



Il livello trasporto

Il protocollo TCP

Ing. Nadia Ranaldo

1



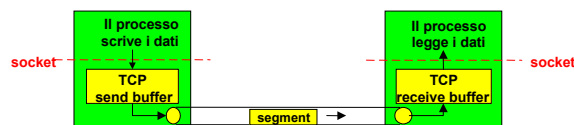
TCP - Transmission Control Protocol

- **Orientato alla connessione**
 - È necessario attivare una connessione (scambio di messaggi di controllo) prima di iniziare a trasmettere dati
 - Protocollo handshake a tre vie
 - Lo stato della connessione risiede sui sistemi terminali (i router non conoscono lo stato della connessione)
- **Affidabile - stream di byte ordinato**
 - Non ci sono confini di messaggio
- **End-to-end**
 - Un mittente, un ricevente
- **Trasmissione full duplex**
 - Una stessa connessione consente un flusso di dati bi-direzionale
- **Controllo del flusso**
 - Il mittente non sovraccarica il ricevente
- **Pipelined**
 - Protocollo simile al go-back-N, inoltre la dimensione della finestra è legata al controllo della congestione e del flusso
 - Utilizzo di buffer dal lato mittente e ricevente

2



MSS: Maximum Segment Size

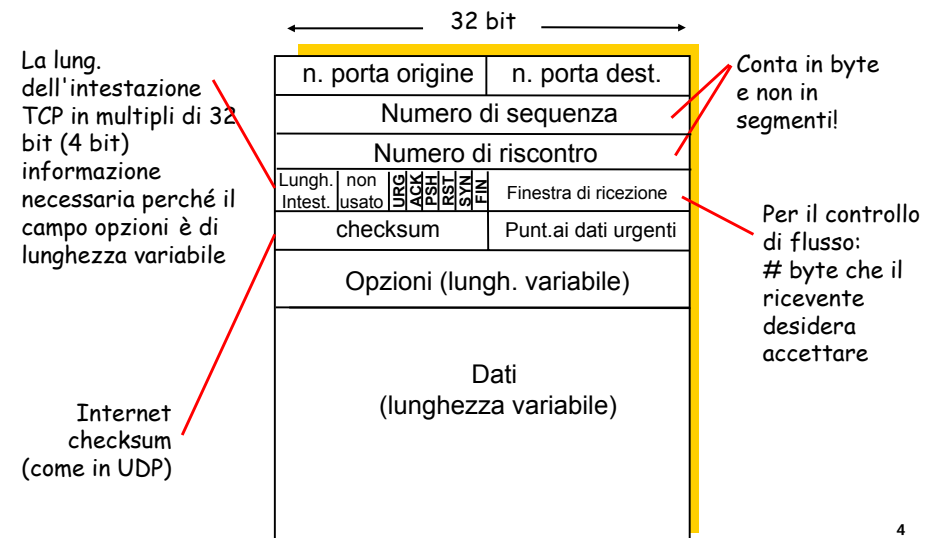


- Quando il processo client invia tramite una socket i dati, questi vengono inseriti in un buffer dal software TCP (**send buffer**)
- Periodicamente, l'entità mittente del TCP estrae un certo numero di byte dal buffer e costruisce i segmenti TCP
- Il numero di byte inseriti in un segmento è limitato dalla **MSS**
 - Generalmente si considera la lunghezza del frame più grande a livello data link che può essere inviato dall'host mittente (**MTU - maximum transmission unit**) e poi si sceglie MSS in modo che il segmento TCP sia incapsulato in un singolo frame a livello data link
- Valori tipici di MSS sono 1460, 536 e 512 byte
- Lo scopo della MSS è di evitare che il livello IP frammenti i segmenti TCP per inviare i datagrammi su rete
- Quando l'entità ricevente del TCP riceve un segmento, lo inserisce nel buffer di ricezione (receive buffer)
- Ogni lato della connessione ha un buffer d'invio e di ricezione

3



Struttura dei segmenti TCP (1)



4

Struttura dei segmenti TCP (2)

Opzioni

- Facoltativo, utilizzato per fornire funzionalità aggiuntive, che non sono fornite mediante i campi regolari dell'intestazione
- L'opzione più importante è quella che permette ad un mittente e un ricevente di negoziare la dimensione massima del segmento (MSS)
 - Se non usata, si suppone di 536 byte

