



# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

### **Introduzione al corso**

**I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso**



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre , Alessio Botta



# Chi sono?

- Alessio Botta
  - Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione
  - Via Claudio, 21 – IV piano, stanza 4.18  
Tel. 081-768-3865  
E-mail: [a.botta@unina.it](mailto:a.botta@unina.it)  
Twitter: [@alessiobot](https://twitter.com/alessiobot)
  - Orario di ricevimento: lunedì ore 17.30-19.30 (o su appuntamento)
- Ing. Stefania Zinno, Ing. Annalisa Navarro



# Orari delle lezioni

- Martedì dalle 16:30 alle 18:30
- Mercoledì dalle 8:30 alle 10:30
  
- E' attivo un canale teams che usiamo per le comunicazioni rapide e per la DAD in caso di necessità, iscrivetevi (link diretto più avanti)
- Verrà aperta l'iscrizione su docenti.unina.it che resta il sito di riferimento per tutte le comunicazioni ufficiali, iscrivetevi!



# Frequenza

- In presenza ☺
- On-line su **Microsoft Teams** solo
  - Se non ci sono più posti disponibili in aula (prenotazioni sulla piattaforma GO-IN)
  - Per gli studenti stranieri nelle more della formalizzazione dell'iscrizione o del rilascio del visto
- Canale MS Teams
  - Codice: g2u32na
  - Link: [https://teams.microsoft.com/l/team/19%3a1SA2Ndwbqh1NM-APbdHtOWf\\_9wHG63kJu7ejLhVCMFo1%40thread.tacv2/conversations?groupId=ba7be823-b599-4626-9a88-211526b5e660&tenantId=2fcfe26a-bb62-46b0-b1e3-28f9da0c45fd](https://teams.microsoft.com/l/team/19%3a1SA2Ndwbqh1NM-APbdHtOWf_9wHG63kJu7ejLhVCMFo1%40thread.tacv2/conversations?groupId=ba7be823-b599-4626-9a88-211526b5e660&tenantId=2fcfe26a-bb62-46b0-b1e3-28f9da0c45fd)



# Cosa Vedremo

- Principi alla base del funzionamento di una Rete di Calcolatori
- Le applicazioni ed I protocolli di Internet
- Protocolli applicativi: HTTP, DNS, SMTP ...
- I protocolli di livello trasporto: TCP e UDP
- Il livello di rete ed il protocollo IP: indirizzamento, routing ...
- IPv6
- IP Multicasting
- Programmazione di applicazioni comunicanti attraverso TCP/IP: la socket API
- Le reti locali cablate
- Le reti locali wireless
- Applicativi per la cattura e l'analisi del traffico di rete (e.g. Wireshark)



# Cosa NON Vedremo (1/6)

- Aspetti avanzati delle reti di calcolatori e dei servizi di rete
- La qualità del servizio nelle reti IP
- Tecniche di scheduling
- Architetture di rete a QoS
- Tecniche di flow-control e Tecniche di error control
- Algoritmi e protocolli di routing
- ATM
- MPLS
- Ingegneria delle reti: network design
- Il network management e SNMP
- Service Level Agreement e Service Level Specification
- Il problema della sicurezza
- Firewall e protezioni
- Tipologie di intrusione



# Cosa NON Vedremo (2/6)

- Evoluzione delle applicazioni web-based
- Web Caching
- CDN
- Service Oriented Architectures (SOA) ed i Java Web Services
- Applicazioni multimediali distribuite (video ed audio streaming, le applicazioni di videoconferenza e le applicazioni di telefonia su IP (VoIP))
  - Protocolli a supporto dello streaming di flussi audio/video. Il protocollo RTP  
Il protocollo RTSP per il controllo di sessioni
  - Session Initiation Protocol (SIP) e Session Description Protocol (SDP)
  - La API di JMF
- Modelli per la fornitura di servizi in reti di telecomunicazione all-IP di prossima generazione
- Architettura delle reti IMS
- Applicazioni basate su nuovi modelli architetturali, alternativi al client/server, in particolare le applicazioni peer-to-peer



# Cosa NON Vedremo (3/6)

- - Standard 802.11
  - + + Livello MAC: DCF e PCF
  - + + Sicurezza
  - + + Qualità del servizio
- Reti Ad Hoc
  - + + Protocolli di routing (AODV e OLSR)
  - + + Metriche di routing
- Reti wireless mesh
  - + + selezione delle frequenze, instradamento
- Broadband wireless: WiMax
- Mobilità: Mobile IP
- Mobilità e multihoming: HIP
- Strumenti e metodi per la simulazione di reti mobili



# Cosa NON Vedremo (4/6)

- Sicurezza di rete
  - in reti wireless
  - a livello rete: protocollo IPsec.
  - a livello trasporto: Secure Socket Layer (SSL).
  - al livello applicativo: posta elettronica; Web (HTTPS, WebRTC Security Architecture).
- Cloud Computing e sicurezza
- Software malevolo: Tassonomia, Advanced Persistent Threats (APTs), Contromisure
- Attacchi
- Intrusion Detection Systems (IDS)
- Firewall e Intrusion Prevention Systems (IPS)
- Hacking in reti IP



# Cosa NON Vedremo (5/6)

- Risoluzione dei Problemi di Rete, Requisiti di Prestazioni delle Reti e delle Applicazioni in Rete
- Analisi ed il Monitoraggio di Internet
  - Del traffico, della banda, dei percorsi e delle topologie etc.
- Miglioramento delle Prestazioni di Internet
  - Varianti di TCP, SCTP, DCCP, SDN, Middlebox, Load balancing
- Analisi e Monitoraggio di DNS, Web, P2P, Streaming, etc.
- Monitoraggio di Security e Outages
- Neutralità di Rete e Censura
- Monitoraggio e Prestazioni di SDN, Cloud e Data Center



# Cosa NON Vedremo (6/6)

- Architettura dei datacenter
- Tecnologie di networking nei datacenter
- Aspetti di networking tipici del cloud computing
- In Amazon AWS, OpenStack, ecc...
- Virtualizzazione di rete
- Evoluzione dei paradigmi di rete (Software Defined Networking, Network Function Virtualization, ...)



# Ma vederemo, qualsiasi altra cosa...

- ... vi va di fare su argomenti legati alle RdC
    - Su base volontaria
    - Di tipo pratico
    - In autonomia
    - Su argomenti indicati dal (concordati col) docente o proposti da voi ma accettati dal docente
- ... è più che benvenuta !!!



# Argomenti per Tirocinio e Tesi

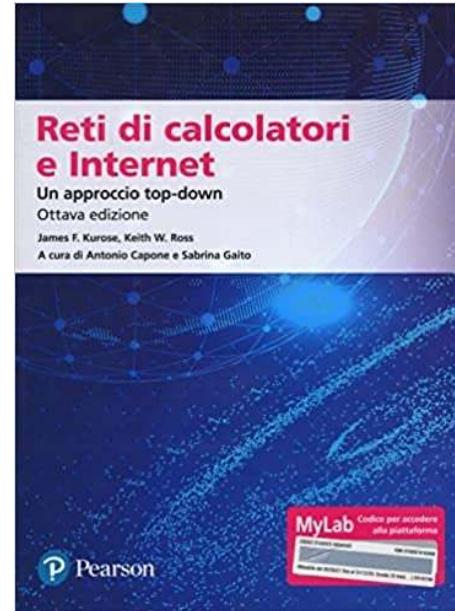
- In generale:
  - Tutto ciò che avete visto in RdC1 ed in generale tutte le tematiche che afferiscono al settore del Networking
- In particolare
  - Monitoraggio di Internet
  - Software Defined Networks (SDN)
  - Reti di infrastrutture Cloud
  - Neutralità della rete
  - Rilevazione di anomalie di rete
  - ...
- Poi ci sono quelli esterni...



# Materiale didattico

- Libro di testo
  - J. Kurose, K. Ross  
“Reti di calcolatori e internet - Un approccio top-down”,  
8a Edizione, Pearson (2022),  
ISBN: 9788891902542
- Altri libri consigliati per approfondimenti

- O. Bonaventure  
“Computer Networking : Principles, Protocols and Practice”  
<https://github.com/obonaventure/cnp3>
- Larry Peterson, Bruce Davie.  
“Reti di calcolatori” (terza edizione)  
Apogeo, 2013  
ISBN: 978-8838786396
- B. Krishnamurthy, J. Rexford.  
“Web Protocols and Practice: HTTP/1.1, Networking  
Protocols, Caching, and Traffic Measurement”.  
Addison-Wesley, 2001  
ISBN: 978-0201710885





# Altro materiale didattico

- Il mio corso di reti su federica.eu  
(<https://lms.federica.eu/enrol/index.php?id=248>)
- Le slide sul sito docenti, che sono più dettagliate di quelle su federica.eu e che verranno rilasciate in tempo utile



# Modalità di esame

Elaborati da fare durante il corso

- Partecipazione alle discussioni
- Homework

**OPPURE**

- Test al calcolatore
- Prova orale

**OPPURE**

- Elaborato finale, su base volontaria, non obbligatorio
- Di tipo pratico
- In autonomia
- Su argomenti indicati dal (concordati col) docente



# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

### **Protocolli applicativi**

**I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso**

# Nota di copyright per le slide COMICS



## Nota di Copyright

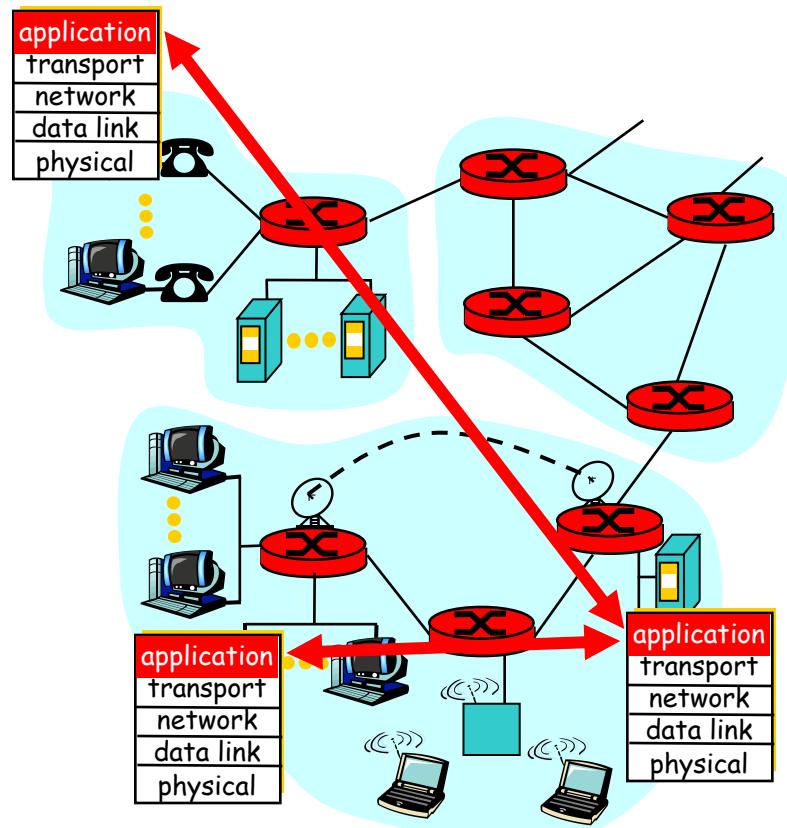
Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta



# Applicazioni e protocolli dello strato applicativo





# Strato dell'Applicazione

- Comunicazioni tra processi
- Cosa definisce un protocollo dello strato di applicazione
  - tipo di messaggi scambiati
  - sintassi dei messaggi
  - semantica dei campi
  - regole di trasmissione
- Applicazioni lato client e lato server (Es.: servizi *Web*, *FTP*)



# Comunicazione tra processi

- API: Application Programming Interface
  - definisce l'interfaccia tra il livello applicativo e il livello trasporto
  - socket: Internet API
    - due processi comunicano mandando dati in un socket, e leggendo dati dal socket
- Come un processo può “identificare” l’altro processo con cui vuole comunicare?
  - indirizzo IP dell’host che esegue l’altro processo
  - “numero di porta” che permette all’host ricevente di sapere a quale processo locale il messaggio è destinato



# Livello Trasporto: cenni

- TCP (Transmission Control Protocol):
  - connection oriented
  - trasferimento affidabile
  - congestion control
  - nessuna garanzia su velocità di trasmissione e ritardo
- UDP (User Datagram Protocol):
  - connection less
  - trasferimento non affidabile
  - no congestion control
  - garanzie sul rispetto di una velocità minima (processi real-time)



# Es.: Requisiti di alcune applicazioni

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Mb-1Mb video: 10Mb-5Mb	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few Kbps up	yes, 100's msec
Instant messaging	no loss	elastic	yes and no

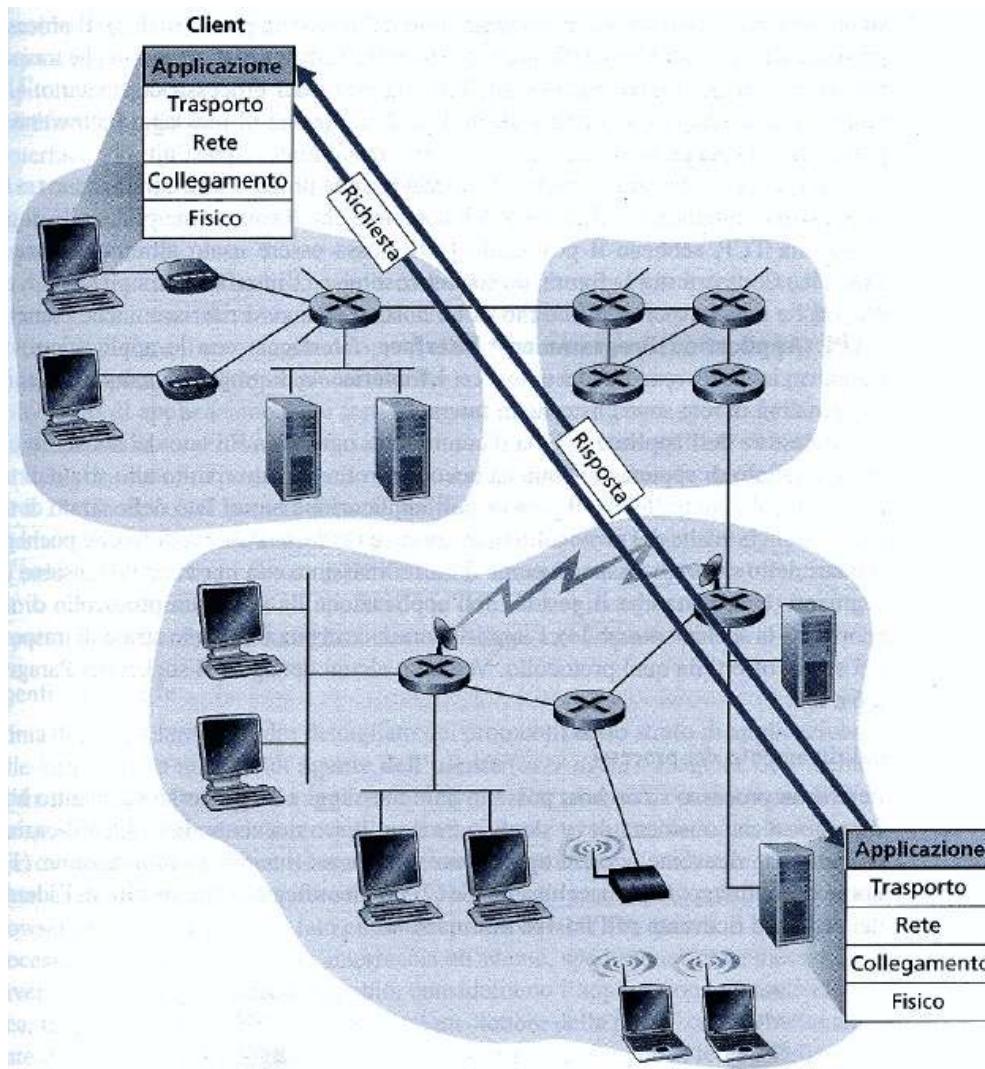


# Es.: applicazioni e protocolli di trasporto

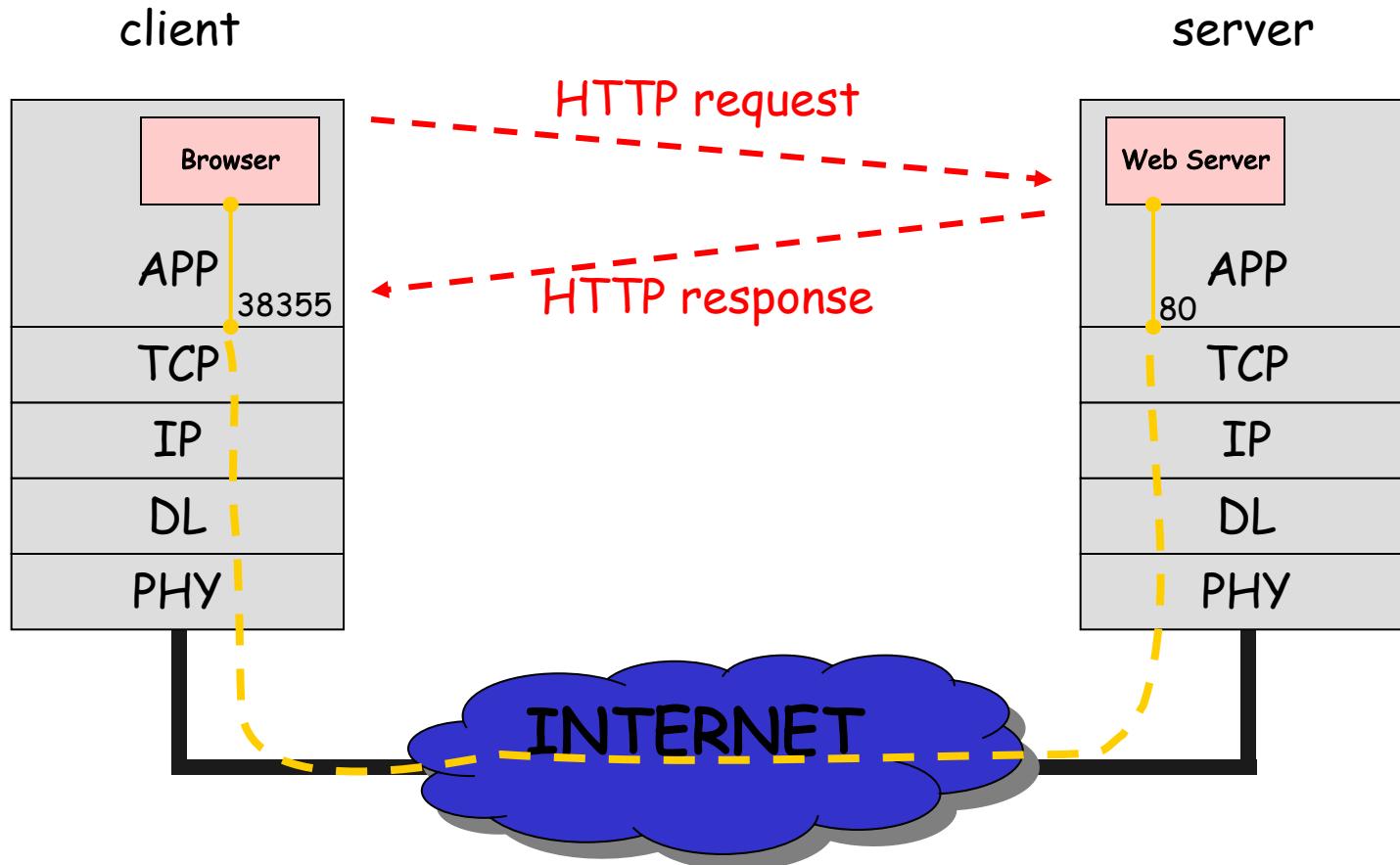
Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068,2616]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimediale	spesso proprietario (e.g. RealNetworks)	TCP or UDP
remote file server	NFS	TCP or UDP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP



# Paradigma Client/Server



# Web: esempio di interazione Client-Server



# Chapter 2

# Application Layer

A note on the use of these PowerPoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part.

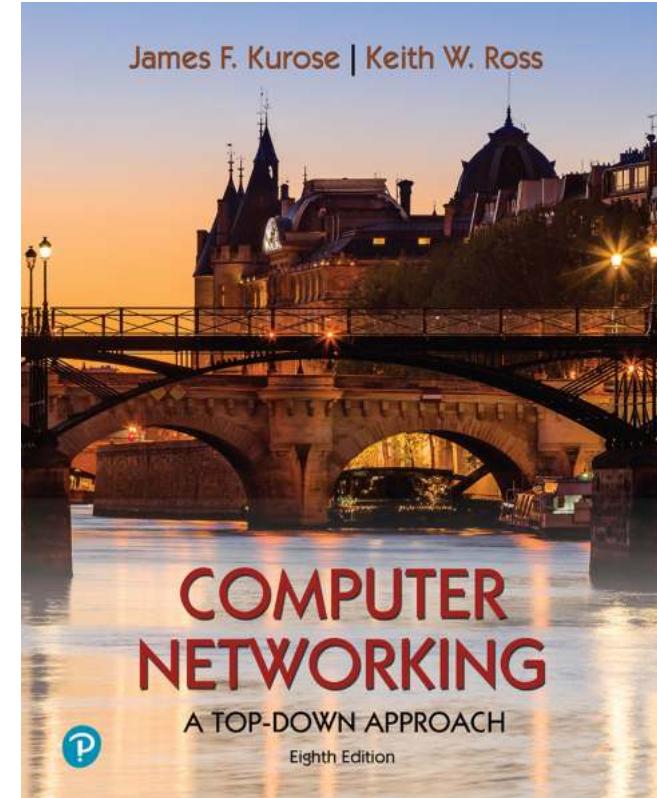
In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2020  
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer Networking: A  
Top-Down Approach*  
8<sup>th</sup> edition n  
Jim Kurose, Keith Ross  
Pearson, 2020

# Application Layer: Overview

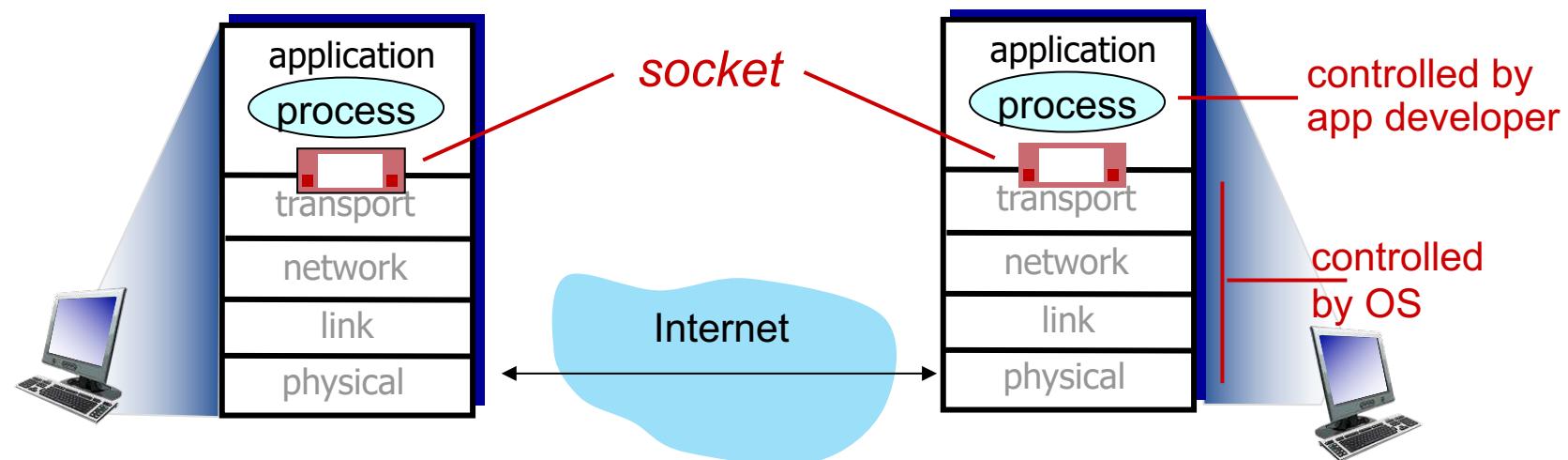
- Principles of network applications
- Web and HTTP
- E-mail, SMTP, IMAP
- The Domain Name System DNS
- P2P applications
- video streaming and content distribution networks
- **socket programming with UDP and TCP**



# Socket programming

*goal:* learn how to build client/server applications that communicate using sockets

*socket:* door between application process and end-end-transport protocol



# Socket programming

Two socket types for two transport services:

- *UDP*: unreliable datagram
- *TCP*: reliable, byte stream-oriented

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

# Socket programming with UDP

**UDP:** no “connection” between client and server:

- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

**UDP:** transmitted data may be lost or received out-of-order

**Application viewpoint:**

- UDP provides *unreliable* transfer of groups of bytes (“datagrams”) between client and server processes

# Client/server socket interaction: 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 serverIP address  
And port=x; send datagram via  
clientSocket
```

```
read datagram from  
clientSocket  
close  
clientSocket
```

# Example app: UDP client

## *Python UDPCClient*

```
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 = raw_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()
```

# Example app: UDP server

## *Python UDPServer*

```
from socket import *
serverPort = 12000
create UDP socket → serverSocket = socket(AF_INET, SOCK_DGRAM)
bind socket to local port number 12000 → serverSocket.bind(("", serverPort))
                                         print ("The server is ready to receive")
loop forever → while True:
Read from UDP socket into message, getting →   message, clientAddress = serverSocket.recvfrom(2048)
client's address (client IP and port)           modifiedMessage = message.decode().upper()
                                               serverSocket.sendto(modifiedMessage.encode(),
send upper case string back to this client →   clientAddress)
```

# Socket programming with TCP

## Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

## Client contacts server by:

- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket*: client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - *source port numbers used to distinguish clients* (more in Chap 3)

## Application viewpoint

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server processes

# Client/server socket interaction: TCP



server (running on hostid)



client

create socket,  
port=x, for incoming  
request:  
`serverSocket = socket()`

wait for incoming  
connection request  
`connectionSocket = serverSocket.accept()`

read request from  
`connectionSocket`

write reply to  
`connectionSocket`

close  
`connectionSocket`

create socket,  
connect to `hostid`, port=x  
`clientSocket = socket()`

send request using  
`clientSocket`

read reply from  
`clientSocket`

close  
`clientSocket`

TCP  
connection setup

# Example app: TCP client

## *Python TCPClient*

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server,  
remote port 12000 → clientSocket = socket(AF\_INET, SOCK\_STREAM)

No need to attach server name, port → clientSocket.connect((serverName, serverPort))

# Example app: TCP server

## *Python TCPServer*

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(("",serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                           encode())
connectionSocket.close()
```

create TCP welcoming socket → from socket import \*

server begins listening for incoming TCP requests → serverPort = 12000  
→ serverSocket = socket(AF\_INET,SOCK\_STREAM)  
→ serverSocket.bind(("",serverPort))  
→ serverSocket.listen(1)

loop forever → print 'The server is ready to receive'  
→ while True:

server waits on accept() for incoming requests, new socket created on return → connectionSocket, addr = serverSocket.accept()

read bytes from socket (but not address as in UDP) → sentence = connectionSocket.recv(1024).decode()  
→ capitalizedSentence = sentence.upper()  
→ connectionSocket.send(capitalizedSentence.  
 encode())

close connection to this client (but *not* welcoming socket) → connectionSocket.close()



# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

**HTTP**

**I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso**

# Nota di copyright per le slide COMICS



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta



# Il protocollo HTTP/1.0

- Si basa su TCP
- Il client apre un socket verso il porto TCP 80 del server (se non diversamente specificato)
- Il server accetta la connessione
- Il client manda una richiesta per uno specifico oggetto identificato mediante una URL
- Il server risponde e chiude la connessione
- **Il protocollo HTTP è stateless: né il server né il client mantengono a livello HTTP informazioni relative ai messaggi precedentemente scambiati**



# URL : Uniform Resource Locator

- Un URL HTTP ha la seguente sintassi (RFC 2396) :  
**`http://host[:port]/path[#fragment] [?query]`**
- Host identifica il server
  - Può essere sia un nome simbolico che un indirizzo IP in notazione dotted decimal
- Port è opzionale; di default è 80
- Path identifica la risorsa sul server
  - es: images/sfondo.gif
- #fragment identifica un punto preciso all'interno di un oggetto
  - es: #paragrafo1
- ?query è usato per passare informazioni dal client al server
  - es: dati inseriti nei campi di una form

Es.: **`http://www.unina.it/immatricolazioni.htm#ingegneria`**



# HTTP: versioni

- Il protocollo è definito in documenti RFC
- Negli anni si sono avute tre versioni
  - HTTP 0.9
  - HTTP 1.0 (1996)
    - RFC 1945
  - HTTP 1.1 (1997 e 1999)
    - RFC 2068
    - RFC 2616
  - HTTP 2.0 (pubblicato 2015, originato da SPDY di Google)
    - RFC 7540

NOTA: un RFC (Request For Comment) è un documento pubblico sottoposto alla comunità Internet al fine di essere valutato. Ciò che è un RFC rappresenta uno standard *de facto* nella comunità Internet. Tutti gli RFC possono essere reperiti al sito dell'Internet Engineering Task Force (<http://www.ietf.org>)

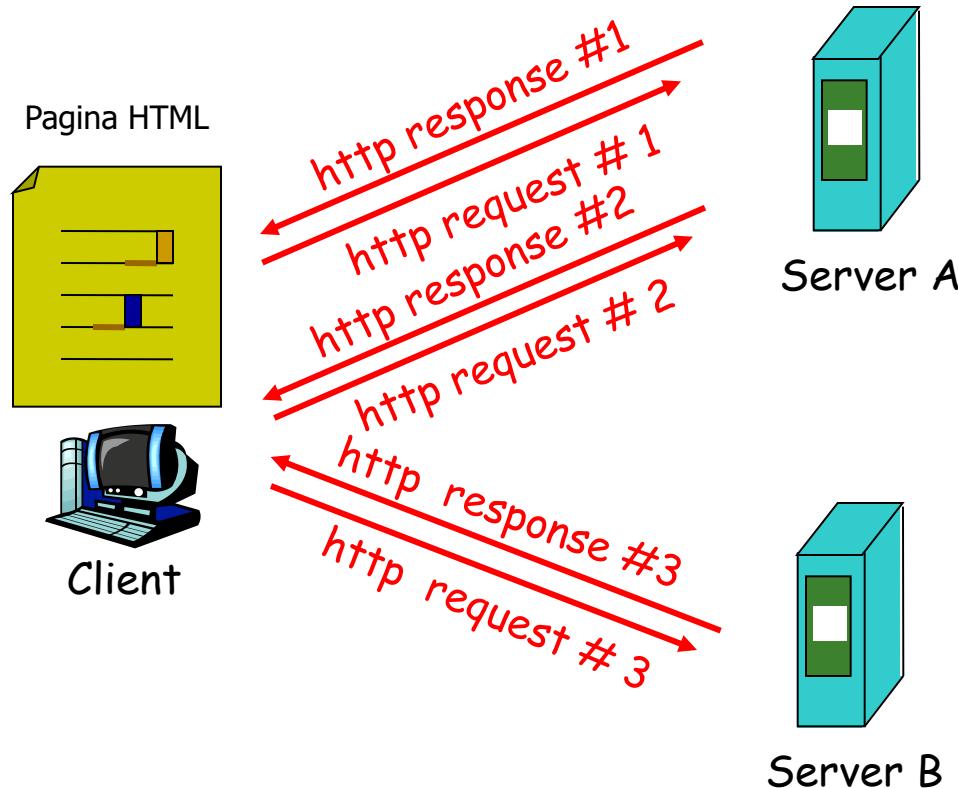


# HTTP per il trasferimento di pagine web

- Tipicamente, una pagina web è descritta da un file testuale in formato HTML (*Hypertext Markup Language*)
- La pagina è identificata mediante un indirizzo, detto URL
- Un file HTML può contenere riferimenti ad altri oggetti che arricchiscono la pagina con elementi grafici
  - Es. sfondo, immagini, ecc.
- Ciascun oggetto è identificato dal proprio URL
- Questi oggetti possono trovarsi anche su server web diversi
- Una volta ricevuta la pagina HTML, il browser estrae i riferimenti agli altri oggetti che devono essere prelevati e li richiede attraverso una serie di connessioni HTTP

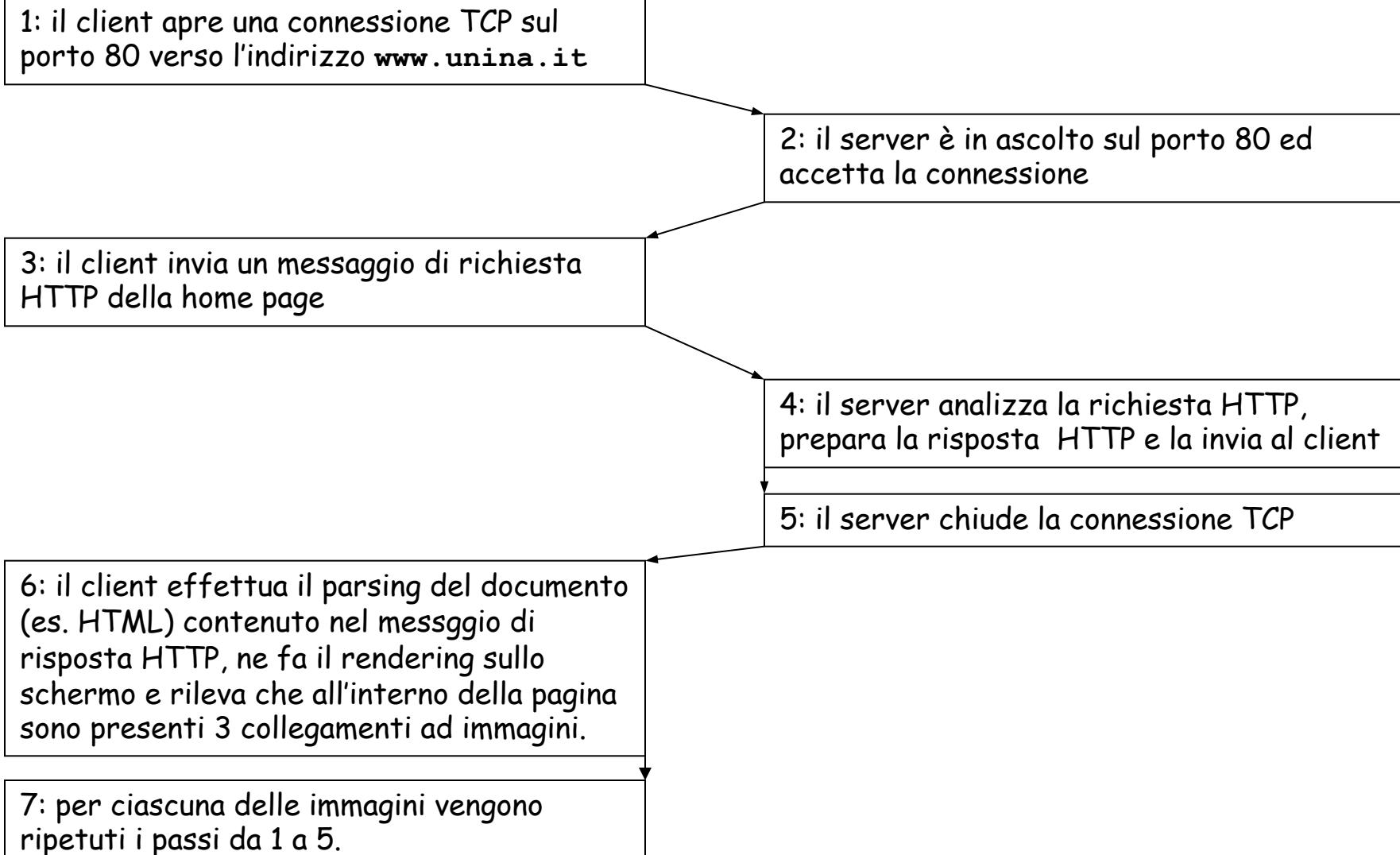


# HTTP per il trasferimento di pagine web (2)





# Es: richiesta di una pagina contenente immagini





# Connessioni persistenti e non persistenti

## non persistente

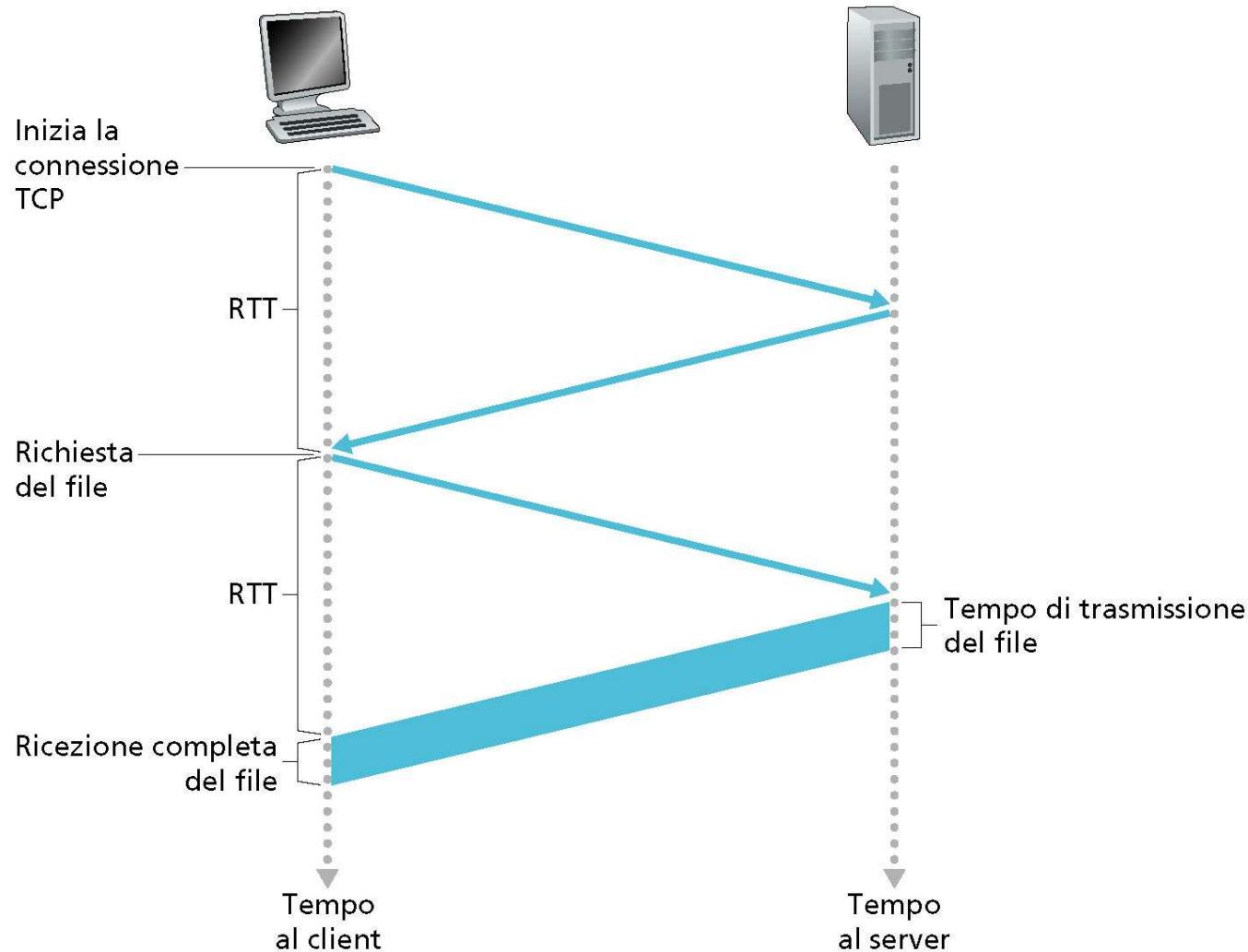
- HTTP/1.0 (RFC 1945)
- Il server analizza una richiesta, la serve e chiude la connessione
- 2 Round Trip Time (RTT) per ciascuna richiesta
- Ogni richiesta subisce lo slow-start TCP

## persistente

- HTTP/1.1 (RFC 2068, RFC 2116)
- Sulla stessa connessione il server analizza tutte le richieste e le serve
- Il client riceve la pagina iniziale e invia subito tutte le altre richieste
- Si hanno meno RTT ed un solo slow-start
- Esiste anche una versione con parallelismo (with pipelining)



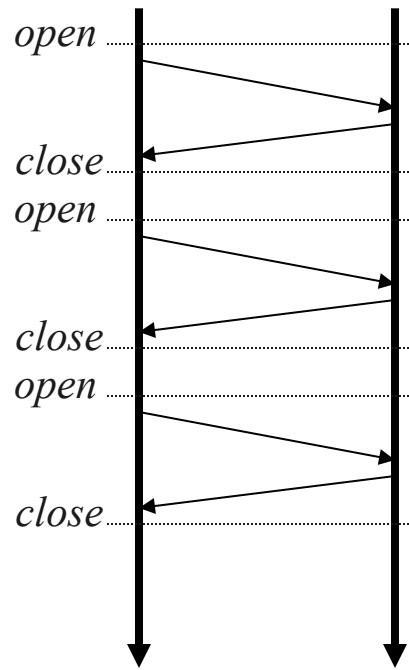
# Round Trip Time e connessioni HTTP





# La connessione HTTP

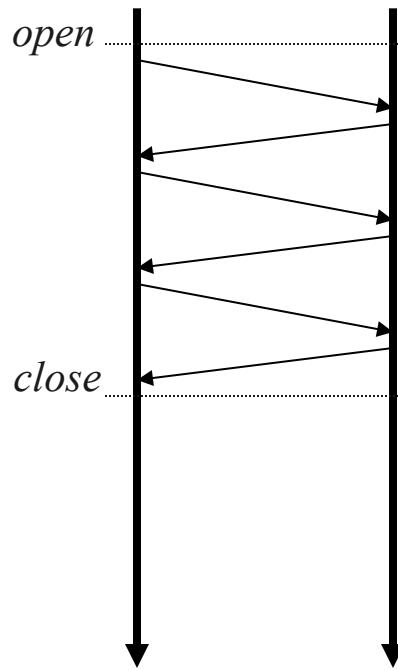
client      server



HTTP 1.0

Non persistente

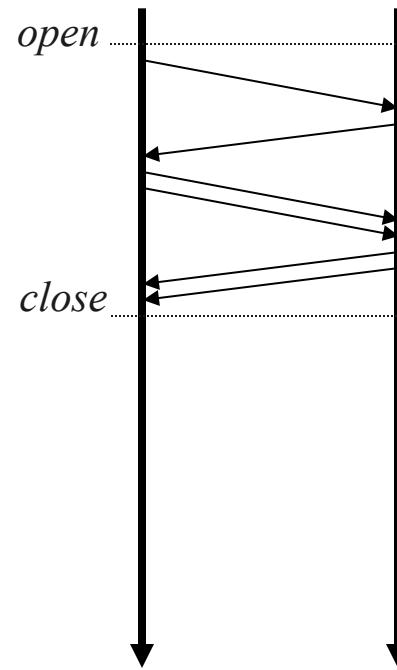
client      server



HTTP 1.1

Persistente

client      server

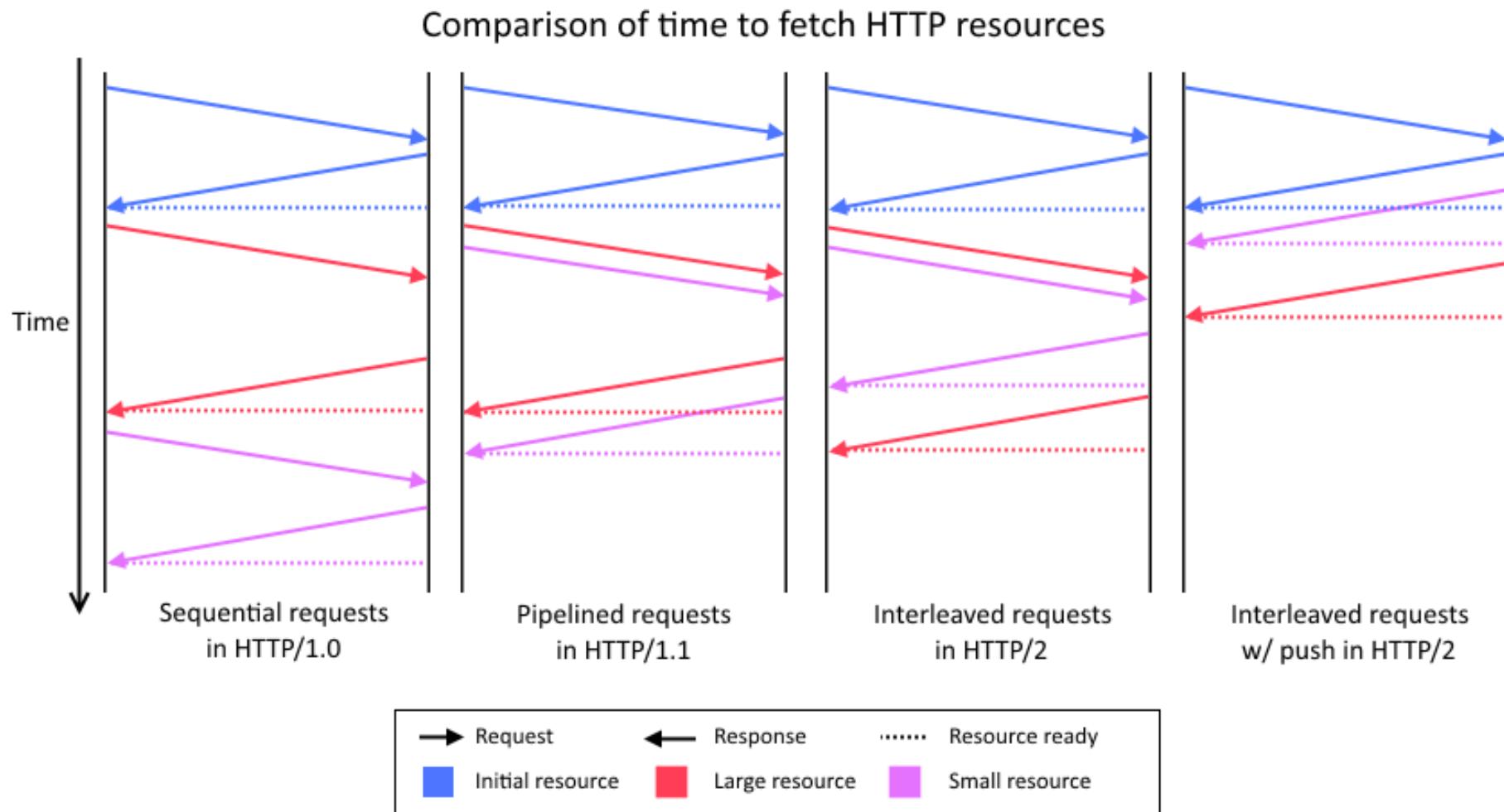


HTTP 1.1  
con pipelining

Persistente con  
pipelining



# La connessione HTTP (2/2)





# Il protocollo HTTP

- HTTP è un protocollo testuale
- I messaggi sono costituiti da sequenze di byte
- Ogni byte identifica un carattere secondo la tabella ASCII
- Il payload dei messaggi può essere comunque anche in formato binario (es. un'immagine GIF, un video, ecc.)



