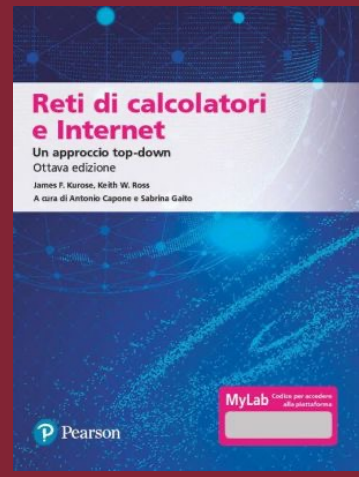
Reti di Elaboratori

Livello di Trasporto: UDP



SAPIENZA
UNIVERSITÀ DI ROMA

Alessandro Checco
alessandro.checco@uniroma1.it

Capitolo 3

Livello di trasporto: sommario

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

UDP: protocollo datagramma utente

- Protocollo di trasporto Internet "senza fronzoli", "bare bone".
- servizio "best effort", i segmenti UDP possono essere:
  - persi
  - consegnati non ordinati all'app
- *senza connessione:*
  - nessun handshaking tra mittente e destinatario UDP
  - ogni segmento UDP gestito indipendentemente dagli altri

Perché UDP

- nessuna connessione (che può aggiungere ritardo RTT)
- semplice: nessuno stato di connessione da gestire
- piccola dimensione dell'intestazione
- nessun controllo della congestione
  - UDP può tentare la trasmissione senza limiti di velocità
  - può funzionare anche quando c'è congestione di rete!

UDP: protocollo datagramma utente

- Uso dell'UDP:
  - app multimediali in streaming (loss tolerant, rate sensitive)
  - DNS
  - SNMP (device management)
  - HTTP/3
- se è necessario un trasferimento affidabile su UDP (ad es. HTTP/3):
  - aggiungere l'affidabilità necessaria a livello di applicazione
  - aggiungere il controllo della congestione a livello di applicazione

UDP: protocollo datagramma utente [RFC 768]

INTERNET STANDARD

RFC 768

J. Postel
ISI
28 August 1980

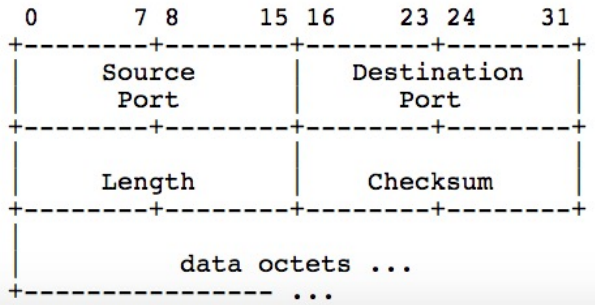
User Datagram Protocol

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.

