PARTE 5b

LIVELLO TRASPORTO

Modulo 4: Protocollo TCP

Livello 4 (transport) [TCP]

- **Protocollo che fornisce un livello di trasporto affidabile ed orientato alla connessione**
- **Servizi aggiuntivi rispetto a UDP**
  - orientato alla connessione: comprende le fasi di instaurazione, utilizzo e chiusura della connessione
  - orientato al flusso di dati: considera il flusso di dati dall'host mittente fino al destinatario (→ considera sia rete sia host terminali)

Livello 4 (transport) [TCP]

- **Servizi aggiuntivi rispetto a UDP**
  - trasferimento con buffer: i dati sono memorizzati in un buffer e poi inseriti in un pacchetto quando il buffer è pieno
  - connessione full duplex (bi-direzionale): una volta instaurata una connessione, è possibile il trasferimento contemporaneo in entrambe le direzioni della connessione

Cosa il TCP non garantisce

- **Comunicazioni in tempo reale**
- **Garanzia di disponibilità di banda tra mittente e destinatario**
- **Multicast (un mittente, molti destinatari) affidabile**

TCP

Transmission Control Protocol

RFC: 793, 1122, 1323, 2018, 2581, 2988

Offre un livello di trasporto affidabile ed orientato alla connessione su di un canale inaffidabile

Problemi da affrontare a livello TCP

- **Eterogeneità degli host e dei processi in comunicazione**
  - c’è bisogno di un meccanismo per attivare e concludere una comunicazione in modo esplicito per entrambi
- **Eterogeneità dei tempi di trasmissione**
  - c’è bisogno di un meccanismo di timeout adattativo
- **Possibilità di ritardi molto lunghi nella rete**
  - c’è bisogno di gestire il possibile arrivo di pacchetti molto vecchi

Problemi da affrontare a livello TCP

- **Possibilità di avere un host destinatario con capacità molto diversa dall'host mittente**
  - c'è bisogno di gestire capacità dei nodi eterogenee
- **Possibilità di avere capacità di rete molto diverse**
  - c'è bisogno di gestire possibili congestioni dovute alla rete

Attributi di TCP [RFC 793]

1) Orientato alle connessioni

2) Trasmissione byte-stream

3) Connessione full-duplex

4) Affidabile

5) Trasmissione con buffer

1. Orientato alle connessioni

- **Connection oriented significa che:**
  - viene creata una connessione tra i due host prima del trasferimento di qualunque dato tra le applicazioni (e quindi in modo trasparente per l'applicazione)
  - viene chiusa dopo il completamento del trasferimento dati
- **3 fasi**
  - Instaurazione
  - Utilizzo
  - Chiusura
- **Il processo applicativo viene avvisato solo se:**
  - non si riesce a stabilire la connessione
  - la connessione viene interrotta

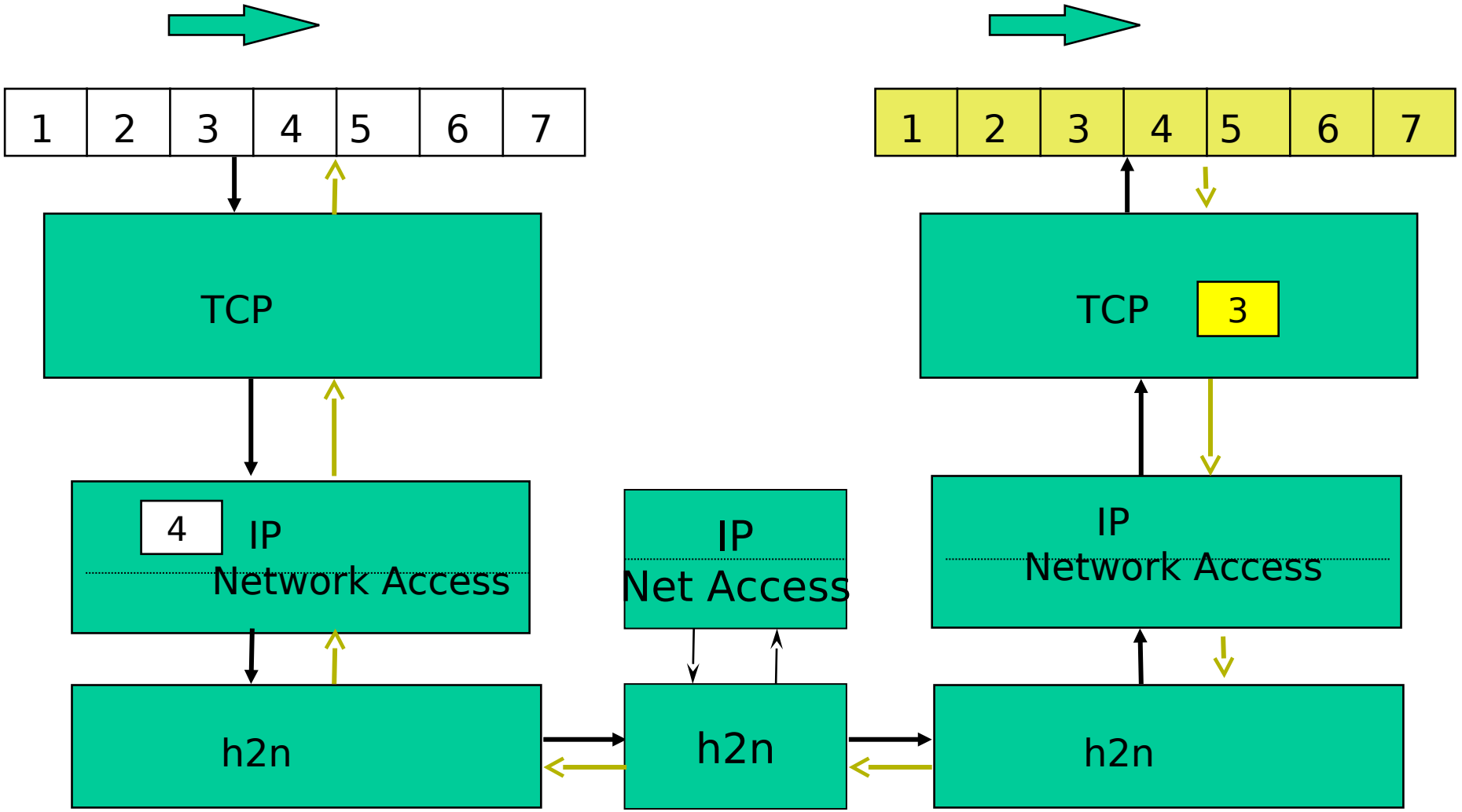
2. Trasmissione byte stream

- **Byte stream significa che la connessione viene trattata come un flusso di byte continuo dal mittente al destinatario**
- **L'unità di trasmissione nello stream è il byte**
- **Il processo applicativo mittente scrive byte**
- **Il livello TCP, per inviarli, accorpa i byte in un segmento TCP**
- **Il livello IP incapsula ogni segmento TCP in un datagram IP**
- **Il processo applicativo destinatario legge byte**

3. Connessioni full duplex

- **TCP può effettuare trasferimenti contemporanei in entrambe le direzioni della connessione, nell'ambito della stessa sessione**
  - Full duplex diverso da Half duplex
- **Ai processi applicativi, questi trasferimenti "appaiono" come due data stream non correlati**
- **Tuttavia, TCP consente di sovrapporre (piggybacking) comunicazioni di dati e comunicazioni di controllo, con l'invio di informazioni di controllo (es., ACK) insieme ai dati utente**

TCP e gli altri layer dello stack



NOTA: I router non gestiscono il livello TCP

4. Affidabile

- **Affidabile significa che TCP gestisce un trasferimento ordinato di uno stream di dati:**
- **acknowledgment + time-out+(ritrasmissione)**
  - Ogni trasmissione andata a buon fine viene notificata (acknowledged) dall'host ricevente
  - Se l'host mittente non riceve un acknowledgement entro un intervallo di tempo predefinito (time-out), il mittente ritrasmette i dati
  - Acknowledgment e ritrasmissioni dovute ad eventuali perdite sono gestite in modo trasparente rispetto al processo applicativo

5.a Trasmissione con buffer

- **Per far fronte a molti dei problemi evidenziati in precedenza, il layer TCP deve utilizzare necessariamente un buffer. Un buffer consente, infatti, di ovviare a:**
  - asincronia dell'invio dati da parte del processo applicativo
  - tempi di trasmissione differenti
  - capacità di invio e di ricezione differenti
  - segmenti persi o fuori ordine

5.a Trasmissione con buffer

- **Il livello TCP è responsabile della gestione del buffering dei dati e di determinare quando è tempo di inviare un certo insieme di dati. Alcune scelte:**
  - congestion control: l'host mittente deve diminuire il tasso di trasmissione dei pacchetti quando la rete è congestionata
  - flow control: l'host mittente non deve sovraccaricare l'host ricevente

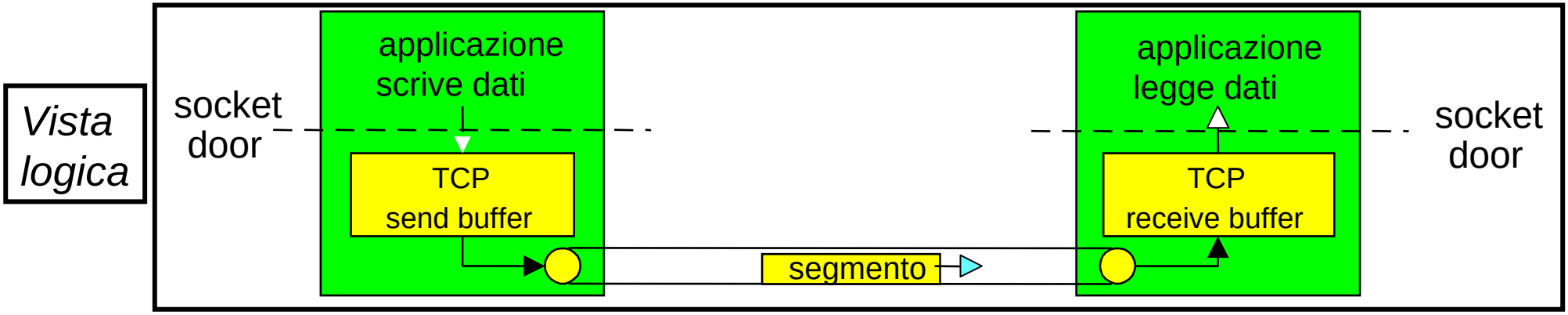
5.b Uso del buffer

1) Dati inseriti temporaneamente in un buffer del mittente e poi inseriti in un segmento TCP quando il segmento è pieno

Maximum Segment Size (MSS): viene concordato o di default dipende dall'implementazione TCP (es., 1460 byte, 536 byte, 512 byte)

Un'applicazione può anche specificare al protocollo TCP di inviare i dati che ha nel buffer, senza aspettare che il buffer sia pieno

2) Il segmento, a livello logico, viene poi inserito nel buffer del destinatario (in realtà, viene incapsulato in un datagram IP, spedito via rete e dopo aver attraversato i vari livelli, arriva nel buffer TCP di destinazione)



