1. Funzioni del livello di trasporto

Il livello di trasporto (layer 4) fornisce servizi di comunicazione end-to-end tra applicazioni in esecuzione su host diversi, mascherando i dettagli della rete sottostante.

1.1 Funzionalità principali

- **Trasporto end-to-end** tra processi applicativi
- **Multiplazione/demultiplazione** tra applicazioni diverse sullo stesso host
- **Controllo di flusso** per evitare sovraccarico del ricevitore
- **Controllo della congestione** per evitare sovraccarico della rete
- **Segmentazione e riassemblaggio** dei dati
- **Recupero errori** (in alcuni protocolli)
- **Garanzia di consegna** (in alcuni protocolli)
- **Ordinamento** dei pacchetti (in alcuni protocolli)

1.2 Posizionamento nel modello ISO/OSI

- Si trova tra il livello di rete (3) e il livello di sessione (5)
- È il primo livello completamente end-to-end (i livelli 1-3 operano hop-by-hop)
- Nasconde i dettagli della rete sottostante alle applicazioni
- Opera sull'intera comunicazione da sorgente a destinazione

1.3 Servizi offerti dai protocolli di trasporto

I protocolli di trasporto forniscono diversi tipi di servizi:

- **Connection-oriented**: stabilisce una connessione prima del trasferimento dati
- **Connectionless**: invia dati senza stabilire una connessione
- **Reliable**: garantisce consegna dei dati
- **Unreliable**: nessuna garanzia di consegna
- **Stream-oriented**: trattamento dei dati come flusso continuo di byte
- **Message-oriented**: trattamento dei dati come messaggi discreti

2. TCP (Transmission Control Protocol)

2.1 Caratteristiche principali

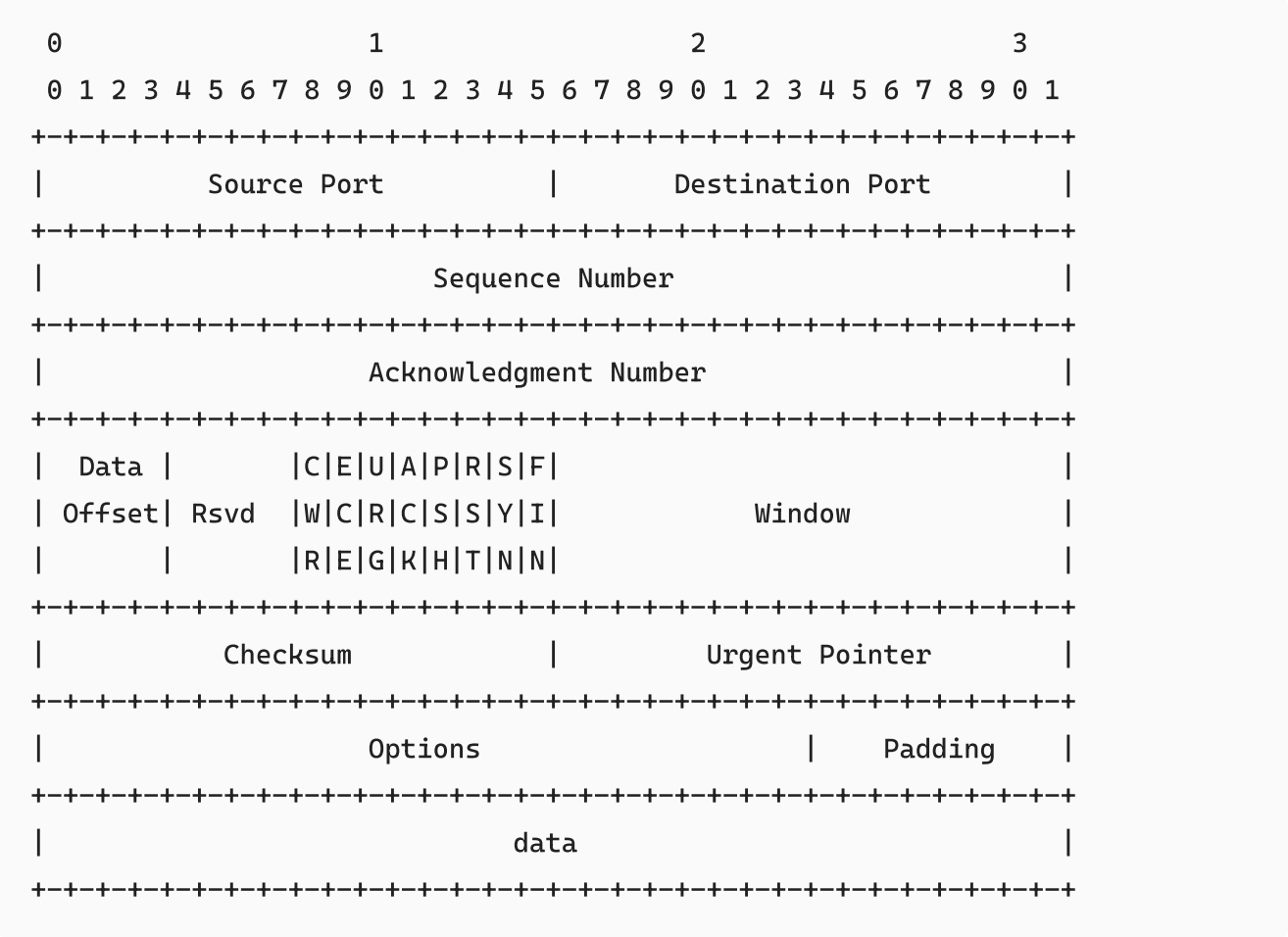
- **Orientato alla connessione**: stabilisce una connessione prima del trasferimento dati
- **Affidabile**: garantisce la consegna dei dati
- **Ordinamento garantito**: i dati vengono consegnati nell'ordine di invio

