



POLITECNICO
MILANO 1863

Fondamenti di TELECOMUNICAZIONI

Prof. Marco Mezzavilla

Lezione 10 – Livello di Trasporto 1

INDICE

10. LIVELLO DI TRASPORTO 1

1. Servizio di trasporto

2. Protocollo UDP

3. Trasporto affidabile I

1. Stop & Wait

Parte I (oggi)

4. Trasporto affidabile II

2. Go-Back-N
3. Selective Repeat
4. Controllo di flusso a finestra mobile

Parte II (domani)

5. Protocollo TCP

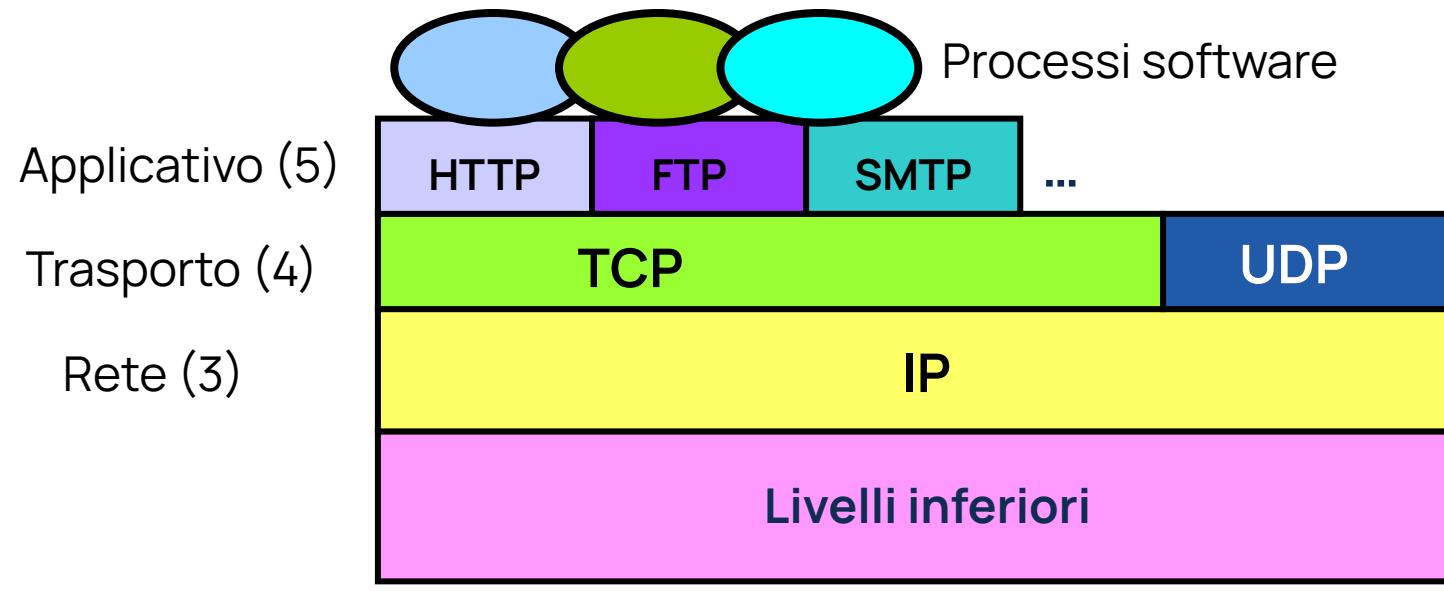
Parte III (dopodomani)