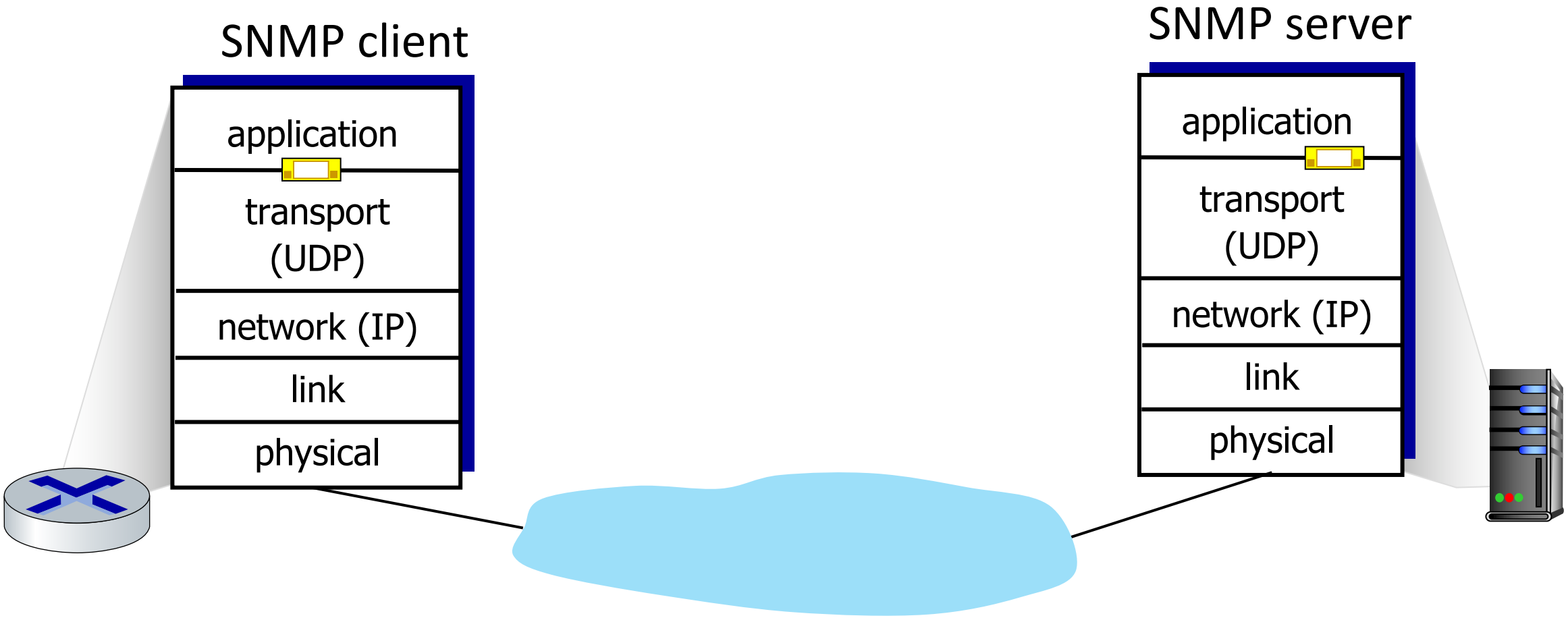
This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should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