Puntatore urgente

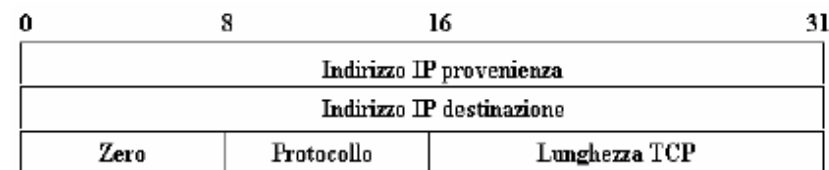
- Il TCP permette la trasmissione **fuori banda** di dati ad alta priorità
- Ad esempio, quando TCP è utilizzato per una sessione di accesso remoto, l'utente può decidere di inviare una sequenza da tastiera che interrompe o termina il processo dall'altro lato (CTRL-C)
- Questi devono essere consegnati al ricevente il prima possibile, indipendentemente dalla loro posizione nello stream
- Questo campo, se valido, conterrà un puntatore che indica **la posizione nel segmento** in cui terminano i dati urgenti
- Il protocollo specifica che quando vengono trovati dati urgenti, il TCP destinatario deve avvisare il processo applicativo di entrare in modalità urgente in cui legge tutti i dati urgenti, ma non specifica come

5

Struttura dei segmenti TCP (3)

Checksum

- Campo di 16 bit contenente un valore intero utilizzato dal TCP (UDP) della macchina host di destinazione, per verificare l'integrità dei dati e la correttezza dell'intestazione
- Informazione di essenziale importanza perché il protocollo IP non prevede nessun controllo di errore sulla parte dati del datagramma
- Per il calcolo della checksum il TCP ha bisogno di aggiungere una **pseudointestazione** al segmento, per effettuare così **un controllo anche sugli indirizzi IP di destinazione e provenienza**
 - Campo Protocollo = 6 per il TCP



6

Struttura dei segmenti TCP (4)

Nel segmento TCP sono presenti 6 bit di flag

URG

- Indica la presenza di dati urgenti. Se posto a 1 il campo puntatore ai dati urgenti conterrà la posizione dell'ultimo byte di dati urgenti (poco usato)

ACK

- Indica se il campo numero di riscontro è valido, ossia il segmento contiene un riscontro

PSH

- Se posto a 1 indica che il destinatario dovrebbe immediatamente inviare i dati al livello applicativo e non bufferizzarli (poco usato)

RST

- Re-inizializza una connessione diventata instabile
- Rifiuta un segmento non valido o l'apertura di una connessione

SYN

- Usato per creare connessioni

FIN

- Utilizzato per chiudere una connessione
- Specifica che il mittente non ha più dati da trasmettere

7

Numeri di sequenza e di ack

di sequenza:

- Posizione relativa all'origine dello stream del 1° byte nei dati di un segmento

di ACK:

- # di sequenza del successivo byte atteso da chi riscontra
- Uso di ACK cumulativi
- Cosa fare con segmenti non in ordine? Scelta implementativa, tipicamente vengono **bufferizzati** per efficienza in termini di banda

Esempio. MSS vale 1000, primo byte del flusso è numerato con 0
File da 500.000 byte



500 segmenti. I numeri di sequenza sono: 0 per il 1°, 1000 per il 2°, etc.

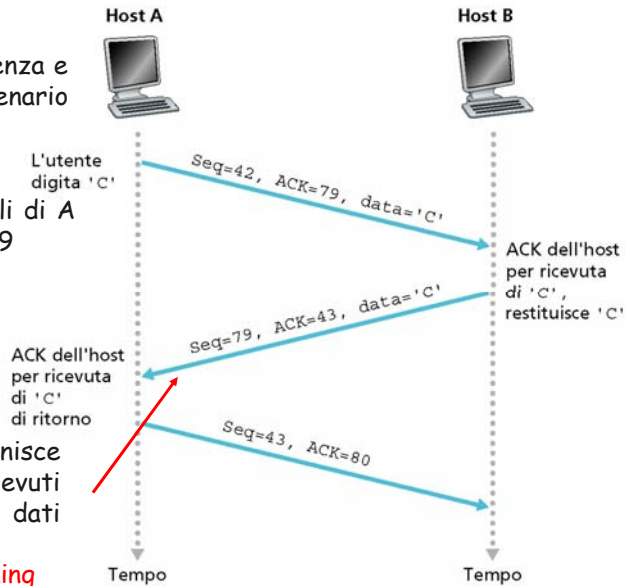
8

Gestione di n. di sequenza e di riscontro

Impiego di n. di sequenza e di ACK in uno scenario tipico di Telnet

N. di sequenza iniziali di A e B, sono risp. 42 e 79

Duplica scopo: fornisce riscontro dei dati ricevuti (ACK=43) invia dati (Seq=79)
Tecnica del piggybacking