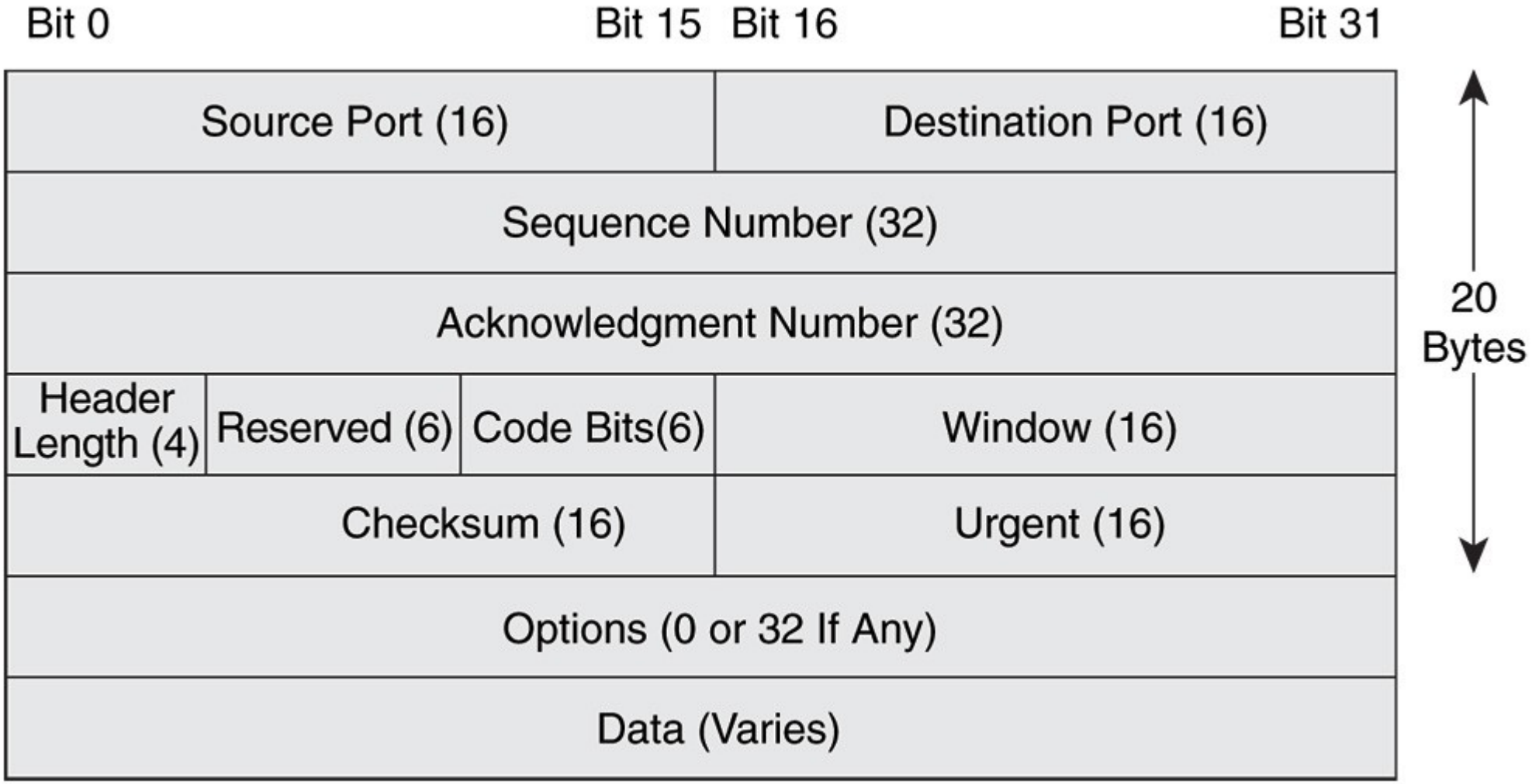
Modulo 5: Segmento TCP

Segmento TCP

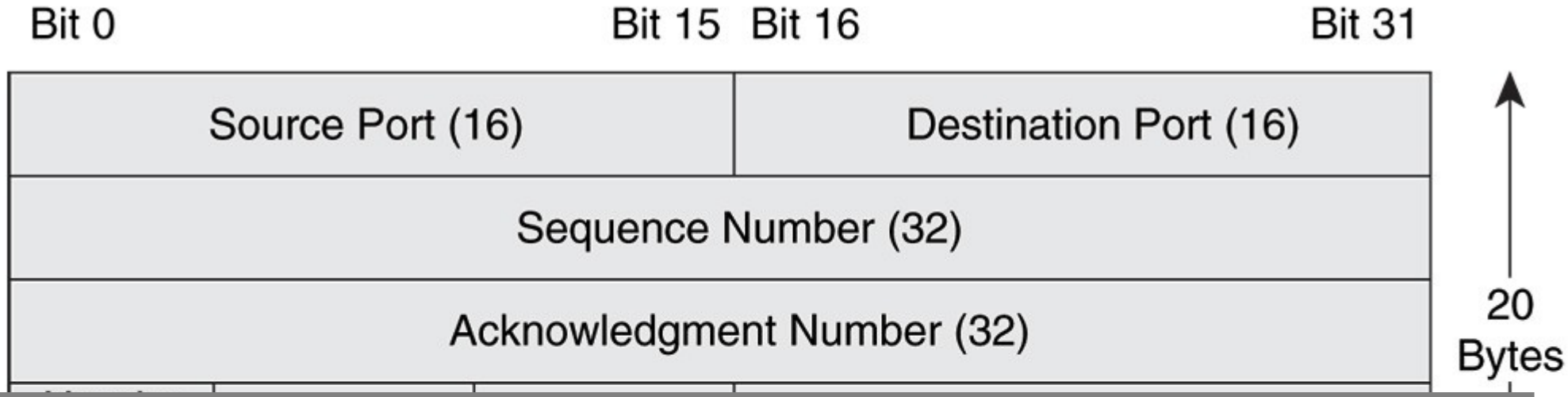
- **L’insieme di dati che il livello TCP chiede di trasferire al livello IP è detto segmento TCP**

- **Ogni segmento TCP contiene:**
  - Payload: dati del byte stream
  - Header: informazioni di controllo per identificare i byte dati

Formato del segmento TCP

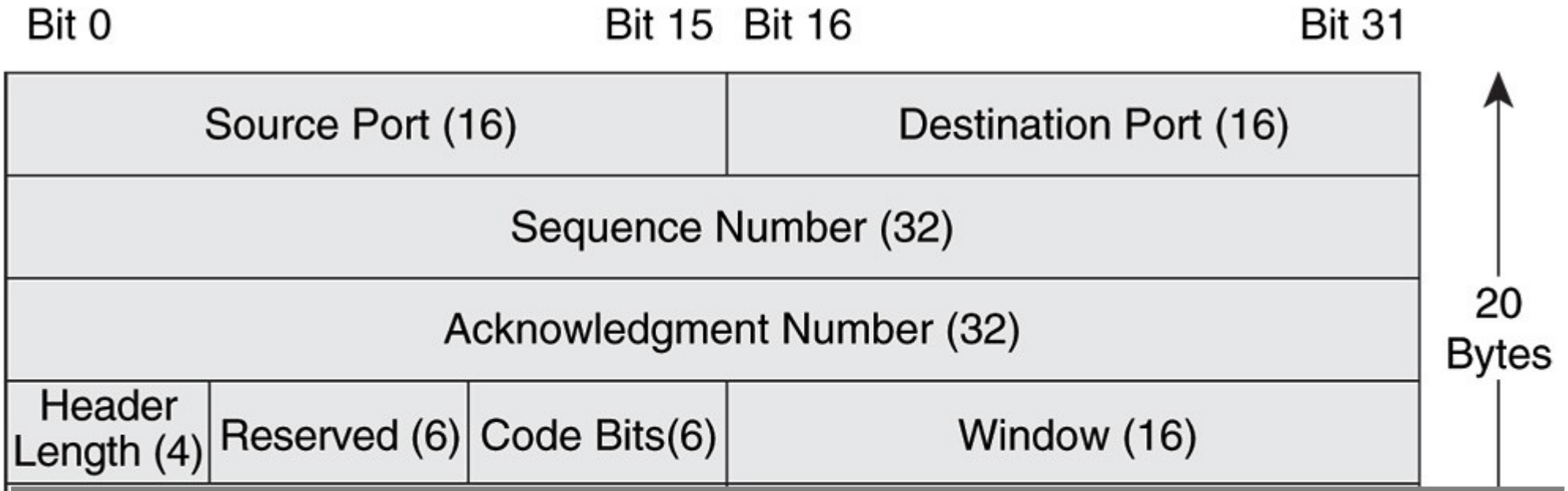


Formato del segmento TCP



- source port (16 bit): numero di porta del mittente
- destination port (16 bit): numero di porta del destinatario
- sequence number (32 bit): numero di sequenza relativo al flusso di byte che si sta trasmettendo
- acknowledgement number (32 bit): ACK relativo ad un numero di sequenza del flusso di byte che si sta ricevendo (poiché il flusso è bi-direzionale, vi è la possibilità di *piggybacking* → si vedrà in seguito)