- **Controllo di flusso**: previene il sovraccarico del ricevitore
- **Controllo della congestione**: evita di sovraccaricare la rete
- **Stream-oriented**: tratta i dati come un flusso continuo di byte
- **Full-duplex**: comunicazione bidirezionale simultanea
- **Byte streaming**: interface basata su stream di dati

2.1.1 Applicazioni tipiche

- Web (HTTP/HTTPS)
- Email (SMTP, POP3, IMAP)
- Trasferimento file (FTP)
- Remote login (SSH)
- Ogni applicazione che richiede affidabilità

2.2 Struttura del segmento TCP

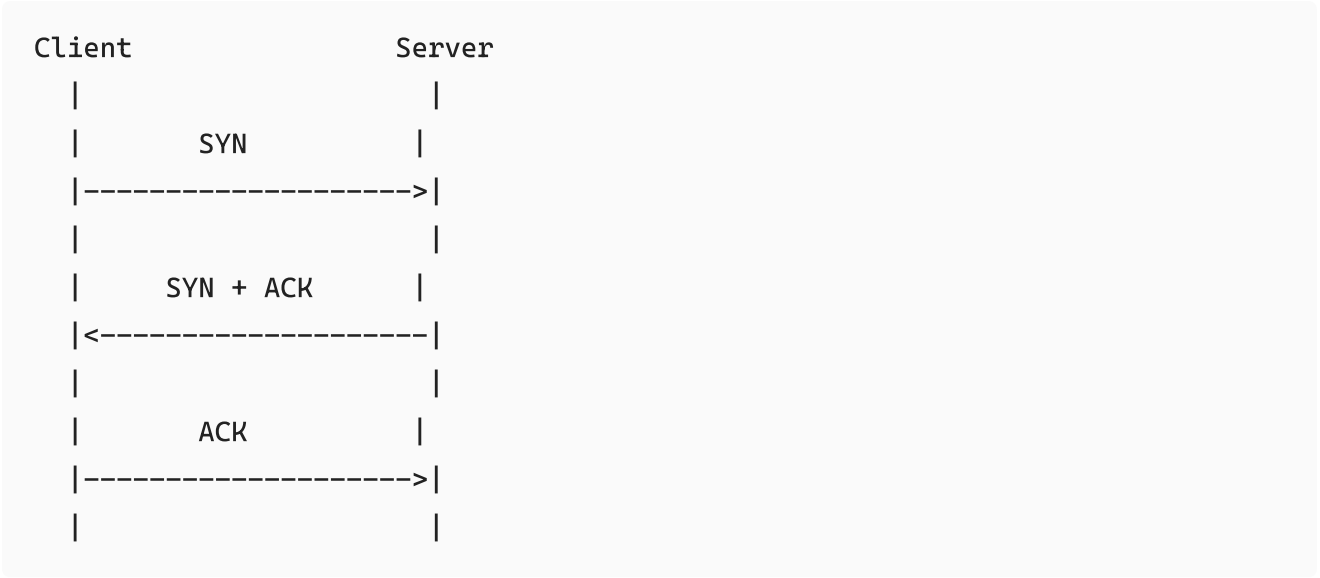


2.2.1 Campi principali

- **Source Port, Destination Port**: identificano le applicazioni mittente e destinataria (16 bit ciascuno)
- **Sequence Number**: numero di sequenza del primo byte di dati nel segmento (32 bit)