Scelta del timeout nel TCP (1)

- Q:** in che modo deve essere scelto il timeout?
- Valore dinamico
 - Più grande dell'RTT
 - ma RTT varia
 - Se è troppo breve: timeout prematuri
 - Ritrasmissioni non necessarie
 - Se è troppo lungo: reazione troppo lenta alla perdita dei segmenti

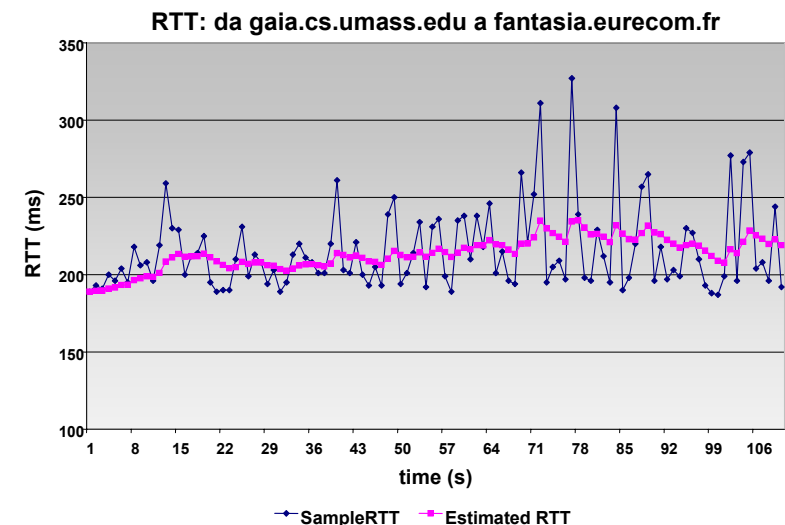
- Q:** In che modo stimare RTT?
- SampleRTT:** tempo misurato dalla trasmissione di un segmento e la ricezione dell'ACK
 - Si ignorano le ritrasmissioni
 - SampleRTT** varia tra una spedizione e la successiva
 - E' necessario un modo per stimare il successivo RTT
 - Si utilizza una media su più misure recenti (non soltanto il valore corrente di **SampleRTT**)

Scelta del timeout nel TCP (2)

$$\text{EstimatedRTT}_i = (1 - \alpha) * \text{EstimatedRTT}_{i-1} + \alpha * \text{SampleRTT}_i$$

- Media esponenziale mobile pesata**
- Esponenziale: l'influenza di un campione decresce esponenzialmente
- Le variazioni di **SampleRTT** vengono "ammorbidite" nel calcolo di **EstimatedRTT**
- Valore tipico: $\alpha = 0.125$ (1/8)
 - Aumentare α significa dare maggiore importanza ai campioni recenti rispetto a quelli vecchi
 - I recenti campioni riflettono meglio la situazione attuale della rete

Esempio di stima di RTT



Scelta del timeout nel TCP (3)

Scelta del timeout:

- EstimatedRTT più un margine di sicurezza
- Maggiore margine quando c'è molta fluttuazione nei valori di SampleRTT (ovvero molta differenza tra SampleRTT e EstimatedRTT)

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

- DevRTT è una stima della **varianza**, ovvero di quanto SampleRTT generalmente si discosta da EstimatedRTT:

$$\text{DevRTT}_i = (1 - \beta) * \text{DevRTT}_{i-1} + \beta * |\text{SampleRTT}_i - \text{EstimatedRTT}_i|$$

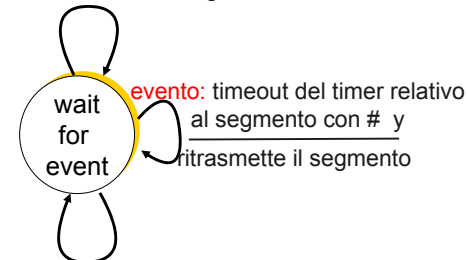
(tipicamente, $\beta = 0.25$)