# Il messaggio HTTP/1.0 request: un esempio

Un esempio di messaggio GET

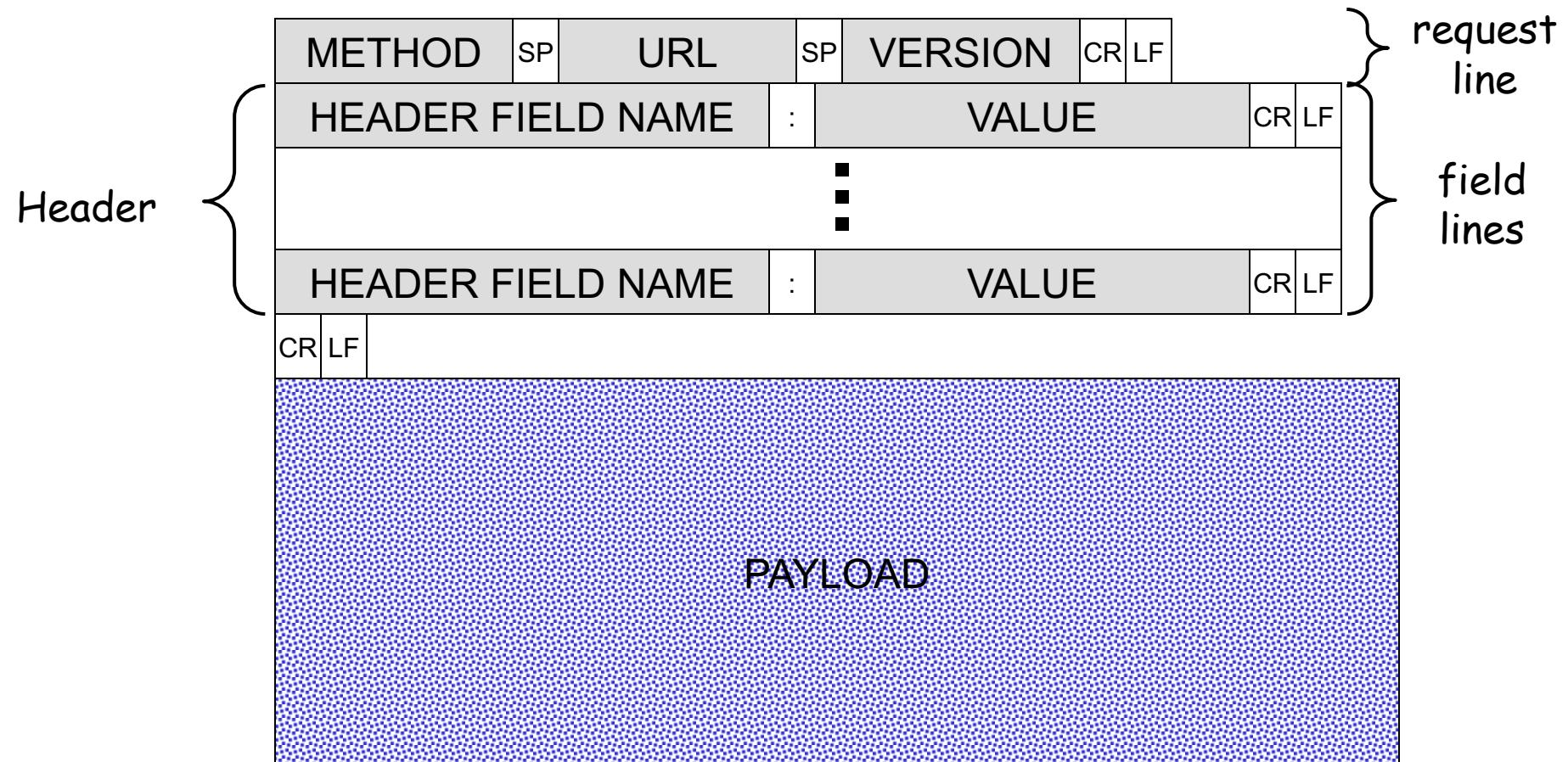
```
GET /path/pagename.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: it
```

RIGA VUOTA

indica la fine del messaggio



# Il messaggio HTTP/1.0 request





# Il messaggio HTTP/1.0 response

HTTP/1.0 200 OK	<i>codice di stato</i>
Date: Mon, 16 Dec 2002 14:00:22 GMT	
Server: Apache/1.3.24 (Win32)	
Last-Modified: Fri, 13 Dec 2002 08:06:44 GMT	
Content-Length: 222	
Content-Type: text/html	
RIGA VUOTA	
PAYLOAD	



# Esempi di codici di stato

## 200 OK

- Successo: l'oggetto richiesto si trova più avanti nel messaggio

## 301 Moved Permanently

- L'oggetto richiesto è stato spostato.  
Il nuovo indirizzo è specificato più avanti nel messaggio  
( Location: )

## 400 Bad Request

- Richiesta incomprensibile al server

## 404 Not Found

- Il documento non è presente sul server

## 505 HTTP Version Not Supported

- La versione del protocollo HTTP usata non è supportata dal server



# Lo scambio di messaggi

- Il metodo **GET**
- Il metodo **HEAD**
- Il metodo **POST**
- Il metodo **PUT**
- La *response*



# Il metodo GET

- Uno dei più importanti metodi di HTTP è **GET**
- Usato per richiedere una risorsa ad un server
- Questo è il metodo più frequente, ed è quello che viene attivato facendo click su un link ipertestuale di un documento HTML, o specificando un URL nell'apposito campo di un browser
- GET può essere:
  - **assoluto**
    - la risorsa viene richiesta senza altre specificazioni
  - **condizionale**
    - si richiede la risorsa se è soddisfatto un criterio indicato negli header **If-match**, **If-modified-since**, **If-range**, ecc.
  - **parziale**
    - si richiede una sottoparte di una risorsa memorizzata



# Il metodo GET: un esempio

(Untitled) - Ethereal

File Edit View Go Capture Analyze Statistics Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	85.182.14.100	143.225.92.106	TCP	4890 > 27047 [SYN] Seq=0 Len=0 MSS=1452
2	3.270549	143.225.229.163	209.85.129.147	TCP	4150 > http [SYN] Seq=0 Len=0 MSS=1460
3	3.304386	209.85.129.147	143.225.229.163	TCP	http > 4150 [SYN, ACK] Seq=0 Ack=1 win=8190 Len=0 MSS=1460
4	3.304443	143.225.229.163	209.85.129.147	TCP	4150 > http [ACK] Seq=1 Ack=1 win=65535 [TCP CHECKSUM INCORR]
5	3.368079	143.225.229.163	209.85.129.147	HTTP	GET / HTTP/1.1
6	3.402881	209.85.129.147	143.225.229.163	TCP	http > 4150 [ACK] Seq=1 Ack=574 win=6611 Len=0
7	3.415274	209.85.129.147	143.225.229.163	TCP	[TCP segment of a reassembled PDU]
8	3.415291	209.85.129.147	143.225.229.163	HTTP	HTTP/1.1 200 OK (text/html)
9	3.415314	143.225.229.163	209.85.129.147	TCP	4150 > http [ACK] Seq=574 Ack=2544 win=65535 [TCP CHECKSUM II]
10	6.038681	85.182.14.100	143.225.92.106	TCP	4890 > 27047 [SYN] Seq=0 Len=0 MSS=1452

Frame 5 (627 bytes on wire, 627 bytes captured)  
Ethernet II, Src: wistron\_2e:cb:5b (00:0a:e4:2e:cb:5b), Dst: Cisco\_6d:04:00 (00:14:f1:6d:04:00)  
Internet Protocol, Src: 143.225.229.163 (143.225.229.163), Dst: 209.85.129.147 (209.85.129.147)  
Transmission Control Protocol, Src Port: 4150 (4150), Dst Port: http (80), Seq: 1, Ack: 1, Len: 573  
Hypertext Transfer Protocol  
+ GET / HTTP/1.1\r\nHost: www.google.com\r\nUser-Agent: Mozilla/5.0 (Windows; U; windows NT 5.1; it-IT; rv:1.7.12) Gecko/20050919 Firefox/1.0.7\r\nAccept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5\r\nAccept-Language: it,it-it;q=0.8,en-us;q=0.5,en;q=0.3\r\nAccept-Encoding: gzip,deflate\r\nAccept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7\r\nKeep-Alive: 300\r\nConnection: keep-alive\r\nCookie: PREF=ID=42ee83292038ae07:FF=4:LD=en:NR=10:CR=2:TM=1169647218:LM=1187706763:GM=1:S=zCPXwUBZ4XF7AyNX; S=awfe=QDZp-Kks\r\n\r\n

0040 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 77 77 77 2e . ....GET / HTTP/  
0050 67 6f 6f 67 6c 65 2e 63 6f 6d 0d 0a 55 73 65 72 /1.1..Ho st: www.  
0060 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c 61 2f google.c om..User  
0070 35 2e 30 20 28 57 69 6e 64 6f 77 73 3b 20 55 3b Agent: Mozilla/  
0080 20 57 69 6e 64 6f 77 73 20 4e 54 20 35 2e 31 3b 5.0 (win dows; U;  
0090 20 69 74 2d 49 54 3b 20 72 76 3a 31 2e 37 2e 31 Windows NT 5.1;  
00a0 32 29 20 47 65 63 6b 6f 2f 32 30 30 35 30 39 31 it-IT; rv:1.7.1  
00b0 39 20 46 69 72 65 66 6f 78 2f 31 2e 30 2e 37 0d 2) Gecko /2005091  
00c0 0a 41 63 63 65 70 74 3a 20 74 65 78 74 7f 78 6d 9 Firefo x/1.0.7.  
.....Accept: text/xm

P: 10 D: 10 M: 0 Drops: 0



# Il metodo HEAD

- Simile al metodo GET, ma il server deve rispondere soltanto con gli header relativi, senza il corpo
- Usato per verificare:
  - **la validità di un URI**
    - la risorsa esiste e non è di lunghezza zero
  - **l'accessibilità di un URI**
    - la risorsa è accessibile presso il server, e non sono richieste procedure di autenticazione del documento
  - **la coerenza di cache di un URI**
    - la risorsa non è stata modificata nel frattempo, non ha cambiato lunghezza, valore hash o data di modifica



# Il metodo POST

- Il metodo **POST** serve per trasmettere delle informazioni dal client al server, ma senza la creazione di una nuova risorsa
- POST viene usato per esempio per sottomettere i dati di una form HTML ad un'applicazione sul server
- I dati vengono trasmessi nel body della richiesta
- Il server può rispondere positivamente in tre modi:
  - **200 Ok**: dati ricevuti e sottomessi alla risorsa specificata; è stata data risposta
  - **201 Created**: dati ricevuti, la risorsa non esisteva ed è stata creata
  - **204 No content**: dati ricevuti e sottomessi alla risorsa specificata; non è stata data risposta
- Non è l'unico modo per inviare dati

# Il metodo POST: un esempio

The screenshot shows the Ethereal interface with the following details:

- Filter:** http
- Protocol:** HTTP
- Info:** POST /email.php HTTP/1.1
- Source:** 192.168.1.2
- Destination:** 212.52.84.18
- Time:** 0.107593

The packet details pane shows the raw HTTP request:

```
+ Frame 6 (881 bytes on wire, 881 bytes captured)
+ Ethernet II, Src: Wistron_2e:cb:5b (00:0a:e4:2e:cb:5b), Dst: D-Link_1e:6a:d1 (00:17:9a:1e:6a:d1)
+ Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 212.52.84.18 (212.52.84.18)
+ Transmission Control Protocol, Src Port: 2839 (2839), Dst Port: http (80), Seq: 1, Ack: 1, Len: 827
+ Hypertext Transfer Protocol
+ POST /email.php HTTP/1.1\r\n
Host: wpop8.libero.it\r\n
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it-IT; rv:1.7.12) Gecko/20050919 Firefox/1.0.7\r\n
Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/plain;q=0.8, image/png,*/*;q=0.5\r\n
Accept-Language: it, it-it;q=0.8, en-us;q=0.5, en;q=0.3\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Charset: ISO-8859-1, utf-8;q=0.7, *;q=0.7\r\n
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
Referer: http://www.libero.it/\r\n
Cookie: __utmz=267072147.1190633699.2.3.utmcsr=HP%2BLiber|utmccn=Infostrada%20ADSL|utmcmd=Lancio1; Liberomail=2dccc1ab3e03
```

The bottom hex dump pane shows the raw bytes of the packet.



# Il metodo PUT

- Il metodo **PUT** serve per trasmettere delle informazioni dal client al server, creando o sostituendo la risorsa specificata
  - Esempio: upload di un file
- In generale, l'argomento del metodo PUT è la risorsa che ci si aspetta di ottenere facendo un GET in seguito con lo stesso nome



# HTTP: Response

- La risposta HTTP è un messaggio testuale formato da una riga iniziale, da header facoltativi ed eventualmente un body (corpo)

**Version status-code reason-phrase CRLF** } request  
[Header]   line  
[Header] } Header  
CRLF  
**Body**

dove:

[Header] indica un elemento opzionale

CRLF indica la sequenza di caratteri di codice ASCII

$0D_{16} = 13_{10}$  → CR = Carriage Return

$0A_{16} = 10_{10}$  → LF = Line Feed



# HTTP: Response (2)

- Un Esempio:

HTTP/1.1 200 OK

Date: Thu, 10 Apr 2003 11:46:53 GMT

Server: Apache/1.3.26 (Unix) PHP/4.0.3pl1

Last-Modified: Wed, 18 Dec 2002 12:55:37 GMT

Accept-Ranges: bytes

Content-Length: 7394

Content-Type: text/html

<HTML> ... </HTML>



# HTTP Response: un esempio

(Untitled) - Ethereal

File Edit View Go Capture Analyze Statistics Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	85.182.14.100	143.225.92.106	TCP	4890 > 27047 [SYN] Seq=0 Len=0 MSS=1452
2	3.270549	143.225.229.163	209.85.129.147	TCP	4150 > http [SYN] Seq=0 Len=0 MSS=1460
3	3.304386	209.85.129.147	143.225.229.163	TCP	http > 4150 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
4	3.304443	143.225.229.163	209.85.129.147	TCP	4150 > http [ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORR]
5	3.368079	143.225.229.163	209.85.129.147	HTTP	GET / HTTP/1.1
6	3.402881	209.85.129.147	143.225.229.163	TCP	http > 4150 [ACK] Seq=1 Ack=574 Win=6611 Len=0
7	3.415274	209.85.129.147	143.225.229.163	TCP	[TCP segment of a reassembled PDU]
8	3.415291	209.85.129.147	143.225.229.163	HTTP	HTTP/1.1 200 OK (text/html)
9	3.415314	143.225.229.163	209.85.129.147	TCP	4150 > http [ACK] Seq=574 Ack=2544 Win=65535 [TCP CHECKSUM II]
10	6.038681	85.182.14.100	143.225.92.106	TCP	4890 > 27047 [SYN] Seq=0 Len=0 MSS=1452

Frame 8 (1167 bytes on wire, 1167 bytes captured)  
 Ethernet II, Src: Cisco\_6d:04:00 (00:14:f1:6d:04:00), Dst: Wistron\_2e:cb:5b (00:0a:e4:2e:cb:5b)  
 Internet Protocol, Src: 209.85.129.147 (209.85.129.147), Dst: 143.225.229.163 (143.225.229.163)  
 Transmission Control Protocol, src Port: http (80), Dst Port: 4150 (4150), seq: 1431, Ack: 574, Len: 1113  
 [Reassembled TCP Segments (2543 bytes): #7(1430), #8(1113)]  
 Hypertext Transfer Protocol  
 HTTP/1.1 200 OK\r\n  
 Cache-Control: private\r\n  
 Content-Type: text/html; charset=UTF-8\r\n  
 Content-Encoding: gzip\r\n  
 Server: gws\r\n  
 Content-Length: 2364\r\n  
 Date: Mon, 24 Sep 2007 10:25:21 GMT\r\n  
 \r\n  
 Content-encoded entity body (gzip): 2364 bytes -> 5472 bytes  
 Line-based text data: text/html

0000	00 0a e4 2e cb 5b 00 14 f1 6d 04 00 08 00 45 00	.....[.. .m....E.
0010	04 81 b6 c6 00 00 33 06 04 43 d1 55 81 93 8f e1	.....3. .C.U....
0020	e5 a3 00 50 10 36 a6 a2 e3 00 f7 46 f9 c0 50 18	...P.6....F..P.
0030	19 d3 82 fa 00 00 d4 67 d4 49 d0 32 35 8c 5e 2f	.....g .I.25.^/
0040	e4 85 45 be 74 74 be 84 3f a4 93 b3 03 79 5c ff	.E.tt.. ?....\`.
0050	21 58 ee b9 69 50 8a 96 a6 35 99 0c c9 89 87 88	!X..ip.. .5.....
0060	69 05 41 a1 28 13 28 10 20 56 74 ee b3 bf 96 c3	i.A.(. .vt.....
0070	ea 0c d5 36 79 ea 25 65 7a 7c d8 40 9f 85 0c 85	61 p 7 r

Frame (1167 bytes) Reassembled TCP (2543 bytes) Uncompressed entity body (5472 bytes)

File: "C:\DOCUMENTI\dis\IMPOSTA\Temp\etherXXXXN2B8YT" 3876 Bytes 00:00:06 P: 10 D: 10 M: 0 Drops: 0



# Status code

- Lo status code è un numero di tre cifre, di cui la prima indica la classe della risposta, e le altre due la risposta specifica
- Esistono le seguenti classi:
  - **1xx: Informational**
    - Una risposta temporanea alla richiesta, durante il suo svolgimento
  - **2xx: Successful**
    - Il server ha ricevuto, capito e accettato la richiesta
  - **3xx: Redirection**
    - Il server ha ricevuto e capito la richiesta, ma sono necessarie altre azioni da parte del client per portare a termine la richiesta
  - **4xx: Client error**
    - La richiesta del client non può essere soddisfatta per un errore da parte del client (errore sintattico o richiesta non autorizzata)
  - **5xx: Server error**
    - La richiesta può anche essere corretta, ma il server non è in grado di soddisfare la richiesta per un problema interno



# Status code: alcuni esempi

- **100 Continue**
  - se il client non ha ancora mandato il body
- **200 Ok**
  - GET con successo
- **201 Created**
  - PUT con successo
- **301 Moved permanently**
  - URL non valida, il server conosce la nuova posizione
- **400 Bad request**
  - errore sintattico nella richiesta
- **401 Unauthorized**
  - manca l'autorizzazione
- **403 Forbidden**
  - richiesta non autorizzabile
- **404 Not found**
  - URL errato
- **500 Internal server error**
  - tipicamente un programma in esecuzione sul server ha generato errore
- **501 Not implemented**
  - metodo non conosciuto dal server



# Gli header di risposta

- Gli header della risposta sono posti dal server per specificare informazioni sulla risposta e su se stesso al client
  - **Server:** una stringa che descrive il server: tipo, sistema operativo e versione
  - **Accept-ranges:** specifica che tipo di range può accettare (valori previsti: byte e none)



# Gli header generali

- Gli header generali si applicano solo al messaggio trasmesso e si applicano sia ad una richiesta che ad una risposta, ma non necessariamente alla risorsa trasmessa
- **Date**: data ed ora della trasmissione
- **MIME-Version**: la versione MIME usata per la trasmissione (sempre 1.0)
- **Transfer-Encoding**: il tipo di formato di codifica usato per la trasmissione
- **Cache-Control**: il tipo di meccanismo di caching richiesto o suggerito per la risorsa
- **Connection**: il tipo di connessione da usare
  - Connection: Keep-Alive → tenere attiva dopo la risposta
  - Connection: Close → chiudere dopo la risposta
- **Via**: usato da proxy e gateway



# Gli header dell'entità

- Gli header dell'entità danno informazioni sul body del messaggio, o, se non vi è body, sulla risorsa specificata
- **Content-Type:** oggetto/formato
  - Ogni coppia oggetto/formato costituisce un tipo MIME dell'entità acclusa
  - Specifica se è un testo, se un'immagine GIF, un'immagine JPG, un suono WAV, un filmato MPG, ecc...
  - Obbligatorio in ogni messaggio che abbia un body
- **Content-Length:** la lunghezza in byte del body
  - Obbligatorio, soprattutto se la connessione è persistente
- **Content-Base, Content-Encoding, Content-Language, Content-Location, Content-MD5, Content-Range:** l'URL di base, la codifica, il linguaggio, l'URL della risorsa specifica, il valore di digest MD5 e il range richiesto della risorsa
- **Expires:** una data dopo la quale la risorsa è considerata non più valida (e quindi va richiesta o cancellata dalla cache)
- **Last-Modified:** la data e l'ora dell'ultima modifica
  - Serve per decidere se la copia posseduta (es. in cache) è ancora valida o no
  - Obbligatorio se possibile



# Una prova con telnet

```
> telnet www.unina.it 80
```

```
GET / HTTP/1.0
```

*Request-line*

```
HTTP/1.1 200 OK
```

```
Date: Mon, 25 Oct 2010 23:02:38 GMT
```

```
Server: Apache/2.2.14 (Unix) mod_jk/1.2.28 DAV/2
```

```
ETag: W/"2097-1213082454000"
```

```
Last-Modified: Tue, 10 Jun 2008 07:20:54 GMT
```

```
Content-Length: 2097
```

```
Connection: close
```

```
Content-Type: text/html
```

*Response*



# I cookies

- **HTTP è stateless:** il server non è tenuto a mantenere informazioni su connessioni precedenti
- Un cookie è una breve informazione scambiata tra il server ed il client
- Tramite un cookie il client mantiene lo stato di precedenti connessioni, e lo manda al server di pertinenza ogni volta che richiede un documento
- Esempio: tramite un cookie viene riproposta la propria username all'atto dell'accesso ad un sito per la posta
- I cookies sono definiti in RFC2109(su proposta di Netscape)



# Cookies: header specifici

- Il meccanismo dei cookies dunque definisce due nuovi possibili header: uno per la risposta, ed uno per le richieste successive
  - **Set-Cookie**: header della risposta
    - il client può memorizzarlo (se vuole) e rispedirlo alla prossima richiesta
  - **Cookie**: header della richiesta
    - il client decide se spedirlo sulla base del nome del documento, dell'indirizzo IP del server, e dell'età del cookie
- Un browser può essere configurato per accettare o rifiutare i cookies
- Alcuni siti web richiedono necessariamente la capacità del browser di accettare i cookies



# Non solo browsing...

- HTTP non è utilizzato solo per il Web
- Ad esempio:
  - XML e VoiceXML trasportati su HTTP
  - Web Services e SOA (Service Oriented Architecture)
  - Video streaming
  - Peer-to-peer



# Adozione in corso: HTTP2

- HTTP2 ha diverse caratteristiche che lo differenziano da HTTP1.1
  - protocollo binario
  - compressione degli header
  - Server push
  - Multiplexing
  - La fase iniziale della connessione è molto simile ad HTTP 1.x, poi:
    - ‘Upgrade’ header
    - ALPN negotiation for TLS (httpS)
    - Oppure direttamente ‘PRI’ request method (se è noto che server supporta HTTP2)

**Corso di Laurea in Informatica**



**Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

**Le socket di Berkeley**

# Le socket di Berkeley

a cura di Marcello Esposito ([mesposit@unina.it](mailto:mesposit@unina.it))

# Nota di Copyright

Quest'insieme di trasparenze è stato realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovrà essere esplicitamente riportata la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

# SOCKET: cosa sono? (1)

- Le socket rappresentano **un'astrazione di canale di comunicazione** tra processi (locali o remoti).
- Attraverso di esse un'applicazione può **ricevere o trasmettere dati**.
- I meccanismi restano (quasi) indipendenti dal supporto fisico su cui le informazioni viaggiano.
- Inizialmente nascono in ambiente UNIX
  - Negli anni 80 la Advanced Research Project Agency finanziò l'Università di Berkeley per implementare la suite TCP/IP nel sistema operativo Unix.
  - I ricercatori di Berkeley svilupparono il set originario di funzioni che fu chiamato *interfaccia socket*.
  - Esse originariamente apparvero nella versione 4.1cBSD di Unix.

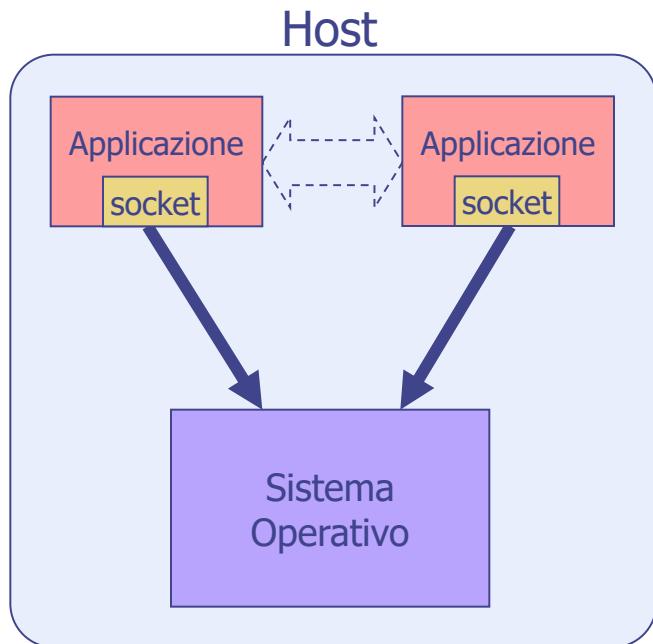
# SOCKET: cosa sono? (2)

- Si presentano sotto la forma di un'API (Application Programming Interface), cioè un **insieme di funzioni scritte in C**, che le applicazioni possono invocare per ricevere il servizio desiderato.
- Rappresentano una estensione delle API di UNIX per la gestione dell'I/O su periferica standard (files su disco, stampanti, etc).
- Questa API è poi divenuta uno standard *de facto*, ed oggi è diffusa nell'ambito di tutti i maggiori sistemi operativi (Linux, FreeBSD, Solaris, Windows... etc.).

# Interazione tra Applicazione e SO

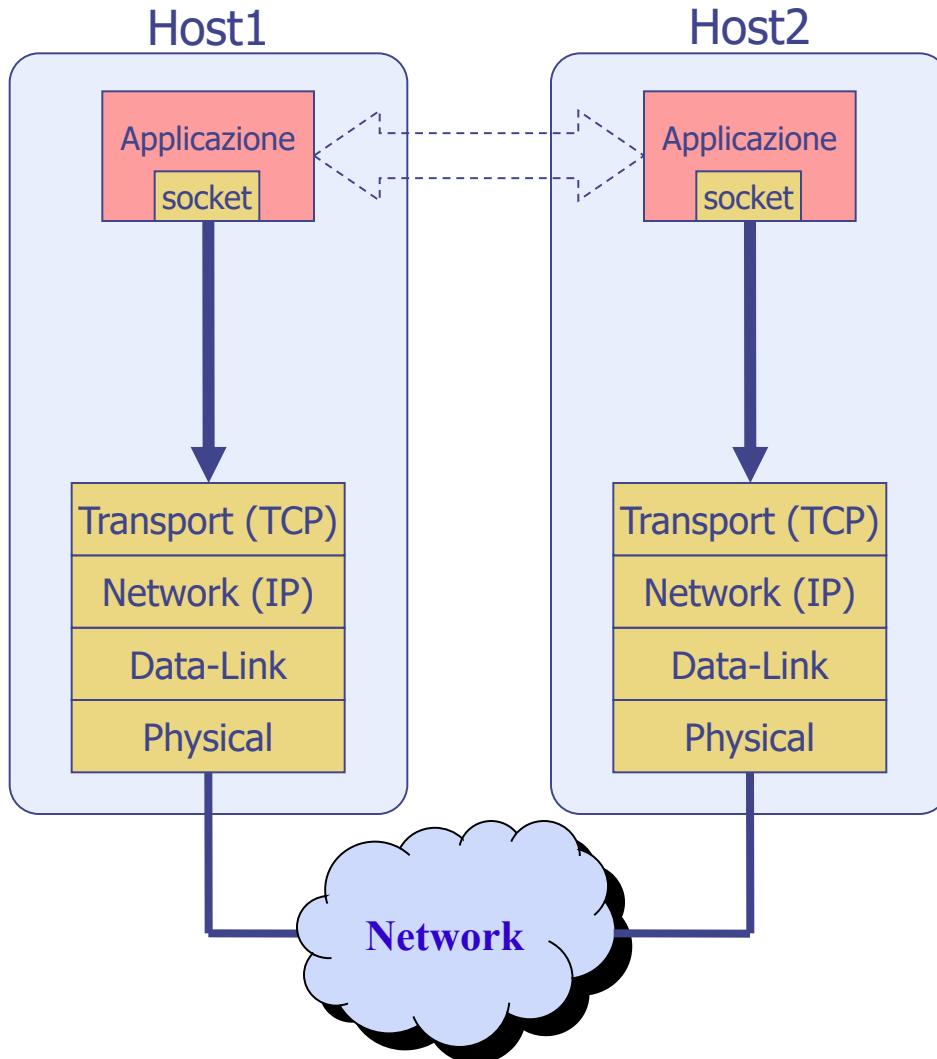
- L'applicazione chiede al sistema operativo di utilizzare i servizi di rete
- Il sistema operativo crea una socket e la restituisce all'applicazione
  - restituito un socket descriptor
- L'applicazione utilizza la socket
  - Open, Read, Write, Close.
- L'applicazione chiude la socket e la restituisce al sistema operativo

# Caso d'uso: comunicazione locale



- Due applicazioni, localizzate sulla stessa macchina, scambiano dati tra di loro utilizzando l'interfaccia delle socket.
- Le socket utilizzate a questo scopo vengono comunemente definite Unix-domain socket.

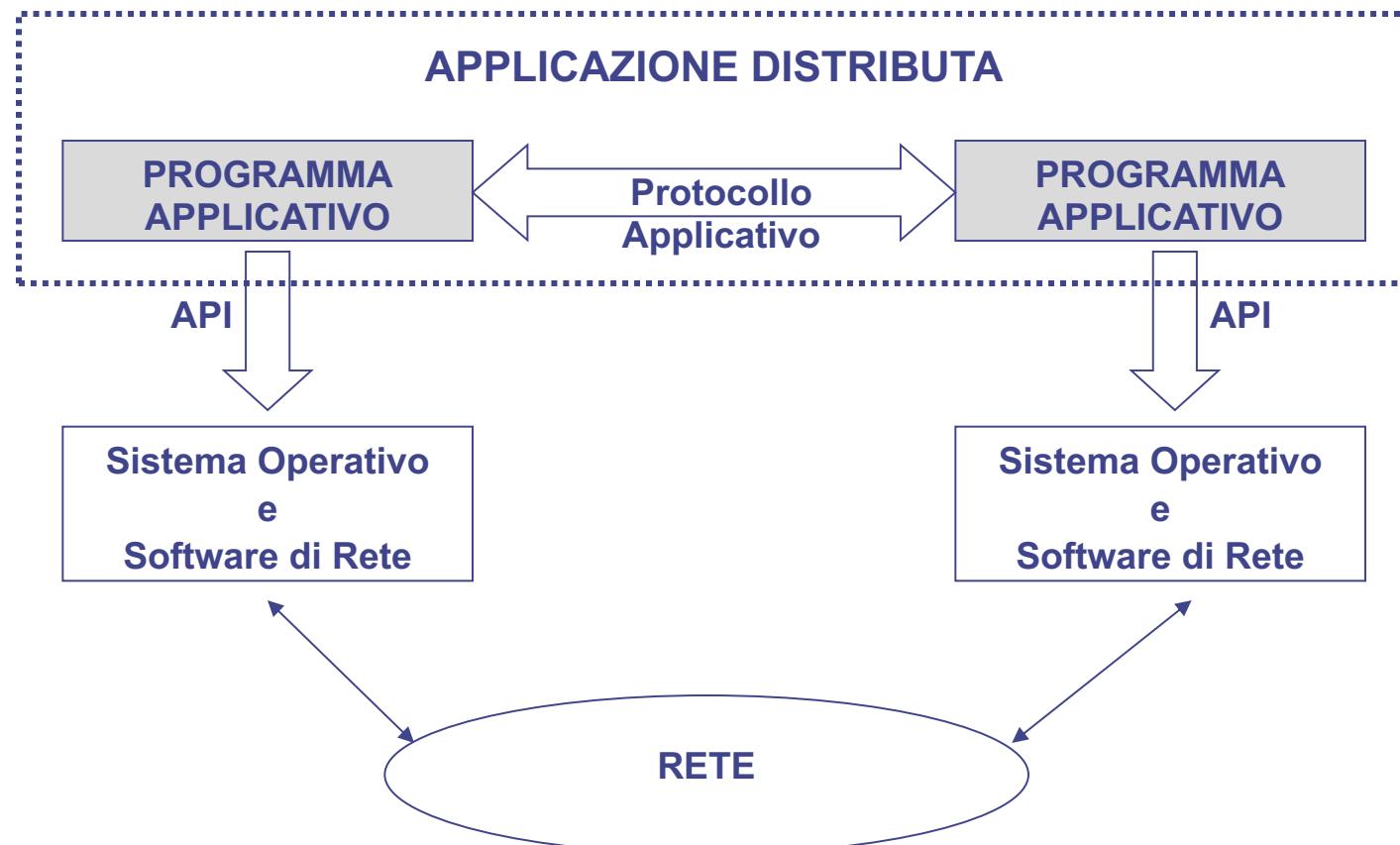
# Caso d'uso: comunicazione remota via TCP/IP



- Anche due applicazioni situate su macchine distinte possono scambiare informazioni secondo gli stessi meccanismi.
- Così funzionano telnet, ftp, ICQ, Napster.

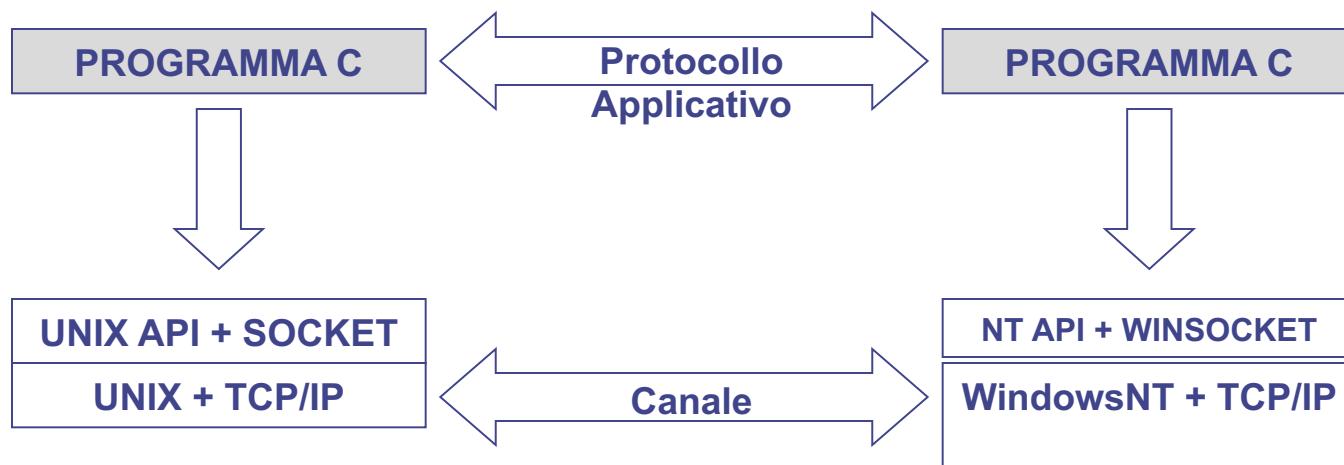
# Interfacce e protocolli

◆ Esempio di applicazione distribuita:



# Interfacce e protocolli

- ◆ La configurazione di riferimento di una applicazione distribuita basata su TCP/IP e socket è il seguente:



# Il problema della connessione

- Nel momento in cui una entità decide di instaurare una comunicazione con un'entità paritaria, come assicurarsi che quest'ultima sia disponibile?
  - La chiamata telefonica: chi desidera instaurare la comunicazione compone **il numero** del destinatario e attende durante il segnale di chiamata. Dall'altro lato **uno squillo** avverte di una chiamata in arrivo. Se si è disponibili alla comunicazione (si è in casa, si può ascoltare lo squillo e non si è sotto la doccia) **si alza la cornetta**. Lo squillo dal lato del chiamante **termina**. Da questo momento in poi la chiamata è instaurata e diviene simmetrica: chiunque può parlare quando vuole.
- E' necessario che il chiamante conosca l'indirizzo del chiamato e che il chiamato sia in attesa di eventuali comunicazioni.

# Il paradigma Client-Server (C/S)

- Il chiamato è il **server**:

- deve aver divulgato il proprio indirizzo
- resta in attesa di chiamate
- in genere viene contattato per fornire un servizio

- Il chiamante è il **client**:

- conosce l'indirizzo del server
- prende l'iniziativa di comunicare
- usufruisce dei servizi messi a disposizione dal server

# Il concetto di indirizzo

- Una comunicazione può essere identificata attraverso la quintupla:  
**{protocol, local-addr, local-process, foreign-addr, foreign-process}**
- Una coppia {addr, process} identifica univocamente un terminale di comunicazione (end-point).
- Nel mondo IP, ad esempio:
  - **local-addr e foreign-addr** rappresentano indirizzi IP
  - **local-process e foreign-process** rappresentano numeri di porto

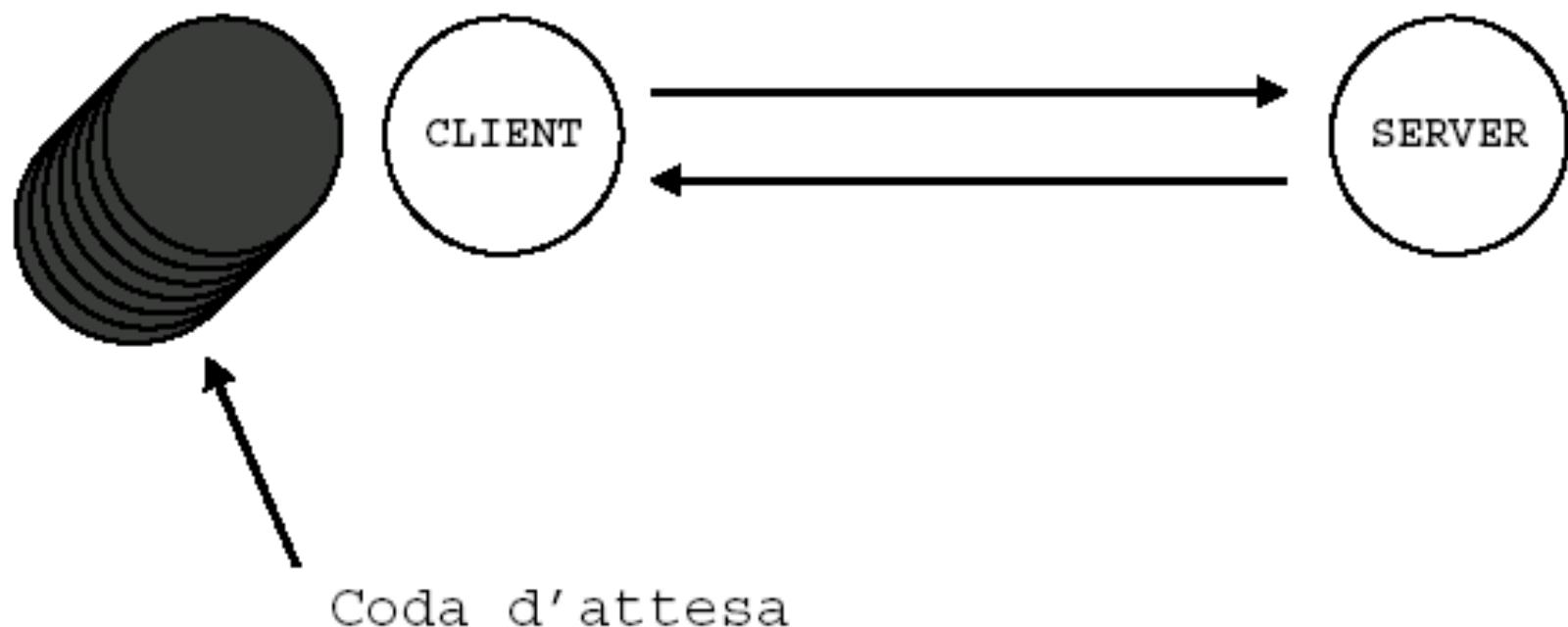
# Server concorrente e iterativo

- Un server può ricevere chiamate anche da più client diversi.
- Ogni comunicazione richiederà un certo tempo prima di potersi considerare conclusa.

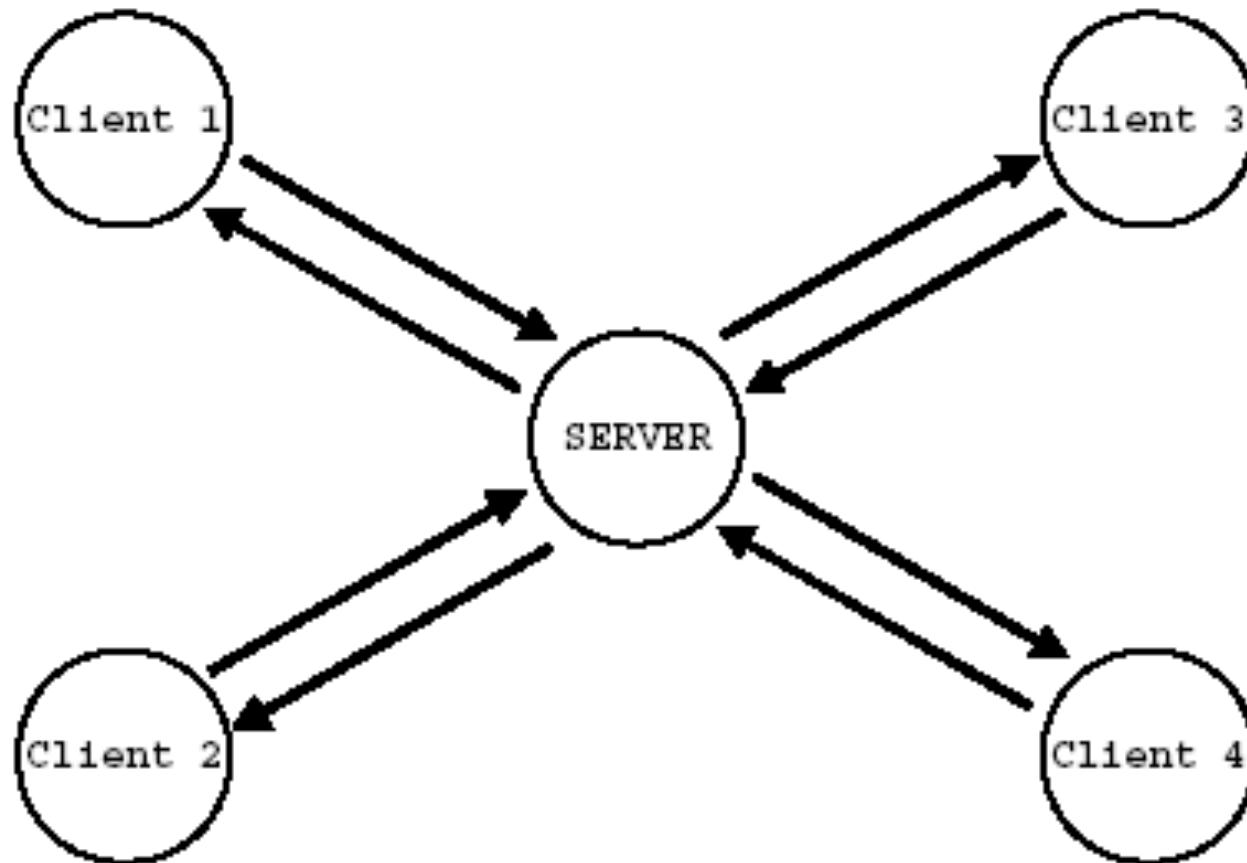
**E se una chiamata arriva mentre il server è già impegnato in una comunicazione?**

- Un server che accetti più comunicazioni contemporaneamente si definisce **concorrente**.
- Un server che accetti una sola comunicazione alla volta è detto **iterativo**.
  - In questo ultimo caso una richiesta può essere servita solo quando la precedente si è già conclusa.
  - Questo è il paradigma applicato nel modello di comunicazione telefonica di base.
  - E l'avviso di chiamata?...

# Server Iterativo



# Server Concorrente



# Il paradigma di comunicazione “Connection-Oriented”

- In una comunicazione dati Connection-Oriented, i due end-point dispongono di un canale di comunicazione che:
  - trasporta flussi
  - è affidabile
  - è dedicato
  - preserva l'ordine delle informazioni
- Il canale si comporta cioè come una sorta di “**tubo**”: tutto quello che viene inserito al suo interno, arriverà inalterato dall'altro lato e nello stesso ordine con cui è stato immesso.
- Non è detto che vengano però mantenuti i limiti dei messaggi.
- La comunicazione telefonica è più simile ad una comunicazione connection-oriented.

# Il paradigma di comunicazione “Datagram”

- In una comunicazione Datagram (anche detta connectionless), il canale
  - trasporta messaggi
  - non è affidabile
  - è condiviso
  - non preserva l'ordine delle informazioni
- Se si inviano dieci messaggi dall'altro lato essi possono anche arrivare mescolati tra di loro e tra i messaggi appartenenti ad altre comunicazioni. I limiti dei messaggi vengono comunque preservati.
- La posta ordinaria è un esempio di comunicazione a datagramma.

# Connection-Oriented vs Datagram

## ◆ Connection oriented.

- Principali vantaggi:
  - ◆ Affidabilità
  - ◆ Controllo di flusso
- Principali svantaggi:
  - ◆ Overhead per instaurare la connessione

## ◆ Datagram.

- Principali Vantaggi
  - ◆ Basso overhead
- Principali svantaggi
  - ◆ Nessun controllo sulla consegna

◆ Le socket che utilizzano i protocolli Internet sfruttano rispettivamente TCP (Transmission Control Protocol) e UDP (User Datagram Protocol) per implementare le due tipologie di comunicazione. In entrambi i casi il protocollo di livello inferiore è IP (che è un protocollo datagram).

# Naming / Binding

- ◆ È l'operazione in cui associamo un indirizzo transport ad una socket già creata
- ◆ In questo modo l'indirizzo diventa noto al sistema operativo ed altre socket sono in grado di stabilire una connessione

# Byte Stream e Datagram

- ◆ È possibile impostare il protocollo che verrà utilizzato per il trasferimento dei dati
- ◆ Nel caso delle socket abbiamo due opzioni:
  - **byte-stream**: I dati vengono trasferiti come una sequenza ordinata ed affidabile di byte (SOCK\_STREAM)
  - **Datagram**: I dati vengono inviati come messaggi indipendenti ed inaffidabili (SOCK\_DGRAM)

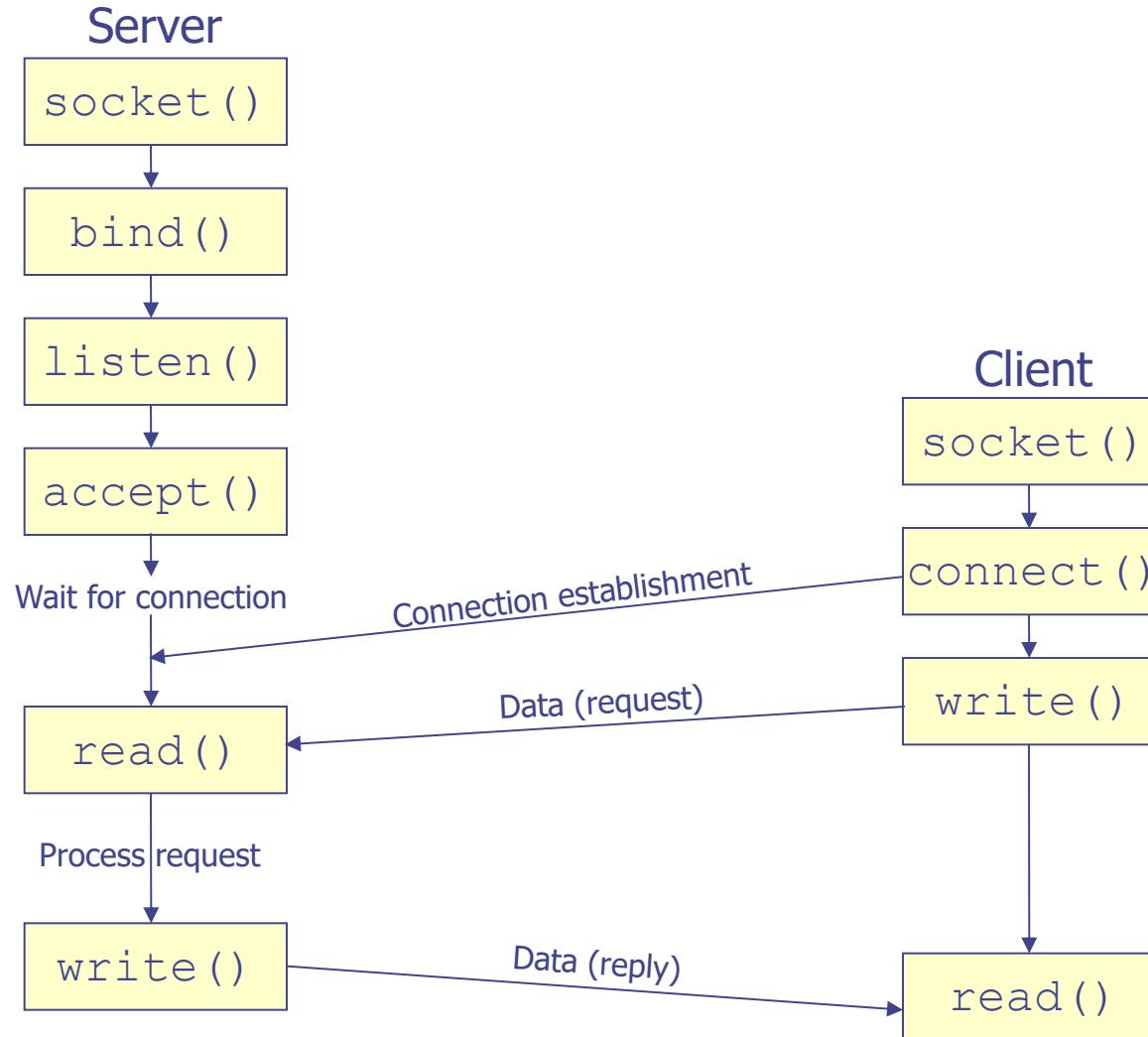
# Progettazione di un Server TCP

- Creazione di un endpoint
  - Richiesta al sistema operativo
- Collegamento dell'endpoint ad una porta
  - Ascolto sulla porta
    - Processo sospeso in attesa
- Accettazione della richiesta di un client
- Letture e scritture sulla connessione
- Chiusura della connessione

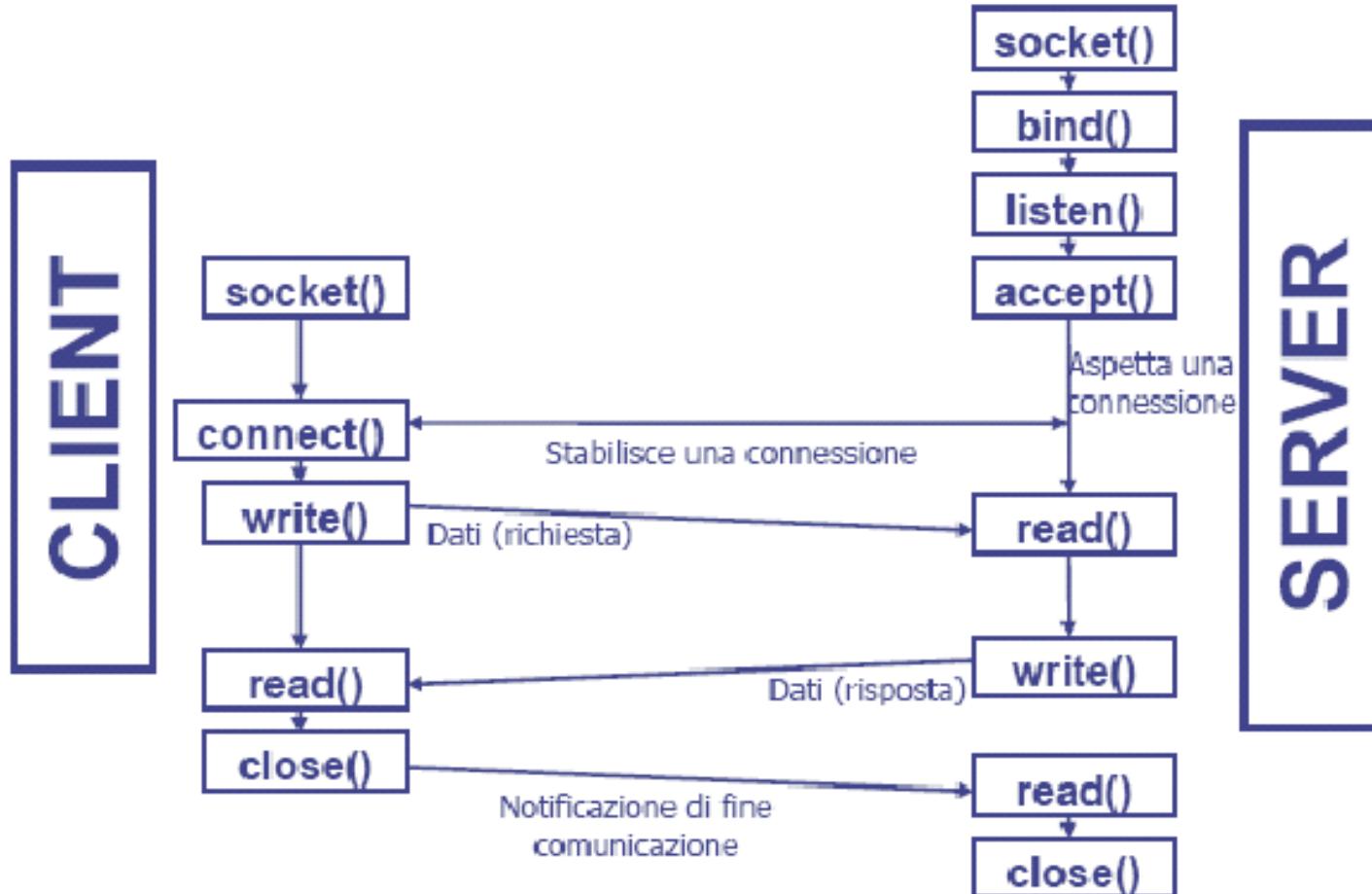
# Progettazione di un Client TCP

- Creazione di un endpoint
  - Richiesta al sistema operativo
- Creazione della connessione
  - Implementa open di TCP (3-way handshake)
- Lettura e scrittura sulla connessione
  - Analogo a operazioni su file in Unix
- Chiusura della connessione
  - Implementa close di TCP (4-way handshake)

# Le chiamate per una comunicazione Connection-Oriented



# Le chiamate per una comunicazione Connection-Oriented



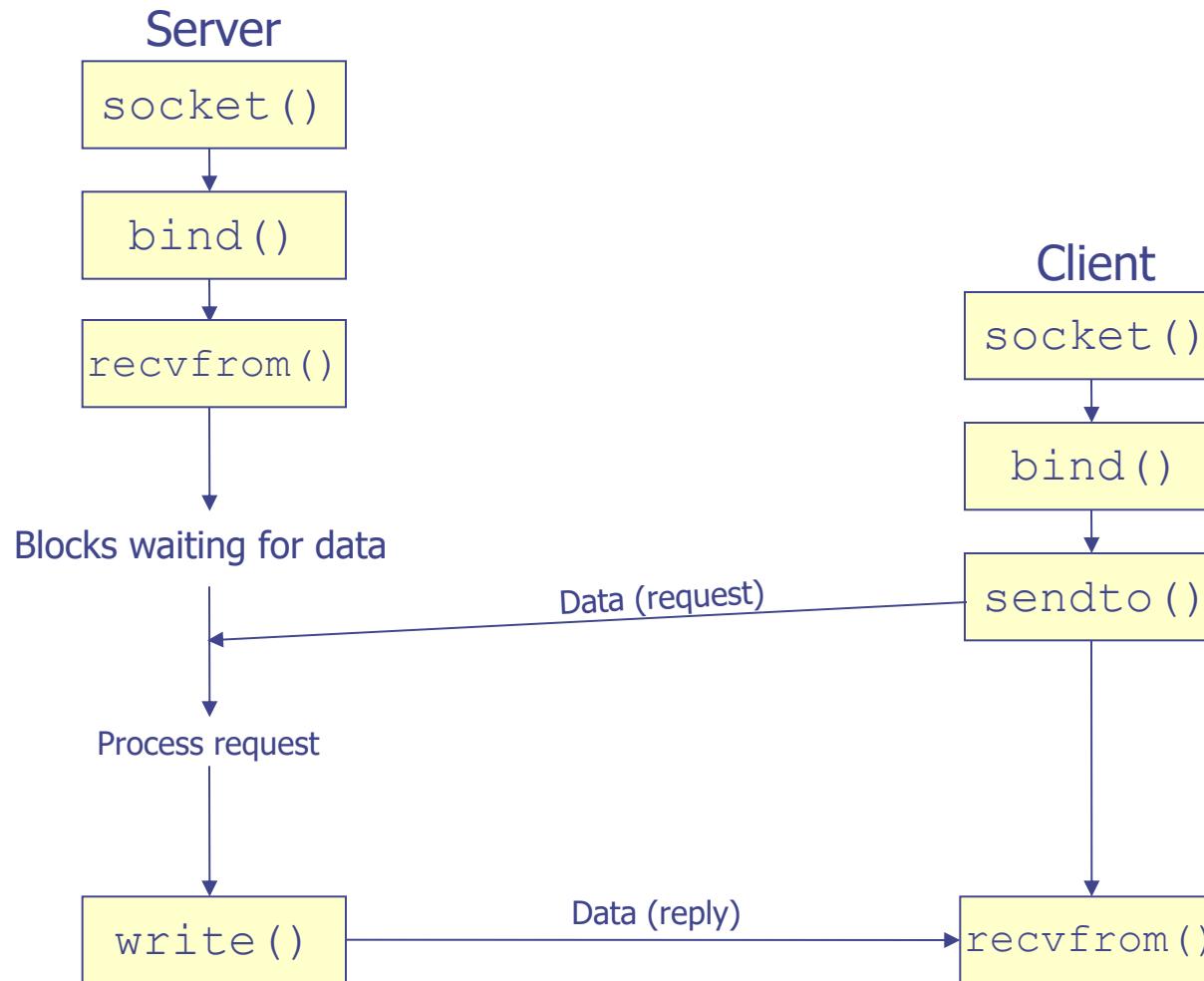
# Progettazione di un Server UDP

- Creazione di un endpoint
  - Richiesta al sistema operativo
- Collegamento dell'endpoint ad una porta
  - open passiva in attesa di ricevere datagram
- Ricezione ed invio di datagram
- Chiusura dell'endpoint

# Progettazione di un Client UDP

- Creazione di un endpoint
  - Richiesta al sistema operativo
- Invio e ricezione di datagram
- Chiusura dell'endpoint

# Le chiamate per una comunicazione Datagram



# Esempi di programmazione delle socket in C

# Le strutture dati per le socket: il trattamento degli indirizzi (1)

```
<sys/socket.h>
struct sockaddr {
    u_short sa_family;    /* address family: AF_xxx value */
    char    sa_data[14];   /* up to 14 bytes of protocol-specific address */
};

<netinet/in.h>
struct in_addr {
    u_long s_addr;        /* 32-bit netid/hostid network byte ordered */
};

struct sockaddr_in {
    short            sin_family;    /* AF_INET */
    u_short          sin_port;      /* 16-bit port number network byte ordered */
    struct in_addr  sin_addr;
    char             sin_zero[8];   /* unused */
};

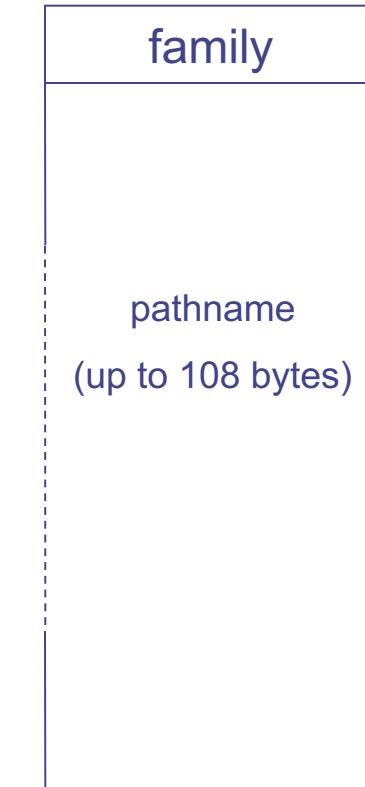
<sys/un.h>
struct sockaddr_un {
    short  sun_family;  /* AF_UNIX */
    char   sun_path[108]; /* pathname */
};
```

# Le strutture dati per le socket: il trattamento degli indirizzi (2)

struct sockaddr\_in



struct sockaddr\_un



# La system-call socket ()



Questa system-call serve ad **istanziare un nuovo descrittore di socket**.



