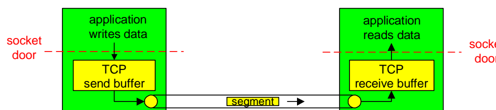
TCP: rassegna

RFCs: 793, 1122, 1323, 2018, 2581

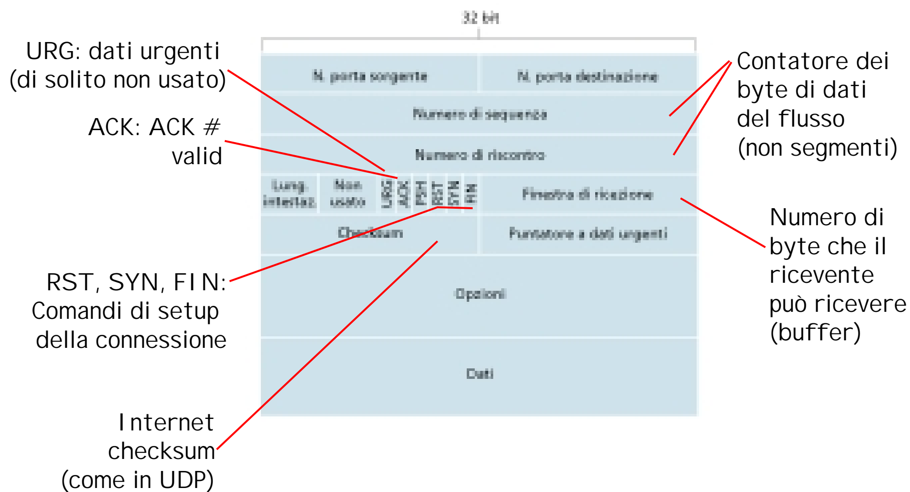
- Protocollo uno-a-uno:
  - Un sender, un receiver
- Flusso di Byte ordinato e affidabile
- Protocollo pipelined:
  - TCP ha controllo di flusso e di congestione basato su finestra scorrevole
  - Il protocollo è eseguito solo sui nodi terminali
- *Buffers su sender e receiver*



- Connessioni full-duplex:
  - Dati viaggiano nelle due direzioni
  - MSS: maximum segment size
- Orientato alla connessione:
  - Messaggi di controllo iniziali definiscono lo stato di sender e receiver prima di inviare i dati
- Controllo di flusso
  - Sender segue il ritmo del receiver
- Controllo di congestione
  - Sender segue il ritmo del router più lento

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

Struttura del segmento TCP



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP: numeri di sequenza e ACKs

Num. Seq. (#seq):

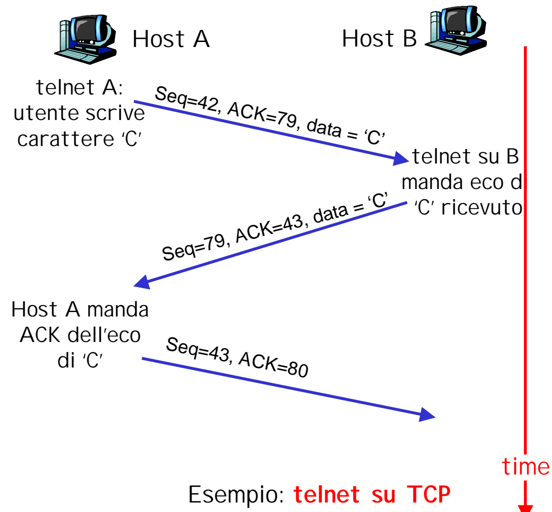
- Numero d'ordine del byte nella sequenza

ACKs:

- Numero d'ordine del prossimo byte atteso
- ACK cumulativo

??: come si gestiscono segmenti fuori ordine?

- TCP non lo specifica! Dipende...



Esempio: **telnet su TCP**

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP: reliable data transfer

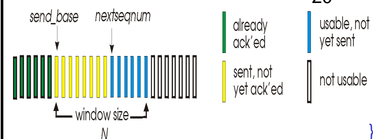
Assumiamo per semplicità che il Sender

- •Spedisca dati solo in un senso
- •Non abbia controllo di flusso e congestione

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP: trasferimento dati affidabile

