



Università
degli Studi
della Campania
Luigi Vanvitelli

Reti di Calcolatori e Cybersecurity

TCP – Transmission Control Protocol

Ing. Vincenzo Abate

TCP

opposto di UDP

- TCP è un protocollo di trasporto connection-oriented che consente la trasmissione bidirezionale affidabile di un flusso di byte tra due endpoint
- TCP garantisce assenza di perdite e rispetto nella sequenza dei dati (*No UDP!*)
- Definito inizialmente in RFC 793 del 1981
- Implementa meccanismi di controllo di flusso e controllo di congestione (*No UDP!*)
- Affidabilità realizzata mediante controllo di sequenza e meccanismi di ritrasmissione *Approccio al problema, VoIP fa solo un checksum*
- Dati sono mantenuti in appositi buffer ad entrambe le estremità dei flussi dei dati
- Le applicazioni interagiscono con TCP tramite API che si avvalgono di punti di accesso al servizio detti **socket**

TCP

UDP è gestito, può essere entrambi.
Di default non lo è, ma lo si può usare
come tale

- TCP offre un servizio **full duplex** ↗ Sia entrata che uscita contemporaneamente
- Le connessioni TCP sono punto-punto (NO MULTICAST!) ↗ No solo 1 a molti
- La ‘connessione’ TCP **NON** è un circuito end-to-end (TDM o FDM) come in una rete a commutazione di circuito, poiché lo stato della connessione risiede completamente negli endpoint. * Solo entità che comunicano
- Infatti TCP è in esecuzione solo negli end-point e non negli elementi intermedi (router, switch..) e quindi questi ultimi non salvano lo stato della sessione TCP.
- I router sono completamente ignari delle sessioni TCP... vedono datagrammi ^{IP} **NON CONNESSIONI**

↑ livello di astraz. non conosciuto

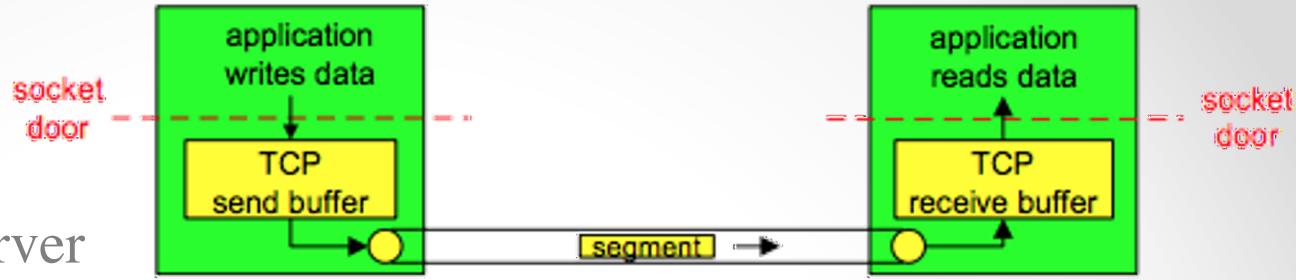
* UDP non orientato alla connessione, non ha filtri problemi

Gli intermedi si fermano al 3° livello.

Step intermedi non sanno nulla.

TCP

Proc. Client → Proc. Server



3-way handshake *3 pacchetti, prima 2 senza payload, resto può avere info su connettore*

Il primo manda un flusso di dati attraverso la socket e arrivano a TCP

TCP dirige i dati verso il buffer di invio (riservato durante il 3-way hand.)

Interessante: [RFC793] non specifica quando TCP debba inviare i dati nel buffer ma afferma che dovrebbe spedire «quando è più conveniente» *Si demanda a implementazione*

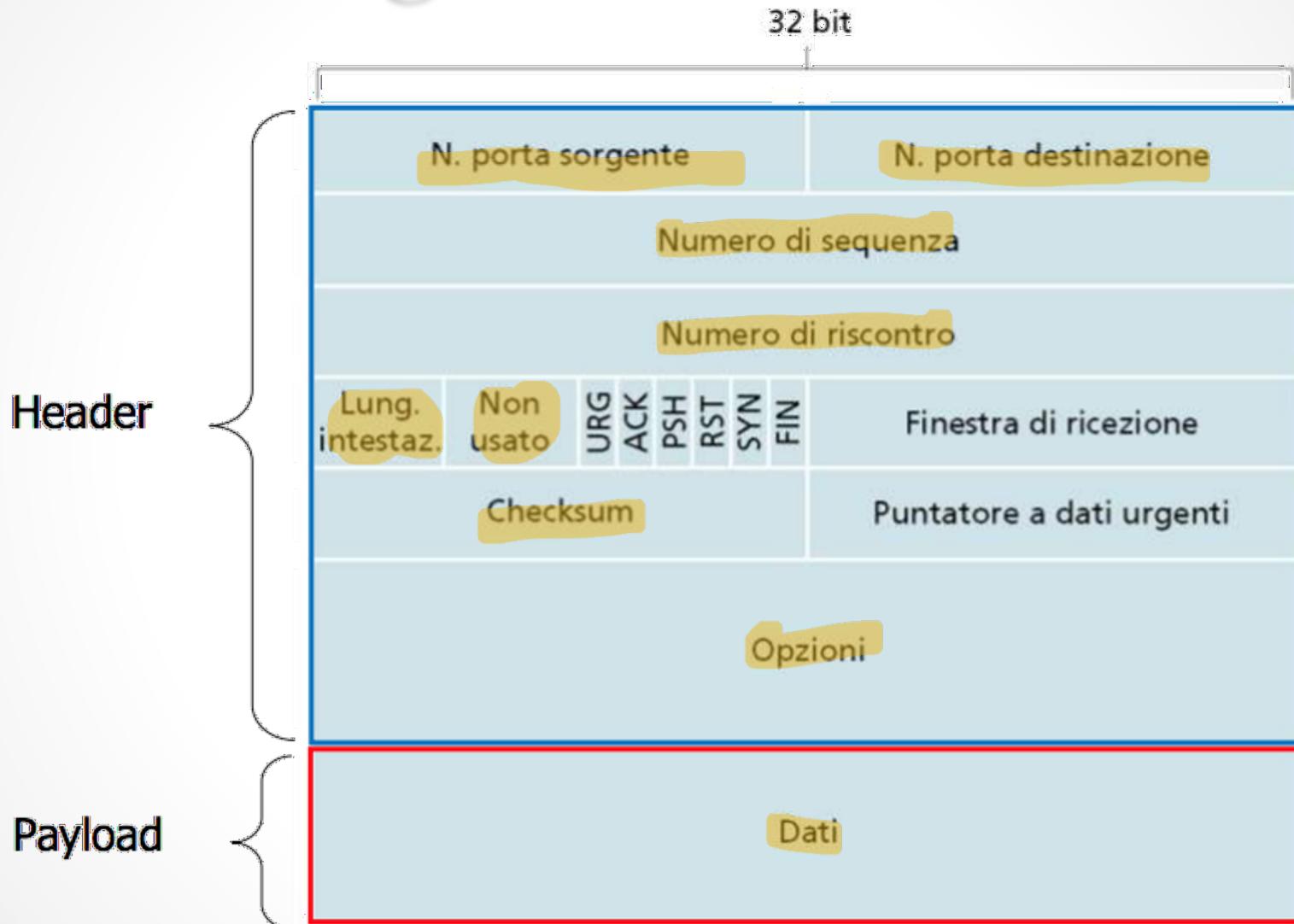
La massima quantità di dati prelevabili e posizionabili in un segmento viene limitata da MSS (Maximum segment size) *Paravello se l'altro durante collegamento (payload) ci mette Header*

TCP accoppia ogni blocco di dati del client a una intestazione TCP, formando i segmenti TCP che vengono passati al livello rete sottostante, dove saranno incapsulati in datagrammi IP.