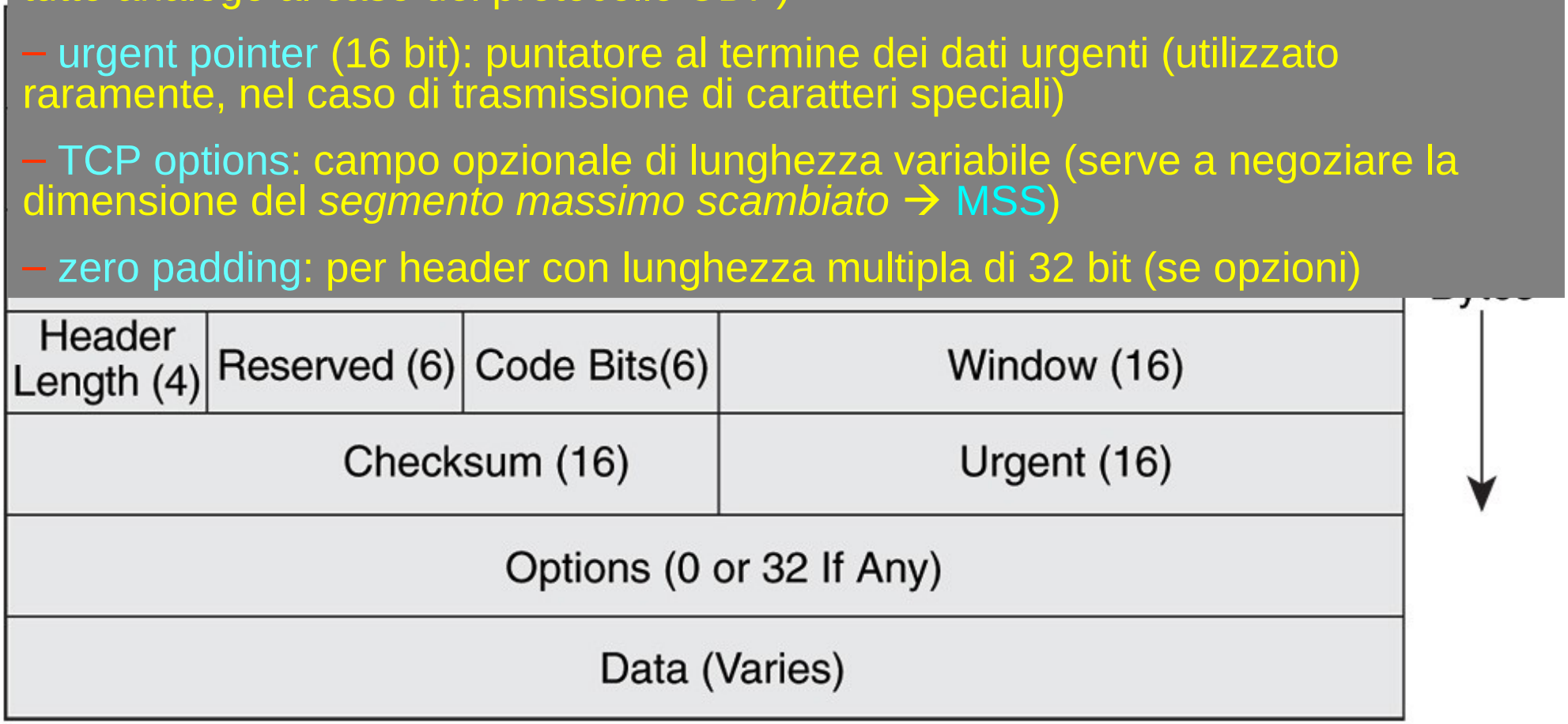
Formato del segmento TCP



- hlen (4 bit): lunghezza dell'header TCP (in multipli di 32 bit) se non vi sono opzioni → hlen = 20 byte
- reserved (6 bit): per usi futuri
- code bit (6 bit): scopo e contenuto del segmento
  - URG (urgent): dati segnati come urgenti dal livello applicativo
  - ACK (acknowledgement): valore del campo acknowledgement è valido
  - PSH (push): il destinatario deve passare i dati all'applicazione immediatamente
  - SYN (synchronize), FIN, RST (reset): usati per instaurazione, chiusura ed interruzione della connessione

Formato del segmento TCP

- window size (16 bit): dimensione della finestra in ricezione (indica il numero di byte che si è disposti ad accettare in ricezione)
- checksum (16 bit): controllo integrità dei dati trasportati nel segmento TCP (del tutto analogo al caso del protocollo UDP)
- urgent pointer (16 bit): puntatore al termine dei dati urgenti (utilizzato raramente, nel caso di trasmissione di caratteri speciali)
- TCP options: campo opzionale di lunghezza variabile (serve a negoziare la dimensione del *segmento massimo scambiato* → MSS)
- zero padding: per header con lunghezza multipla di 32 bit (se opzioni)



Checksum TCP

Utilizzato per rilevazione errori nei dati trasportati:

calcolato usando un maggior numero di informazioni di quelle presenti nell'header TCP (vi sono anche informazioni IP)

→ definizione di uno *pseudo-header* TCP

32 bit

indirizzo IP mittente		
indirizzo IP destinatario		
zero padding	protocollo	lunghezza TCP

- *zero padding*: dimensione dello pseudo-header, multiplo di 32 bit
- *protocollo*: campo protocollo del datagram IP
- pseudo-header anteposto al segmento TCP
- checksum calcolato su pseudo-header e intero segmento TCP
- pseudo-header *non* è trasmesso dal mittente

Dati urgenti (trasmissione fuori banda)

- **Servono a trasportare segnali speciali come ^C, ^Z, … in modo che possano essere recapitati immediatamente al processo applicativo**
- **All'arrivo all'host destinatario, scavalcano lo stream e vengono recapitati immediatamente al processo applicativo**
- **Il puntatore punta alla fine del blocco dei dati urgenti**
- **I dati urgenti iniziano all'inizio del segmento**

Negoziazione del MSS

- **Le TCP OPTIONS consentono di negoziare il Maximum Segment Size (MSS) per**
  - garantire che il segmento entri nei rispettivi buffer
  - evitare il più possibile la frammentazione al livello h2n
  - sfruttare al meglio la banda

Negoziazione del MSS

- **MSS troppo piccolo → overhead eccessivo dovuto agli header**
- **MSS troppo grande → elevati rischi di frammentazione nell'attraversamento dei livelli dello stack sottostanti IP-h2n**
- **→ Default MSS = 536 byte (il minimo possibile)**
- **Spesso si usa MSS = 1460 byte**

- **L'ACK viene mandato a livello del segmento originale. Quindi se un frammento viene perso, tutto il segmento deve essere riinviato**

Modulo 6: Instaurare e chiudere una connessione TCP

Instaurazione di una connessione

- **Nel TCP il mittente ed il destinatario, prima di iniziare il trasferimento dei segmenti contenenti i dati, instaurano la connessione**
- **Modello client/server**
  - client: inizia la connessione
  - server: deve essere già attivo, in attesa, viene contattato dal client
- **Inizializzazione delle variabili del TCP**
  - numeri di sequenza dei segmenti
  - informazioni necessarie per la gestione del buffer di trasmissione e ricezione

Instaurazione di una connessione

- **Quando un client richiede una connessione, invia un segmento TCP speciale, detto "SYN" segment (SYN sta per synchronize) al server**
- **Il client deve conoscere a chi spedire la richiesta, per cui nell'header del segmento deve specificare:**
  - La porta del server?
  - L'indirizzo IP?

SI
NO

- **Per accettare la connessione, il server deve essere già in attesa di ricevere connessioni**

Instaurazione di una connessione

Il segmento SYN del client include:

- **Initial Sequence Number (ISN) del client:**
  - un numero di 32 bit
  - Il numero è scelto in modo pseudo-casuale tra 0 e 2^32-1
  - Se il numero iniziale è 2032 e ci sono da spedire 5000 byte, tutti i byte saranno numerati da 2032 a 7031
  - La numerazione dei byte in una direzione è indipendente dalla numerazione dei byte nell'altra direzione
  - Nell'intestazione di ogni segmento è riportato solo il numero di sequenza del primo byte dei dati contenuto nel segmento. Gli altri byte del segmento sono numerati di conseguenza

Instaurazione di una connessione

Il segmento SYN del client include anche:

- **Maximum Receive Window (MRW) del client: il massimo numero di byte che il client è in grado di ricevere nel suo buffer – necessario per la regolazione del flusso dello stream di byte**
- **Maximum Segment Size (MSS): la massima dimensione del segmento (informazione non sempre inviata)**
- **NON HA payload (dati del messaggio), ma solo il TCP header!**

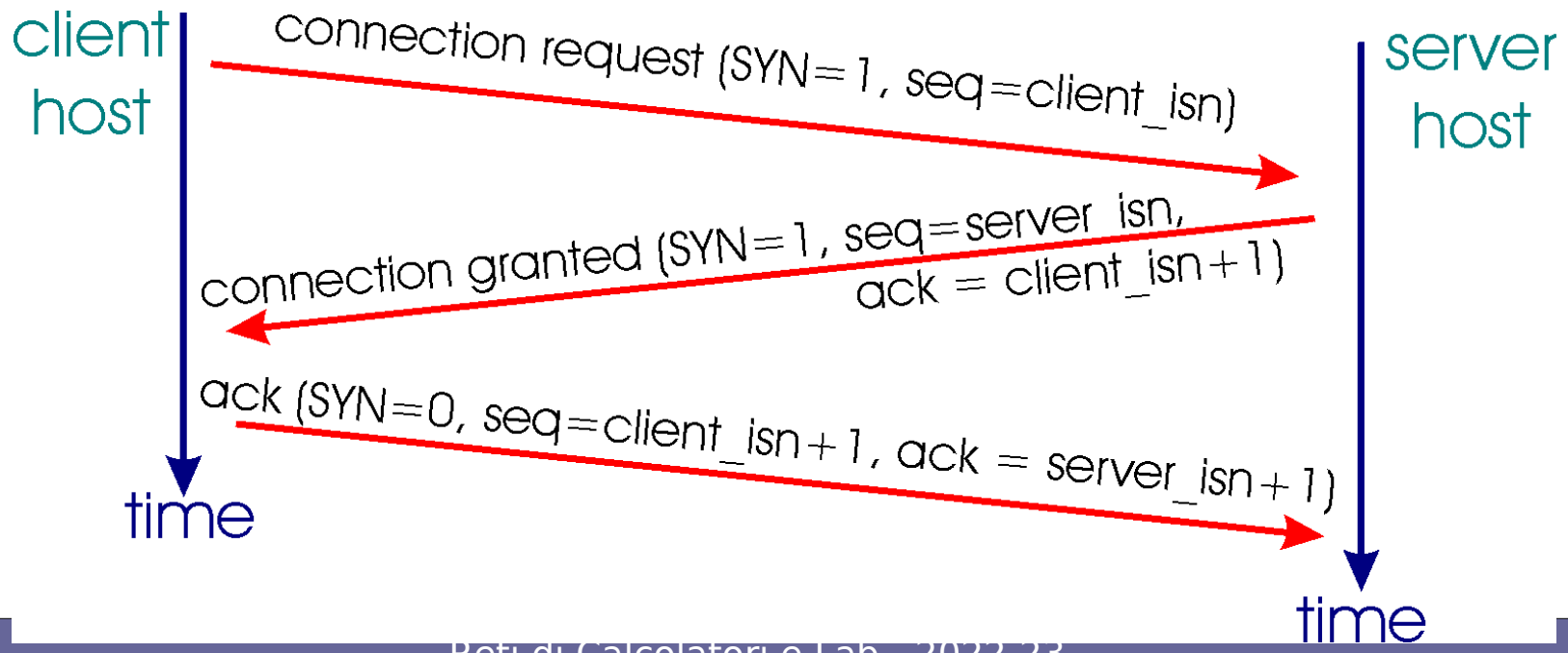
Instaurazione di una connessione

Il segmento SYN del server include:

- **Initial Sequence Number (ISN) del server: un numero pseudo-casuale**
- **ACK del server: client_ISN+1**
- **Maximum Receive Window (MRW) del server**
- **Maximum Segment Size (MSS): la massima dimensione del segmento (informazione non sempre inviata)**
- **NON HA payload (dati del messaggio), ma solo il TCP header**

Three-way handshaking

- Il client invia al server un segmento di controllo con SYN=1, e specifica nello stesso segmento il proprio numero iniziale di sequenza (*client_isn*)
- Il server riceve il segmento del client con SYN=1
- Se accetta, il server invia un segmento di controllo con SYN=1, ACK=*client_isn*+1, ed il proprio numero iniziale di sequenza (*server_isn*)
- Il client segnala la definitiva apertura della connessione inviando un segmento di controllo con SYN=0, ACK=*server_isn*+1, e numero di sequenza *client_isn* +1



Three-way handshacking

1) **Client: “Hello server, voglio parlare con te, e comincerò con il byte che indicherò con il numero X”**

2) **Server: “OK, sono disponibile a parlare. Il mio primo byte sarà indicato con il numero Y e so che il tuo prossimo byte sarà indicato con X+1”**

3) **Client: “D’accordo, so che il prossimo byte che mi invierai sarà indicato con il numero Y+1”**

