

Lezione 7 - Livello di Trasporto

Servizi e protocolli di trasporto

- Forniscono la **comunicazione logica** tra processi applicativi di host differenti
- I protocolli di trasporto vengono eseguiti nei sistemi periferici:
 - lato invio: scinde (se necessario) i messaggi dell'applicazione, combinando ciascuna parte con un'intestazione per creare un segmento e lo passa al livello di rete
 - lato ricezione: riassembla i segmenti in messaggi e li passa al livello di applicazione
- I router nel cammino da un host all'altro operano solo sull'intestazione del datagramma, ignorando il segmento incapsulato al suo interno
- Più protocolli di trasporto sono a disposizione delle applicazioni
 - TCP, UDP

Relazione tra livello di trasporto e livello di rete

- **livello di trasporto** : comunicazione logica tra *processi*
 - si basa sui servizi del livello di rete e li potenzia
- **livello di rete** : comunicazione logica tra *host*

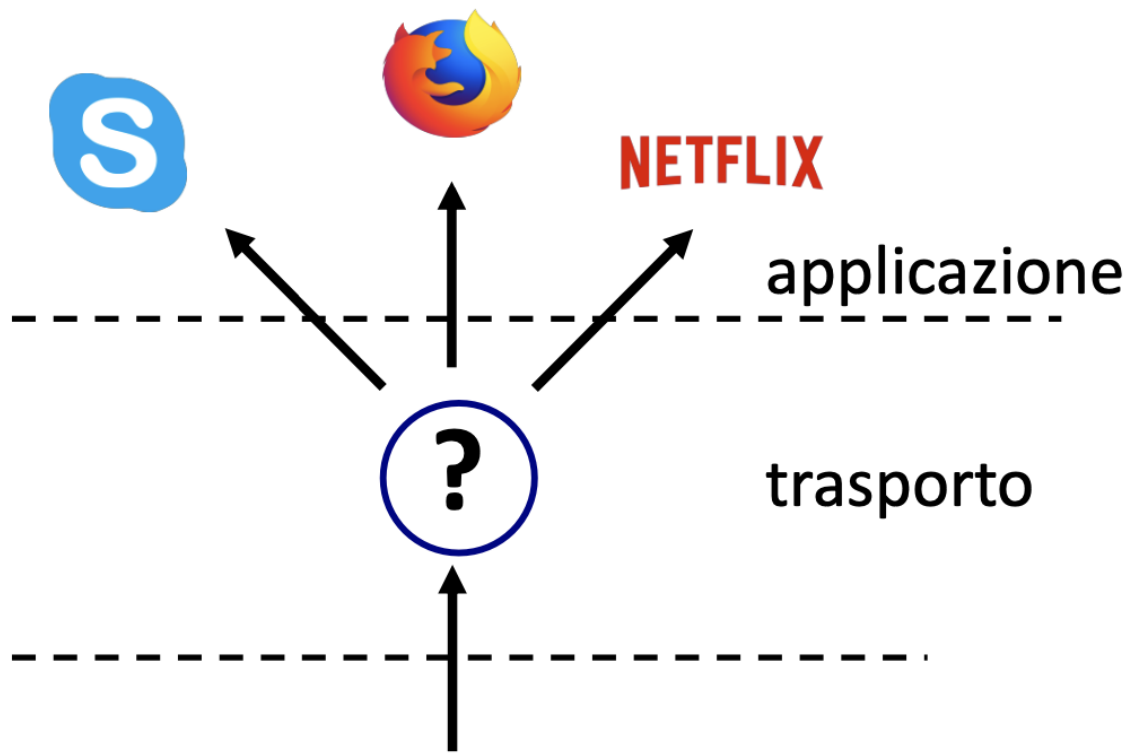
Protocolli del livello di trasporto in Internet

- **UDP** : User Datagram Protocol
 - inaffidabile, consegna senz'ordine
 - estensione "senza fronzoli" di IP: solo comunicazione tra processi e controllo degli errori

