

Università degli Studi di Roma "Tor Vergata"
Laurea in Informatica

Sistemi Operativi e Reti
(modulo Reti)
a.a. 2023/2024

Livello di trasporto

dr. Manuel Fiorelli

manuel.fiorelli@uniroma2.it

<https://art.uniroma2.it/fiorelli>

Basate sulle slide del libro di testo:

https://gaia.cs.umass.edu/kurose_ross/ppt.php

Livello di trasporto: panoramica

Obiettivi:

- capire i principi che sono alla base dei servizi del livello di trasporto:
 - multiplexing, demultiplexing
 - trasferimento dati affidabile
 - controllo di flusso
 - controllo della congestione
- conoscere i protocolli del livello di trasporto di Internet:
 - UDP: trasporto senza connessione
 - TCP: trasporto orientato alla connessione
 - controllo della congestione TCP

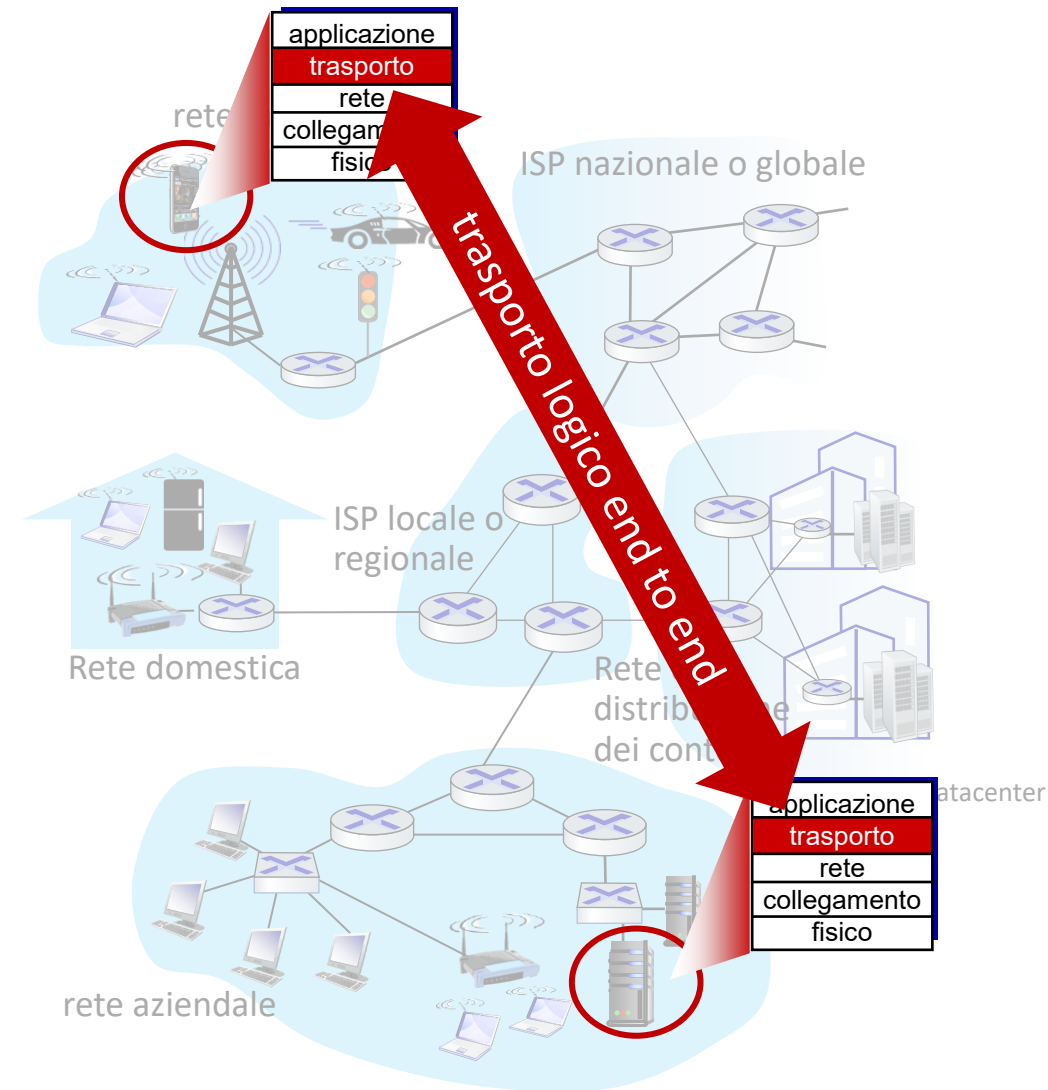
Livello di trasporto: tabella di marcia

- Servizi a livello di trasporto
- Multiplexing e demultiplexing
- Trasporto senza connessione: UDP
- Principi del trasferimento dati affidabile
- Trasporto orientato alla connessione: TCP
- Principi del controllo della congestione
- Controllo della congestione TCP
- Evoluzione della funzionalità del 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: riassume 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



Analogia domestica:

12 ragazzi a casa di Ann inviano lettere a 12 ragazzo a casa do Bill:

- host = case
- processi = ragazzi
- messaggi delle applicazioni= lettere nelle buste