Chiusura (polite) della connessione

- Il client invia un segmento di controllo con bit FIN=1 al server

- Il server riceve FIN, invia ACK

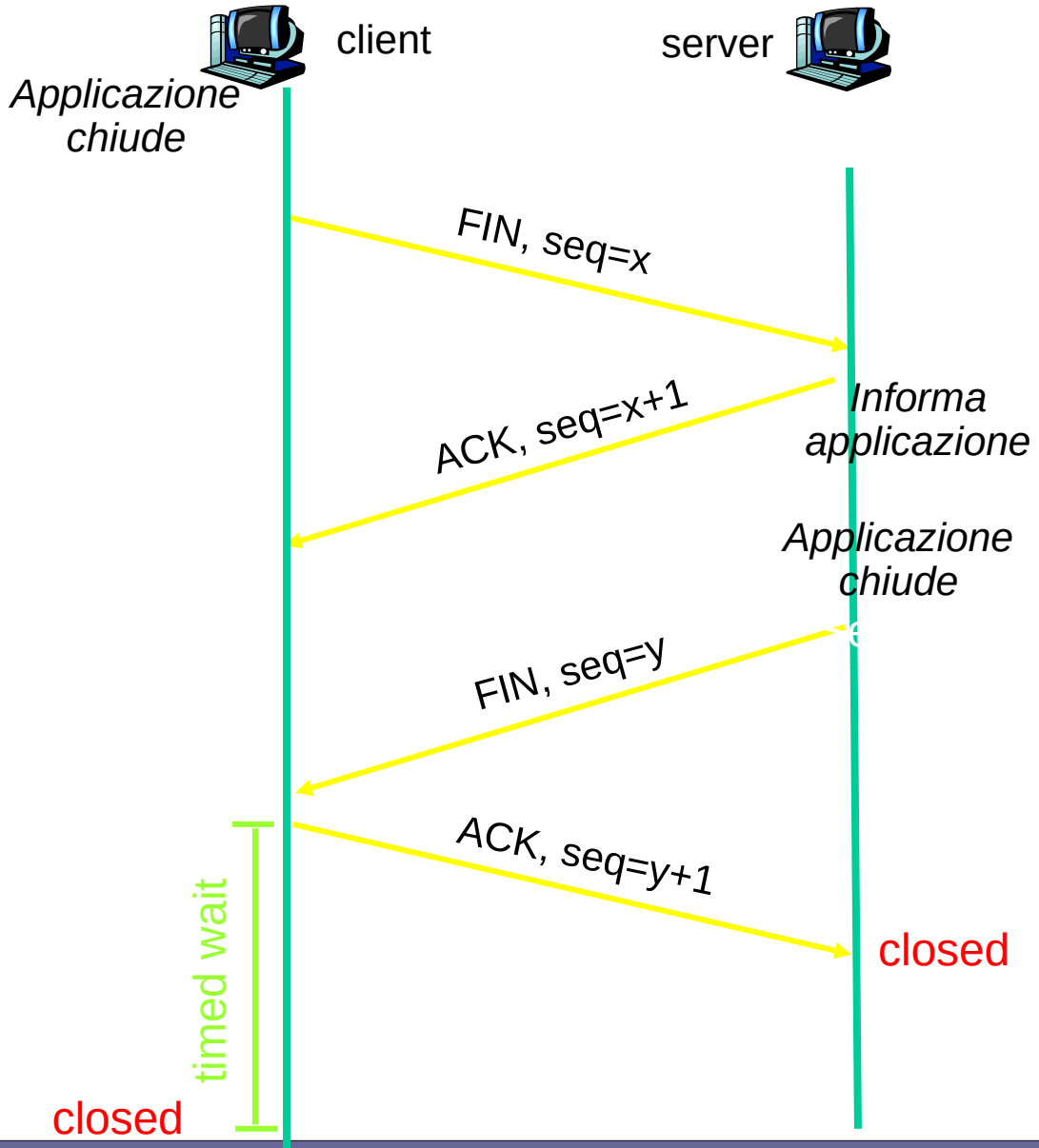
- Il server chiude la connessione lato client-server ed invia FIN=1 al client

- Il client riceve il segmento con FIN=1 ed invia ACK

- Il server riceve ACK

- Il client attende il timeout dell'ACK inviato; allo scadere anche la connessione lato server-client viene chiusa

Perché tante "fasi" nella chiusura?



Chiusura (polite) della connessione

- **Dopo che la connessione TCP è stata chiusa (ovvero è stato inviato l'ultimo ACK dal client), ci potrebbe essere ancora qualcosa da fare. Per esempio:**
  - **Se l'ultimo ACK si perde?**

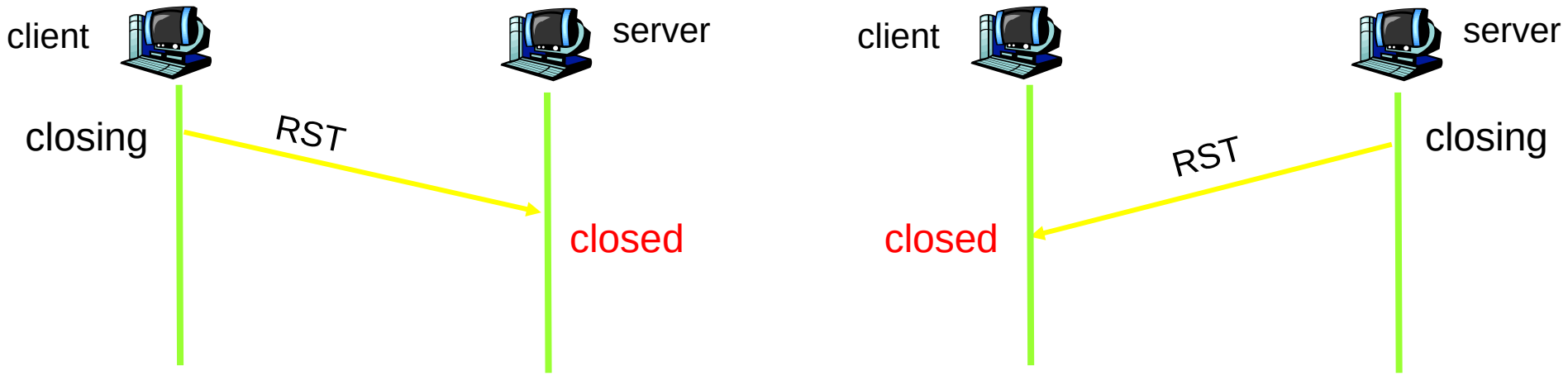
    → L'ultimo segmento FIN verrà inviato nuovamente, e dovrà essere notificato con un ACK
  - **Se segmenti persi o duplicati dovessero raggungere la destinazione dopo un lungo ritardo?**

- **Quindi, il client TCP attende per un tempo TIME_WAIT (es., 30 secondi) prima di chiudere definitivamente la connessione per poter gestire queste situazioni anomale**

Chiusura (reset) della connessione

In condizioni normali, la connessione viene chiusa in modo “polite” tramite lo scambio di segmenti di controllo FIN e ACK, visto in precedenza

- Talvolta, si verificano condizioni che portano ad interrompere la connessione in modo “brusco”, o dal lato server (tipicamente, per errori o sovraccarico) o dal lato client (tipicamente, in seguito a azione dell’utente)
- TCP fornisce un meccanismo per la chiusura rapida: reset
- L’host che decide il reset pone il campo del segmento RST=1
- L’altro nodo chiude immediatamente la connessione
- Vengono rilasciate tutte le risorse utilizzate dalla connessione



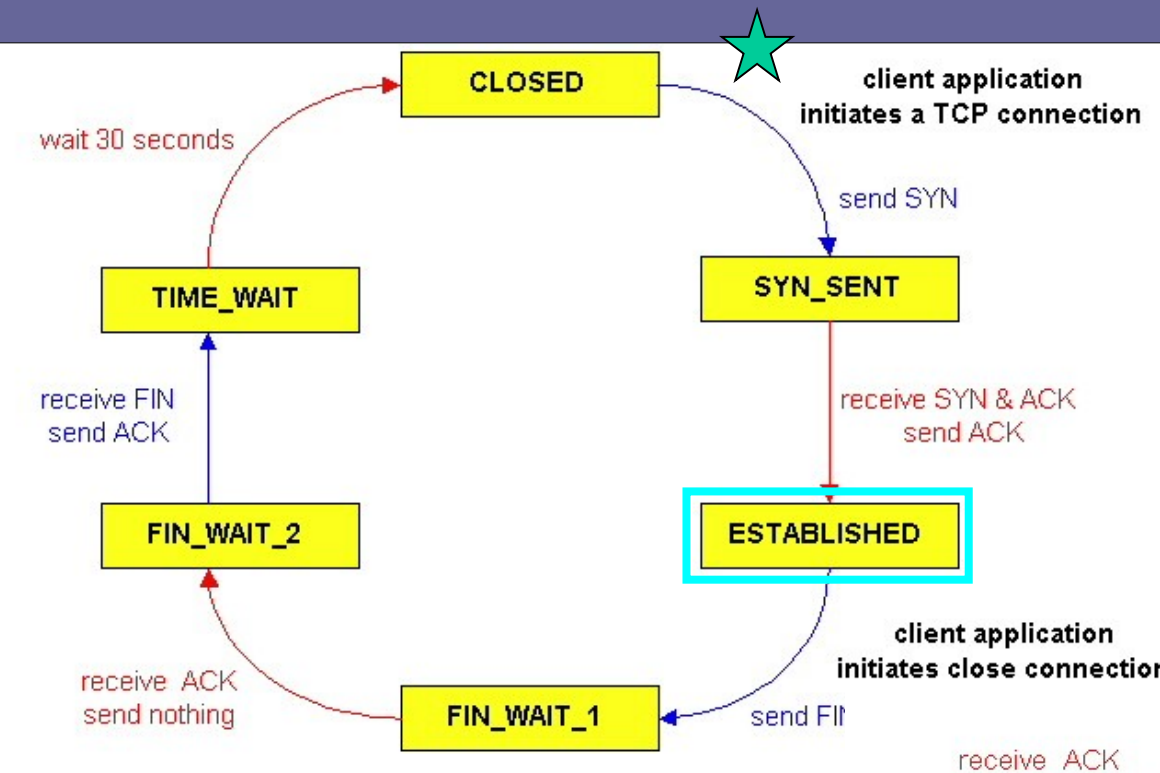
Protocollo TCP: funzionamento

Il trasferimento dati tra host end-to-end avviene

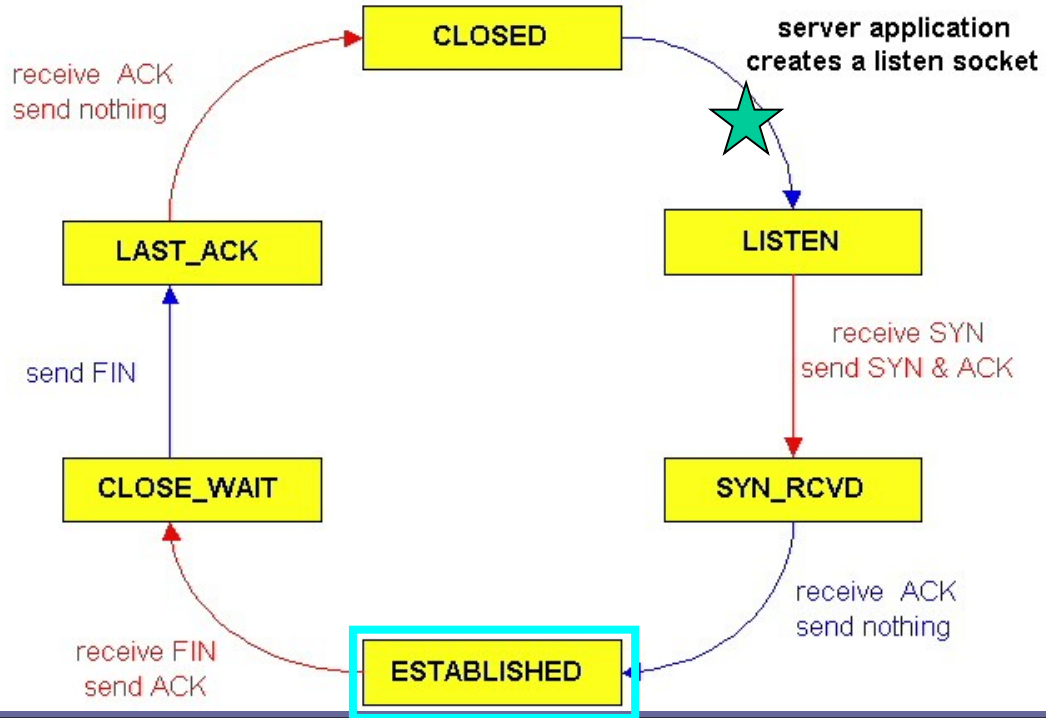
in 3 fasi:

- **1. Handshaking: fase di setup (con trasferimento di 3 segmenti) in cui ci si prepara al trasferimento dei dati**
  - Si deve arrivare ad uno stato di setup riconosciuto da parte di entrambi gli host che devono comunicare
- **2. Trasmissione: fase di trasmissione (bidirezionale) di uno o più segmenti**
- **3. Chiusura connessione**

Ciclo di vita delle connessioni TCP



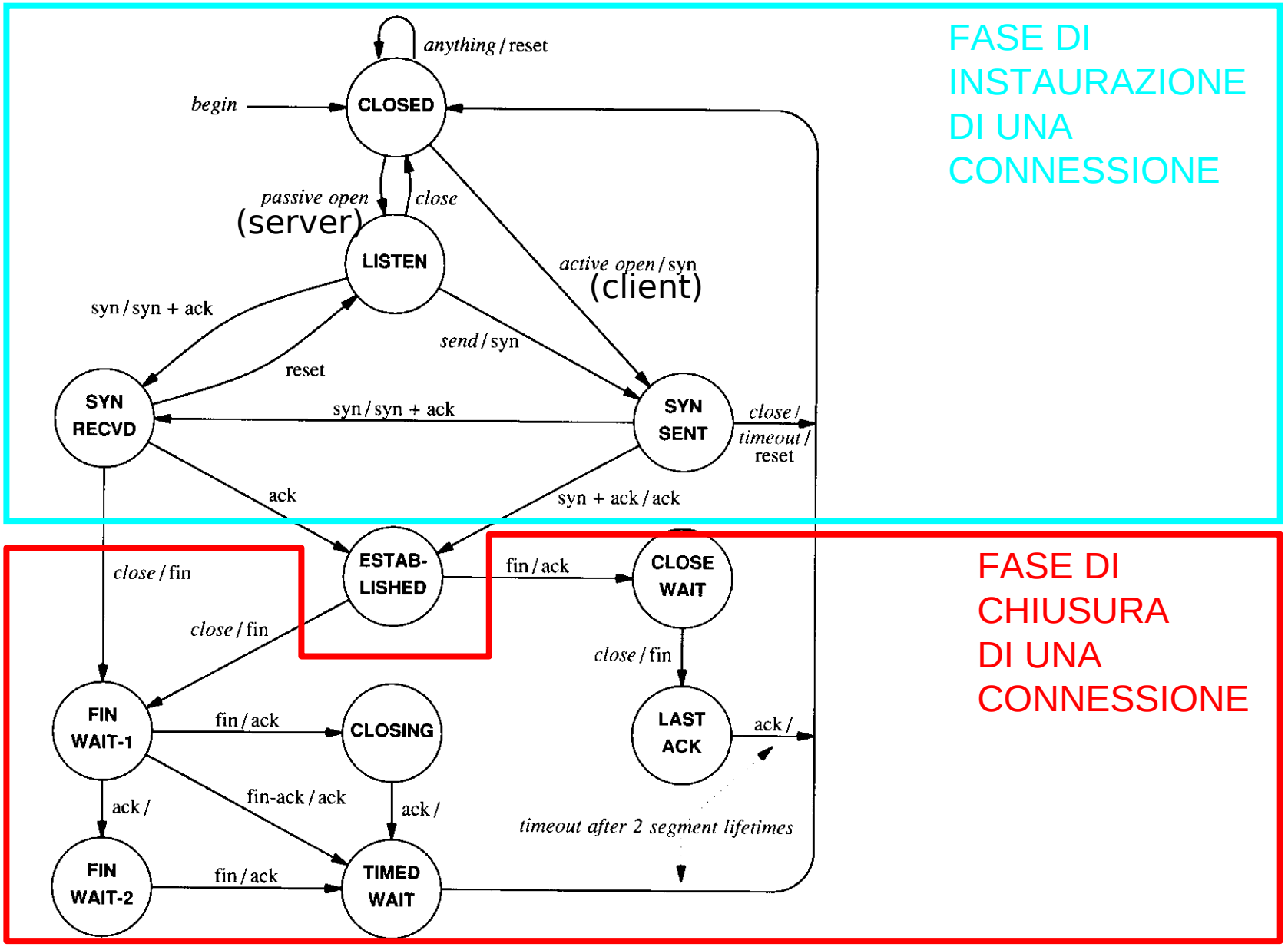
Ciclo di vita del client TCP



Ciclo di vita del server TCP

NOTA
I diagrammi si riferiscono solo all'apertura-chiusura delle connessioni. Ciò che succede durante la connessione è dentro lo stato "established"

Diagramma delle transizioni di stato del protocollo TCP



Modulo 7:
Affidabilità del protocollo TCP

Meccanismi per l'affidabilità

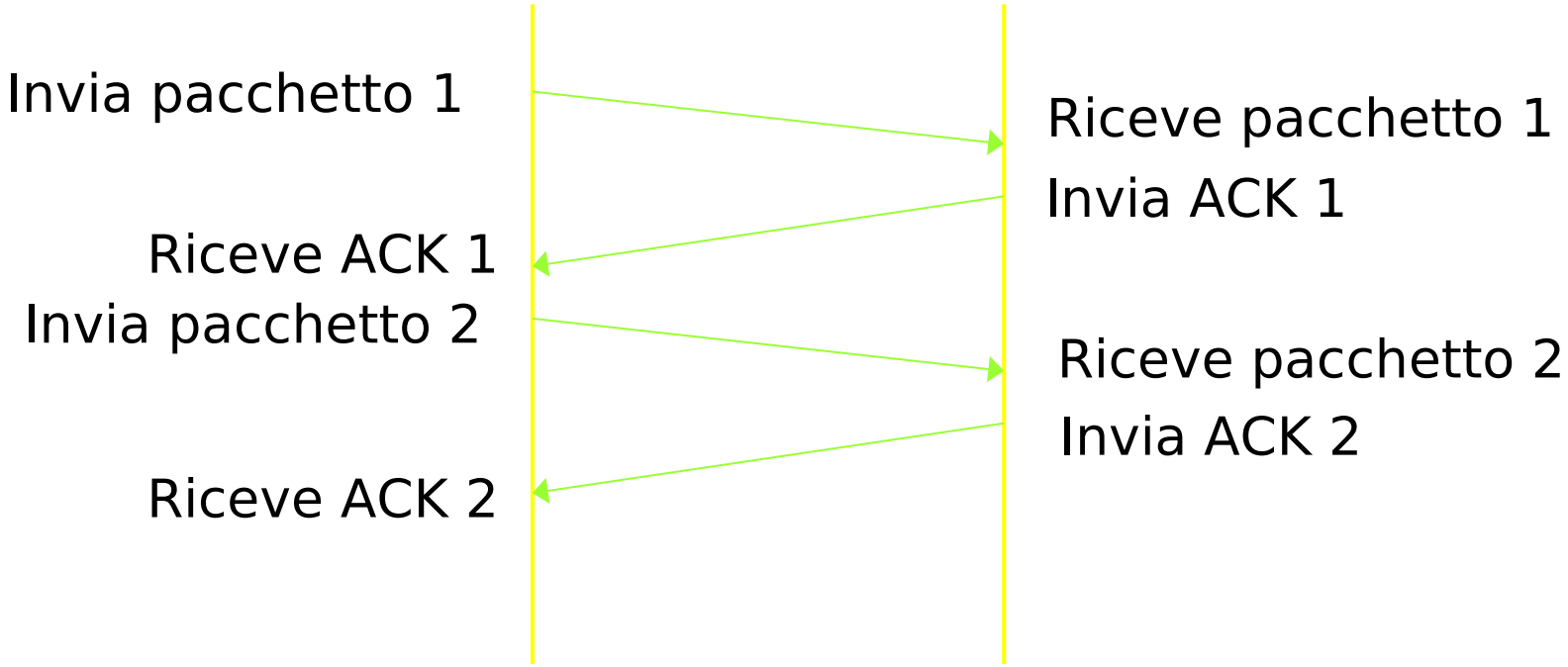
- **Acknowledgment (positivo)**
- **Time-out**
- **Ritrasmissione**

Meccanismi gestiti dal livello TCP in modo del tutto trasparente rispetto al processo applicativo

- **Principî:**
  - Ogni trasmissione andata a buon fine viene notificata (acknowledged) dall'host ricevente
  - Se l'host mittente non riceve un acknowledgement entro il time-out, il mittente ritrasmette i dati

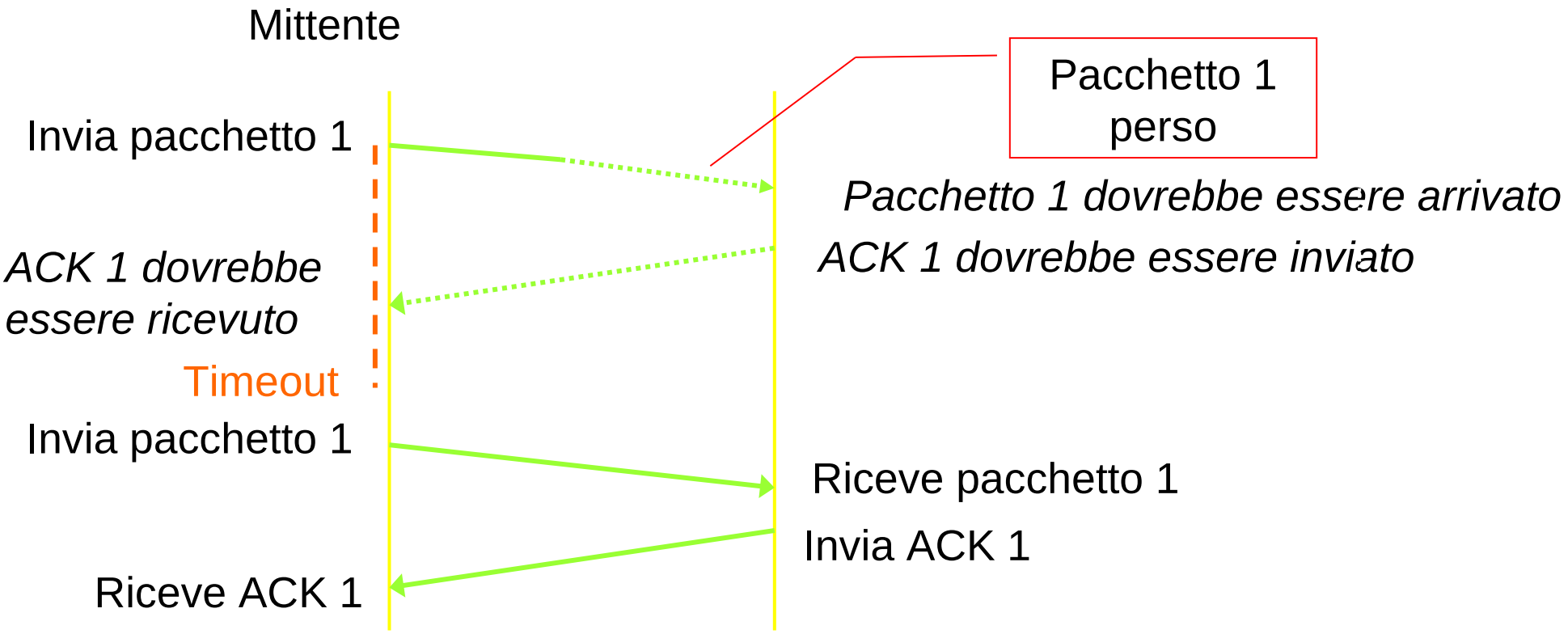
Affidabilità nel protocollo TCP

- **Affidabilità:**
  - uso della tecnica di acknowledgement positivo con ritrasmissione
  - Il destinatario, quando riceve i dati, invia un acknowledgement (ACK) al mittente, che attende di ricevere un ACK prima di inviare il segmento successivo.