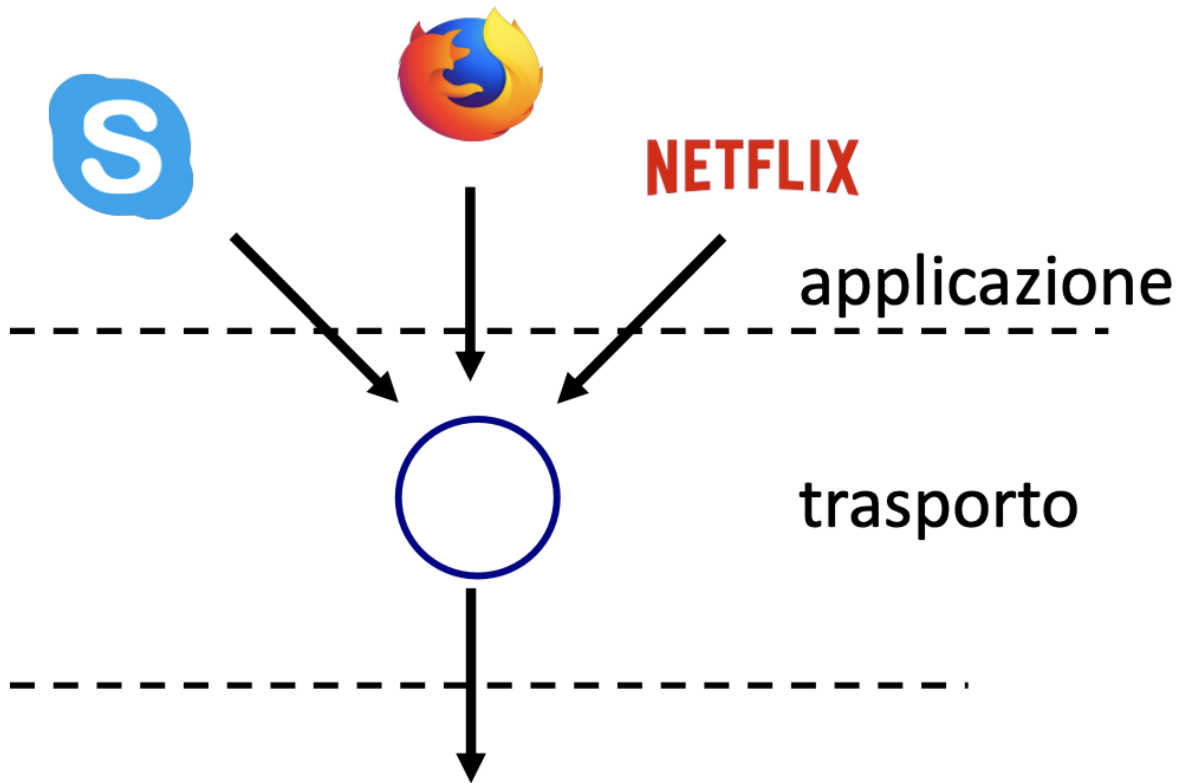
- **TCP** : Transmission Control Protocol
- comunicazione tra processi affidabile, consegne nell'ordine originario
- controllo di flusso
- controllo della congestione
- instaurazione della connessione
- servizi non disponibili:
 - garanzie su ritardi
 - garanzie su ampiezza di banda

Multiplexing / demultiplexing

- **Multiplexing lato mittente** : raccoglie i dati da varie socket , incapsulati con l'intestazione (utilizzata poi per il demultiplexing)
- **Demultiplexing lato ricevente** : utilizzare le informazioni nell'intestazione per consegnare i segmenti ricevuti alla socket corretta



de-multiplexing



multiplexing

Come funziona il demultiplexing

- l'host riceve i datagrammi IP
 - ogni datagramma ha un indirizzo IP di origine e un indirizzo IP di destinazione
 - ogni datagramma trasporta 1 segmento a livello di trasporto
 - ogni segmento ha un numero di porta di origine e un numero di porta di destinazione
- l'host usa gli indirizzi IP e i numeri porta per inviare il segmento alla socket appropriata



