



# **7 – Il Livello di Trasporto**

## **Parte I**

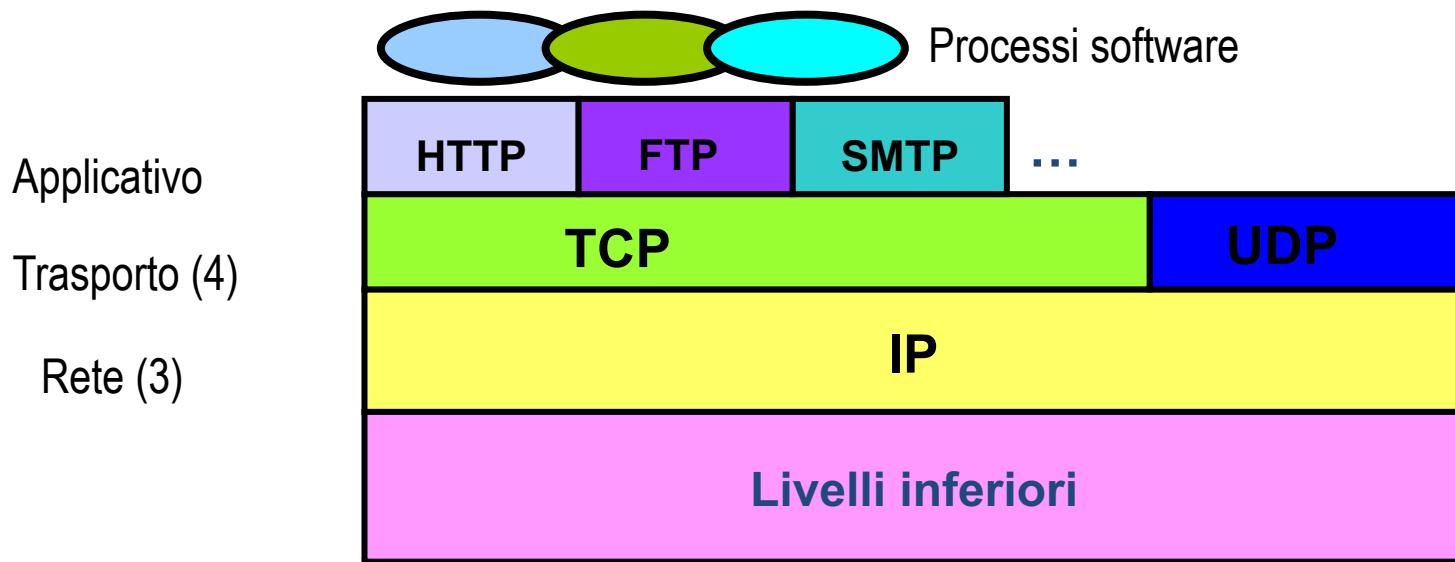
# Livello di Trasporto

- **Introduzione**
- **Protocollo UDP**
- **Trasporto affidabile**
  - Protocolli di ritrasmissione
  - Controllo di flusso a finestra mobile
- **Protocollo TCP**
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