- Se i valori di SampleRTT presentano variazioni limitate, allora DevRTT sarà piccolo, e viceversa

13

Gestione dell'affidabilità

evento: dati ricevuti dal livello superiore (applicazione)
crea ed invia un segmento



evento: ricezione ACK con numero y
elaborazione ACK

Ipotesi semplificative:

1. Trasferimento in una sola direzione
2. Nessun meccanismo di controllo del flusso e della congestione
3. I dati dal livello applicazione hanno dimensione inferiore a MSS

Commenti:

- Timeout calcolato con l'algoritmo descritto precedentemente

14

Gestione dell'affidabilità in TCP

```
sendbase = initial_sequence number
nextseqnum = initial_sequence number
```

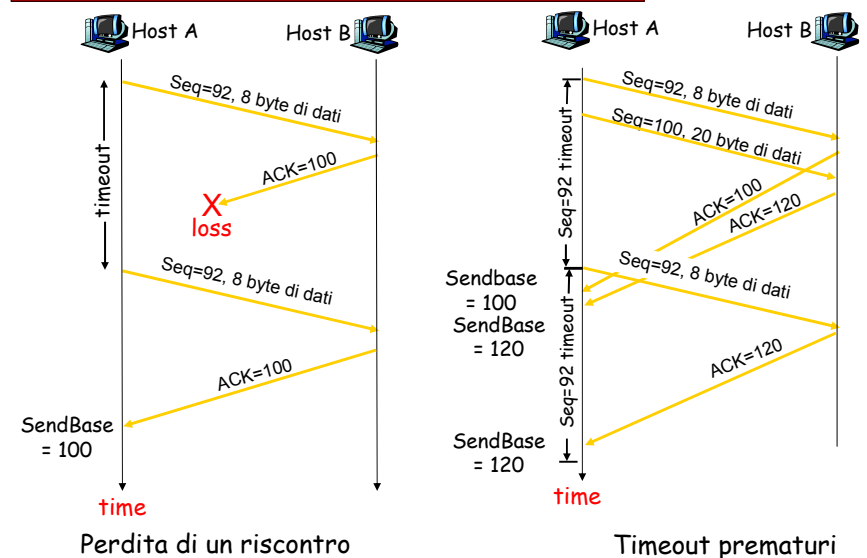
```
loop (forever) {
  switch(event) {
    evento: dati ricevuti dall'applicazione
    crea il segmento TCP con numero di sequenza nextseqnum
    if (il timer attualmente non è in funzione)
      Avvia il timer per il segmento nextseqnum
    Passa il segmento a IP
    nextseqnum = nextseqnum + length(data)
    break;
    evento: timeout del timer
    Ritrasmette il segmento non ancora riscontrato
    con il più piccolo numero di sequenza
    riavvia il timer
    break;
    evento: ACK ricevuto, con valore del campo ACK pari a y
    if (y > sendbase) { /* ACK cumulativo di tutti i dati fino a y */
      sendbase = y
      if (esistono attualmente segmenti non ancora riscontrati)
        riavvia il timer
    }
    break;
  } /* fine del loop */
}
```

Go-Back-N
non puro

Utilizzo di
un unico timer
associato al più
vecchio
segmento non
riscontrato

15

Alcuni casi di ritrasmissione (1)

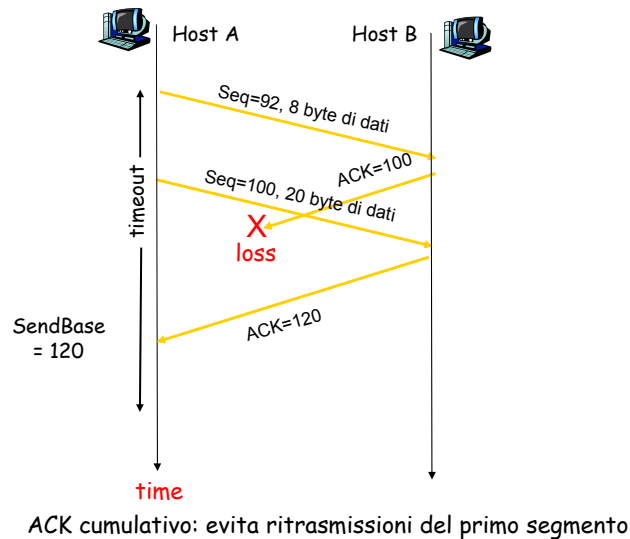


Perdita di un riscontro

Timeout prematuri

16

Alcuni casi di ritrasmissione (2)