Esso verrà utilizzato in tutte le successive chiamate

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int socket(int family, int type, int protocol);
```



family può essere: AF\_UNIX, AF\_INET...



type può essere: SOCK\_STREAM, SOCK\_DGRAM, SOCK\_RAW, SOCK\_SEQPACKET, SOCK\_RDM.



protocol indica il protocollo utilizzato.



il valore restituito è un descrittore di socket (di tipo int secondo lo stile Unix).



Della quintupla, dopo la chiamata `socket()`, resta specificato solo il primo campo:

{**protocol**, local-addr, local-process, foreign-addr, foreign-process}

# La system-call bind() (1)

- ◆ Questa system-call serve ad **assegnare un indirizzo locale** (name) ad una socket.

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *myaddr, int addrlen);
```

- ◆ sockfd è il descrittore di socket restituito da `socket()`.
- ◆ myaddr punta ad un generico indirizzo, mentre addrlen è la lunghezza di quest'ultimo.
- ◆ il valore restituito è indicativo del successo dell'operazione.
- ◆ Della quintupla, dopo la chiamata `bind()`, restano specificati il secondo ed il terzo campo, cioè gli estremi locali della comunicazione:  
**{protocol, local-addr, local-process, foreign-addr, foreign-process}**

# La system-call bind() (2)

- ◆ Essa può essere invocata in una molteplicità di casi:
  - un **server** vuole registrare i suoi estremi (già divulgati precedentemente) presso il sistema sul quale si trova. In questo modo è come se dicesse:- "*Questo è il mio indirizzo e tutti i messaggi inviati ad esso devono essere consegnati a me*".
    - ◆ Ciò accade sia per i server connection-oriented che per quelli connectionless.
  - un **client** vuole registrare uno specifico indirizzo per se stesso.
  - un **client connectionless** vuole assicurarsi uno specifico indirizzo poiché è solo attraverso di esso che può essere raggiunto da un server al quale aveva in precedenza inoltrato una richiesta.
- ◆ Può restituire una condizione di errore, per esempio, se l'indirizzo al quale si desidera "legarsi" risulta già occupato.

# La system-call connect () (1)

- ◆ Attraverso la chiamata `connect()`, subito dopo una chiamata `socket()`, un processo **client** stabilisce una connessione con un **server**.

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *servaddr, int addrlen);
```

- ◆ `sockfd` è il descrittore di socket restituito da `socket()`.
- ◆ `servaddr` punta all'indirizzo (generico) del server, e `addrlen` è sempre la lunghezza di quest'ultimo.
- ◆ il valore restituito è indicativo del successo dell'operazione.
- ◆ Della quintupla, dopo la chiamata `connect()`, restano specificati tutti i campi relativi agli indirizzi:

{**protocol**, **local-addr**, **local-process**, **foreign-addr**, **foreign-process**}

# La system-call connect () (2)

- ◆ Per le **comunicazioni connection-oriented**, la chiamata `connect()` scatena una fase di segnalazione tra il client ed il server intesa a stabilire realmente la connessione, ed a concordare una serie di parametri che la caratterizzano. In questo caso la `connect()` è bloccante e non ritorna se non dopo aver instaurato la connessione (o, eventualmente, con una condizione di errore).
- ◆ Un client non deve necessariamente realizzare una chiamata a `bind()` prima della `connect()`, poiché è il sistema che assegna automaticamente un indirizzo valido tra quelli disponibili. Questo è il motivo per cui tutta la quintupla risulterà valorizzata dopo una chiamata a `connect()`.
- ◆ Per un client **connectionless** c'è ancora la possibilità di invocare la chiamata `connect()`. In questo caso, però, il server non viene realmente contattato (potrebbe anche non essere attivo), ma si produce semplicemente la memorizzazione locale dell'indirizzo del server con la conseguenza che:
  - ogni successivo messaggio scritto sulla socket sarà diretto a quel server;
  - ogni messaggio ricevuto sulla socket verrà accettato solo se proveniente da quel server.

# La system-call listen()

- ◆ Attraverso la chiamata `listen()`, un **server** manifesta la sua volontà di **apprestarsi a ricevere connessioni**.
- ◆ Generalmente si esegue dopo `socket()` e `bind()` e prima di `accept()`.

```
#include <sys/types.h>
#include <sys/socket.h>

int listen(int sockfd, int qlen);
```

- ◆ `sockfd` è il descrittore di socket restituito da `socket()`.
- ◆ `qlen` indica quante richieste di connessione possono essere accodate dal sistema in attesa di essere servite.
- ◆ il valore restituito è indicativo del successo dell'operazione.

# La system-call accept () (1)

- ❖ Dopo la chiamata `listen()`, un **server** si mette realmente in **attesa di connessioni** attraverso una chiamata ad `accept()`.

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *peer, int *addrlen);
```

- ❖ `sockfd` è il descrittore di socket restituito da `socket()`.
- ❖ `peer` e `addrlen` sono parametri di ingresso-uscita:

- prima dell'`accept()`, devono essere impostati coerentemente con le caratteristiche delle strutture dati allocate.
- dopo l'`accept()`, contengono l'indirizzo del client di cui si è accettata la connessione, con la sua lunghezza (minore o uguale a quella preimpostata).

# La system-call accept () (2)

- ◆ accept () preleva la prima richiesta di connessione dalla coda e crea una nuova socket avente le stesse proprietà di sockfd, supponendo implicitamente che il servente sia concorrente.
- ◆ Se la coda è vuota la chiamata è invece bloccante.
- ◆ accept () restituisce fino a tre valori:
  - se è andata a buon fine restituisce
    - il nuovo descrittore di socket;
    - l'indirizzo del client di cui si è accettata la connessione;
    - la sua lunghezza dell'indirizzo.
  - se non è andata a buon fine
    - il codice relativo all'errore verificatosi.

# La system-call accept () (3)

- Il server quindi utilizza due socket diversi per ogni connessione con un client
  - il **socket di ascolto** (listening socket) è quello creato dalla funzione **socket()**
    - utilizzato per tutta la vita del processo
    - in genere usato solo per accettare richieste di connessione
  - il **socket connesso** (connected socket) è quello creato dalla funzione **accept()**
    - usato solo per la connessione con un certo client
    - usato per lo scambio dei dati con il client
- I due socket identificano due connessioni distinte

# Esempio di accept () in un server concorrente.

```
int sockfd, newsockfd;

if ( (sockfd = socket( ... ) ) < 0 )
err_sys("socket error");
if ( bind(sockfd, ... ) < 0 )
err_sys("bind error");
if ( listen(sockfd, 5) < 0 )
err_sys("listen error");

for ( ; ; ) {
newsockfd = accept(sockfd, ... ); /* blocks */
if (newsockfd < 0)
    err_sys("accept error");

if (fork() == 0) {
    close(sockfd); /* child */
    doit(newsockfd); /* process the request */
    close(newsockfd);
    exit(0);
}

close(newsockfd); /* parent */
}
```

- Dopo l'accept () il descrittore newsockfd ha la quintupla tutta impostata, ed è pronto ad essere utilizzato.
- sockfd, invece, continua ad avere impostati solo i primi tre campi e può essere usato per accettare le altre connessioni, senza la necessità di istanziare, per questo, una nuova socket.

# Esempio di accept() in un server iterativo.

```
int sockfd, newsockfd;

if ( (sockfd = socket( ... ) ) < 0 )
err_sys("socket error");
if ( bind(sockfd, ... ) < 0 )
err_sys("bind error");
if ( listen(sockfd, 5) < 0 )
err_sys("listen error");

for ( ; ; ) {
newsockfd = accept(sockfd, ... ); /* blocks */
if (newsockfd < 0)
    err_sys("accept error");

doit(newsockfd); /* process the request */
close(newsockfd);
}
```

# Ricevere ed inviare i dati

- ◆ Una volta utilizzate le precedenti chiamate, la “connessione” è stata predisposta.
- ◆ La quintupla risulta completamente impostata.
- ◆ A questo punto chiunque può inviare o ricevere dati.
- ◆ Per questo, è necessario aver concordato un protocollo comune.
  - Per esempio, nella comunicazione telefonica, il chiamato parla per primo e risponde:- “Pronto, chi è?”, e quindi il chiamante fornisce la propria identità.

# Le system-call send() e sendto()

- send() e sendto() si utilizzano per **inviare dati** verso l'altro terminale di comunicazione.

```
int send(int sockfd, char *buff, int nbytes, int flags);  
int sendto(int sockfd, char *buff, int nbytes, int flags,  
          struct sockaddr *to, int addrlen);
```

- sockfd è il descrittore restituito dalla chiamata socket().
- buff punta all'inizio dell'area di memoria contenente i dati da inviare.
- nbytes indica la lunghezza in bytes del buffer, e quindi, il numero di bytes da inviare.
- to e addrlen indicano l'indirizzo del destinatario, con la sua lunghezza.
- flags abilita particolari opzioni. In generale è pari a 0.
- entrambe restituiscono il numero di bytes effettivamente inviati.

# Le system-call `recv()` e `recvfrom()`

- ◆ `recv()` e `recvfrom()` si utilizzano per **ricevere dati** dall'altro terminale di comunicazione.

```
int recv(int sockfd, char *buff, int nbytes, int flags);  
int recvfrom(int sockfd, char *buff, int nbytes, int flags,  
             struct sockaddr *from, int *addrlen);
```

- ◆ `sockfd` è il descrittore restituito dalla chiamata `socket()`.
- ◆ `buff` punta all'inizio dell'area di memoria in cui devono essere ricopiatati i dati ricevuti.
- ◆ `nbytes` è un parametro di ingresso che indica la lunghezza del buffer.
- ◆ `from` e `addrlen` contengono, dopo la chiamata, l'indirizzo del mittente con la sua lunghezza.
- ◆ `flags` abilita particolari opzioni. In generale è pari a 0.
- ◆ entrambe restituiscono il numero di bytes ricevuti (minore o uguale a `nbytes`).
- ◆ in assenza di dati da leggere, la chiamata è bloccante.

# La system-call close()

- ◆ Chiude una socket e rilascia le risorse ad essa associate.

```
int close(int fd);
```

- ◆ in seguito a questa chiamata, eventuali dati pendenti, vengono inviati al destinatario prima che la socket venga chiusa.

# Il marshalling dei parametri (1)

- ◆ Quando si invia un dato sulla rete, in generale, nulla si può ipotizzare sulla architettura dell'host ricevente.
- ◆ E' necessario quindi che i dati in transito siano codificati secondo una convenzione standard.
- ◆ Con il termine *marshalling* si intende la traduzione di un'informazione in un formato prefissato e comprensibile universalmente.
- ◆ L'operazione inversa è detta *un-marshalling*.
- ◆ E' un'operazione tipica del sesto livello della pila OSI (presentazione) il cui intento è di assicurare **portabilità** ad un programma.
  
- ◆ Se, ad esempio, l'host trasmittente è dotato di un processore Intel (little-endian) e l'host ricevente è dotato invece di un processore Motorola (big-endian), lo scambio tra questi non può avvenire senza uniformare le convenzioni sulla codifica dei dati.

# Il marshalling dei parametri (2)

- ◆ Una serie di funzioni è stata prevista, nell'ambito della comunicazione su Internet, proprio a questo scopo .
- ◆ Vanno sotto il nome di *Byte Ordering Routines*:
  - da host byte order a TCP/IP byte order (big-endian) e viceversa
- ◆ Sui sistemi che adottano la stessa convenzione fissata per Internet, queste routines sono implementate come “null-macros” (funzioni ‘vuote’).

```
#include <sys/types.h>
#include <netinet/in.h>

u_long htonl(u_long hostlong); /* host to net long */
u_short htons(u_short hostshort); /* host to net short */
u_long ntohl(u_long netlong); /* net to host long */
u_short ntohs(u_short netshort); /* net to host short */
```

- ◆ E' implicito in queste funzioni che uno short occupi 16 bit e un long ne occupi 32.

# Esercizio

- ◆ Scrivere un programma server ed un programma client che funzionino nel seguente modo:
  - il client chiede sullo standard input una stringa di caratteri e la invia al server;
  - il server riceve la stringa, la visualizza e la invia nuovamente al client;
  - anche il client visualizza la stringa appena ricevuta.
- ◆ Questo è un esempio di programma *echo*.



# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

### **Il livello rete**

**I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso**

# Nota di copyright per le slide COMICS



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

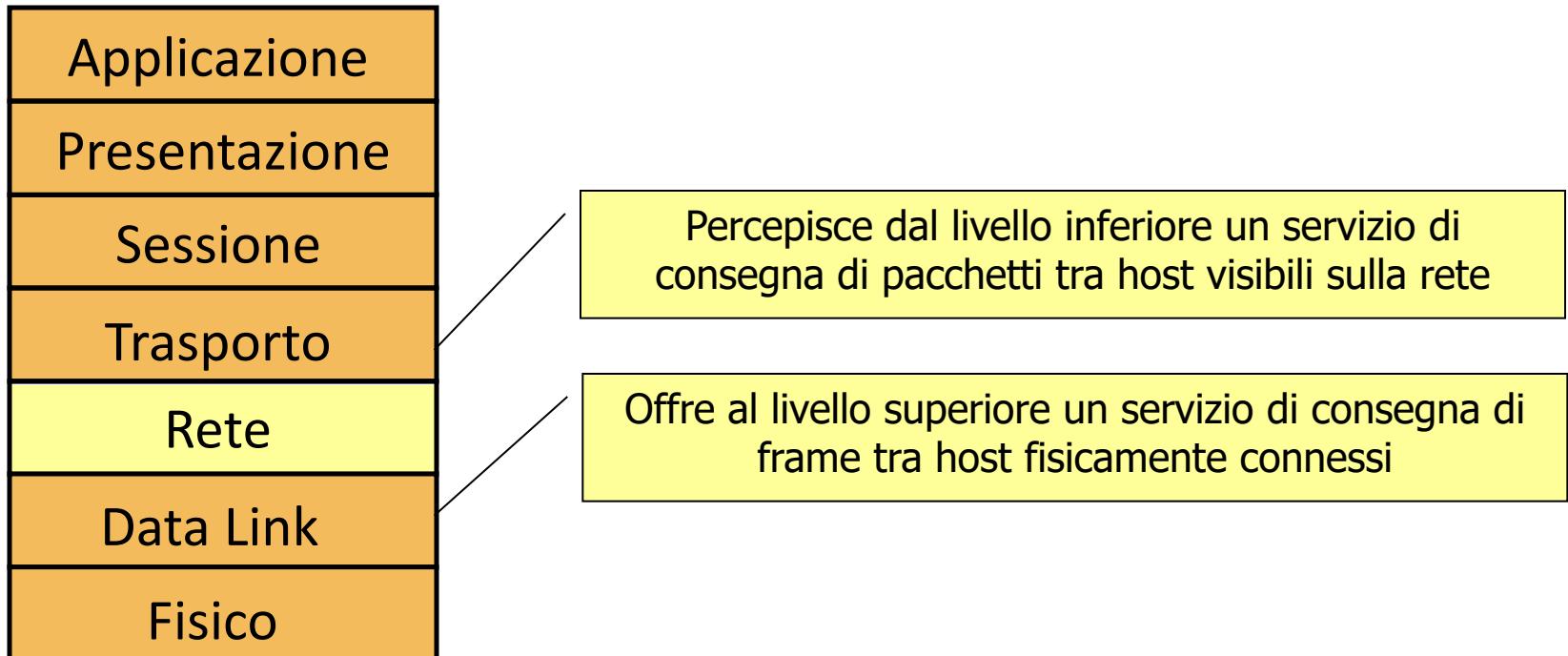
Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta



# Il livello rete

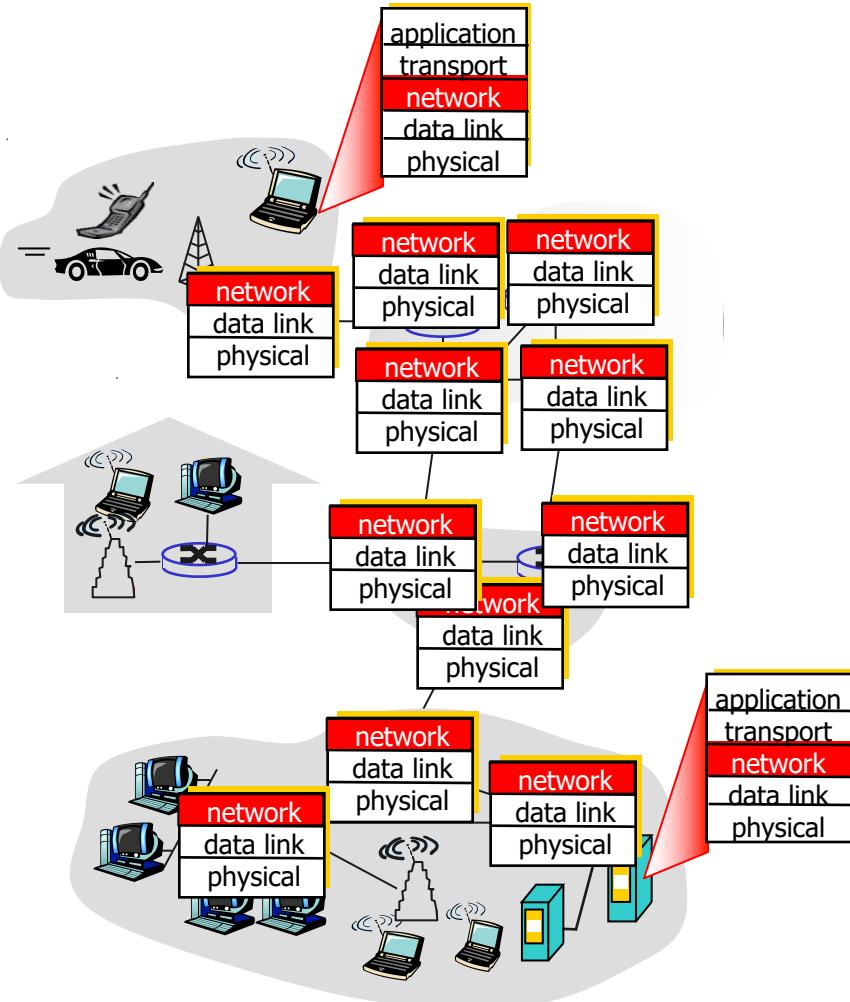
- Stack OSI





# Le funzioni del livello rete

- Trasportare i pacchetti dall'host mittente a quello ricevente
- Implementare protocolli di livello rete in *tutti* i router e in *tutti* gli host





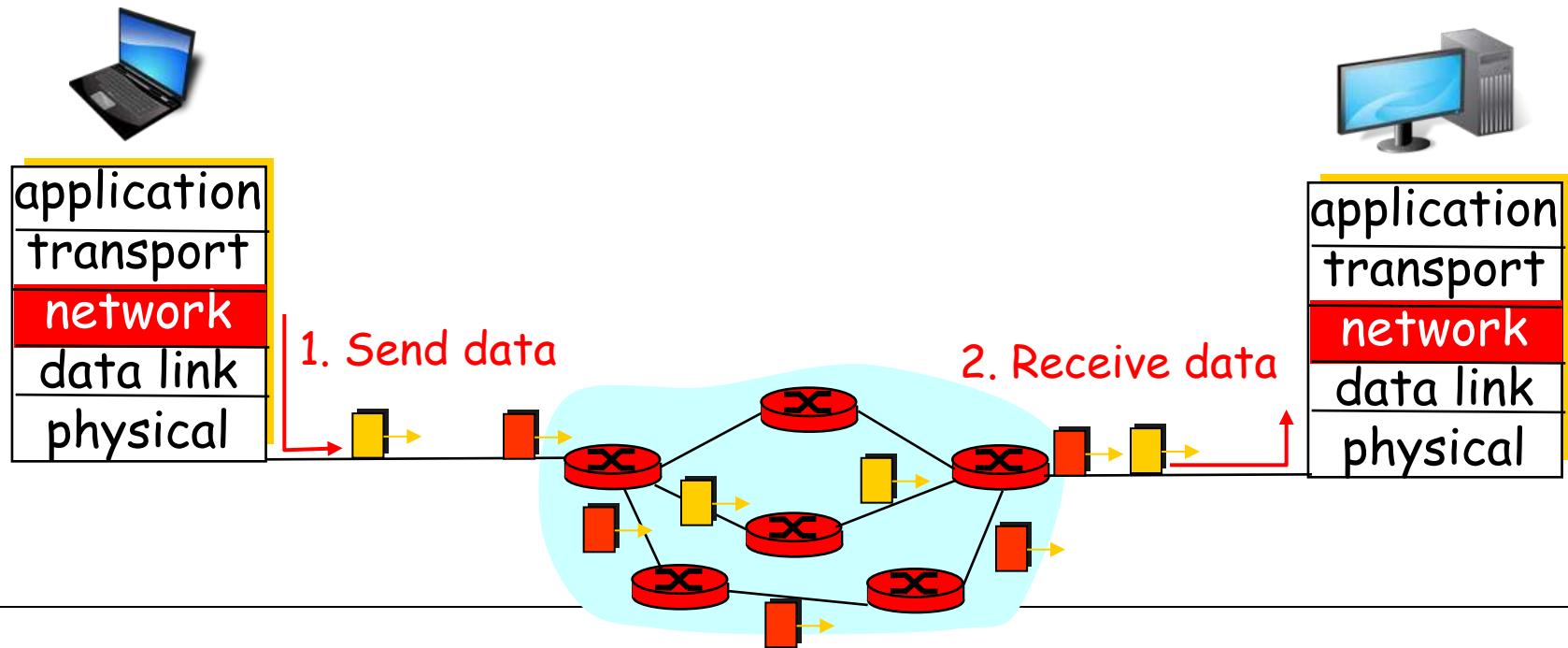
# Le funzioni del livello rete

- Le reti possono essere classificate a seconda del metodo utilizzato per trasportare i pacchetti dalla sorgente alla destinazione
  - **Reti a datagrammi**  
ogni pacchetto è instradato indipendentemente dagli altri pacchetti dello stesso flusso
  - **Reti a circuiti virtuali**  
viene precalcolato un percorso e tutti i pacchetti del flusso seguono questo percorso
  - **NB: parliamo comunque di reti a commutazione di pacchetto!**



# Packet switching: reti a datagrammi

- Ogni nodo che riceve un pacchetto decide in maniera **indipendente** a quale altro nodo inoltrarlo, sulla base dell'indirizzo destinazione contenuto nel pacchetto
  - **Indipendente** rispetto agli altri nodi
  - **Indipendente** rispetto agli altri pacchetti passanti per lo stesso nodo
- Pacchetti tra la stessa coppia sorgente-destinazione possono seguire percorsi differenti





# Le funzioni di **forwarding** e **routing**

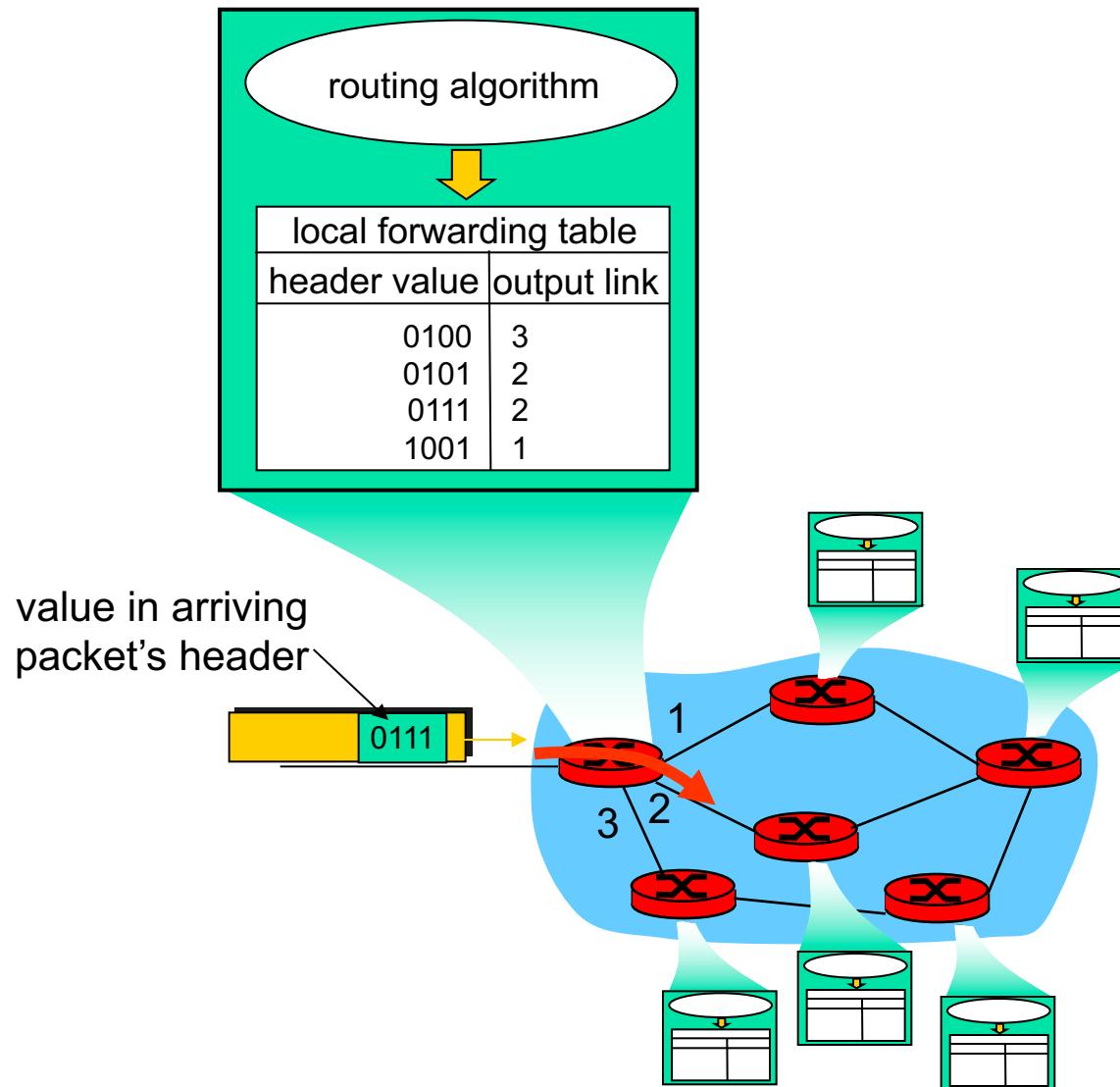
- **forwarding:** spostare i pacchetti dalla coda/interfaccia di ingresso a quella di uscita
- **routing:** determinare la strada che un pacchetto deve seguire da una sorgente ad una destinazione:
  - *routing algorithms*

## analogia:

- **routing:** processo di costruzione di un viaggio dalla partenza all'arrivo
  - A mano
  - Usando un software
  - ...
- **forwarding:** processo di movimentazione ad ogni singola rotonda...



# La relazione tra forwarding e routing





# Forwarding table

4 miliardi ( $2^{32}$ ) di entry nella tabella di inoltro: occorrono soluzioni per compattarla !

	<u>Destination Address Range</u>				<u>Link Interface</u>
Da:	11001000	00010111	00010000	00000000	
A:	11001000	00010111	00010111	11111111	0
-----					
Da:	11001000	00010111	00011000	00000000	
A:	11001000	00010111	00011000	11111111	1
-----					
Da:	11001000	00010111	00011001	00000000	
A:	11001000	00010111	00011111	11111111	2
-----					
Altrimenti ( <i>regola di default</i> )					3



# Longest prefix matching

<u>Prefix Match</u>	<u>Link Interface</u>
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
otherwise	3

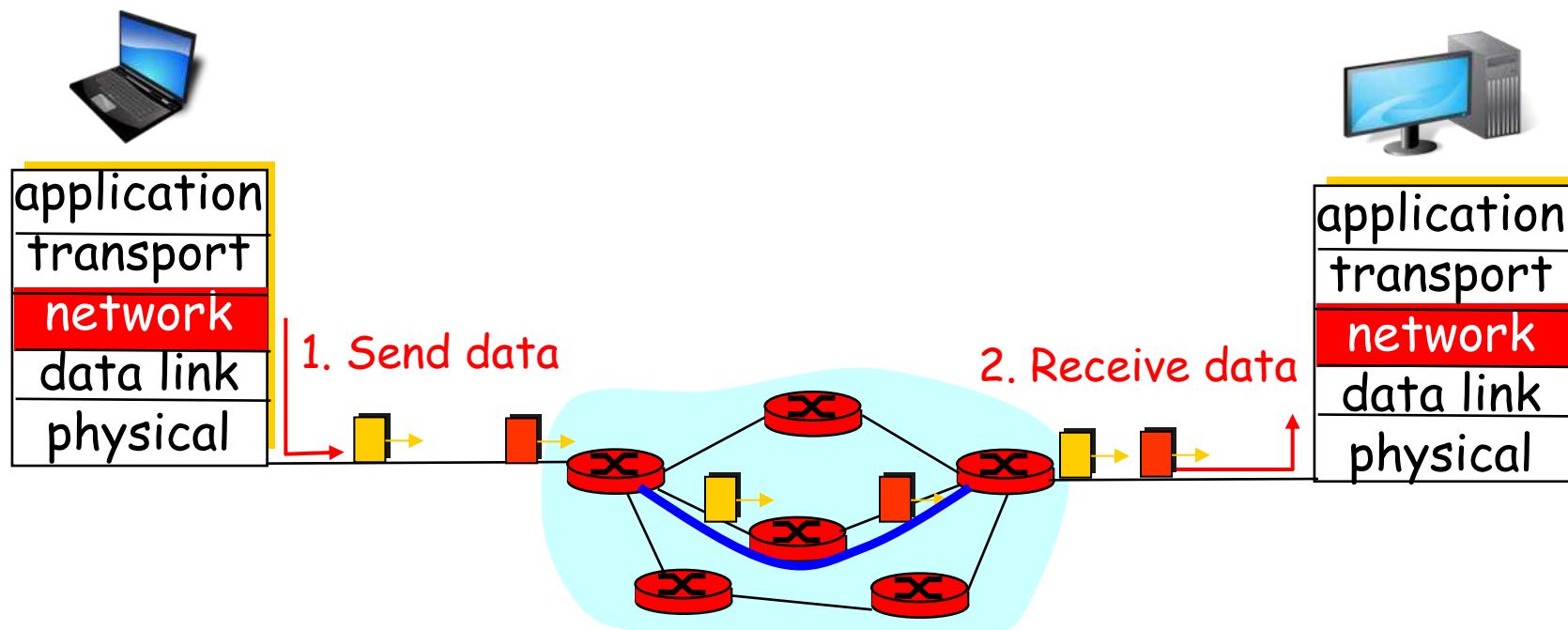
## Esempi

- DA: 11001000 00010111 00010110 10100001 Quale interfaccia? 0
- DA: 11001000 00010111 00011000 10101010 Quale interfaccia? 1
- DA: 11001000 00010111 00011001 10101010 Quale interfaccia? 2



# Packet switching: reti a circuiti virtuali

- Ogni pacchetto contiene il numero del circuito virtuale
- Il circuito virtuale è stabilito prima della trasmissione dei dati
- I nodi devono conservare informazioni sui circuiti virtuali che li attraversano





# Circuito virtuale

Percorso associato ad una coppia di terminali che intendono comunicare

- Insieme di risorse assegnate in ciascun dispositivo intermedio (router) per il trattamento dei pacchetti associati al circuito virtuale
- Fase di **call setup** prima che i dati possano fluire attraverso la rete
- Ogni pacchetto associato ad un circuito virtuale porta nell'header un identificatore di circuito virtuale (VC)
- I router inoltrano i pacchetti sulla base del VC e non dell'indirizzo destinazione
- Ogni router attraversato dal circuito virtuale deve mantenere delle informazioni di stato per quel circuito
- E' possibile assegnare delle risorse (banda trasmissiva, buffer) a ciascun VC → possibilità di prestazioni predibili



# Forwarding table (rete a circuiti virtuali)

- Un circuito virtuale è identificato da un numero VC su ogni tratto (link) del percorso
- Lo stesso circuito sarà identificato da numeri di VC diversi sui vari tratti che lo compongono

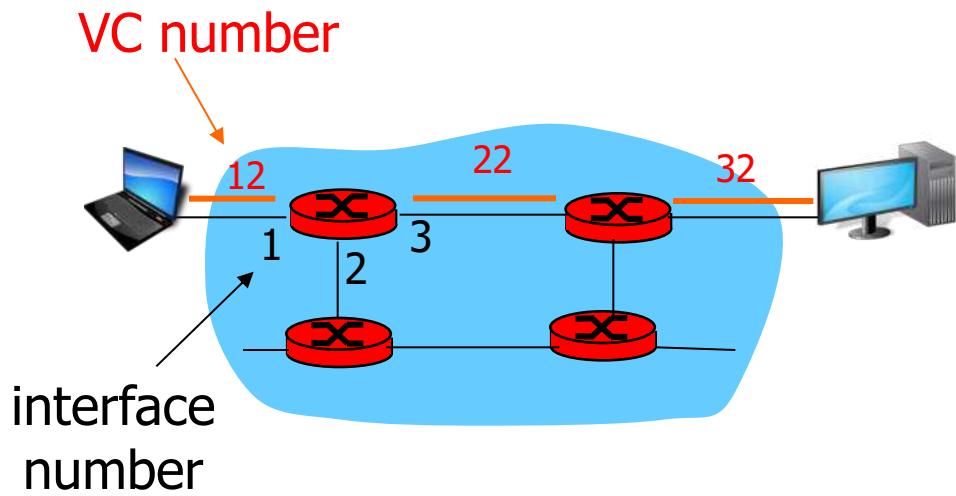


Tabella di forwarding nel router in alto a sx:

Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
...	...	...	...



# Datagrammi vs circuiti virtuali

Proprietà	datagrammi	circuiti virtuali
Creazione del circuito	Non richiesta	Richiesta
Indirizzamento	Ogni pacchetto contiene l'intero indirizzo della sorgente e della destinazione	Ogni pacchetto contiene un numero di VC
Informazioni sullo stato	I nodi di rete non mantengono informazioni sullo stato	Ogni VC richiede uno spazio di memoria sui nodi
Instradamento	Ogni pacchetto è instradato indipendentemente	Percorso pre-calcolato: ogni pacchetto segue questo percorso
Effetti di guasti ai nodi	Nessuno (solo i pacchetti persi durante il guasto)	Tutti i VC che attraversano quel nodo sono chiusi
Controllo di congestione	Complicato	Semplice se possiamo allocare spazio sufficiente per ogni VC



# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

**ARP – RARP – DHCP**

**I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso**



## Nota di Copyright

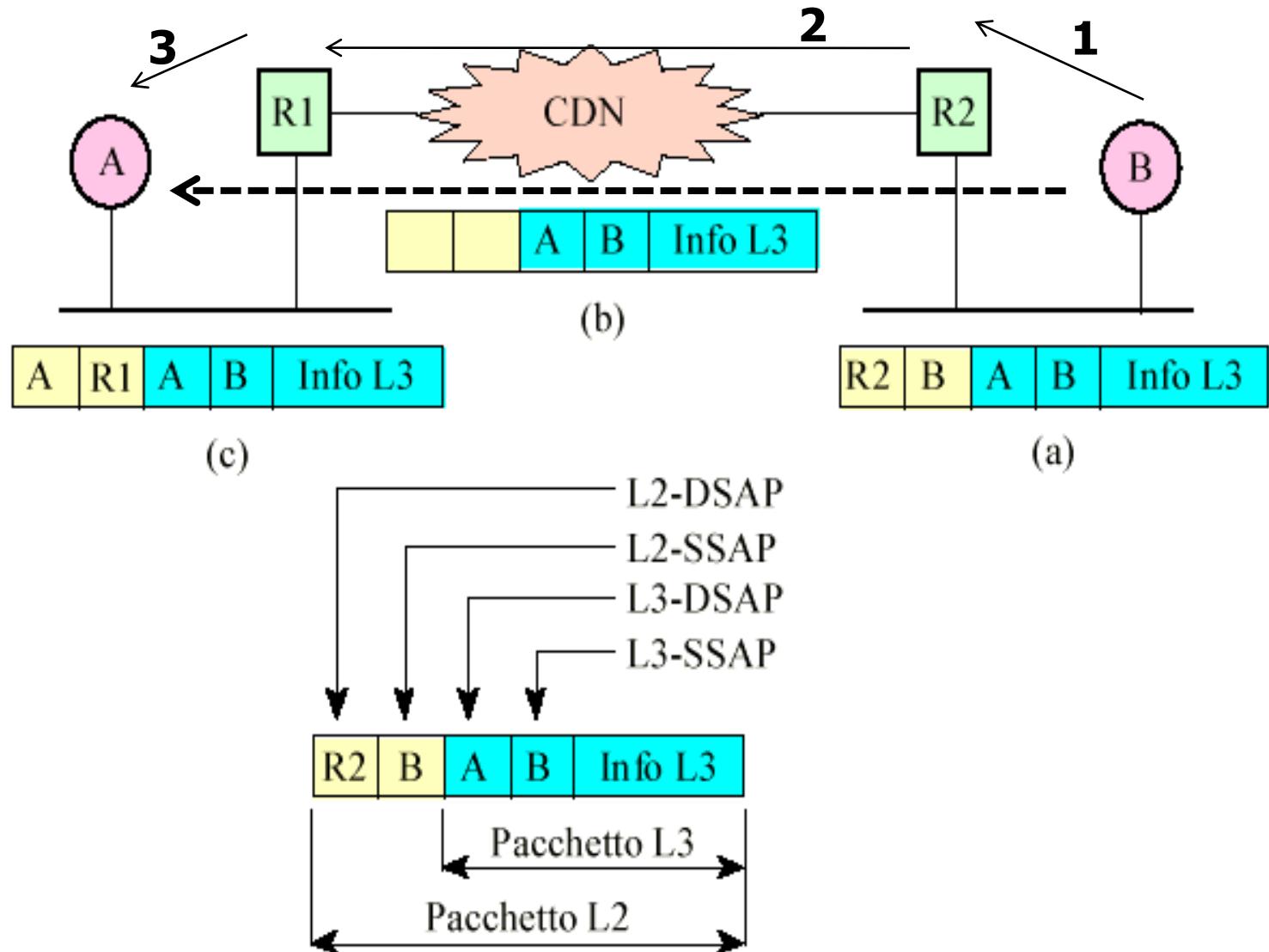
Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre; Alessio Botta



# Indirizzi IP ed Indirizzi di Livello 2





# Problema della risoluzione dell'indirizzo

- Due host possono comunicare direttamente solo se sono collegati alla stessa rete fisica
  - Per potersi scambiare informazioni devono conoscere i rispettivi indirizzi fisici
- Il protocollo IP consente di individuare univocamente un host tramite un indirizzo logico (indirizzo IP)
  - Tutte le applicazioni usano gli indirizzi logici ed ignorano la rete fisica. Ma per inviare un messaggio occorre necessariamente conoscere anche l'indirizzo fisico
  - Pertanto, serve un meccanismo di corrispondenza tra gli indirizzi logici e gli indirizzi fisici. Tale meccanismo è offerto dal protocollo ARP



# ARP - Address Resolution Protocol

- Uno scenario tipico
  - A deve spedire un datagram a *B*, host appartenente alla medesima rete logica (cioè, alla medesima rete IP)
  - A conosce l'indirizzo IP di *B*, ma non il suo indirizzo fisico
- Soluzione tramite ARP
  - A manda in broadcast a tutti gli host della rete un pacchetto contenente l'indirizzo di rete di *B*, allo scopo di conoscere l'indirizzo fisico di *B*
  - *B* riconosce il suo indirizzo di rete e risponde ad A
  - Finalmente A conosce l'indirizzo fisico di *B*, quindi può spedire il datagram a *B*



# Formato del pacchetto ARP

Hardware Type	Protocol Type
HLEN	PLEN
Operation	
Sender Hardware Address	
Sender HW Address	Sender IP Address
Sender IP Address	Target HW Address
Target Hardware Address	
Target IP Address	



# Incapsulamento dei pacchetti ARP

- Il protocollo ARP interagisce direttamente con il livello data link
- Il pacchetto ARP viene incapsulato in un frame e spedito in broadcast sulla rete
  - L'header del frame di livello 2 specifica che il frame contiene un pacchetto ARP



# Esempio: richiesta ARP

No..	Time	Source	Destination	Protocol	Info
1	0.000000	java.comics.unina.it	ff:ff:ff:ff:ff:ff	ARP	who has 143.225.229.3? Tell 143.225.229.186
2	0.000239	grid.grid.unina.it	java.comics.unina.it	ARP	143.225.229.3 is at 00:90:27:d0:bb:56

Frame 1 (42 on wire, 42 captured)

Ethernet II

Destination: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)

Source: 00:08:0d:6a:a3:07 (java.comics.unina.it)

Type: ARP (0x0806)

Address Resolution Protocol (request)

Hardware type: Ethernet (0x0001)

Protocol type: IP (0x0800)

Hardware size: 6

Protocol size: 4

Opcode: request (0x0001)

Sender MAC address: 00:08:0d:6a:a3:07 (java.comics.unina.it)

Sender IP address: java.comics.unina.it (143.225.229.186)

Target MAC address: 00:00:00:00:00:00 (grid.grid.unina.it)

Target IP address: grid.grid.unina.it (143.225.229.3)



# Esempio: risposta ARP

No..	Time	Source	Destination	Protocol	Info
1	0.000000	java.comics.unina.it	ff:ff:ff:ff:ff:ff	ARP	who has 143.225.229.3? Tell 143.225.229.186
2	0.000239	grid.grid.unina.it	java.comics.unina.it	ARP	143.225.229.3 is at 00:90:27:d0:bb:56

Frame 2 (60 on wire, 60 captured)

Ethernet II

Destination: 00:08:0d:6a:a3:07 (java.comics.unina.it)  
Source: 00:90:27:d0:bb:56 (grid.grid.unina.it)  
Type: ARP (0x0806)  
Trailer: 00000000000000000000000000000000...

Address Resolution Protocol (reply)

Hardware type: Ethernet (0x0001)  
Protocol type: IP (0x0800)  
Hardware size: 6  
Protocol size: 4  
Opcode: reply (0x0002)  
Sender MAC address: 00:90:27:d0:bb:56 (grid.grid.unina.it)  
Sender IP address: grid.grid.unina.it (143.225.229.3)  
Target MAC address: 00:08:0d:6a:a3:07 (java.comics.unina.it)  
Target IP address: java.comics.unina.it (143.225.229.186)

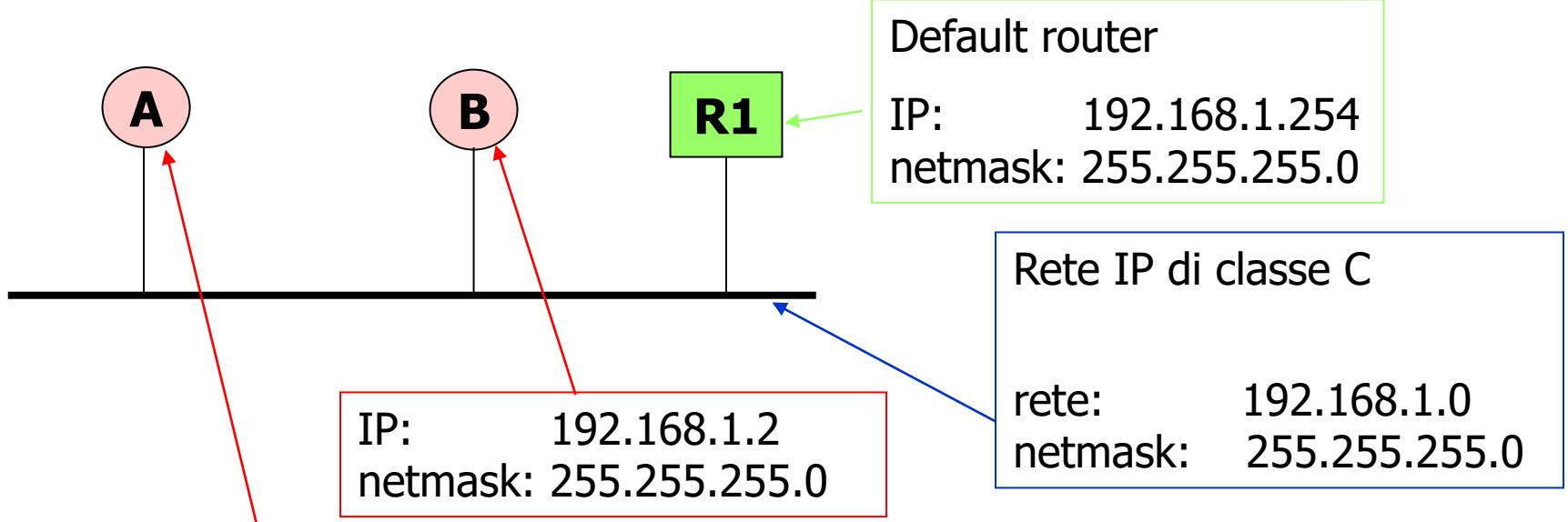


# ARP: scenari tipici

- **Primo caso:** l'host destinazione è sulla stessa LAN (stessa subnet IP)
- **Secondo caso:** l'host destinazione non è sulla stessa LAN (subnet IP)



# ARP: primo caso A→B (1/3)



**A** ha intenzione di inviare un pacchetto a **B**. Prima Domanda: come fa **A** a sapere se **B** è sulla propria sottorete?