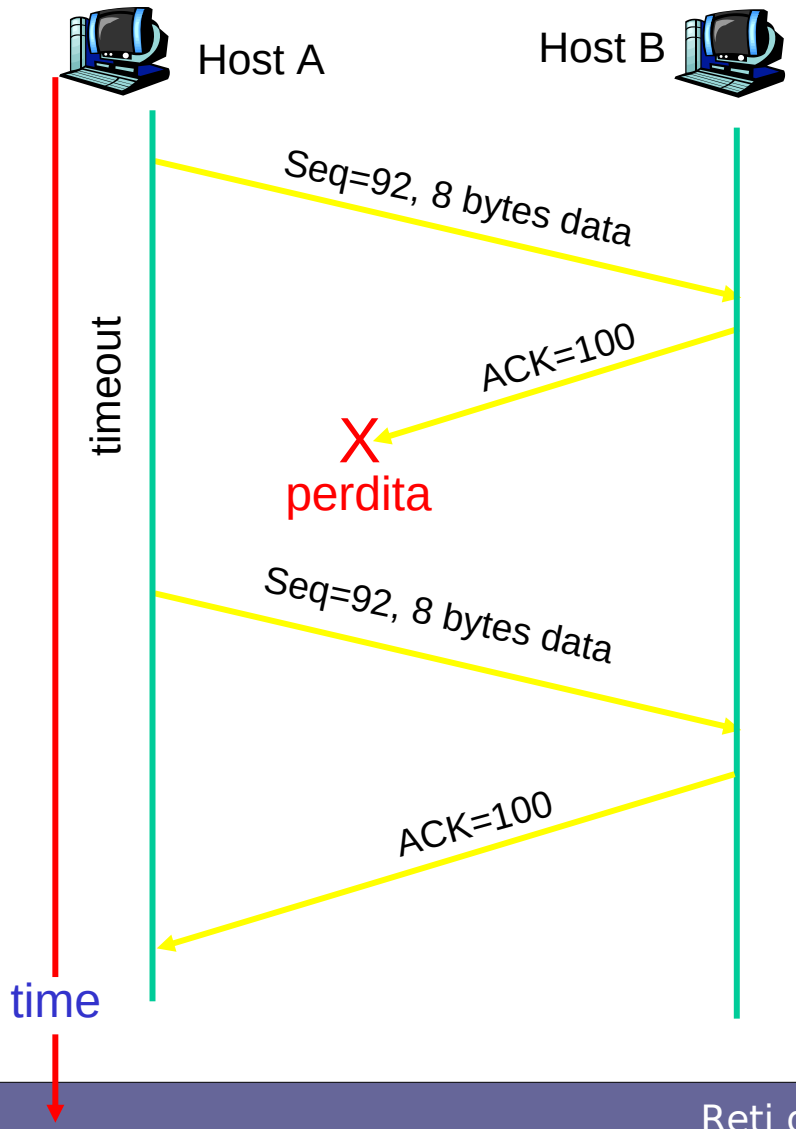
Scenari di ritrasmissione

Affidabilità: uso della tecnica di acknowledgement positivo con timeout e ritrasmissione → se il mittente non ha ricevuto ACK di un segmento dopo un certo periodo (*timeout*), ritrasmette il segmento

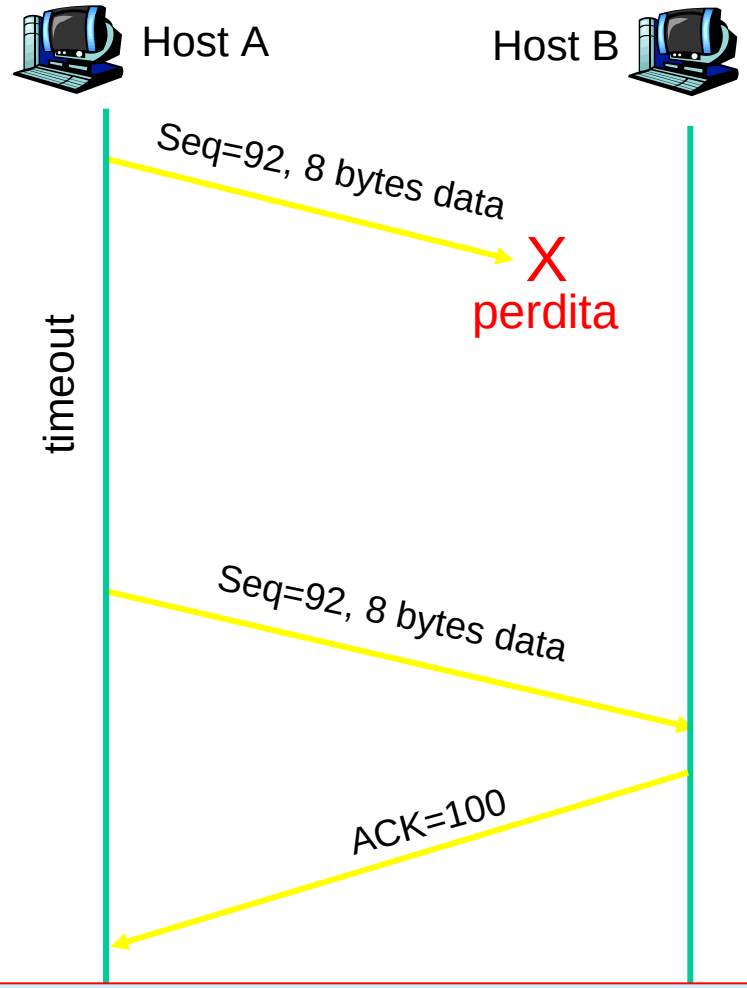


Scenari di ritrasmissione (protocollo stop-and-wait)

Ritrasmissione causata dalla perdita dell'ACK



Ritrasmissione causata dalla perdita del segmento



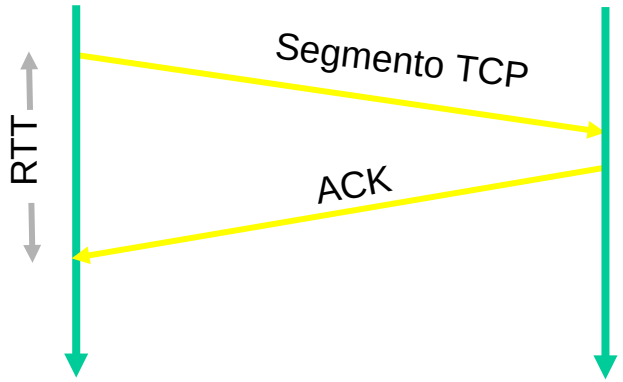
In entrambi i casi, l'ack=100 non arriva.
Nel primo caso, c'è una duplicazione.

Come stimare il “time-out”?

Round trip time e timeout

Come stabilire il valore del timeout in trasmissioni TCP?

- **Timeout troppo breve: si effettuano ritrasmissioni non necessarie**
- **Timeout troppo lungo: reazione lenta alla perdita di segmenti**

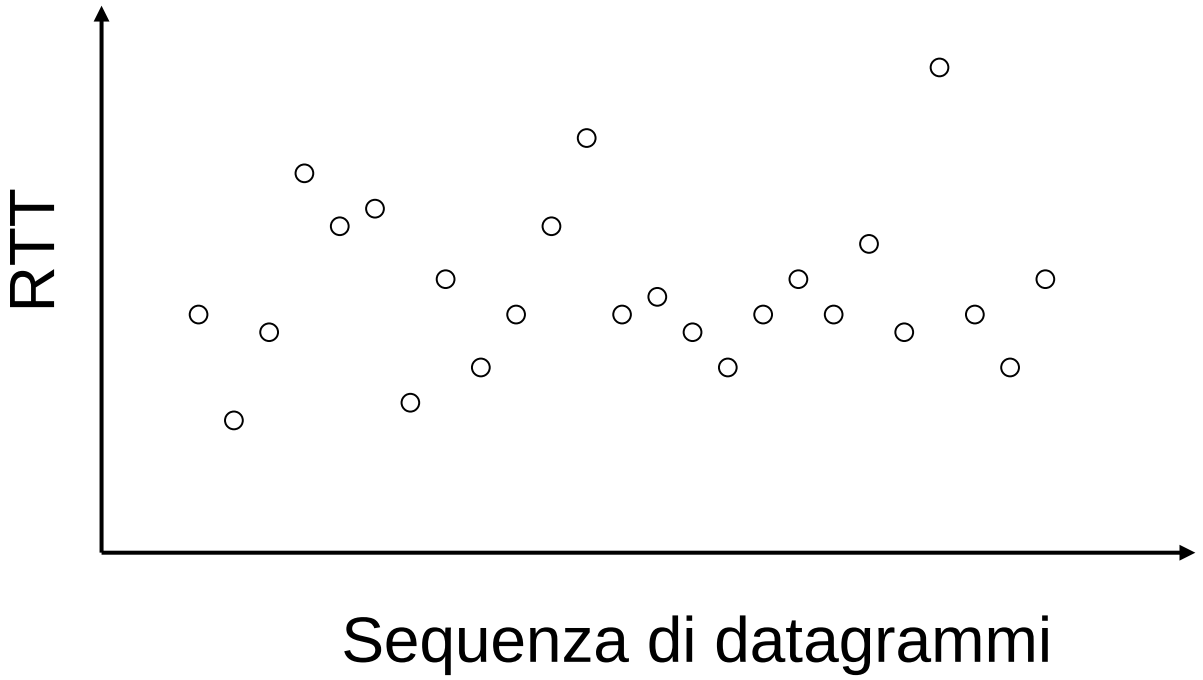


UNICA CERTEZZA: Il timeout deve essere maggiore del Round Trip Time (RTT). Ma di quanto?

