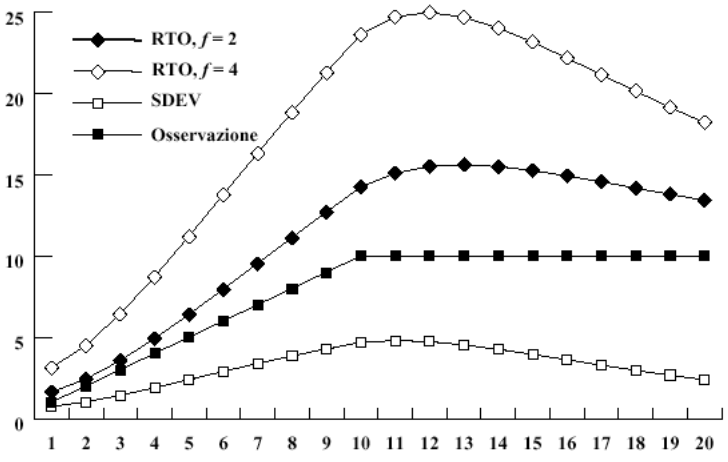
$$\mathrm{SDEV}(K+1) = (1-h)\times \mathrm{SDEV}(K) + h \times |\,\mathrm{SERR}(K+1)\,| \qquad h = 1/4$$

$$\mathrm{RTO}(K+1) = \mathrm{SRTT}(K+1) + f \times \mathrm{SDEV}(K+1) \qquad f = 4$$

8.75

Transmission Control Protocol (TCP)

Controllo di congestione

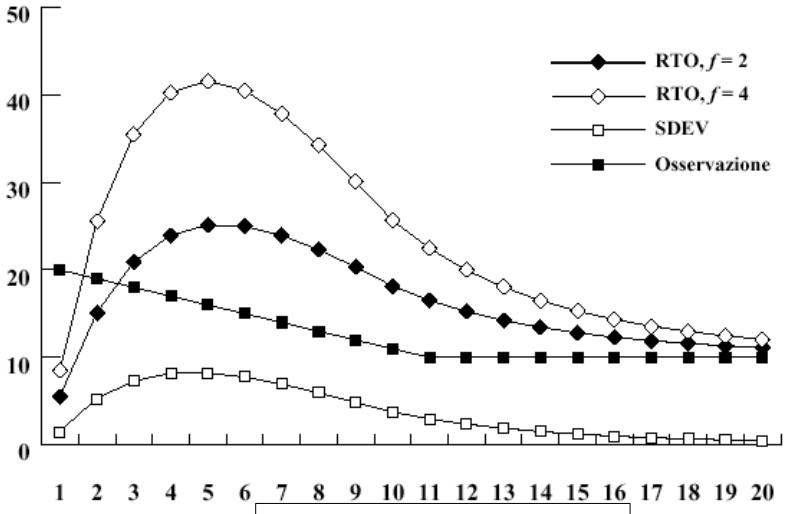


Valori osservati crescenti

8.76

Transmission Control Protocol (TCP)

Controllo di congestione



8.77

Transmission Control Protocol (TCP)

Controllo di congestione

- Bisogna considerare altri due fattori:
  - Quale valore RTO si deve usare per un segmento ritrasmesso?
    - » A questo scopo viene usato un algoritmo di *backoff* esponenziale dell'RTO.
  - Quali campioni di tempo di andata e ritorno si dovrebbe usare come ingresso all'algoritmo di Jacobson (in relazione a segm. ritrasmessi) ?
    - » L'algoritmo di Karn determina quali campioni usare.

8.78

Transmission Control Protocol (TCP)
Controllo di congestione

- Una prima ipotesi è usare il RTO precedente.
- Una politica più ragionevole suggerisce che una sorgente TCP aumenti il proprio RTO ogni volta che un segmento viene ritrasmesso (perché ipotizza congestione); questo meccanismo viene chiamato **algoritmo di *backoff***. In particolare, ad ogni ritrasmissione viene posto

$$\text{RTO} = q \times \text{RTO}$$

- Il valore più frequentemente usato per q è 2, da cui il nome associato a questa tecnica di *backoff* esponenziale binario (come nel CSMA/CD)

8.79

Transmission Control Protocol (TCP)
Controllo di congestione

- Quando arriva un ACK di un segmento ritrasmesso, il trasmettitore può interpretarlo in due modi:
  - Si tratta dell'ACK relativo alla prima trasmissione del segmento. In questo caso, l'RTT è semplicemente più lungo del previsto ma riflette l'effettive condizioni di rete.
  - Si tratta l'ACK relativo alla seconda ritrasmissione.
- Non potendo riconoscere le due situazioni si è scelto di ignorare l'informazione.

8.80

Transmission Control Protocol (TCP)
Controllo di congestione

- Quindi **l'algoritmo di Karn** aggiunge le seguenti regole:
  - Non usare l'RTT, misurato su di un segmento ritrasmesso, per aggiornare SRTT e SDEV.
  - Quando ha luogo una ritrasmissione, calcolare l'RTO usando la procedura di *backoff*.
  - Continuare ad usare la procedura di *backoff* per il calcolo dell'RTO nei successivi segmenti, fino a che non arriva un riscontro di un segmento che non sia stato ritrasmesso.
  - Quando viene ricevuto un riscontro di un segmento non ritrasmesso, l'algoritmo di Jacobson è di nuovo attivato per il calcolo dei futuri valori di RTO.