17

Raddoppio dell'intervallo di timeout

- Lo scadere di un timer viene probabilmente causata dalla congestione della rete
- Nei periodi di congestione, se i mittenti continuano a trasmettere (ritrasmettere) pacchetti, la congestione può peggiorare
- Nel TCP si può introdurre un primo meccanismo di controllo della congestione forzando il mittente a ritrasmettere ad intervalli sempre più lunghi finché non si riesce a trasmettere il segmento senza timeout
- Dopo lo scadere di un timer, la maggior parte delle implementazioni del TCP re-impostano il valore del timeout a:

$$\text{timeout} = \lambda * \text{timeout}$$
 - Il valore tipico di λ è 2 => doppio del valore precedente
 - Gli intervalli crescono esponenzialmente ad ogni ritrasmissione
 - Algoritmo di Karn con tecnica del **backoff del timer** (riportare indietro)
- Per l'invio di nuovi segmenti e quando viene ricevuto un ACK per un segmento senza ritrasmissione il TCP aggiorna la stima dell'RTT e aggiorna il timeout

18

Controllo del flusso in TCP (1)

ricevente:

- comunica esplicitamente e dinamicamente l'ammontare di spazio di buffer libero
- Campo **finestra di ricezione** nel segmento TCP

mittente:

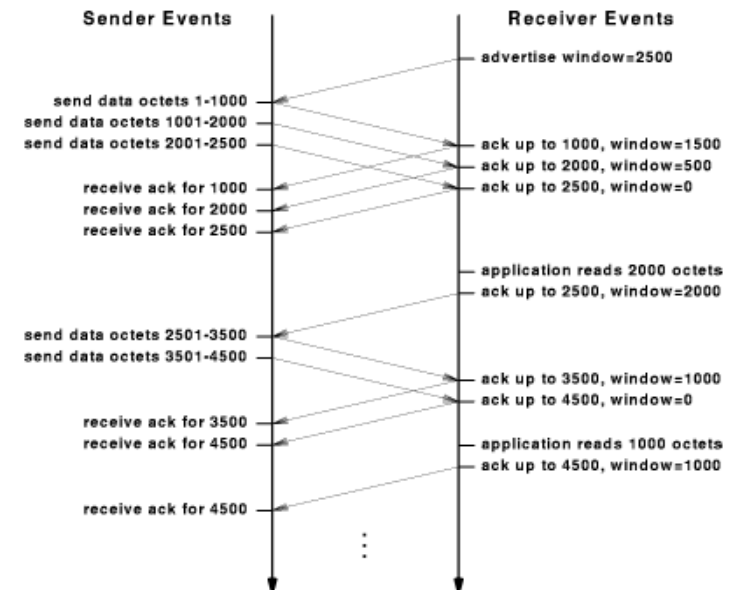
- la quantità di byte trasmessi e di cui non si è ricevuto un riscontro deve essere inferiore all'ultima **finestra di ricezione** ricevuta
- $$\text{nextseqnum} - \text{sendbase} \leq \text{RcvWindow}$$



Il mittente non sovraccarica il ricevente trasmettendo molto e troppo velocemente

19

Controllo del flusso in TCP (2)



20

Eccesso di piccoli pacchetti

- Cosa succede se le operazioni di lettura da parte del processo ricevente e quelle di scrittura da parte di quello mittente sono lente?
 - L'appl. mittente è lenta, invia segmenti con pochi byte di dati
 - L'appl. ricevente è lenta, riscontra pochi byte alla volta
 - Annuncia una piccola finestra disponibile
 - Ad esempio per una connessione Telnet, nel caso peggiore, appena l'utente digita un carattere, il TCP invia un segmento (20 + 20 + 1 = 41 byte) e così via, e il ricevente invia un riscontro per ogni byte ricevuto (20 + 20 + 1 = 41) e così via
- Meccanismi per risparmiare banda
 - Mittente. Soluzione → **algoritmo di Nagle**
 - Ricevente. Soluzione → **algoritmo di Clark, riscontro ritardato**