Risposta: attraverso la netmask!



## ARP: primo caso (2/3)

- Ogni computer ha un indirizzo IP ed una netmask. La netmask serve ad individuare la propria sottorete IP
  - Digitare da una shell windows il comando
    - ipconfig /all
  - Il computer **A** esegue una AND tra l'indirizzo IP destinazione e la propria netmask.
    - Nel caso precedente

E' proprio l'indirizzo della sottorete IP cui appartiene A

$$\begin{array}{ll} \text{IP di B} & 192.168.1.2 \\ \text{AND} & \\ \text{netmask A} & 255.255.255.0 \\ = & \\ & 192.168.1.0 \end{array}$$

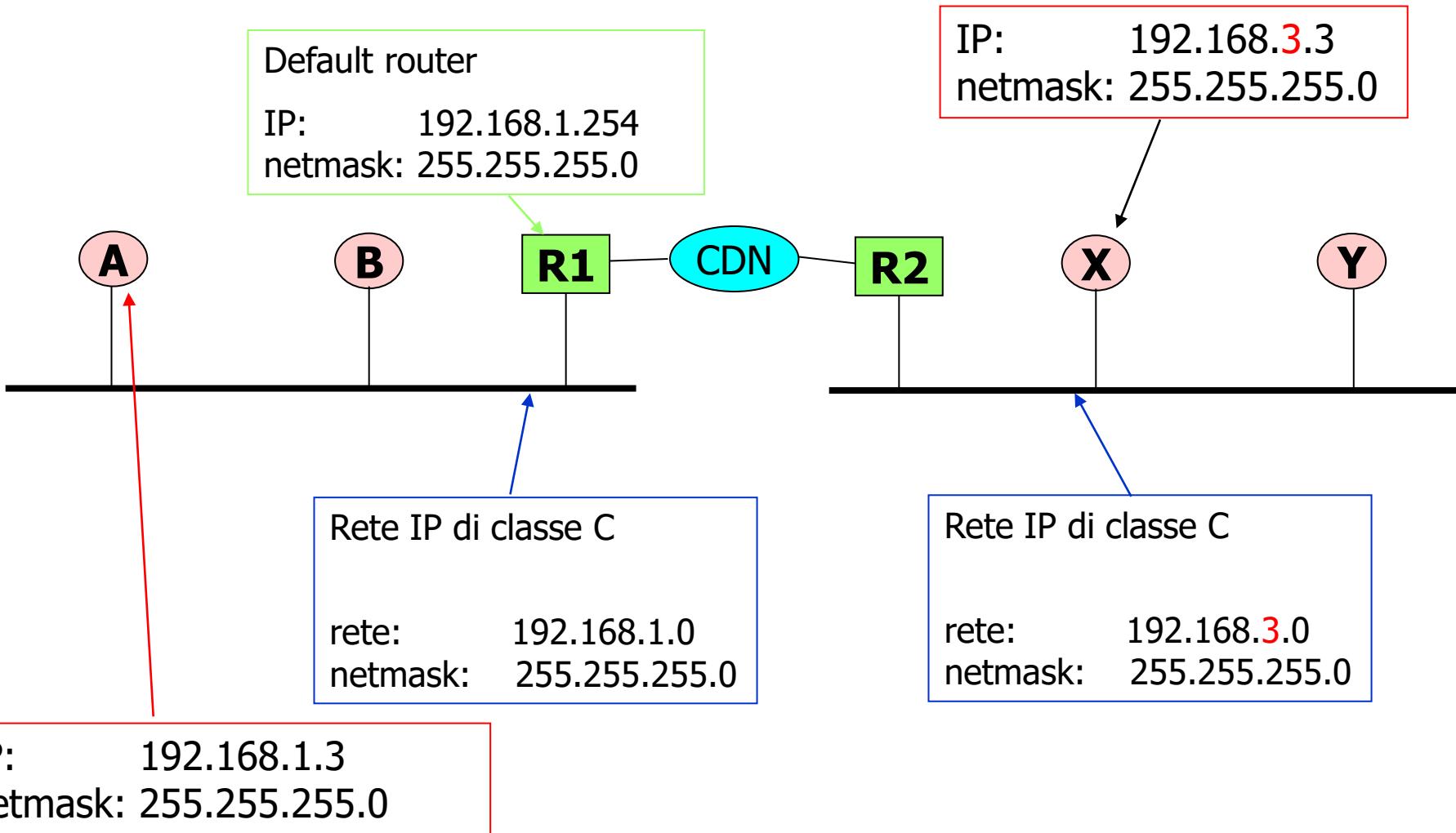


## ARP: primo caso (3/3)

- Se il computer **B** è sulla stessa sottorete IP
  - allora mando un pacchetto *ARP request* in broadcast
    - tale pacchetto contiene, nel campo **DEST IP**, l'indirizzo IP di B



# ARP: secondo caso A→X (1/2)





## ARP: secondo caso (2/2)

- Se A intende mandare un pacchetto a X, l'operazione di AND tra la netmask e l'indirizzo IP DEST fornisce un risultato differente

Non è l'indirizzo della sottorete cui appartiene A → Occorre inviare il pacchetto al router.

$$\begin{array}{ll} \text{IP di X} & 192.168.3.3 \\ & \text{AND} \\ \text{netmask A} & 255.255.255.0 \\ & = \\ & 192.168.3.0 \end{array}$$

In questo caso, pertanto, si prepara un pacchetto ARP in cui si specifica come indirizzo IP DEST proprio l'indirizzo IP del router



# ARP: ricapitolando...

- Operazione di AND logico tra l'indirizzo IP della destinazione e la propria netmask
  - Se il risultato fornisce l'indirizzo della propria subnet IP
    - Invia una richiesta ARP per risolvere l'indirizzo della destinazione
  - ...altrimenti
    - Il pacchetto deve essere inviato al router di default
      - Nel caso in cui l'indirizzo MAC del router non sia noto
        - » Invia una richiesta ARP per risolvere l'indirizzo IP del router



# Raffinamenti del protocollo

- Per ridurre il traffico sulla rete, ogni host mantiene una cache con le corrispondenze tra indirizzi logici e fisici
  - Prima di spedire una richiesta ARP controlla nella cache
- Il pacchetto ARP contiene indirizzo fisico e logico del mittente
  - Gli host che leggono il pacchetto possono aggiornare le loro ARP cache



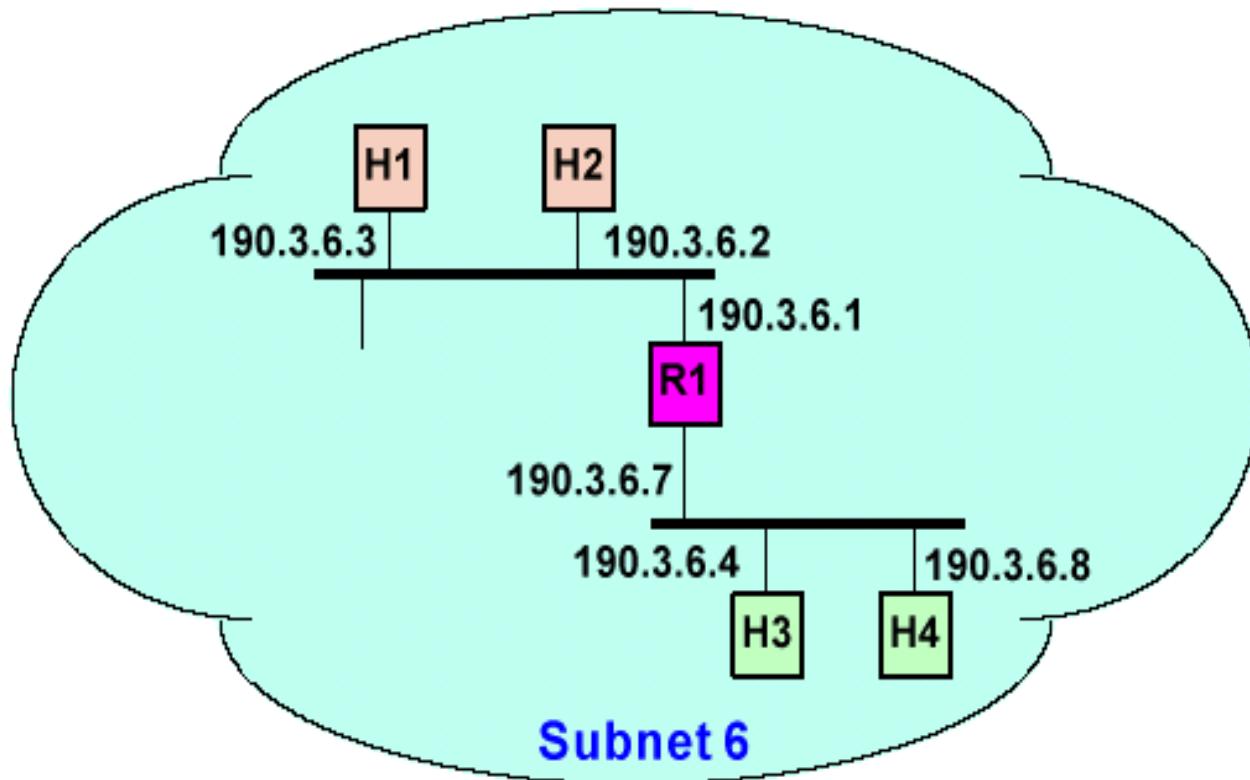
# Monitoraggio di ARP

- Con il comando *arp* è possibile leggere e modificare il contenuto della arp cache
  - **arp -a** (legge il contenuto di tutta la cache)
- Con il comando *tcpdump* (o con un software tipo Ethereal...) è possibile monitorare tutto il traffico che viaggia sulla rete
  - È possibile filtrare solo i pacchetti spediti da un dato protocollo su una data interfaccia
  - **tcpdump arp** (legge solo i pacchetti arp)



# Proxy ARP

- Permette di usare la stessa subnet su due o più reti fisiche diverse





# Reverse ARP

- Il protocollo RARP svolge il ruolo opposto ad ARP
  - fisico → logico
- Usato per sistemi diskless
  - X terminal, diskless workstation
  - Al boot non conoscono il loro indirizzo IP



# Scenario RARP

- A conosce il proprio indirizzo MAC, ma non conosce il proprio indirizzo IP
- L'host B (server RARP) conosce l'indirizzo IP di A
- Soluzione
  - RARP request sulla rete (in broadcast)
  - B risponde con un messaggio RARP reply contenente l'indirizzo IP di A

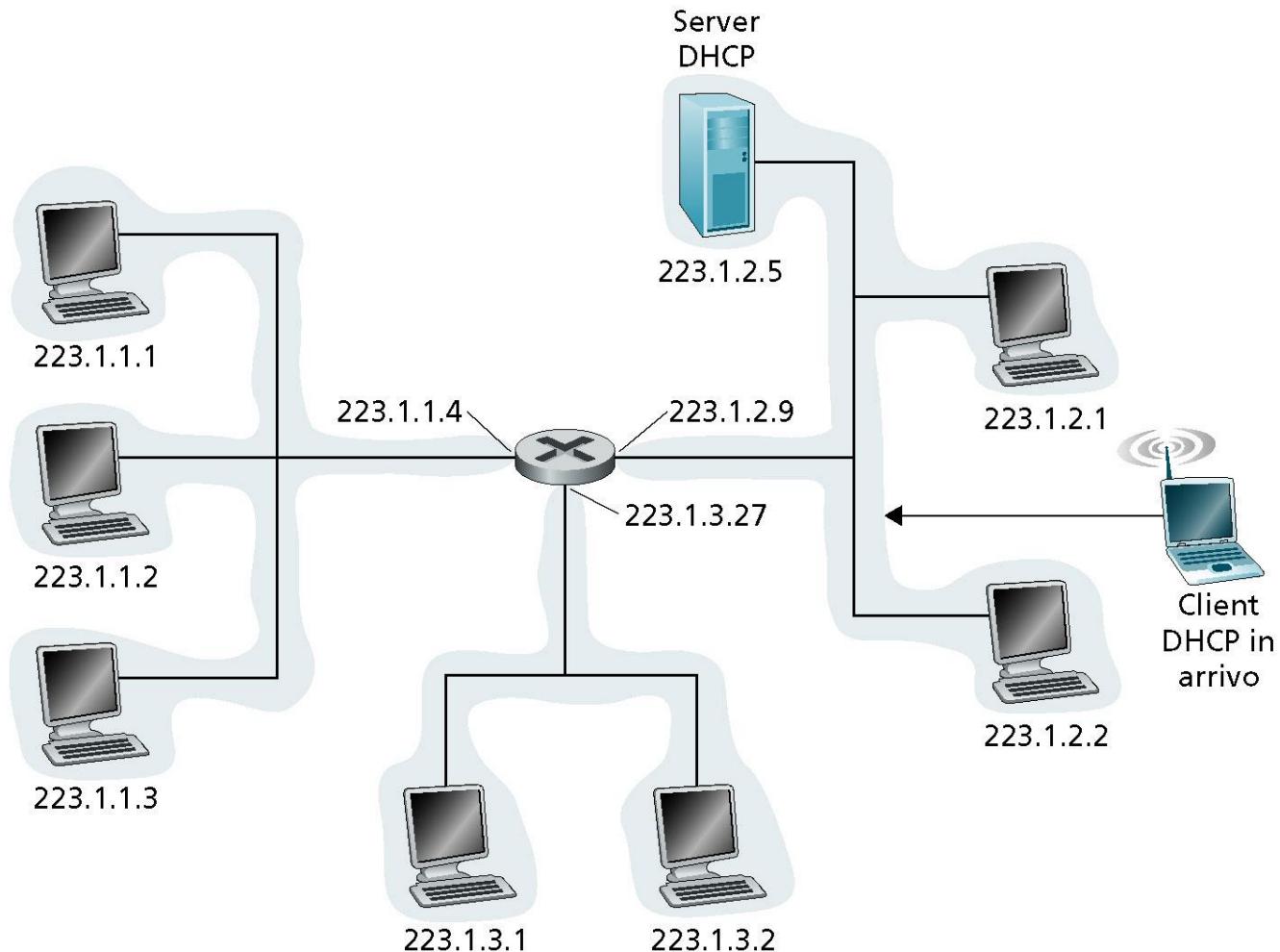


# Altre soluzioni per boot remoto

- Il protocollo RARP è stato sostituito da altri protocolli più flessibili e potenti
  - BOOTP: **BOOT**strap **P**rotocol
  - DHCP: **D**ynamic **H**ost **C**onfiguration **P**rotocol
  - Utilizzati per assegnare dinamicamente gli indirizzi agli host di una rete IP

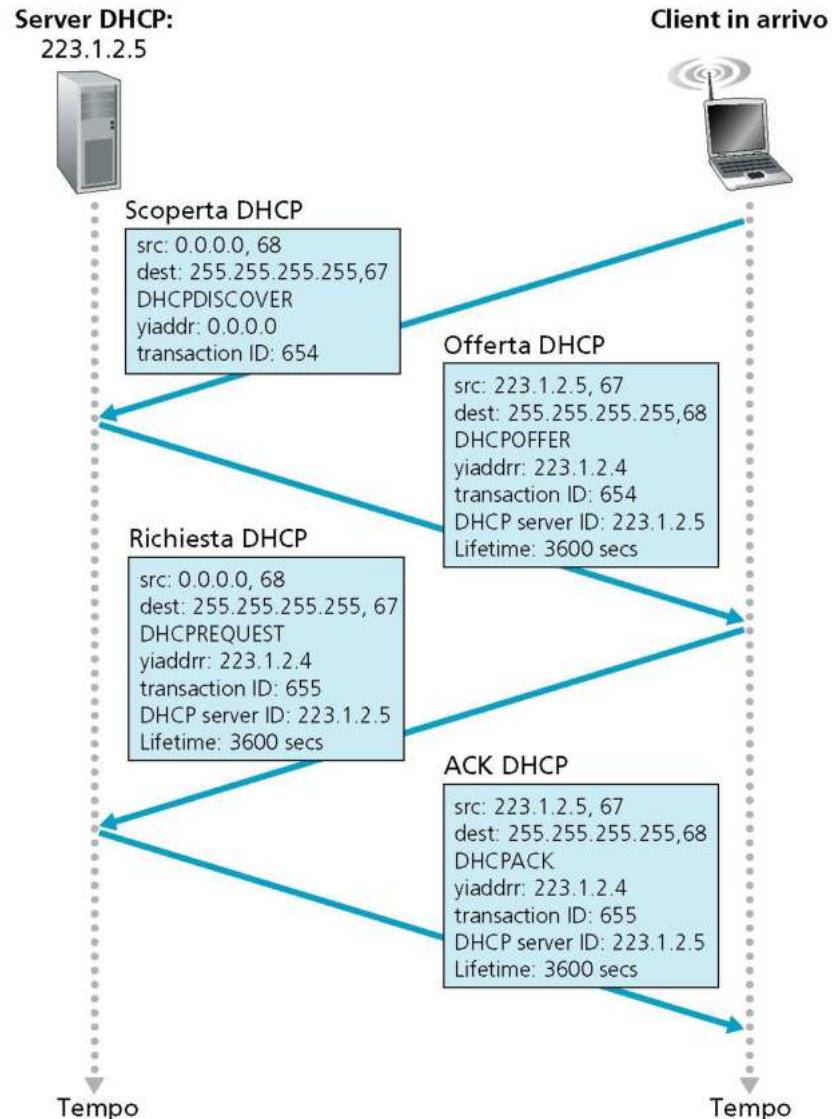


# DHCP: scenario tipico





# Interazione client-server via DHCP





# DHCP discover

No..	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xb2c065b
2	0.004628	192.168.2.1	192.168.2.13	DHCP	DHCP Offer - Transaction ID 0xb2c065b
3	0.005392	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xb2c065b
4	0.009656	192.168.2.1	192.168.2.13	DHCP	DHCP ACK - Transaction ID 0xb2c065b
					.....

Frame 1 (342 on wire, 342 captured)

Ethernet II

Destination: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)  
Source: 00:02:2d:09:17:be (Agere\_09:17:be)  
Type: IP (0x0800)

Internet Protocol, Src Addr: 0.0.0.0 (0.0.0.0), Dst Addr: 255.255.255.255 (255.255.255.255)

User Datagram Protocol, Src Port: bootpc (68), Dst Port: bootps (67)

Source port: bootpc (68)  
Destination port: bootps (67)  
Length: 308  
Checksum: 0xae22 (correct)

Bootstrap Protocol

Message type: Boot Request (1)  
Hardware type: Ethernet  
Hardware address length: 6  
Hops: 0  
Transaction ID: 0xb2c065b  
Seconds elapsed: 0  
Broadcast flag: 0x0000  
Client IP address: 0.0.0.0 (0.0.0.0)  
Your (client) IP address: 0.0.0.0 (0.0.0.0)  
Next server IP address: 0.0.0.0 (0.0.0.0)  
Relay agent IP address: 0.0.0.0 (0.0.0.0)  
Client hardware address: 00:02:2d:09:17:be  
Server host name not given  
Boot file name not given  
Magic cookie: (OK)  
Option 53: DHCP Message Type = DHCP Discover  
Option 116: DHCP Auto-Configuration (1 bytes)

Option 61: Client identifier

Option 50: Requested IP Address = 192.168.2.13  
Option 12: Host Name = "java"  
Option 60: Vendor class identifier = "MSFT 5.0"

Option 55: Parameter Request List

End Option  
Padding



# DHCP offer

No..	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xb2c065b
2	0.004628	192.168.2.1	192.168.2.13	DHCP	DHCP Offer - Transaction ID 0xb2c065b
3	0.005392	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xb2c065b
4	0.009656	192.168.2.1	192.168.2.13	DHCP	DHCP ACK - Transaction ID 0xb2c065b
					.....

Frame 2 (590 on wire, 590 captured)

Ethernet II

Destination: 00:02:2d:09:17:be (Agere\_09:17:be)  
Source: 00:30:bd:96:28:fa (BELKIN\_96:28:fa)  
Type: IP (0x0800)

Internet Protocol, Src Addr: 192.168.2.1 (192.168.2.1), Dst Addr: 192.168.2.13 (192.168.2.13)

User Datagram Protocol, Src Port: bootps (67), Dst Port: bootpc (68)

Source port: bootps (67)  
Destination port: bootpc (68)  
Length: 556  
Checksum: 0xc29a (correct)

Bootstrap Protocol

Message type: Boot Reply (2)  
Hardware type: Ethernet  
Hardware address length: 6  
Hops: 0  
Transaction ID: 0xb2c065b  
Seconds elapsed: 0  
Broadcast flag: 0x0000  
Client IP address: 0.0.0.0 (0.0.0.0)  
Your (client) IP address: 192.168.2.13 (192.168.2.13)  
Next server IP address: 0.0.0.0 (0.0.0.0)  
Relay agent IP address: 0.0.0.0 (0.0.0.0)  
Client hardware address: 00:02:2d:09:17:be  
Server host name not given  
Boot file name not given  
Magic cookie: (OK)  
Option 53: DHCP Message Type = DHCP Offer  
Option 54: Server Identifier = 192.168.2.1  
Option 51: IP Address Lease Time = 12427 days, 7 hours, 45 minutes, 41 seconds  
Option 1: Subnet Mask = 255.255.255.0  
Option 3: Router = 192.168.2.1

Option 6: Domain Name Server

IP Address: 217.9.64.200  
IP Address: 217.9.64.220  
IP Address: 217.9.64.3  
Option 15: Domain Name = "napoli.consorzio-cini.it"  
Option 44: NetBIOS over TCP/IP Name Server = 217.9.64.200  
End Option  
Padding



# DHCP request

File Edit Capture Display Tools					
No..	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xb2c065b
2	0.004628	192.168.2.1	192.168.2.13	DHCP	DHCP Offer - Transaction ID 0xb2c065b
3	0.005392	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xb2c065b
4	0.009656	192.168.2.1	192.168.2.13	DHCP	DHCP ACK - Transaction ID 0xb2c065b

Frame 3 (361 on wire, 361 captured)

Ethernet II

    Destination: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)  
    Source: 00:02:2d:09:17:be (Agere\_09:17:be)  
    Type: IP (0x0800)

Internet Protocol, Src Addr: 0.0.0.0 (0.0.0.0), Dst Addr: 255.255.255.255 (255.255.255.255)

User Datagram Protocol, Src Port: bootpc (68), Dst Port: bootps (67)

Bootstrap Protocol

    Message type: Boot Request (1)  
    Hardware type: Ethernet  
    Hardware address length: 6  
    Hops: 0  
    Transaction ID: 0xb2c065b  
    Seconds elapsed: 0  
    Broadcast flag: 0x0000  
    Client IP address: 0.0.0.0 (0.0.0.0)  
    Your (client) IP address: 0.0.0.0 (0.0.0.0)  
    Next server IP address: 0.0.0.0 (0.0.0.0)  
    Relay agent IP address: 0.0.0.0 (0.0.0.0)  
    Client hardware address: 00:02:2d:09:17:be  
    Server host name not given  
    Boot file name not given  
    Magic cookie: (OK)  
    Option 53: DHCP Message Type = DHCP Request

Option 61: Client identifier

    Hardware type: Ethernet  
    Client hardware address: 00:02:2d:09:17:be

Option 50: Requested IP Address = 192.168.2.13

Option 54: Server Identifier = 192.168.2.1

Option 12: Host Name = "java"

Option 81: Client Fully Qualified Domain Name (23 bytes)

Option 60: Vendor class identifier = "MSFT 5.0"

Option 55: Parameter Request List

    1 = Subnet Mask  
    15 = Domain Name  
    3 = Router  
    6 = Domain Name Server  
    44 = NetBIOS over TCP/IP Name Server  
    46 = NetBIOS over TCP/IP Node Type  
    47 = NetBIOS over TCP/IP Scope  
    31 = Perform Router Discover  
    33 = Static Route  
    Unknown Option Code: 249  
    43 = Vendor-Specific Information

End Option



# DHCP ACK

No.	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xb2c065b
2	0.004628	192.168.2.1	192.168.2.13	DHCP	DHCP Offer - Transaction ID 0xb2c065b
3	0.005392	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xb2c065b
4	0.009656	192.168.2.1	192.168.2.13	DHCP	DHCP ACK - Transaction ID 0xb2c065b

Frame 4 (590 on wire, 590 captured)

Ethernet II

    Destination: 00:02:2d:09:17:be (Agere\_09:17:be)  
    Source: 00:30:bd:96:28:fa (BELKIN\_96:28:fa)  
    Type: IP (0x0800)

Internet Protocol, Src Addr: 192.168.2.1 (192.168.2.1), Dst Addr: 192.168.2.13 (192.168.2.13)

User Datagram Protocol, Src Port: bootps (67), Dst Port: bootpc (68)

Bootstrap Protocol

    Message type: Boot Reply (2)  
    Hardware type: Ethernet  
    Hardware address length: 6  
    Hops: 0  
    Transaction ID: 0xb2c065b  
    Seconds elapsed: 0  
    Broadcast flag: 0x0000  
    Client IP address: 0.0.0.0 (0.0.0.0)  
    Your (client) IP address: 192.168.2.13 (192.168.2.13)  
    Next server IP address: 0.0.0.0 (0.0.0.0)  
    Relay agent IP address: 0.0.0.0 (0.0.0.0)  
    Client hardware address: 00:02:2d:09:17:be  
    Server host name not given  
    Boot file name not given  
    Magic cookie: (OK)  
    Option 53: DHCP Message Type = DHCP ACK  
    Option 54: Server Identifier = 192.168.2.1  
    Option 51: IP Address Lease Time = 12427 days, 13 hours, 37 minutes, 3 seconds  
    Option 1: Subnet Mask = 255.255.255.0  
    Option 3: Router = 192.168.2.1

Domain Name Server

    IP Address: 217.9.64.200  
    IP Address: 217.9.64.220  
    IP Address: 217.9.64.3

    Option 15: Domain Name = "napoli.consorzio-cini.it"  
    Option 44: NetBIOS over TCP/IP Name Server = 217.9.64.200  
    End Option  
    Padding



# Domanda

- C'è qualche differenza tra lo schema di funzionamento del DHCP riportato nella slide 24 e le catture wireshark riportate dopo?
  - Confrontate sia le catture riportate nelle slide sia quelle fatte sui vostri PC
  - Fatemi sapere via mail...



# Corso di Laurea in Informatica

## Corso di Reti di Calcolatori 1

Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))

### ICMP: ping e traceroute

I lucidi presentati al corso sono uno strumento didattico  
che **NON** sostituisce i testi indicati nel programma del corso



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonico, Giorgio Ventre, Alessio Botta



# ICMP (Internet Control Message Protocol)

- Funzionalità
  - Verificare lo stato della rete
    - echo request, echo reply
  - Riportare anomalie
    - destination unreachable
    - time exceeded
    - parameter problem
  - Scoprire la netmask
    - mask request
    - address mask reply
  - Migliorare il routing
    - Redirect
- **Messaggi ICMP trasportati in datagrammi IP con Protocol Type = 0x01**



# ICMP

- I messaggi sono individuati da un tipo e da un codice

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header



- Applicazioni
  - Ping
    - Utilizzato per verificare la connettività a livello rete tra due host, A e B
      - l'host A invia un pacchetto “echo request”
      - alla ricezione di tale messaggio, l'host B risponde con un pacchetto “echo reply”
  - Traceroute
    - Utilizzato per scoprire il percorso seguito per raggiungere una certa destinazione
    - Viene inviata una serie di pacchetti con TTL via via crescente, a partire da 1
      - il router che, decrementando il TTL, lo azzera invierà indietro un messaggio “time exceeded”
        - » in questo modo si riesce a determinare il percorso fino alla destinazione



# Esempio di traceroute (1/9)

```
C:\Documents and Settings\spromano>tracert 143.225.229.3
```

Rilevazione instradamento verso grid.grid.unina.it [143.225.229.3]  
su un massimo di 30 punti di passaggio:

1	2 ms	2 ms	2 ms	192.168.2.1
2	4 ms	8 ms	9 ms	217.9.64.193
3	11 ms	19 ms	19 ms	192.55.101.129
4	3 ms	3 ms	3 ms	grid.grid.unina.it [143.225.229.3]

Rilevazione completata.



# Esempio di traceroute (2/9)

<capture> - Ethereal

File Edit Capture Display Tools Help

No.	Time	Source	Destination	Protocol	Info
1	0.000000	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request

Frame 1 (106 on wire, 106 captured)

Ethernet II

- Destination: 00:30:bd:96:28:fa (BELKIN\_96:28:fa)
- Source: 00:02:2d:09:17:be (java.comics.unina.it)
- Type: IP (0x0800)

Internet Protocol, Src Addr: java.comics.unina.it (192.168.2.6), Dst Addr: grid.grid.unina.it (143.225.229.3)

- Version: 4
- Header length: 20 bytes
- Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
- Total Length: 92
- Identification: 0x2042
- Flags: 0x00
- Fragment offset: 0
- Time to live: 1
- Protocol: ICMP (0x01)
- Header checksum: 0x61cc (correct)
- Source: java.comics.unina.it (192.168.2.6)
- Destination: grid.grid.unina.it (143.225.229.3)

Internet Control Message Protocol

- Type: 8 (Echo (ping) request)
- Code: 0
- Checksum: 0xaeaff (correct)
- Identifier: 0x0300
- Sequence number: 46:00
- Data (64 bytes)



# Esempio di traceroute (3/9)

<capture> - Ethereal

File Edit Capture Display Tools Help

No.	Time	Source	Destination	Protocol	Info
1	0.000000	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request
2	0.002111	192.168.2.1	java.comics.unina.it	ICMP	Time-to-live exceeded

Frame 2 (134 on wire, 134 captured)  
Ethernet II  
Internet Protocol, Src Addr: 192.168.2.1 (192.168.2.1), Dst Addr: java.comics.unina.it (192.168.2.6)  
Internet Control Message Protocol  
Type: 11 (Time-to-live exceeded)  
Code: 0 (TTL equals 0 during transit)  
Checksum: 0xf4ff (correct)  
Internet Protocol, Src Addr: java.comics.unina.it (192.168.2.6), Dst Addr: grid.grid.unina.it (143.225.229.3)  
Version: 4  
Header length: 20 bytes  
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)  
Total Length: 92  
Identification: 0x2042  
Flags: 0x00  
Fragment offset: 0  
Time to live: 1  
Protocol: ICMP (0x01)  
Header checksum: 0x61cc (correct)  
Source: java.comics.unina.it (192.168.2.6)  
Destination: grid.grid.unina.it (143.225.229.3)  
Internet Control Message Protocol  
Type: 8 (Echo (ping) request)  
Code: 0  
Checksum: 0xaeff (correct)  
Identifier: 0x0300  
Sequence number: 46:00  
Data (64 bytes)



# Esempio di traceroute (4/9)

<capture> - Ethereal

File Edit Capture Display Tools Help

No.	Time	Source	Destination	Protocol	Info
5	0.004468	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request
6	0.006490	192.168.2.1	java.comics.unina.it	ICMP	Time-to-live exceeded
7	1.006318	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request

Frame 7 (106 on wire, 106 captured)  
Ethernet II  
Internet Protocol, Src Addr: java.comics.unina.it (192.168.2.6), Dst Addr: grid.grid.unina.it (143.225.229.3)  
Version: 4  
Header length: 20 bytes  
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)  
Total Length: 92  
Identification: 0x2045  
Flags: 0x00  
Fragment offset: 0  
Time to live: 2  
Protocol: ICMP (0x01)  
Header checksum: 0x60c9 (correct)  
Source: java.comics.unina.it (192.168.2.6)  
Destination: grid.grid.unina.it (143.225.229.3)  
Internet Control Message Protocol  
Type: 8 (Echo (ping) request)  
Code: 0  
Checksum: 0xabff (correct)  
Identifier: 0x0300  
Sequence number: 49:00  
Data (64 bytes)



# Esempio di traceroute (5/9)

Screenshot of the Ethereal network traffic analyzer showing a trace route from a local host to a destination host.

The table below shows the captured frames:

No.	Time	Source	Destination	Protocol	Info
5	0.004468	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request
6	0.006490	192.168.2.1	java.comics.unina.it	ICMP	Time-to-live exceeded
7	1.006318	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request
8	1.011137	217.9.64.193	java.comics.unina.it	ICMP	Time-to-live exceeded

The details for Frame 8 (highlighted in blue) are as follows:

- Frame 8 (70 on wire, 70 captured)
- Ethernet II
- Internet Protocol, Src Addr: 217.9.64.193 (217.9.64.193), Dst Addr: java.comics.unina.it (192.168.2.6)
- Internet Control Message Protocol
  - Type: 11 (Time-to-live exceeded)
  - Code: 0 (TTL equals 0 during transit)
  - Checksum: 0x4fff (correct)
- Internet Protocol, Src Addr: java.comics.unina.it (192.168.2.6), Dst Addr: grid.grid.unina.it (143.225.229.3)
  - Version: 4
  - Header length: 20 bytes
  - Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  - Total Length: 92
  - Identification: 0x2045
  - Flags: 0x00
    - Fragment offset: 0
    - Time to live: 1
    - Protocol: ICMP (0x01)
    - Header checksum: 0x61c9 (correct)
    - Source: java.comics.unina.it (192.168.2.6)
    - Destination: grid.grid.unina.it (143.225.229.3)
- Internet Control Message Protocol
  - Type: 8 (Echo (ping) request)
  - Code: 0
  - Checksum: 0xabff (correct)
  - Identifier: 0x0300
  - Sequence number: 49:00



# Esempio di traceroute (6/9)

<capture> - Ethereal

File Edit Capture Display Tools Help

No..	Time	Source	Destination	Protocol	Info
13	2.023553	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request

Frame 13 (106 on wire, 106 captured)  
Ethernet II  
Internet Protocol, Src Addr: java.comics.unina.it (192.168.2.6), Dst Addr: grid.grid.unina.it (143.225.229.3)  
Version: 4  
Header length: 20 bytes  
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)  
Total Length: 92  
Identification: 0x2049  
Flags: 0x00  
Fragment offset: 0  
Time to live: 3  
Protocol: ICMP (0x01)  
Header checksum: 0x5fc5 (correct)  
Source: java.comics.unina.it (192.168.2.6)  
Destination: grid.grid.unina.it (143.225.229.3)  
Internet Control Message Protocol  
Type: 8 (Echo (ping) request)  
Code: 0  
Checksum: 0xa8ff (correct)  
Identifier: 0x0300  
Sequence number: 4c:00  
Data (64 bytes)



# Esempio di traceroute (7/9)

<capture> - Ethereal

File Edit Capture Display Tools Help

No..	Time	Source	Destination	Protocol	Info
13	2.023553	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request
14	2.034492	192.55.101.129	java.comics.unina.it	ICMP	Time-to-live exceeded

Frame 14 (70 on wire, 70 captured)  
Ethernet II  
Internet Protocol, Src Addr: 192.55.101.129 (192.55.101.129), Dst Addr: java.comics.unina.it (192.168.2.6)  
Internet Control Message Protocol  
Type: 11 (Time-to-live exceeded)  
Code: 0 (TTL equals 0 during transit)  
Checksum: 0xf4ff (correct)  
Internet Protocol, Src Addr: java.comics.unina.it (192.168.2.6), Dst Addr: grid.grid.unina.it (143.225.229.3)  
Version: 4  
Header length: 20 bytes  
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)  
Total Length: 92  
Identification: 0x2049  
Flags: 0x00  
Fragment offset: 0  
Time to live: 1  
Protocol: ICMP (0x01)  
Header checksum: 0x61c5 (correct)  
Source: java.comics.unina.it (192.168.2.6)  
Destination: grid.grid.unina.it (143.225.229.3)  
Internet Control Message Protocol  
Type: 8 (Echo (ping) request)  
Code: 0  
Checksum: 0xa8ff (correct)  
Identifier: 0x0300  
Sequence number: 4c:00



# Esempio di traceroute (8/9)

<capture> - Ethereal

File Edit Capture Display Tools Help

No..	Time	Source	Destination	Protocol	Info
15	2.034848	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request
16	2.054220	192.55.101.129	java.comics.unina.it	ICMP	Time-to-live exceeded
17	2.054538	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request
18	2.074440	192.55.101.129	java.comics.unina.it	ICMP	Time-to-live exceeded
19	16.070350	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request

Frame 19 (106 on wire, 106 captured)

Ethernet II

Internet Protocol, Src Addr: java.comics.unina.it (192.168.2.6), Dst Addr: grid.grid.unina.it (143.225.229.3)  
Version: 4  
Header length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)  
Total Length: 92  
Identification: 0x2053

Flags: 0x00  
Fragment offset: 0  
Time to live: 4  
Protocol: ICMP (0x01)  
Header checksum: 0x5ebb (correct)  
Source: java.comics.unina.it (192.168.2.6)  
Destination: grid.grid.unina.it (143.225.229.3)

Internet Control Message Protocol  
Type: 8 (Echo (ping) request)  
Code: 0  
Checksum: 0xa5ff (correct)  
Identifier: 0x0300  
Sequence number: 4f:00  
Data (64 bytes)



# Esempio di traceroute (9/9)

<capture> - Ethereal

File Edit Capture Display Tools Help

No..	Time	Source	Destination	Protocol	Info
16	2.054220	192.55.101.129	java.comics.unina.it	ICMP	Time-to-live exceeded
17	2.054538	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request
18	2.074440	192.55.101.129	java.comics.unina.it	ICMP	Time-to-live exceeded
19	16.070350	java.comics.unina.it	grid.grid.unina.it	ICMP	Echo (ping) request
20	16.074284	grid.grid.unina.it	java.comics.unina.it	ICMP	Echo (ping) reply

Frame 20 (106 on wire, 106 captured)  
Ethernet II  
Internet Protocol, Src Addr: grid.grid.unina.it (143.225.229.3), Dst Addr: java.comics.unina.it (192.168.2.6)  
Version: 4  
Header length: 20 bytes  
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)  
Total Length: 92  
Identification: 0xa0b4  
Flags: 0x00  
Fragment offset: 0  
Time to live: 252  
Protocol: ICMP (0x01)  
Header checksum: 0xe658 (correct)  
Source: grid.grid.unina.it (143.225.229.3)  
Destination: java.comics.unina.it (192.168.2.6)  
Internet Control Message Protocol  
Type: 0 (Echo (ping) reply)  
Code: 0  
Checksum: 0xadff (correct)  
Identifier: 0x0300  
Sequence number: 4f:00  
Data (64 bytes)



# Corso di Laurea in Informatica

## Corso di Reti di Calcolatori 1

Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))

### Routing

—

### Parte prima

I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso

# Nota di copyright per le slide COMICS



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta



# Il ruolo dei livelli OSI

Dobbiamo Pavimentare le strade		Livello Fisico Cablaggio Strutturato
Dobbiamo scegliere il tipo di strada (Autostrada, Provinciale, Urbana,...)		Livello Data Link
Dobbiamo scegliere le indicazioni della prossima rotonda		Livello Rete
Dobbiamo scegliere come trasportare		Livello Trasporto



# Il livello rete nella pila OSI

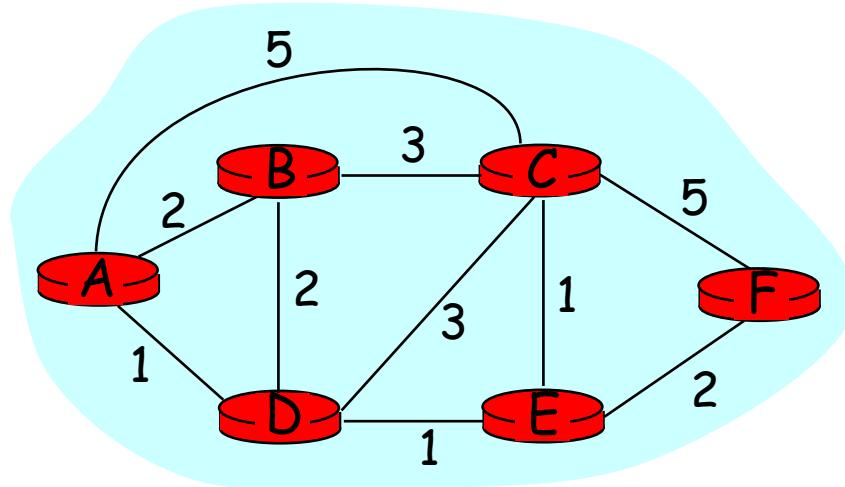




# Reti di calcolatori e grafi

Rete modellata come grafo

- **nodi** = router
- **archi** = link fisici
  - costo link
    - ritardo,
    - costo trasmissione,
    - congestione,...
- Scelta del cammino
  - cammino a costo minimo
  - altre possibilità (un cammino calcolato in base a specifici vincoli...)
- Gli algoritmi di routing fanno uso della teoria dei grafi





# Parametri del processo decisionale

- *Bandwidth*
  - capacità di un link, tipicamente definita in bit per secondo (bps)
- *Delay*
  - il tempo necessario per spedire un pacchetto da una sorgente ad una destinazione
- *Load*
  - una misura del carico di un link
- *Reliability*
  - riferita, ad esempio, all'error rate di un link
- *Hop count*
  - il numero di router da attraversare nel percorso dalla sorgente alla destinazione
- *Cost*
  - un valore arbitrario che definisce il costo di un link
  - ad esempio, costruito come funzione di diversi parametri (tra cui bandwidth, delay, packet loss, MTU,...)



# Il processo di routing

- Il processo di routing è un processo decisionale
- Ogni entità che partecipa a questo processo
  - mantiene delle informazioni
  - in base ad uno specifico algoritmo ed in funzione di determinate metriche definisce il procedimento di instradamento verso le possibili destinazioni
  - può spedire informazioni di aggiornamento alle altre entità coinvolte, secondo diversi paradigmi

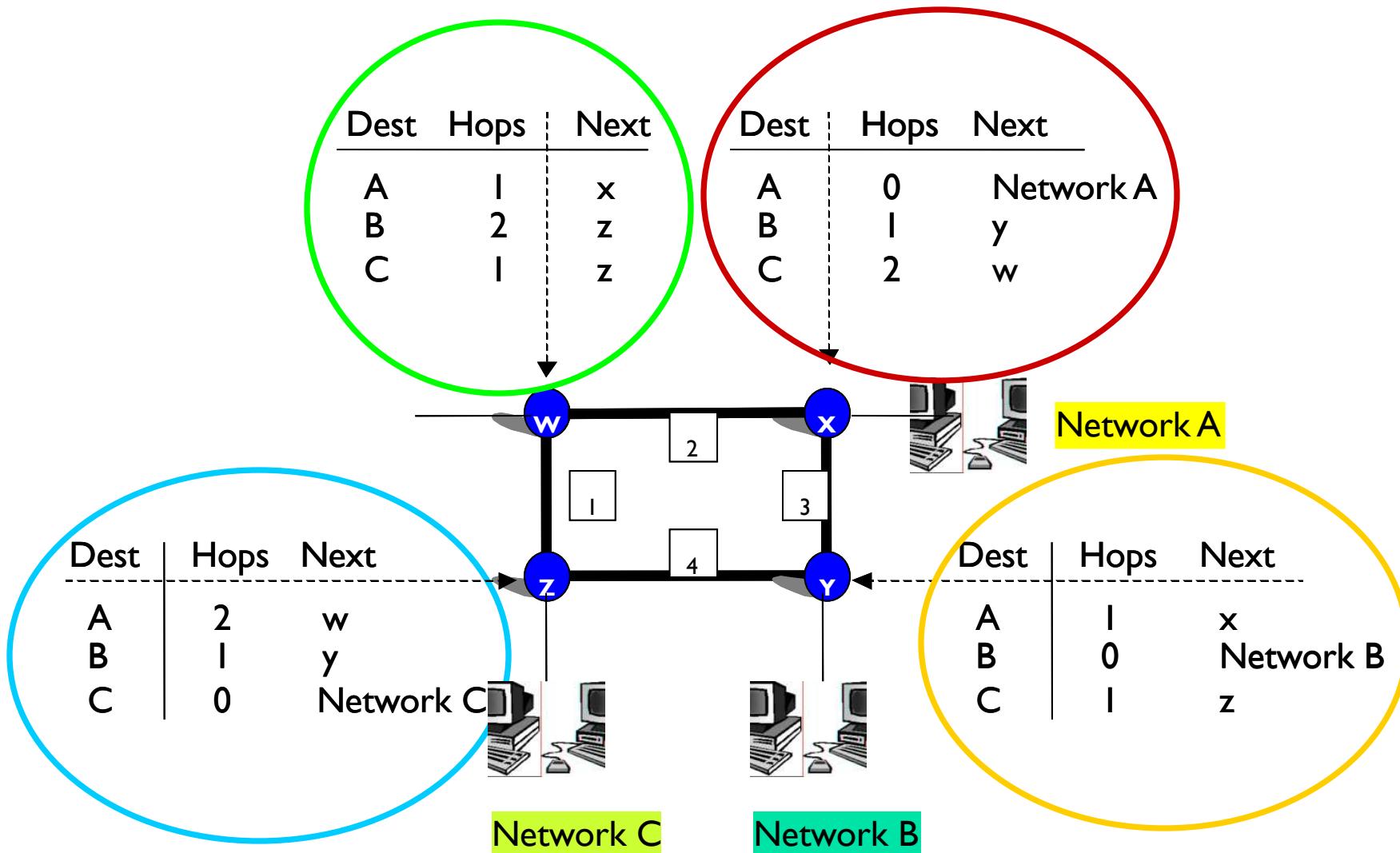


# Il routing e la funzione di un router

- La funzione principale di un router è quella di determinare i percorsi che i pacchetti devono seguire per arrivare a destinazione, partendo da una data sorgente
  - ogni router si occupa, quindi, del processo di ricerca di un percorso per l'instradamento di pacchetti tra due nodi qualunque di una rete
- Problemi da risolvere
  - Quale sequenza di router deve essere attraversata?
  - Esiste un percorso migliore (più breve, meno carico, ...)?
  - Cosa fare se un link si guasta?
  - Trovare una soluzione robusta e scalabile ...



# Un esempio di tabelle di routing





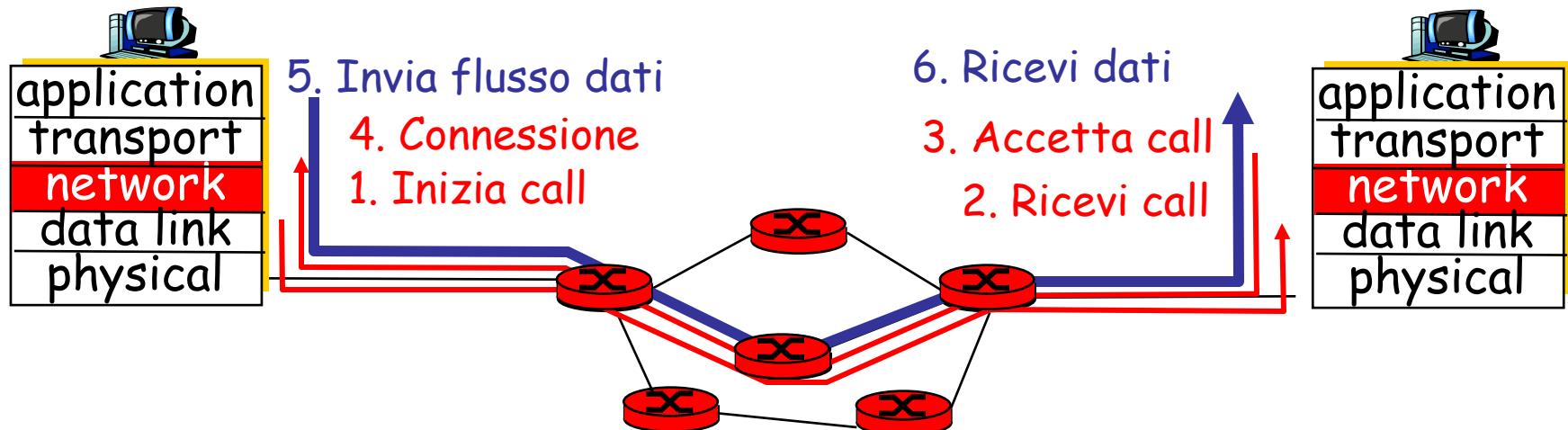
# Tecniche di routing

- Routing by Network Address
  - ogni pacchetto contiene l'indirizzo del nodo destinatario, che viene usato come chiave di accesso alle tabelle di instradamento
  - usato tipicamente nei protocolli non orientati alla connessione
    - IPv4 e IPv6, bridge trasparenti, OSI CLNP, ...
- Label Swapping
  - ogni pacchetto è marcato con una *label* (etichetta) che
    - identifica la connessione
    - viene usata come chiave per determinare l'instradamento
  - generalmente usato nei protocolli orientati alla connessione
    - X.25, ATM, MPLS, ...