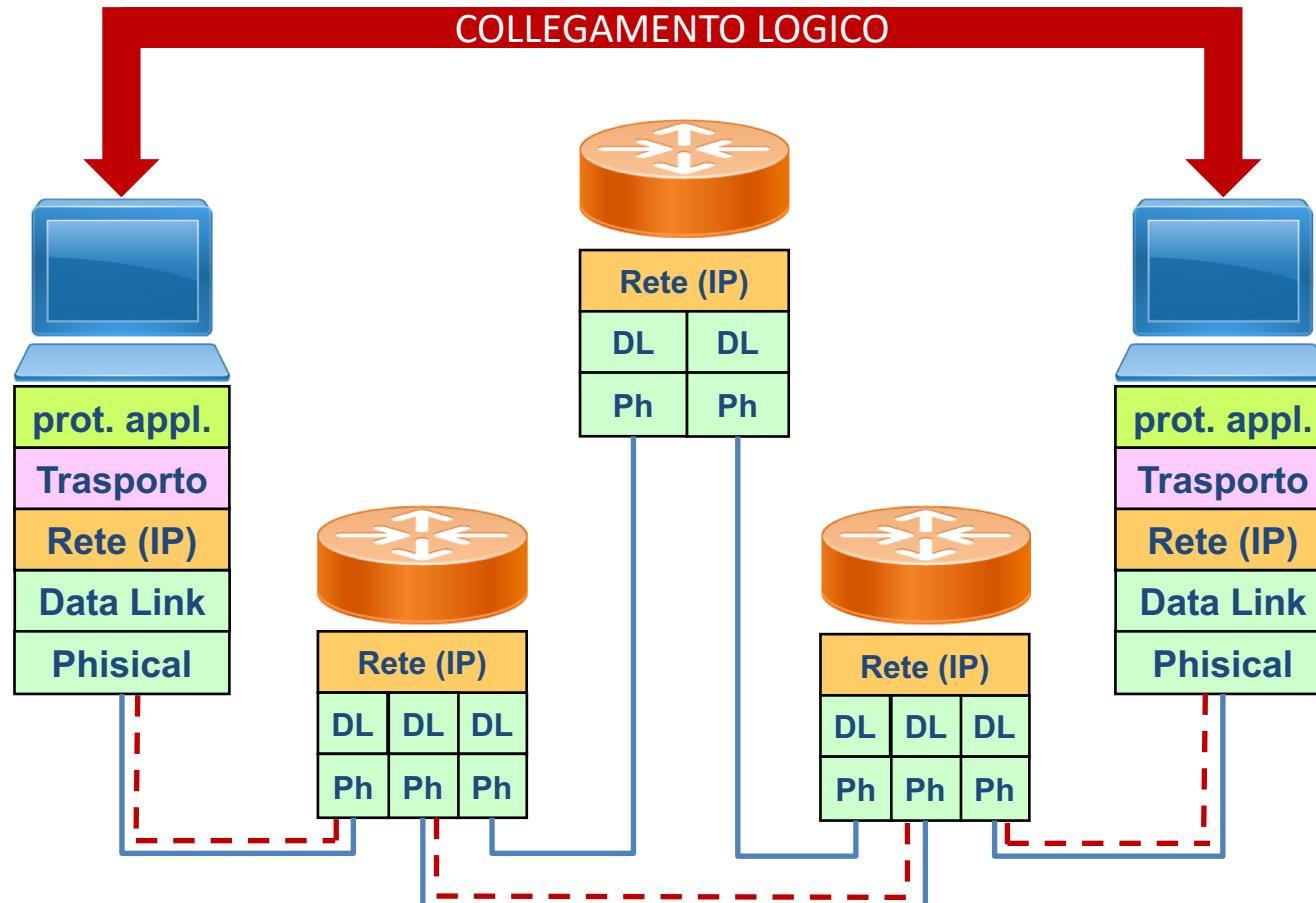
# Servizio di trasporto

- Il livello di trasporto ha il compito di instaurare un collegamento logico tra le applicazioni residenti su host remoti
- Il livello di trasporto rende trasparente il trasporto fisico (attraverso la rete) dei messaggi alle applicazioni



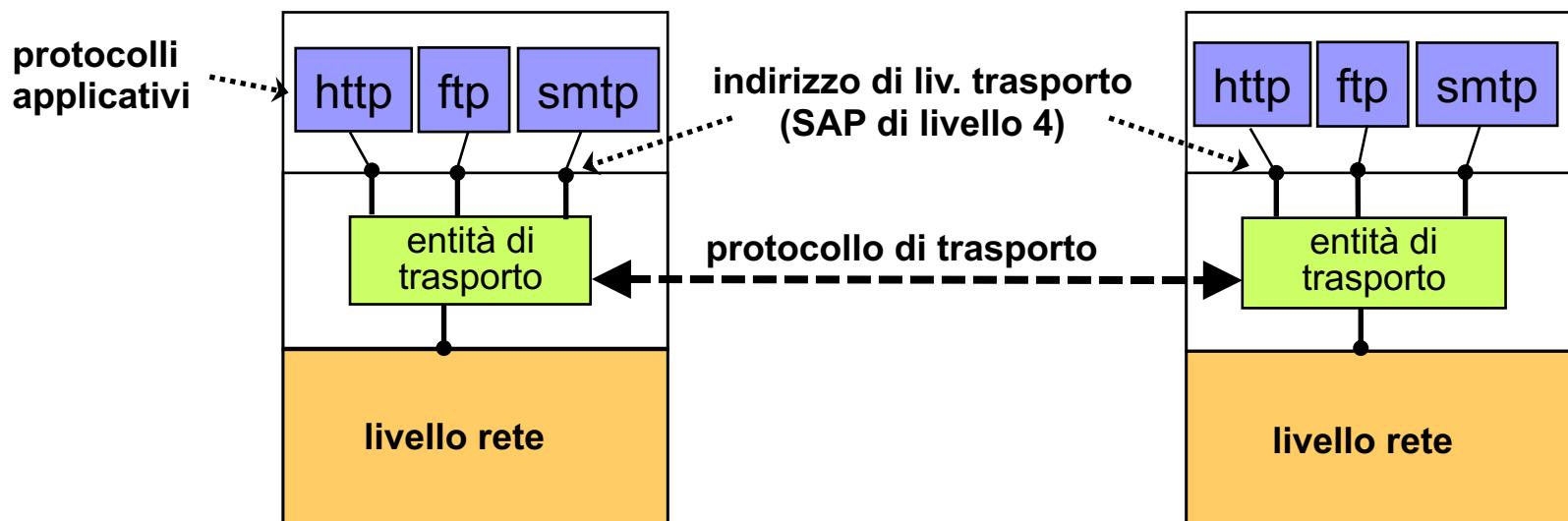
# Servizio di trasporto

- Il livello di trasporto è presente solo negli *end systems (host)*
- Esso consente il collegamento logico tra processi applicativi



# Servizio di trasporto

- Più applicazioni possono essere attive su un *end system*
  - Il livello di trasporto svolge funzioni di *multiplexing/demultiplexing* per applicazioni
  - Ciascun collegamento logico tra applicazioni è indirizzato dal livello di trasporto



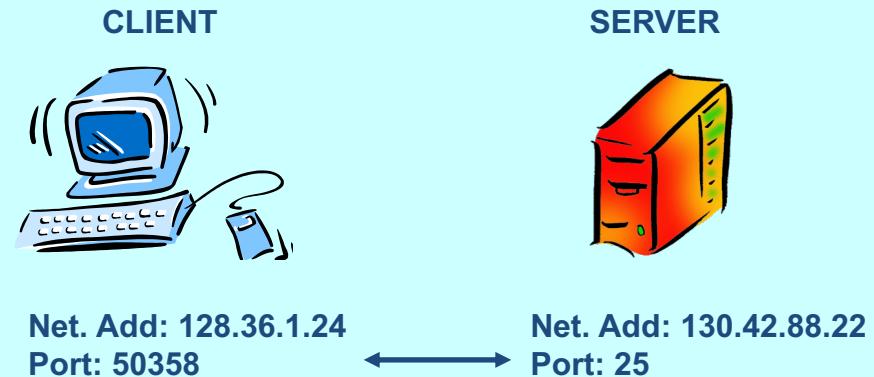
# Indirizzamento: le *porte*

- In Internet le funzioni di *multiplexing/demultiplexing* vengono gestite mediante indirizzi contenuti nelle PDU di livello di trasporto
- Tali indirizzi sono lunghi 16 bit e prendono il nome di *porte*
- I numeri di porta possono assumere valori compresi tra 0 e 65535
- I **numeri noti** sono assegnati ad importanti applicativi dal lato *server* (HTTP, FTP, SMTP, DNS, ecc.)
- I **numeri dinamici** sono assegnati dinamicamente ai processi applicativi lato *client*
- I **numeri registrati** sono assegnati a specifiche applicazioni da chi ne faccia richiesta (tipicamente protocolli proprietari)