8.81

Transmission Control Protocol (TCP)
Controllo di congestione

- Gestione della finestra
  - Il TCP usa un meccanismo a credito per gestire la finestra di trasmissione
  - Le modalità di crescita e decrescita della finestra vengono però controllate opportunamente in relazione alla congestione
  - In particolare si ha che

    $$awnd = \text{MIN}[credito, cwnd]$$

  - Dove
    - » ***awnd*** = finestra concessa, in segmenti.
    - » ***credito*** = la quantità di credito accordato nel riscontro più recente, in segmenti
    - » ***cwnd*** = finestra di congestione, in segmenti

8.82

Transmission Control Protocol (TCP)
Controllo di congestione

- In fase di apertura della connessione, la scelta della finestra e critica, perché non si hanno informazioni sullo stato della rete.
- Allora si attua lo ***slow start*** (partenza lenta)
  - inizializza *cwnd* = 1 segmento
  - Per ogni riscontro ricevuto si aumenta *cwnd* di uno
- Se tutti i segmenti vengono riscontrati la finestra raddoppia ogni RTT circa e quindi n realtà la crescita è esponenziale

8.83

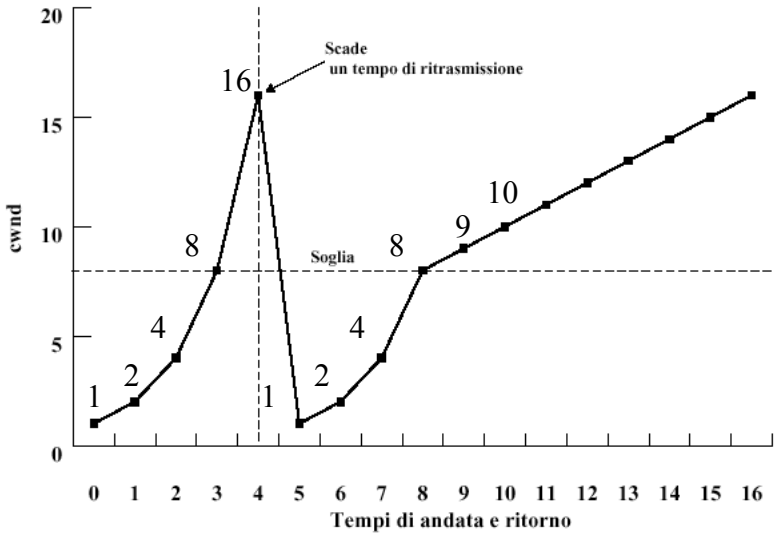
Transmission Control Protocol (TCP)
Controllo di congestione

- Come reagire in caso di congestione (identificata dallo scadere di un timer)?
- Siccome uscire della condizioni di congestione è difficile, quando si esaurisce il tempo di ritrasmissione di un segmento, si attua la seguente strategia (**TCP Tahoe**):
  - Si inizializza una soglia a partenza lenta pari a metà della finestra di congestione corrente; ovvero, si fissa *ssthresh* = *cwnd/2*.
  - Si pone *cwnd* = 1 e si procede con la tecnica a partenza lenta fino a che *cwnd* = *ssthresh* (*cwnd* è incrementato di 1 per ogni ACK ricevuto).
  - Per *cwnd* ≥ *ssthresh*, si aumenta *cwnd* di uno per ogni tempo di andata e ritorno.

8.84

Transmission Control Protocol (TCP)

Controllo di congestione



8.85

Transmission Control Protocol (TCP)

Controllo di congestione

- Per accelerare la identificazione di una perdita estemporanea di un segmento, alcune realizzazioni di TCP usano la strategia ***fast retrasmit***:
  - In corrispondenza dell'arrivo di tre ACK che confermano sempre lo stesso numero di sequenza, il trasmettitore ritrasmette un segmento con i dati successivi alla conferma senza attendere la scadenza del *timeout*.
  - Infatti, se un segmento viene perduto ma non i successivi, ogni segmento arrivato fuori sequenza comporta la generazione di un ACK con la conferma degli ultimi dati in sequenza ricevuti corretti; siccome in virtù del meccanismo a finestra i segmenti sono spesso inviati uno di seguito all'altro in gruppi, se quello perso non è l'ultimo, è facile che gli ACK arrivino prima dello scadere del timeout.

8.86

Transmission Control Protocol (TCP)

Controllo di congestione

- In presenza del *Fast-retrasmit*, si può attuare un meccanismo modificato per l'aggiornamento della finestra:
  - Nel caso in cui scada il *timeout*, il comportamento è quello del TCP Taohe, la finestra riparte da 1.
  - Nel caso in cui arrivino tre ack con lo stesso numero di sequenza, la fase di slow-start viene saltata, ponendo la finestra pari alla soglia *ssthresh* e cominciando subito l'incremento lineare.
  - La realizzazione di TCP che applica questo meccanismo si chiama "Reno".

8.87