# Routing: reti a circuiti virtuali

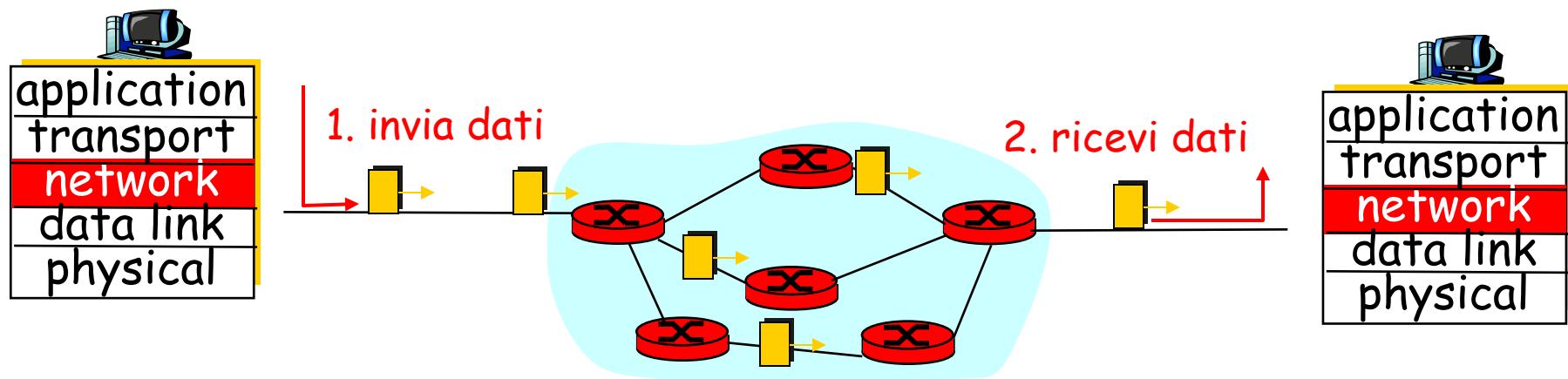
- Viene aperta una connessione *prima* di inviare dati





# Routing: reti a datagramma

- Non esiste la fase di *call setup* a livello rete
- Nei router non esiste il concetto di connessione
- I pacchetti sono indirizzati usando un ID di destinazione
  - pacchetti fra la stessa coppia sorgente-destinazione possono seguire strade diverse





# Tipologie di routing

- La scelta del percorso di instradamento può essere realizzata mediante due approcci
  - centralizzato
    - più semplice, ma non scalabile
  - distribuito
    - più complesso, ma scalabile e robusto
- Lo scopo ultimo di un protocollo di routing consiste nel creare una **tabella di instradamento** in ciascun nodo della rete
  - ciascun nodo deve prendere una decisione locale sulla base della conoscenza dello stato dell'intera rete
  - **Questa è, probabilmente, la difficoltà principale del routing**



# Routing centralizzato

- Esiste un nodo centrale che calcola e distribuisce le tabelle
  - tale nodo riceve informazioni sullo stato della rete da tutti gli altri e calcola le nuove tabelle
- Ottimizza le prestazioni, ma è poco robusto
  - aggiornamenti parziali delle tabelle dovuti a guasti possono generare loop
  - induce un notevole carico sulla rete, specialmente in prossimità del nodo centrale



# Routing distribuito

- Ogni router calcola le sue tabelle dialogando con gli altri router
  - Ogni router informa gli altri riguardo le “rotte” che conosce
- Il dialogo tra router avviene tramite dei protocolli ausiliari di livello rete
- Comprende due approcci principali
  - Algoritmi *distance vector*
  - Algoritmi *link state*
- Utilizzato in varie reti proprietarie, in OSI, ed in Internet



# Problematiche associate al routing

- Un router deve opportunamente sintetizzare le informazioni rilevanti utili alle proprie decisioni
  - per prendere correttamente decisioni locali bisogna avere almeno una conoscenza parziale dello stato globale della rete
  - lo stato globale della rete è difficile da conoscere in quanto si può riferire ad un dominio molto esteso e che cambia in maniera estremamente dinamica
- Le tabelle di routing devono essere memorizzate all'interno dei router
  - bisogna minimizzare l'occupazione di spazio e rendere rapida la ricerca
  - bisogna minimizzare il numero di messaggi che i router si scambiano
- Si deve garantire la robustezza dell'algoritmo



# Scambio delle informazioni di *update*

- Broadcast periodico
  - i router possono trasmettere agli altri router informazioni circa la raggiungibilità delle reti (destinazioni) di propria competenza ad intervalli regolari di tempo
  - questa tecnica risulta inefficiente, in quanto si spediscono informazioni anche quando non è cambiato nulla rispetto all'update precedente
- Event-driven
  - in questo caso gli update sono inviati solo quando è cambiato qualcosa nella topologia oppure nello stato della rete
  - questa tecnica garantisce un uso più efficiente della banda disponibile



# Scelta dell'algoritmo di routing: problematiche

- Possono esistere più criteri di ottimalità contrastanti
  - Es: “minimizzare il ritardo medio di ogni pacchetto” vs “massimizzare l'utilizzo dei link della rete”
- Il numero di nodi può essere elevato
- La topologia può essere complessa
- Algoritmi troppo complessi, operanti su reti molto grandi, potrebbero richiedere tempi di calcolo inaccettabili
- Vincoli di tipo amministrativo



# Scelta dell'algoritmo di routing: parametri

- **Semplicità**
  - I router hanno CPU e memoria finite
- **Robustezza**
  - Adattabilità alle variazioni (di topologia, di carico, ...)
- **Stabilità**
  - L'algoritmo deve convergere in tempo utile
- **Equità**
  - Stesso trattamento a tutti i nodi
- **Metrica da adottare**
  - Numero di salti effettuati, somma dei costi di tutte le linee attraversate, ecc.



# Distance Vector

- Ogni router mantiene una tabella di tutti gli instradamenti a lui noti
  - inizialmente, solo le reti a cui è connesso direttamente
- Ogni entry della tabella indica
  - una rete raggiungibile
  - il *next hop*
  - il numero di hop necessari per raggiungere la destinazione
- Periodicamente, ogni router invia **a tutti i vicini** (due router sono vicini se sono collegati alla stessa rete fisica)
  - un messaggio di aggiornamento contenente tutte le informazioni della propria tabella (vettore delle distanze)
- I router che ricevono tale messaggio aggiornano la tabella nel seguente modo
  - eventuale modifica di informazioni relative a cammini già noti
  - eventuale aggiunta di nuovi cammini
  - eventuale eliminazione di cammini non più disponibili



# Distance Vector: un esempio

Destin.	Dist.	Route
net 1	0	direct
net 2	0	direct
net 4	8	router L
net 17	5	router M
net 24	6	router A
net 30	2	router Q
net 42	2	router A

Tabella del router B

Destin.	Dist.
net 1	2
net 4	3
net 17	6
net 21	4
net 24	5
net 30	10
net 42	3



Messaggio di  
aggiornamento  
del router A  
(vicino di B)

Destin.	Dist.	Route
net 1	0	direct
net 2	0	direct
net 4	4	router A
net 17	5	router M
net 24	6	router A
net 30	2	router Q
net 42	4	router A
net 21	5	router A

Tabella aggiornata del router B



# Distance Vector: analisi

- Vantaggi
  - facile da implementare
- Svantaggi
  - ogni messaggio contiene un'intera tabella di routing
  - lenta propagazione delle informazioni sui cammini
    - se lo stato della rete cambia velocemente, le rotte possono risultare inconsistenti



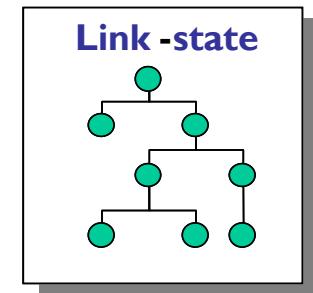
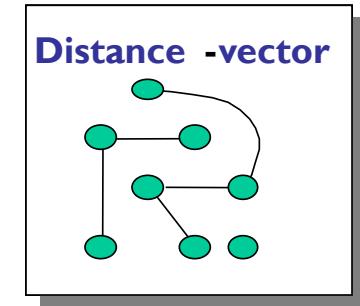
# Link State

- Utilizza l'algoritmo *Shortest Path First (SPF)*
  - non basato sullo scambio delle tabelle
  - tutti i router devono conoscere l'intera topologia della rete
- Ogni router esegue due azioni
  - controlla lo stato di tutti i router vicini
  - periodicamente invia, **in broadcast**, un messaggio contenente lo stato dei link a cui è collegato
- I router utilizzano i messaggi ricevuti per aggiornare la loro mappa della rete
  - se la mappa cambia, il router ricalcola i percorsi di instradamento applicando l'algoritmo di Dijkstra



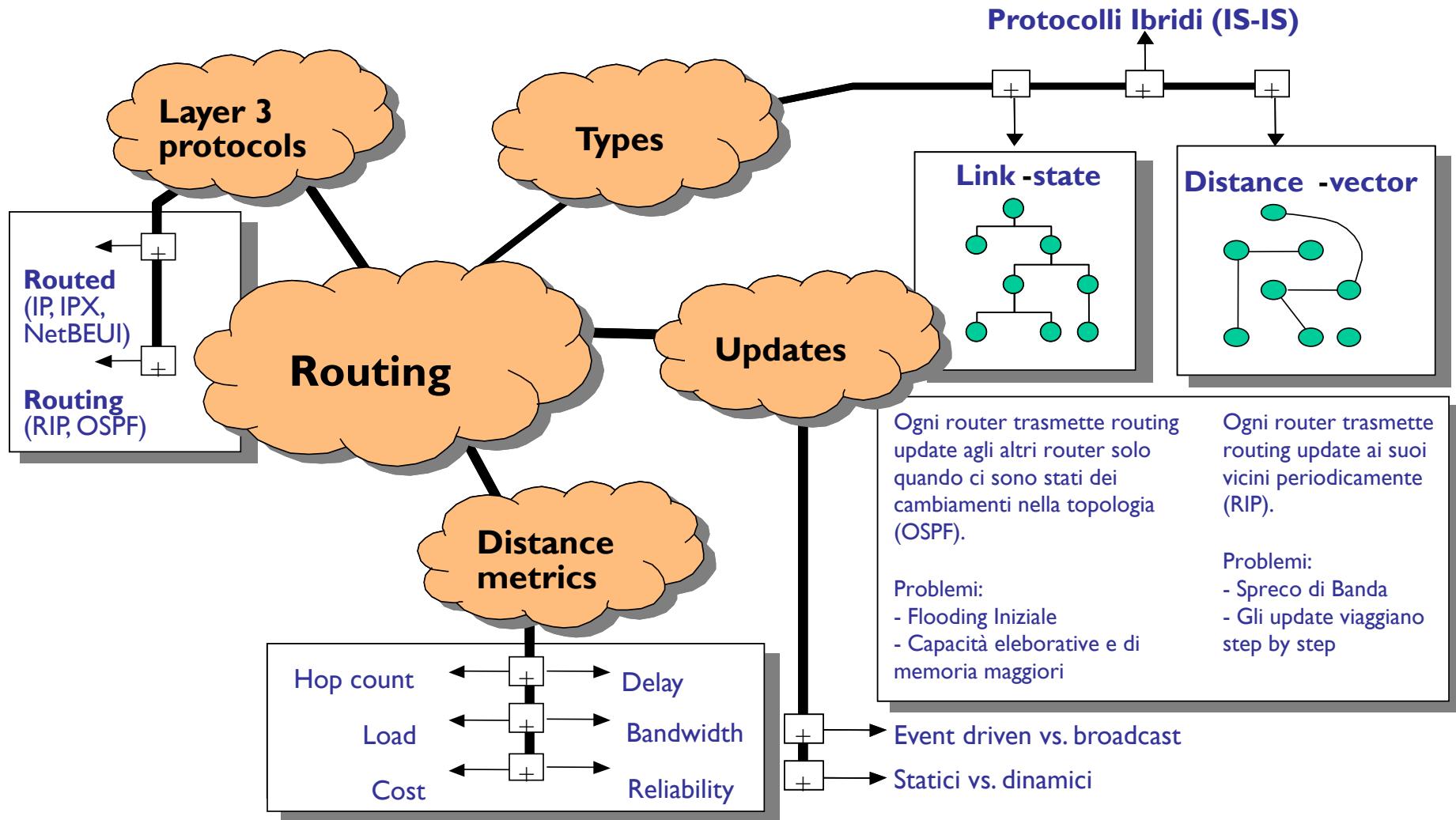
# Distance Vector vs LINK STATE

- *Distance Vector*
  - vengono mandate ai vicini le informazioni su tutti (il vettore delle distanze)
- *Link State*
  - vengono mandate a tutti informazioni sui vicini
    - si invia in broadcast un Link State Packet contenente informazioni circa lo stato dei vicini
    - ogni router contiene la topologia della rete





# Ricapitolando...





# Corso di Laurea in Informatica

## Corso di Reti di Calcolatori 1

Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))

### Introduzione alle Reti di Calcolatori

Concetti fondamentali sulla commutazione di pacchetto  
e sulla pila protocolare

I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso

# Nota di copyright per le slide COMICS



## Nota di Copyright

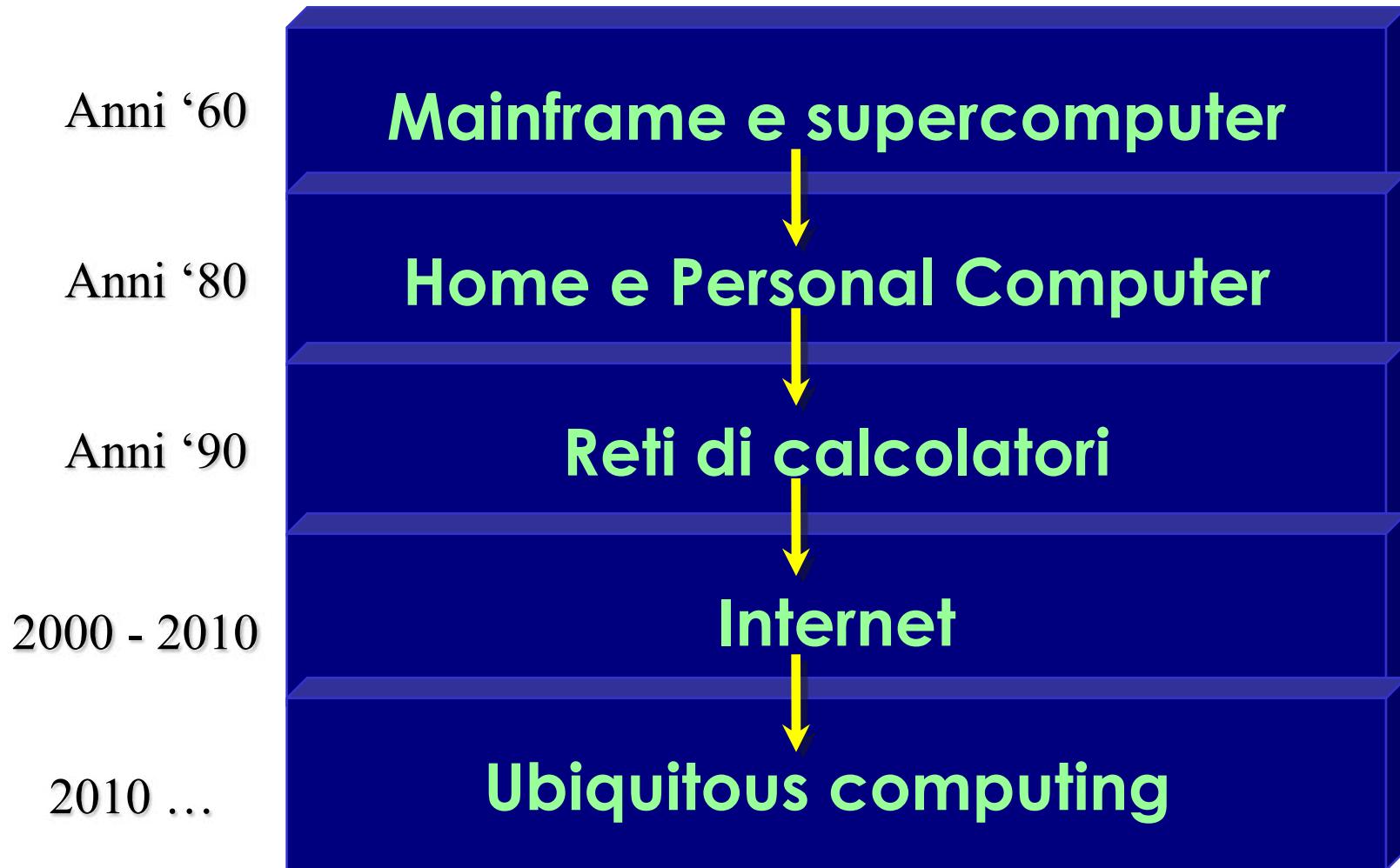
Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta



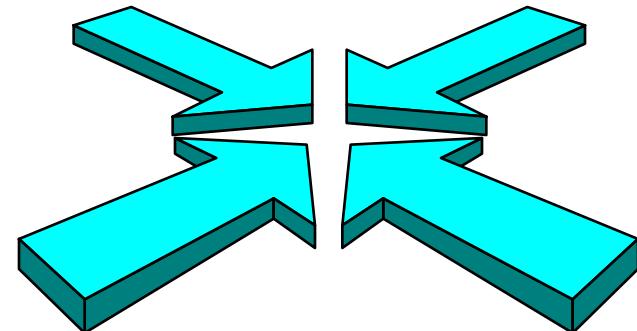
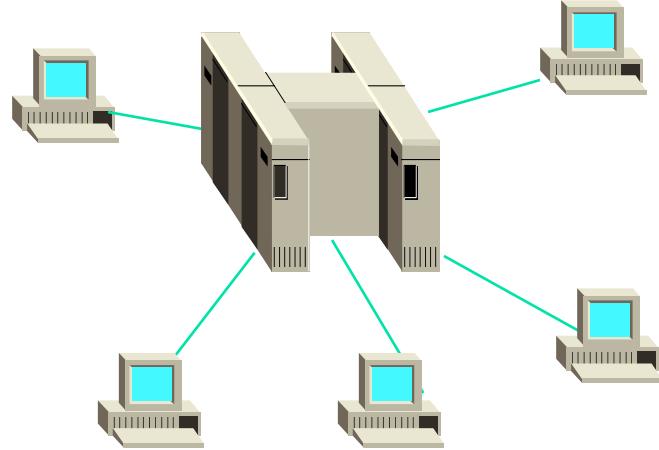
# L'evoluzione





# L'evoluzione: dal “computing centralizzato”...

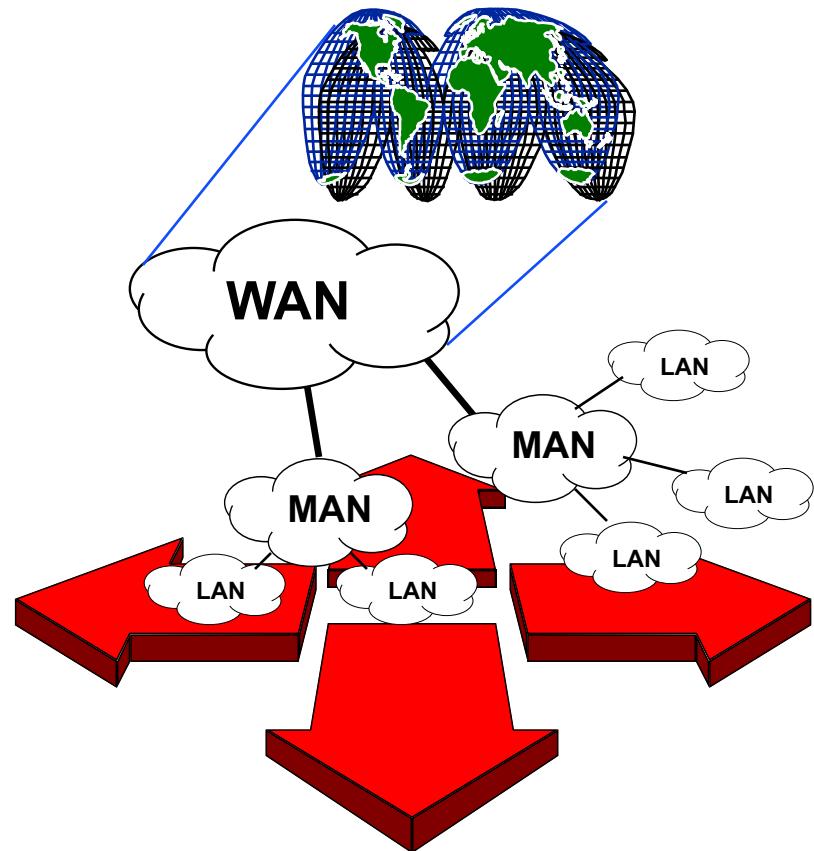
- Sistemi fortemente centralizzati
- Non reti in senso stretto ma collegamenti tra periferiche e CPU





# L'evoluzione: ...al “computing distribuito”

- Avvento del PC
- Paradigma client-server
- Nascita e boom delle reti locali
- Evoluzione verso sistemi *aperti*
- Periferia eterogenea ed intelligente
- Nascita degli standard per:
  - Cablaggi strutturati
  - Protocolli di Comunicazione





# Le reti di calcolatori: scopi

- **Condivisione dell'informazione**
- **Condivisione delle risorse**
- **Accesso a risorse remote**
- **Accesso a servizi**



# Elementi costitutivi delle reti di calcolatori (1)

- Alle estremità della rete si trovano gli **end-system** o **host**

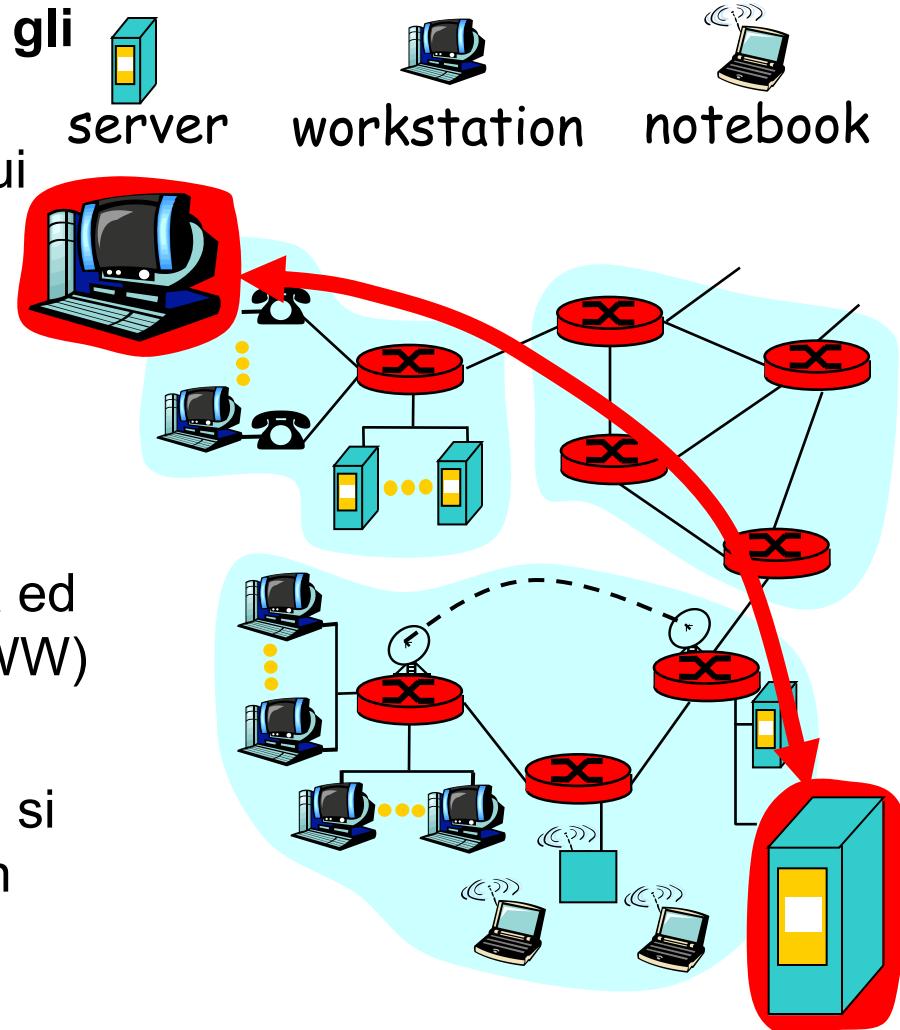
- sono calcolatori di vario tipo su cui girano i programmi applicativi
- i programmi applicativi possono essere progettati secondo due modelli principali

- Client-Server**

- Il client invia una richiesta ed il server risponde (es. WWW)

- Peer-to-peer**

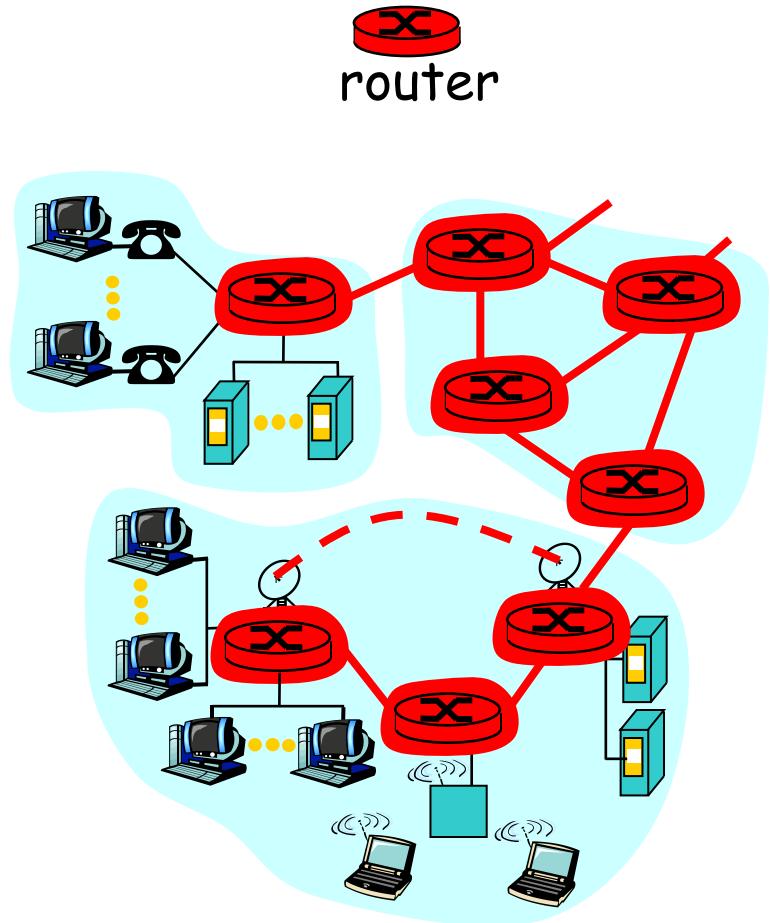
- Le due entità comunicanti si scambiano informazioni in modo paritetico (es. eMule, Skype)





# Elementi costitutivi delle reti di calcolatori (2)

- L'infrastruttura della rete è fatta di
  - **apparati** tra loro interconnessi
    - hub, switch, bridge
    - modem
    - access point
    - router
    - ...
  - **supporti trasmissivi** che realizzano le interconnessioni
    - doppini in rame
    - cavi coassiali
    - fibre ottiche
    - collegamenti radio
    - ...





# Struttura delle reti di calcolatori (1)

- L'infrastruttura di rete si può dividere grossolanamente in
  - Reti di accesso

- Forniscono la connettività agli end-system

- Utilizzano svariate tecnologie

- Rete telefonica tradizionale
    - Ethernet
    - ATM
    - X.25
    - Frame Relay
    - WLAN
    - Bluetooth
    - 3/4/5G

Tecnologie “Wired”

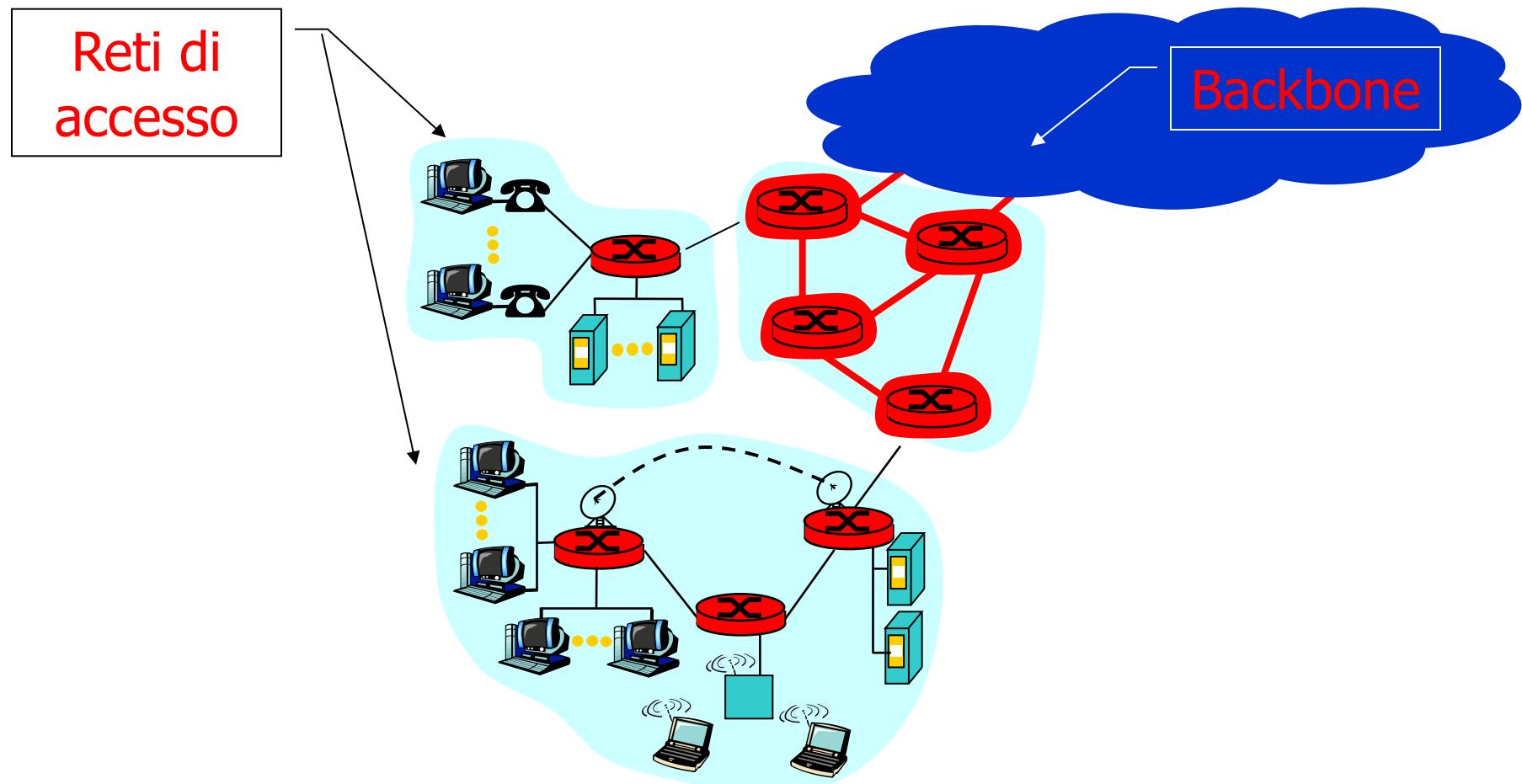
Tecnologie “Wireless”

- Reti di backbone

- Costituiscono la dorsale della rete vera e propria
  - Sono strutturate in sottoreti tra loro interconnesse
  - Si collegano alle reti di accesso



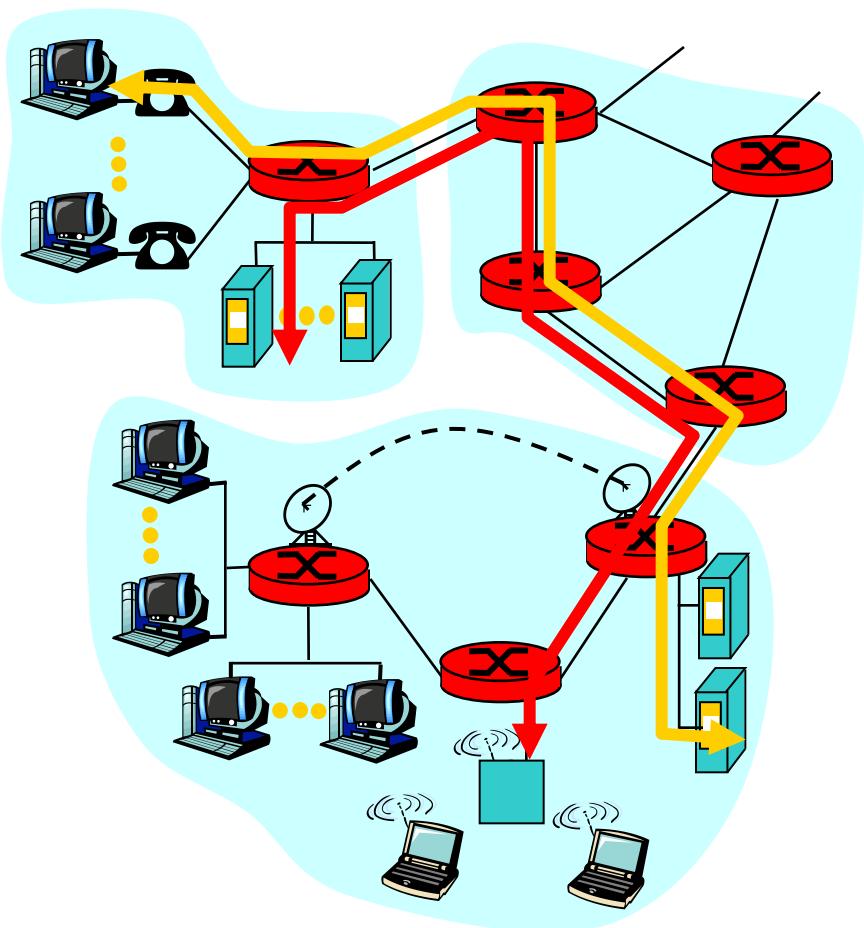
# Struttura delle reti di calcolatori (2)





# Commutazione di circuito

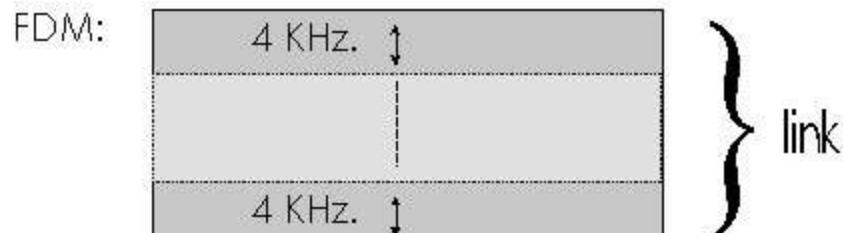
- Nelle reti a commutazione di circuito, la capacità trasmissiva all'interno della rete è assegnata per ciascuna “chiamata”
  - È definita una porzione di capacità trasmissiva che è allocata in modo esclusivo per servire ciascuna comunicazione
  - È il modello dell'attuale rete telefonica



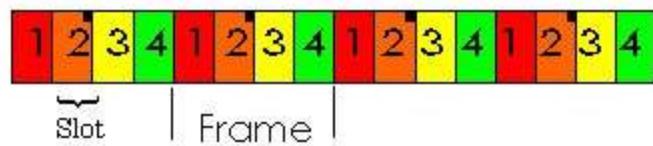


# Commutazione di circuito

- Nella telefonia tradizionale, due coppie di conduttori venivano impegnate per ciascuna conversazione
- Successivamente, le coppie “fisiche” sono state sostituite da:
  - **“porzioni di banda”** (multiplazione a divisione di frequenza, **FDM**)
  - **“porzioni di tempo”** (multiplazione a divisione di tempo, **TDM**)



TDM:



All slots labelled **2** are dedicated to a specific sender-receiver pair.



# Commutazione di Pacchetto

- La natura discontinua della trasmissione di dati digitali può essere sfruttata per far sì che flussi di dati differenti possano condividere la stessa connessione, a patto di poterli distinguere
- Questo principio è alla base della tecnica detta “comutazione di pacchetto” o *packet switching*



# Commutazione di Pacchetto (2)

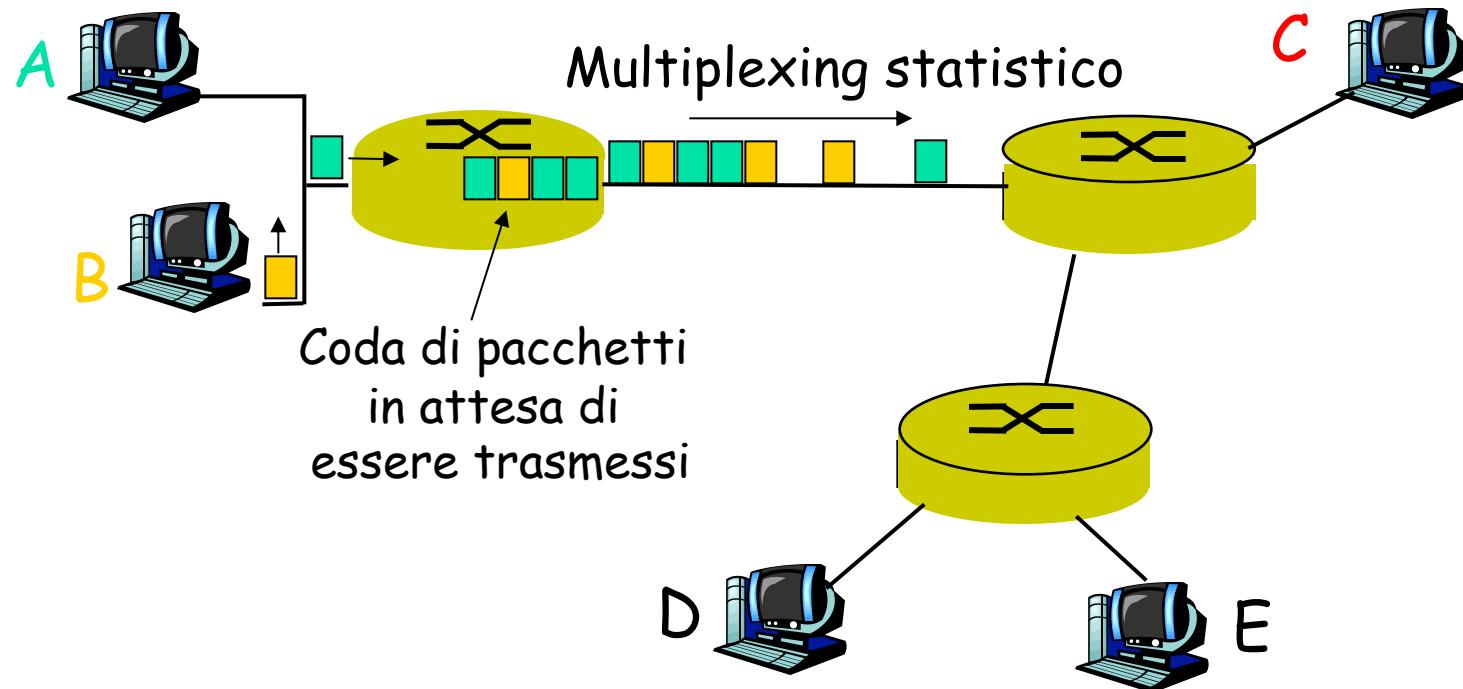
- Nel *Packet Switching* ciascun flusso di dati è diviso in pacchetti, cioè in entità composte da
  - un'intestazione (*header*), utilizzata ai fini dell'identificazione e gestione
  - i dati veri e propri (*payload*)



- Una rete a commutazione di pacchetto è composta da
  - sistemi terminali (*End System* o *host*): producono o ricevono dati
  - apparati che si occupano dell'instradamento dei pacchetti tra sorgente e destinazione, detti nodi della rete (*Network Nodes*)
- Ogni nodo memorizza i pacchetti in ingresso, per poi instradarli verso il nodo successivo (*store & forward*)
- I collegamenti fisici tra i nodi sono detti *link*



# Commutazione di Pacchetto (3)





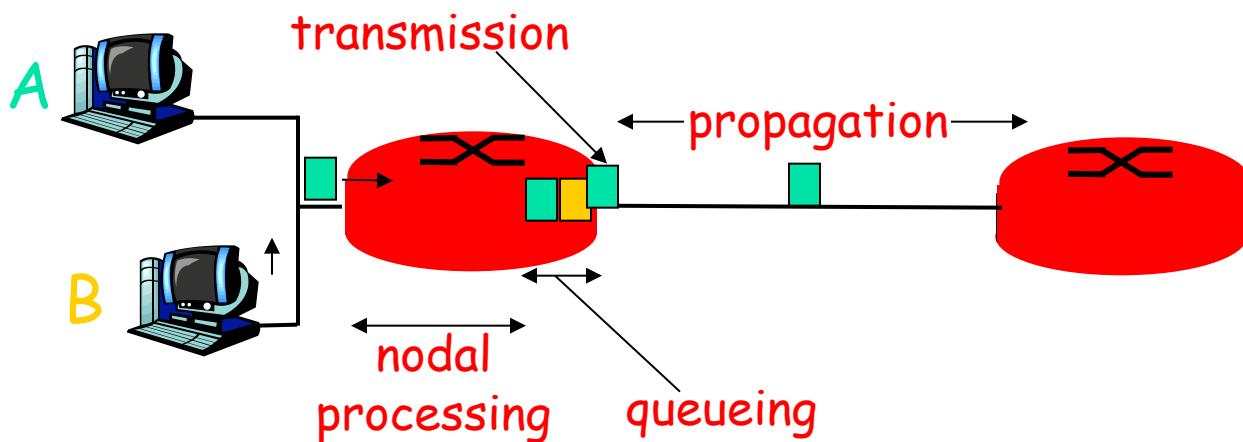
# Commutazione di Pacchetto (4)

- La Qualità del Servizio di una rete a commutazione di pacchetto è misurata da una molteplicità di “indici di prestazione”
- I più importanti sono:
  - **Ritardo** nella consegna dei pacchetti [s]
  - **Throughput**: quantità di bit al secondo che la rete è in grado di trasferire tra due terminali [b/s]
  - **Loss-Rate**: probabilità che un pacchetto non venga consegnato a destinazione
  - **Jitter**: Variazione temporale del ritardo



# Ritardo nelle reti a commutazione di pacchetto

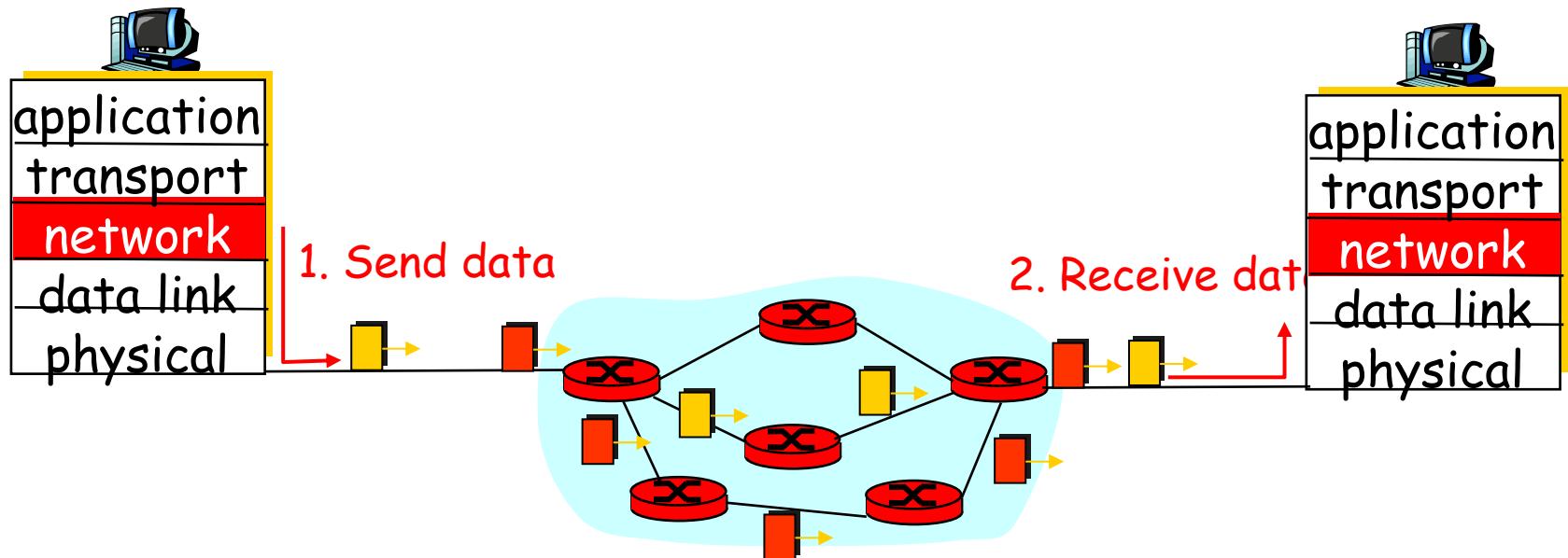
- Il ritardo nella consegna di un pacchetto alla destinazione è determinato da
  - Tempo di elaborazione nel nodo
    - controllo di errori, determinazione link di uscita, ...
  - Tempo di trasmissione su ciascun link = Lunghezza in bit / velocità in bps
  - Tempo di attesa nelle code dei router (variabile)
  - Tempo di propagazione sulle linee = lunghezza della linea / velocità del segnale





# Packet switching: reti a datagrammi

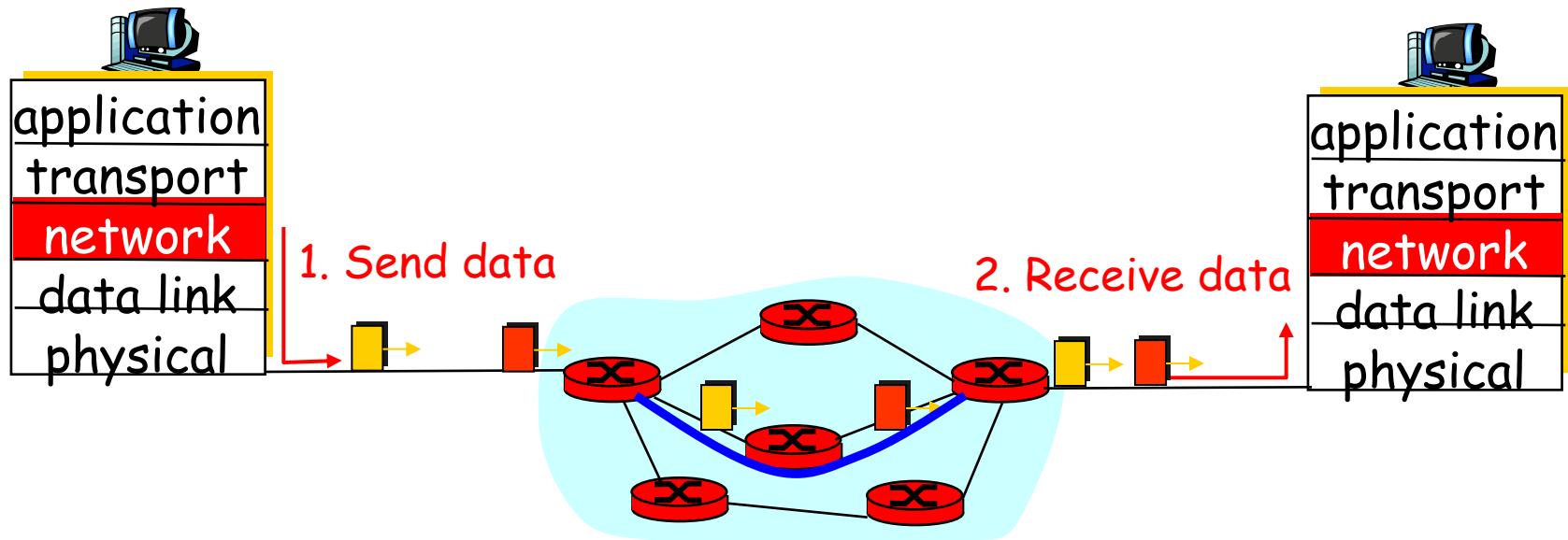
- Ogni nodo che riceve un pacchetto decide in maniera **indipendente** a quale altro nodo inoltrarlo, sulla base dell'indirizzo destinazione contenuto nel pacchetto
  - **Indipendente** rispetto agli altri nodi
  - **Indipendente** rispetto agli altri pacchetti passanti per lo stesso nodo
- Pacchetti tra la stessa coppia sorgente-destinazione possono seguire percorsi differenti





# Packet switching: reti a circuiti virtuali

- Ogni pacchetto contiene il numero del circuito virtuale
- Il circuito virtuale è stabilito prima della trasmissione dei dati
- I nodi devono conservare informazioni sui circuiti virtuali che li attraversano





# Datagrammi vs circuiti virtuali

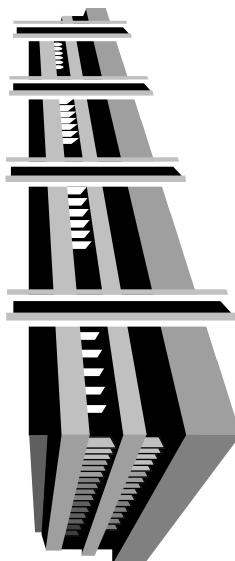
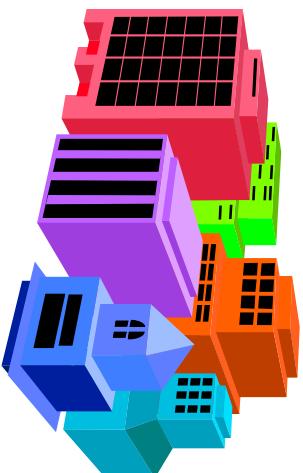
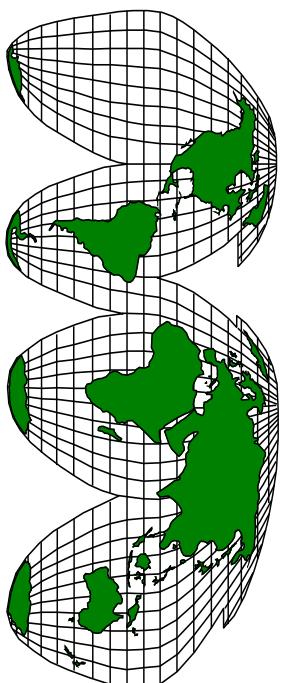
Proprietà	datagrammi	circuiti virtuali
Creazione del circuito	Non richiesta	Richiesta
Indirizzamento	Ogni pacchetto contiene l'intero indirizzo della sorgente e della destinazione	Ogni pacchetto contiene un numero di VC
Informazioni sullo stato	I nodi di rete non mantengono informazioni sullo stato	Ogni VC richiede uno spazio di memoria sui nodi
Instradamento	Ogni pacchetto è instradato indipendentemente	Percorso pre-calcolato: ogni pacchetto segue questo percorso
Effetti di guasti ai nodi	Nessuno (solo i pacchetti persi durante il guasto)	Tutti i VC che attraversano quel nodo sono chiusi
Controllo di congestione	Complicato	Semplice se possiamo allocare spazio sufficiente per ogni VC