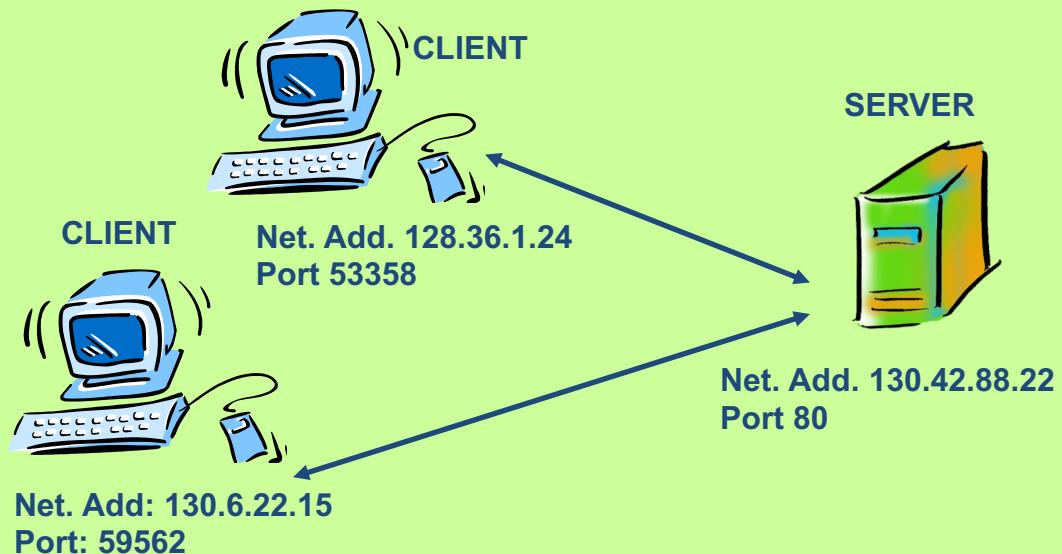
# Socket e multiplazione

Un *client* trasmette segmenti verso la porta di un server SMTP remoto



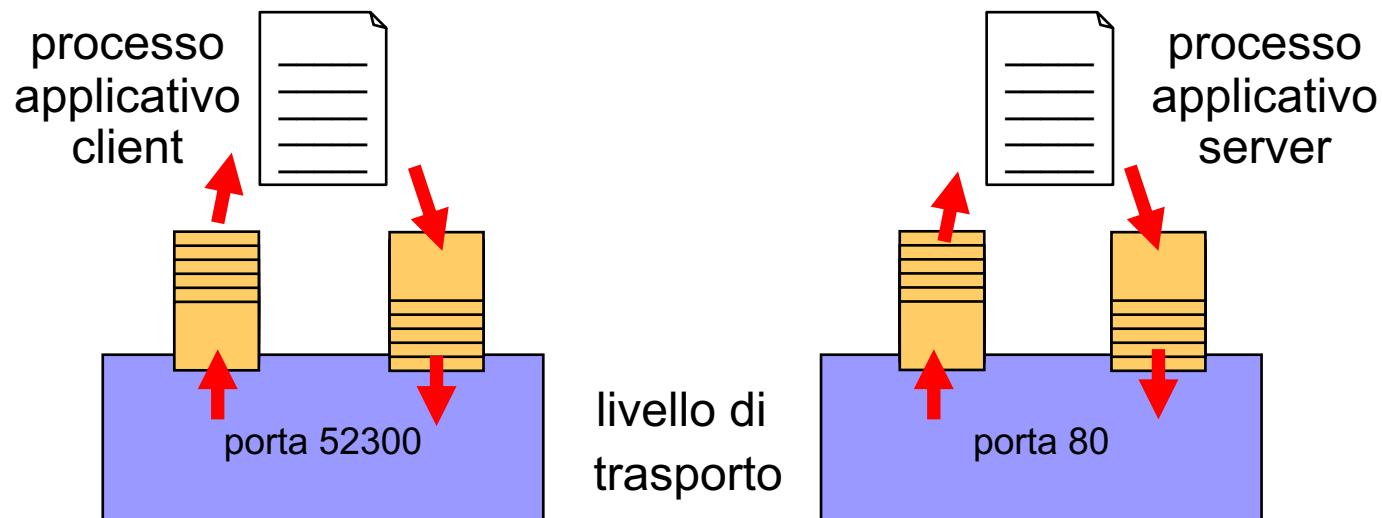
Due client accedono alla stessa porta di un server HTTP.