SERVIZIO DI TRASPORTO

01

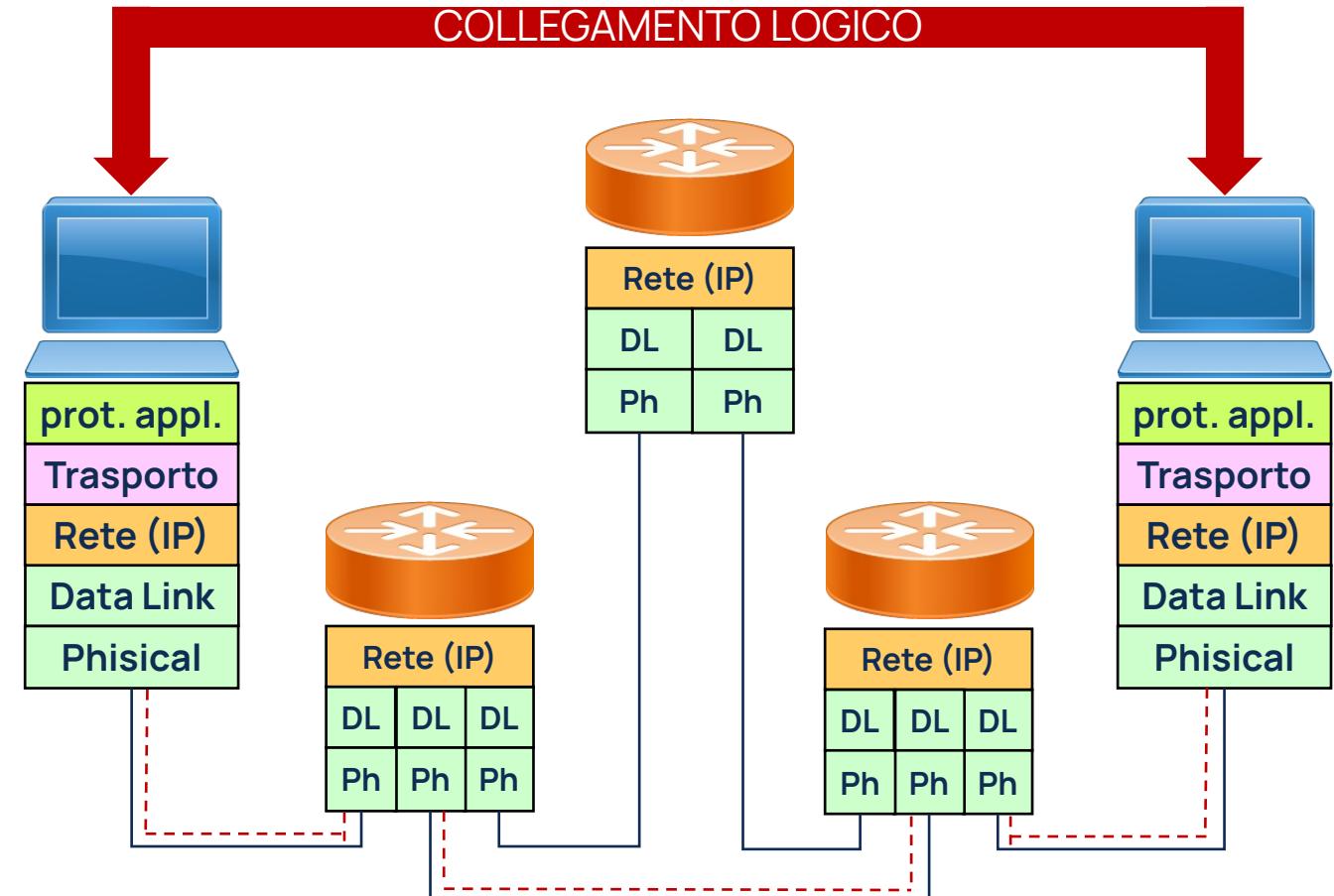
Servizio di trasporto

- Il livello di trasporto ha il compito di instaurare un **collegamento logico** tra le applicazioni residenti su host remoti
- Un collegamento logico (dunque non fisico) è una **connessione virtuale** che consente a due applicazioni su dispositivi diversi (host remoti) di comunicare in modo strutturato, *indipendentemente dal collegamento fisico che li separa*
- Il livello di trasporto rende **trasparente** il trasporto fisico (attraverso la rete) dei messaggi alle applicazioni



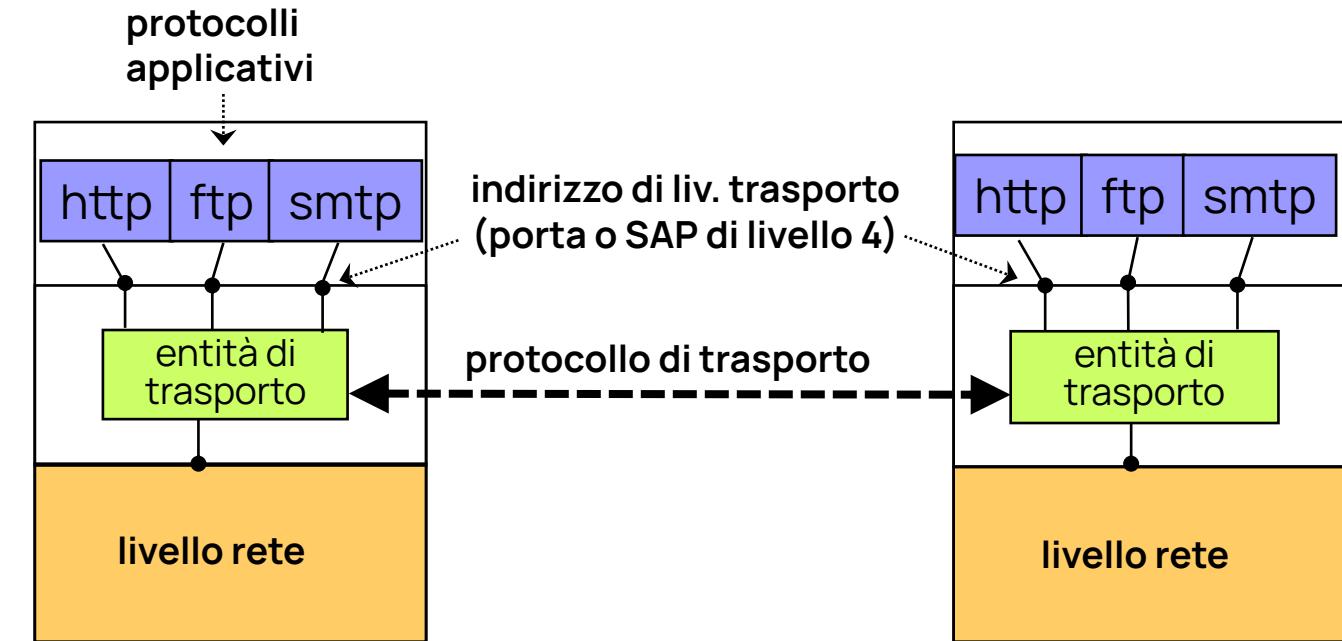
Servizio di trasporto

- Il livello di trasporto è presente solo negli *end systems (hosts)*
- 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)

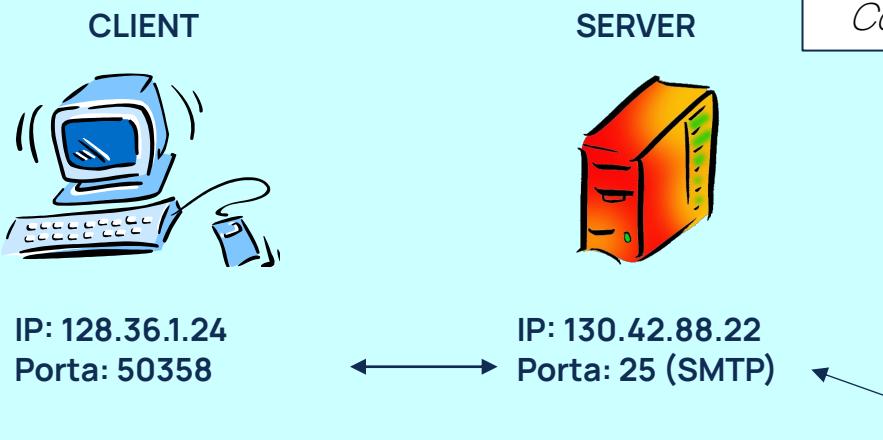


| Protocollo | Porta | Descrizione |
|------------|-------|---------------------------------------|
| HTTP | 80 | HTTP non criptato |
| HTTPS | 443 | HTTP sicuro (criptato) |
| FTP | 21 | Controllo della connessione FTP |
| FTP | 20 | Trasferimento dati in modalità attiva |
| SMTP | 25 | Invio di email tra server di posta |
| SMTP | 587 | Invio di email con autenticazione |
| SMTP | 465 | SMTP sicuro (meno comune) |
| DNS | 53 | Risoluzione dei nomi (UDP e TCP) |

Socket e multiplazione

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

stesso client, due applicazioni diverse (web browsing + posta)