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*

➡ multiplexing/demultiplexing

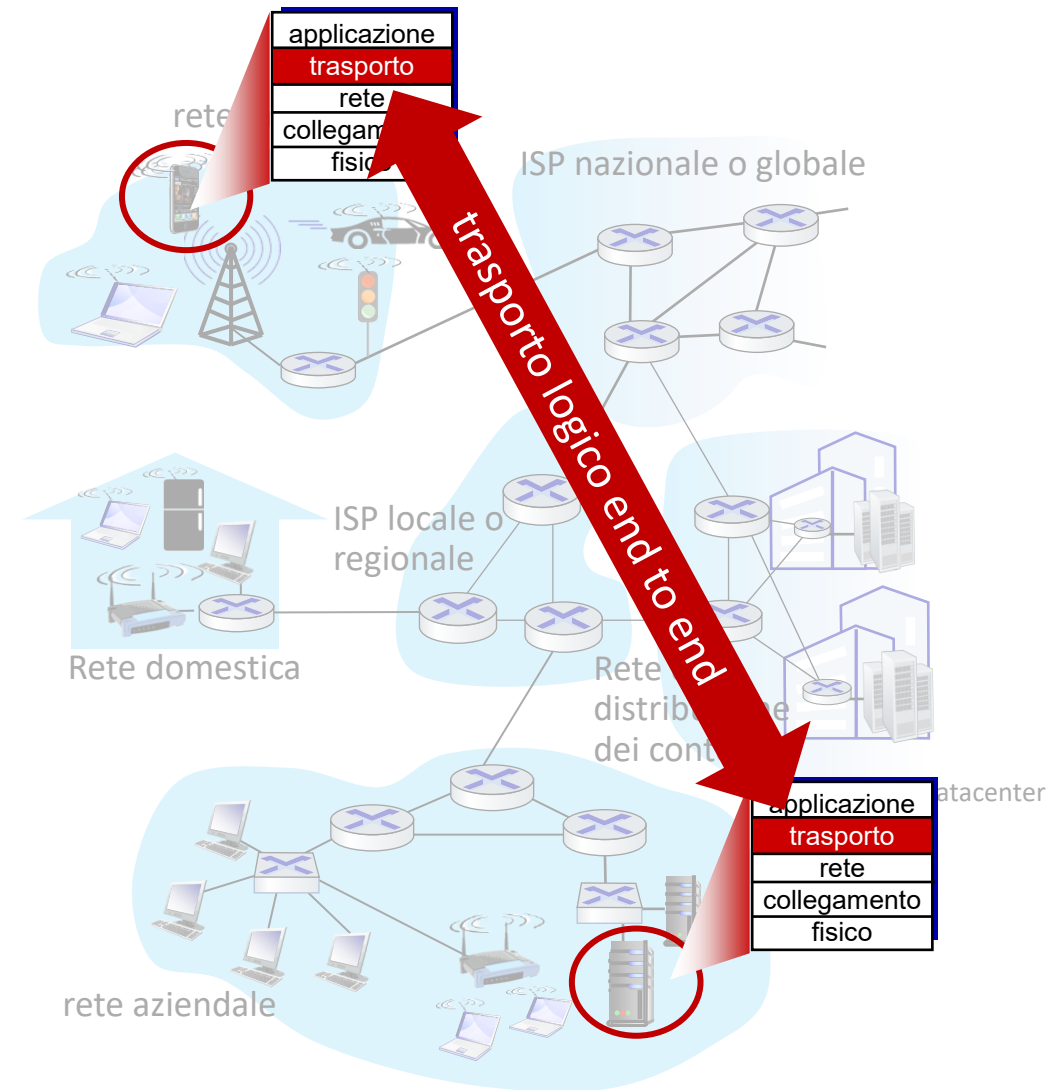
Analogia domestica:

12 ragazzi a casa di Ann inviano lettere a 12 ragazzo a casa do Bill:

- host = case
- processi = ragazzi
- messaggi delle applicazioni= lettere nelle buste

Protocolli del livello di trasporto in Internet

- **UDP:** User Datagram Protocol
 - inaffidabile, consegne 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



Capitolo 3: tabella di marcia

- Servizi a livello di trasporto
- **Multiplexing e demultiplexing**
- Trasporto senza connessione: UDP
- Principi del trasferimento dati affidabile
- Trasporto orientato alla connessione: TCP
- Principi del controllo della congestione
- Controllo della congestione TCP
- Evoluzione della funzionalità del livello di trasporto