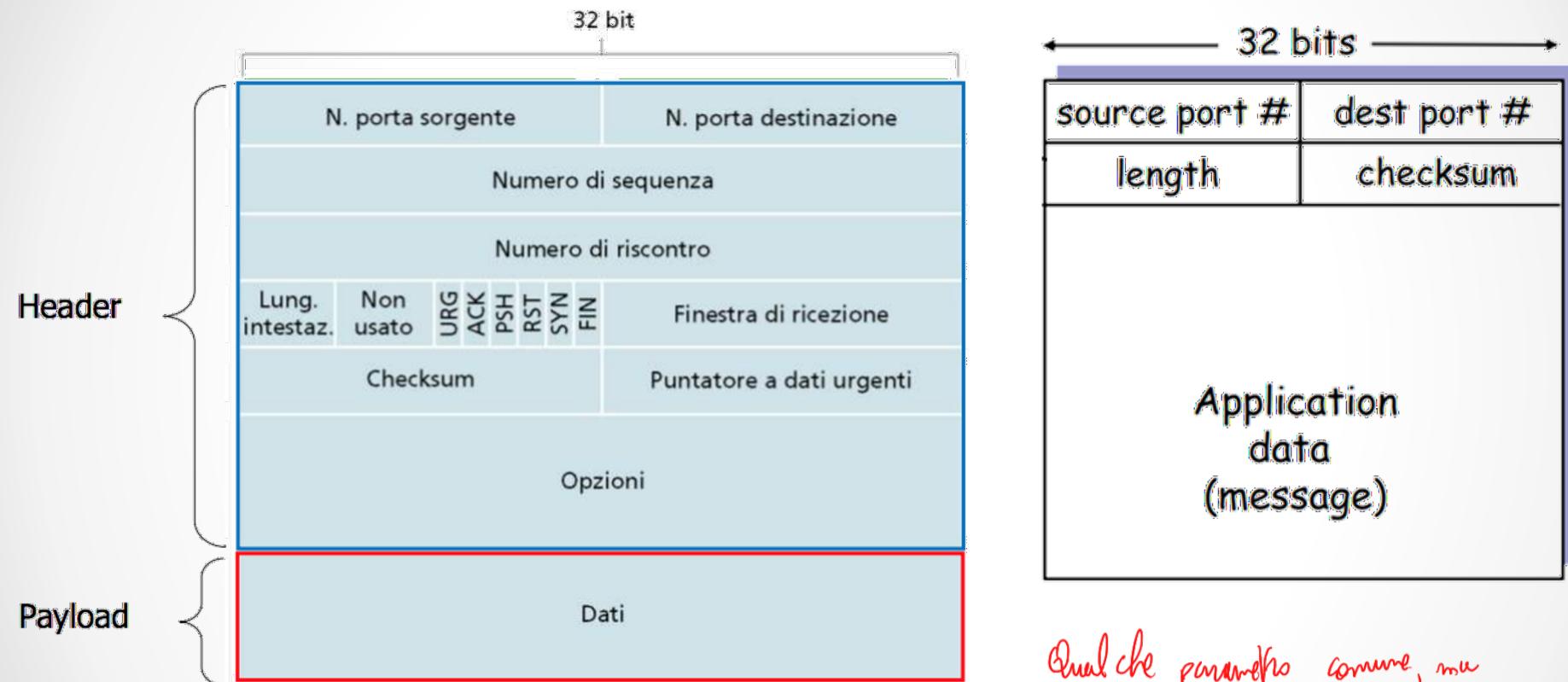
In TCP riceve un segmento i dati del segmento vengono memorizzati nel buffer di ricezione della connessione TCP. L'applicazione legge il flusso dati da questo buffer.

Ogni lato della connessione ha un suo buffer di invio e ricezione

Segmento TCP



Segmento TCP vs UDP



Qualche parametro comune, ma
TCP molto più complesso di UDP

Header TCP

- Numeri di porta sorgente e destinazione
 - Contengono i numeri di porta di protocollo TCP che identificano gli end-point della connessione (mux/demux)
- Lunghezza header HLEN
 - 4 bit, contiene un numero intero che indica la lunghezza dell'intestazione TCP del datagramma in parole da 32 bit
 - Questa informazione è necessaria perché il campo opzioni è di lunghezza variabile
 - HLEN assume valori compresi tra 5 (minimo) e 15 (massimo)
 - HLEN = 5 -> lunghezza header TCP: 20 byte -> no opzioni
 - HLEN max è 15 -> lunghezza massima header TCP: 60 byte

Dove sapere dove stanno inserendo i dati.

Header TCP

Flag

Per identificare il tipo di informazione contenuta nel segmento vengono impiegati i 6 bit di codice:

- ACK: Il campo riscontro è valido
- RST: Effettua il reset della connessione
- SYN: Sincronizza i numeri di sequenza
- FIN: Il trasmettitore ha raggiunto la fine del suo stream di byte
- PSH: Questo segmento richiede una “spinta” (a destinazione)
- URG: Il campo puntatore urgente è valido

↑ allora dati urgenti

Falta la RST miss.
↑
Deve essere mandato immediatamente
non devo aspettare riempimento del
buffer.

Header TCP

Finestra di ricezione

Numero intero senza segno di 16 bit che specifica la dimensione del buffer che il TCP ha a disposizione per immagazzinare dati in arrivo

Puntatore ai dati urgenti

- Il TCP permette la trasmissione di dati informativi ad alta priorità che devono essere trasmessi il prima possibile
- Questo campo, se valido (flag URG ad 1), conterrà un puntatore alla posizione nello stream dei dati NON urgenti (ultimo byte dei dati urgenti)

Checksum

- Campo di 16 bit contenente un valore intero utilizzato dal ricevitore per verificare l'integrità del segmento ricevuto
 - IP non prevede nessun controllo di errore sulla parte dati del frame
 - In caso di rilevazione di errore il segmento viene scartato

Segmenti riscontro

L'informazione di riscontro viaggia in normali segmenti TCP identificati dal valore 1 del flag ACK *è un segmento di riscontro se ACK=1*

Per ogni connessione TCP tra due end-point A e B esistono due flussi dati distinti:

- quello da A a B
- quello inverso da B ad A

Un segmento inviato dall'host B all'host A:

- può contenere o meno dati relativi al flusso da B ad A
- può contenere o meno un informazione di riscontro relativa al flusso dati da A a B

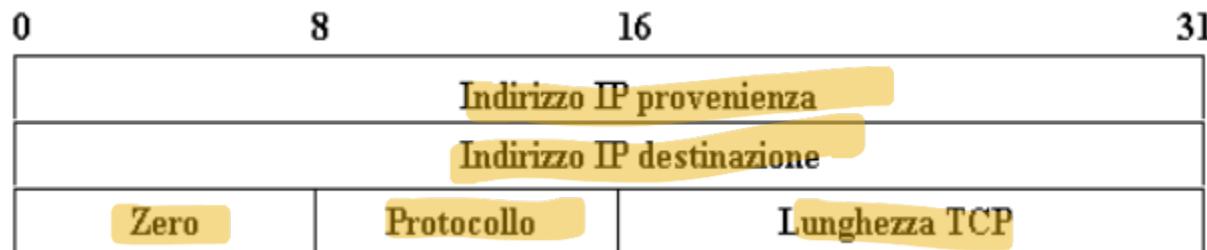
Se un segmento contiene sia dati che riscontro, si dice che il riscontro viaggia "a cavalluccio" dei dati (**piggy-backing**)

Mando dati e riscontro insieme

Pseudo Header

Ai soli fini del calcolo della checksum, TCP aggiunge fintiziamente al segmento effettivo uno pseudo-header costituito come in figura