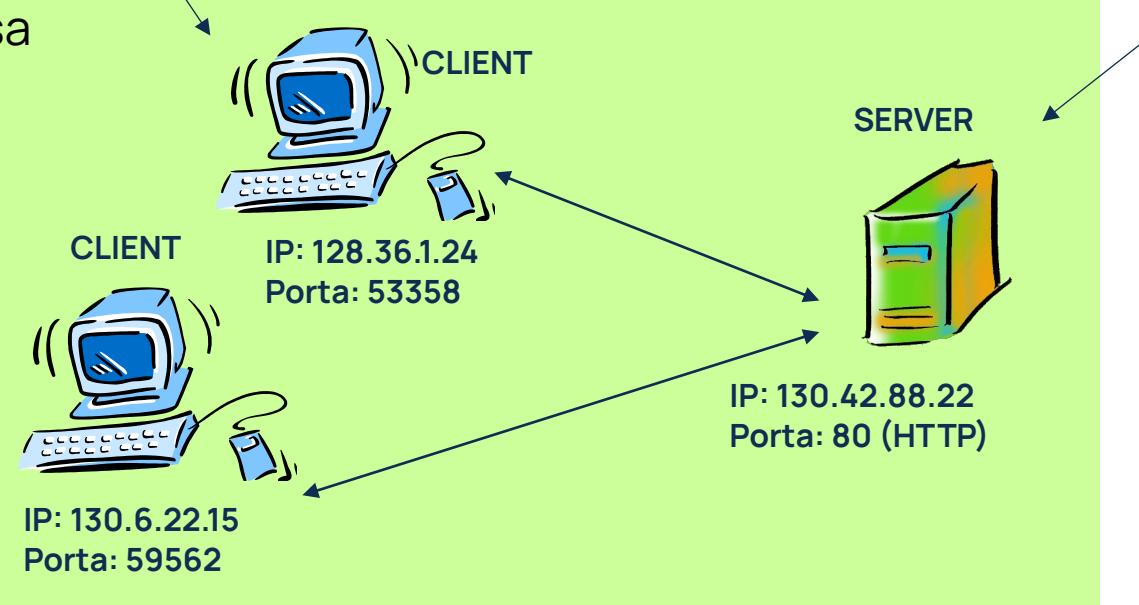
L'assegnazione dinamica delle porte **lato client** viene gestita dal sistema operativo!

Come tutto il livello di trasporto del resto..

Due client accedono alla stessa porta di un server HTTP.

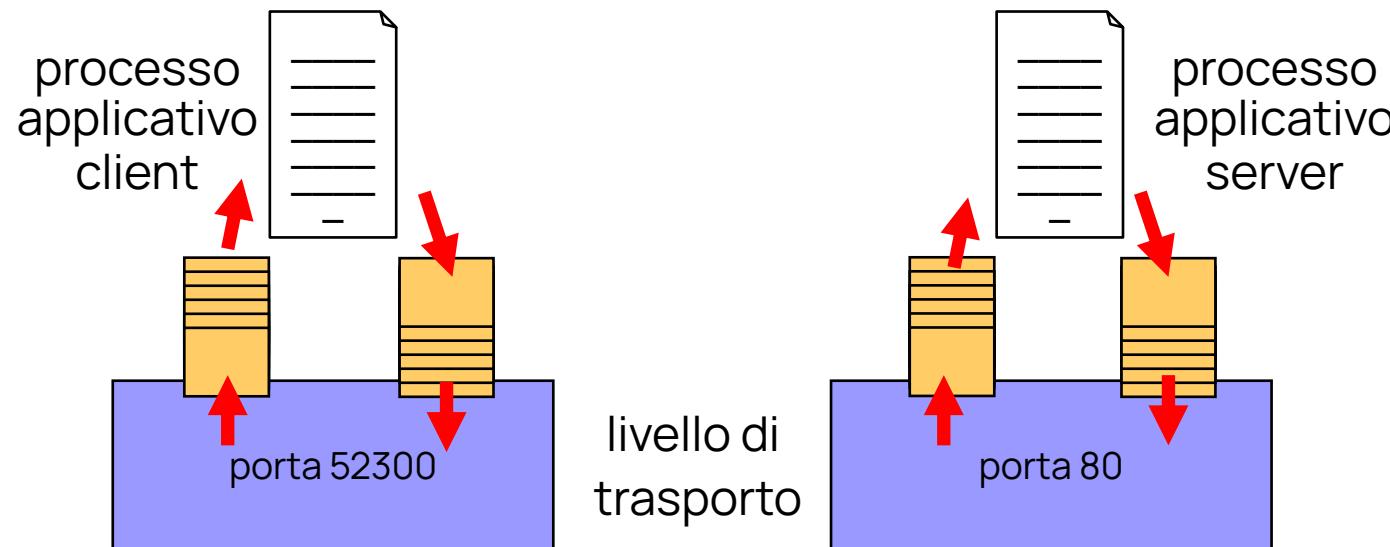
Non c'è comunque ambiguità, perché la coppia di socket è diversa

stesso server, due applicazioni diverse (web browsing + posta)



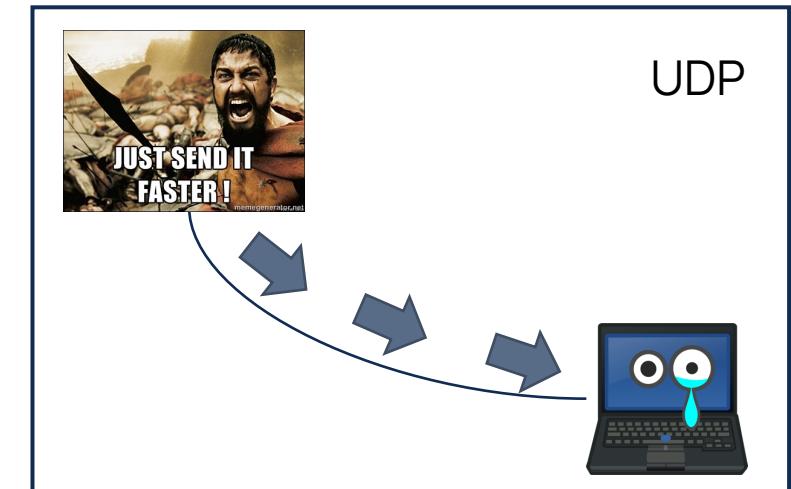
Servizio di buffering

- 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 (memorizzazione delle code d'ingresso e d'uscita)



Servizio di trasporto

- ❑ La rete internet non è affidabile, ci possono essere **errori**
 - ❑ Fa del proprio meglio per consegnare i singoli messaggi indipendentemente dalla 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