- **Acknowledgment Number**: prossimo byte atteso dal mittente (32 bit)

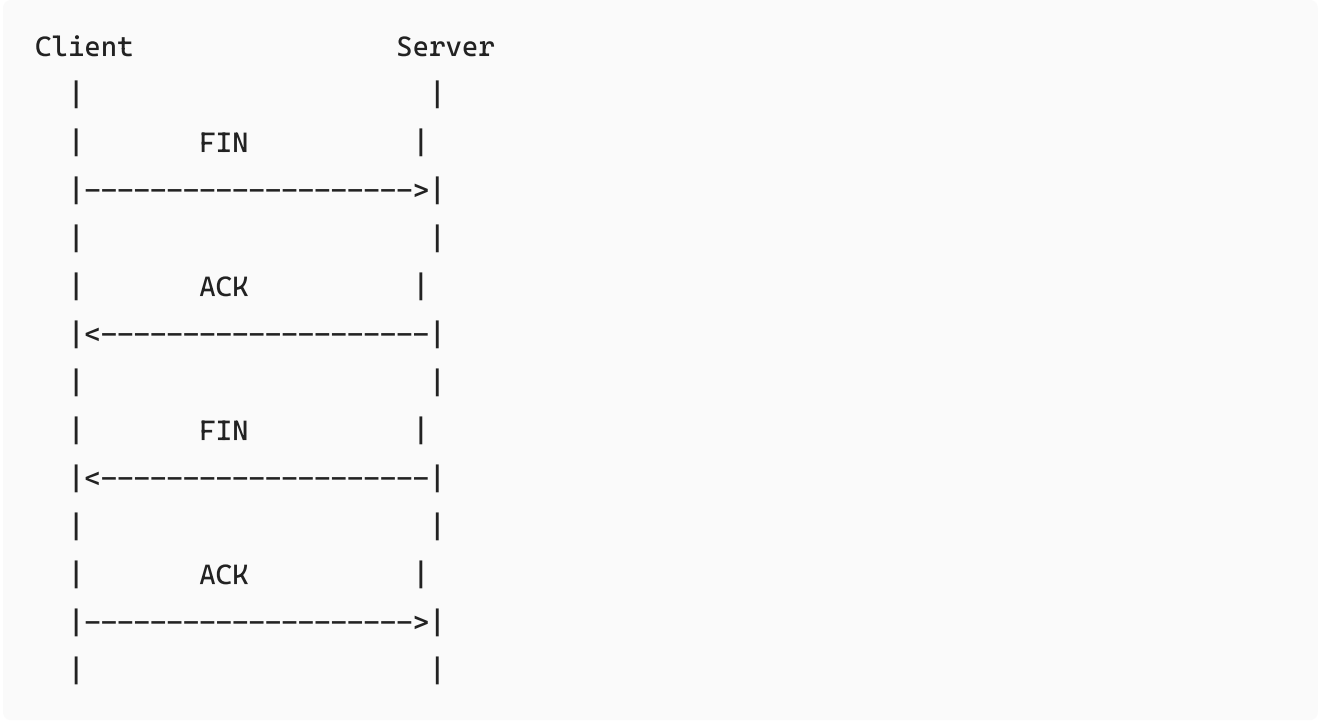
- **Data Offset**: lunghezza dell'header in parole di 32 bit (4 bit)
- **Flags**: bit di controllo (9 bit)
  - URG: campo Urgent Pointer valido
  - ACK: campo Acknowledgment valido
  - PSH: Push function (consegna immediata all'applicazione)
  - RST: Reset della connessione
  - SYN: Sincronizzazione dei numeri di sequenza
  - FIN: Fine dei dati (chiusura connessione)
- **Window**: dimensione della finestra di ricezione in byte (16 bit)
- **Checksum**: integrità del segmento (16 bit)
- **Urgent Pointer**: offset dei dati urgenti (16 bit)
- **Options**: opzioni (MSS, window scaling, timestamp, SACK)