RTT varia continuamente

Motivazioni delle differenze del RTT:

- **fluttuazioni delle condizioni di traffico della rete**
- **possibili cambiamenti di router nel percorso tra mittente e destinatario**



Scelta del Timeout (old)

Inizialmente si sceglieva:

Timeout = beta * RTTmedio

dove la raccomandazione era: beta = 2

Adesso, si utilizzano stime molto più sofisticate

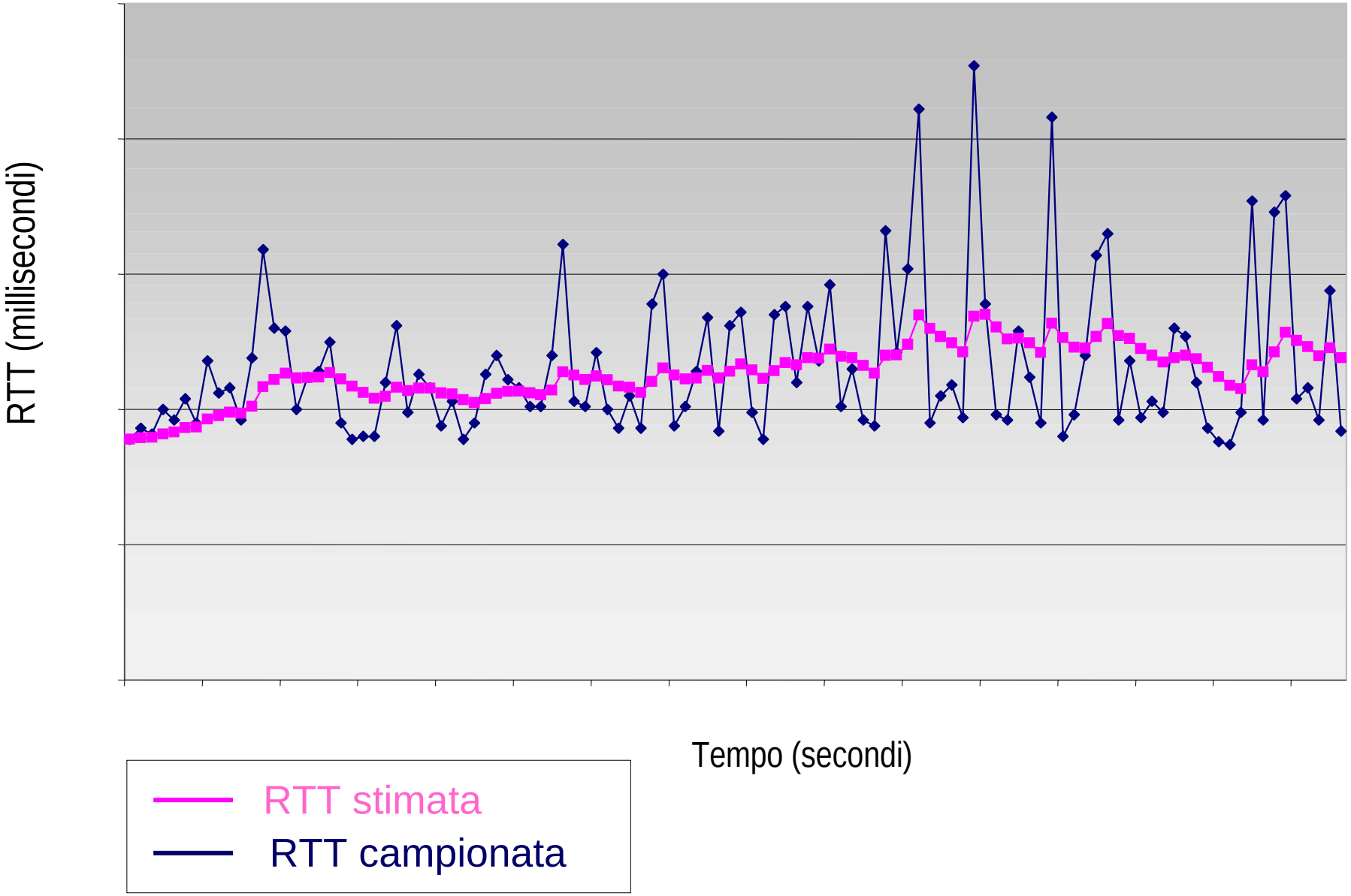
Scelta del Timeout

- **SampleRTT:**
  - misura del tempo trascorso dalla trasmissione del segmento alla ricezione del suo ACK
  - Ignora ritrasmissioni, segmenti con ack cumulativi
  - SampleRTT varia dinamicamente → si usa una media pesata

Scelta del Timeout

- **EstimatedRTT:**
  - media pesata per stimare RTT al tempo t
  - *EstRTT(t) = (1-x)*EstRTT(t-1) + x*SampRTT(t)*
  - Exponential Weighted Moving Average (EWMA)
  - L'influenza dei campioni passati diminuisce in modo esponenziale
  - Il valore di x è compreso tra 0 e 1. Inizialmente, si sceglie tipicamente
  - x=1/(n+1) dove n è il numero di campioni di RTT usati per il calcolo

Esempio della stima RTT



Scelta del Timeout

- **Scelta del timeout:**
- **Valore del RTT stimato più un margine di errore**
- *Timeout(t) = EstRTT(t) + 4*Dev(t)*

  Dove:
- *Dev(t) = (1-x)*Dev(t-1) + x*abs[SampRTT(t)-EstRTT(t)]*

Modulo 8:
Gestione del buffer TCP

Motivazioni per buffering: asincronia

- **Il TCP fa parte del Sistema Operativo, non del livello applicativo che gestisce l'invio e la ricezione dei dati**
- **Deve tener conto di tutti gli eventi che si verificano in modo asincrono:**
  - Il livello TCP del mittente non sa quando il processo applicativo deciderà di spedire dati
  - Il livello TCP del destinatario non sa quando il processo applicativo accetterà (o chiederà) di prendere i dati arrivati

Motivazioni per buffering: asincronia

- **→ L’unica possibilità è inserire temporaneamente i dati ricevuti in un buffer, in modo da poterli ricevere/trasmettere dal/al livello applicativo quando vengono inviati/richiesti**
- **→ CONTROLLO DI FLUSSO (si vedrà in seguito)**

Motivazioni per buffering: prestazioni

- **Il meccanismo di trasporto stop-and-wait è estremamente affidabile e semplice da implementare**
- **Tuttavia, utilizza le risorse di rete e degli host in modo non efficiente**
- **→ Il buffering consente di passare da un protocollo stop-and-wait al pipelining**
- **MOTIVAZIONI...**

Prestazioni *(alcune definizioni)*

- **Round Trip Time (RTT): tempo impiegato dal pacchetto per andare dal mittente al destinatario e ritorno**
- **Tempo di propagazione: RTT/2**
- **Utilizzazione: percentuale di utilizzo di una risorsa**
- **Rate = transmission rate**
- **L(pkt) = lunghezza pacchetto**

Prestazioni *(alcune definizioni)*

- **Tempo di trasmissione pacchetto**

  L(pkt) / Rate

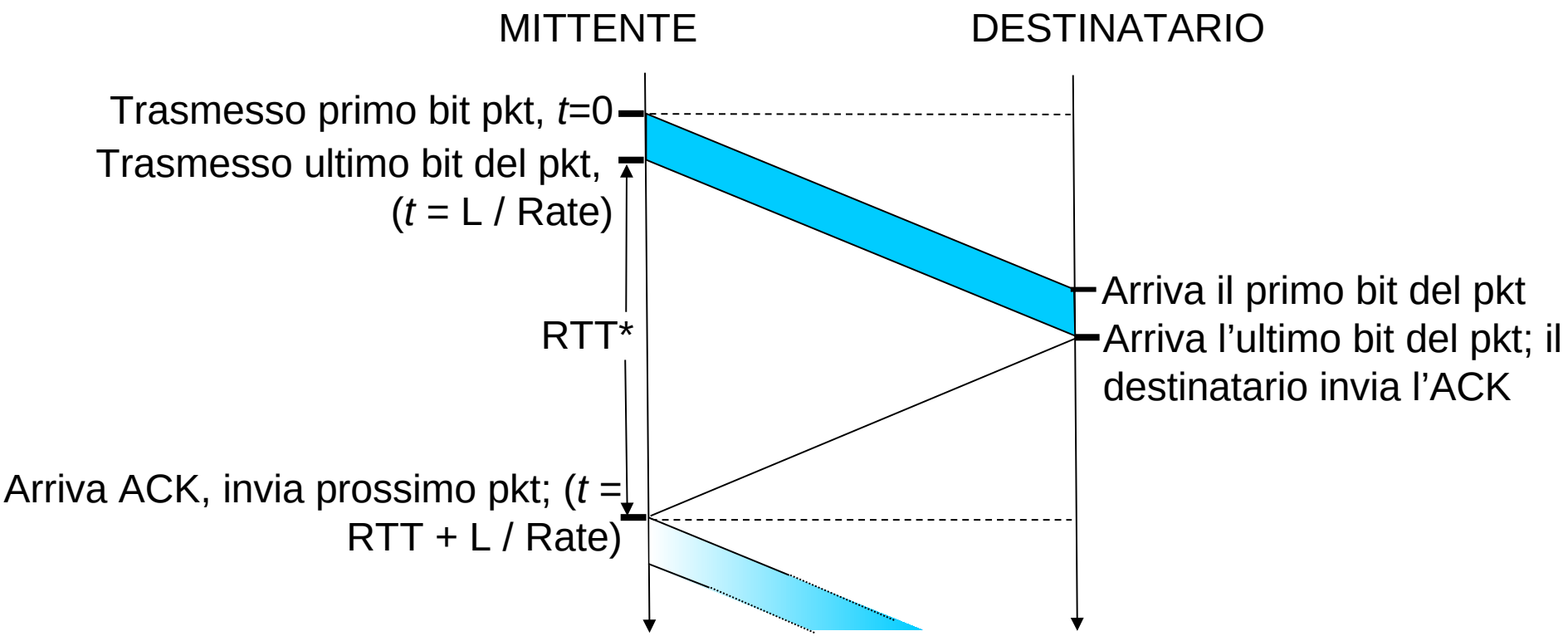
  [dimensionalmente: (bit) / (bit/sec) = sec]

- **Tempo di trasferimento pacchetto (vista mittente)**

  [propagazione + trasmissione] + [propagazioneack]

  = [RTT/2pkt + L(pkt) / Rate] + [RTT/2ack]

Prestazioni protocollo stop-and-wait



RTT*= Round Trip Time ideale (se il destinatario non è rallentato da altri processi)

Prestazioni protocollo stop-and-wait

Esempio

- Canale fisico di capacità 1 Gbps (= 10^9 bit/sec)
- Ritardo di propagazione (RTT/2) = 15 msec
- Pacchetto da 1KB (=8 Kb)