Non c'è comunque ambiguità, perché la coppia di socket è diversa



# Servizio di Buffering

- I protocolli di trasporto sono implementati nei più diffusi **sistemi operativi**
- Quando un processo viene associato ad una porta (lato *client* o lato *server*) viene associato dal sistema operativo a due code, una d'ingresso e una d'uscita
- Funzionalità di *buffering* dei dati



# Servizio di trasporto

- Il servizio di rete è non affidabile
  - Fa del proprio meglio per consegnare i singoli messaggi indipendentemente a destinazione
- Il servizio di trasporto fornito può essere di vari tipi
  - Trasporto affidabile (garanzia di consegna dei messaggi nel corretto ordine)
  - Trasporto non affidabile (solo funzionalità di multiplexing)
  - Trasporto orientato alla connessione
  - Trasporto senza connessione
- Sono definiti due tipi di trasporto
  - **TCP (Transmission Control Protocol)**, orientato alla connessione e affidabile
  - **UDP (User Datagram Protocol)**, senza connessione e non affidabile



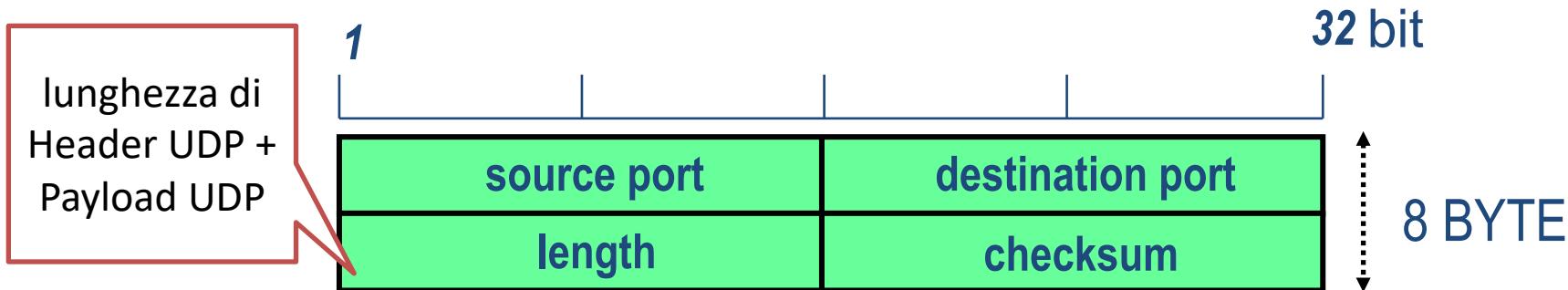
# Livello di Trasporto

- Introduzione
- Protocollo UDP
- Trasporto affidabile
  - Protocolli di ritrasmissione
  - Meccanismo a finestra mobile
- Protocollo TCP
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



# User Datagram Protocol (UDP) – RFC 768

- E' il modo più semplice di usare le funzionalità di IP
- Non aggiunge nulla a IP se non:
  - Indirizzamento delle applicazioni (mux/demux)
  - Basso controllo d'errore sull'header (senza correzione)
- ... e quindi
  - E' un protocollo datagram
  - Non garantisce la consegna
  - Non esercita nessun controllo (né di flusso, né di errore)



# Esempio di pacchetto UDP

**Numero porta sorgente (o destinatario): 2 Byte (16 bit)**

esempio: 4336 in sistema decimale

*0001 0000 1111 0000* in sistema binario

*1 0 f 0* in sistema esadecimale

esempio: 161 in sistema decimale

*0000 0000 1010 0001* in sistema binario

*0 0 a 1* in sistema esadecimale

<b>0<sub>hex</sub></b> = 0 <sub>dec</sub>	0	0	0	0
<b>1<sub>hex</sub></b> = 1 <sub>dec</sub>	0	0	0	1
<b>2<sub>hex</sub></b> = 2 <sub>dec</sub>	0	0	1	0
<b>3<sub>hex</sub></b> = 3 <sub>dec</sub>	0	0	1	1
<b>4<sub>hex</sub></b> = 4 <sub>dec</sub>	0	1	0	0
<b>5<sub>hex</sub></b> = 5 <sub>dec</sub>	0	1	0	1
<b>6<sub>hex</sub></b> = 6 <sub>dec</sub>	0	1	1	0
<b>7<sub>hex</sub></b> = 7 <sub>dec</sub>	0	1	1	1
<b>8<sub>hex</sub></b> = 8 <sub>dec</sub>	1	0	0	0
<b>9<sub>hex</sub></b> = 9 <sub>dec</sub>	1	0	0	1
<b>A<sub>hex</sub></b> = 10 <sub>dec</sub>	1	0	1	0
<b>B<sub>hex</sub></b> = 11 <sub>dec</sub>	1	0	1	1
<b>C<sub>hex</sub></b> = 12 <sub>dec</sub>	1	1	0	0
<b>D<sub>hex</sub></b> = 13 <sub>dec</sub>	1	1	0	1
<b>E<sub>hex</sub></b> = 14 <sub>dec</sub>	1	1	1	0
<b>F<sub>hex</sub></b> = 15 <sub>dec</sub>	1	1	1	1

**Numero massimo di porta:** si hanno al massimo 2 byte a disposizione

$$2^{16} - 1 = \underline{\underline{65535}} \quad [\text{in binario tutti e 16 bit a "1"}]$$

**Lunghezza massima possibile per il payload UDP:**

si hanno al massimo 2 byte a disposizione nel campo “length”

$$2^{16} - 1 = \underline{\underline{65535}} \text{ byte di lunghezza TOTALE del pacchetto UDP}$$

**Si devono togliere gli 8 byte dell'header UDP:**

$$\underline{\underline{65527}} \text{ byte disponibile per il payload UDP}$$

Se il payload è più lungo lo si trasmette in più segmenti.