In ricezione, TCP ricrea la pseudo-header interagendo con lo strato IP sottostante, calcola la checksum e verifica la correttezza del messaggio ricevuto



Checksum TCP

Mittente:

Tratta il contenuto del segmento (e dello pseudo-header) come una sequenza di interi espressi su 16 bit

Calcola la checksum come complemento ad 1 della somma in complemento ad 1 dei «pezzi» da 16 bit che costituiscono il segmento TCP e lo pseudo-header

Pone il valore di checksum nell'apposito campo del segmento trasmesso

Ricevente:

Calcola la somma in complemento ad 1 dei campi del segmento ricevuto compresa la checksum

Se il risultato NON è composto da tutti 1, questo è indicativo di un errore

Se il risultato è composto da tutti 1, il controllo di checksum è superato

Opzioni header TCP

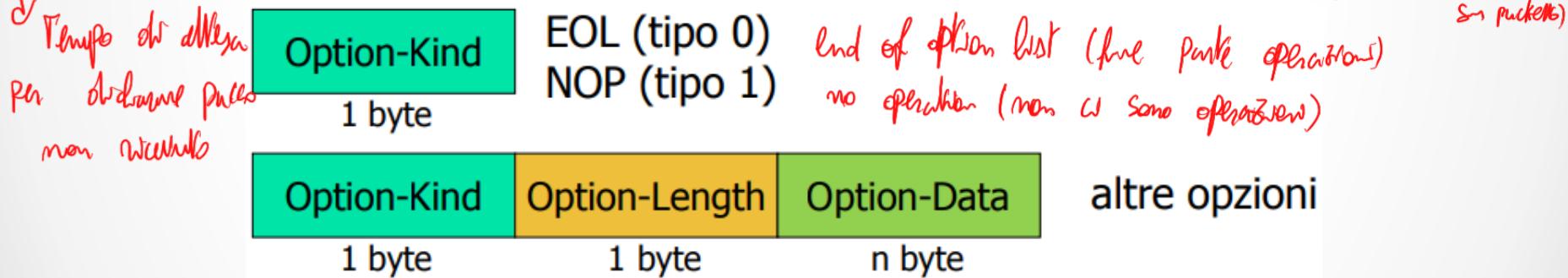
Le opzioni sono di lunghezza variabile

La lunghezza totale dell'header TCP deve essere multipla di 4 byte

Se necessario si aggiungono zeri di padding dopo l'ultima opzione

Sono previsti 7 diversi tipi di opzioni: MSS, Window scale, Ttimestamp, Selective ACK permitted, Selective ACK, NOP, EOL

Il formato delle opzioni è di due tipi:



La maggior parte delle opzioni è consentita solo nella fase di instaurazione della connessione (segmenti con flag SYN=1)

<https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml>

Opzioni header TCP

Selective ACK

Sack-Permitted Option

This two-byte option may be sent in a SYN by a TCP that has been extended to receive (and presumably process) the SACK option once the connection has opened. It MUST NOT be sent on non-SYN segments.

TCP Sack-Permitted Option:

Kind: 4

| + | - | - | + |
|---|--------|---|----------|
| | Kind=4 | | Length=2 |
| + | - | - | + |

Sack Option Format

The SACK option is to be used to convey extended acknowledgment information from the receiver to the sender over an established TCP connection.

TCP SACK Option:

Kind: 5

Length: Variable

| + | - | - | + |
|---|-------------------------|---|--------|
| | Kind=5 | | Length |
| + | - | - | + |
| | Left Edge of 1st Block | | |
| + | - | - | + |
| | Right Edge of 1st Block | | |
| + | - | - | + |
| / | ... | / | |
| / | | / | |
| + | - | - | + |
| | Left Edge of nth Block | | |
| + | - | - | + |
| | Right Edge of nth Block | | |
| + | - | - | + |

Opzioni header TCP

MSS: durante la fase di connessione, ciascun end-point annuncia la massima dimensione di payload (MSS – Maximum Segment Size) che desidera accettare (**MSS announcement**) *→ definire buffer, finestra da usc.*

Window Scale: per negoziare un fattore di scala per la finestra; utile per connessioni a larga banda ed elevato ritardo di trasmissione (long-fat pipes)

Selective Repeat: nel caso in cui un segmento corrotto sia stato seguito da segmenti corretti, introduce i NAK (Not AcKnowledge), per permettere al receiver di richiedere la ritrasmissione di quello specifico segmento *Ricevo solo quelli che mi servono*

Timestamp: utilizzata per aiutare i due endpoint a determinare il RTT (tempo intercorso dalla trasmissione di un segmento alla ricezione del relativo ACK)

NOP: No-OPerations, separa opzioni diverse quando non allineate su multipli di lunghezza di quattro byte

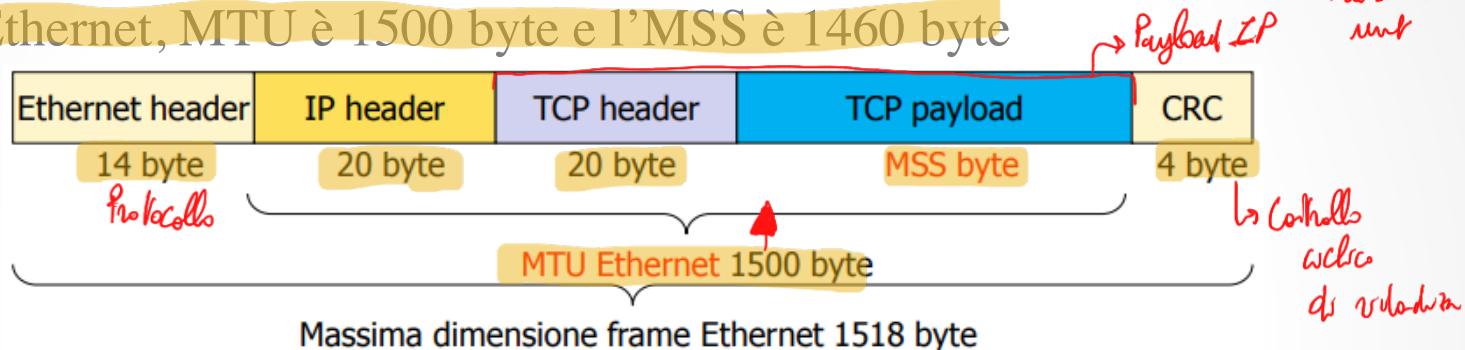
EOL: End of Options List, indica la fine delle opzioni, necessario se la fine delle opzioni non coincide con la fine dell'header TCP

Opzioni header TCP

Per **MSS** in TCP si intende la massima dimensione di payload consentita

Questo valore è determinato dall'MTU del link sul quale l'host trasmette

Per un link Ethernet, MTU è 1500 byte e l'MSS è 1460 byte



L'opzione **MSS** di TCP consente a ciascuno dei due endpoint di notificare all'altro la massima dimensione di segmento da utilizzare



Se l'altro endpoint non ha trasmesso l'opzione MSS nel segmento SYN, RFC1122 (Requirements for Internet Hosts) prescrive che MSS sia posto a 536 byte

Affidabilità TCP

Ottenuta tramite:

Riscontro e ritrasmissione:

Consiste nella ritrasmissione di un segmento se non è giunta conferma entro un tempo massimo (time-out)

Time-Out:

Al momento della trasmissione di un segmento, il TCP attiva un timer

Numeri di sequenza e riscontro

TCP vede i dati come un flusso di byte non strutturati ma ordinati

Il campo «numero di sequenza» di un segmento TCP indica il numero di sequenza progressivo del primo byte nel segmento

Flusso lungo 500.000 byte, MSS uguale a 1000 byte, primo byte numerato con 0

TCP costruisce 500 segmenti, con numeri di sequenza 0, 1000, 2000 ...