Es. Sender TCP semplificato




```
00   sendbase = initial_sequence number (inizio finestra scorrevole)
01    nextseqnum = initial_sequence number (puntatore alla parte libera)
02
03     loop (ciclo infinito) {
04       A seconda del tipo di evento esegui:
05       evento: ricevuti dati da spedire dall'applicazione
06             crea segmento TCP con numero nextseqnum
07             Avvia timer per il segmento nextseqnum
08             Spedisci segmento mediante servizio di livello rete (IP)
09             nextseqnum = nextseqnum + length(data)  (aggiorna finestra)
10        evento: scatta timer per segmento numero y
11             Ritrasmetti segmento numero y
12             Calcola nuovo timeout per il timer del segmento y
13             Riavvia il timer per il segmento y
14        evento: ricevuto ACK per il segmento numero y
15             se (y > sendbase) { /* ACK cumulativo per tutti i byte fino a y */
16                 Cancella tutti i timer per i segmenti con numero < y
17                  sendbase = y (aggiorna il lato di inizio della finestra)
18                  }
19             altrimenti { /* ACK duplicato per segmento già ricevuto */
20                  Incrementa contatore di ACK duplicati del segmento y
                    se (questo è il terzo ACK duplicato per segmento y) {
                       /* attiva fase "fast retransmit" di TCP*/
                       Rispedisci segmento con numero y
                       Riavvia timer per segmento y
                 }
       }  /* fine ciclo infinito */
```

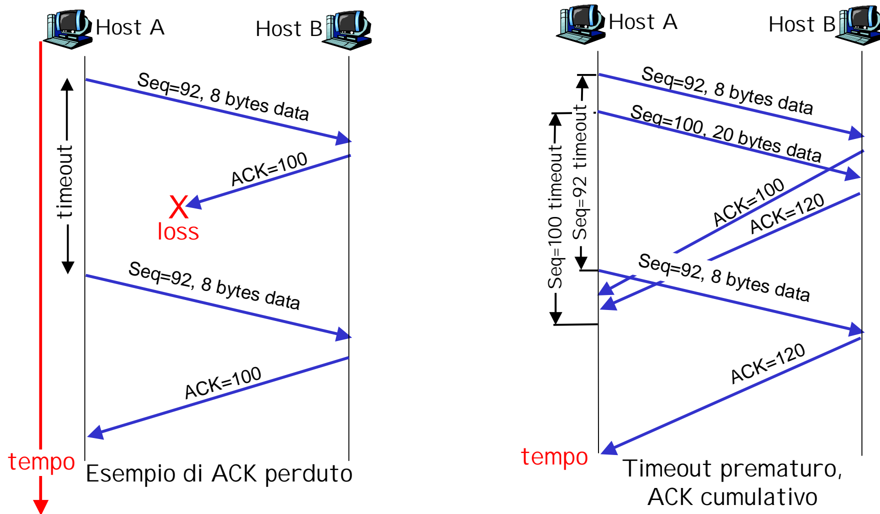
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP: regole per ACK [RFC 1122, RFC 2581]

Evento	Azione del receiver TCP
Segmento arriva in ordine Non ci sono "buchi"	ACK ritardato. Aspetta per 500ms l'arrivo anche del prossimo segmento. Se non arriva entro 500ms spedisci ACK
Segmento arriva in ordine Non ci sono "buchi" Ho già un ACK ritardato...	Spedisci subito un ACK cumulativo per i due segmenti ricevuti
Segmento arriva fuori ordine con numero superiore. Si crea un "buco".	Spedisci un ACK duplicato che indica di nuovo quale è il numero del prossimo byte atteso
Arriva segmento che copre del tutto o parzialmente un "buco"	Spedisci ACK immediato se il segmento cade all'inizio del "buco"

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP: esempi



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

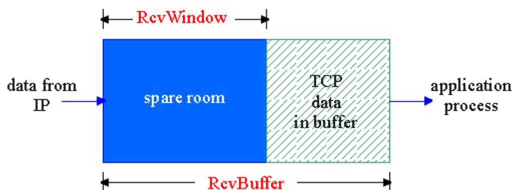
TCP controllo di flusso

controllo di flusso

Il sender evita di trasmettere dati troppo velocemente rispetto al buffer del receiver

`RcvBuffer` = dimensione del Buffer TCP del receiver

`RcvWindow` = dimensione del Buffer libero sul receiver



receiver buffer

receiver: informa esplicitamente il sender sulla quantità residua du buffer disponibile

- nel campo **RcvWindow** del segmento TCP

sender: limita il numero di segmenti per i quali non ha ricevuto ACK inferiore all'ultimo **RcvWindow ricevuto**

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP Round Trip Time e Timeout

D: come si definisce il valore del timeout?
- Maggiore di RTT
  - ma: RTT può variare
- Troppo corto: timeout prematuro
  - Implica ritrasmissione inutile
- Troppo lungo: reazione lenta in caso di perdita di segmento

D: come stimare RTT?
- **SampleRTT**: tempo misurato dalla trasmissione del segmento alla ricezione dell'ACK
  - Si ignorano le ritrasmissioni
  - Valgono gli ACK cumulativi
- **SampleRTT** può variare rapidamente: occorre un modo per rendere "morbida" la variazione
  - Si usa una media pesata delle ultime stime di **SampleRTT** sperimentate

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP Round Trip Time and Timeout

```
EstimatedRTT = (1-x)*EstimatedRTT + x*SampleRTT
```

- Medie mobili pesate (esponenziali)
- L'effetto di una singola stima decade nel tempo esponenzialmente
- Valore tipico di x: 0.1

Come si definisce quindi il timeout?