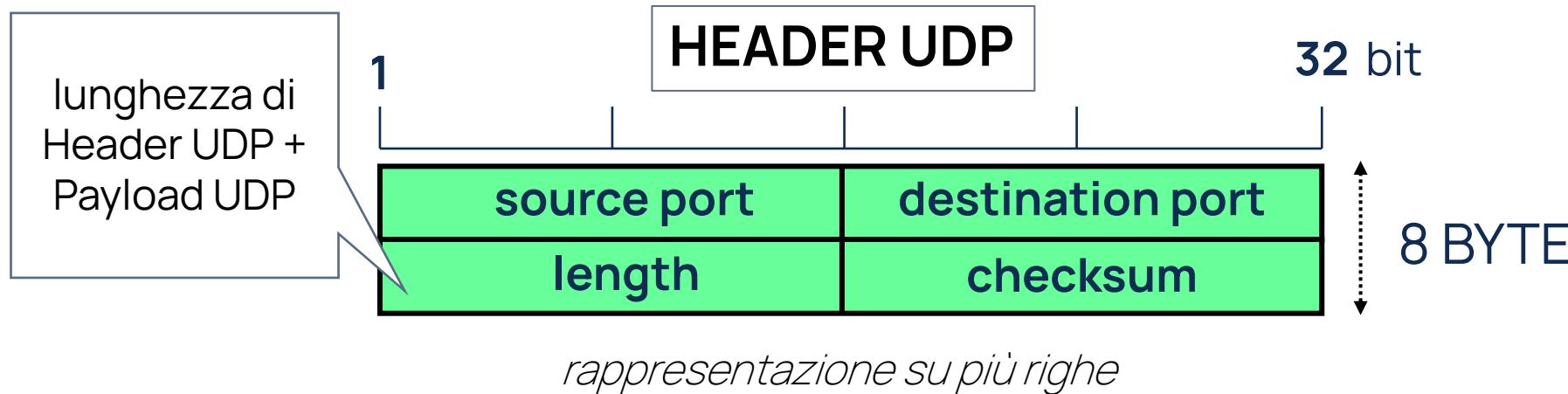


PROTOCOLLO UDP

02

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)
 - ❑ Blando controllo d'errore sull'header (senza correzione)
- ❑ Perciò:
 - ❑ E' un protocollo datagram (a differenza dei "segmenti" di TCP, i datagrammi sono autonomi e non richiedono una connessione stabile tra mittente e destinatario.)
 - ❑ Non garantisce la consegna
 - ❑ Non esercita nessun controllo (né di flusso, né di errore)



Esempio di pacchetto UDP

- Numero porta sorgente (o destinatario): 2 B (16 b)

- 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

- Numero massimo di porta: si hanno al massimo 2 byte a disposizione
 $2^{16} - 1 = \underline{65535}$ [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"
 $2^{16} - 1 = 65535$ B di lunghezza TOTALE del pacchetto UDP

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

65527 B disponibili per il payload UDP (524216 b, 524.216 Kb)

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

| | |
|--|---------|
| 0_{hex} = 0 _{dec} | 0 0 0 0 |
| 1_{hex} = 1 _{dec} | 0 0 0 1 |
| 2_{hex} = 2 _{dec} | 0 0 1 0 |
| 3_{hex} = 3 _{dec} | 0 0 1 1 |
| 4_{hex} = 4 _{dec} | 0 1 0 0 |
| 5_{hex} = 5 _{dec} | 0 1 0 1 |
| 6_{hex} = 6 _{dec} | 0 1 1 0 |
| 7_{hex} = 7 _{dec} | 0 1 1 1 |
| 8_{hex} = 8 _{dec} | 1 0 0 0 |
| 9_{hex} = 9 _{dec} | 1 0 0 1 |
| A_{hex} = 10 _{dec} | 1 0 1 0 |
| B_{hex} = 11 _{dec} | 1 0 1 1 |
| C_{hex} = 12 _{dec} | 1 1 0 0 |
| D_{hex} = 13 _{dec} | 1 1 0 1 |
| E_{hex} = 14 _{dec} | 1 1 1 0 |
| F_{hex} = 15 _{dec} | 1 1 1 1 |

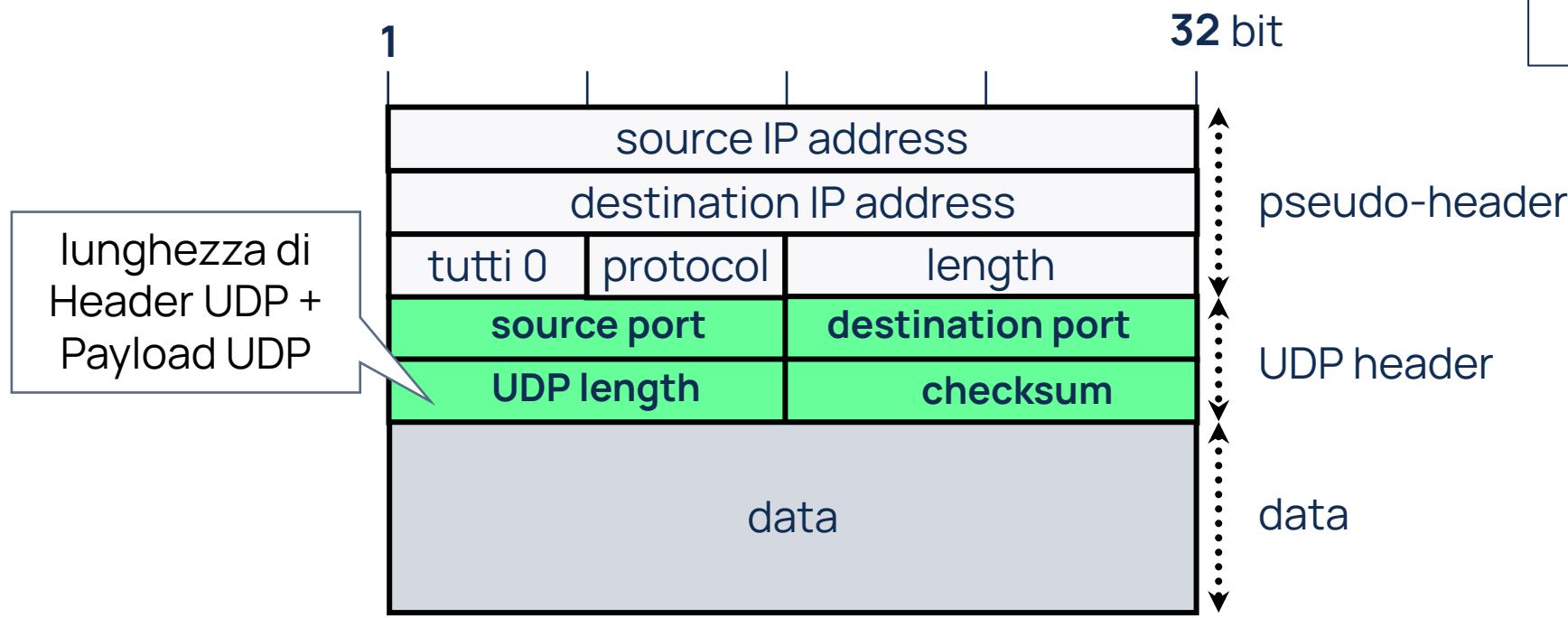
Perché usare UDP e non TCP?