Per i numeri di riscontro, vista la natura full-duplex della connessione TCP, si ha che ad es.:

A invia e contemporaneamente riceve da B

I segmenti B -> A, contengono un numero di sequenza relativo ai dati B -> A

Il numero di riscontro che A scrive nei propri segmenti è il numero di sequenza del byte successivo che A attende da B (e lo può mandare anche inviando dati a B)

TCP invia riscontri cumulativi = Se mi arriva riscontro con numero più alto

di quel che mi aspetto, do per scattato che quelli prima
sono arrivati

Numeri di sequenza e riscontro

Numeri di sequenza:

“numero” del primo byte del segmento nel flusso di byte

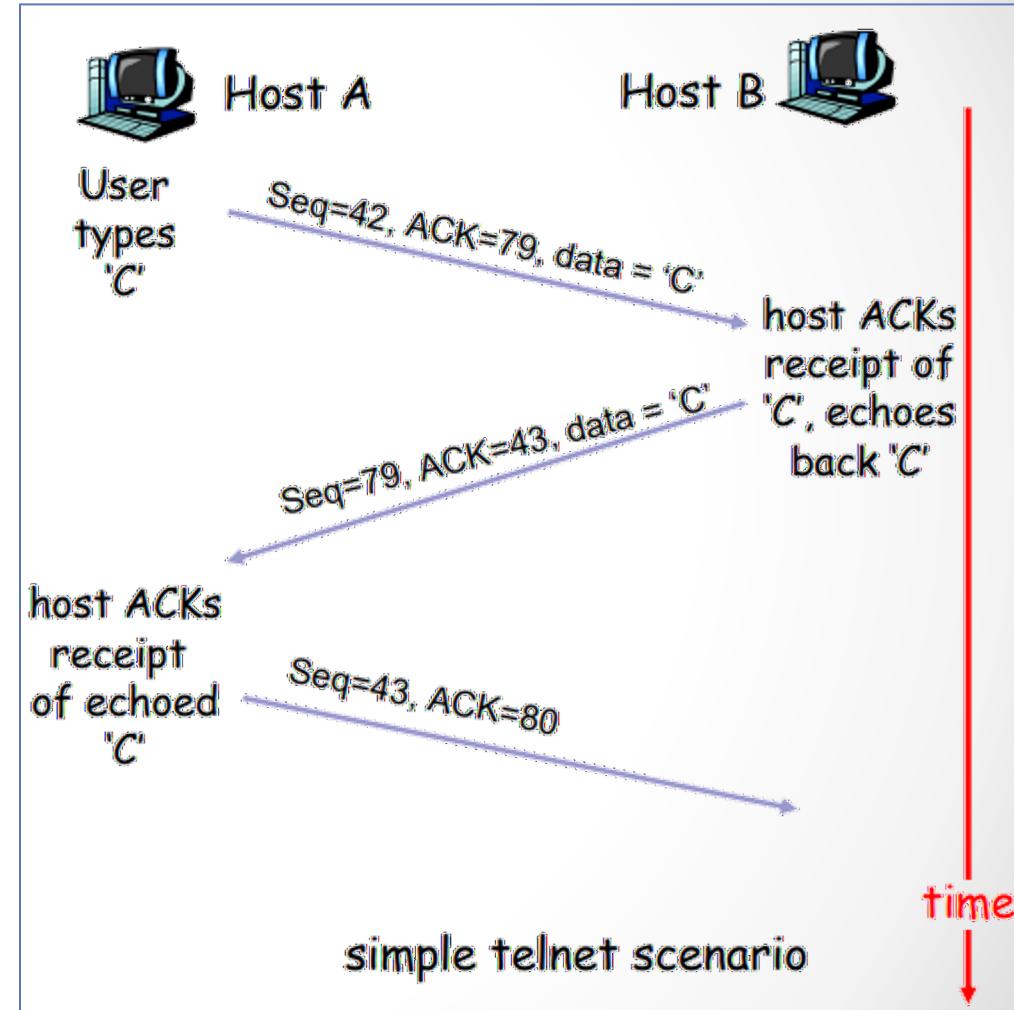
ACK:

numero di sequenza del prossimo byte atteso dall’altro lato

ACK cumulativo

D: come gestisce il destinatario i segmenti fuori sequenza? (Segnali che non arrivano da ordine)

R: la specifica TCP non lo dice – dipende dall’implementatore



Una semplice applicazione: Telnet Il server invia l’eco dei caratteri digitati dal client.

Riscontro cumulativo

Nel datagramma di riscontro la destinazione comunica quale byte dello stream si aspetta di ricevere successivamente

- I riscontri specificano il numero sequenziale del primo byte non ancora ricevuto *Receiver manda riscontro sul prossimo byte che mi aspetta*
- Esempio: in uno stream di 1000 byte segmentato in blocchi di 100 byte, partendo da 0, il primo riscontro conterrà il numero sequenziale 100
- Con questo metodo di riscontro cumulativo si ha il vantaggio che la perdita di un riscontro non blocca la trasmissione se confermato dal riscontro successivo

Se perdo 42, arriva 43 dipende da implementazione cosa fare.

Mando riscontro con numero 92

92 →
X ← OK 93

93 → OK 94 → è arrivato nexting, 93 OK

RTT e timeout

Domanda: A quale valore impostare il timeout?

Di sicuro maggiore di RTT (Round Trip Time) (ma RTT varia nel tempo)

Se timeout è scelto troppo breve:

timeout prematuro – ritrasmissioni ridondanti – scarsa efficienza

Se timeout è scelto troppo lungo:

scarsa efficienza nella gestione delle ritrasmissioni

Problema: Come stimare RTT corrente?

A causa della variabilità, occorre considerare anche la varianza

RTT e timeout

Stima di RTT: media esponenziale pesata dei campioni (EWMA: Exponential Weighted Moving Average)

L'influenza di un singolo campione sul valore della stima decresce in maniera esponenziale, e si dà più importanza a campioni recenti.