Format

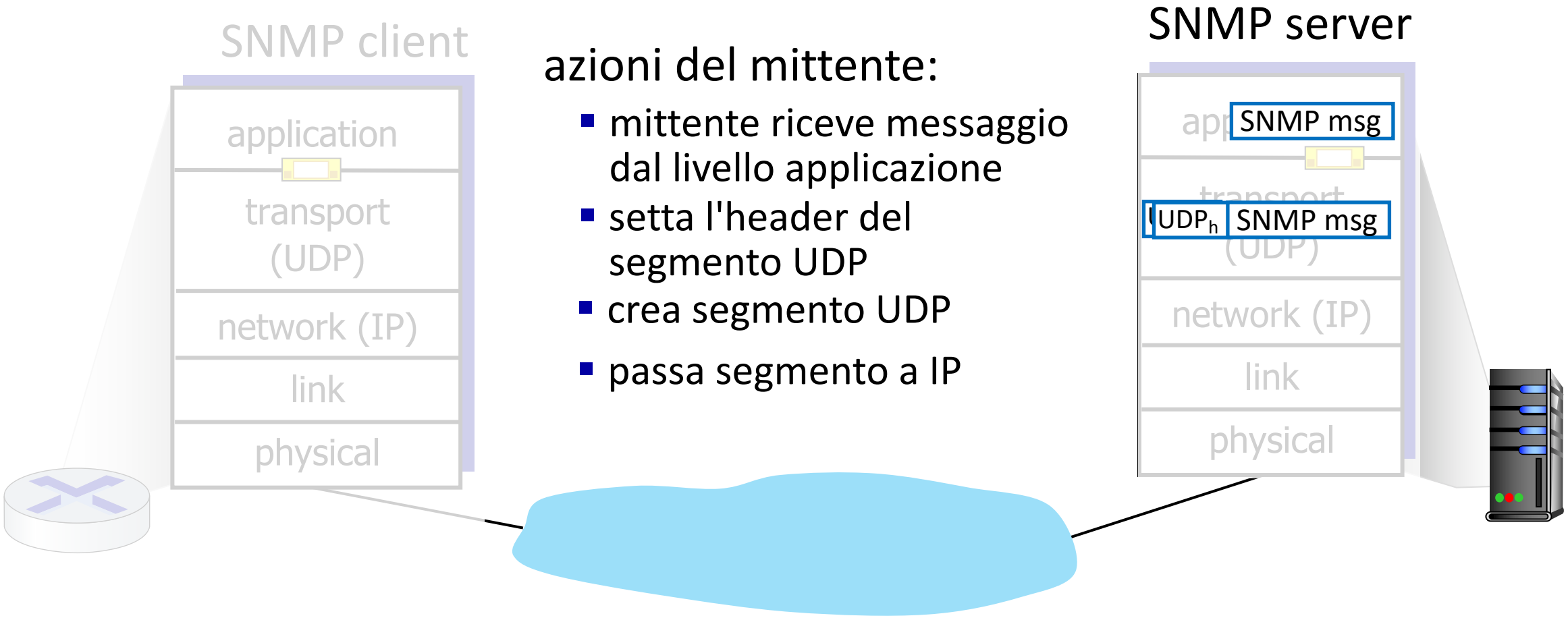


DA LEGGERE!