2.3 Stabilimento connessione: Three-way handshake



1. **Client** → **Server**: SYN, seq=x
   - Il client invia un segmento con flag SYN impostato
   - Seleziona un numero di sequenza iniziale casuale x
2. **Server** → **Client**: SYN+ACK, seq=y, ack=x+1
   - Il server risponde con SYN e ACK impostati
   - Seleziona il suo numero di sequenza iniziale y
   - Conferma la ricezione del SYN con ack=x+1
3. **Client** → **Server**: ACK, seq=x+1, ack=y+1
   - Il client conferma la ricezione del SYN del server
   - La connessione è stabilita

2.4 Chiusura connessione: Four-way handshake



1. **Client** → **Server**: FIN, seq=x
   - Il client invia un segmento con flag FIN impostato
   - Indica che ha completato l'invio dei dati
2. **Server** → **Client**: ACK, ack=x+1
   - Il server conferma la ricezione del FIN
   - Può continuare a inviare dati
3. **Server** → **Client**: FIN, seq=y
   - Il server ha completato l'invio dei dati
   - Invia un segmento con flag FIN impostato
4. **Client** → **Server**: ACK, ack=y+1
   - Il client conferma la ricezione del FIN
   - La connessione è chiusa dopo un timeout (TIME_WAIT)

2.5 Stati di una connessione TCP

- **CLOSED**: nessuna connessione attiva
- **LISTEN**: in attesa di connessione
- **SYN-SENT**: inviato SYN, attesa risposta
- **SYN-RECEIVED**: ricevuto SYN, inviato SYN+ACK
- **ESTABLISHED**: connessione stabilita
- **FIN-WAIT-1**: inviato FIN
- **FIN-WAIT-2**: ricevuto ACK per FIN
- **CLOSE-WAIT**: ricevuto FIN, invio ACK
- **LAST-ACK**: inviato FIN, attesa ultimo ACK
- **TIME-WAIT**: attesa tempo 2MSL dopo chiusura
- **CLOSING**: inviato e ricevuto FIN contemporaneamente

2.6 Controllo di flusso

Il controllo di flusso evita che il mittente saturi il buffer di ricezione del destinatario.

- **Window Size**: campo nel header TCP che indica quanti byte il ricevitore può accettare
- Il mittente non può inviare più dati di quanto indicato nella finestra
- La finestra si riduce man mano che arrivano dati e si espande con gli ACK
- Meccanismo adattivo: la dimensione della finestra varia in base alla disponibilità del buffer
- **Window scaling** (opzione TCP) permette finestre fino a 1GB

2.7 Controllo della congestione

Il controllo della congestione evita il sovraccarico della rete adattando il tasso di trasmissione.

2.7.1 Slow Start

- Inizia con una piccola finestra di congestione (1 MSS)
- Raddoppia ad ogni RTT (Round Trip Time)
- Cresce esponenzialmente fino a raggiungere la soglia (ssthresh)
- Formula: cwnd = cwnd + MSS per ogni ACK ricevuto

2.7.2 Congestion Avoidance

- Inizia dopo lo Slow Start al raggiungimento della soglia
- Incremento lineare della finestra (1 MSS per RTT)
- Formula: cwnd = cwnd + MSS * (MSS/cwnd) per ogni ACK ricevuto
- Approccio più cauto per evitare congestione

2.7.3 Fast Retransmit

- Ritrasmissione rapida in caso di 3 ACK duplicati
- Non attende timeout, migliorando l'efficienza
- Indica perdita di singoli segmenti, non congestione grave

2.7.4 Fast Recovery

- Dopo Fast Retransmit, evita di ricominciare da Slow Start
- Dimezza la finestra di congestione e passa a Congestion Avoidance
- Permette di mantenere un throughput migliore durante perdite occasionali

2.8 Problemi tipici e ottimizzazioni

- **Slow Start**: avvio lento della connessione

- **Head of Line Blocking**: pacchetti bloccati in attesa di quelli persi
- **Bufferbloat**: buffer eccessivi che aumentano latenza
- **RTT fairness**: connessioni con RTT diversi ricevono bandwidth diverse
- **Nagle's Algorithm**: riduce overhead raggruppando dati piccoli
- **Delayed ACK**: ritarda gli ACK per ridurre il numero di segmenti