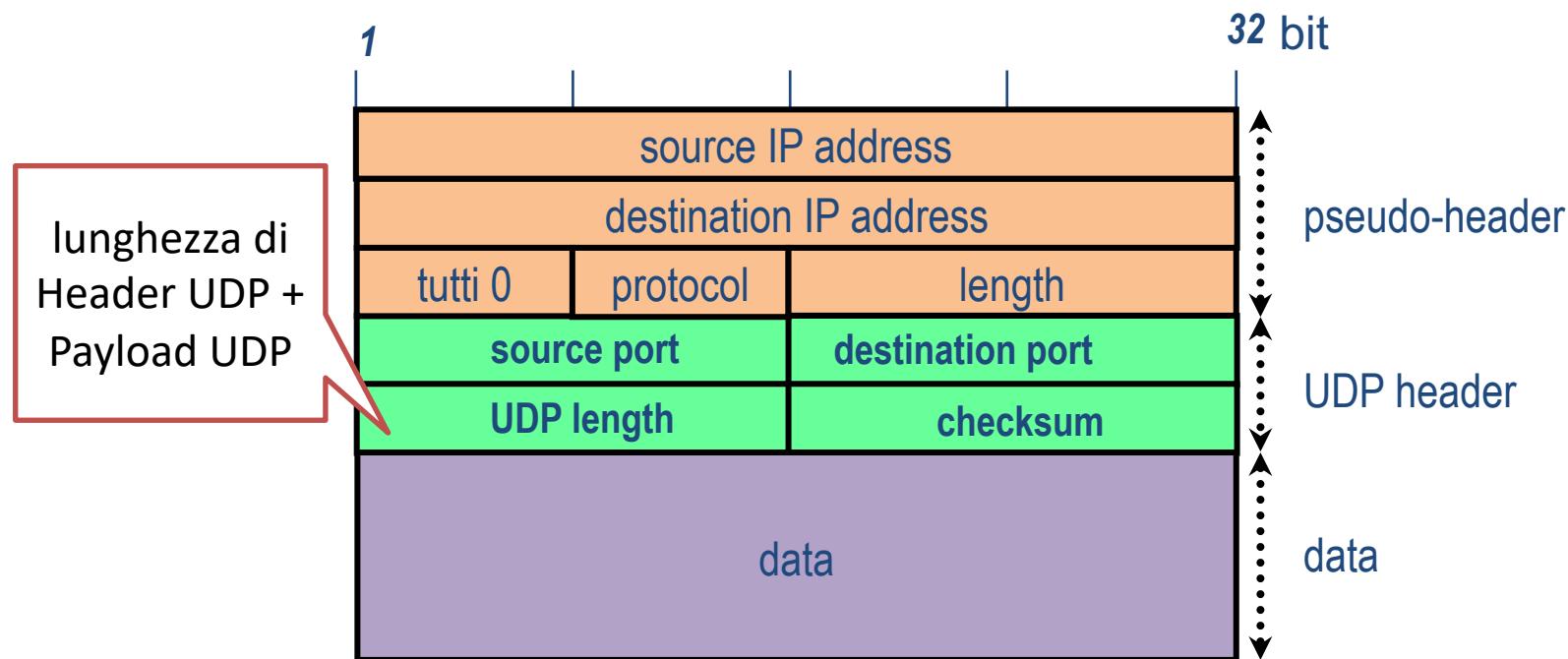
# Perchè usare UDP e non TCP?

- **Minore latenza**
  - Non occorre stabilire una connessione
- **Maggiore semplicità**
  - Non occorre tenere traccia dello stato della connessione
  - Poche regole da implementare
- **Minore overhead**
  - Header UDP è minore dell'header TCP



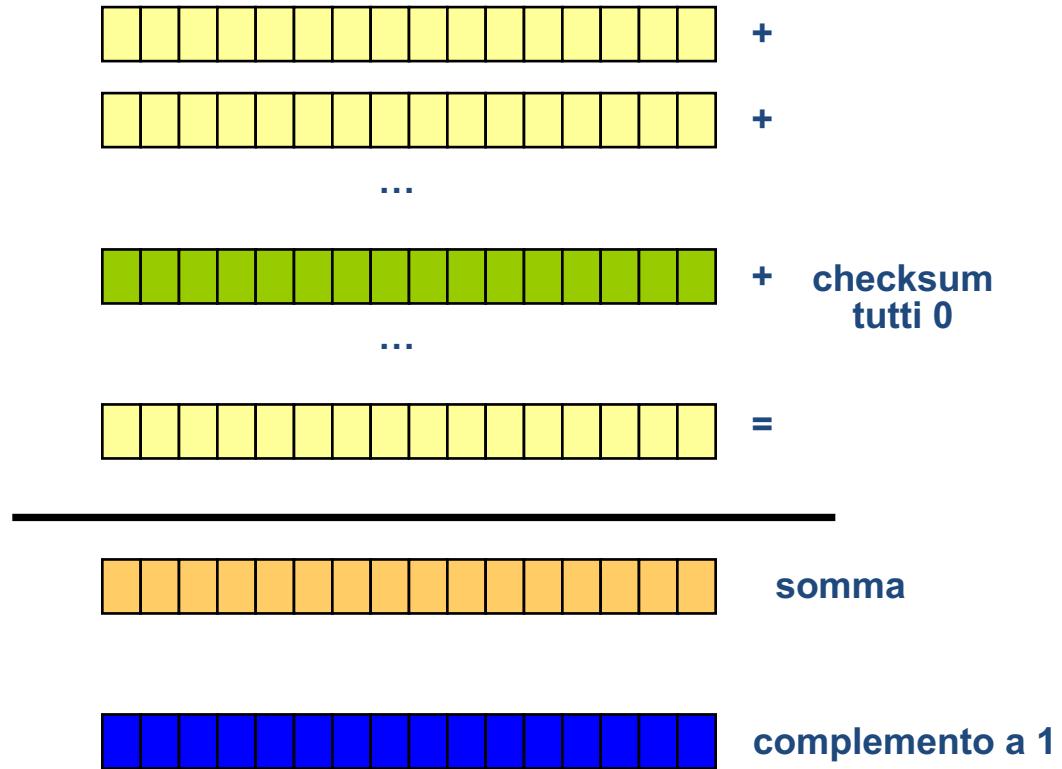
# Il campo *Checksum*: controllo di integrità