formato dei segmenti TCP/UDP

Demultiplexing senza connessione

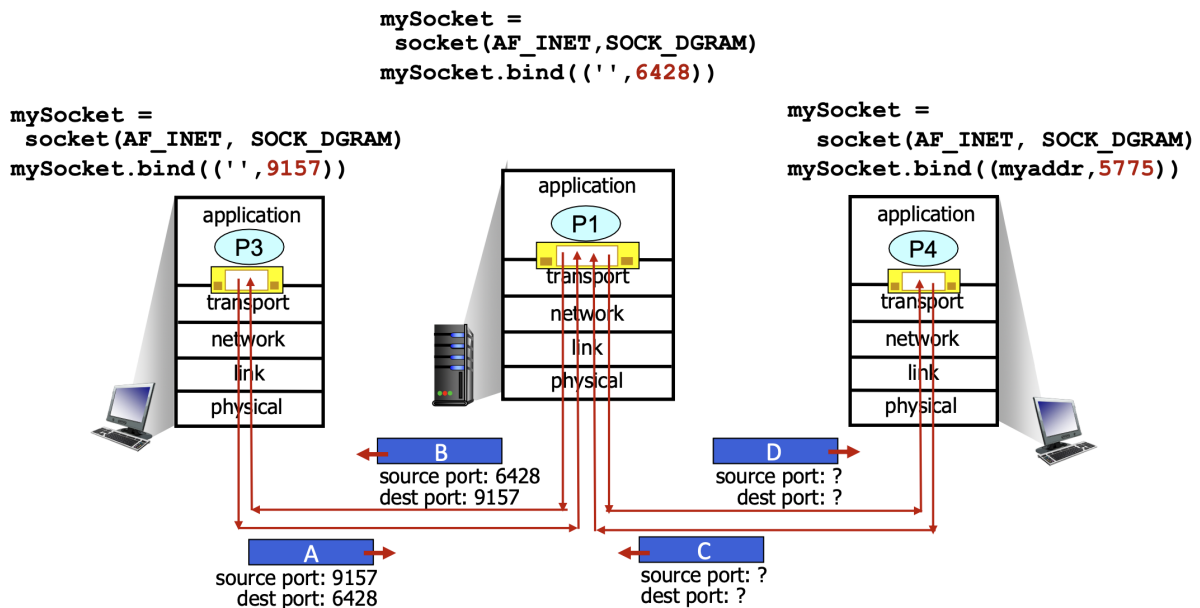
- quando si crea una socket, si deve specificare il numero di porta:
mySocket = socket(AF_INET, SOCK_DGRAM)
mySocket.bind(("",*9157**))**
- quando si crea il datagramma da inviare al socket UDP, si deve specificare
 - indirizzo IP di destinazione
 - numero di porta di destinazione
- Il segmento viene passato al livello di rete

quando l'host riceve il segmento UDP:

- controlla il numero della porta di destinazione nel segmento
- invia il segmento UDP alla socket con quel numero di porta



Datagrammi IP/UDP con lo *stesso indirizzo IP e numero di porta di destinazione*, ma indirizzi IP e/o numeri di porta di origine differenti vengono inviati alla *stessa socket* sull'host ricevente. Indirizzo IP e numero di porta di origine servono come "indirizzo di ritorno" per una eventuale risposta.



Demultiplexing orientato alla connessione

- la socket TCP è identificata da quattro parametri:
 - indirizzo IP di origine
 - numero di porta di origine
 - indirizzo IP di destinazione
 - numero di porta di destinazione
- demux: il lato ricevente usa i *quattro valori* (quadrupla) per inviare il segmento alla socket appropriata

- Un host server crea una *socket passiva* specificando un numero di porta
- La socket passiva viene usata per accettare le richieste di connessione, per ciascuna delle quali verrà creata una nuova socket connessa (con la medesima porta e indirizzo IP locale, ma diversa porta e indirizzo remoto, discriminando pertanto le socket connesse di client diversi)