- ❑ 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 (come vedremo più avanti)

Il campo Checksum: controllo di integrità

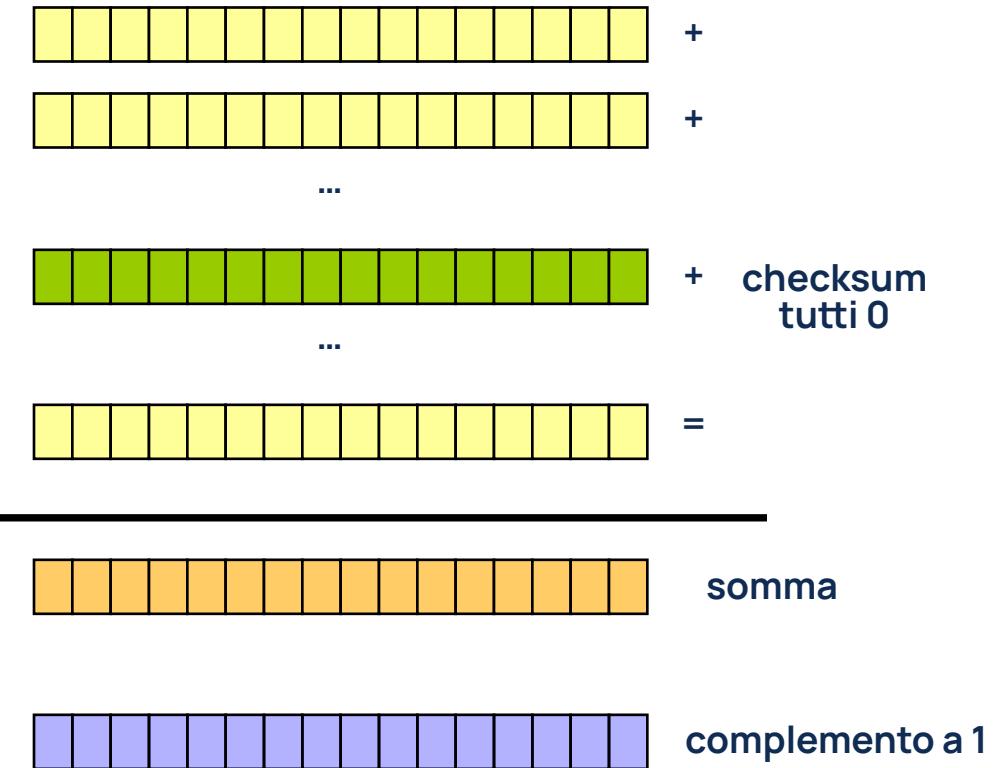
- ❑ Informazione ridondante inserita nell'header del segmento UDP per controllo (blando) d'errore
- ❑ Il campo di *checksum* (16 bit) è calcolato dal trasmettitore ed inserito nell'header
- ❑ Il ricevitore ripete lo stesso calcolo sul segmento ricevuto (comprendivo di *checksum*)
- ❑ Se il risultato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

viola il principio della gerarchia dei protocolli, in quanto il protocollo di trasporto dovrebbe essere all'oscuro delle operazioni svolte dal sottostante protocollo di rete..



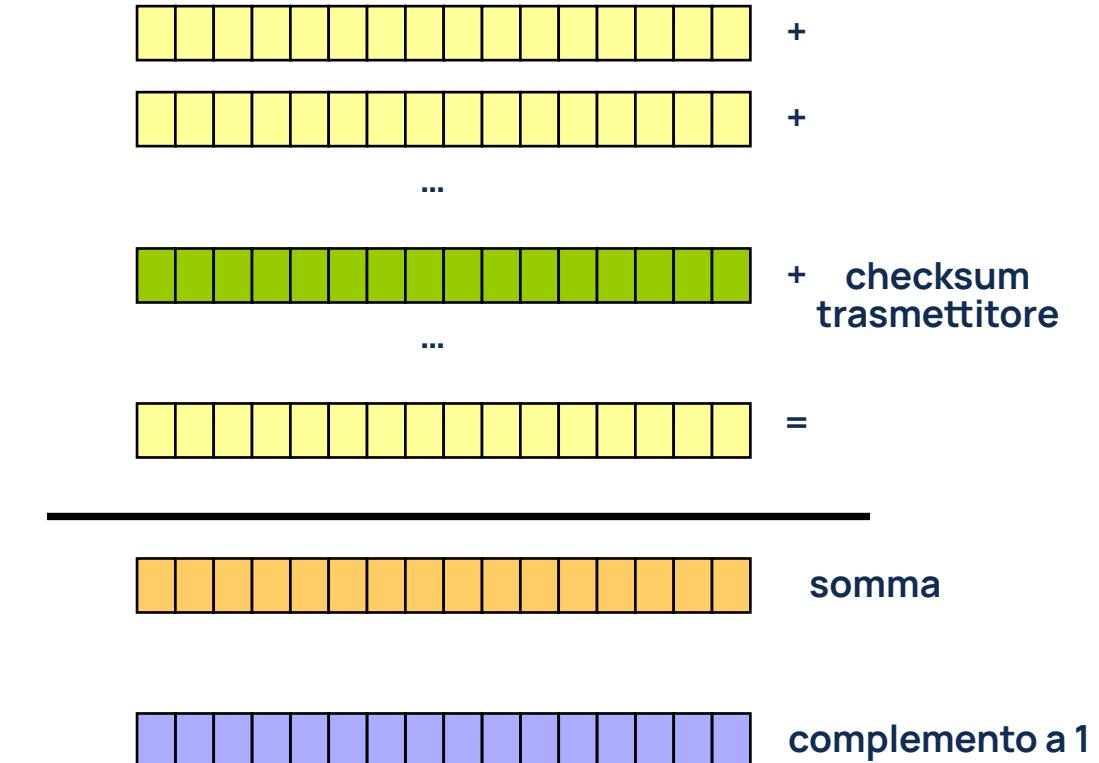
Calcolo del Checksum lato trasmettitore

- L'insieme di bit è diviso in blocchi da 16 bit
- Il campo *Checksum* è inizializzato a 0
- Tutti i blocchi da 16 bit vengono sommati tra loro
 - $0+0=0$
 - $0+1=1$
 - $1+0=1$
 - $1+1=0$ (con riporto 1)
- Il risultato è complementato (gli 0 diventano 1 e viceversa) ed inserito nel campo di *checksum* del segmento inviato