UDP: azioni del livello di trasporto



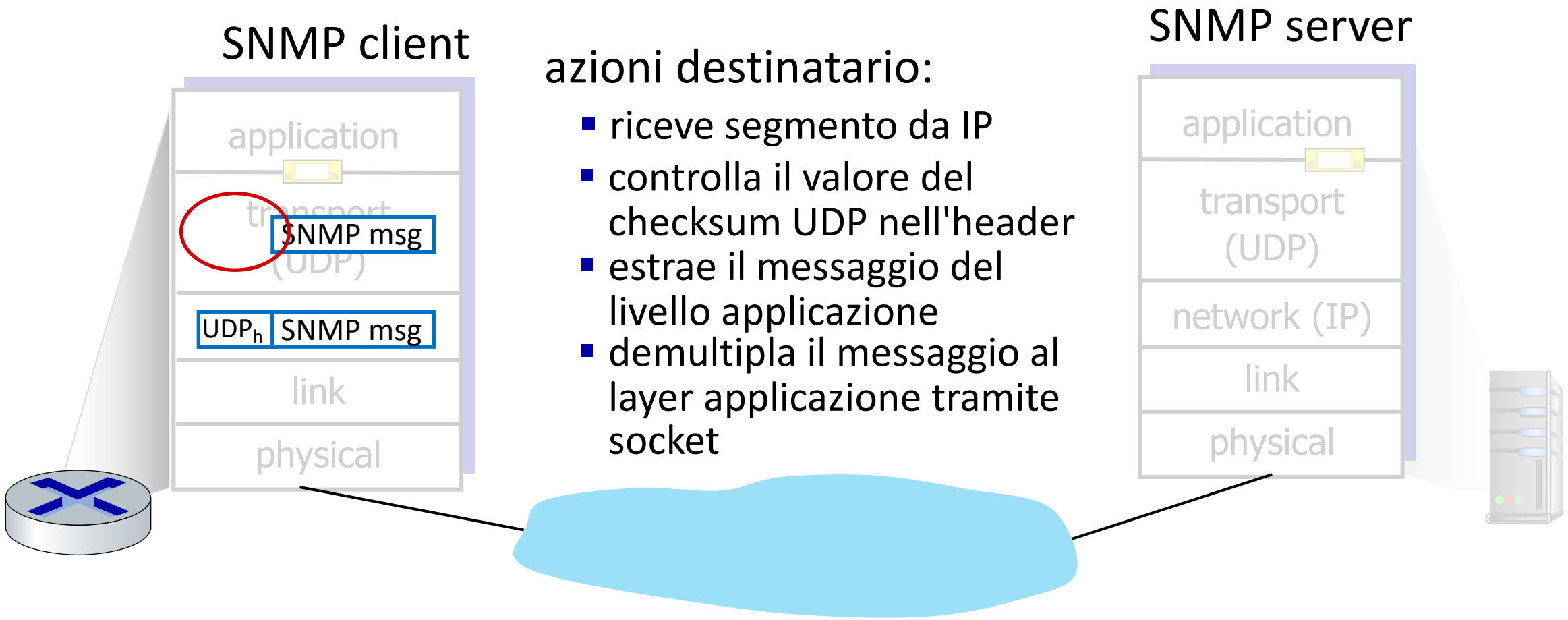
UDP: azioni del livello di trasporto



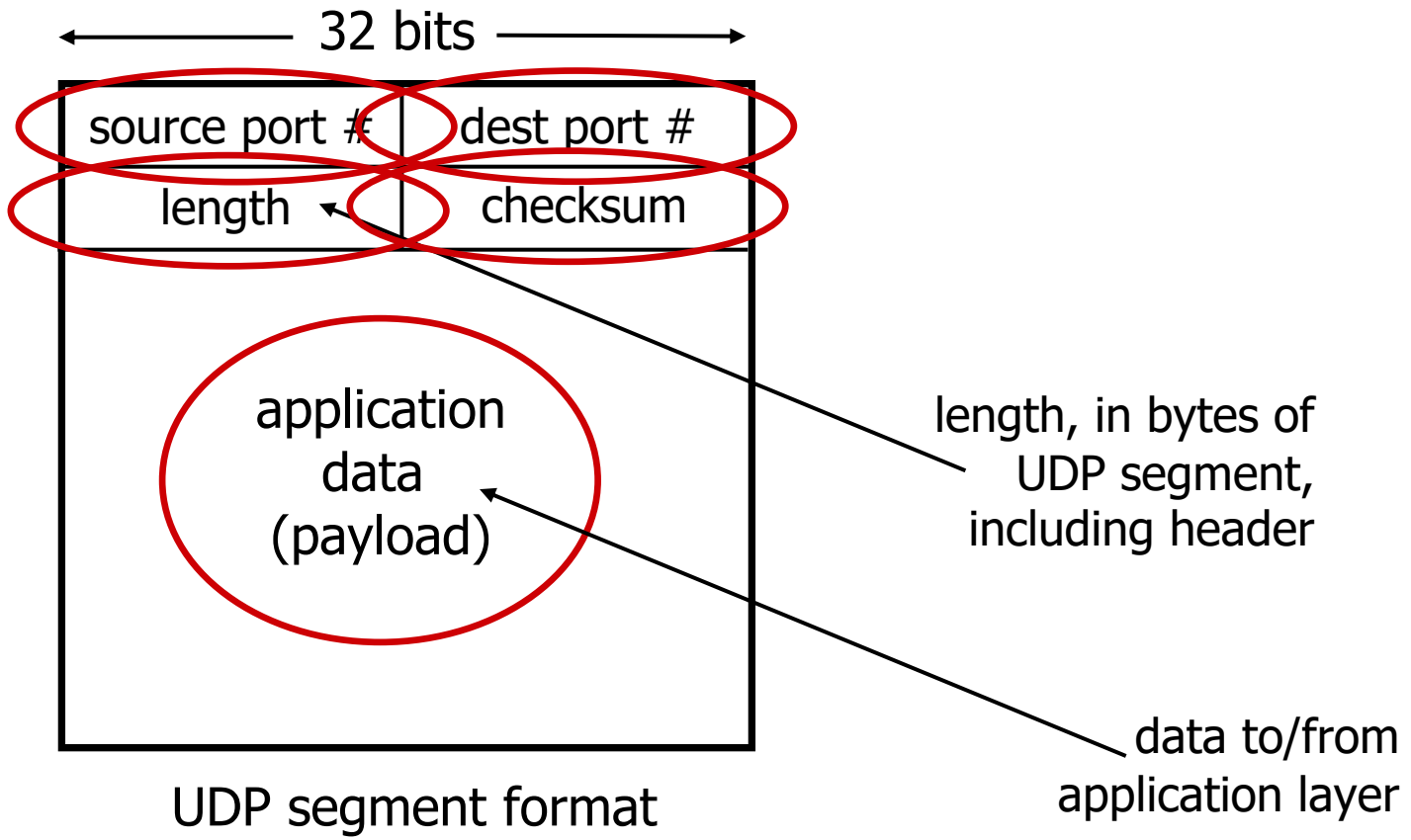
azioni del mittente:

- mittente riceve messaggio dal livello applicazione
- setta l'header del segmento UDP
- crea segmento UDP
- passa segmento a IP

UDP: Transport Layer Actions

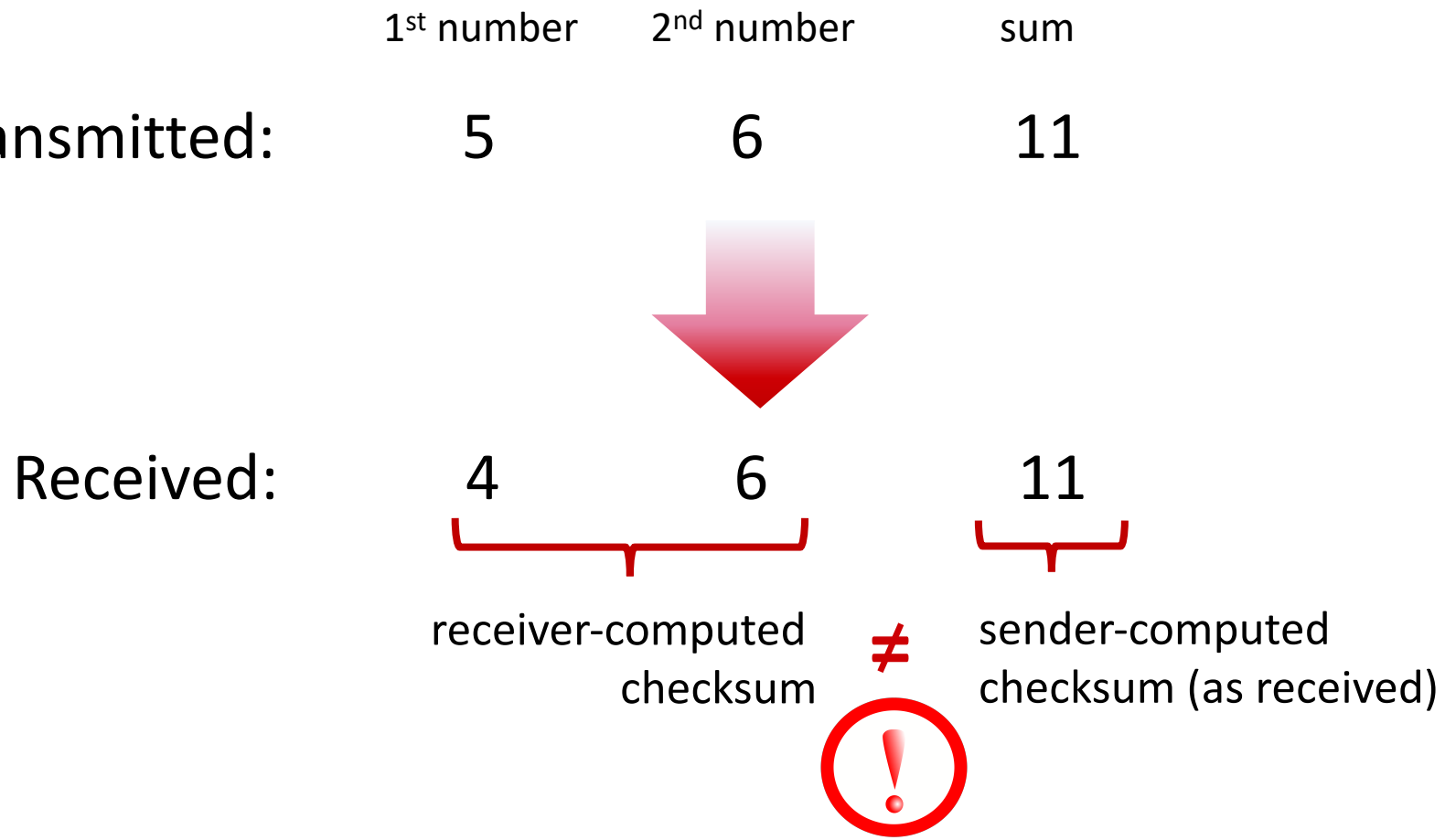


Intestazione del segmento UDP



checksum di UDP

Obiettivo: rilevare gli errori (ad *es.* i flipped bit) nel segmento trasmesso



Internet checksum

Obiettivo: rilevare gli errori (ad *es.* i flipped bit) nel segmento trasmesso

mittente:

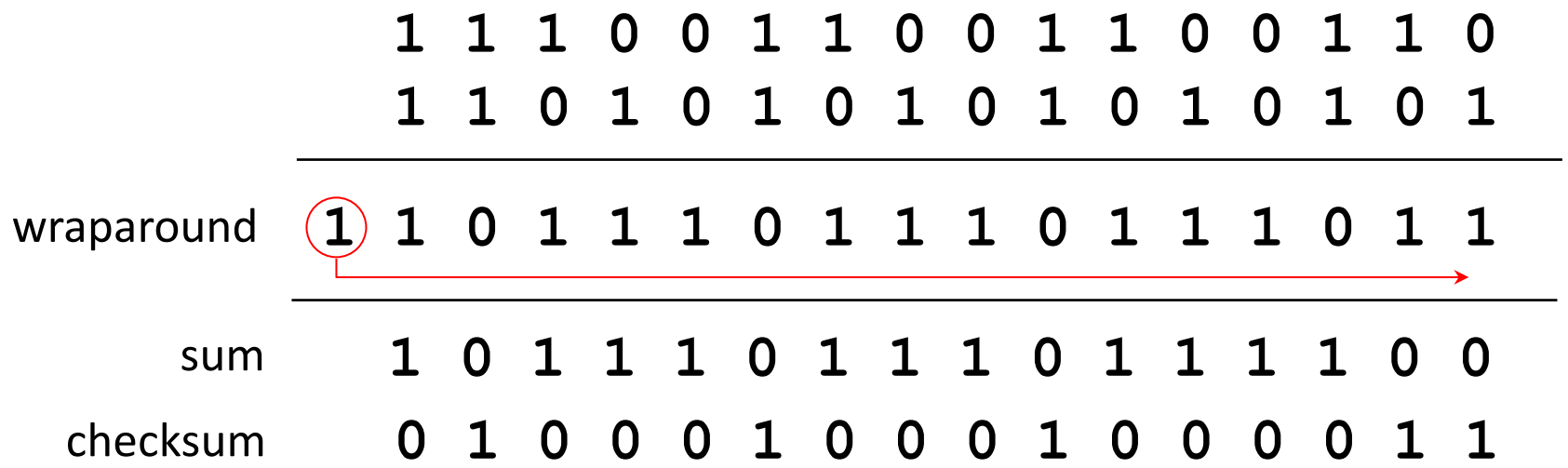
- considera i contenuti del segmento UDP (compresi i campi di intestazione UDP e gli indirizzi IP) come una sequenza di numeri interi a 16 bit
- checksum: addizione (somma in complemento a uno) del contenuto del segmento
- valore di checksum inserito nel campo checksum UDP

destinatario:

- calcola il checksum del segmento ricevuto
- controlla se il checksum calcolato è uguale al valore del campo checksum:
  - non uguale - errore rilevato
  - uguale - nessun errore rilevato. *Ma possono esserci comunque degli errori? Lo vedremo più avanti….*