Valore tipico per α : 0.125

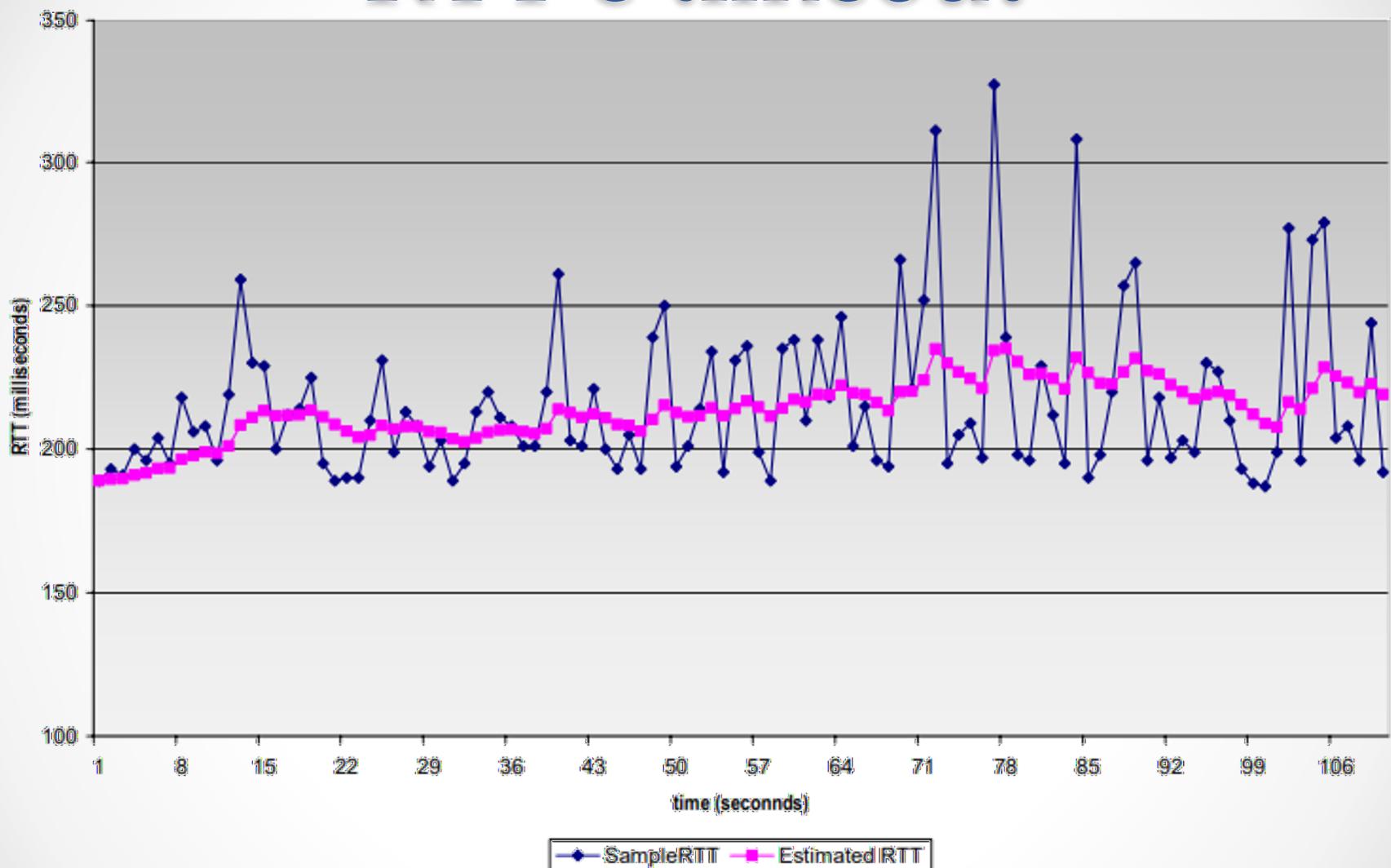
$$\text{EstimatedRTT} = (1-\alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

↑
Fino a quel
momento

RTT da quel momento

↑ Se ne lo stato attuale

RTT e timeout



SampleRTT: tempo misurato dalla trasmissione del segmento fino alla ricezione di ACK, ignorando le ritrasmissioni (se ne sceglie uno ad ogni istante di tempo).

RTT e timeout

Stima della deviazione di RTT (quanto SampleRTT devia da Estimated RTT):

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

Valore raccomandato per β : 0,25

↑
Fino a quel
momento

Settiamo il timeout

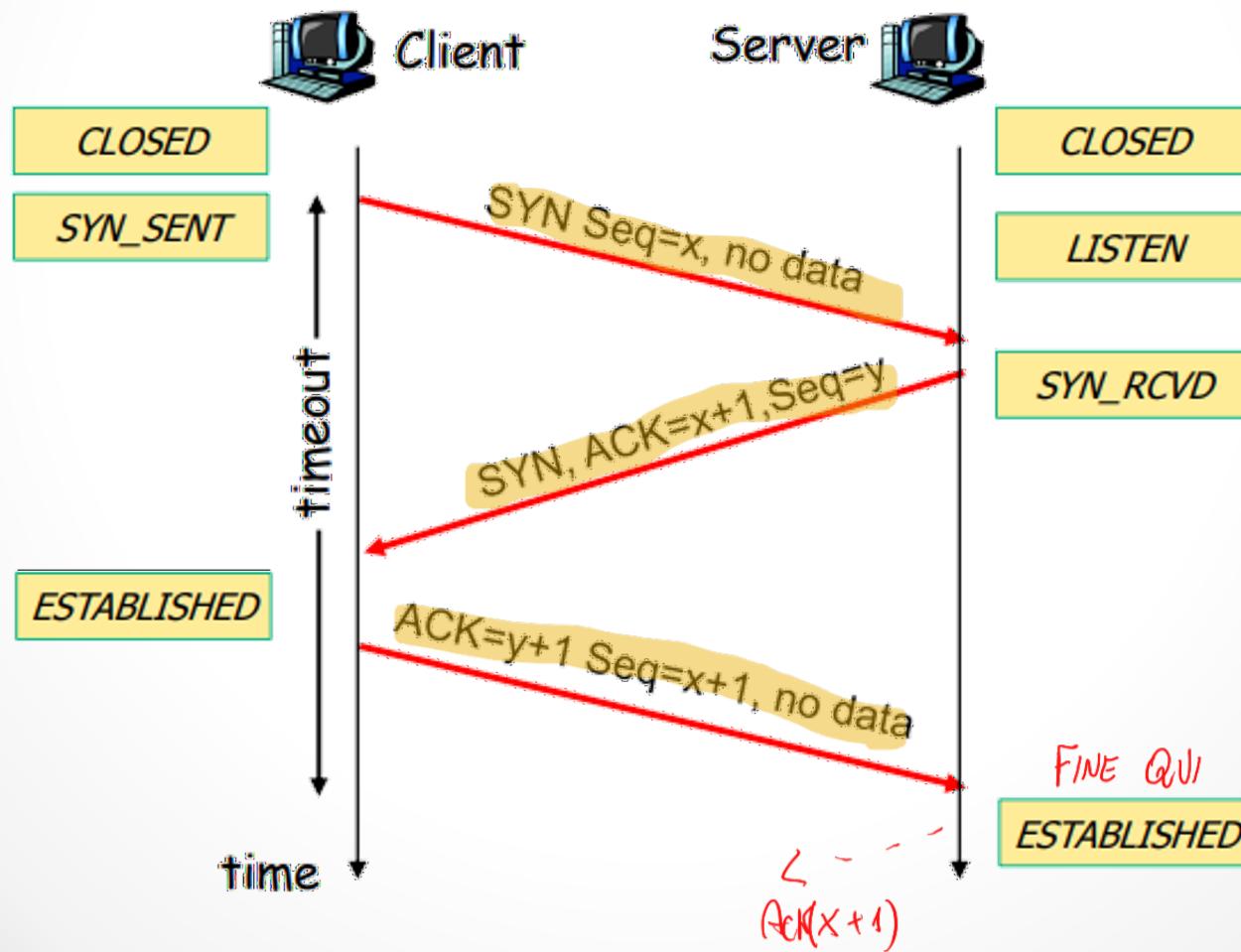
Valore del timeout: EstimatedRTT più un “margine di sicurezza” proporzionale alla variabilità della stima effettuata

variazione significativa di EstimatedRTT → margine più ampio:

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

TCP avvio connessione

- 3 way-handshake
- Il client prende l'iniziativa inviando il primo segmento



TCP avvio connessione

- Nella fase three-way-handshake i due endpoint allocano i buffer per i dati e concordano sui valori iniziali dei numeri di sequenza da utilizzare per gli stream dati in entrambi i versi (Recv Window)
- Passo 1: client invia segmento di controllo TCP SYN al server Specifica il 1° seq #
- Passo 2: server riceve SYN, risponde con segmento di controllo SYN/ACK
 - ACK del SYN ricevuto
 - Alloca buffer
 - Specifica il 1°seq. # per la connessione server → client
- Passo 3: client riceve SYN/ACK, invia ACK al server Connessione instaurata
- Nella fase di three-way-handshake inoltre:
 - le due parti apprendono il valore iniziale di Recv Window
 - utilizzano le opzioni TCP per determinare il valore di MSS, RTT, ecc.