# Le reti di calcolatori

## Dimensione della rete



Numero di host connessi



**WAN**

**MAN**

**LAN**



Capacità di Banda





# Reti eterogenee

- Gli esempi presentati non sono mutuamente esclusivi
- Nella maggior parte dei casi, soluzioni architetturali differenti coesistono in un singolo sistema distribuito complesso, organizzato in maniera gerarchica (LAN, MAN, WAN)
- L'esempio per eccellenza: **Internet**



# Le reti di calcolatori: gestire la complessità

- La comunicazione tra computer richiede soluzioni tecniche complesse riguardanti una serie di problemi
  - Ricezione e Trasmissione fisica
  - Controllo degli errori
  - Controllo di flusso
  - Conversione dei dati
  - Crittografia e sicurezza
  - Sincronizzazione
- Un approccio logico è quello di analizzare tali problematiche singolarmente:  
*“Divide et Impera”*
- Nelle reti di calcolatori questo ha condotto a modelli “a strati”



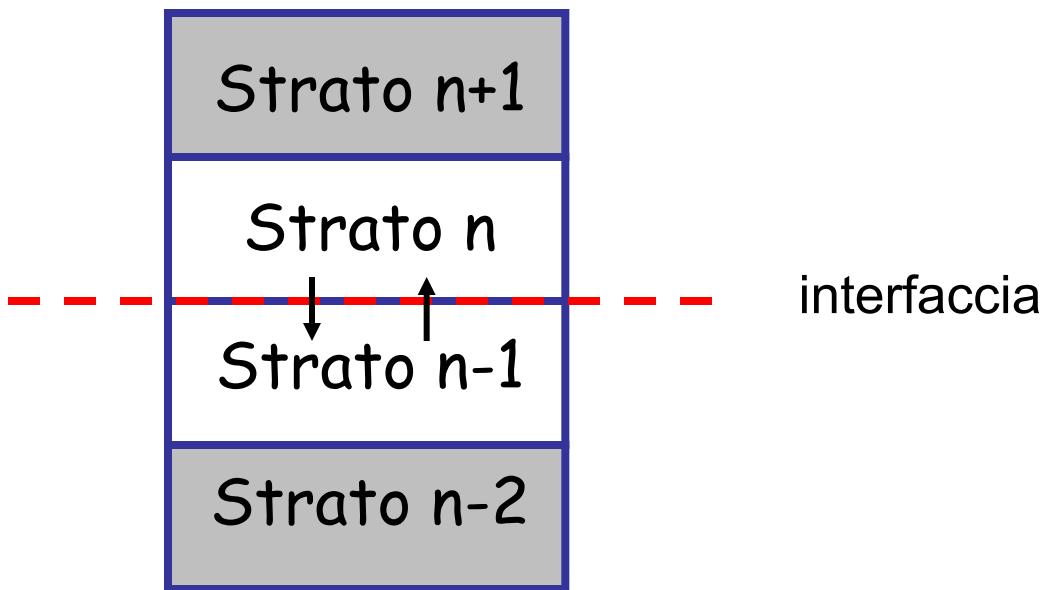
# Modelli a strati: perché

- Come vedremo, la suddivisione delle funzionalità secondo un modello a strati agevola la gestione della complessità
- Ciascuno **strato** (o **livello**)
  - è responsabile di un sottoinsieme definito e limitato di compiti
  - funziona in maniera lascamente accoppiata con gli altri
  - interagisce solo con gli strati immediatamente superiore ed inferiore
  - fa affidamento sui “servizi” forniti dallo strato immediatamente inferiore
  - fornisce “servizi” allo strato immediatamente superiore
- Alcuni strati sono realizzati in software altri in hardware
- Vantaggi
  - l’indipendenza tra gli strati consente la sostituzione di uno strato con un altro di pari livello che offre i medesimi servizi allo strato superiore
  - limitare le funzionalità di uno strato ne semplifica la realizzazione
- Svantaggi
  - L’eccessivo numero di strati può portare ad inefficienze



# Modelli a strati: interfacce

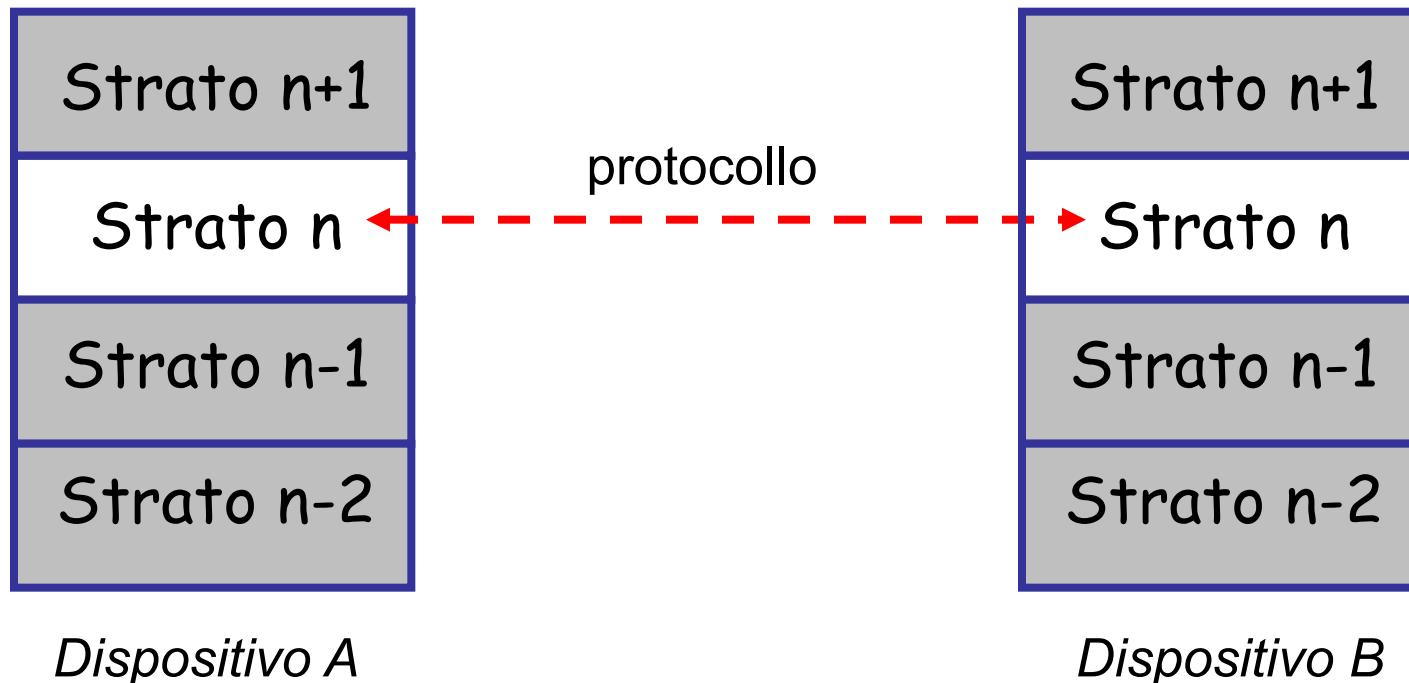
- All'interno di ciascun dispositivo di rete, lo scambio di informazioni tra due strati adiacenti avviene attraverso una **interfaccia**, che definisce i servizi offerti dallo strato inferiore allo strato superiore





# Modelli a strati: protocolli

- Lo strato n-esimo di un dispositivo comunica con lo strato n-esimo di un'altra entità secondo un **protocollo** assegnato





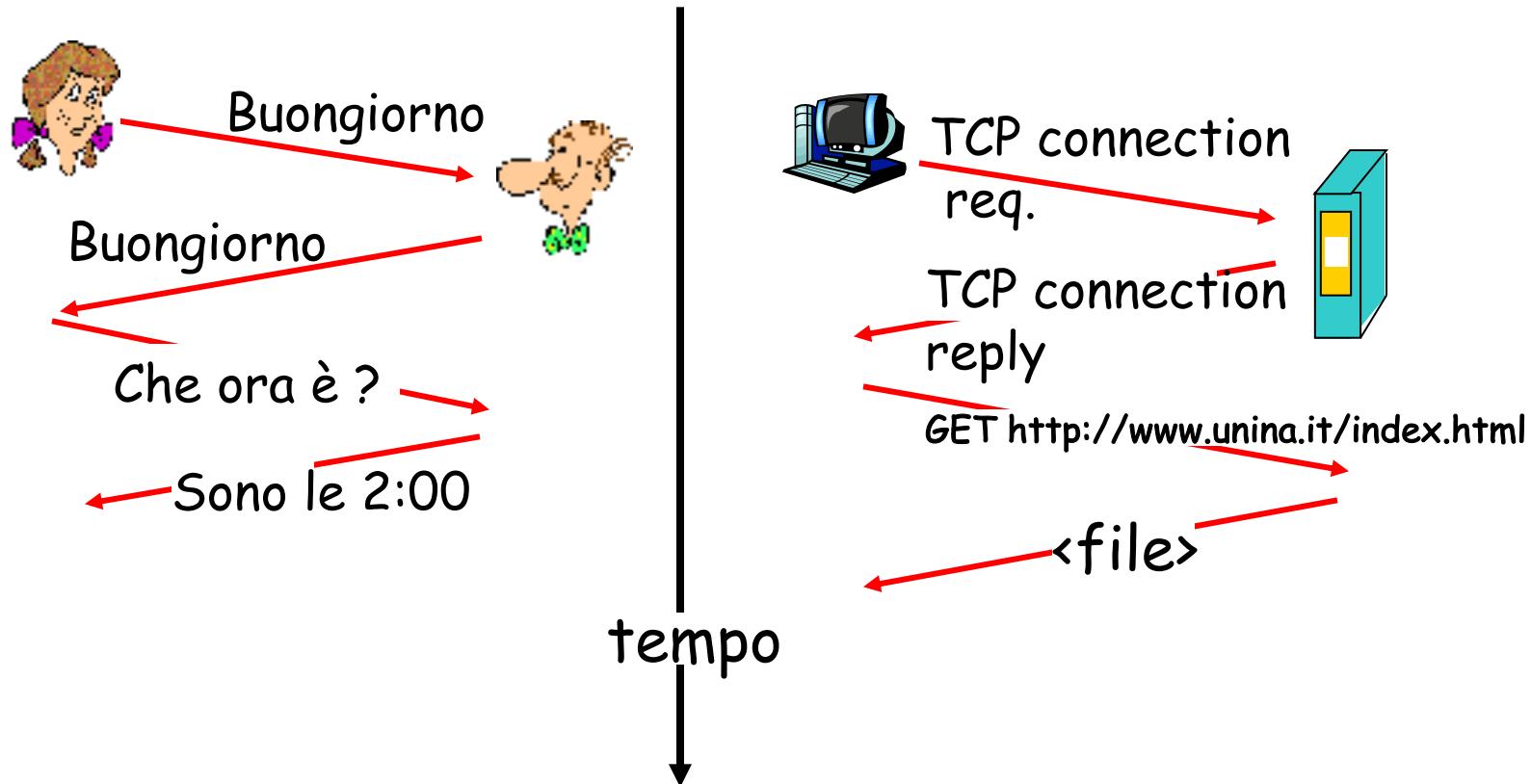
# Protocolli di comunicazione

- Per *protocollo di comunicazione* si intende un insieme di regole che permette la corretta instaurazione, mantenimento e terminazione di una comunicazione di qualsiasi tipo tra due o più entità
- Un protocollo di comunicazione definisce il formato e l'ordine dello scambio di messaggi tra le entità comunicanti
- Nelle reti di calcolatori, un protocollo regola la comunicazione tra entità di pari livello esistenti in due dispositivi della rete tra loro comunicanti
- Nell'ambito delle reti di computer un notevole sforzo è stato compiuto per definire protocolli standard, allo scopo di consentire l'integrazione di reti differenti



# Protocolli: un esempio

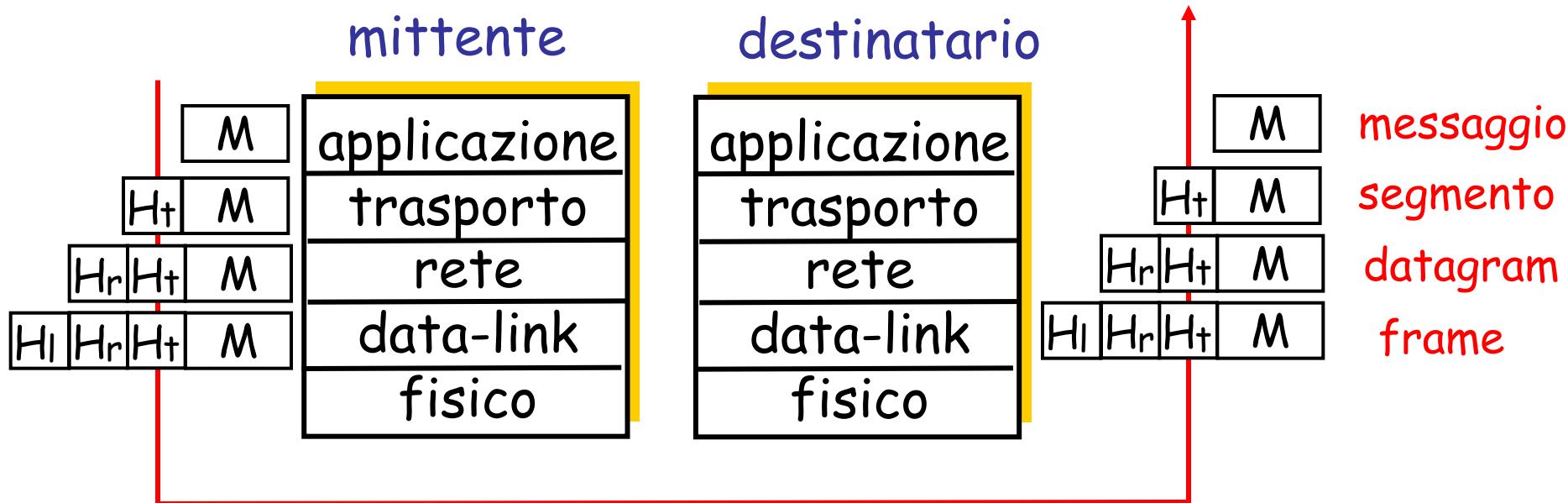
Un confronto tra un protocollo tra persone ed un protocollo per la comunicazione tra computer





# “Imbustamento” dei messaggi

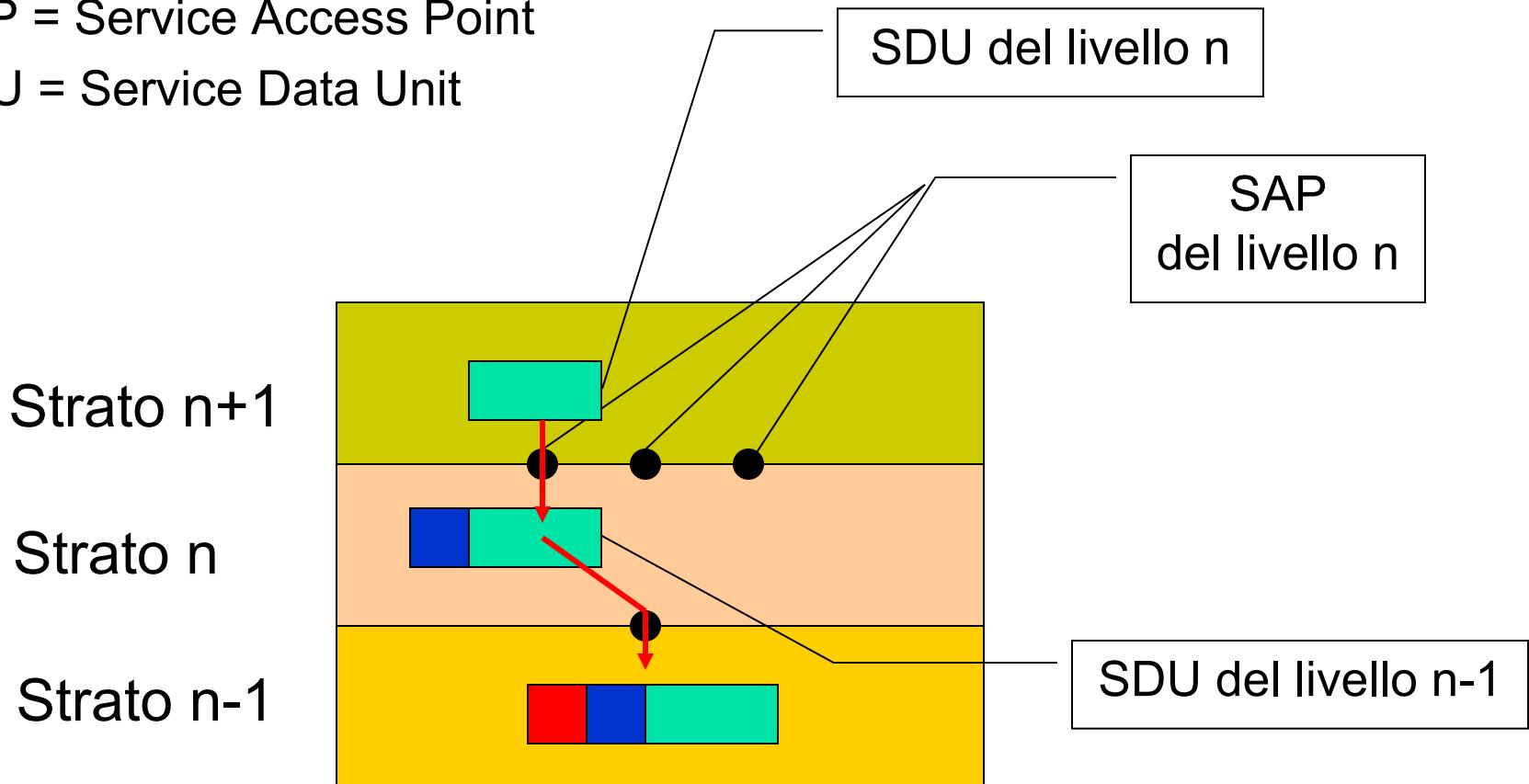
- In trasmissione, ogni strato antepone una intestazione (*header*) al messaggio ricevuto dallo strato soprastante
  - Paragone con la busta di una lettera
- L'insieme messaggio+header viene passato allo strato sottostante
- A destinazione il messaggio risale la pila
- In ricezione, ad ogni strato l'header viene rimosso





# SDU e SAP

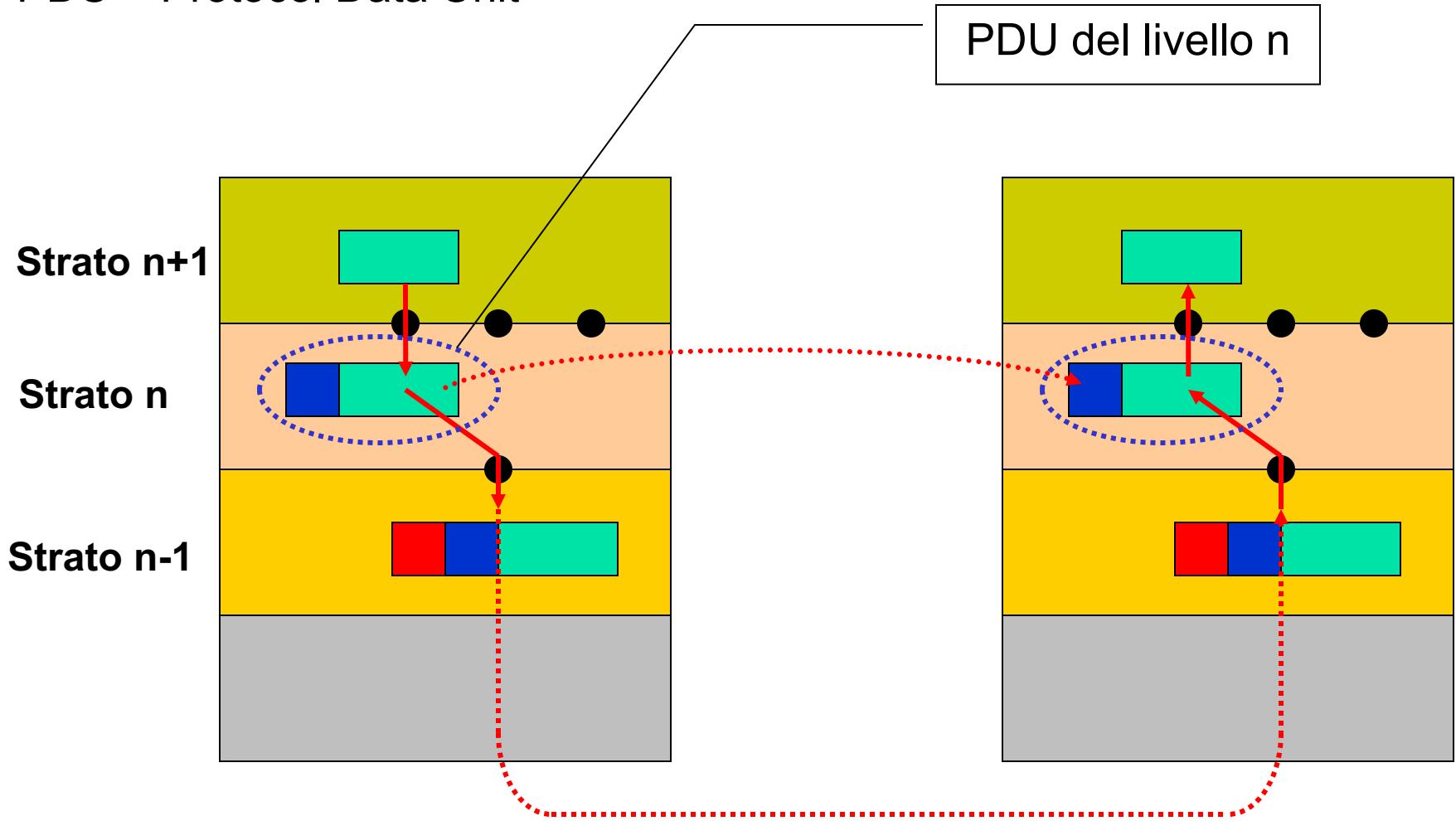
- SAP = Service Access Point
- SDU = Service Data Unit





# PDU

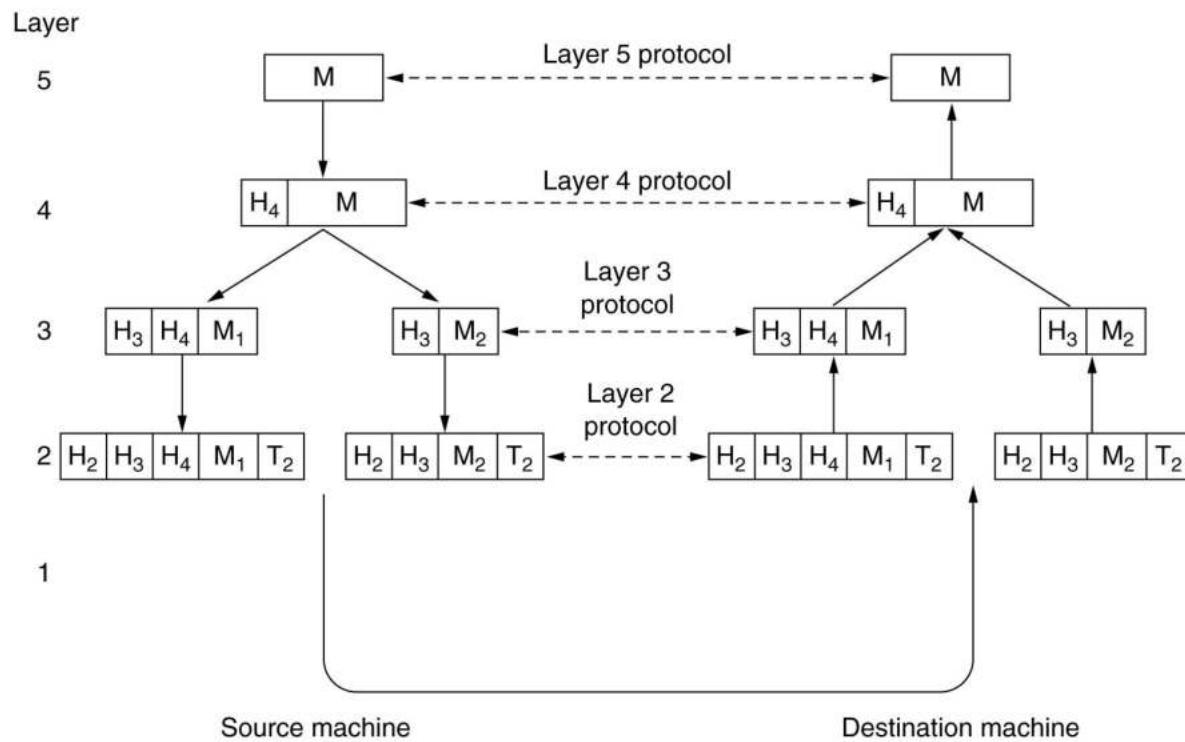
- PDU = Protocol Data Unit





# Frammentazione dei messaggi ad un livello

- Un livello della pila protocollare può essere costretto a frammentare il pacchetto ricevuto dallo strato superiore prima di passarlo allo strato inferiore
- Si rende necessaria una operazione di ricostruzione mediante riassemblaggio





# Il modello OSI

- Negli anni '80 l' *ISO, International Standards Organization*, ha definito un modello di riferimento per reti di calcolatori a commutazione di pacchetto: il modello *OSI, Open System Interconnection*
- Il modello OSI è un modello a strati su 7 livelli
  - Applicazione
  - Presentazione
  - Sessione
  - Trasporto
  - Rete
  - Data link
  - Fisico
- Il modello OSI non è risultato vincente, a causa della sua eccessiva complessità



# Il modello OSI (2)



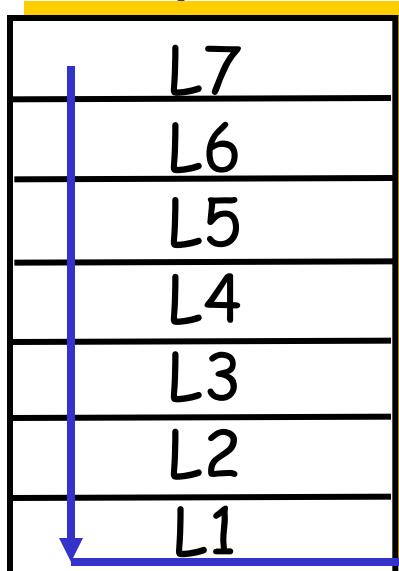
**OSI: Open Systems Interconnection**



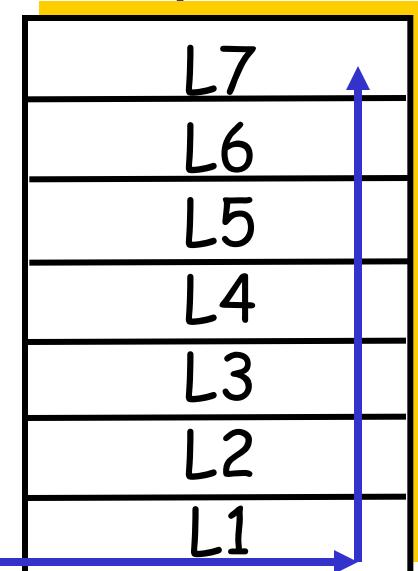
# Dispositivi di rete e livelli

- L'intera pila di livelli è realizzata negli end system
- I dispositivi di rete si differenziano per il numero di livelli fino a cui operano
  - Fino a L1 operano i **ripetitori**
  - Fino a L2 operano i **bridge / switch** di rete locale
  - Fino a L3 operano i **router**

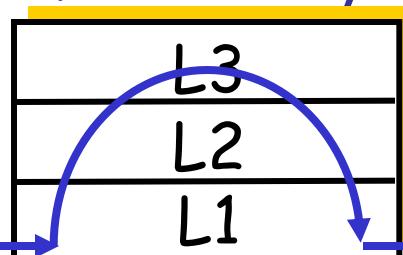
End System A



End System B

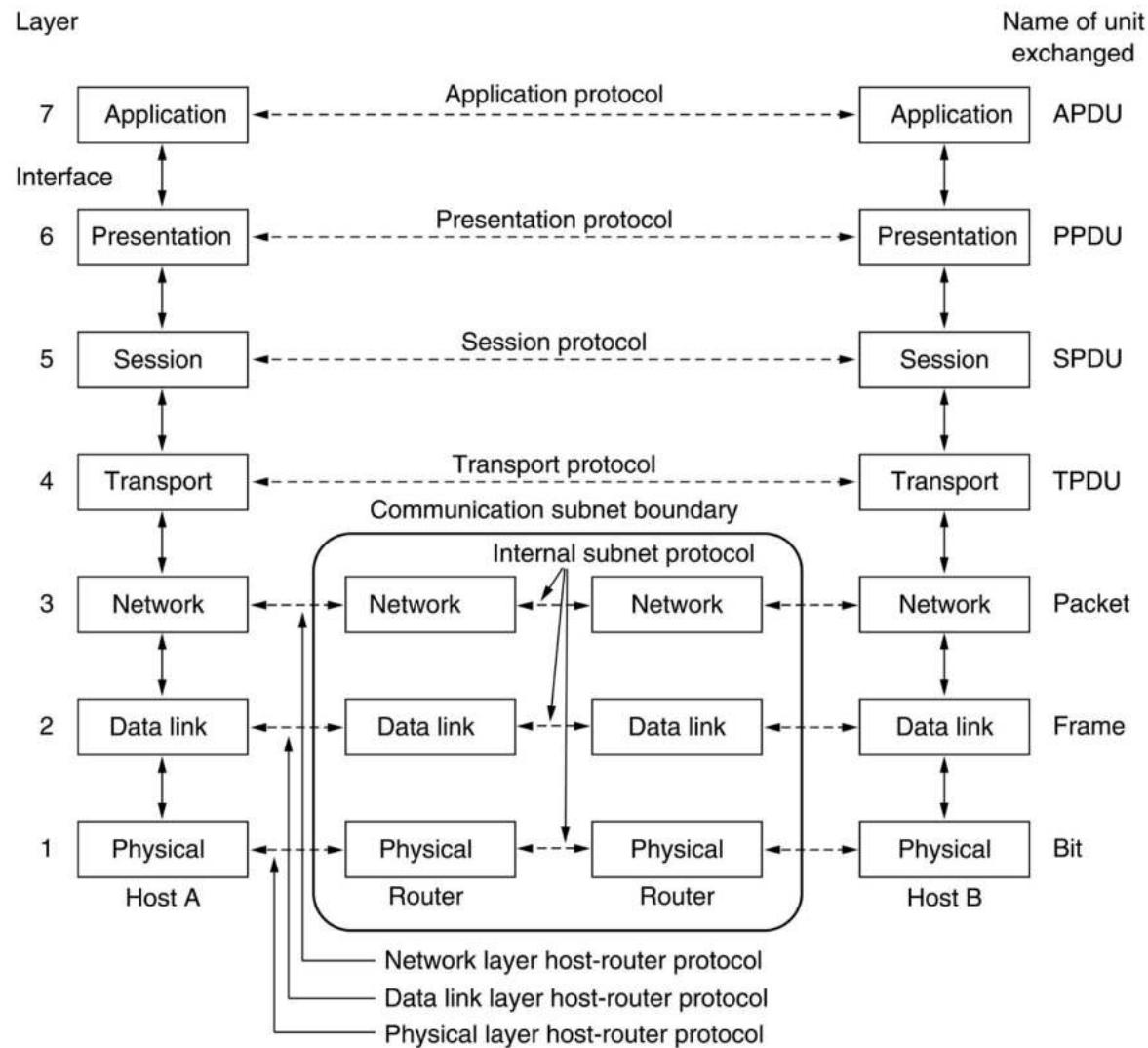


Intermediate System



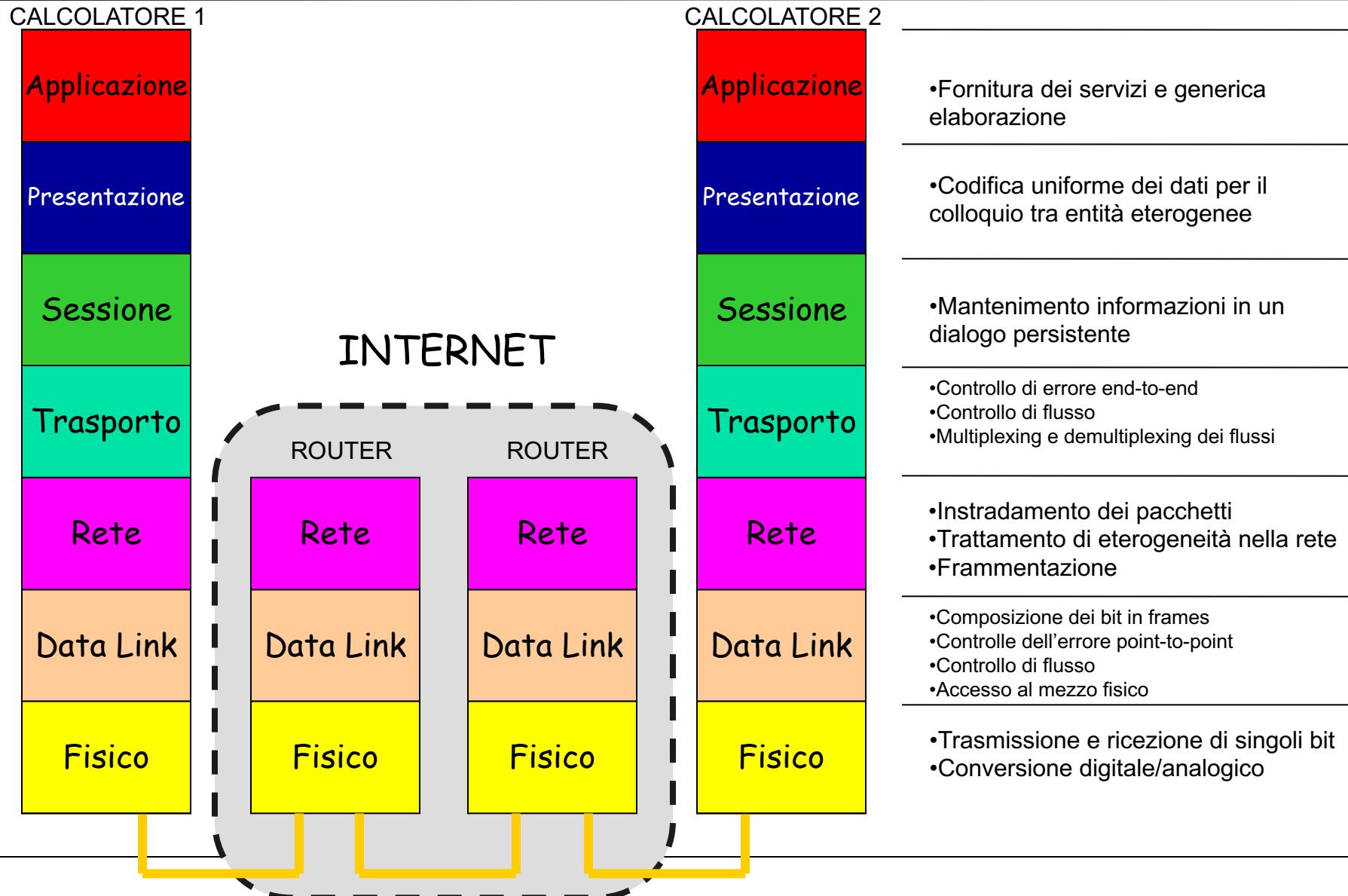


# Modello OSI: nome delle PDU





# Un approccio a livelli per la risoluzione dei problemi





# Livello 1: Fisico

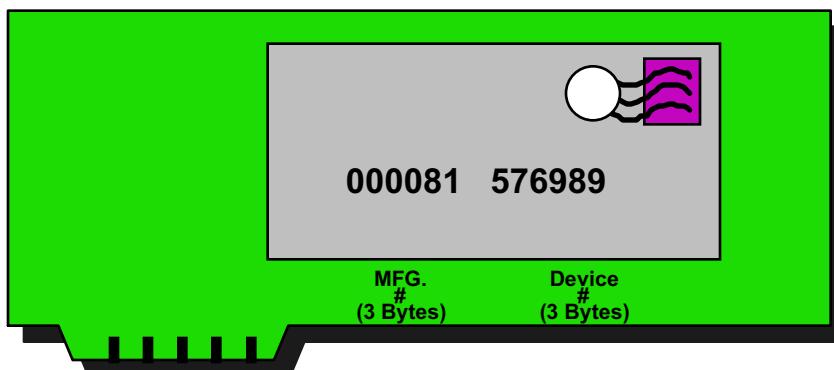
- Si occupa di trasmettere sequenze binarie sul canale di comunicazione
- A questo livello si specificano:
  - Caratteristiche elettriche dei segnali
  - Tecniche di codifica/decodifica
  - Caratteristiche dei mezzi trasmittivi
  - Tipi di connettori
- Il livello fisico è nel dominio dell'ingegneria elettronica: descrizione elettrico/meccanica dell'interfaccia

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0



## Livello 2: Data Link

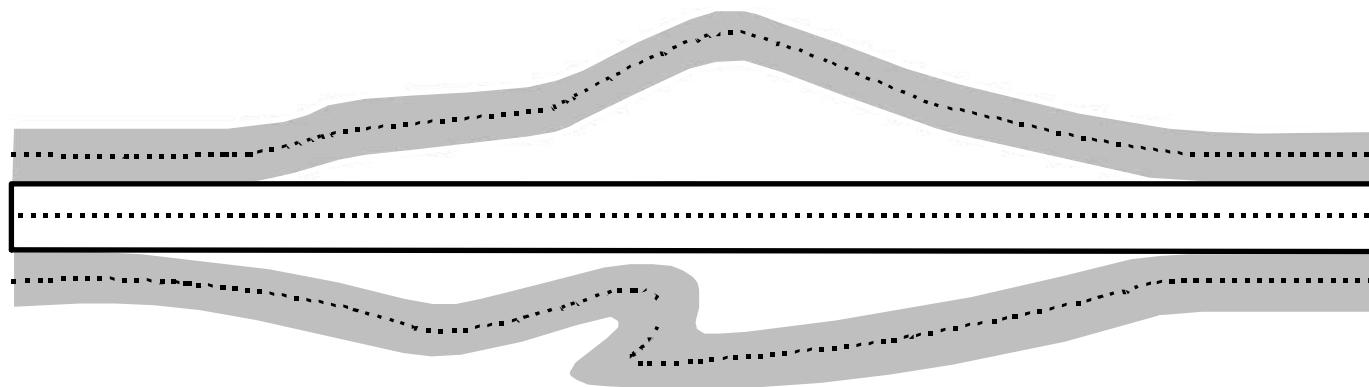
- Ha come scopo la trasmissione affidabile di pacchetti di dati (*frame*)
  - Affidabile nel senso di “garanzia di inoltro”
- Accetta come input i *frame* (tipicamente poche centinaia di byte) e li trasmette sequenzialmente
- Verifica la presenza di errori di trasmissione aggiungendo delle informazioni aggiuntive di controllo
  - *Frame Control Sequence*, FCS
- Può gestire meccanismi di correzione di errori tramite ritrasmissione





## Livello 3: Rete

- Questo livello gestisce l'instradamento dei messaggi
- Determina quali sistemi intermedi devono essere attraversati da un messaggio per giungere a destinazione
- Il livello 3 gestisce, quindi, delle tabelle di instradamento per ottimizzare il traffico sulla rete





## Livello 4: Trasporto

- Fornisce servizi per il trasferimento dei dati da terminale a terminale (ovvero *end-to-end*), indipendentemente dalla rete sottostante
- In particolare il livello 4 può
  - frammentare i pacchetti in modo che abbiano dimensioni idonee al livello 3
  - rilevare/correggere gli errori
  - controllare il flusso
  - controllare le congestioni





## Livello 5: Sessione

- Il livello 5 è responsabile dell'organizzazione del dialogo e della sincronizzazione tra due programmi applicativi e del conseguente scambio di dati
- Si occupa cioè di stabilire la **sessione**





# Livello 6: Presentazione

- Il livello di presentazione gestisce la sintassi dell'informazione da trasferire
- L'informazione è infatti rappresentata in modi diversi su elaboratori diversi (es. ASCII o EBCDIC)



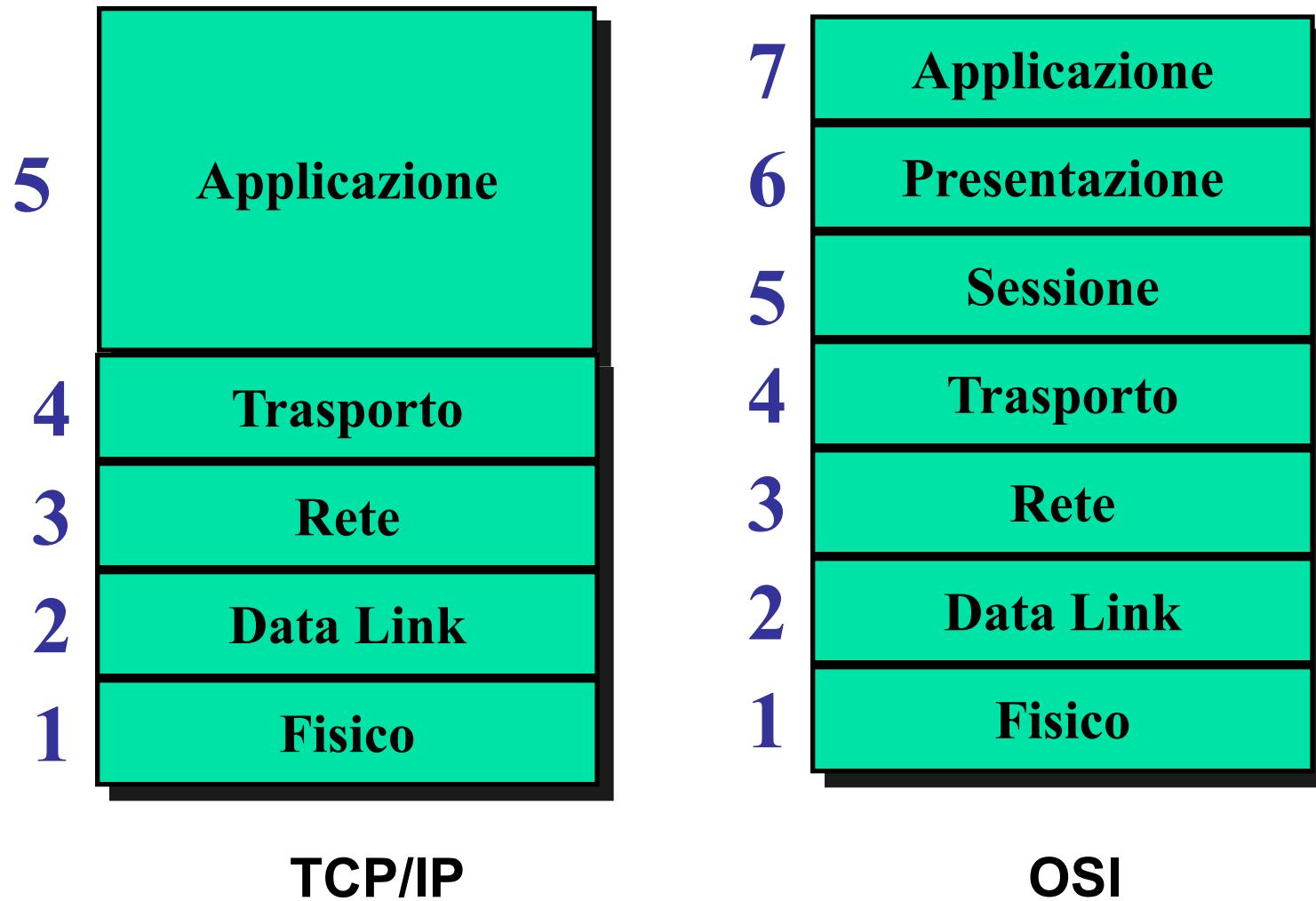


# Livello 7: Applicazione

- È il livello dei programmi applicativi, cioè di quei programmi appartenenti al sistema operativo o scritti dagli utenti, attraverso i quali l'utente finale utilizza la rete
- Esempi di applicazioni previste dall'OSI
  - VT: Virtual Terminal, connessione interattiva ad un elaboratore remoto
  - X.400: Posta Elettronica
  - X.500: Directory Service
  - ...
- Nel mondo Internet, le applicazioni sono
  - WWW (World Wide Web)
  - Mail
  - FTP (File Transfer Protocol)
  - NTP (Network Time Protocol)
  - SNMP (Simple Network Management Protocol)
  - Skype
  - Remote desktop
  - ...