21

Soluzione lato mittente

- **Algoritmo di Nagle (1984)**
 - Il mittente invia subito il primo byte di dati disponibile
 - Quindi accumula i dati successivi nel buffer in modo da inviarli insieme in un unico segmento TCP
 - Fino a raggiungere la dimensione di un MSS o metà della finestra del ricevente
 - ... o fino alla ricezione di un ACK da parte del ricevente
 - Tale algoritmo garantisce che una connessione TCP non può presentare più di un segmento piccolo di cui non si è ricevuto il riscontro
 - Segmento *piccolo* significa: "di dimensione inferiore a MSS"
 - Per alcune applicazioni occorre disabilitare l'algoritmo di Nagle per aumentare l'interattività
 - In Java `setTcpNoDelay(false)`

22

Soluzione lato ricevente

- Situazione in cui il buffer del ricevente è pieno e l'applicazione legge un byte alla volta, liberando dal buffer un byte alla volta
 - Il mittente, di conseguenza, invia un byte alla volta
- **Soluzione di Clark (1982)**
 - Il **riscontro** inviato dal ricevente contiene un valore **zero** per `rcvWindow` finché lo spazio libero nel buffer sarà minore di $\min(MSS, \text{RecvBuffer}/2)$
- Soluzione mediante **ritardo del riscontro**
 - Il **ricevente non invia** un riscontro per ogni segmento ricevuto
 - Ritarda la spedizione del riscontro fino a che nel buffer non vi è uno spazio ragionevole
 - Questa tecnica consente anche di ridurre il traffico su rete
 - La soluzione di Clark potrebbe richiedere due segmenti (uno di riscontro e uno per aggiornare la dimensione della finestra)
 - ma può determinare la rispeditura di segmenti e una stima di RTT troppo alta
 - Lo standard pone il limite di attesa a 500 ms
 - Per garantire che TCP riceva un sufficiente numero di stime di RTT, lo standard consiglia di inviare una conferma di ricezione almeno ogni due segmenti

23

rcvWindow == zero

- Cosa accade se **rcvWindow** vale zero?
 - Il ricevente aggiorna la finestra quando l'applicazione legge i dati dal buffer
 - Se il ricevente non ha nulla da inviare al mittente, quest'ultimo non verrà informato che si è liberato spazio nel buffer
 - Cosa succede, inoltre, se il segmento contenente l'aggiornamento viene perso?
- **Timer di persistenza**
 - Allo scadere del timer il mittente invia periodicamente pacchetti di dimensione 1 byte
 - Il ricevente risponde con un ACK anche se non può memorizzare pacchetti
 - Ad un certo punto il buffer inizierà a svuotarsi e il mittente verrà avvisato ricevendo un valore non nullo di `rcvWindow`

24

Evento (lato ricevente)	Azione del ricevente TCP
Arrivo ordinato di segmento con n. di seq. atteso. Tutti i dati con n. di seq. atteso sono già stati riscontrati	ACK ritardato. Attende fino a 500 ms l'arrivo ordinato di un altro segmento. Se in questo intervallo non arriva il successivo segmento, invia un ACK
Arrivo ordinato di segmento con n. di seq. atteso. Un altro segmento ordinato è in attesa di trasmissione dell'ACK	Immediatamente invia un singolo ACK cumulativo, riscontrando entrambi i segmenti ordinati
Arrivo non ordinato di segmento con n. di seq. superiore a quello atteso. Viene rilevato un buco	Immediatamente invia un ACK duplicato , indicando come n. di seq. il prossimo byte atteso (che corrisponde all'estremità inferiore del buco)
Arrivo di un segmento che colma parzialmente o completamente il buco nei dati ricevuti	Immediatamente invia un ACK, ammesso che il segmento cominci all'estremità inferiore del buco

- In alcuni casi il periodo di timeout può rilevarsi molto lungo, causando una lunga attesa prima di rispedire il segmento perso
- D'altra parte la perdita di segmenti può essere rilevata prima dello scadere del timeout, mediante gli **ACK duplicati** inviati dal ricevente quando riceve un segmento non in ordine
- Poiché il mittente invia spesso molti segmenti, se un segmento viene smarrito, riceverà più ACK duplicati che hanno lo stesso numero di riscontro
- Se il mittente riceve **3 ACK per lo stesso segmento** (stesso n. di riscontro), suppone che il segmento successivo al segmento riscontrato è andato perduto
 - Il mittente TCP effettua una ritrasmissione rapida: rispedisce il segmento mancante prima che scada il timer

```

event: ACK ricevuto con campo ACK pari a y
if (y > SendBase) {
    SendBase = y
    if (esistono attualmente seg. non ancora riscontrati)
        riavvia il timer
}
else { /* un ACK duplicato per un segmento
        già riscontrato */
    incrementa il n. di ACK duplicati ricevuti per y
    if (n. di ACK duplicati per y == 3) {
        rispedisci il segmento con numero di sequenza y
    }
}
    
```