TCP chiatura connessione

Procedura detta four-way-handshake

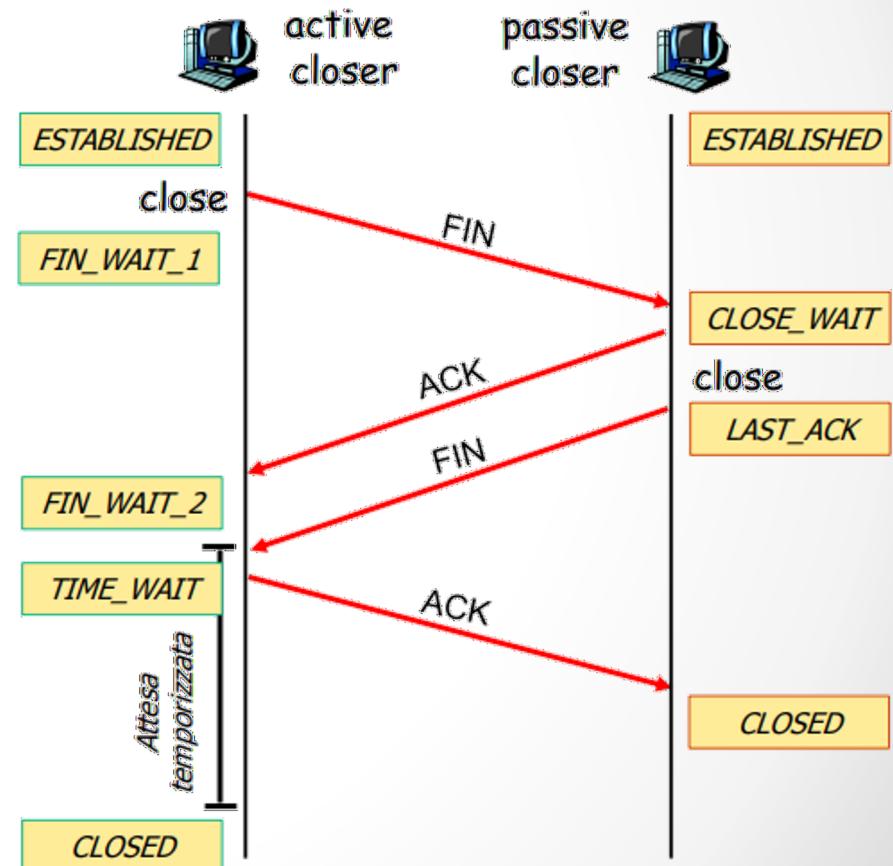
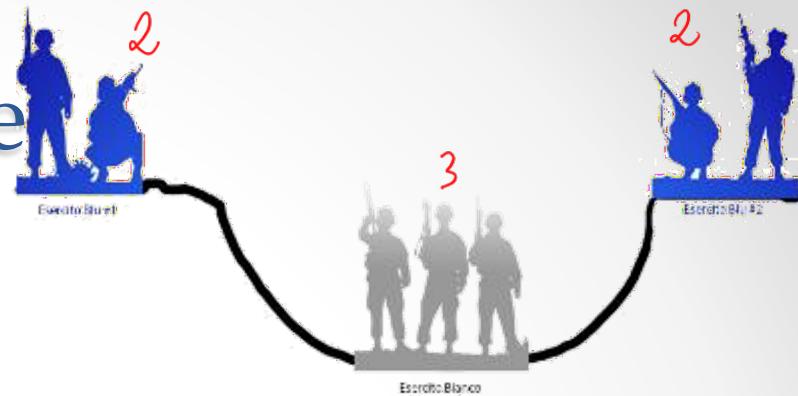
La parte che manifesta per prima la volontà di chiudere la connessione è detta **active closer**, l'altra è il **passive closer**

- L'active closer può essere il client oppure il server a seconda dei protocolli applicativi

Se non ha altri dati da inviare, il passive closer può inviare ACK e FIN nello stesso segmento

L'active closer attende in uno stato TIME_WAIT (nel caso in cui l'ultimo ACK vada perso, e riceva un ulteriore FIN dal passive closer)

La durata della permanenza nello stato TIME_WAIT dipende dal sistema operativo (tipicamente 120s)



Stessa situazione per chiusura connessione. Se 1 o 2 ha fatto di finire e vuole chiudere, se A manda mess. di chiusura e chiude, rischio che si perde messaggio B rimane aperto.

Allora A manda close, B risponde OK, lo chiude. Ma se mess. di B non arriva? Soluzione:
4 way HS: active closer manda FIN (vuole chiudere). Altro endpoint manda ACK + messaggio di FIN. Active closer manda ACK e si mette in attesa; Se B non riceve riscontro ritorna FIN. Allo scadere del timeout si chiude.

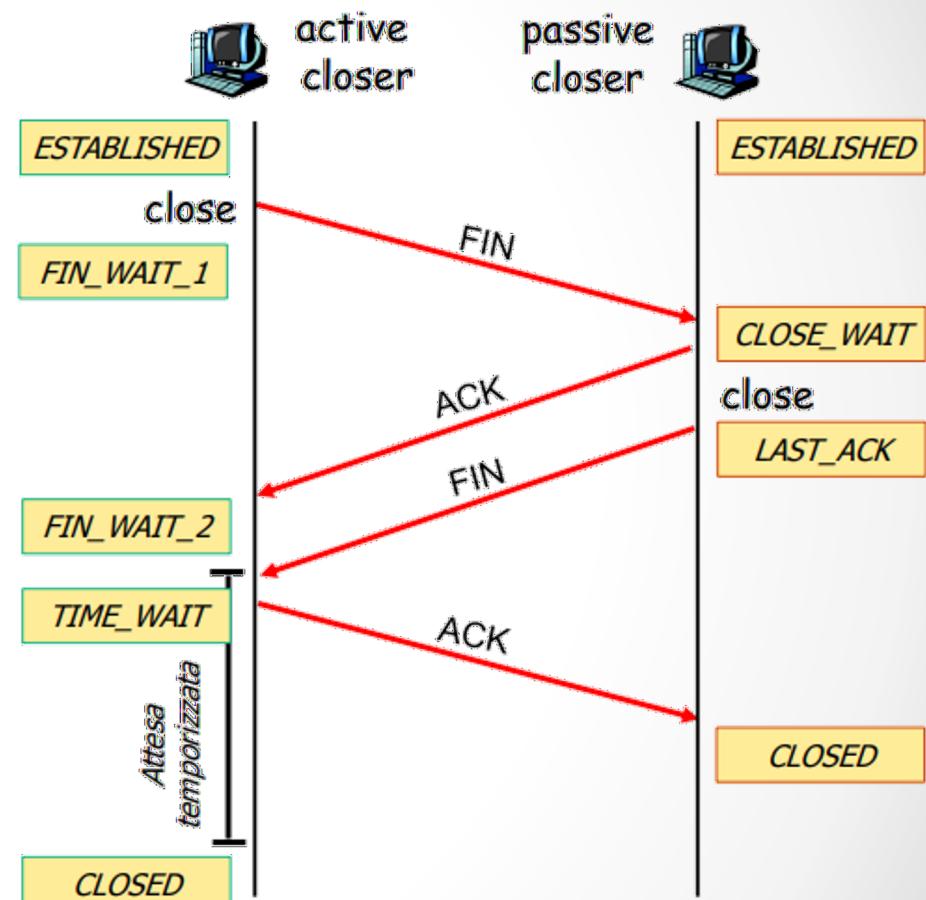
TCP chiusura connessione

Lo stato TIME_WAIT è raggiunto dall'endpoint che ha preso l'iniziativa di chiudere la connessione TCP (active closer)

Finché l'endpoint (A,x) rimane in TIME_WAIT non è possibile la creazione di una nuova connessione tra gli stessi endpoint (A,x) e (B,y)

Ciò serve ad evitare che un segmento dati duplicato prodotto da una precedente connessione tra (A,x) e (B,y) il cui numero di sequenza iniziale rientri nella finestra di ricezione corrente sarebbe erroneamente ritenuto un segmento «valido» per la nuova connessione