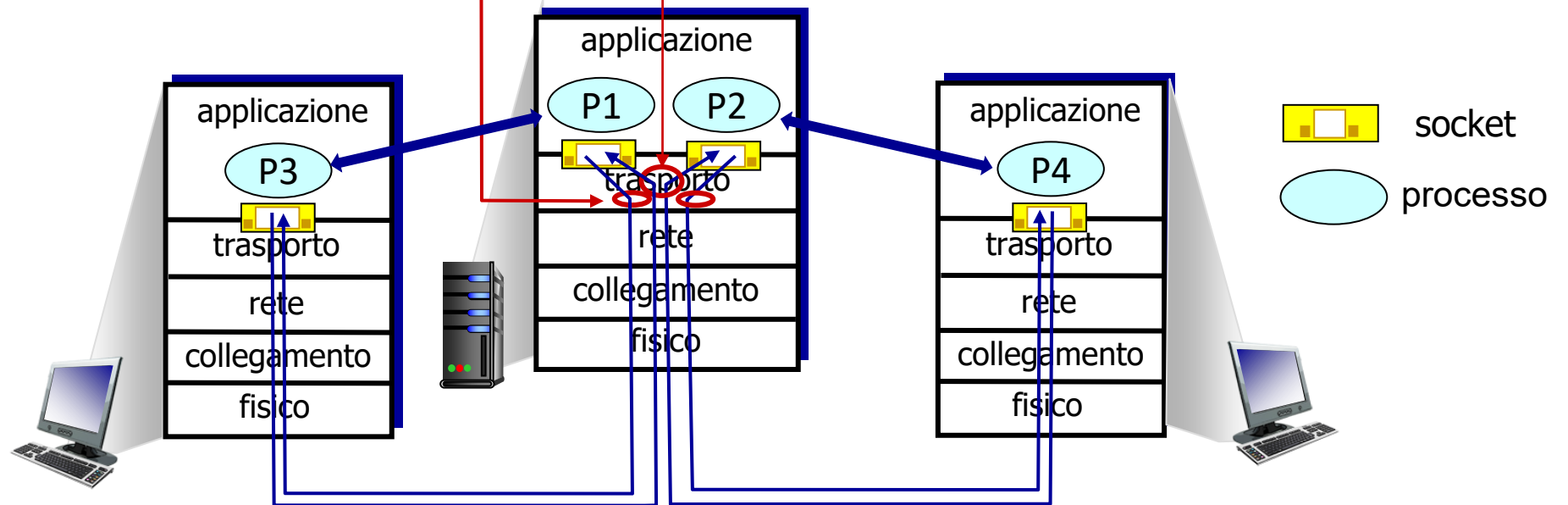
Multiplexing/demultiplexing

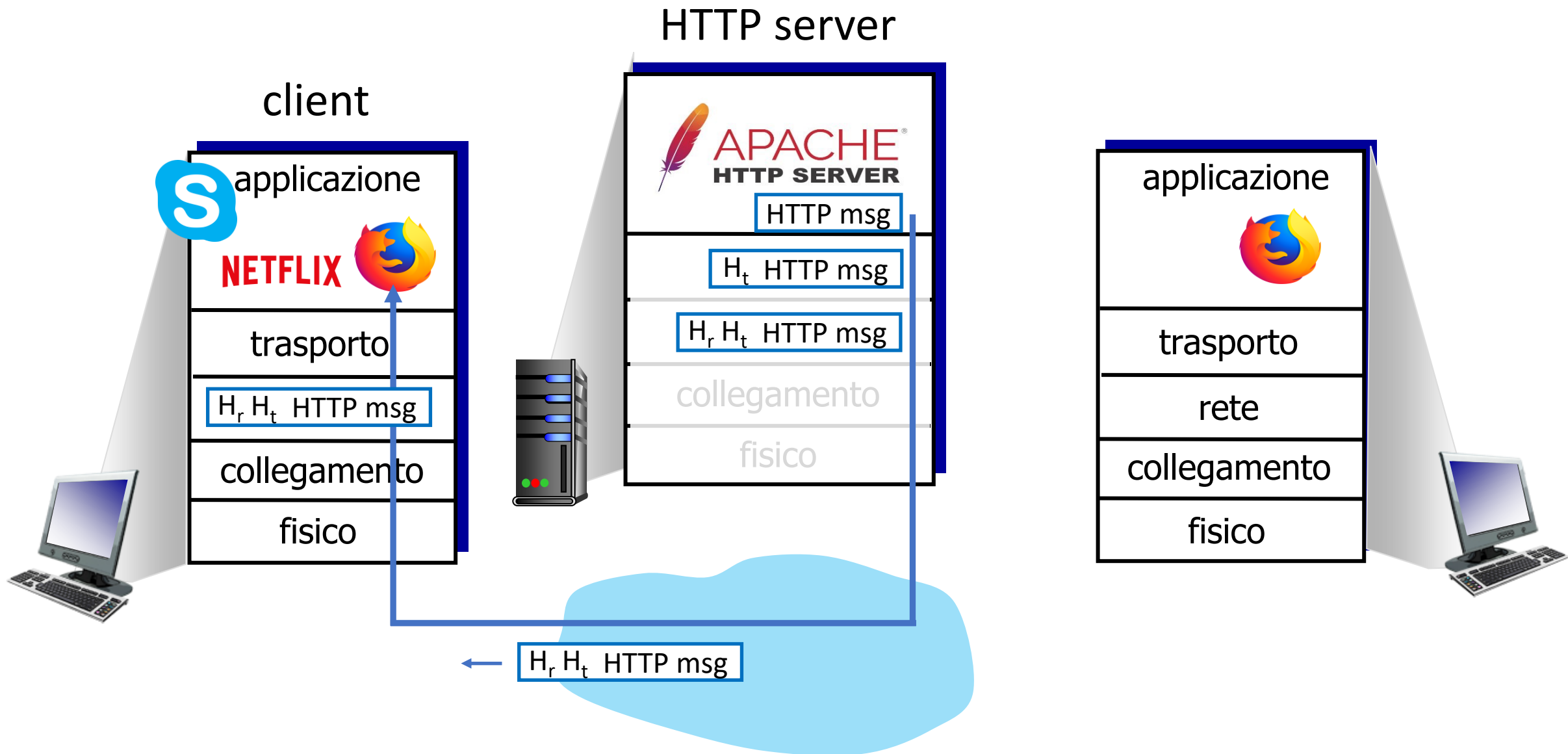
Multiplexing lato mittente:

raccogliere i dati da varie socket, incapsularli con l'intestazione (utilizzata poi per il demultiplexing)

demultiplexing lato ricevente:

utilizzare le informazioni nell'intestazione per consegnare i segmenti ricevuti alla socket corretta