- Informazione ridondante inserita nell'header del segmento UDP per controllo d'errore
- Il campo di *checksum* (16 bit) è calcolato dal trasmettitore ed inserito nell'header
- Il ricevitore ripete lo stesso calcolo sul segmento ricevuto (comprensivo di *checksum*)
- Se il risultato è soddisfacente accetta il segmento, altrimenti lo scarta
- Viene calcolato considerando l'**header** UDP, uno **pseudo-header** IP ed i **dati**



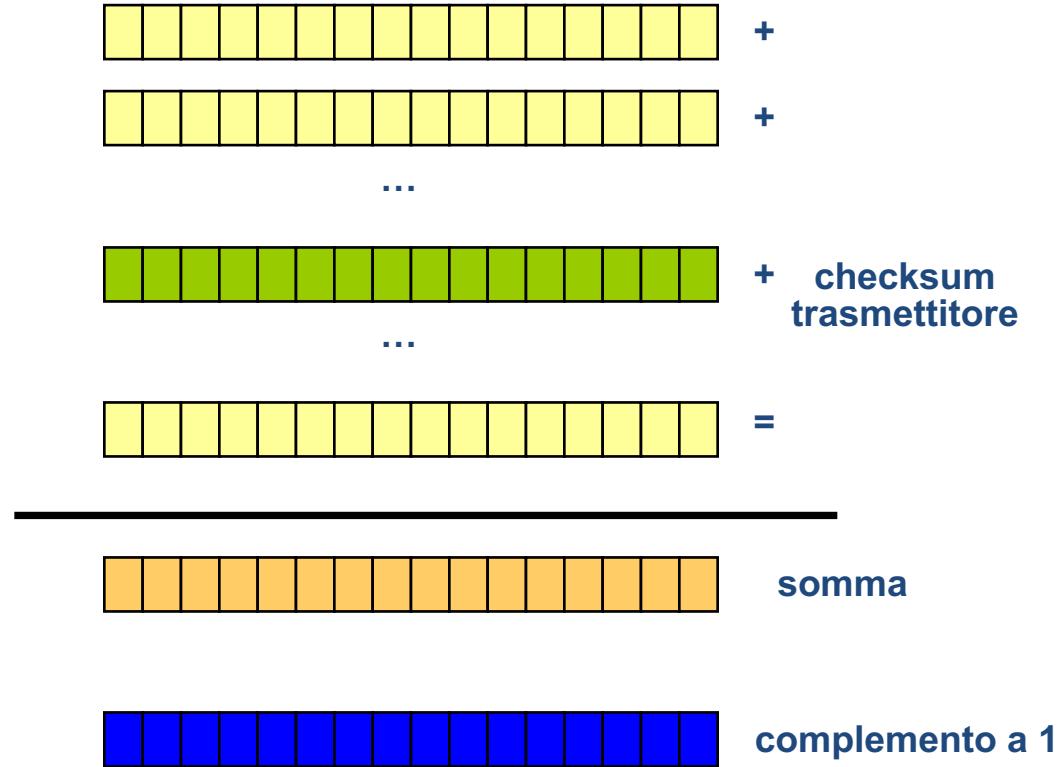
# Calcolo del Checksum *lato trasmettitore*

- L'insieme di bit è diviso in blocchi da 16 bit
- Il campo *Checksum* è inizializzato a 0
- Tutti i blocchi vengono sommati modulo 2
- Il risultato è complementato ed inserito nel campo di *checksum* del segmento inviato



# Calcolo del Checksum *lato ricevitore*

- L'insieme di bit è diviso ancora in blocchi da 16 bit
- Tutti i blocchi vengono sommati modulo 2
- Il risultato è complementato
  - Se sono tutti 0 il pacchetto è accettato
  - Altrimenti è scartato



# Livello di Trasporto

- Introduzione
- Protocollo UDP
- **Trasporto affidabile**
  - Protocolli di ritrasmissione
  - Controllo di flusso a finestra mobile
- **Protocollo TCP**
  - Generalità
  - Formato
  - Controllo d'errore
  - Controllo di congestione



# Collegamento ideale

- Collegamento ideale
  - Tutto ciò che viene trasmesso arriva nello stesso ordine e viene correttamente interpretato a destinazione
- Es., come essere sicuri che una sequenza di ordini impartiti sia stata compresa con certezza?
- E' possibile oppure è uno sforzo velleitario?
- E' necessario esserne certi?



# Recupero d'errore

- Ingredienti di un **Protocollo di Ritrasmissione**
  - Ciascuna trama ricevuta correttamente viene riscontrata positivamente con un messaggio di (Acknowledgment o ACK)
  - A volte l'errore può essere segnalato da un messaggio detto di NACK (Not ACK)
  - La mancanza di ACK o la presenza di NACK segnala la necessità di ritrasmettere
  - La procedura si ripete finché la trama viene ricevuta corretta
- Necessita di canale di ritorno e di messaggi di servizio (ACK, NACK)
- Nota: anche i messaggi di servizio possono essere corrotti da errori!



# Controllo d'integrità e recupero d'errore

- Queste procedure possono essere attivate a qualunque livello
- Storicamente sono state sempre presenti a livello di linea sui collegamenti fisici a causa delle cattive linee fisiche del passato
- Presente a livello di trasporto per recupero *end-to-end*
  - Proteggere i collegamenti fisici non basta, i pacchetti possono andare persi nei buffer dei router
- Nei sistemi moderni il recupero d'errore a livello di linea può essere assente



# Protocolli di ritrasmissione

- Obiettivo:
  - integrità del messaggio/pacchetto
  - ordine della sequenza di pacchetti
  - no duplicazione
- Usando i messaggi:
  - ACK: riscontro positivo
  - NACK: riscontro negativo
- Tre tipologie:
  - *Stop & Wait*
  - *Go-Back-N*
  - *Selective Repeat*