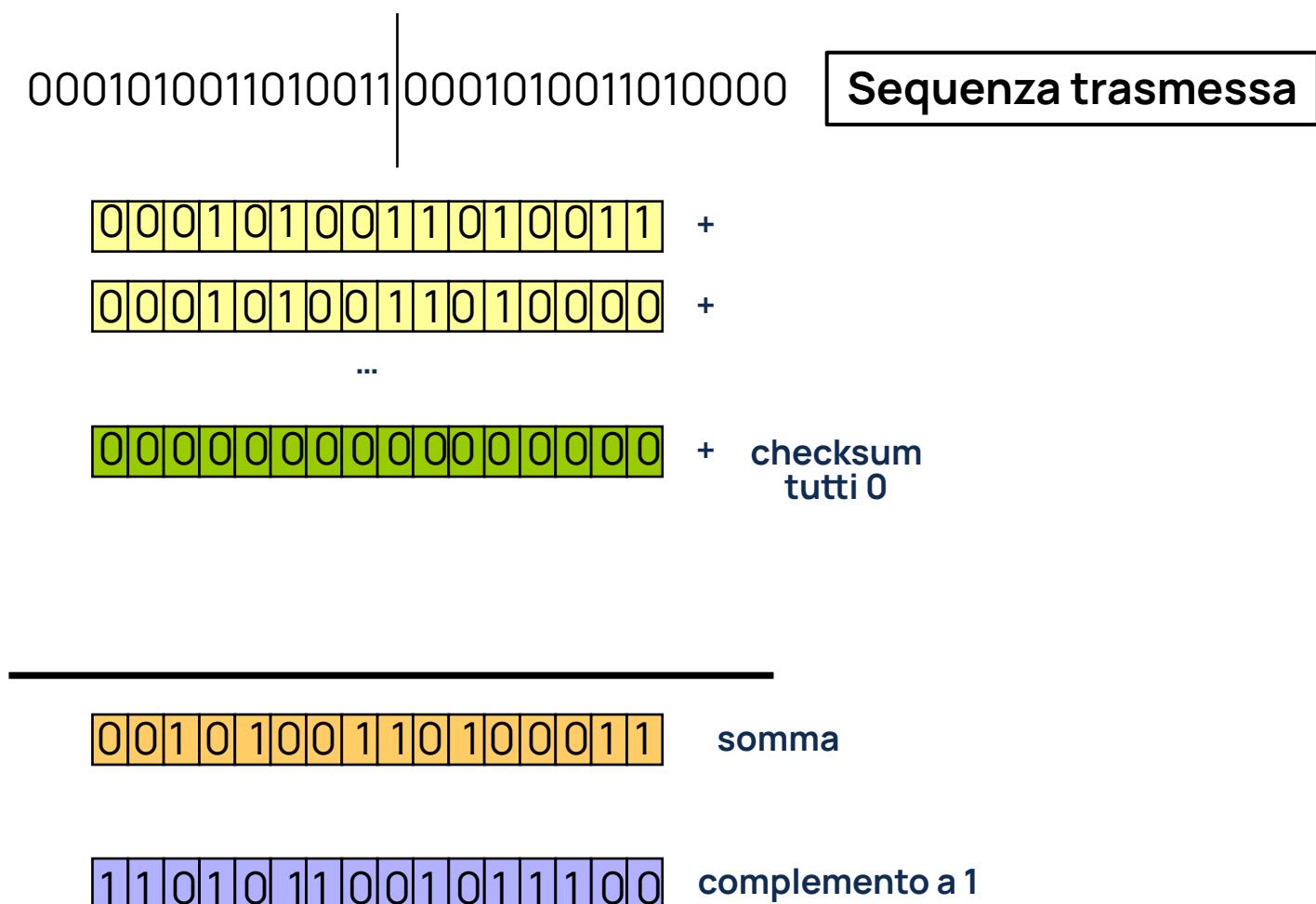
Calcolo del Checksum lato ricevitore

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



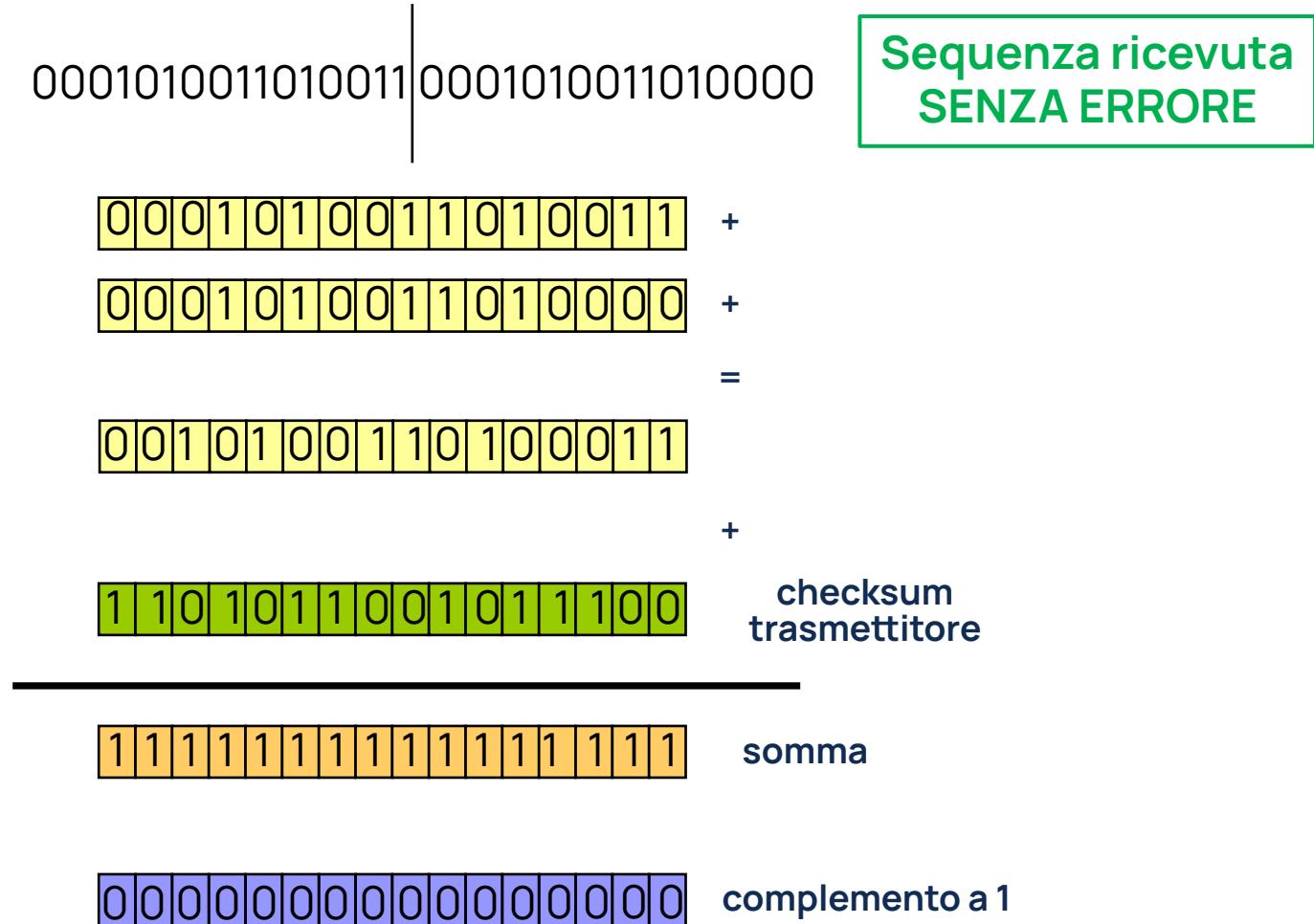
Calcolo del Checksum lato trasmettitore

- L'insieme di bit è diviso in blocchi da 16 bit
- Il campo *Checksum* è inizializzato a 0
- Tutti i blocchi vengono sommati tra loro
- 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 tra loro
- Il risultato è complementato
 - Se sono tutti 0 il pacchetto è accettato
 - Altrimenti è scartato



Calcolo del Checksum lato ricevitore

Sequenza ricevuta CON ERRORE

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

0001010011010011|0000010011010000

0001010011010011 +

0000010011010000 =

0001100110100011

1101011001011100

checksum
trasmettitore

1110111111111111

somma

0001bb0000000000

complemento a 1

Sequenza ricevuta in modo SBAGLIATO

TRASPORTO AFFIDABILE I

03

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?