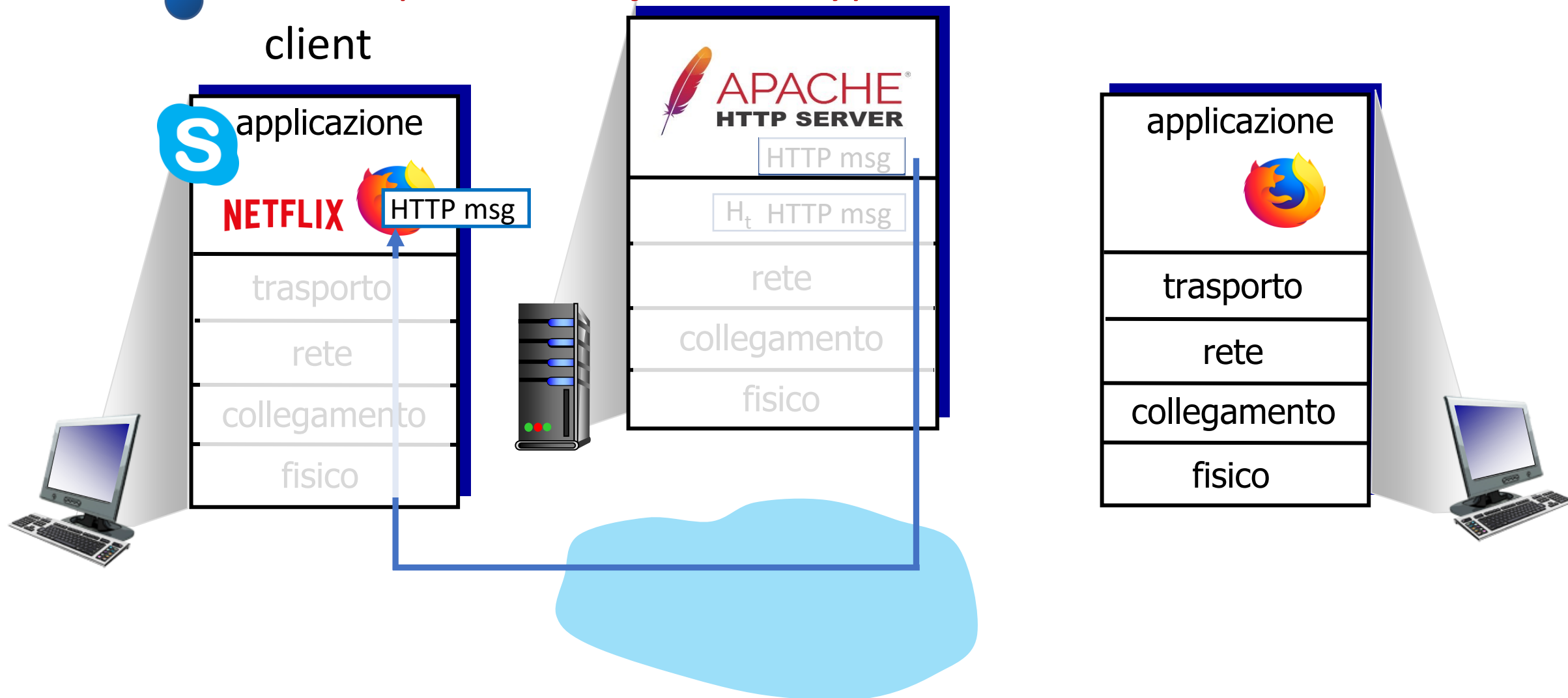


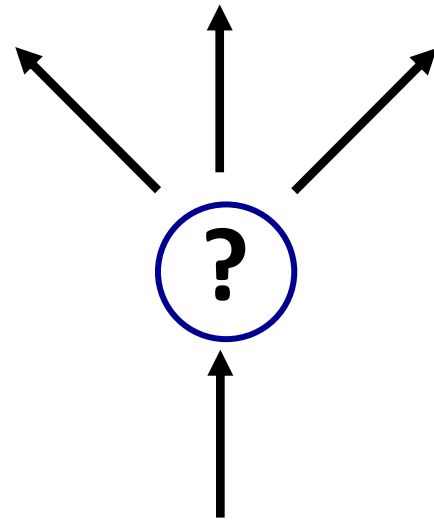




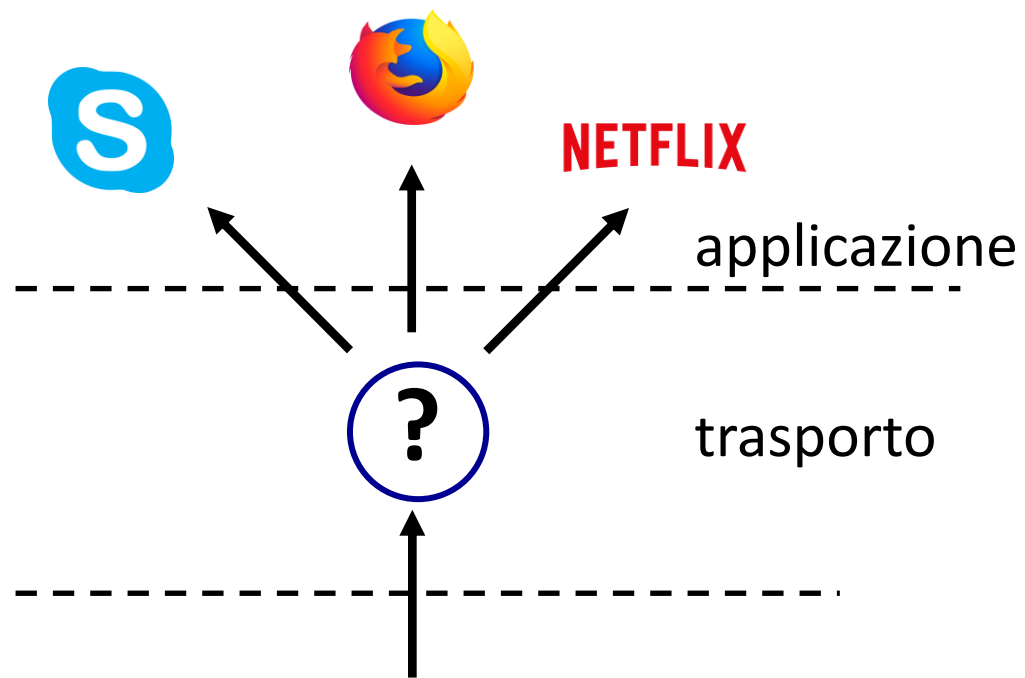
D: come ha fatto il livello di trasporto a sapere di dover consegnare il messaggio al processo del browser Firefox piuttosto che a quello di Netflix o di Skype?