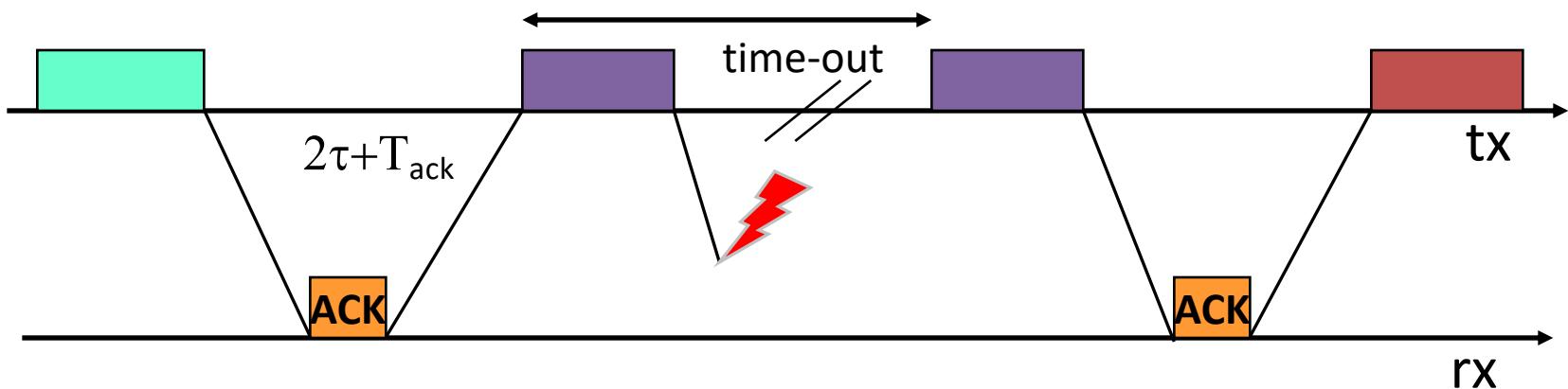
# Protocollo Stop and Wait

- Utilizza il solo ACK
- e un contatore di *time out*
- Ogni messaggio ricevuto correttamente è riscontrato dal ricevitore (ACK)



# Protocollo Stop and Wait

- Il trasmettitore trasmette un pacchetto e inizializza un contatore (*time-out*)
- Se il primo evento successivo è
  - l'ACK, trasmette il pacchetto successivo
  - il time-out, ritrasmette il pacchetto corrente