- **EstimtedRTT** al quale si aggiunge un margine di sicurezza
- se **EstimatedRTT varia molto ->** occorre un margine di sicurezza superiore

```
Timeout = EstimatedRTT + 4*Deviation

Deviation = (1-x)*Deviation +
            x*|SampleRTT-EstimatedRTT|
```

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP: instaurazione di una connessione

N.B.: TCP sender e receiver devono stabilire una connessione prima di scambiare segmenti

- Per inizializzare le variabili di TCP:
  - Numeri di sequenza #s
  - Buffer e controllo di flusso (es. **RcvWindow**)
- *client:* chi inizia la connessione

```
Socket clientSocket = new
Socket("hostname","port
number");
```

- *server:* chi riceve la richiesta

```
Socket connectionSocket =
welcomeSocket.accept();
```

Handshake a 3 vie:

passo 1: client spedisce segmento TCP SYN al server
- specifica num.seq. iniziale

passo 2: appena il server riceve SYN, risponde con SYNACK
- Conferma ricezione SYN
- Alloca spazio per buffer
- Specifica il seq.num. del receiver

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

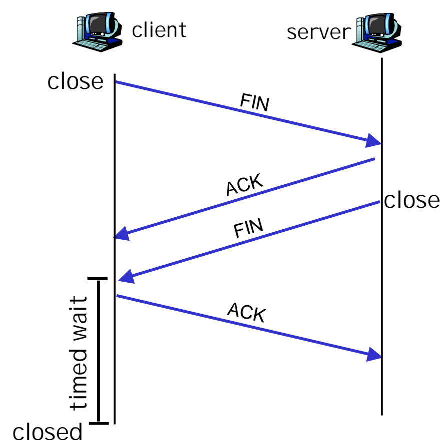
TCP: gestione della connessione

Chiusura della connessione:

client closes socket:
clientSocket.close();

Passo 1: client spedisce segmento TCP FIN al server

passo 2: server riceve FIN e conferma con ACK, poi chiude la connessione e spedisce segmento FIN a sua volta.



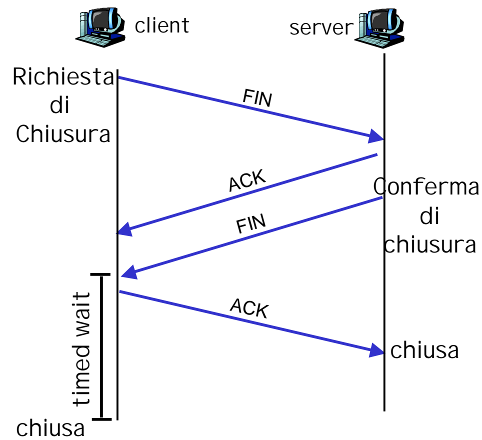
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP: gestione della connessione

passo 3: client riceve FIN e risponde con ACK.

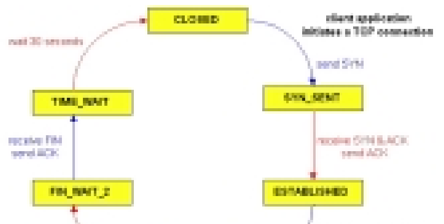
- ...poi entra in fase di attesa limitata nella quale risponde con ACK a eventuali FIN replicati

passo 4: server, riceve ACK e la connessione è chiusa.