client





de-multiplexing



de-multiplexing



Demultiplexing

AIRFRANCE 

ECONOMY 



AIRFRANCE 

SKY
PRIORITY™



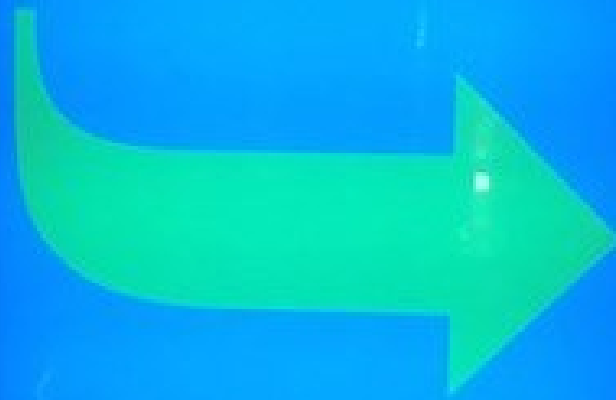
TSA Pre✓



Transportation
Security
Administration

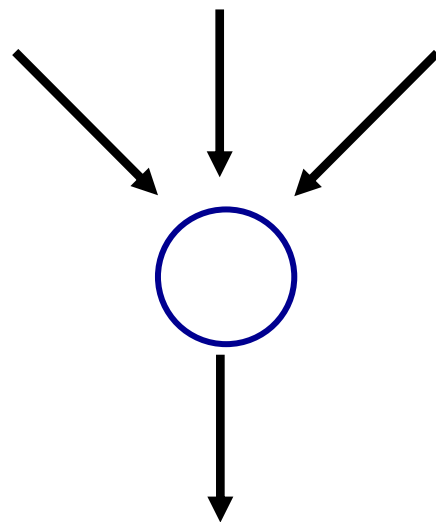
tsa.gov

Main
Checkpoint

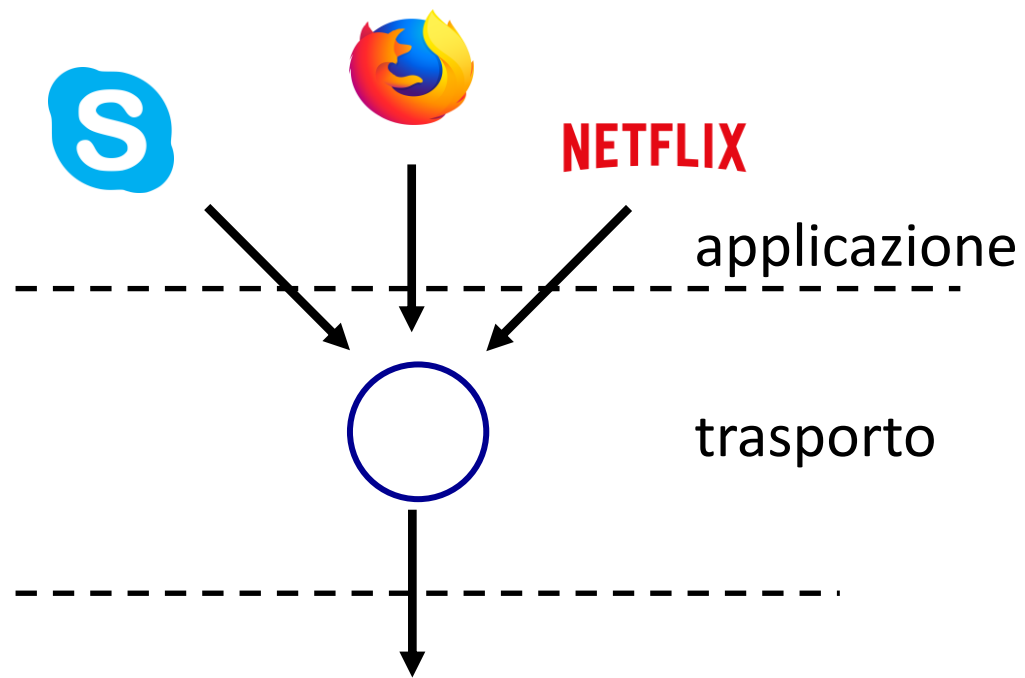


Transportation
Security
Administration

tsa.gov



multiplexing



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

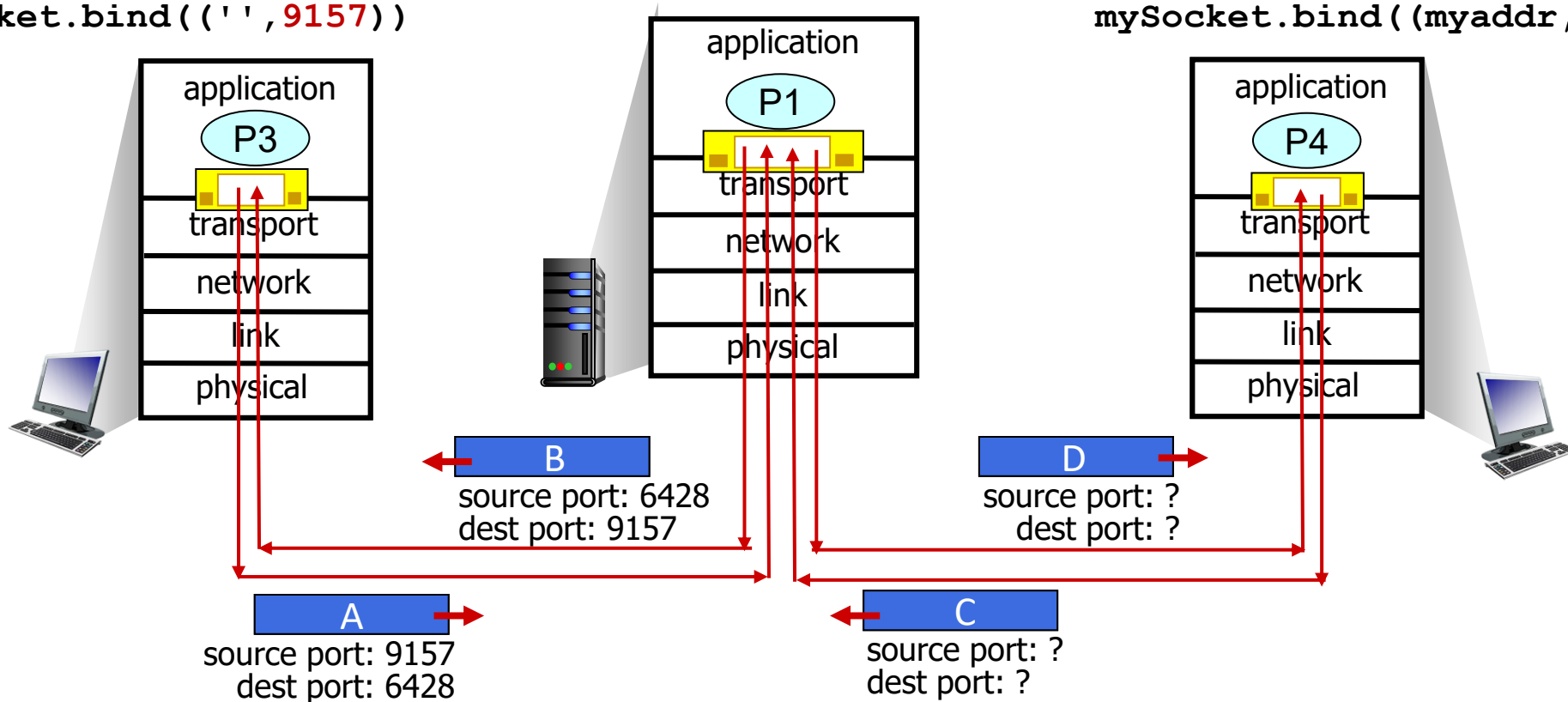
Indirizzo IP e numero di porta di origine servono come "indirizzo di ritorno" per una eventuale risposta

Demultiplexing senza connessione

```
mySocket =  
    socket(AF_INET, SOCK_DGRAM)  
mySocket.bind(('', 6428))
```

```
mySocket =  
    socket(AF_INET, SOCK_DGRAM)  
mySocket.bind(('', 9157))
```

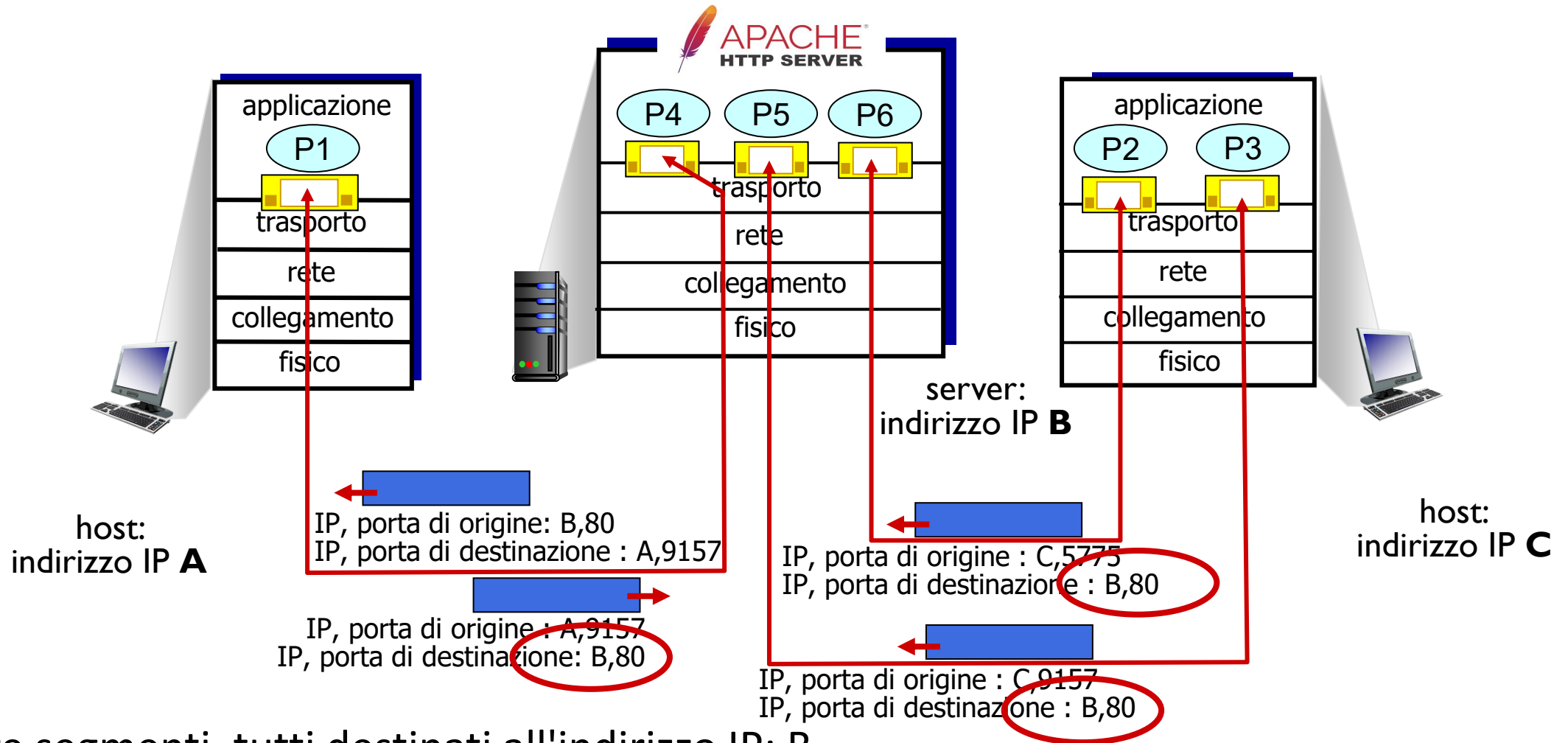
```
mySocket =  
    socket(AF_INET, SOCK_DGRAM)  
mySocket.bind((myaddr, 5775))
```



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)

Demultiplexing orientato alla connessione



Tre segmenti, tutti destinati all'indirizzo IP: B,
porta di destinazione: 80: ne viene fatto il demultiplexing verso socket *differenti*

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)

Nella creazione di una socket abbiamo usato ' ' (che in Python equivale a '0.0.0.0' nel caso di IPv4) per indicare qualunque indirizzo IP dell'host; tuttavia, avremmo potuto specificare uno in particolare.