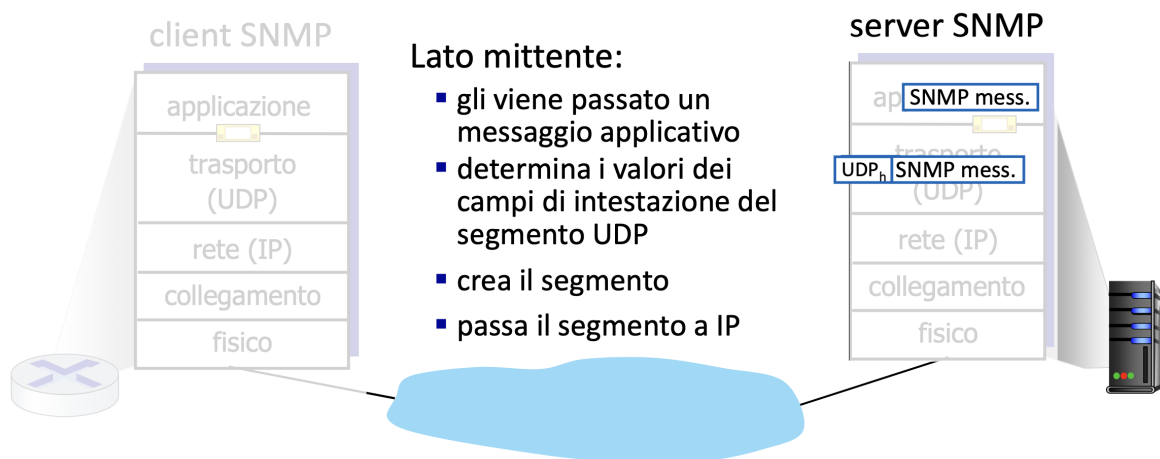
Riassunto

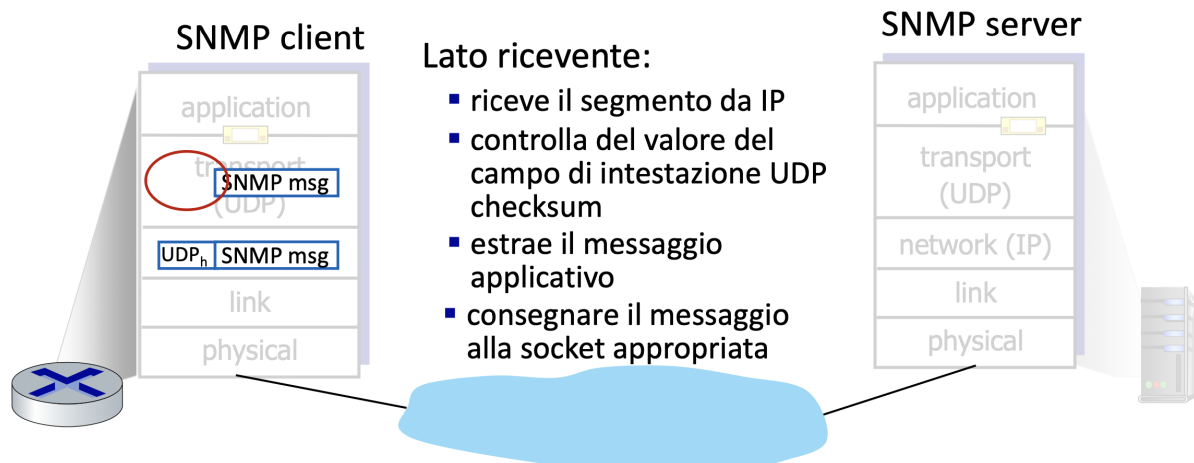
- Multiplexing, demultiplexing: basato sui valori dei campi dell'intestazione del segmento o del datagramma
- UDP: demultiplexing usando (solo) il numero di porta e indirizzo IP di destinazione
- TCP: demultiplexing usando la quadrupla di valori: indirizzi di origine e di destinazione, e numeri di porta
- Multiplexing/demultiplexing avviene a tutti i livelli (ogni volta che entità diverse vogliono usare i servizi del protocollo di livello inferiore)