- È possibile oppure è uno sforzo velleitario?
- È 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

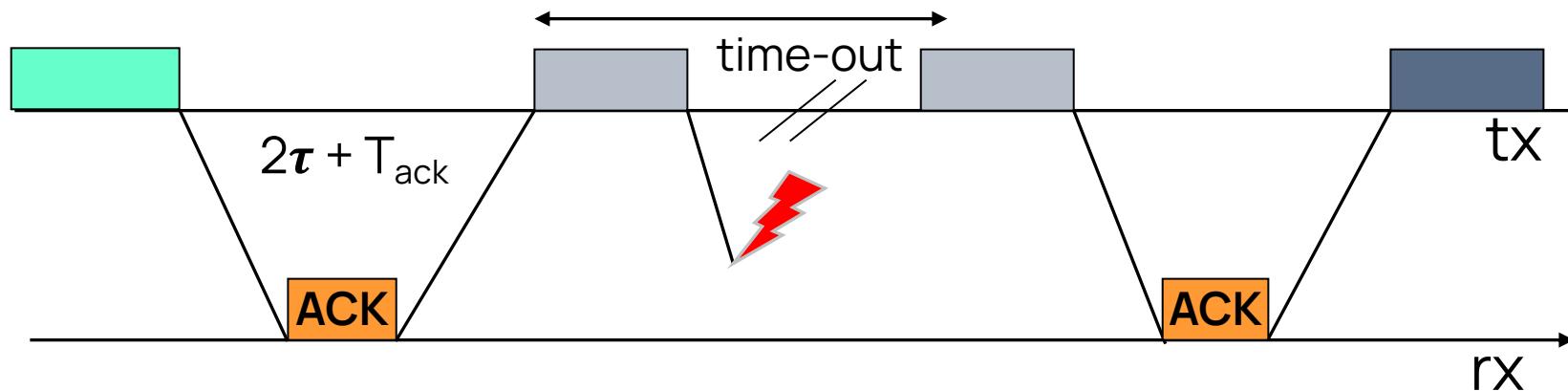
- ❑ Queste procedure possono essere attivate a qualunque livello
- ❑ Storicamente sono state sempre presenti a livello di linea sui collegamenti fisici a causa delle cattive linee fisiche del passato
- ❑ Presente a livello di trasporto per recupero *end-to-end*
 - ❑ Proteggere i collegamenti fisici non 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

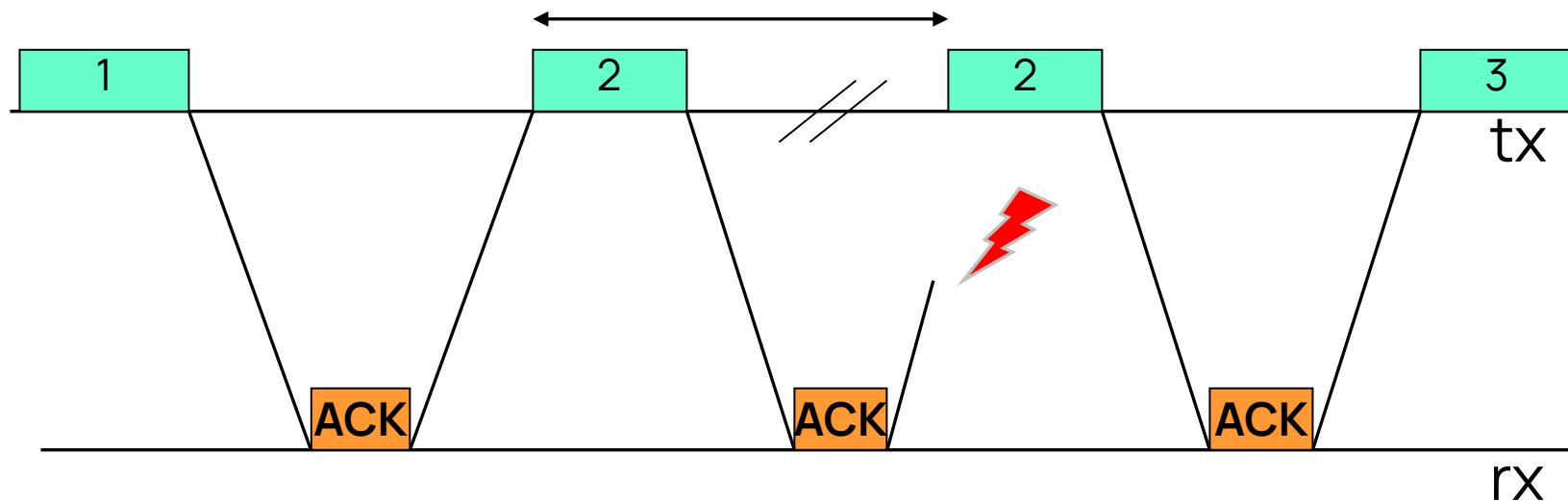
- Utilizza il solo ACK e un contatore di *time out*
- Ogni messaggio ricevuto correttamente è riscontrato dal ricevitore (ACK)
- Come funziona?
 - Il trasmettitore trasmette un pacchetto e inizializza un contatore (time-out)
 - Se il primo evento successivo è
 - l'ACK, trasmette il pacchetto successivo
 - il time-out, ritrasmette il pacchetto corrente