Checksum Internet: un esempio

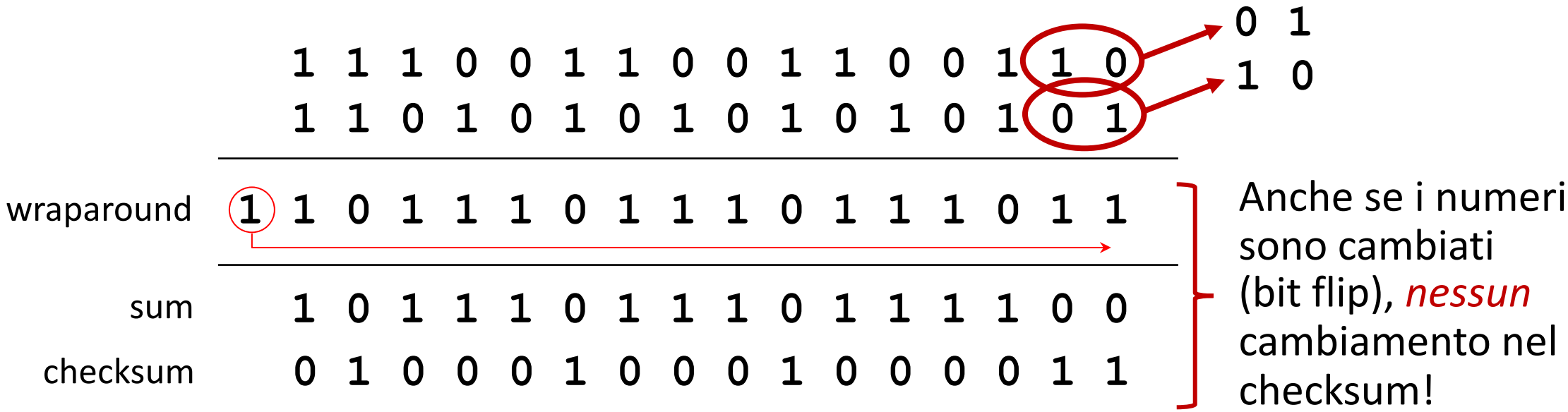
esempio: somma di due numeri interi a 16-bit



Nota: quando si aggiungono numeri, è necessario aggiungere al risultato il riporto dal bit più significativo in caso di overflow

Checksum Internet: protezione debole!

esempio: somma di due numeri interi a 16 bit



Somma 16 bit alla volta dell'header UDP (senza checksum) + pseudoheader IP + data -> complemento a 1 -> checksum

Se il risultato è 0 va invertito di nuovo (tutti 1) perché:
se non viene usato checksum = 0

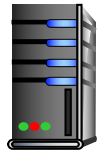
Sommario : UDP

- Protocollo “senza fronzoli”:
  - i segmenti possono andare persi, consegnati non ordinati
  - servizio best effort: “invia e spera per il meglio”
- UDP ha i suoi vantaggi:
  - nessuna configurazione/handshaking necessaria (nessun RTT sostenuto)
  - può funzionare quando il servizio di rete è compromesso
  - aiuta con l'affidabilità (checksum)
- si possono creare funzionalità aggiuntive su UDP a livello di applicazione (ad es. HTTP/3)

Livello di applicazione: sommario

- Principi delle applicazioni di rete
- Web e HTTP
- Posta elettronica, SMTP, IMAP
- FTP
- Domain Name System: DNS
- Applicazioni P2P
- streaming video e content distribution networks
- programmazione socket con UDP e TCP

Interazione socket client/server: UDP



server (running on serverIP)

create socket, port= x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

client

create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)

Create datagram with server IP and
port=x; send datagram via
clientSocket

read datagram from
clientSocket

close
clientSocket

App di esempio: client UDP

Python UDPClient

include Python's socket library →
```
from socket import *
```
```
serverName = 'hostname'
serverPort = 12000
```
create UDP socket for server →
```
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
```
get user keyboard input →
```
message = input('Input lowercase sentence:')
```
attach server name, port to message; send into socket →
```
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
```
read reply characters from socket into string →
```
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
```
print out received string and close socket →
```
print modifiedMessage.decode()
clientSocket.close()
```

App di esempio: server UDP

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

create UDP socket →

bind socket to local port number 12000 →

loop forever →

Read from UDP socket into message, getting client's address (client IP and port) →

send upper case string back to this client →