# Il modello TCP/IP vs OSI





# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

---

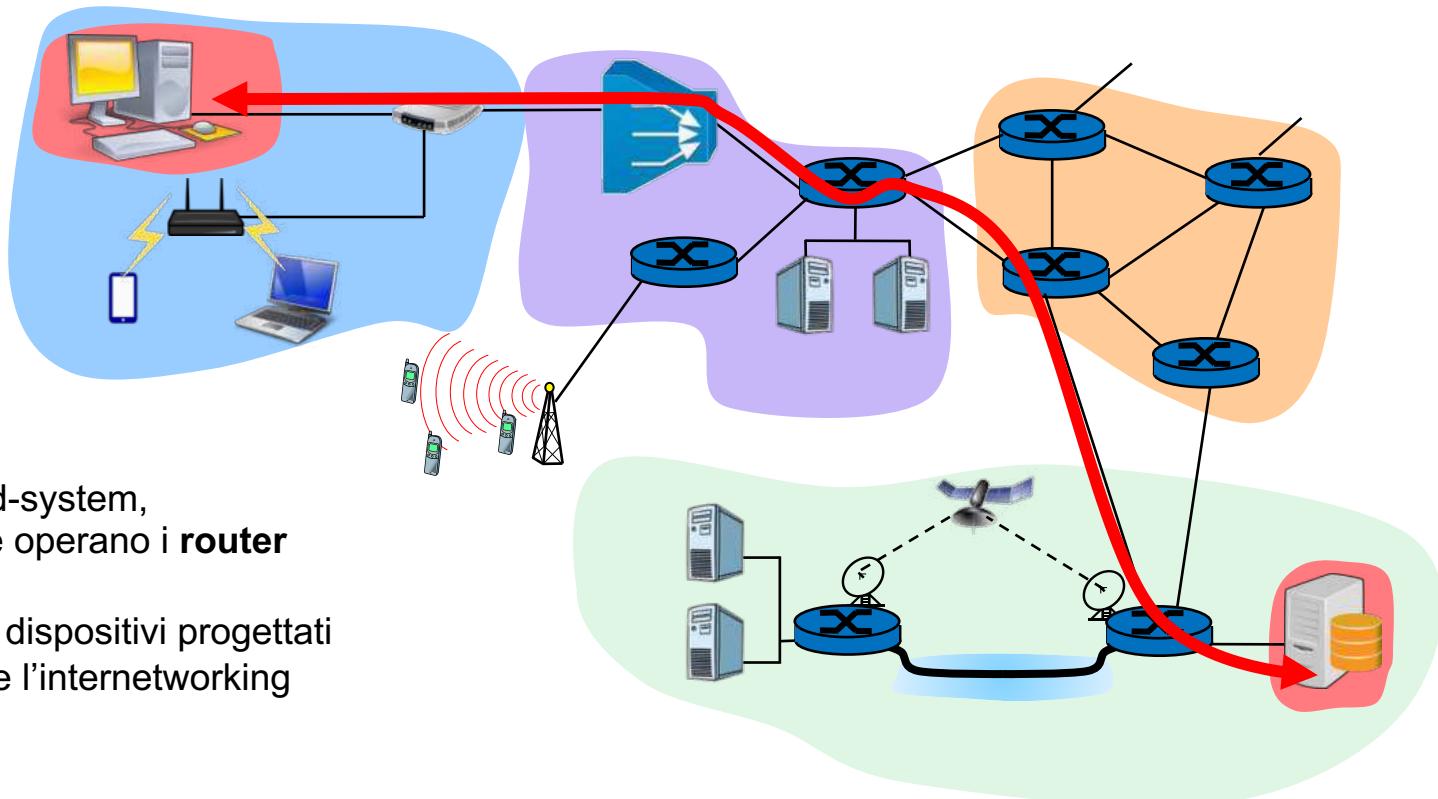
**Una prima panoramica sulle funzioni  
del livello rete e del livello trasporto in Internet**

I lucidi presentati al corso sono uno strumento didattico  
che **NON** sostituisce i testi indicati nel programma del corso

# Il compito del Livello Rete (layer 3)



In una rete di computer ottenuta attraverso la interconnessione di reti distinte (*internetwork*), il compito del **livello rete** è quello di definire i percorsi dei pacchetti nel loro transito da host mittente a host destinazione



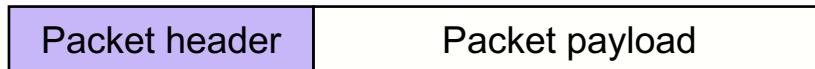
Oltre agli end-system,  
al livello rete operano i **router**

I router sono dispositivi progettati  
per realizzare l'internetworking

# Reti di calcolatori e packet switching



- Le reti di calcolatori operano secondo il modello detto ***packet switching o commutazione di pacchetto***
- In una rete a commutazione di pacchetto l'informazione è trasmessa in ***pacchetti*** formati da una intestazione (***header***) ed un ***payload***
  - l'header contiene informazioni di controllo, tra le quali un indirizzo destinazione che serve ad identificare i terminali al quale il pacchetto deve essere consegnato

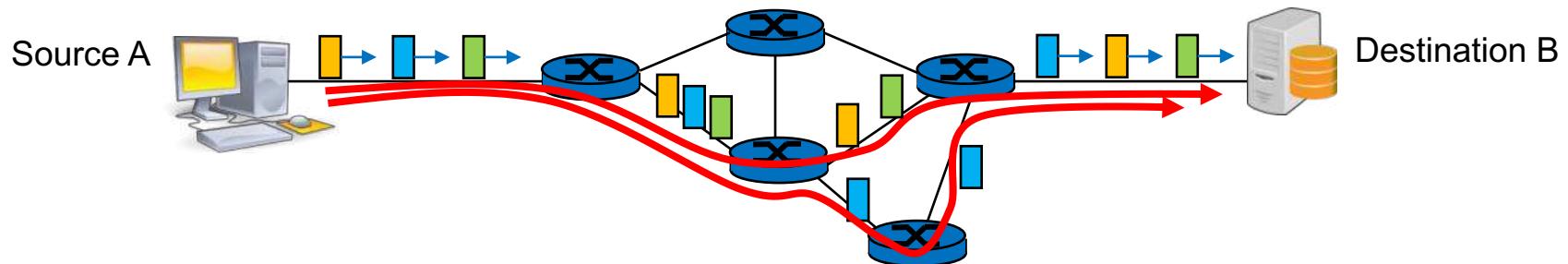


- I dispositivi intermedi che operano al livello rete funzionano in una modalità detta ***store-and-forward***
  - ogni pacchetto è ricevuto interamente, se ne controlla l'assenza di errori e se ne opera la ritrasmissione su un link di uscita
  - all'interno dei dispositivi intermedi, i pacchetti sono mantenuti in buffer di memoria gestiti come delle code

# Packet switching: modello a datagram



- In una rete a commutazione di pacchetto basata sul **modello a datagram**, ciascun pacchetto è inoltrato verso la sua destinazione indipendentemente dagli altri
  - Ogni volta che un pacchetto arriva ad un dispositivo intermedio che opera al livello rete (cioè un **router**), il dispositivo inoltra il pacchetto verso un successivo dispositivo intermedio (o verso il destinatario finale del pacchetto, qualora esso sia direttamente raggiungibile)
  - Pacchetti inviati da un terminale A verso un terminale B in momenti successivi possono seguire percorsi differenti nella rete e, quindi, arrivare a destinazione in ordine diverso da quello con il quale sono stato trasmessi



E' possibile che dei pacchetti non arrivino a destinazione

# Qualità del Servizio



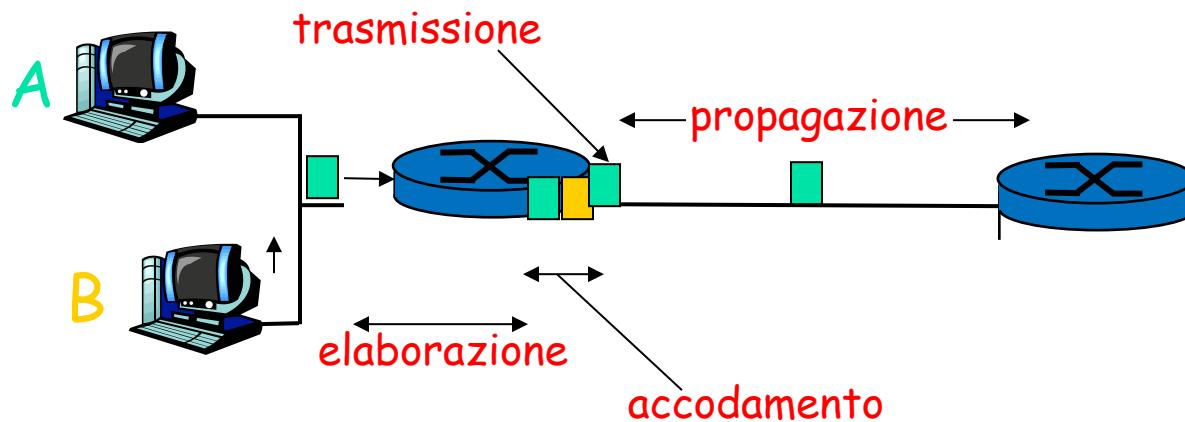
- Il servizio offerto da una rete a commutazione di pacchetto consiste nel recapitare pacchetti da un qualunque terminale mittente ad un qualunque terminale destinatario
- La **Qualità del Servizio (QoS)** di una rete a commutazione di pacchetto è misurata da una molteplicità di “indici di prestazione”
- Relativamente alla comunicazione tra due terminali collegati ad una rete, i parametri di QoS più comunemente utilizzati sono:
  - **End-to-end delay**: ritardo nella consegna dei pacchetti [s]
  - **Packet delay variation (PDV)**: variazione temporale del ritardo one-way (spesso anche indicata con il termine **packet jitter**)
  - **Throughput**: quantità di bit al secondo che la rete è in grado di trasferire tra i due terminali [b/s]
  - **Loss-Rate**: probabilità che un pacchetto non venga consegnato a destinazione

# Ritardo nelle reti a commutazione di pacchetto

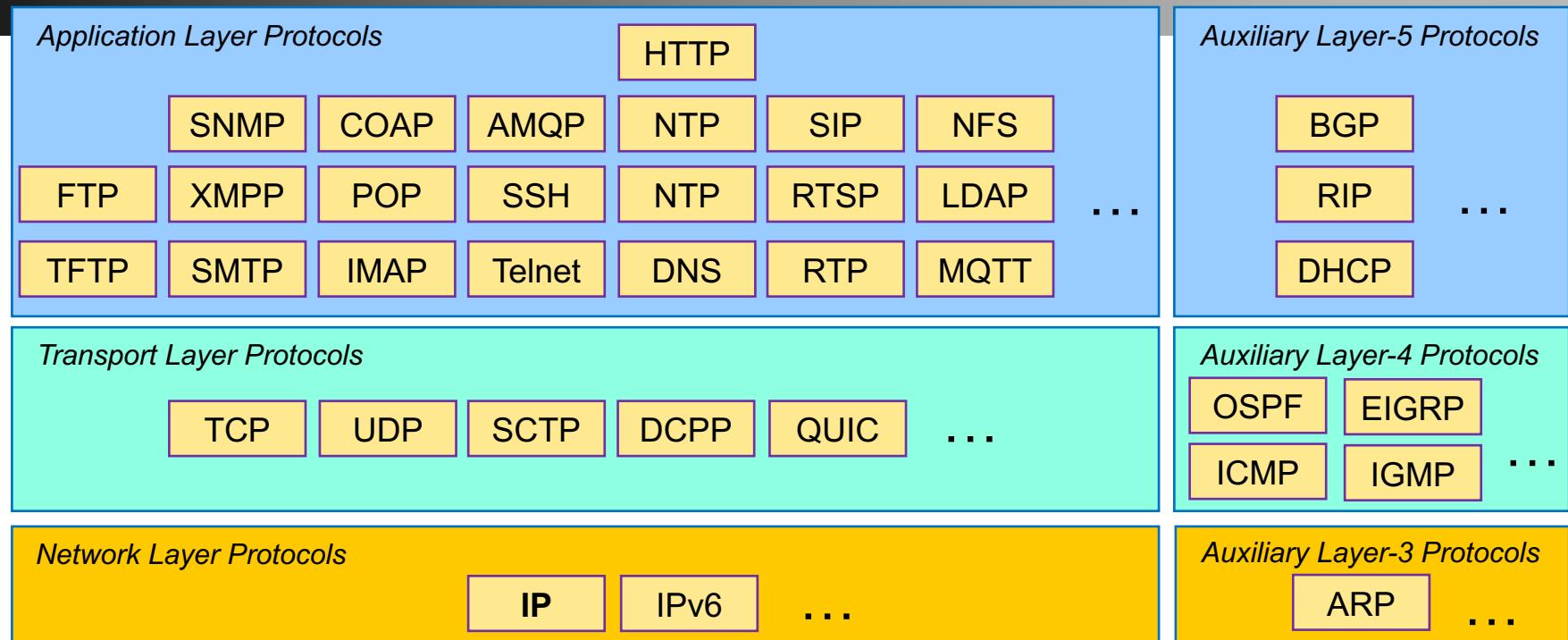


- Il ritardo nella consegna di un pacchetto alla destinazione è determinato da:

- Tempo di elaborazione nel nodo:
  - controllo di errori, determinazione link di uscita, ...
- Tempo di trasmissione su ciascun link = Lunghezza in bit / velocità in bps
- Tempo di attesa nelle code dei router (variabile)
- Tempo di propagazione sulle linee = lunghezza della linea / velocità del segnale



# La Internet Protocol suite ed il protocollo IP



- Nella rete Internet, la funzione principale del livello rete è svolta dal protocollo IP
- La versione ancora oggi prevalentemente utilizzata è la versione 4 del protocollo IP
  - IP versione 6 è progressivamente introdotto ed utilizzato
- La caratteristica principale del protocollo IP è quella di offrire un servizio di consegna elementare e senza garanzie (*best effort*) di pacchetti
  - La semplicità rende IP adattabile ad un'ampia varietà di tecnologie di livello inferiore

# Chi definisce come funziona Internet: l'IETF



- La rete Internet è una “rete di reti” basata su standard aperti
- I protocolli di comunicazione utilizzati nei livelli Rete, Trasporto ed Applicazione in Internet sono definiti da una comunità aperta di esperti detta

**Internet Engineering Task Force (IETF)**



- L'IETF è organizzata in gruppi di lavoro (*working groups*) che operano soprattutto tramite mailing list, aperte alla partecipazione di chiunque sia interessato
- Tre volte l'anno l'IETF organizza dei meeting plenari
  - IETF 101 a Londra – Marzo 2018
- I gruppi di lavoro si occupano ciascuno di uno specifico argomento e sono organizzati in aree (protocolli applicativi, sicurezza, ecc...)
- Ogni gruppo produce dei documenti detti RFC (*Request For Comments*) che vengono sottoposti alla IESG (*Internet Engineering Steering Group*) per il loro avanzamento a standard ufficiale
  - Prima di arrivare allo stato di RFC i documenti condivisi nei working group sono denominati *Internet Draft* (I-D)

# Il protocollo IP versione 4



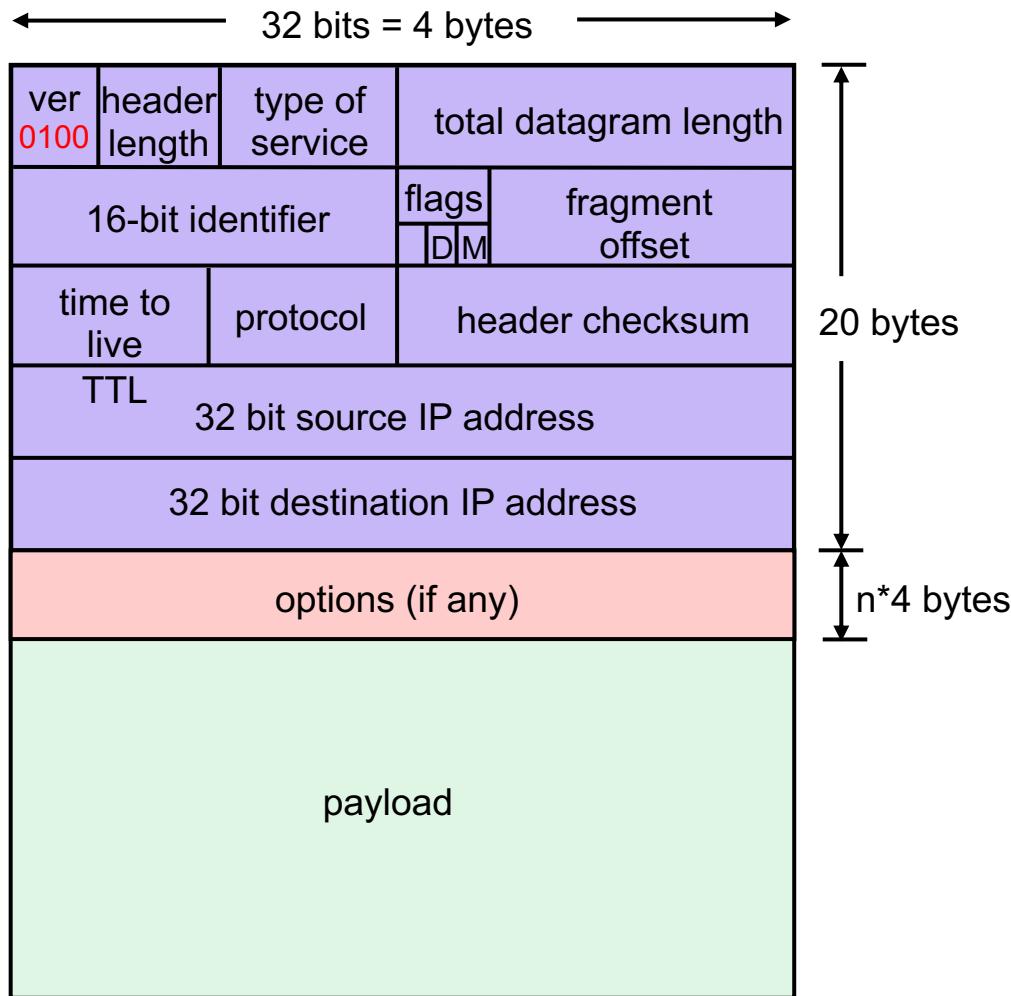
- Nella rete Internet, la funzione principale del livello rete è svolta dal protocollo IP
  - IPv4 definito in RFC 791 (settembre 1981)
- IP realizza un servizio di consegna best-effort di pacchetti singoli (*datagram*)
- Al di sopra di IP, nello stack TCP/IP (*Internet Protocol Suite*), operano i protocolli di livello trasporto (UDP e TCP)
- Il protocollo IP gestisce indirizzamento, frammentazione, ri-assemblaggio e multiplexing dei protocolli
- È implementato sia negli end-system (terminali) che nei router
- È responsabile dell'**instradamento** dei pacchetti, cioè della scelta dell'interfaccia sulla quale un pacchetto deve essere trasmesso per arrivare a destinazione
- Un datagramma IPv4 può avere una dimensione massima di 65535 byte ( $2^{16} - 1$ ) ed è costituito da un header ed un payload
- In IPv4 l'**header** è costituito da una parte a struttura fissa (20 byte) ed una opzionale
- Il **payload** è creato di norma da un protocollo di trasporto (TCP o UDP)
  - In circostanze particolari, il payload di un pacchetto IP può contenere un altro pacchetto IP: *incapsulamento IP in IP*
  - Alcuni protocolli ausiliari (cioè non intesi a supportare la comunicazione di applicazioni eseguite nei terminali) inviano i loro messaggi inserendoli direttamente in un payload IP: ICMP, IGMP, OSPF



# IP: servizio best effort

- IP non garantisce di prevenire:
  - pacchetti duplicati
  - consegna ritardata o fuori ordine
  - corruzione di dati
  - perdita di pacchetti
- La consegna affidabile dei messaggi alle applicazioni può avvenire grazie a meccanismi di controllo realizzati nei protocolli di livello superiore (negli end-system)
- Ogni router che riceve un pacchetto IP decide a quale altro nodo inoltrarlo, sulla base dell'indirizzo destinazione contenuto nel pacchetto, in maniera indipendente ...
  - rispetto agli altri router
  - rispetto agli altri pacchetti passati in precedenza per lo stesso router
- Il protocollo IP è stato progettato per realizzare un servizio *best-effort*
- Servizio best-effort significa che la rete
  - non fornisce alcuna garanzia sulla consegna di un pacchetto
  - ma non discrimina un pacchetto rispetto ad altri
    - ***network neutrality***

# Struttura di un datagram IP versione 4



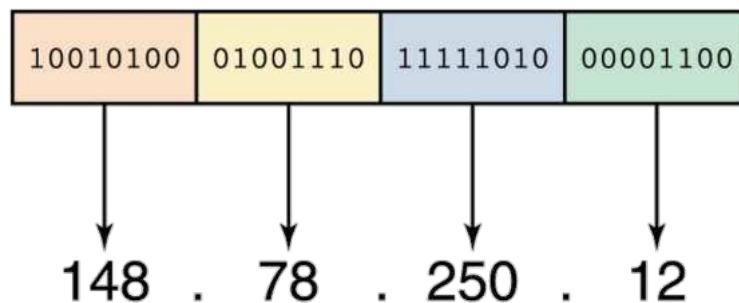
# Campi dell'header IP versione 4



- In IPv4 l'**header** è costituito da una parte a struttura fissa (20 byte) ed una opzionale di lunghezza multipla di 4 byte
- **IP header length (4 bit)**: lunghezza dell'header, in multipli di 32 bit (max 60 byte)
- **Type-of-Service (8 bit)**: specifica il tipo di servizio che si richiede alla rete
  - usato, in pratica, per scopi differenti
- **Total length (16 bit)**: indica la lunghezza in byte dell'intero pacchetto (header+dati)
- **Time-to-live TTL (8 bit)**: numero residuo di router attraversabili
  - viene decrementato di 1 da ogni router, a 0 il pacchetto viene scartato
  - serve, in caso di percorsi circolari (*loop*), ad evitare che un pacchetto resti perennemente in circolo
- **Protocol (8 bit)**: indica il protocollo di livello superiore associato al payload
  - il valore 6 indica TCP, 17 indica UDP
  - serve al de-multiplexing dei pacchetti a destinazione
- **Header checksum (16 bit)**: serve a verificare l'integrità dell'header IP
- **Source IP Address (32 bit)**: indirizzo IP del nodo mittente del pacchetto
- **Destination IP Address (32 bit)**: indirizzo IP del nodo destinatario del pacchetto
- **Identification (16 bit), Flags (3 bit), Fragment Offset (13 bit)**: sono usati in caso di frammentazione del pacchetto da parte di un router
  - consentono al nodo destinatario di ricostruire il pacchetto originario



- Un indirizzo IP è una sequenza di 32 bit
- Un pacchetto IP ha, nell'header, l'indirizzo IP del mittente e quello del destinatario
- In forma testuale, per un uso da parte di un utente umano, un indirizzo IP è solitamente rappresentato nella **notazione dotted decimal**:
  - i 32 bit sono composti in 4 byte, il valore di ciascuno dei quali è riportato in decimale come numero naturale tra 0 e 255
  - i quattro numeri decimali sono scritti in sequenza separati dal punto

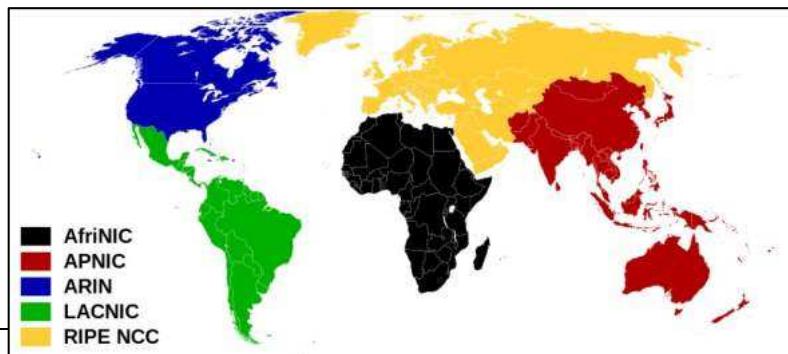


- In una rete IP (ad esempio, la rete Internet) un indirizzo IP serve ad identificare univocamente un'interfaccia di rete di un dispositivo
  - Un end system può avere una sola interfaccia di rete, un router almeno due
  - I terminali moderni hanno diverse interfacce di rete (**multi-homed**) e dunque diversi indirizzi IP (es. interfaccia Ethernet, WiFi, Bluetooth, ecc.)

# Chi assegna gli indirizzi IP



- L'assegnazione degli indirizzi IP avviene attraverso un sistema gerarchico di autorità
- Il gestore globale dell'intero spazio di indirizzamento è IANA
  - IANA - Internet Assigned Numbers Authority
  - In origine IANA era una persona: Jon Postel
- IANA dipartimento di ICANN (*Internet Corporation for Assigned Names and Numbers*)
- IANA delega la gestione degli indirizzi IP a cinque autorità regionali (RIR)
  - In Europa opera come *Regional Internet Registry* il RIPE NCC



# Funzioni di un router: forwarding e routing



- Un router è un dispositivo dotato di più interfacce di rete che serve a collegare due o più reti tra di loro
- Un router ha il compito di inoltrare pacchetti nella rete verso la destinazione finale
- All'interno del router sono esplicate due funzioni fondamentali: **forwarding** e **routing**
- La funzione di **forwarding** consiste nell'inoltrare ciascun pacchetto che entra da un'interfaccia verso un'altra interfaccia
  - L'azione di forwarding effettuata dai router deve essere coordinata, in modo da far sì che un pacchetto, generato da un qualunque host mittente, possa arrivare verso un qualsiasi host destinatario
- La funzione di **routing** ha il compito di determinare i percorsi (*path*)
- Le due funzioni sono svolte contemporaneamente da due distinte sezioni del router:

**Forwarding:** funzione esplicata dal ***data plane***

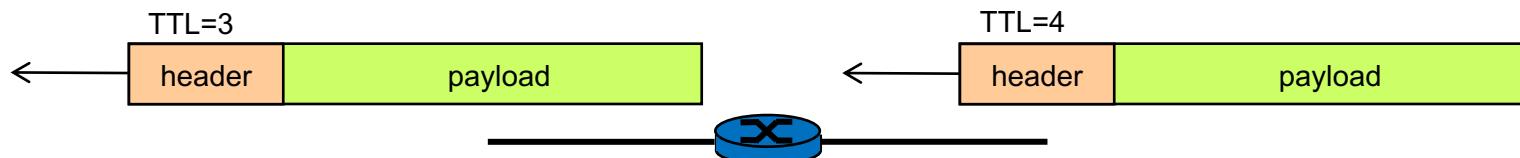
**Routing:** funzione esplicata dal ***control plane***

- Il data plane deve essere in grado di operare alla velocità dei link
  - La funzione di forwarding è tipicamente realizzata mediante hardware specializzato
- Il control plane può operare a velocità più bassa (le scelte di percorso cambiano nell'ordine dei secondi)
  - La funzione di routing è tipicamente realizzata mediante software eseguito da CPU

# Funzioni di un router IP: forwarding (1)



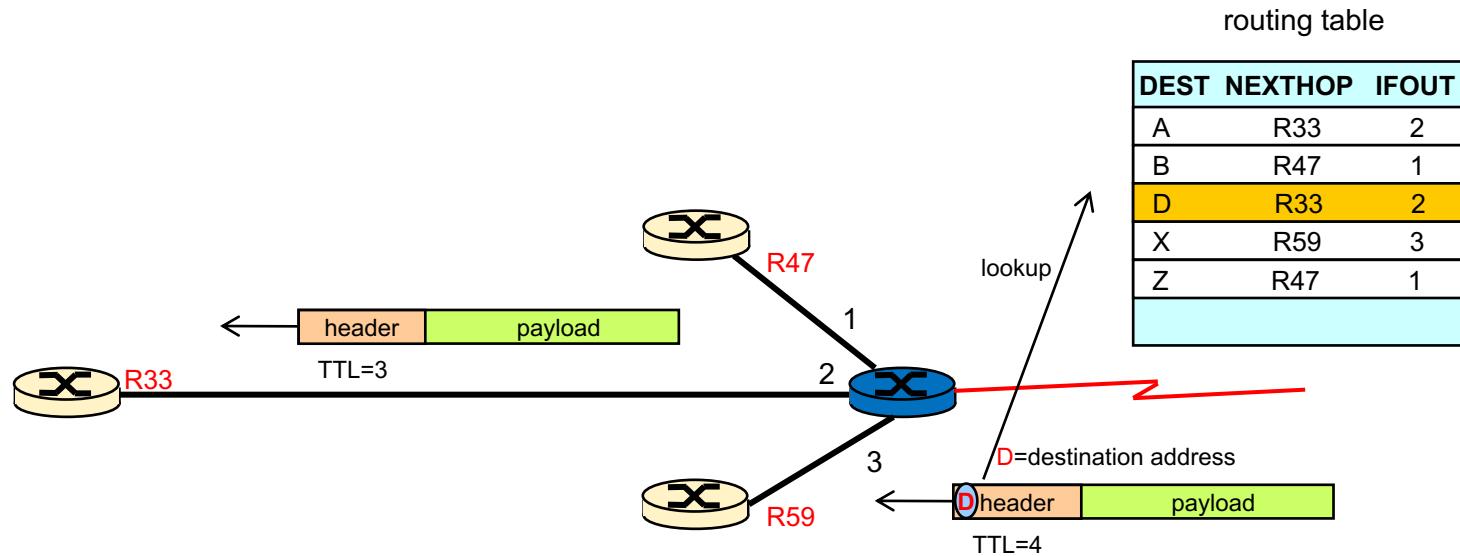
- Un router IP è un dispositivo dotato di più interfacce di rete che serve a collegare due o più reti tra di loro
- A ciascuna interfaccia di un router è assegnato un indirizzo IP appartenente alla subnet associata alla rete a cui l'interfaccia si collega
- Internamente, il router identifica le proprie interfacce mediante degli identificatori locali come fa0, eth0, eth1, ecc.
- La funzione di **forwarding** svolta da un router IP è la seguente:
  - Per ciascun pacchetto, viene determinata l'interfaccia di uscita sulla base dell'indirizzo IP destinazione contenuto nel pacchetto
  - Prima della ritrasmissione, il campo TTL (*time-to-live*) nell'header del pacchetto inoltrato viene decrementato di 1
    - Se il TTL diventa zero, il pacchetto non è inoltrato ma viene eliminato
  - La modifica del TTL impone il ricalcolo del valore del campo *header checksum*



# Funzioni di un router IP: forwarding (2)



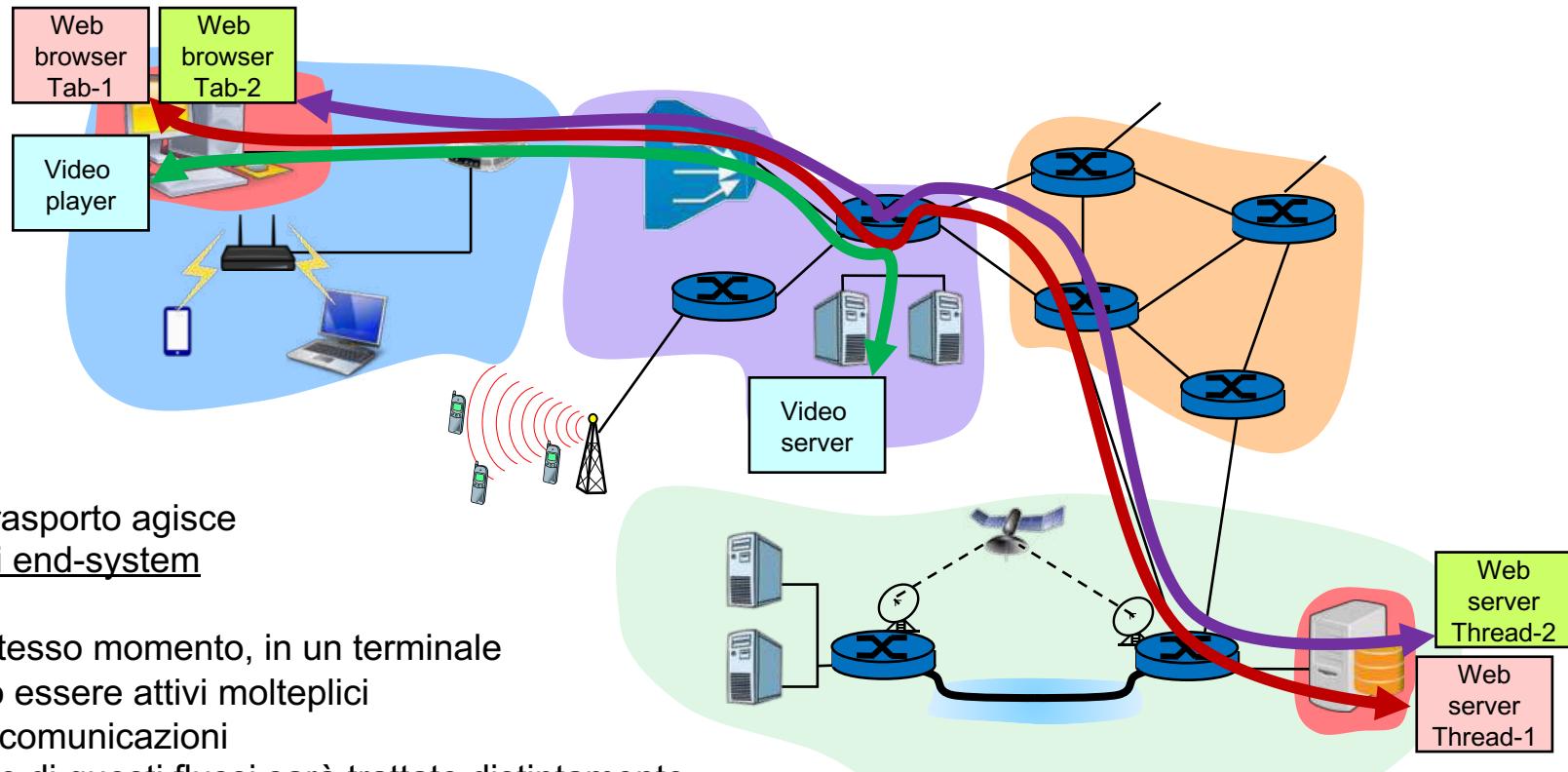
- La scelta dell'interfaccia verso la quale il router realizza la ritrasmissione è determinata dall'indirizzo IP del destinatario del pacchetto
- Tale scelta è operata sulla base delle regole di instradamento contenute in una tabella: la **tabella di routing**
- Ogni volta che il router deve inoltrare un pacchetto, viene consultata la tabella di routing per determinare l'interfaccia di uscita del pacchetto
- Il router effettua un'operazione di ricerca nella tabella (*lookup*) per determinare la regola da applicare



# Il compito del Livello Trasporto (layer 4)



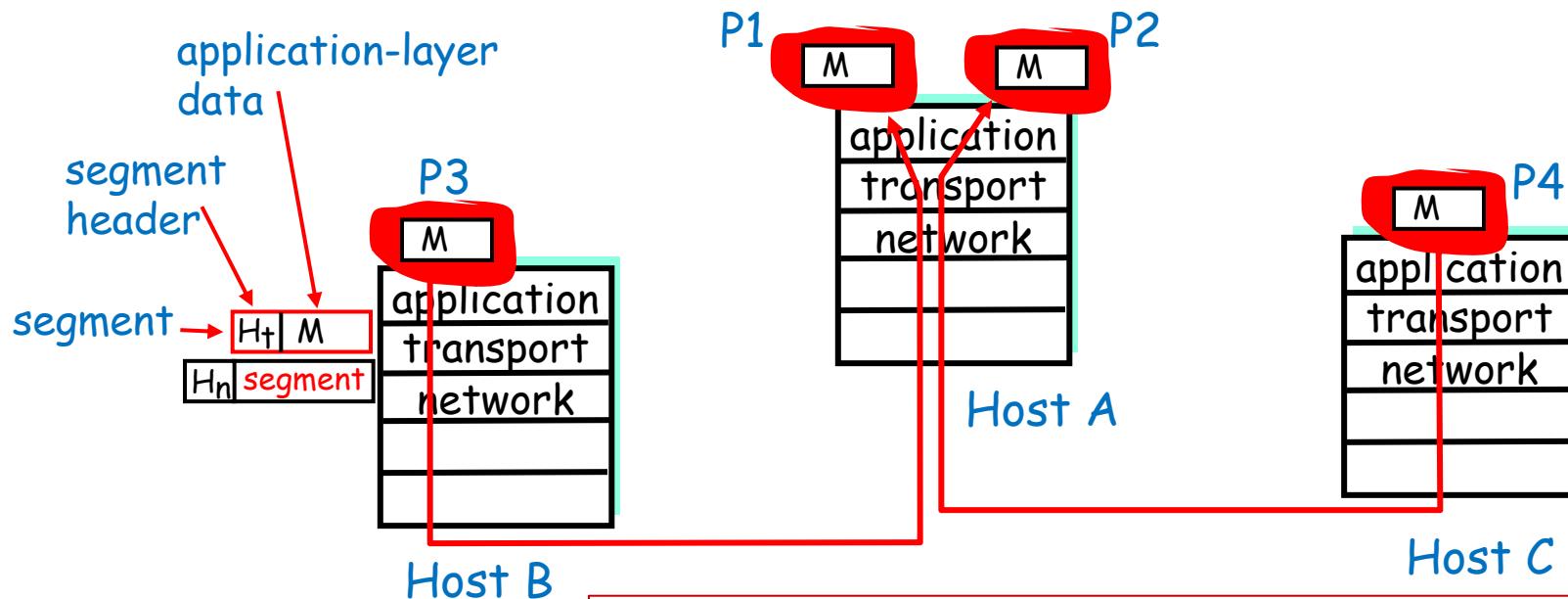
In una rete di computer, il compito del **livello trasporto** è quello di consentire alle molteplici applicazioni eseguite in un end-system lo scambio di informazioni attraverso il servizio offerto dal livello rete





# Multiplexing/demultiplexing a livello trasporto

- I diversi programmi in esecuzione concorrente all'interno di un terminale (end-system) sono detti **processi** (nella terminologia dei sistemi operativi)
- Il compito del livello trasporto è quello di consegnare i messaggi applicativi contenuti nei pacchetti che arrivano dal livello rete al processo al quale sono destinati
- Questa funzione di smistamento (*demultiplexing*) è esplicata al livello trasporto attraverso un'informazione ausiliaria contenuta nell'header dei protocolli di livello trasporto: i **port number**



Esempio: due flussi di comunicazione contemporanei in A  
1. tra i processi P1@A e P3@B  
2. tra i processi P2@A e P4@B

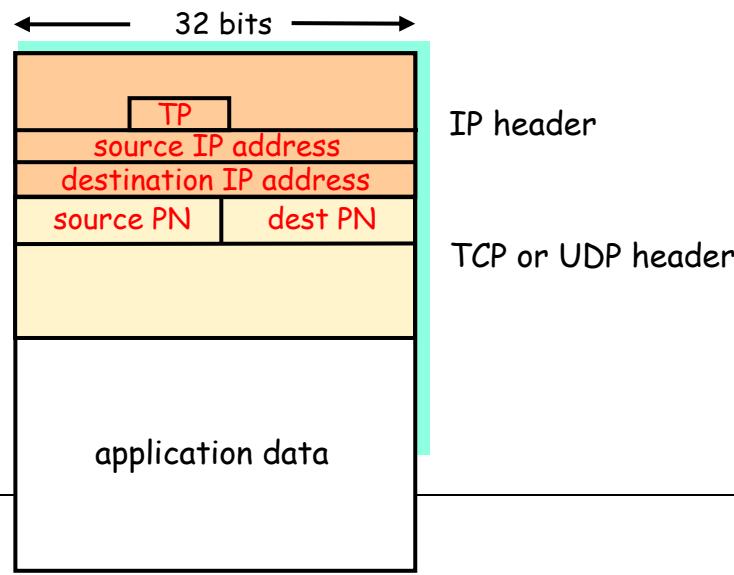
# Demultiplexing a livello trasporto: port number (1)



- Un pacchetto in arrivo ad un host è associato ad uno specifico flusso di informazione mediante la quintupla:

`Transport_protocol, Source_IP_address, Source_port_number, Destination_IP_address, Destination_port_number`

- **Transport\_protocol** (TP) è un numero naturale rappresentato su 8 bit che identifica il protocollo di livello trasporto ed è collocato nell'header di livello rete (IP):
  - 6 per TCP, 17 per UDP (valori specificati in RFC 790)
- **Source\_IP\_address** e **Destination\_IP\_address** sono gli indirizzi IP (32 bit) del mittente e destinatario del pacchetto
- **Source\_port\_number** e **Destination\_port\_number** sono numeri naturali espressi su 16 bit che servono ad identificare il flusso di informazione e sono collocati nell'header TCP e UDP
  - **Source\_port number** è scelto dall'host mittente
  - **Destination\_port\_number** è scelto dall'host destinatario



# Demultiplexing a livello trasporto: port number (2)

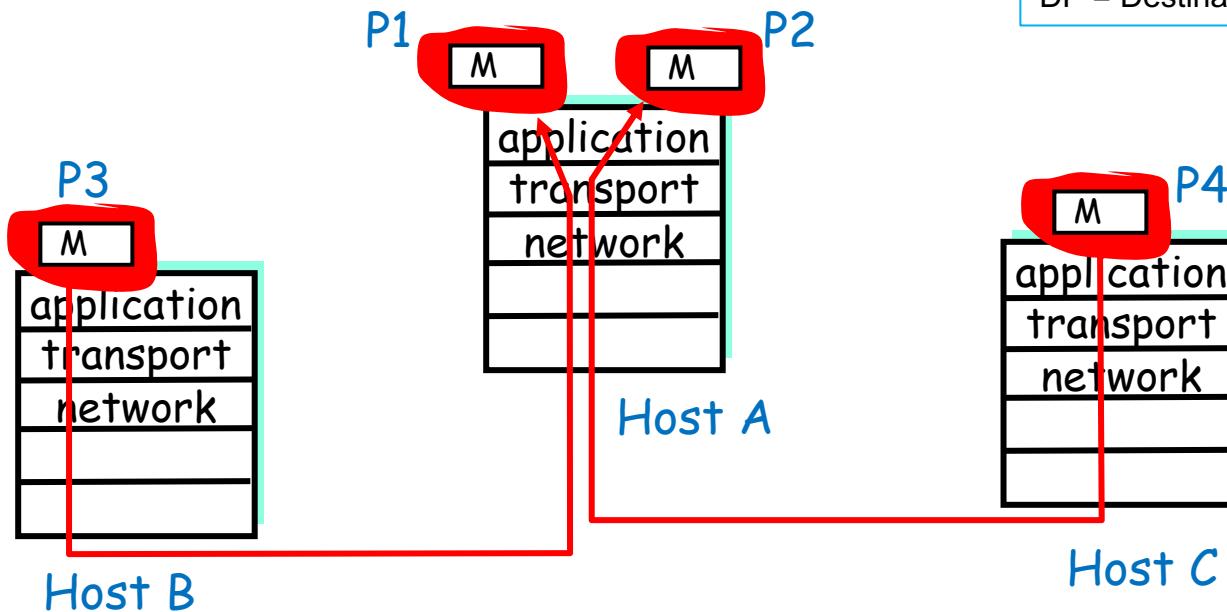


Esempio: due flussi di comunicazione contemporanei in A

1. tra i processi P1@A e P3@B con TCP
2. tra i processi P2@A e P4@C con TCP

Legenda:

TP = Transport Protocol  
SA = Source IP Address  
DA = Destination IP Address  
SP = Source Port Number  
DP = Destination Port Number



- Flusso P1@A → P3@B: TP=TCP; SA=A; DA=B; SP=80; DP=3127
- Flusso P3@B → P1@A: TP=TCP; SA=B; DA=A; SP=3127; DP=80
- Flusso P2@A → P4@C: TP=TCP; SA=A; DA=C; SP=80; DP=1984
- Flusso P4@C → P2@A: TP=TCP; SA=C; DA=A; SP=1984; DP=80

# Demultiplexing a livello trasporto: port number (3)

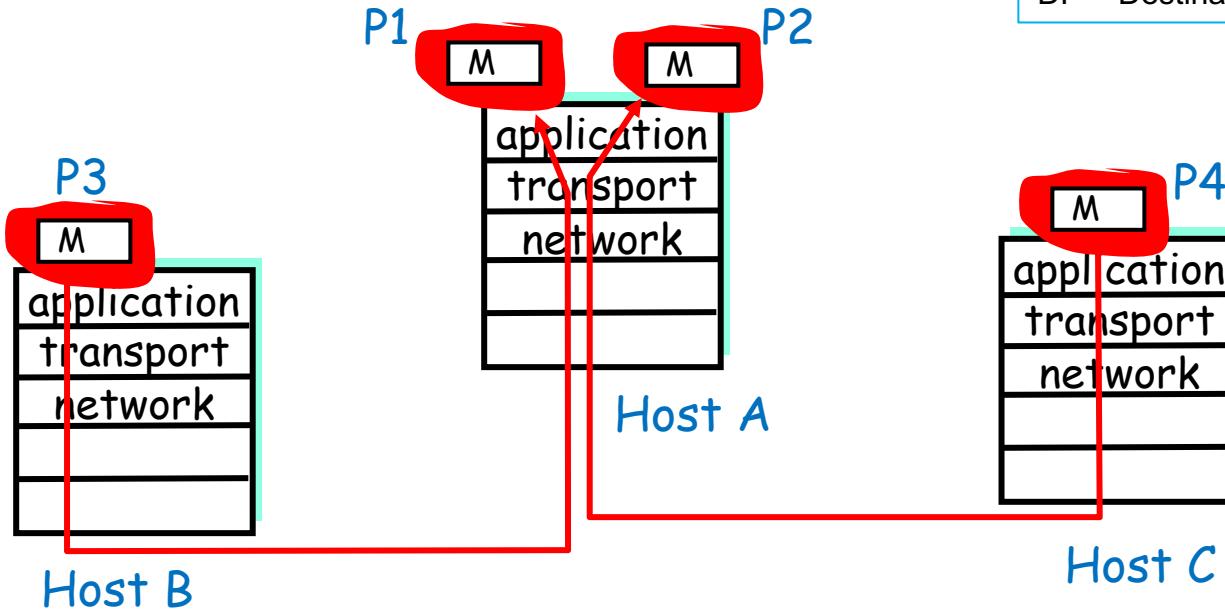


Esempio: due flussi di comunicazione contemporanei in A

1. tra i processi P1@A e P3@B con TCP
2. tra i processi P2@A e P4@B con TCP

Legenda:

TP = Transport Protocol  
SA = Source IP Address  
DA = Destination IP Address  
SP = Source Port Number  
DP = Destination Port Number

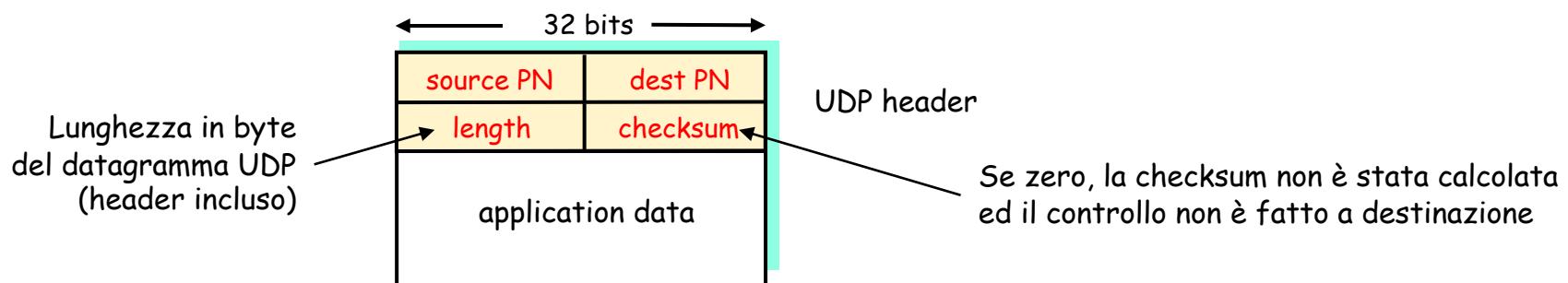


- I due host B e C possono eventualmente scegliere lo stesso valore di Port Number
- Non si genera ambiguità perché i flussi entranti in A sono distinti in base al Source IP address
  - Flusso P3@B → P1@A: TP=TCP; SA=B; DA=A; SP=3127; DP=80
  - Flusso P4@C → P2@A: TP=TCP; SA=C; DA=A; SP=3127; DP=80

# Protocollo UDP: modello di servizio e header



- Offre alle applicazioni un servizio di consegna best-effort ma senza garanzie di messaggi
- UDP forma un datagram IP per ciascun messaggio applicativo ricevuto dall'applicazione
- Non è garantita né la consegna né il rispetto dell'ordine di sequenza
- In sostanza aggiunge al servizio di IP solo il multiplexing attraverso il port number
- Opzionalmente, UDP effettua un controllo di correttezza del datagramma ricevuto tramite una checksum; se il controllo da esito negativo, il datagramma UDP è scartato
- Le applicazioni che usano UDP devono, se lo ritengono opportuno, implementare meccanismi che implementino contromisure nel caso di perdita di datagrammi o di consegna fuori ordine
- Le applicazioni che usano UDP sono orientate a semplici interazioni richiesta-risposta



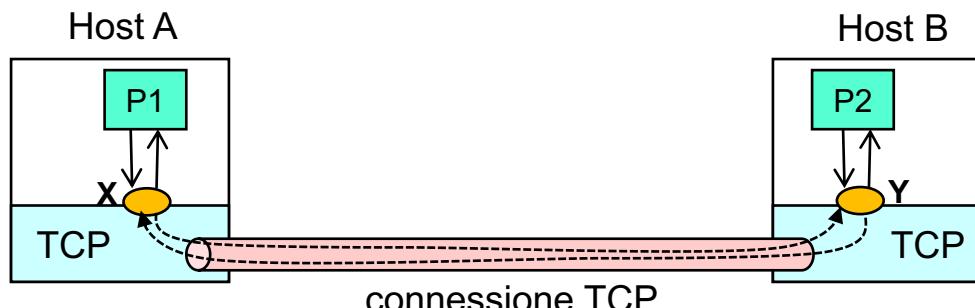
# Protocollo TCP: modello di servizio



- TCP offre alle applicazioni un servizio di consegna senza perdite e con rispetto dell'ordine di uno stream di byte bidirezionale tra due endpoint
- Il servizio è di tipo ***connection-oriented***, cioè, prima di iniziare la trasmissione dei dati, occorre stabilire una associazione (connessione) tra i due endpoint attraverso una procedura di *connection establishment*
  - E' anche prevista una procedura di *connection teardown* per l'eliminazione di una connessione precedentemente stabilita tra due endpoint
- Una connessione TCP tra due endpoint X ed Y consente un flusso di dati bidirezionale (da X ad Y e viceversa)
- Un pacchetto è associato ad una connessione TCP mediante la quintupla:

`Transport_protocol, Source_IP_address, Source_port_number, Destination_IP_address, Destination_port_number`

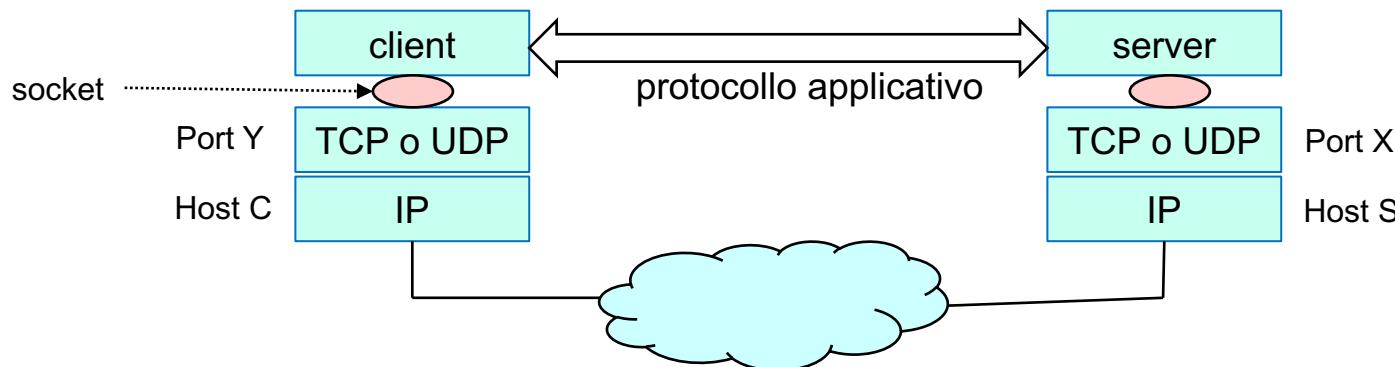
- Negli end-system, i processi si “agganciano” agli endpoint di una connessione TCP mediante opportuni strumenti di comunicazione previsti dal sistema operativo (*socket*)



# Applicazioni client/server e protocolli applicativi



- Il modello comportamentale a cui si ispira la maggioranza delle applicazioni distribuite è il **modello client-server**:
  - un **processo server** è in esecuzione su un host S, e si rende disponibile alla comunicazione attraverso il protocollo TCP o UDP, ad un numero di porta X noto a priori (*well-known*) ai client
  - un **processo client** viene eseguito su un host C, acquisisce dal sistema operativo in esecuzione sull'host C la disponibilità all'uso del numero di porta Y e "chiede" al sistema operativo di stabilire una comunicazione tramite TCP o UDP con il processo server, del quale conosce a priori l'indirizzo IP S ed il port number X
- L'interazione tra le componenti client e server delle applicazioni è governata da **protocolli di livello applicativo**
- Un protocollo applicativo definisce il formato dei messaggi ed i pattern di interazione





# Protocolli applicativi e protocolli di livello trasporto

- La suite TCP/IP mette a disposizione delle applicazioni due servizi di livello trasporto:
  - Connectionless, senza garanzie, orientato ai messaggi: tramite UDP
  - Connection-oriented, con garanzie di consegna in ordine e senza perdite: tramite TCP
- La scelta tra TCP o UDP dipende dai requisiti dell'applicazione
  - Per applicazioni che si basano su semplici interazioni richiesta-risposta è conveniente usare il protocollo UDP
  - Per applicazioni che devono realizzare trasferimenti di dati significativi tra due processi è conveniente usare il protocollo TCP
  - Per classi di applicazioni con requisiti particolari sono stati definiti ulteriori protocolli di trasporto
    - Ad esempio, per le applicazioni che trasmettono flussi di dati time-sensitive è stato realizzato il protocollo di trasporto RTP, che è implementato “su UDP” (l'header RTP è inserito nel payload di un datagramma UDP)
- La definizione di protocolli applicativi standard consente di svincolare i servizi da una specifica implementazione
  - Ad esempio, il servizio World-Wide Web è accessibile da una molteplicità di client (*browser*)
    - Internet Explorer, Firefox, Chrome, Opera, ecc.
  - Lo stesso servizio WWW è erogabile da una molteplicità di server:
    - Apache Web Server, Microsoft IIS, NGINX, ecc.