Capitolo 3: tabella di marcia

- Servizi a livello di trasporto
- Multiplexing e demultiplexing
- **Trasporto senza connessione: UDP**
- Principi del trasferimento dati affidabile
- Trasporto orientato alla connessione: TCP
- Principi del controllo della congestione
- Controllo della congestione TCP
- Evoluzione della funzionalità del livello di trasporto



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)

UDP: User Datagram Protocol

- 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

UDP: User Datagram Protocol [RFC 768]

INTERNET STANDARD

RFC 768

J. Postel

ISI

28 August 1980

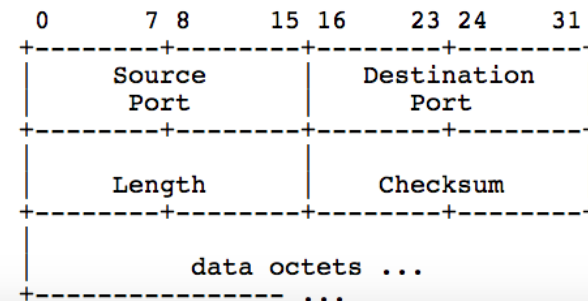
User Datagram Protocol

Introduction

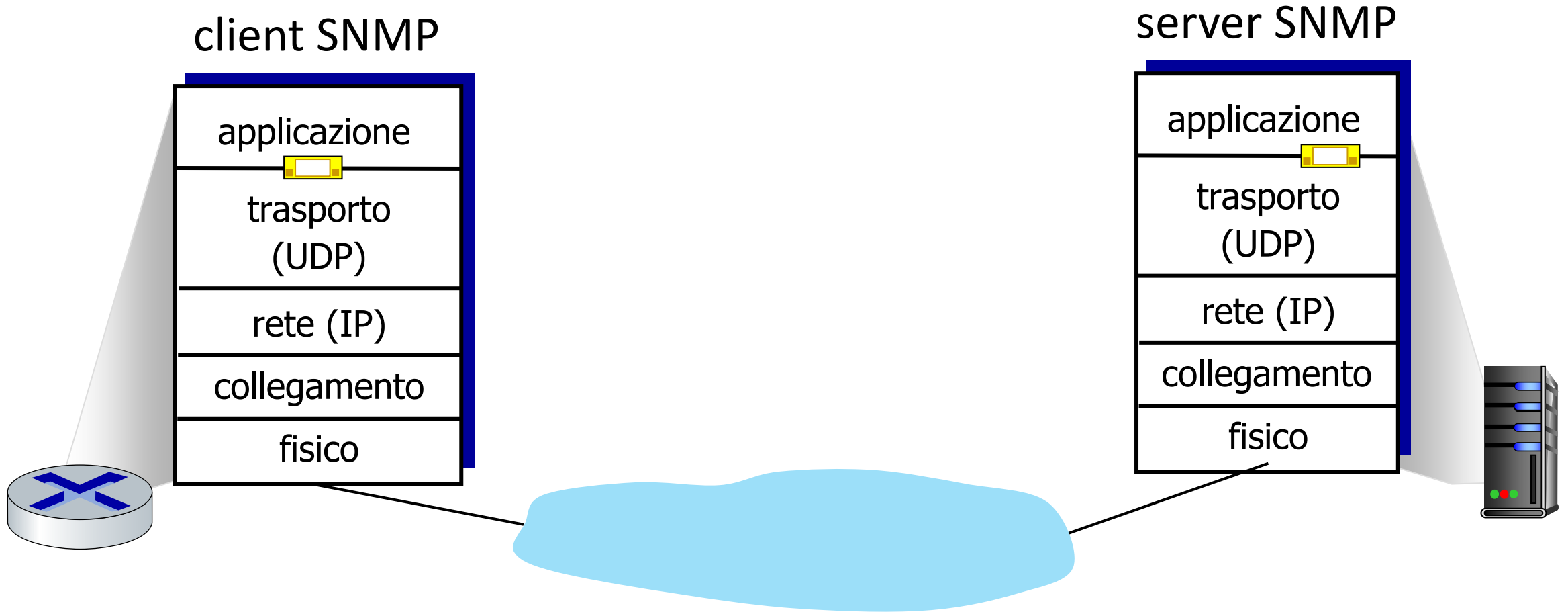
This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