Ritrasmissione rapida (prima dello scadere del timeout)

Three way handshake:

Step 1: l'host client invia un segmento TCP (SYN) di controllo al server

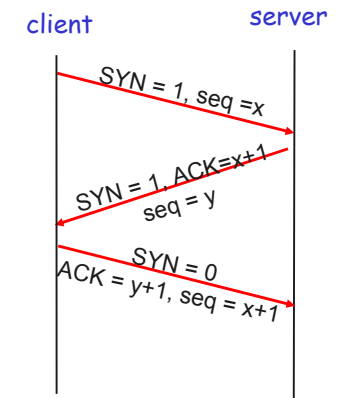
- fissa il numero di sequenza iniziale, lato client (scelto a caso)

Step 2: l'host server riceve il seg. SYN, replica con un segmento di controllo SYN/ACK

- riscontra il segmento SYN
- alloca buffer e variabili
- sceglie il numero di sequenza iniziale, lato server (scelto a caso)

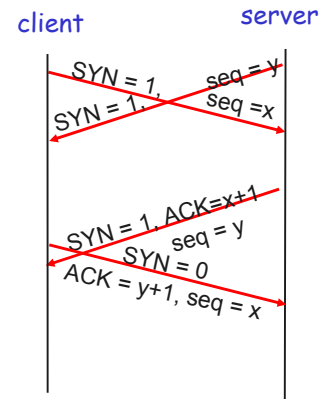
Step 3: l'host client riceve il SYN/ACK, replica con un segmento di ACK che può contenere dati

- alloca buffer e variabili



Gestione della connessione: apertura (2)

- Apertura simultanea da parte dei due host
 - Il risultato sarà un'unica connessione stabilita, identificata da (x,y)



29

Gestione della connessione: chiusura (1)

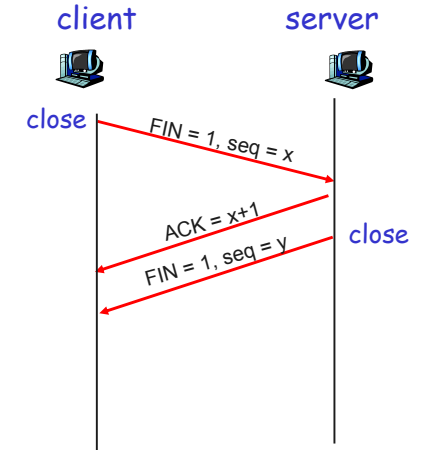
Chiusura di una connessione

Una connessione è trattata come una coppia di connessioni unidirezionali separate, per evitare perdita di dati. Può essere richiesta da entrambi gli host.

Ad es. il client chiude una socket

Step 1: l'host client invia un segmento TCP (FIN) al server

Step 2: l'host server riceve il segmento FIN, risponde con un ACK. Il server chiude la connessione ed invia un segmento FIN



30

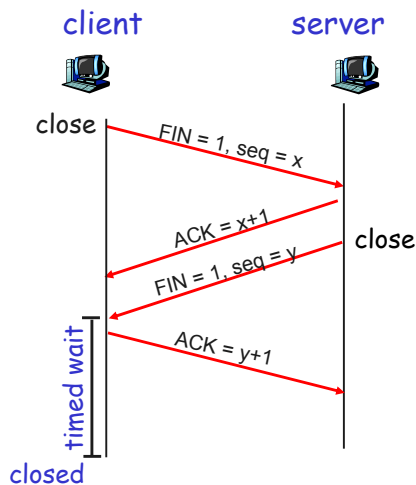
Gestione della connessione: chiusura (2)

Step 3: l'host client riceve il seg. FIN e risponde con un ACK

- Entra nello stato "timed wait"
- Gestire la ritrasmissione dell'ultimo ACK nel caso in cui sia andato perso