UDP : User Datagram Protocol

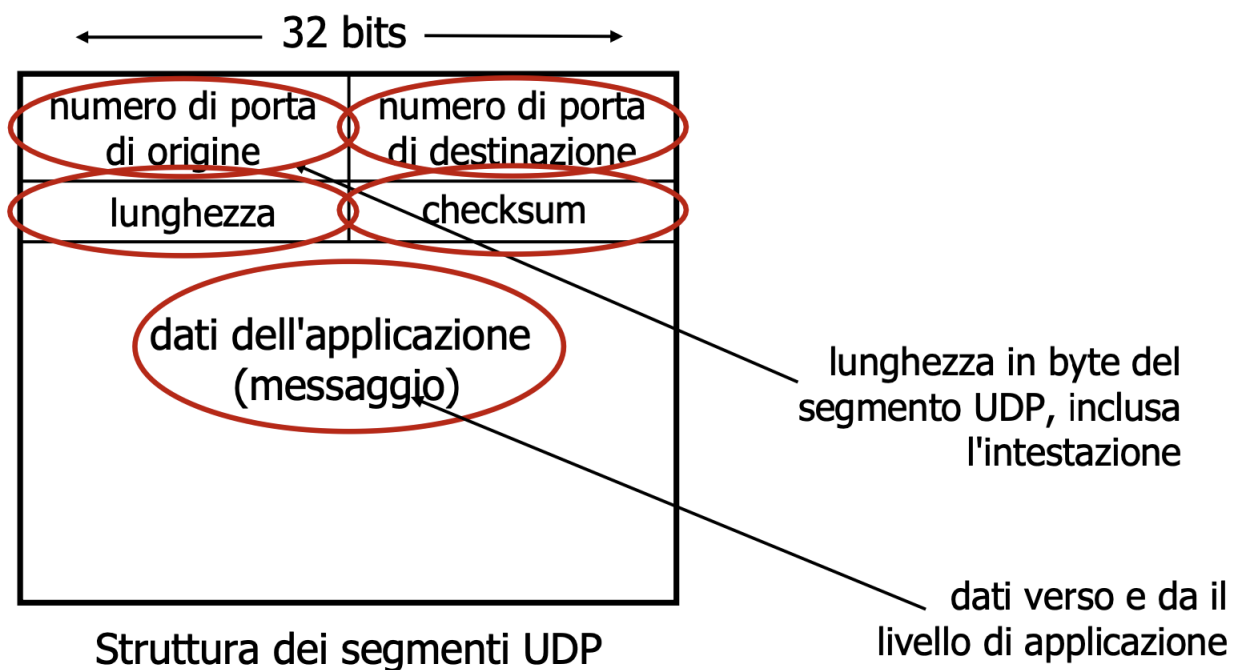
- protocollo di trasporto di Internet "senza fronzoli"
- servizio di consegna "best effort" (massimo sforzo), i segmenti UDP possono essere:
 - perduti
 - consegnati fuori sequenza all'applicazione
- **senza connessione**
 - no handshaking tra mittente e destinatario UDP
 - ogni segmento UDP è gestito indipendentemente
- **Perché esiste UDP?**

- nessuna connessione stabilita (che potrebbe aggiungere ritardo)
- semplice: nessuno stato di connessione nel mittente e nel destinatario (perciò un server può gestire più client)
- intestazioni di segmento corte
- senza controllo della congestione
 - UDP può sparare dati a raffica!
- controllo più preciso a livello di applicazione su quali dati sono inviati e quando (utile per requisiti di tasso di invio minimo e ritardi limitati)
- Utilizzo di UDP:
 - applicazioni per lo streaming multimediale (tolleranti alle perdite, sensibili alla frequenza)
 - DNS
 - SNMP
 - HTTP/3
- trasferimento affidabile con UDP (ad esempio, HTTP/3):
- aggiungere affidabilità a livello di applicazione
- aggiungere controllo della congestione a livello di applicazione






Struttura dei segmenti UDP



Checksum UDP

Obiettivo : rilevare gli "errori" (bit alternati) nel segmento trasmesso

	1° numero		2° numero		somma	
messaggio:	5		6		11	
						
trasmesso:	5	+	6	+	-11	= 0
						≠
ricevuto:	4	+	6	+	-11	= -1

errore rilevato dal ricevente

Mittente :

- tratta il contenuto del segmento come una sequenza di interi da 16 bit (inclusi i campi dell'intestazione UDP e gli indirizzi IP)
- **checksum:** complemento a 1 della somma (in complemento a 1) della sequenza di interi a 16 bit (considerando il campo checksum uguale a 0)
- pone il valore della checksum nel campo checksum del segmento UDP

Ricevente :

- calcola la checksum allo stesso modo del mittente (ma sommando anche il valore ricevuto del checksum)
- Il risultato è costituito da tutti bit 1 (-0 nell'aritmetica del complemento a 1)?
 - Sì - nessun errore rilevato. Ma potrebbero esserci errori nonostante questo? Lo scopriremo più avanti....
 - No - errore rilevato
- Altre implementazioni verificano la checksum calcolandola (allo stesso modo del mittente) e confrontandola col valore ricevuto

Esempio Checksum Internet

- esempio : sommare due interi da 16 bit

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
a capo	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
somma	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Nota: quando si sommano i numeri, un riporto dal bit più significativo deve essere sommato al risultato

Checksum Internet : protezione debole

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
a capo	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
somma	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Nonostante i numeri siano cambiati (bit alternati), *nessun* cambiamento nella checksum!