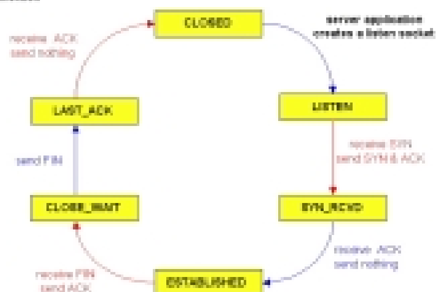
Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP: gestione della connessione



Ciclo di vita degli stati del client TCP

Ciclo di vita degli stati del server TCP



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

Il controllo della congestione

Congestione:

- Troppi host spediscono troppi dati e la rete non è in grado di inoltrarli tutti verso le destinazioni
- È un problema dei router intermedi del cammino (e non solo del receiver, come per il controllo di flusso)
- Cosa causa:
  - Pacchetti perduti (buffer dei router saturi)
  - Lunghi ritardi (lunghe code nei buffer)

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

Come si realizza il controllo di congestione

Due approcci possibili:

Controllo di congestione End-end:

Non c'è indicazione esplicita dalla rete

- La congestione si interpreta sulla base dei pacchetti perduti e dei ritardi stimati
- TCP usa questa tecnica

Controllo di congestione assistito dalla rete

- I router forniscono segnali di rischio
  - Un bit nell'header indica presenza o rischio di congestione (explicit congestion notification) TCP/IP ECN
  - Si indica anche il ritmo di invio che il sender dovrebbe assumere

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP: controllo della congestione

- Controllo end-end (non assistito dai router intermedi della rete)
- Il ritmo di invio è limitato dalla dimensione della finestra di congestione **Congwin**:

- Dati w segmenti, di MSS bytes spediti ogni RTT, posso raggiungere un throughput massimo di:

$$\text{throughput} = \frac{w * MSS}{RTT} \text{ Bytes/sec}$$

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP: controllo della congestione

- "probing" della banda disponibile
  - idea: inviare al ritmo massimo (**Congwin** più grande possibile) evitando perdita segmenti
  - *incrementa* **Congwin** finchè non c'è perdita
  - *decrementa* **Congwin** se c'è perdita e poi riparti con la fase di incremento

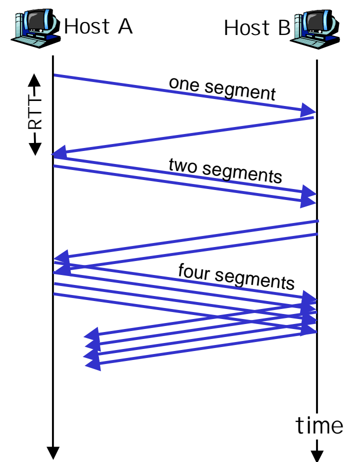
- Due fasi
  - slow start
  - congestion avoidance
- variabili usate:
  - **Congwin (dimensione della finestra)**
  - **threshold:** definisce la soglia di dimensione della finestra oltre la quale termina la fase "slow start" e inizia "congestion avoidance"

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP Slow(!?)start

Algoritmo Slowstart

inizializza: Congwin = 1
Per ogni ACK ricevuto
    Congwin*2
finchè (ACK non ricevuto
oppure
    CongWin > threshold)



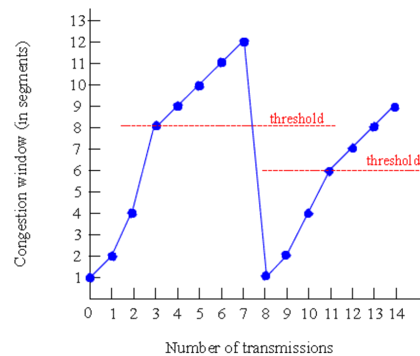
- Crescita esponenziale (per RTT) della finestra (quindi non troppo LENTA!!)

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

TCP "Congestion Avoidance"

Congestion avoidance

/* slowstart terminata */
/* Congwin > threshold */
finchè (non perde segm.) {
  ogni w ACK ricevuti:
    Congwin++
  }
/* se perde ACKs */
threshold = Congwin/2
Congwin = 1 [1]
Riparti da fase "slowstart"



1: TCP Reno evita slowstart (fase di fast recovery) dopo 3 ACK duplicati (ok per reti Wireless).

Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)

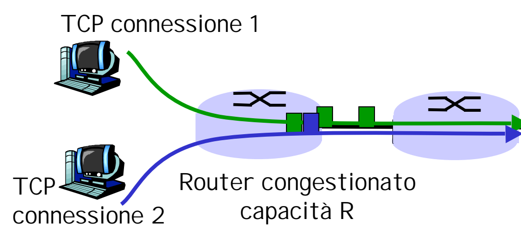
AIMD

La fase "congestion avoidance" di TCP si dice

- AIMD: *additive increase, multiplicative decrease*
  - Xchè incrementa finestra di 1 per RTT
  - Divide la finestra per 2 in caso di perdita segmento

TCP Fairness

Fairness: se N connessioni TCP condividono lo stesso router, ogni connessione dovrebbe ricevere 1/N della capacità del router (e del link)



Copyright © Luciano Bononi 2004 (some figure credits to Kurose, Ross, Internet e reti di calcolatori)