Se avessi parametri segnali



TCP trasferimento affidabile

TCP crea un servizio di trasferimento dati affidabile sul servizio inaffidabile di IP

Segmenti pipelined

ACK cumulativi

TCP usa un solo timer di ritrasmissione

Le ritrasmissioni sono avviate da:

- eventi di timeout
- ACK duplicati

Inizialmente consideriamo un mittente TCP semplificato:

- ignoriamo gli ACK duplicati
- ignoriamo il controllo di flusso e il controllo di congestione

TCP eventi del mittente

Dati ricevuti dall'applicazione:

- Crea un segmento con il numero di sequenza
- Il numero di sequenza è il numero del primo byte del segmento nel flusso di byte
→ go-back-N
- Avvia il timer, se non è già in funzione (pensate al timer come se fosse associato al più vecchio segmento non riscontrato)
- Intervallo di scadenza: TimeOutInterval

Timeout:

- Ritrasmette il segmento che ha causato il timeout
- Riavvia il timer *sempre per quello*

ACK ricevuti:

- Se riscontra segmenti precedentemente non riscontrati aggiorna ciò che è stato completamente riscontrato *aggiorno nulls quelli presenti*
- avvia il timer se ci sono altri segmenti da completare

TCP sender semplificato

Supponiamo:

- Sender non limitato dal controllo di flusso o di congestione del TCP
- Dati da sopra di dimensioni inferiori a MSS
- Trasferimento dati in una sola direzione

```
NextSeqNum = InitialSeqNum    Prossimo numero di sequenza
SendBase = InitialSeqNum

loop (forever) {
    switch(event)

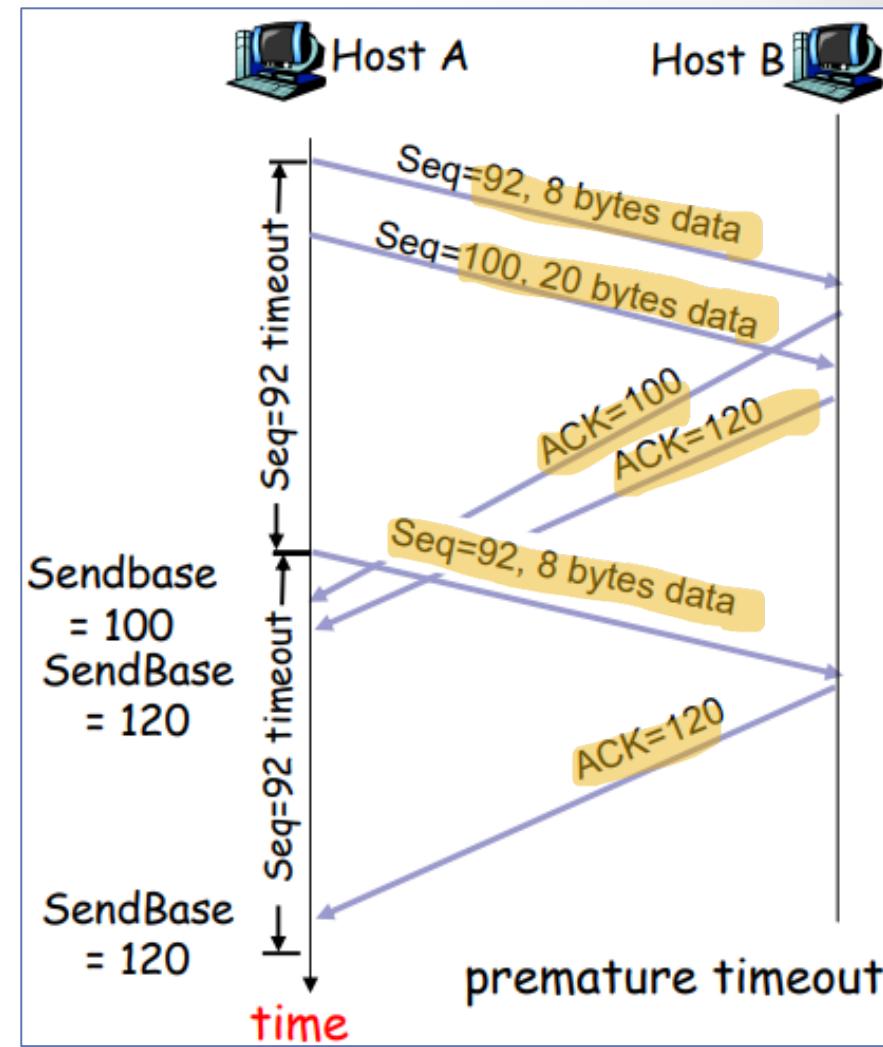
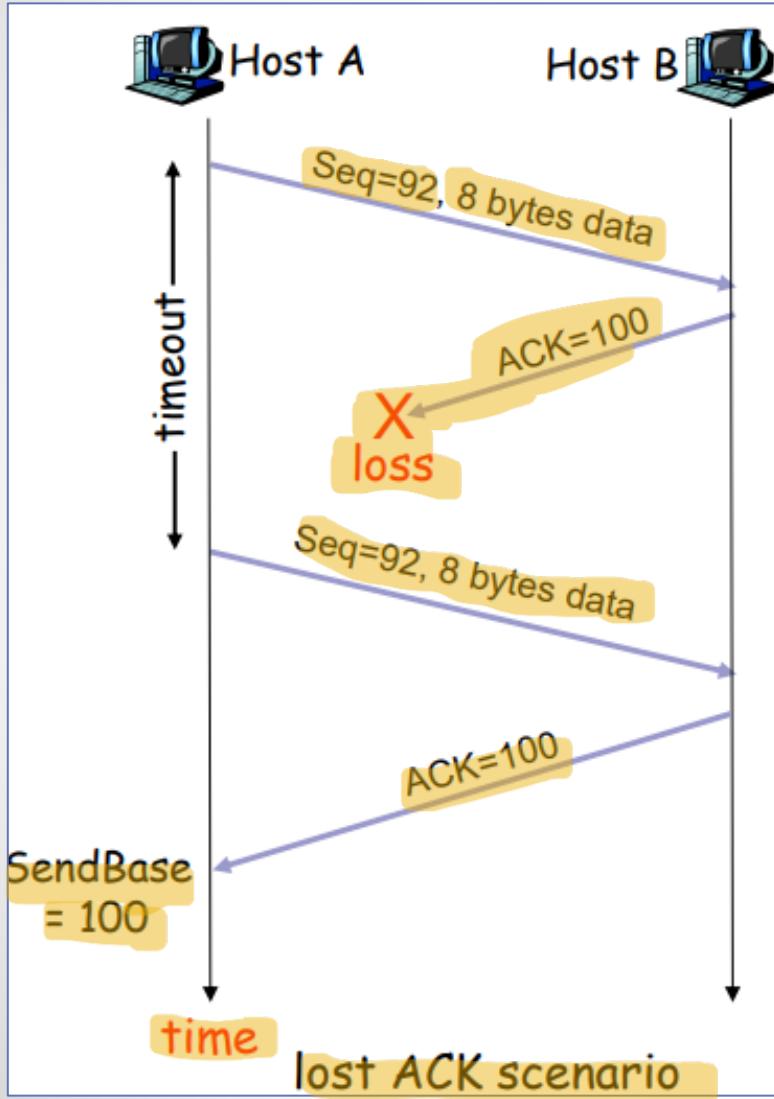
    event: data received from application above
        create TCP segment with sequence number NextSeqNum
        if (timer currently not running)
            start timer
        pass segment to IP
        NextSeqNum = NextSeqNum + length(data)

    event: timer timeout
        retransmit not-yet-acknowledged segment with
            smallest sequence number
        start timer

    event: ACK received, with ACK field value of y
        if (y > SendBase) {
            SendBase = y    → numero del primo byte che sto mandando
            if (there are currently not-yet-acknowledged segments)
                start timer
        }

    } /* end of loop forever */
```