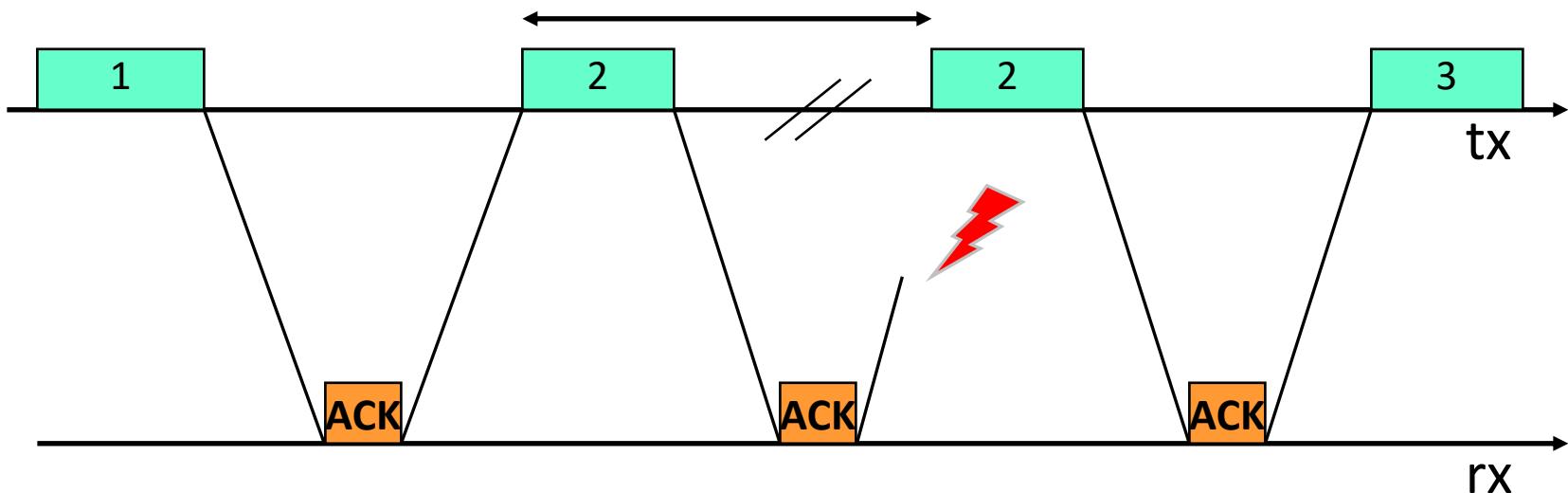


Funzionamento corretto se **time-out  $\geq 2\tau + T_{ack}$**



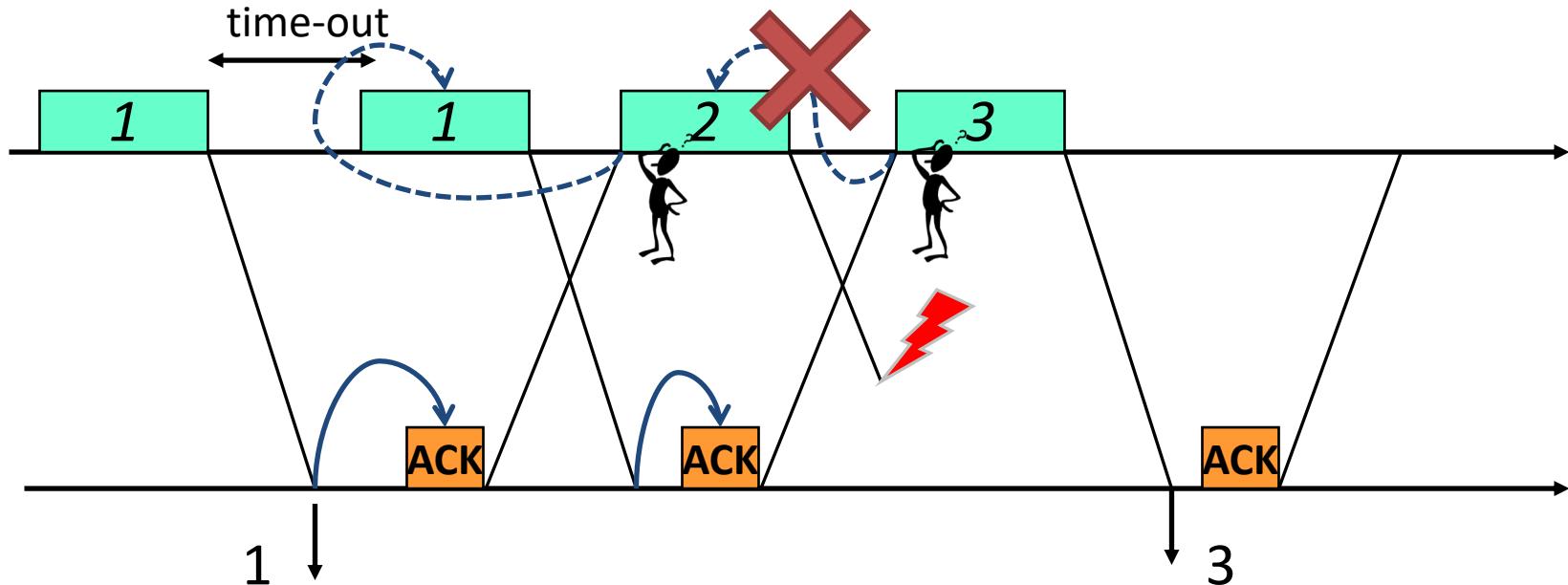
# Protocollo Stop and Wait

- Problema: se si perde l'ACK il pacchetto viene ritrasmesso (e ricevuto) di nuovo (duplicazione di pacchetti)
- Rimedio: **numerazione dei pacchetti (SN)**
- Il ricevitore riconosce che il nuovo pacchetto è un duplicato perché ha lo stesso numero dell'ultimo ricevuto



# Protocollo Stop and Wait

- Problema: errata interpretazione di un ACK. Il ricevitore crede un pacchetto ricevuto (il 2) mentre non lo è
- Rimedio: **numerazione anche degli ACK (RN)**



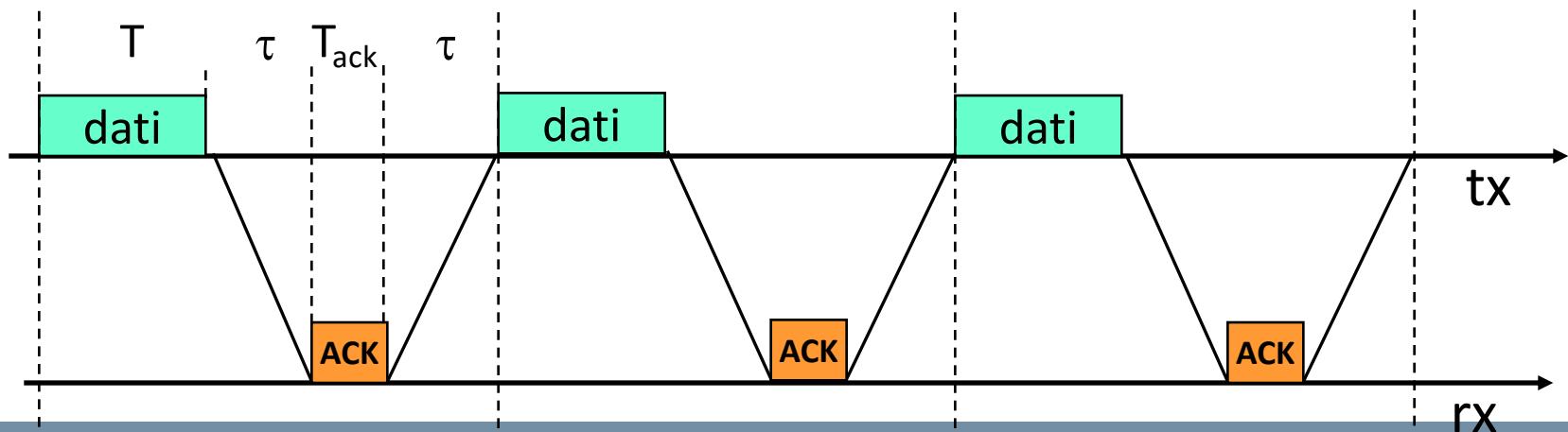
# Protocollo Stop and Wait

- Efficienza del protocollo:
  - frazione di tempo in cui il canale è usato per trasmettere informazione utile in assenza di errori

$$\eta = \frac{T}{T + T_{ack} + 2\tau}$$

$T$  = tempo di trasmissione

$\tau$  = tempo di propagazione



# Protocollo Stop and Wait

- Efficienza bassa se  $T \ll \tau$
- Protocollo non adatto a situazioni con elevato ritardo di propagazione e/o elevato ritmo di trasmissione
- Utilizzato spesso in modalità *half-duplex*

$$\eta = \frac{T}{T + T_{ack} + 2\tau} = \frac{1}{1 + \frac{T_{ack}}{T} + \frac{2\tau}{T}}$$