2.9 Parametri di connessione TCP

- **RTT (Round Trip Time)**: tempo di andata e ritorno
- **RTO (Retransmission Timeout)**: timeout per ritrasmissione
- **MSS (Maximum Segment Size)**: dimensione massima dati in un segmento
- **MTU (Maximum Transmission Unit)**: dimensione massima frame a livello 2
- **Bandwidth-delay product**: quantità di dati "in volo" = bandwidth * RTT
- **Initial congestion window**: finestra di congestione iniziale

3. UDP (User Datagram Protocol)

3.1 Caratteristiche principali

- **Connectionless**: senza connessione (non richiede handshake)
- **Non affidabile**: nessuna garanzia di consegna
- **Nessun ordinamento** garantito dei pacchetti
- **Nessun controllo di flusso o congestione**
- **Overhead minimo**: header piccolo e semplice
- **Latenza potenzialmente inferiore** rispetto a TCP
- **Message-oriented**: mantiene i confini dei messaggi

3.1.1 Vantaggi

- Semplicità di implementazione
- Bassa latenza (nessun handshake o controllo)
- Ridotto overhead per pacchetto
- Nessuna gestione dello stato della connessione
- Adatto per applicazioni in tempo reale

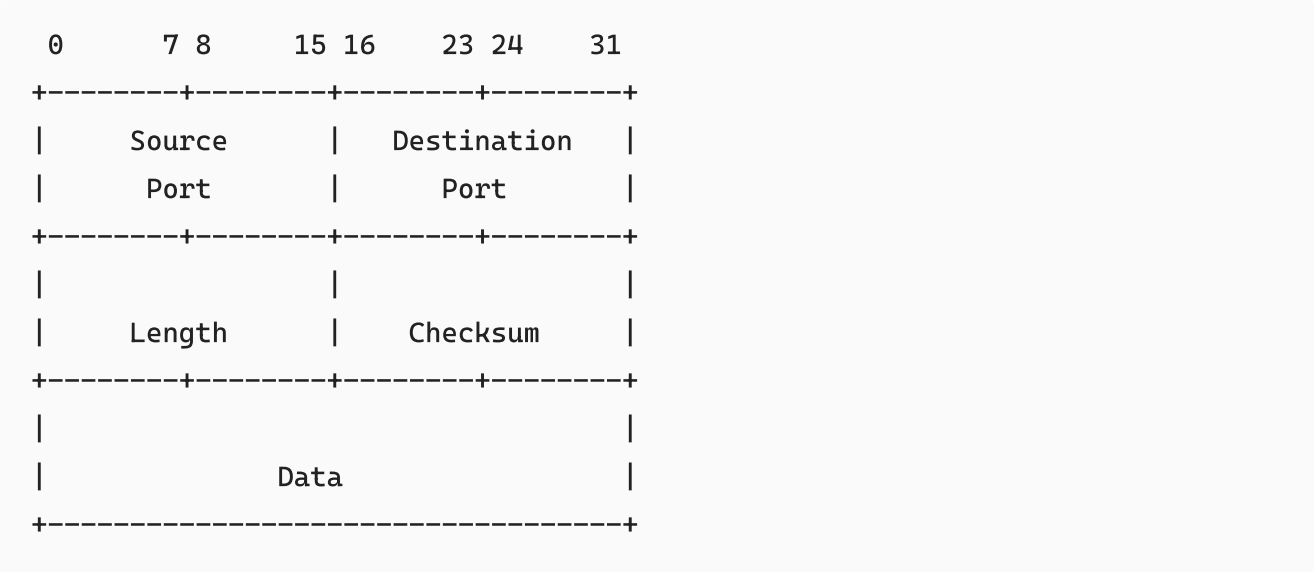
3.1.2 Svantaggi

- Nessuna garanzia di consegna
- Possibili perdite di pacchetti non rilevate
- Nessun riordinamento automatico
- Nessuna protezione contro la congestione
- Richiede gestione degli errori a livello applicativo

3.1.3 Applicazioni tipiche

- DNS (Domain Name System)
- Streaming video/audio
- VoIP (Voice over IP)
- Online gaming
- Sensori IoT
- Protocolli di discovery di rete