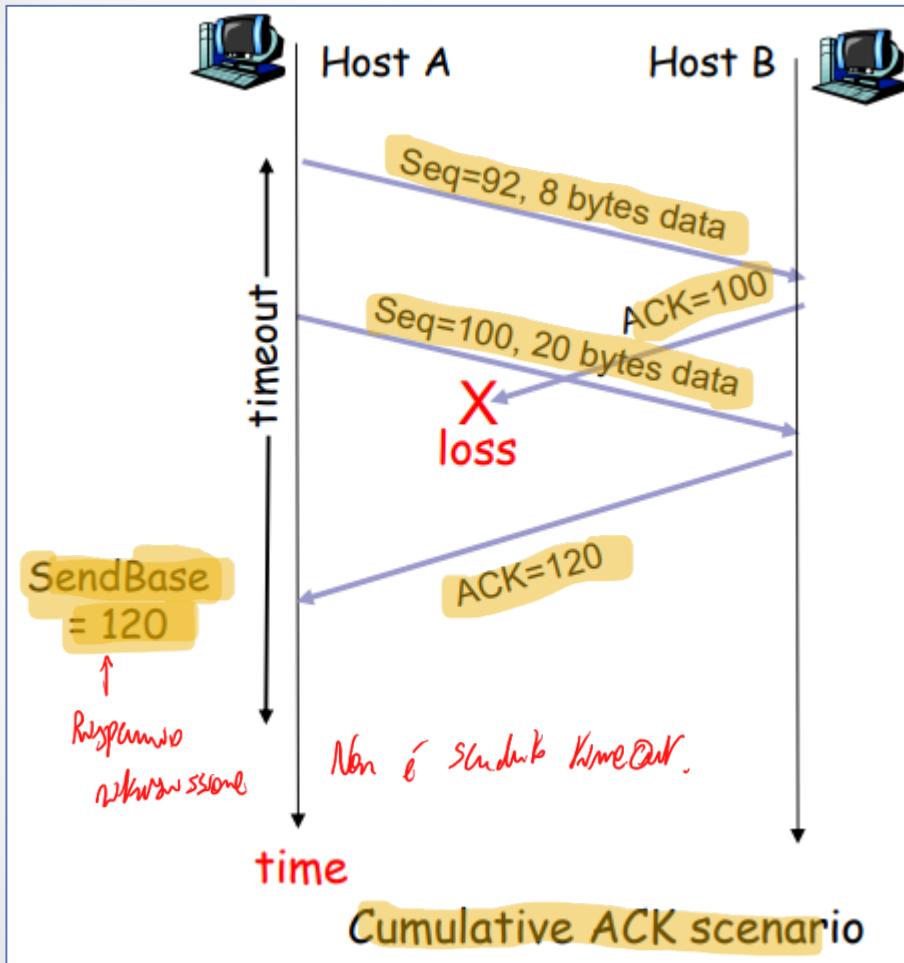
TCP ritrasmissione



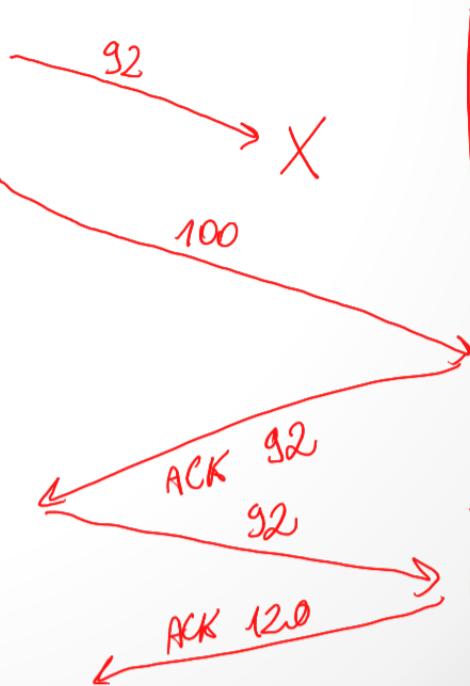
Timeout prematuro, azione dell' ACK cumulativo: il segmento 100 non è ritrasmesso

Torna del timeout solo sul primo pacchetto. Ciò possiede la bufferizzazione, dipende da implementazione.
Ma se scende timeout molto SOLO quello scadrà. Dopo solito si mette buffer.

TCP ritrasmissione



Il riscontro cumulativo
evita la ritrasmissione del
primo segmento



TCP generazione ACK [RFC 1122, RFC 2581]

| Evento Destinatario | Azione receiver TCP |
|--|--|
| Arrivo ordinato di un segmento con numero di sequenza atteso. Tutti i dati fino al numero di sequenza atteso sono già stati riscontrati. ✓ | ACK ritardato. Attende fino a 500 ms l'arrivo del prossimo segmento. Se il segmento non arriva, invia un ACK. → Aspetto un po' più ma sovraccarico rete |
| Arrivo ordinato di un segmento con numero di sequenza atteso. Un altro segmento è in attesa di trasmissione dell'ACK. | Invia immediatamente un singolo ACK cumulativo, riscontrando entrambi i segmenti ordinati. |
| Arrivo non ordinato di un segmento con numero di sequenza superiore a quello atteso. Viene rilevato un buco. | Invia immediatamente un ACK (duplicato), indicando il numero di sequenza del prossimo byte atteso. (Relativo al buco) |
| Arrivo di un segmento che colma parzialmente o completamente il buco. | Invia immediatamente un ACK, ammesso che il segmento cominci all'estremità inferiore del buco. Se eff. ho colmato buco. mn → mn → ACK da qui |

TCP modifiche

Raddoppio dell'intervallo di timeout

Allo scadere di un timeout:

si imposta il prossimo intervallo al doppio del valore precedente
(invece di usare la stima di RTT) – Crescita esponenziale degli
intervalli dopo ogni ritrasmissione *Se è scaduto il timer magari siamo congesti*

Quando il timer viene riavviato (ricezione di un ACK o di nuovi dati
dall'applicazione):

l'intervallo di timeout viene nuovamente configurato in funzione dei
valori più recenti di EstimatedRTT e DevRTT

Fornisce una forma limitata di controllo della congestione:

Il mittente, nel caso supponga una situazione di congestione (perdita
di un segmento), ritrasmette ad intervalli sempre più lunghi.

TCP modifiche

Ritrasmissione veloce

ACK duplicati:

Consentono di rilevare la perdita di un pacchetto prima del timeout
un receiver che rileva un “buco” nei segmenti ricevuti (ricezione
di un segmento con numero di sequenza maggiore di quello
atteso):

invia un nuovo riscontro per l’ultimo byte di dati che ha ricevuto
correttamente

poiché il mittente spesso manda molti segmenti contigui, se uno di
tali segmenti si perde, ci saranno molti ACK duplicati contigui:
un sender che riceve tre ACK duplicati per gli stessi dati assume
che il segmento successivo a quello riscontrato tre volte è andato
perso ed effettua, quindi, una ritrasmissione prima della scadenza
del timeout

Sarà per forza quello che manda del timeout

TCP modifie

event: ACK received, with ACK field value of y

```
if ( $y > \text{SendBase}$ ) {  
    \text{SendBase} =  $y$   
    if (there are currently not-yet-acknowledged segments)  
        start timer  
    }  
else {  
    increment count of dup ACKs received for  $y$   
    if (count of dup ACKs received for  $y = 3$ ) {  
        resend segment with sequence number  $y$   
    }  
}
```

a duplicate ACK for
already ACKed segment

fast retransmit

3 perché potremmo pure non arrivare da online.