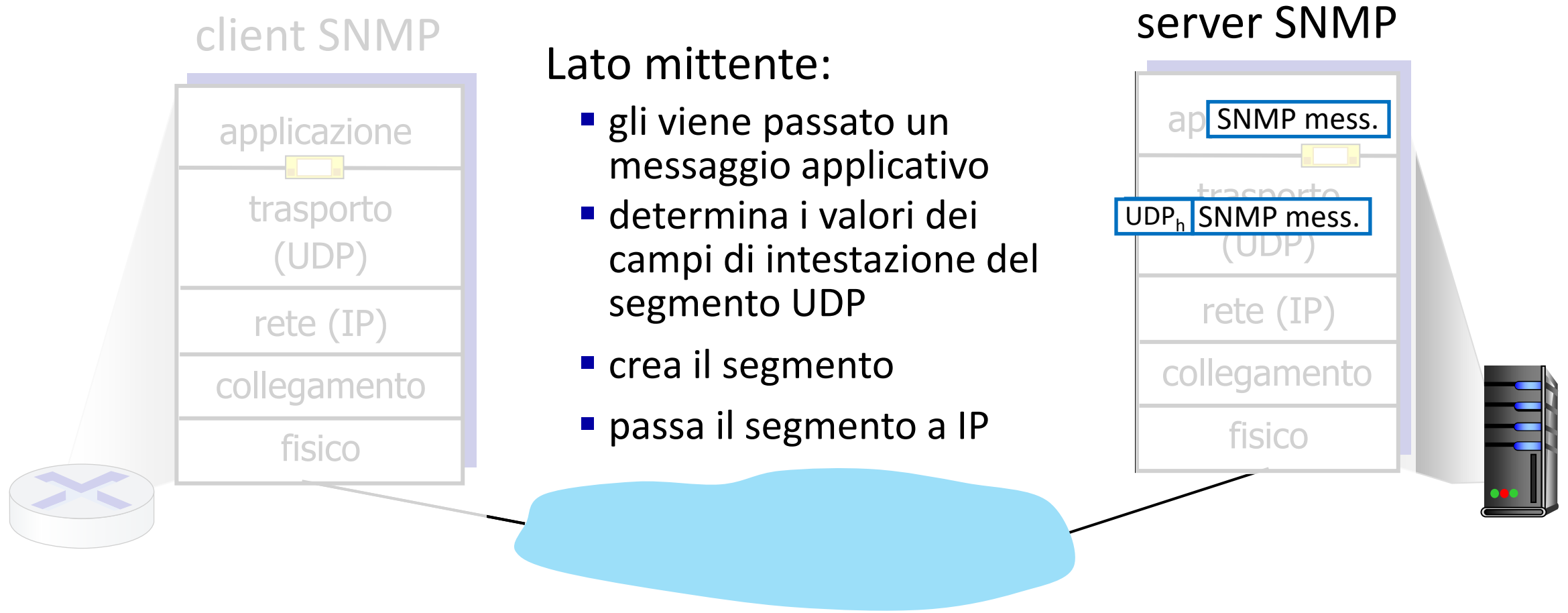
Format



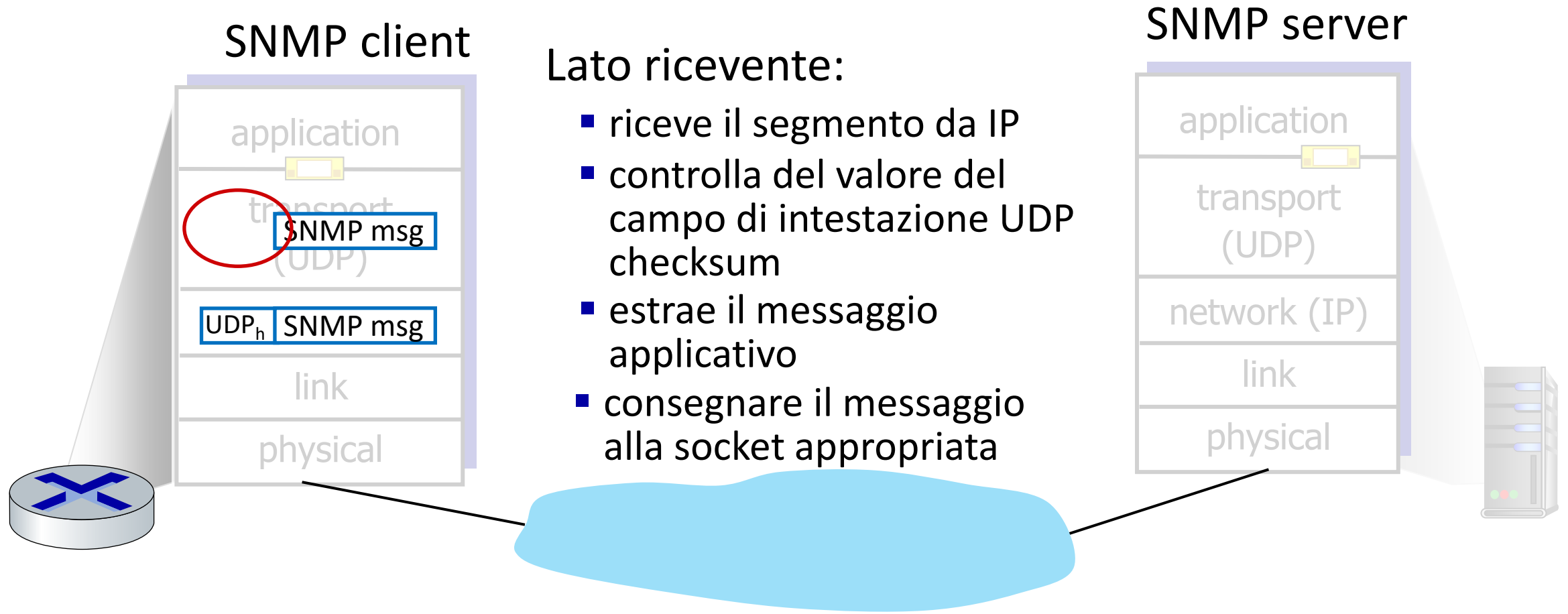
UDP: azioni del livello di trasporto



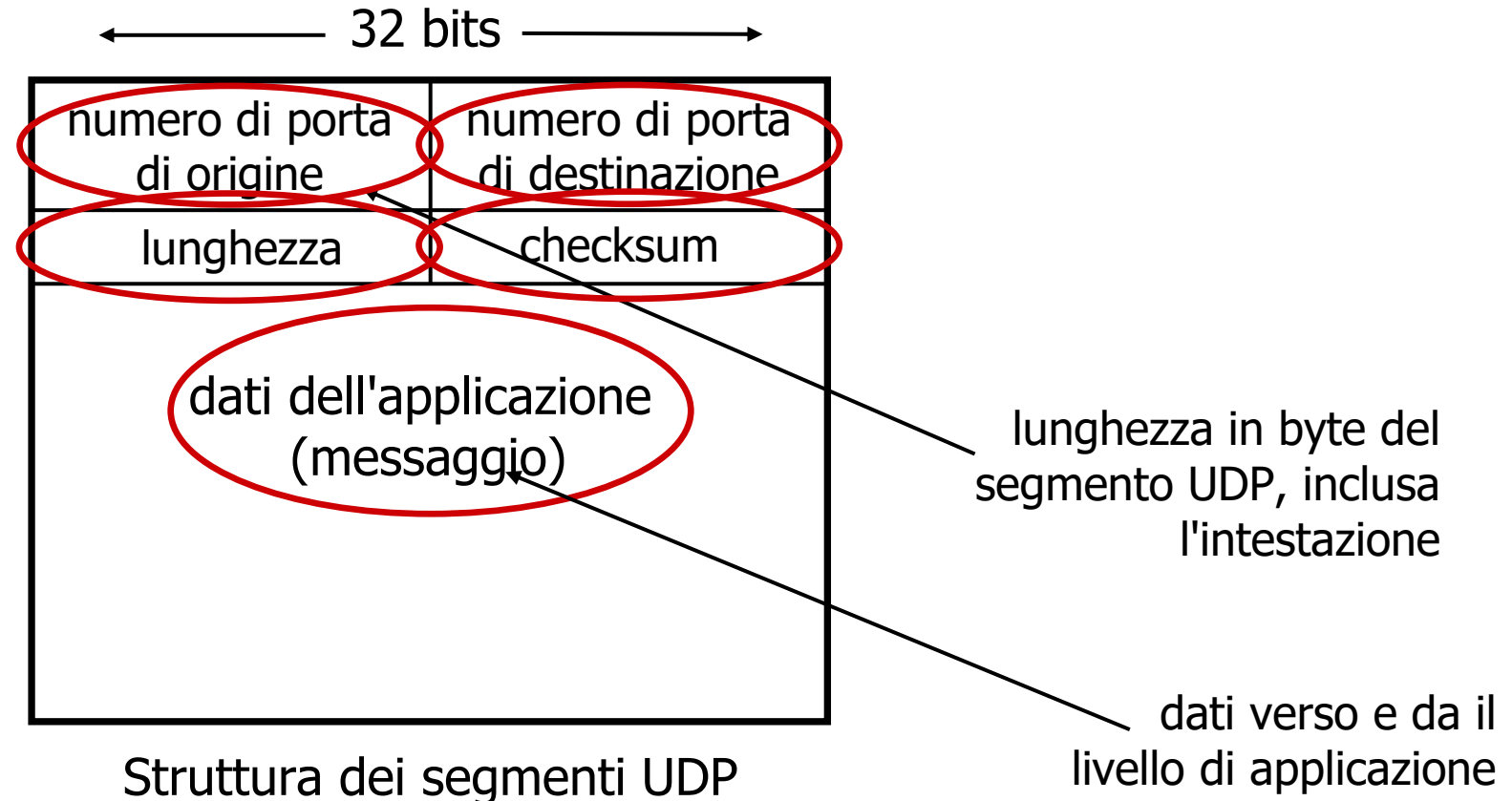
UDP: azioni del livello di trasporto



UDP: azioni del livello di trasporto

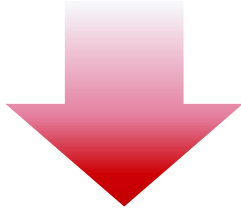


Struttura dei segmenti UDP



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

Checksum Internet [RFC 1071, RFC 1624]

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

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

Checksum Internet: un esempio

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	
<hr/>																	
a capo	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
<hr/>																	
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	0	1	1

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

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

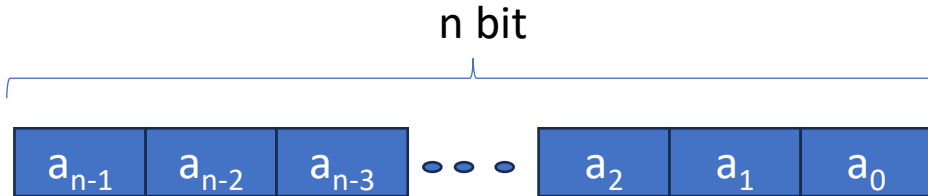
Checksum Internet: protezione debole

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
	<hr/>															
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!

Rappresentazione interi relativi in complemento a 1



Se $a_{n-1} = 0$, rappresenta un numero positivo (effettivamente di $n-1$ bit):

$$\sum_{i=0}^{n-2} a_i 2^i$$

Se $a_{n-1} = 1$, rappresenta un numero negativo, ottenuto invertendo tutti i bit:

$$-\sum_{i=0}^{n-2} \overline{a_i} 2^i$$

Possiamo rappresentare i numeri x nell'intervallo

$$-(2^{n-1} - 1) \leq x \leq 2^{n-1} - 1$$

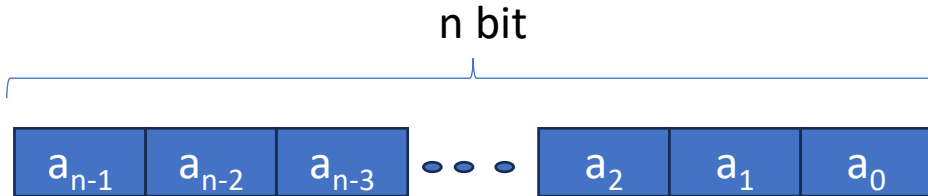
due rappresentazioni dello zero

a_2	a_1	a_0	Valore rappresentato in complemento a 1	Valore rappresentato senza segno