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

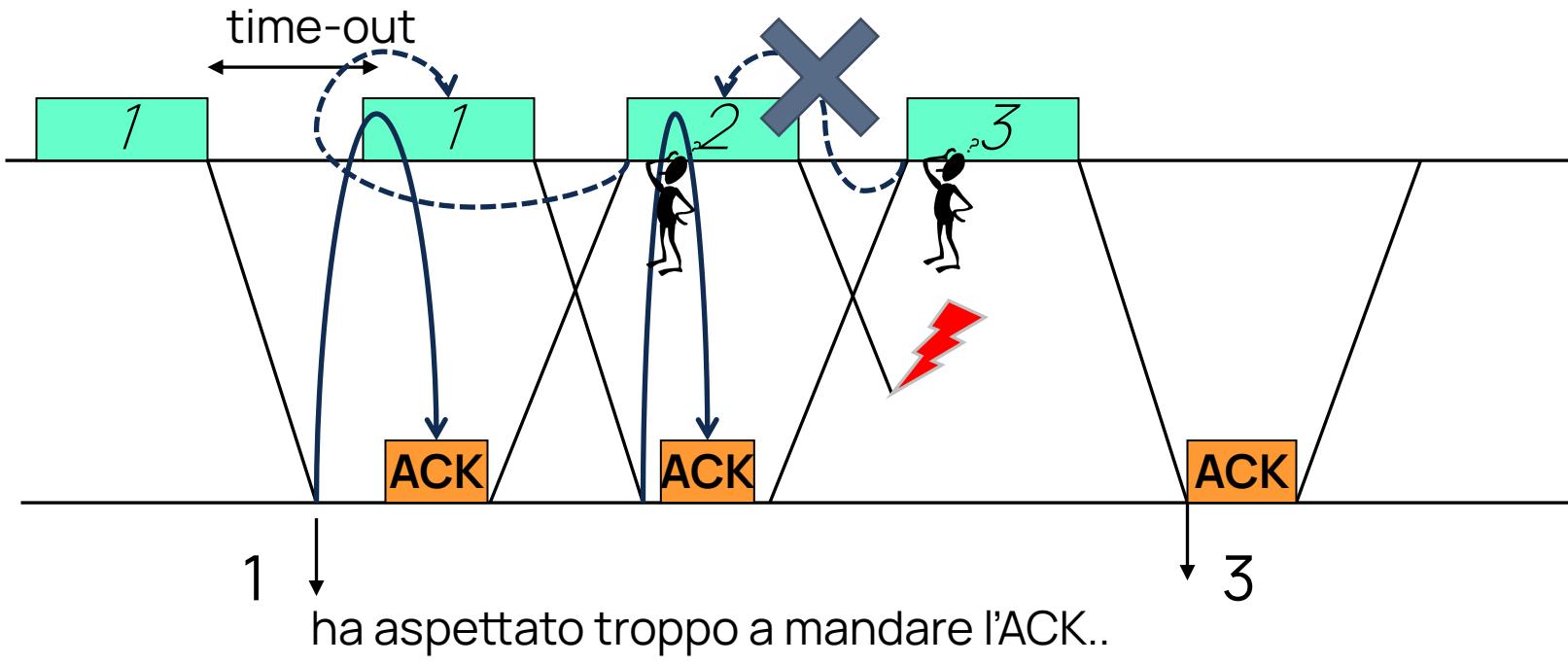
Protocollo *Stop and Wait*

- Problema 1: 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 2: errata interpretazione di un ACK. Il ricevitore crede un pacchetto ricevuto (il 2) mentre non lo è
- Rimedio: **numerazione anche degli ACK (RN)**

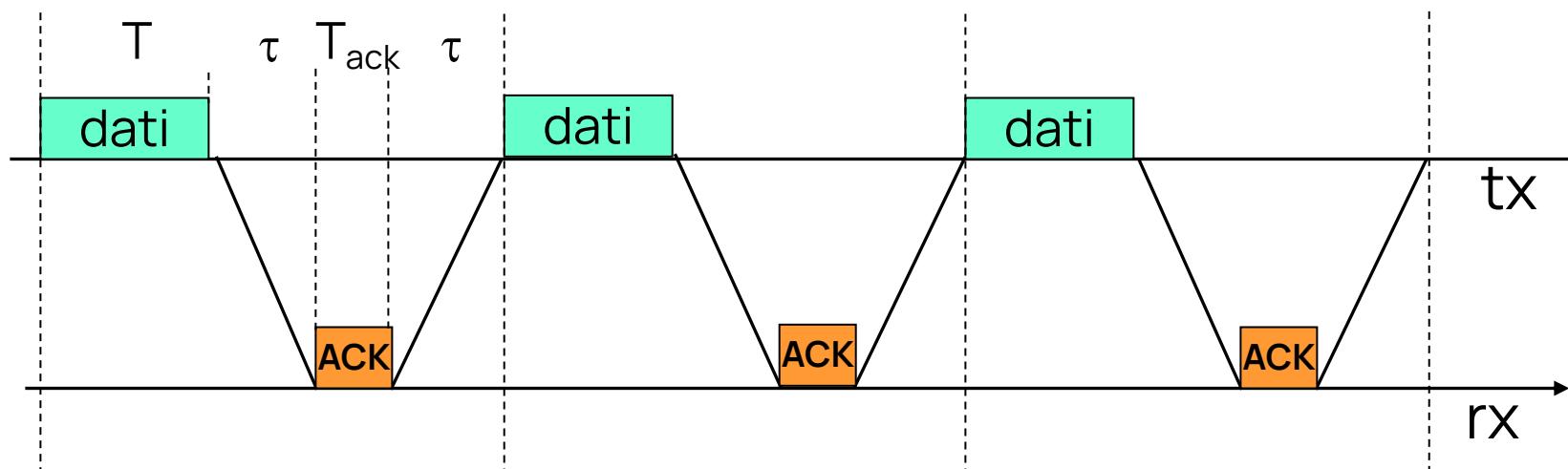


Protocollo *Stop and Wait*

- 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
 τ = 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*

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

Fondamenti di TELECOMUNICAZIONI

Prof. Marco Mezzavilla
marco.mezzavilla@polimi.it