3.2 Struttura del datagramma UDP



3.2.1 Campi dell'header UDP

- **Source Port**: porta di origine (facoltativa, 0 se non usata)
- **Destination Port**: porta di destinazione
- **Length**: lunghezza totale (header + dati)
- **Checksum**: verifica integrità (facoltativa in IPv4, obbligatoria in IPv6)

3.2.2 Dimensione dell'header

- Solo 8 byte (contro i 20-60 byte di TCP)
- Riduce drasticamente l'overhead per datagrammi piccoli

3.3 UDP Lite

- Variante di UDP (RFC 3828)
- Checksum parziale che protegge solo header e parte iniziale dei dati
- Utile per applicazioni multimedia che possono tollerare errori nei dati
- Adatto per contenuti audio/video in tempo reale
- Migliora la QoE (Quality of Experience) in reti con perdite

3.4 Confronto UDP vs TCP

Caratteristica	TCP	UDP
Connessione	Connection-oriented	Connectionless
Affidabilità	Garantita	Non garantita
Ordinamento	Garantito	Non garantito
Controllo flusso	Sì	No
Controllo congestione	Sì	No
Overhead	Alto	Basso
Velocità	Potenzialmente più lenta	Potenzialmente più veloce
Uso bandwidth	Più efficiente	Meno efficiente
Dimensione header	20-60 byte	8 byte
Orientamento	Byte stream	Messaggi
Use case	Dati affidabili	Velocità, tempo reale

4. Porte e Socket

4.1 Concetto di porta

- Identificatore numerico (0-65535) per processi applicativi
- Permette multiplazione/demultiplazione delle comunicazioni
- Abbinata all'indirizzo IP forma un "socket address"

4.1.1 Categorie di porte

- **Well-known ports** (0-1023): servizi standard assegnati da IANA
  - HTTP: 80
  - HTTPS: 443
  - FTP: 20/21
  - SSH: 22
  - SMTP: 25
  - DNS: 53
- **Registered ports** (1024-49151): applicazioni registrate presso IANA
- **Dynamic/private ports** (49152-65535): allocate dinamicamente

4.2 Socket

- Endpoint di comunicazione
- Identificato da indirizzo IP + porta

- Rappresenta un canale di comunicazione bidirezionale
- Base per la programmazione di rete

4.2.1 Tipi principali di socket

- **Stream socket**: basati su TCP
  - Orientati alla connessione
  - Stream continuo di byte
  - Affidabili
- **Datagram socket**: basati su UDP
  - Senza connessione
  - Orientati ai messaggi
  - Non affidabili
- **Raw socket**: accesso diretto al livello IP
  - Permettono manipolazione diretta dei pacchetti
  - Usati per protocolli personalizzati o diagnostica

4.2.2 Funzioni tipiche API socket

Lato server:

- `socket()` : crea un nuovo socket
- `bind()` : associa un socket a un indirizzo locale
- `listen()` : predispone un socket per accettare connessioni
- `accept()` : accetta una connessione in entrata
- `recv()/read()` : riceve dati
- `send()/write()` : invia dati
- `close()` : chiude il socket

Lato client:

- `socket()` : crea un nuovo socket
- `connect()` : inizia una connessione verso un server
- `send()/write()` : invia dati
- `recv()/read()` : riceve dati
- `close()` : chiude il socket

5. Applicazioni dei protocolli di trasporto

5.1 Quality of Service (QoS)

La Quality of Service rappresenta l'insieme di tecnologie che garantiscono determinate prestazioni nella trasmissione dei dati.

5.1.1 Parametri di QoS

- **Bandwidth** (larghezza di banda): quantità di dati trasferibili nell'unità di tempo
- **Delay** (ritardo): tempo necessario per trasferire un pacchetto
- **Jitter** (variazione del ritardo): variazione nel tempo di arrivo dei pacchetti
- **Packet loss** (perdita di pacchetti): percentuale di pacchetti persi
- **Throughput** (capacità effettiva): quantità di dati effettivamente trasferiti

5.2 Problemi comuni del livello di trasporto

5.2.1 Fairness (equità)

- Distribuzione equa della banda disponibile tra flussi diversi
- TCP tende naturalmente a equità tra connessioni con RTT simili
- Problemi di fairness tra TCP e UDP (UDP non riduce il rate in caso di congestione)