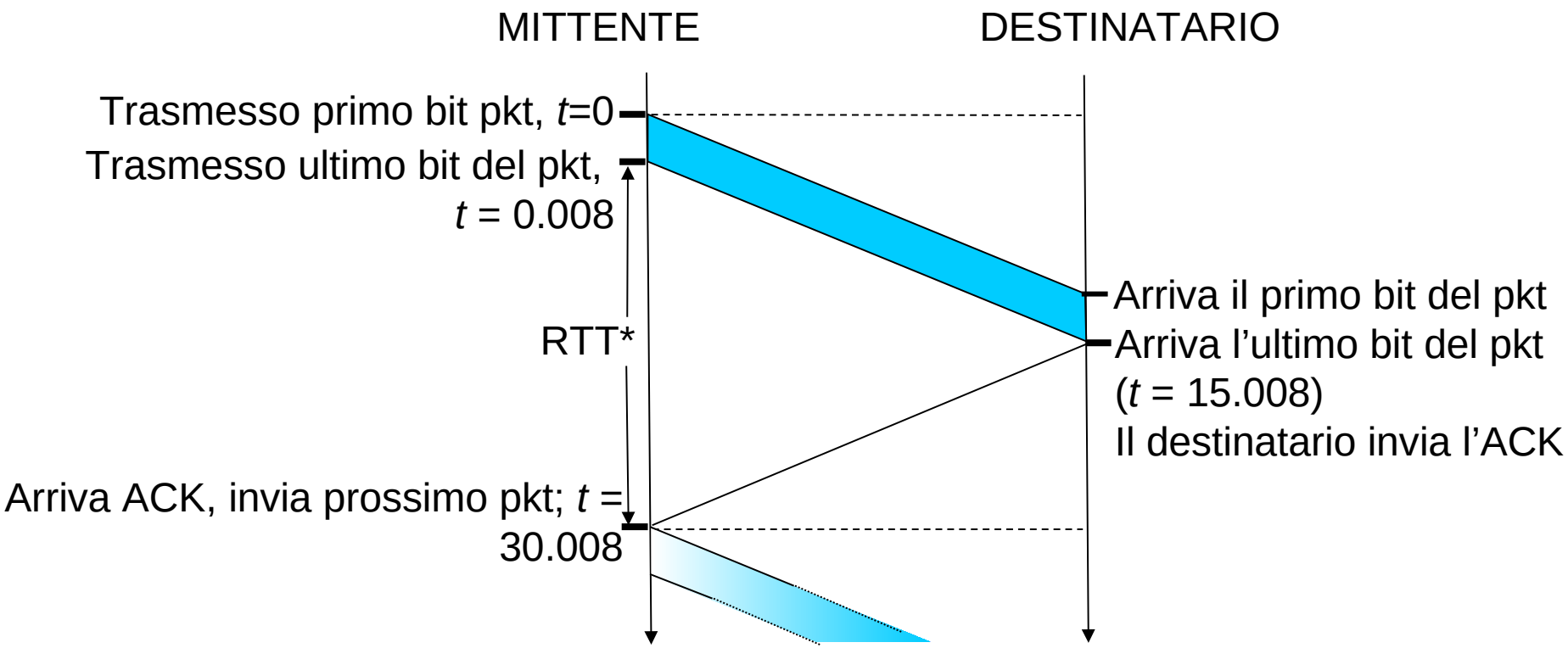
Rate = *transmission rate*
L(pkt) = lunghezza pacchetto

Calcolare il tempo di trasmissione pacchetto:

$$T_{trasm/pkt} = \frac{L(pkt)}{Rate} = \frac{8Kb}{10^9 \text{ bit/sec}} = \frac{8 \cdot 10^3 \text{ bit}}{10^9 \text{ bit/sec}} =$$

$$= \frac{8 \text{ bit}}{10^6 \text{ bit/sec}} \qquad = 0.000008 \text{ sec} = 0.008 \text{ msec}$$

Prestazioni protocollo stop-and-wait



$$\text{Utilizzazione}_{(\textit{mittente})} = \frac{\text{Tempo trasm.}}{\text{Tempo totale}} = \frac{T}{RTT + T} = \frac{L/R}{RTT + L/R} = \frac{0.008}{30 + 0.008} = 0.00027$$

In pratica, si riesce a trasmettere ~1 KB ogni 30 msec

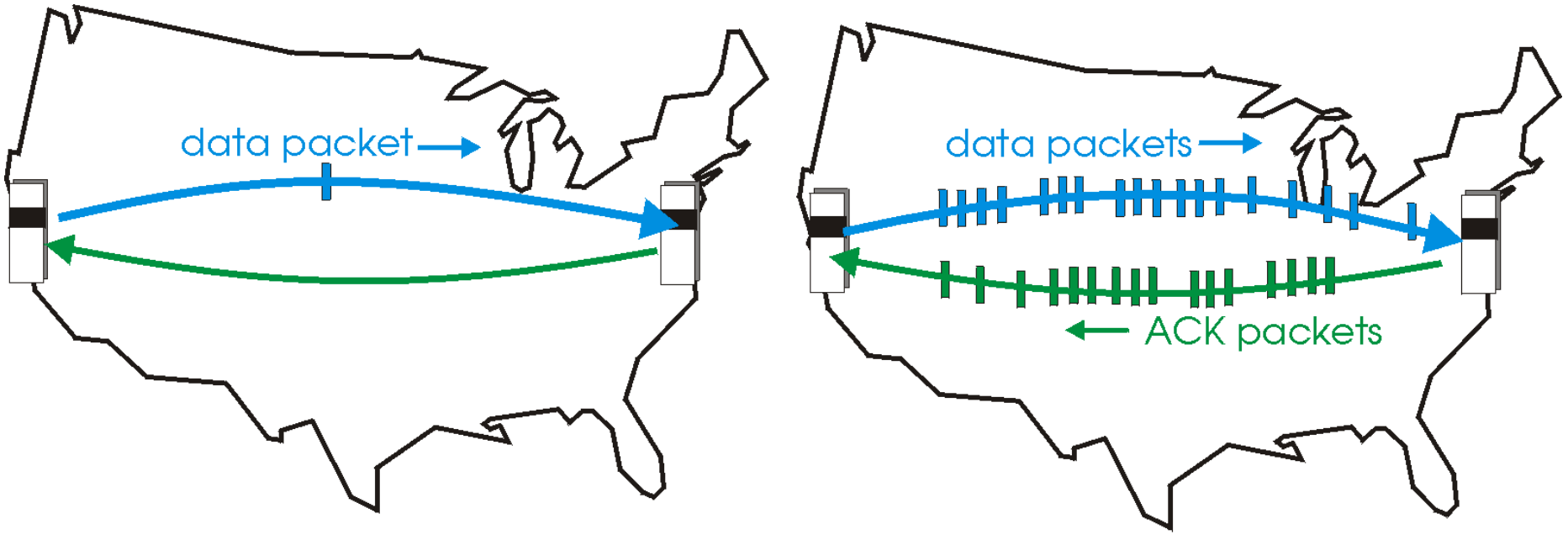
→ ~264 Kbit ogni secondo, su un link da 1 Gbps!!

➔ Evidenza dell'uso limitato delle risorse fisiche di rete

Combinare affidabilità e efficienza

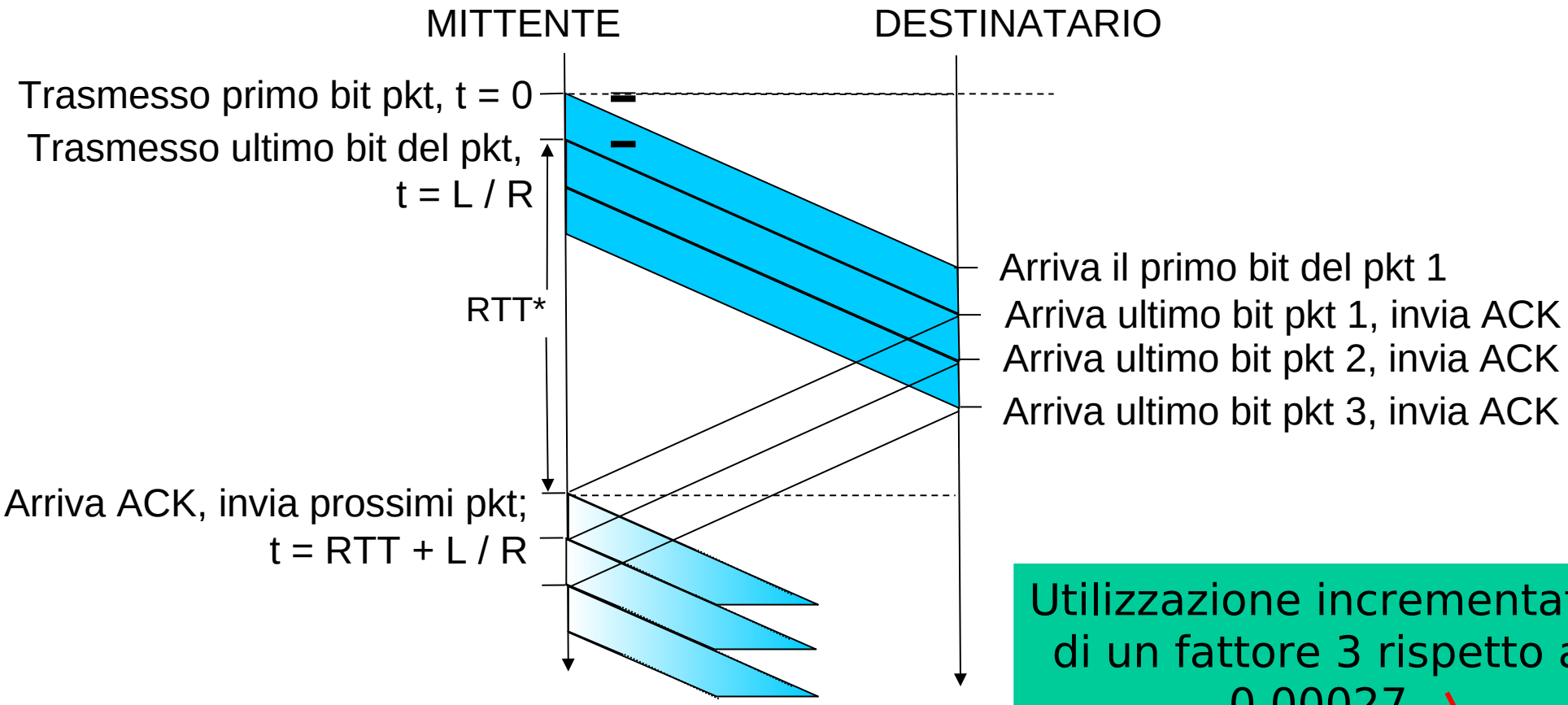
SOLUZIONE

Pipelining → il mittente invia un numero multiplo di segmenti prima di ricevere un ACK



(a) a Dal protocollo stop-and-wait ad un protocollo pipelined

Incremento delle prestazioni (es. per 3)



Utilizzazione incrementata di un fattore 3 rispetto a 0.00027

$$\text{Utilizzazione}_{\text{(mittente)}} = \frac{\text{Tempo trasm.}}{\text{Tempo totale}} = \frac{T}{RTT + T} = \frac{3*L/R}{RTT + L/R} = \frac{0.024}{30 + 0.008} = 0.0008$$

Requisiti per l'implementazione

- **1. L'intervallo dei numeri di sequenza dei pacchetti e degli ack deve essere sufficientemente lungo**
- **2. Necessità di un buffer lato mittente**
  - Per mantenere i pacchetti inviati e di cui non ha ancora ricevuto l'ack
- **3. Necessità di un buffer lato destinatario**
  - Per mantenere le sequenze di pacchetti dove non tutti i pacchetti sono arrivati o sono arrivati correttamente
- **4. Necessità di una "finestra a scorrimento" che denota il numero massimo di pacchetti che il mittente può inviare senza aver ricevuto un ACK dal destinatario**