0	0	0	0	0
0	0	1	1	1
0	1	0	2	2
0	1	1	3	3
1	0	0	-3	4
1	0	1	-2	5
1	1	0	-1	6
1	1	1	-0	7

provate a sommare -2 e -1 per capire l'importanza del bit di riporto!

Se sommiamo un numero e il suo complemento a 1, otteniamo sempre tutti 1

Rappresentazione interi relativi in complemento a 1



Se $a_{n-1} = 1$, rappresenta un numero negativo, ottenuto invertendo tutti i bit:

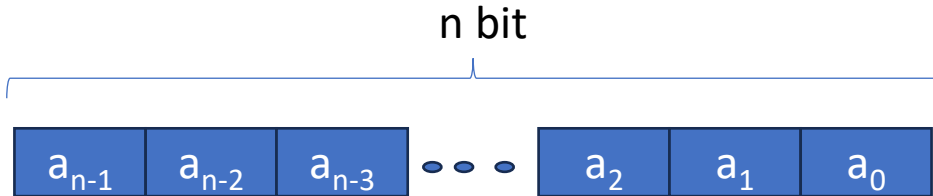
$$-\sum_{i=0}^{n-2} \overline{a_i} 2^i$$

Cioè

$$\begin{aligned} -\sum_{i=0}^{n-2} \overline{a_i} 2^i &= -\sum_{i=0}^{n-2} (1 - a_i) 2^i \\ &= -\sum_{i=0}^{n-2} 2^i + \sum_{i=0}^{n-2} a_i 2^i \\ &= (-2^{n-1} + 1) + \sum_{i=0}^{n-2} a_i 2^i \end{aligned}$$

a_2	a_1	a_0	Valore rappresentato in complemento a 1	Valore rappresentato senza segno
0	0	0	0	0
0	0	1	1	1
0	1	0	2	2
0	1	1	3	3
1	0	0	-3	4
1	0	1	-2	5
1	1	0	-1	6
1	1	1	-0	7

Rappresentazione interi relativi in complemento a 1



Rappresenta il numero relativo

$$a_{n-1}(-2^{n-1} + 1) + \sum_{i=0}^{n-2} a_i 2^i$$

a_2	a_1	a_0	Valore rappresentato in complemento a 1	Valore rappresentato senza segno
0	0	0	0	0
0	0	1	1	1
0	1	0	2	2
0	1	1	3	3
1	0	0	-3	4
1	0	1	-2	5
1	1	0	-1	6
1	1	1	-0	7

Riassunto: UDP

- protocollo “senza fronzoli” :
 - i segmenti possono essere persi, consegnati fuori sequenza
 - servizio best effort: “invia e spera per il meglio”
- UDP ha i propri vantaggi:
 - nessun bisogno di setup/handshaking (non si incorre in alcun RTT)
 - può funzionare quando il servizio di rete è compromesso
 - aiuta con l'affidabilità (checksum)
- costruire funzionalità aggiuntive su UDP nel livello di applicazione (ad esempio, HTTP/3)