Step 4: l'host server, riceve un ACK. Chiude la connessione

- A questo punto tutte le risorse degli host sono state deallocate
- Il server potrebbe utilizzare un unico segmento per inviare ACK e FIN
- Può succedere che il client e il server richiedano di chiudere la connessione contemporaneamente



31

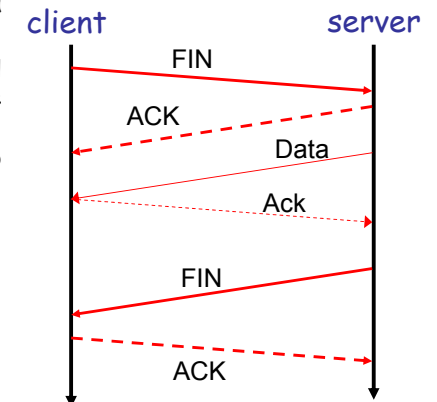
Chiusura parziale della connessione

- Una connessione può essere chiusa solo parzialmente

- La connessione è chiusa dal client al server, ma il server potrebbe continuare a inviare dati al client
- Il client non può spedire ma può continuare a ricevere

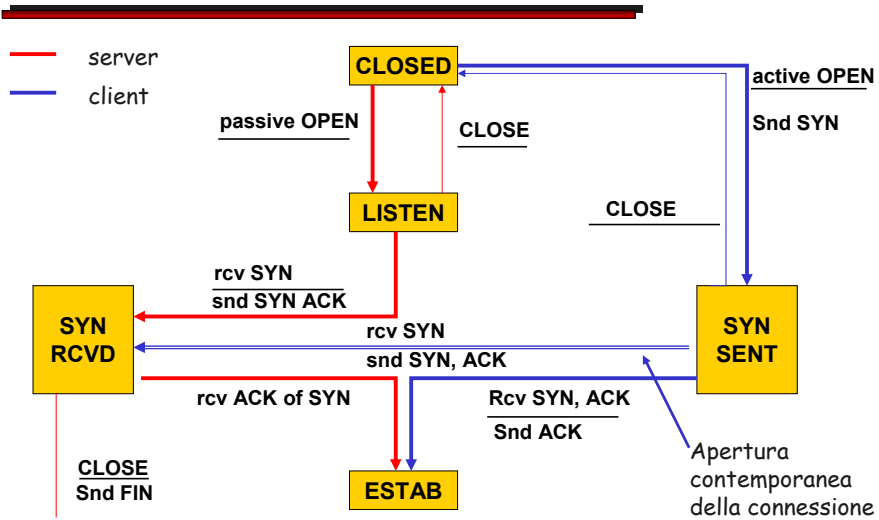
- In Java:

- `Socket.shutdownOutput();`

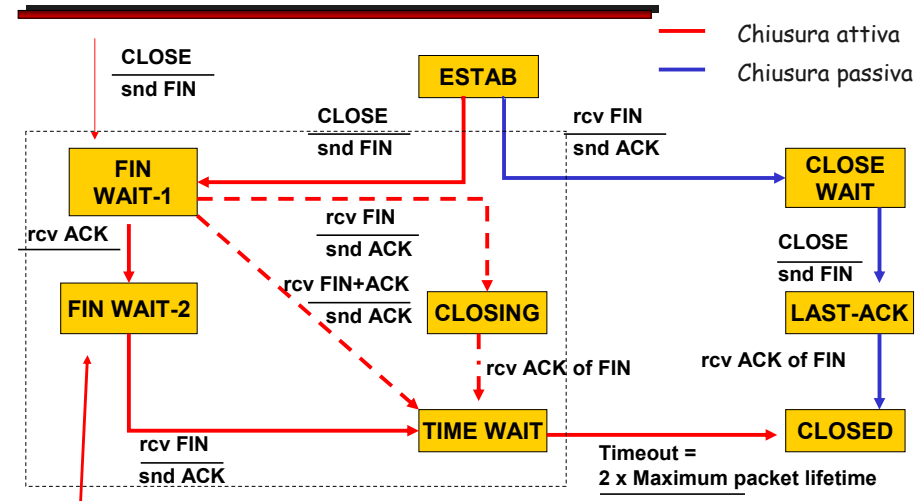


32

Attivazione della connessione



Chiusura della connessione



La connessione è chiusa da un lato