# Protocolli applicativi standard

- Uno sviluppatore che realizzi un'applicazione client/server è libero di definire ed implementare il proprio protocollo applicativo (purché sviluppi sia la componente client che quella server)
- I servizi offerti tramite Internet si attengono a protocolli applicativi standard definiti dall'IETF e definiti in specifici documenti RFC (*Request For Comments*)

Applicazione	Protocollo applicativo	Protocollo di trasporto
world-wide web	HTTP [RFC 2068,2616]	TCP
DNS	DNS [RFC 1034,1035]	UDP
file transfer	FTP [RFC 959]	TCP
e-mail (invio)	SMTP [RFC 821]	TCP
E-mail (lettura)	POP3 [RFC 1939]	TCP
remote file server	NFS [RFC 1813]	TCP o UDP
remote terminal access	TELNET [RFC 854]	TCP
VOIP (segnalazione)	SIP [RFC 3261]	UDP
VOIP (flussi multimediali)	-	RTP su UDP
network management	SNMP [RFC 3416]	UDP



# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

---

**Introduzione alla architettura di Internet  
ed ai protocolli TCP/IP**

**I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso**

# Nota di copyright per le slide COMICS



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

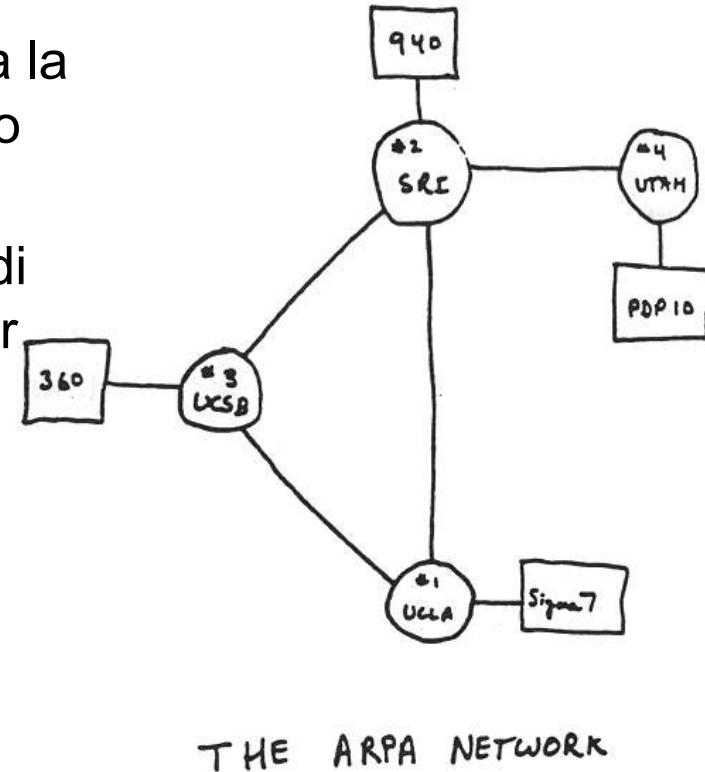
Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta



# Internet: Origini ed Evoluzione

- Internet nasce come interconnessione di diverse reti di carattere sperimentale
- Kleinrock attraverso la teoria delle code mostra la fattibilità delle reti a commutazione di pacchetto (1961)
- Prime attività di rilievo come sperimentazione di reti a commutazione di pacchetto finanziate per scopi militari (Department of Defence – DoD, tramite Defence Advanced Research Projects Agency – DARPA) (1964)
- ARPANET: Rete sperimentale basata su Interface Message Processors (IMP) con meccanismi Store-and-Forward (1967/1969)
- Approccio Datagram per garantire la sopravvivenza in caso di eliminazione di nodi e linee





# Internet: Origini ed Evoluzione

- Un primo gruppo di Università crea un nucleo di rete per sperimentare applicazioni (1969-72)
  - Collegamenti da 56 kbps
- Le sperimentazioni mostrano l'inadeguatezza dei protocolli ARPANET per differenti architetture
  - Esigenza di protocolli per “internetworks”
- V. Cerf e R. Kahn progettano la suite TCP-IP (1974)
- BBN e UCB vengono finanziate per inserire TCP-IP in Unix BSD
- Quattro fattori di successo: PDP/Vax, LAN, TCP-IP, Unix



# Internet: Origini ed Evoluzione

- 1984: la parte militare di ARPANET si separa (MILNET)
- 1986: La *National Science Foundation* (NSF) finanzia lo sviluppo di una rete basata su TCP-IP (NSFNET)
  - NSFNET ha una struttura gerarchica: una dorsale ad alta velocità ed una serie di reti regionali
  - Collegamenti da 56 Kbps a T1 (1,544 Mbps)
- 1989: ARPANET è smantellata
- 1990: NSF smette di finanziare la rete e cede la struttura ad una organizzazione non-profit
  - Nasce *Advanced Network and Services* (ANS), una organizzazione fondata da Merit, IBM ed MCI
  - Collegamenti da T1 a T3 (45 Mbps)
- 1995: NSFNET è smantellata
  - Nascono i *Network Access Point* (NAP) per interconnettere varie reti commerciali



# Internet: architettura della rete

- **Una architettura completa, dai servizi alle applicazioni**
- **È di dominio pubblico e di diffusione enorme**
- **In evoluzione con i requisiti degli utenti**
- **Gestita da specifici organismi**
  - Internet Society (ISOC): IETF, IRTF
  - IANA/ICANN
- **I protocolli in uso sono standardizzati mediante documenti approvati dall'IETF: le “Request for Comments” (RFC):**
  - Informational e Standard Track
- **Opera su tecnologie di rete standard e non**
  - SLIP, PPP, Dialup
  - LAN 802.X, FDDI
  - X.25, FR, ATM, SMDS

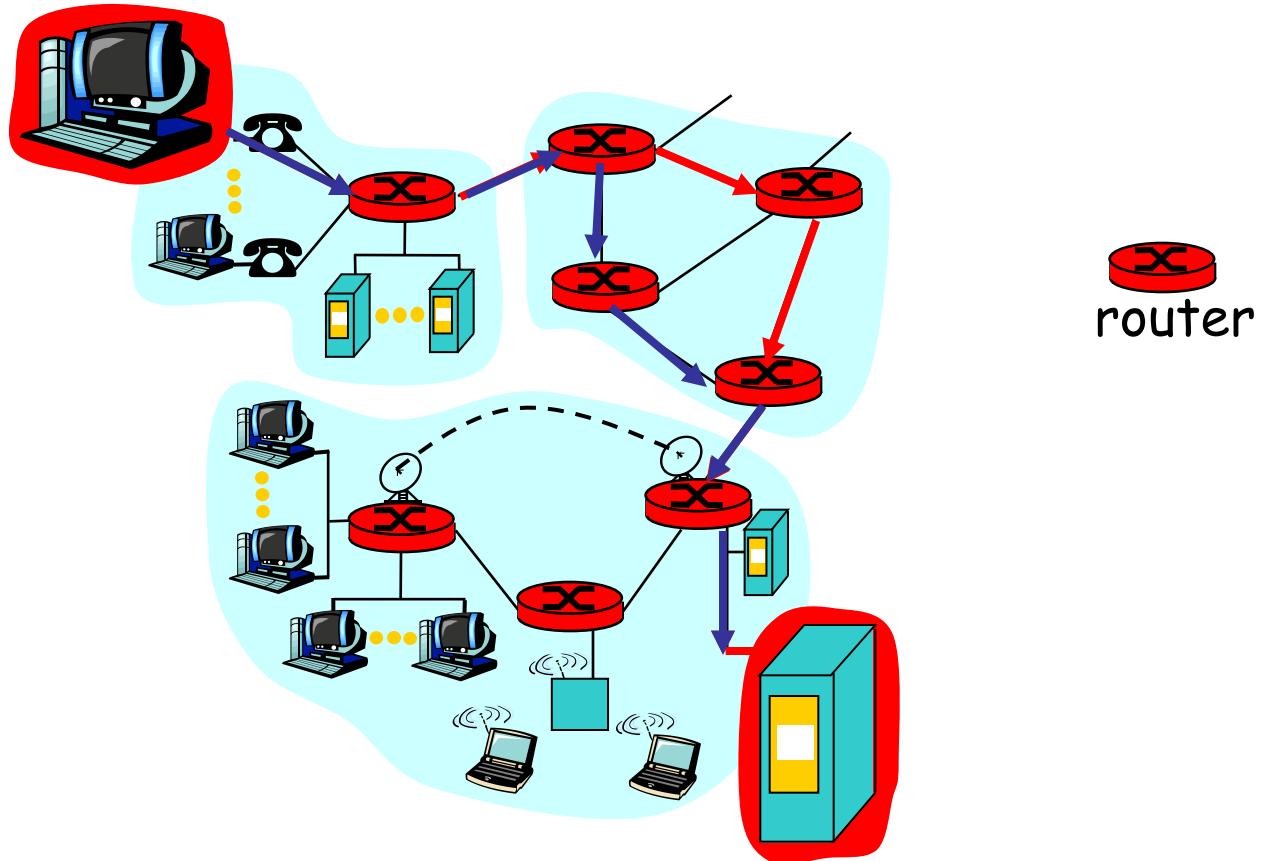


# Internet: architettura della rete

- La rete è progettata secondo un modello a **datagram**
- L'informazione viaggia in pacchetti (*datagram*) che vengono trattati dalla rete indipendentemente l'uno dagli altri
- Ogni terminale è univocamente individuato da un indirizzo associato alla interfaccia che lo collega alla rete
- Ogni pacchetto contiene l'indirizzo del mittente e l'indirizzo del destinatario
- L'infrastruttura della rete è costituita dai **router** che hanno il compito di instradare i pacchetti e consegnarli a destinazione
- Non c'è garanzia che un pacchetto venga realmente consegnato a destinazione
  - I pacchetti possono andare persi nella rete
  - I pacchetti possono seguire percorsi diversi ed arrivare in un ordine diverso da quello con cui sono stati trasmessi



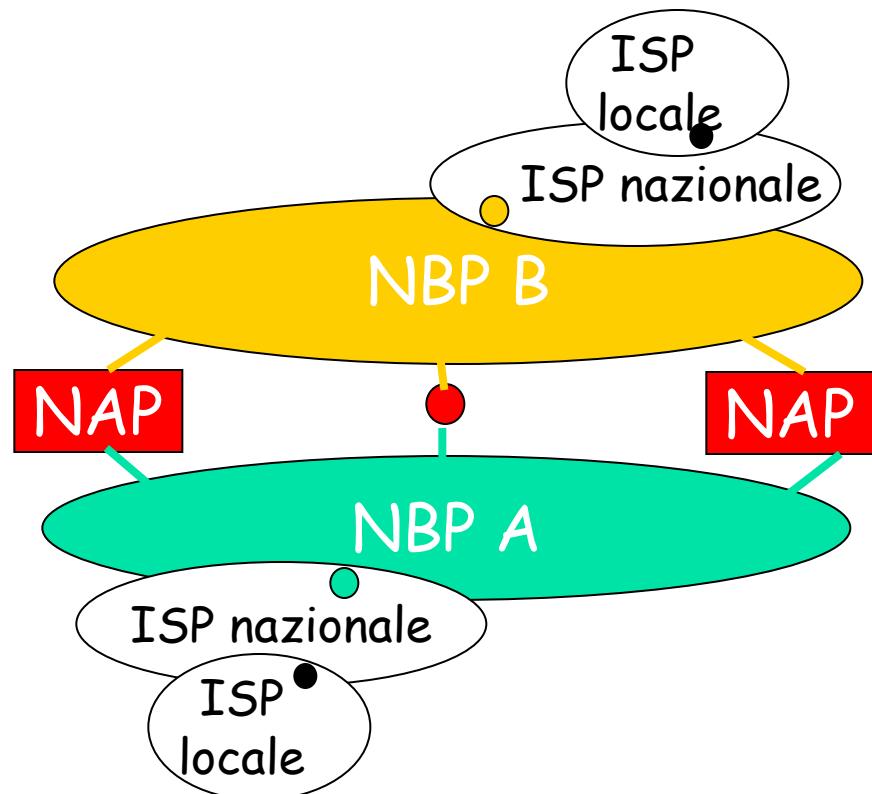
# Internet: architettura della rete





# Struttura di Internet

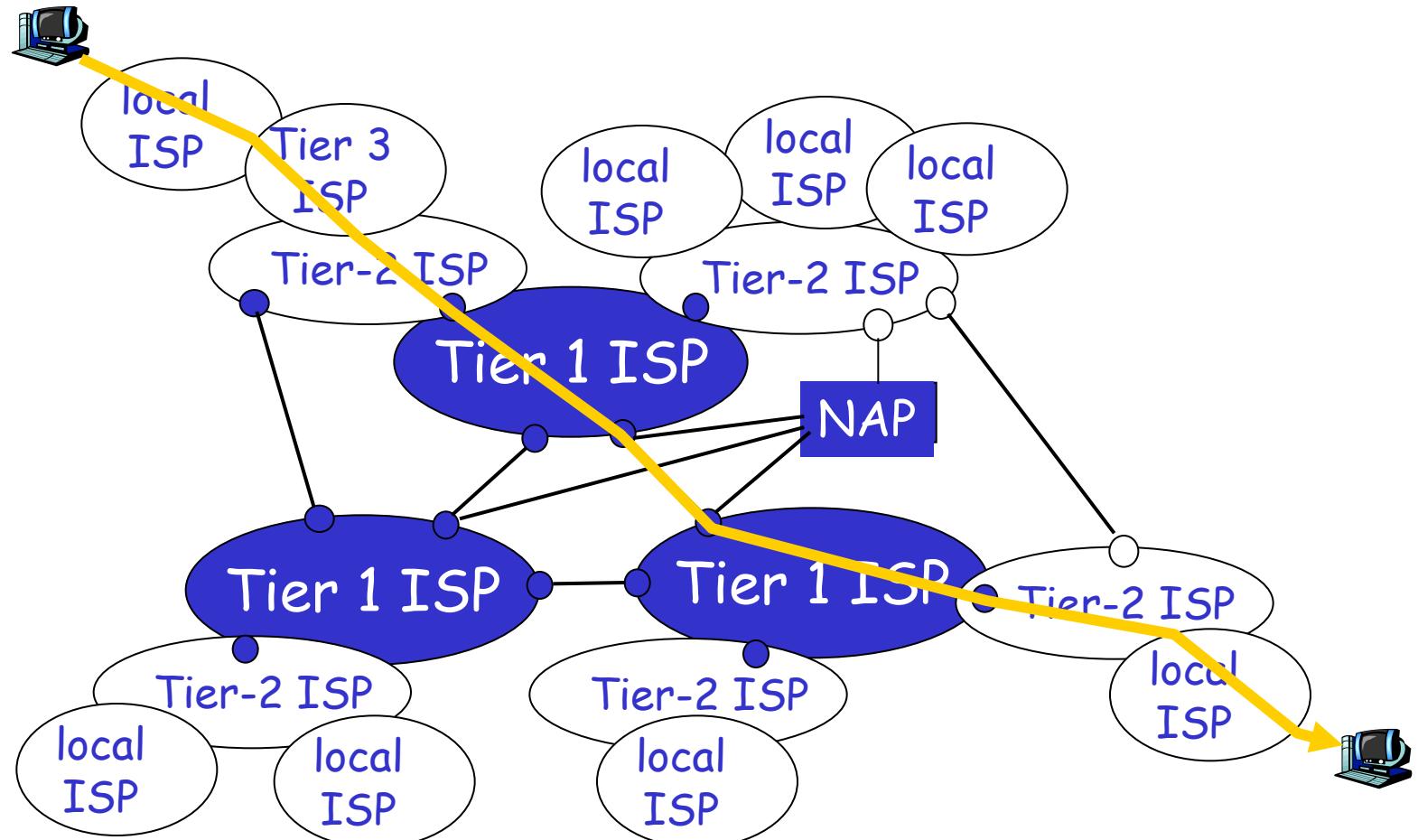
- L'accesso ad Internet avviene per mezzo di un fornitore di servizi o *Internet Service Provider*, ISP
- Gli ISP sono collegati tra loro secondo una struttura gerarchica
  - ISP locali
  - ISP nazionali
- Gli ISP nazionali si collegano a fornitori di connettività internazionali: i *Network Backbone Provider* (NBP)
  - BBN/GTE, Sprint, UUNet
- Gli NBP sono tra loro collegati in punti di interscambio detti NAP, *Network Access Point*





# Struttura di Internet: rete di reti

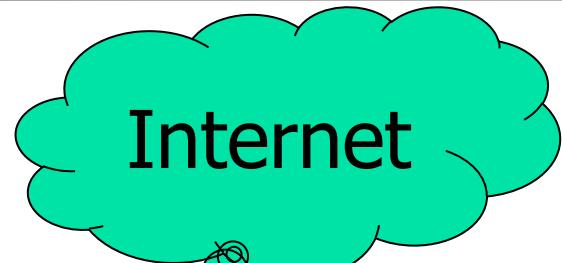
- Dalla sorgente alla destinazione, un pacchetto attraversa diverse reti.



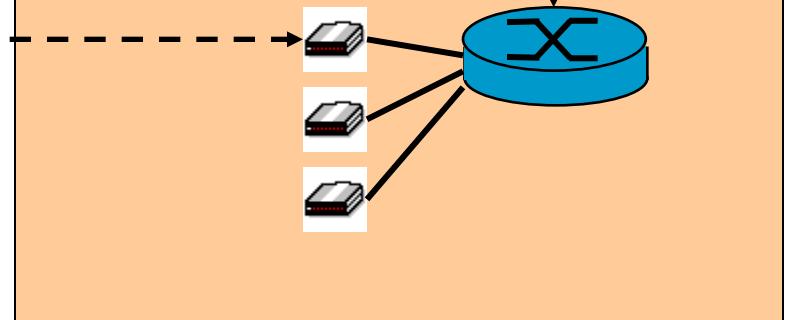


# Accesso ad Internet residenziale

Personal Computer +  
Modem



ISP

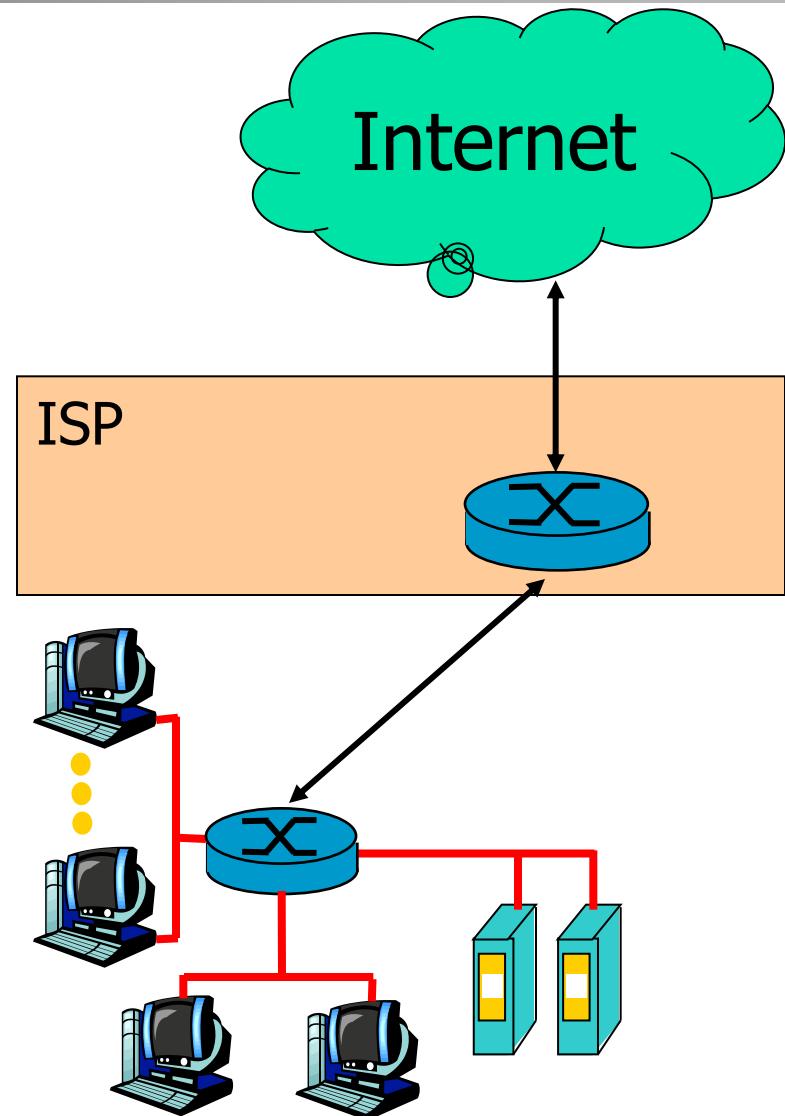


- ADSL (*Asymmetric Digital Subscriber Line*)
  - fino ad 1 Mbps upstream e fino a 20 Mbps downstream
- Fibra ottica
  - Fino 1 diversi Gbps full duplex (offerte fino a 100Mbps)



# Collegamento ad Internet di LAN aziendali

- Collegamento mediante:
  - ADSL, Fibra ottica
    - Per reti aziendali di piccole dimensioni
  - Linea dedicata con collegamento permanente con l'ISP
    - Per reti aziendali di medie/grandi dimensioni





# Internet in Italia

- Esistono degli *Internet Service Provider* commerciali che operano a livello nazionale fornendo la connessione ad Internet a privati ed aziende
  - Telecom Italia
  - Infostrada
  - FastWeb
  - Tele2
  - Tiscali
  - ...
- Esistono anche ISP che operano su scala locale



# La rete GARR-B

L'accesso ad Internet per le Università è gestito dal GARR

*Gruppo  
Armonizzazione  
Reti  
di Ricerca*



1989



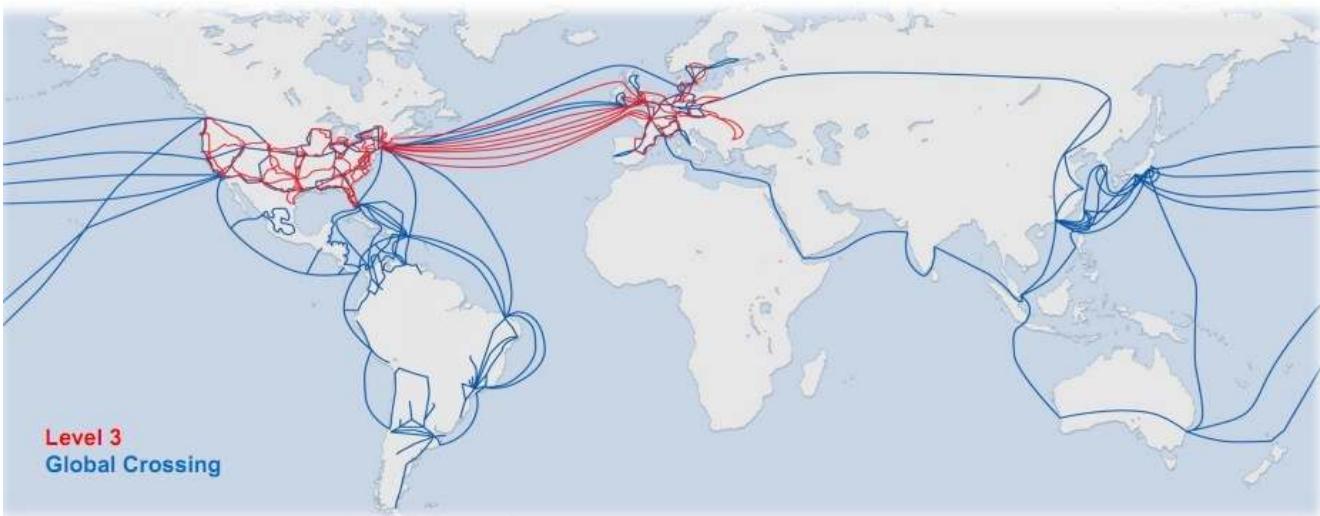
2004



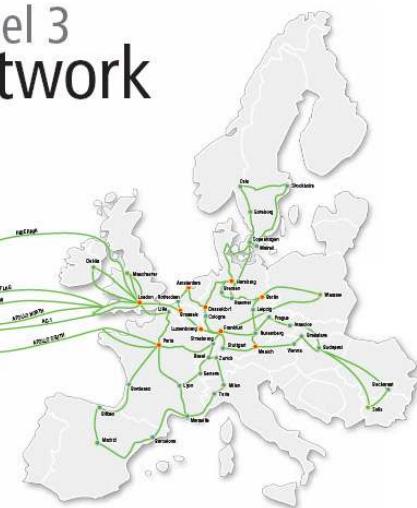
2015/09



# La topologia di Level 3



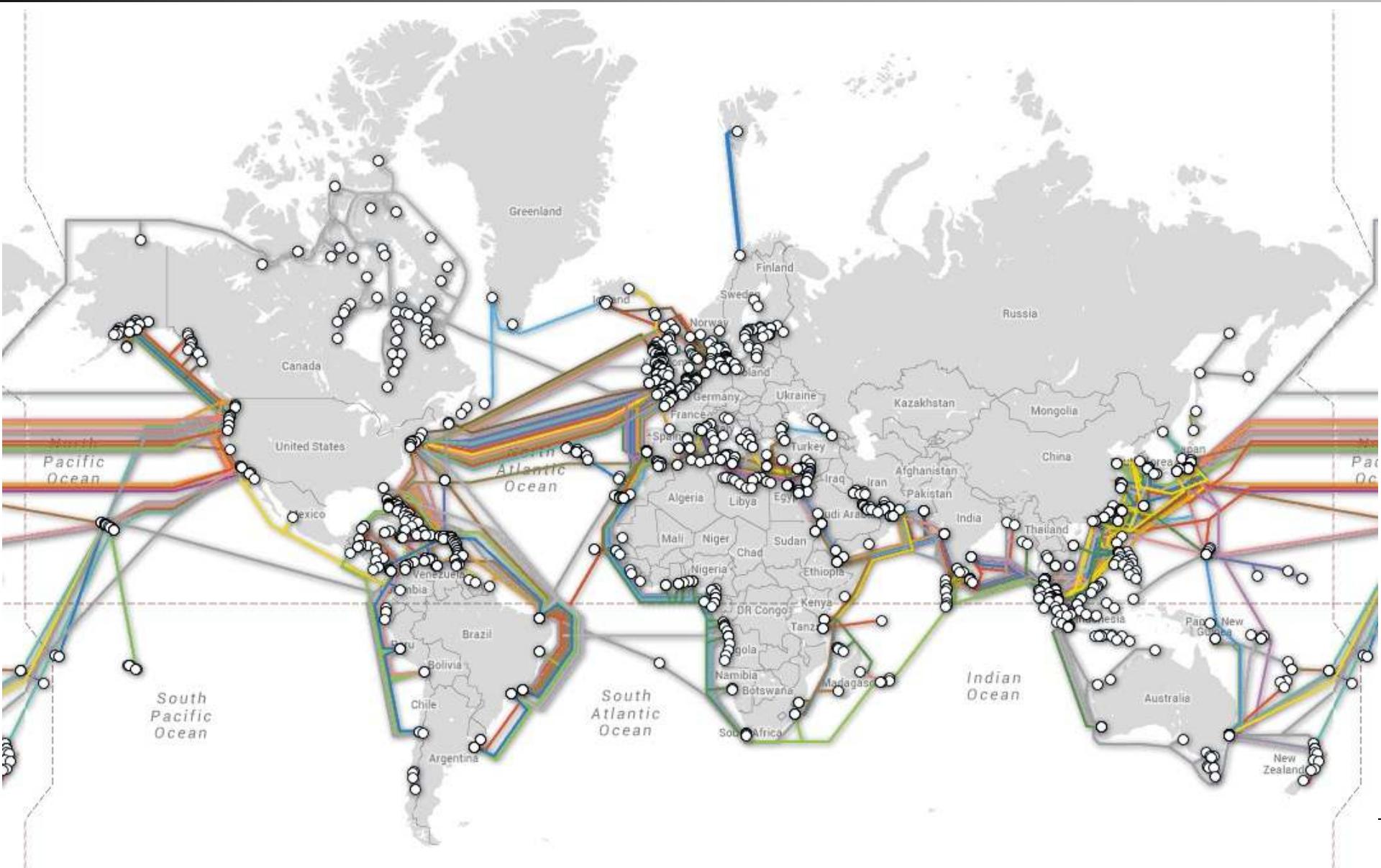
The Level 3 Network



- On-Net Market with Metro Fiber Network
- On-Net Market
- Multiple Tier 2 Metro Routes
- Longhaul Network  
European network includes wavelength capacity.



# (alcuni) collegamenti transoceanici





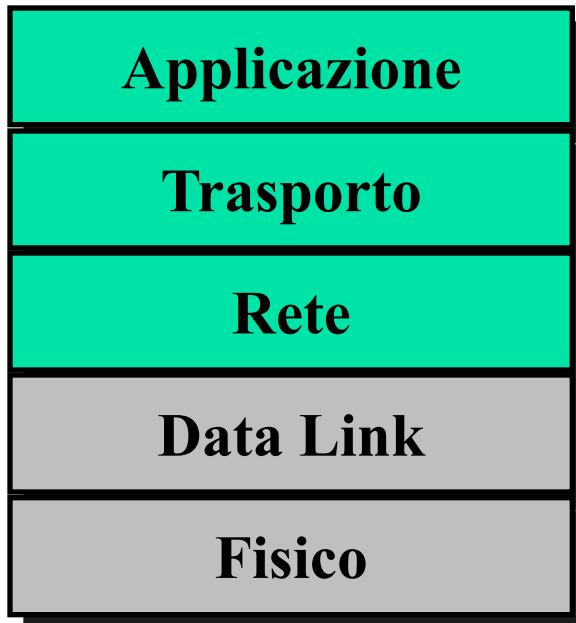
# Chi regolamenta Internet

- La standardizzazione dei protocolli in uso su Internet è fatta dall'*Internet Engineering Task Force*, IETF ([www.ietf.org](http://www.ietf.org))
- L'assegnazione degli indirizzi e dei nomi di dominio è oggi supervisionata dall'*Internet Corporation for Assigned Names and Numbers*, ICANN, che ha preso il posto dell' *Internet Assigned Numbers Authority*, IANA, una authority federale degli USA



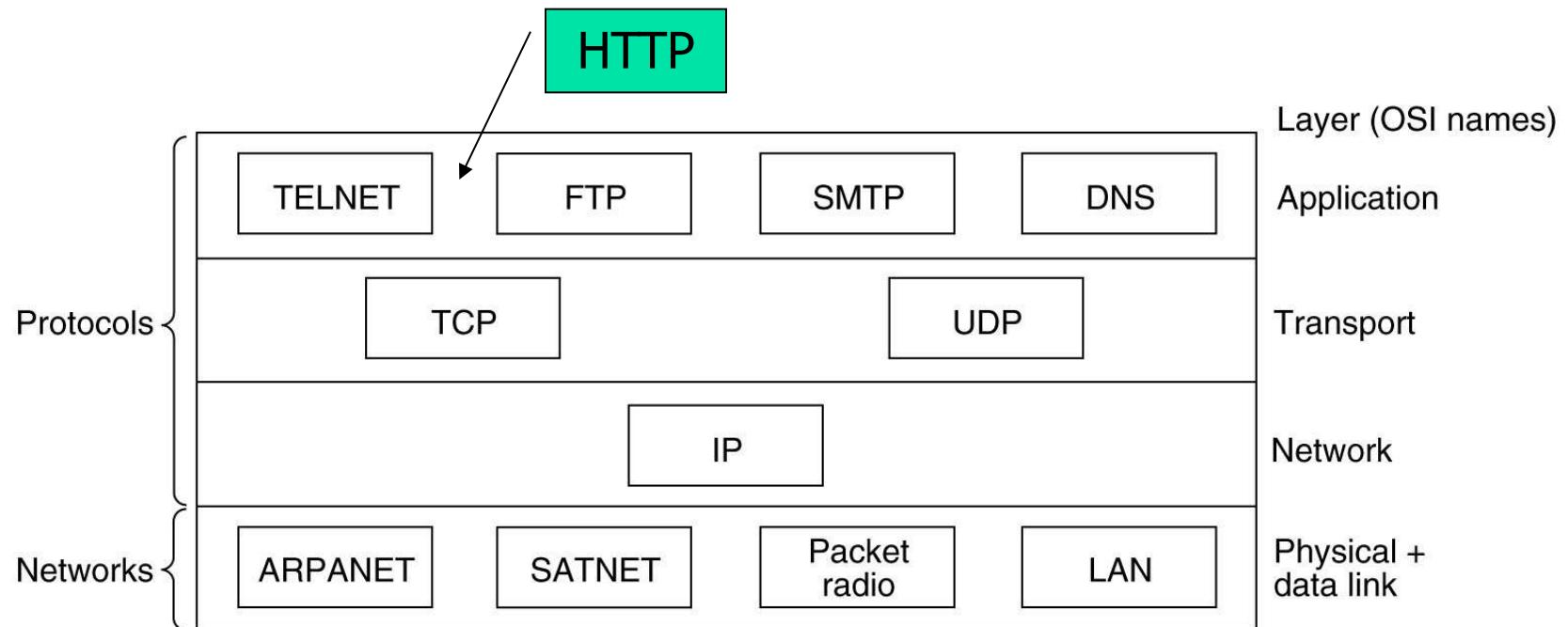
# Lo stack TCP/IP

- Internet si basa su un modello definito da una collezione di protocolli standardizzati dall'IETF, il modello TCP/IP
- Siccome i protocolli sono organizzati secondo una struttura a pila (*stack*), si parla dello “stack TCP/IP”
- Il modello prende il nome da due protocolli fondamentali:
  - TCP, *Transmission Control Protocol*, di livello Trasporto
  - IP, *Internet Protocol*, di livello Rete





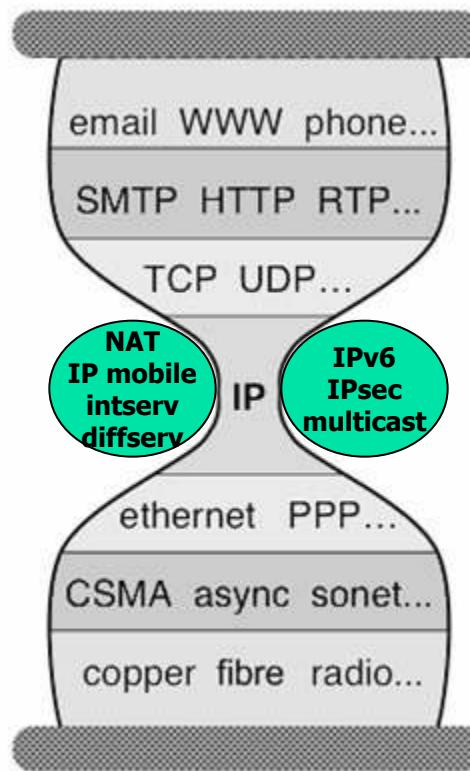
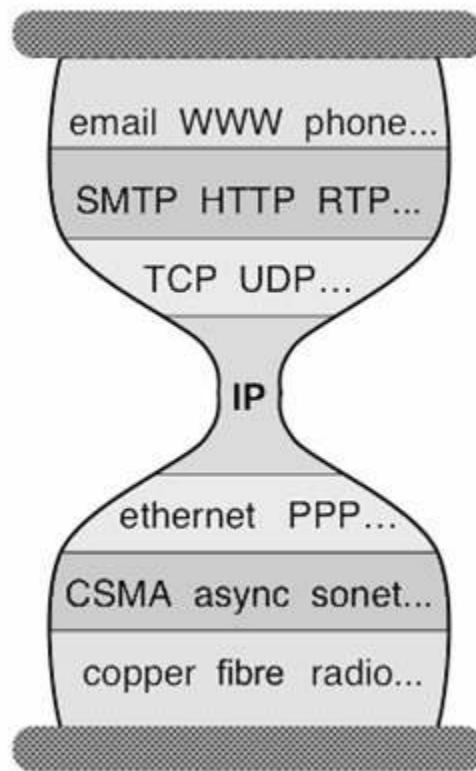
# Alcuni protocolli usati in Internet





# Lo stack di protocolli di Internet

- Lo stack di protocolli di Internet come una clessidra
  - Steve Deering. "Watching the waist of the protocol hourglass". IETF 51, London, Aug. 2001
  - <http://www.iab.org/documents/docs/hourglass-london-ietf.pdf>





# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

---

**Protocolli applicativi: FTP, SMTP, POP/IMAP**

I lucidi presentati al corso sono uno strumento didattico  
che **NON sostituisce i testi indicati nel programma del corso**

# Nota di copyright per le slide COMICS



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta

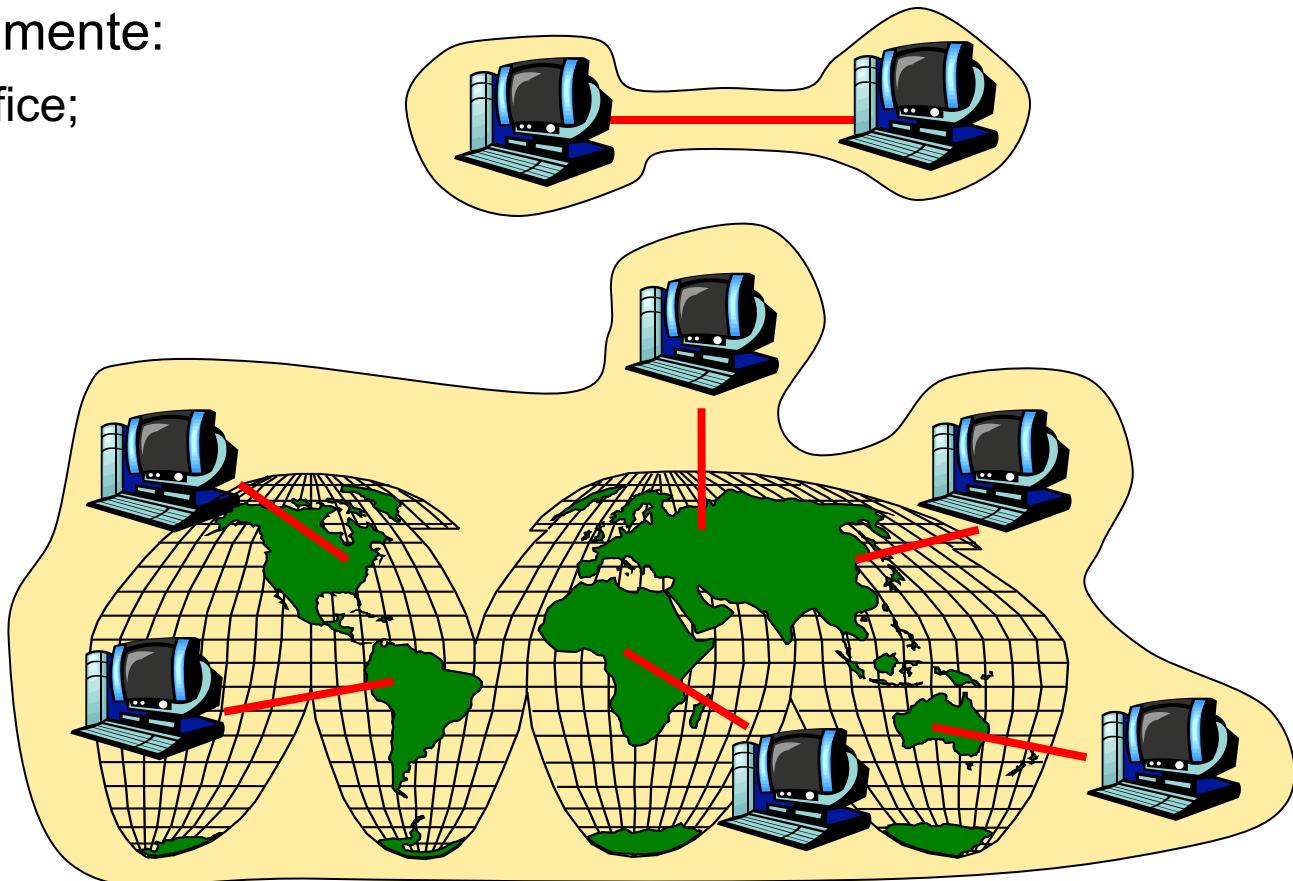


# FTP



# File Transfer Protocol (FTP)

- Internet oggi si presenta come una rete ad estensione globale che connette molti milioni di macchine sparse su tutto il globo.
- Spesso sorge l'esigenza di copiare un file da una macchina ad un'altra per poterlo utilizzare localmente:
  - un documento di Office;
  - un file eseguibile;
  - un file di testo;
  - etc...
- Ciò può accadere sia tra macchine molto distanti tra di loro che tra macchine direttamente connesse, presenti nello stesso locale.



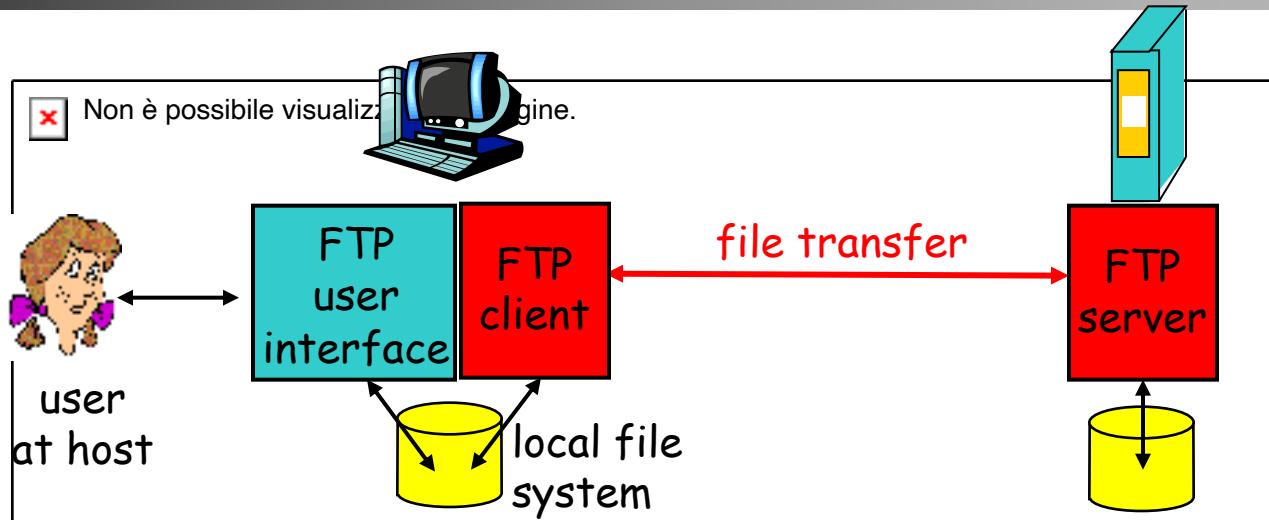


# Il protocollo FTP

- Un apposito protocollo è stato definito a questo scopo
- Si chiama File Transfer Protocol (FTP)
- Attraverso di esso è possibile trasferire uno o più file di qualsiasi tipo tra due macchine
- Il protocollo FTP è descritto in RFC959
- Lavora utilizzando due connessioni: una dati e una controllo (*out of band*)



# Come funziona FTP

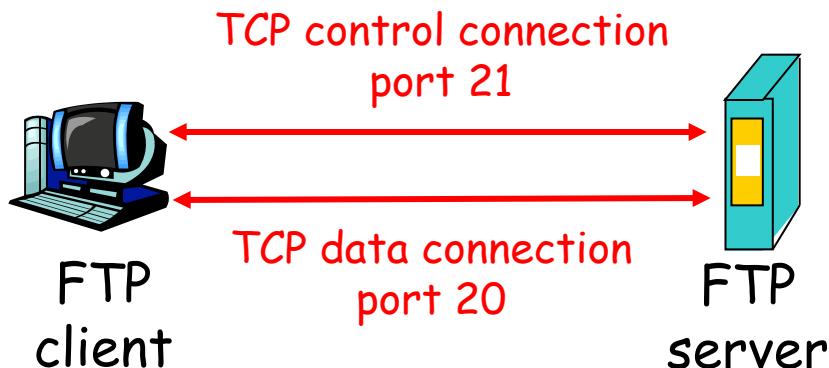


- Trasferisce file da o verso una macchina remota
- Usa il modello client/server
  - **client**: è l'entità che dà luogo al trasferimento (sia in un senso che nell'altro)
  - **server**: è l'entità remota che è in continua attesa di connessioni FTP da parte di altre entità
- ftp server: numero di porto 21



# Le connessioni di una sessione FTP

- Il client ftp contatta il server ftp al porto 21
- Vengono aperte due connessioni parallele
  - **Controllo:** scambio di comandi, messaggi di risposta tra il client e il server (*controllo “out of band” (fuori banda)*)
  - **Dati:** file che fluiscono dal client al server o viceversa
- Un server ftp mantiene uno stato, per es.
  - la directory corrente;
  - i dati dell'autenticazione.





# Scambio delle informazioni

- I comandi vengono inviati come testo ASCII sulla connessione di controllo
- Anche le risposte sono costituite da testo ASCII
- *NOTA: il testo ASCII è una sequenza di caratteri testuali stampabili*



# Esempi di comandi e codici del protocollo

## Esempi di comandi:

- **USER** *username*
- **PASS** *password*
- **LIST**  
restituisce la lista dei files presenti nella directory corrente
- **GET** *filename*  
preleva il file dalla macchina remota
- **PUT** *filename*  
invia il file alla macchina remota

## Esempi di codici di stato:

- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can't open data connection
- 452 Error writing file



# Cosa è un server FTP

- Non è possibile per un client stabilire una connessione FTP verso una qualsiasi macchina
- Il tipo di paradigma adottato (client/server) presuppone infatti che il server debba essere stato opportunamente configurato per accettare connessioni
- Solitamente, per questioni di sicurezza, le macchine non sono configurate per accettare connessioni di tipo FTP; se si tenta di stabilire una connessione verso una macchina non abilitata la sessione fallisce e nessun trasferimento risulta possibile
- I client FTP sono invece disponibili pressoché su tutti i sistemi operativi



# Client ftp da linea di comando

```
roberto@rob-ibmX40: ~
File Modifica Visualizza Terminale Ajuto
roberto@rob-ibmX40:~$ ftp ftp.unina.it
Connected to ftp.unina.it.
220 ftp.unina.it NcFTPd Server (free educational license) ready.
Name (ftp.unina.it:roberto): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230-You are user #4 of 50 simultaneous users allowed.
230-
230 Logged in anonymously.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> hash
Hash mark printing on (1024 bytes/hash mark).
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
drwxr-xr-x  2 ftpuser  ftpusers   1808 Sep  5  2011 images
-rw-r--r--  1 ftpuser  ftpusers  28319 Sep  6  2011 index.html
-rw-r--r--  1 ftpuser  ftpusers  24855 Jan 14  2011 index2.html
-rw-r--r--  1 ftpuser  ftpusers  28252 Sep  5  2011 indexprova.html
-rw-r--r--  1 ftpuser  ftpusers  28305 Sep  5  2011 indexprova.html-2
drwxr-xr-x 10 ftpuser  ftpusers    360 May  2  2011 pub
226 Listing completed.
ftp> get index2.html
local: index2.html remote: index2.html
200 PORT command successful.
150 Opening BINARY mode data connection for index2.html (24855 bytes).
#####
226 Transfer completed.
24855 bytes received in 0.38 secs (63.9 kB/s)
ftp>
```



# Esempi di client ftp grafici

**gFTP 2.0.19**

FTP Locale Remoto Segnalibri Trasferimento Registro Strumenti Aiuto

Host: Porta: Utente: Password: FTP

/home/roberto [Local] [Tutti i file]

Nome file	Dimensione	U
Screenc	1000 rc	
Video	4096 rc	
VMs	4096 rc	
.asadminpass	102 rc	
.bash_history	10942 rc	
.bash_logout	220 rc	
.bash_profile	409 rc	

Nome file Avanzamento

220 Transfer complete.  
CWD /didattica  
250 CWD command successful  
PWD  
257 "/didattica" is current directory.  
Caricamento elenco directory /didattica dal server (LC\_TIME=it\_IT.utf8)  
PASV  
227 Entering Passive Mode (192,132,34,19,202,8).  
LIST -aL

Indice di ftp://ftp.unina.it/pub/Linux/distributions/ - Mozilla Firefox

File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto GBookmarks

Corso di Reti di Calcolatori I Indice di ftp://ftp.unina.it/pub... +

ftp://ftp.unina.it/pub/Linux/distributions/ Google

Indice di ftp://ftp.unina.it/pub/Linux/distributions/

Vai alla cartella superiore

Nome	Dimensione	Ultima modifica
SuSE	16/11/2009 00:00:00	
centos	10/05/2010 00:00:00	
debian	20/07/2010 00:00:00	
debian-cd	30/06/2010 00:00:00	
distfiles	21/11/2010 00:00:00	
experimental	21/11/2010 00:00:00	
gentoo	01/03/2007 00:00:00	
grp	21/11/2010 00:00:00	
knoppix	20/07/2010 00:00:00	
releases	21/11/2010 00:00:00	
slackware	24/05/2010 00:00:00	
snapshots	21/11/2010 00:00:00	
ubuntu	10/10/2012 12:10:00	



# FTP passivo (1/2)

## Active FTP

- Il Client apre una connessione di controllo verso l'indirizzo del server sul porto 21 utilizzando un *ephemeral port*
- Il Server apre una connessione dati dal porto 20 verso il client

## Passive FTP

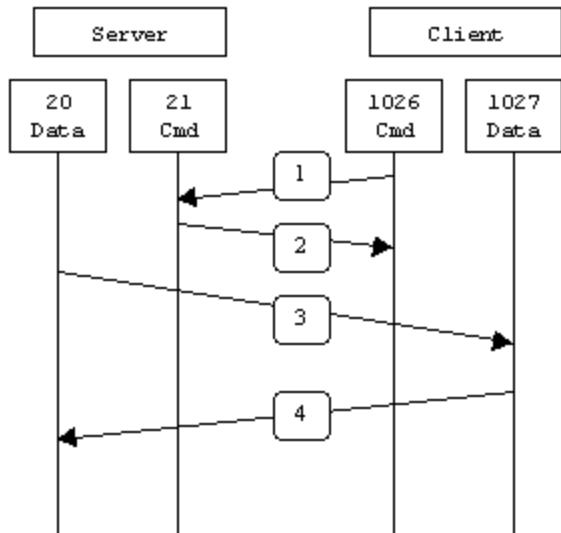
- Il Client apre una connessione di controllo verso l'indirizzo del server sul porto 21 utilizzando un *ephemeral port*
- Il Server sceglie un *ephemeral port* per la connessione dati e la comunica al Client
- Il Client apre la connessione dati sul porto indicato dal server

Nel caso passive FTP, è il client ad aprire una connessione verso il server, sia per il canale di controllo che per il canale dati

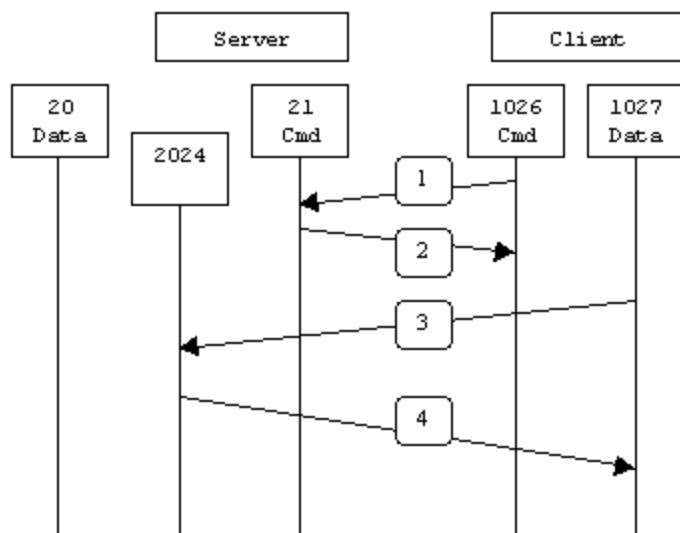


# FTP passivo (2/2)

## Active FTP



## Passive FTP





# SMTP



# Il protocollo SMTP

- Una volta che una e-mail è stata scritta attraverso l'uso di un programma su un personal computer, è necessario inviarla al destinatario
- Come è noto, il destinatario potrebbe non essere in quel momento disponibile ad accettare messaggi di posta
  - utente impegnato
  - computer spento
- La posta elettronica sfrutta degli intermediari per il trasferimento delle e-mail tra le parti, alla stregua degli uffici postali che ospitano pacchi nell'attesa che i destinatari passino a ritirarli
- Per trasferire messaggi di posta elettronica tra gli intermediari si utilizza un apposito protocollo
- Si chiama *Simple Mail Transfer Protocol*, definito in RFC821



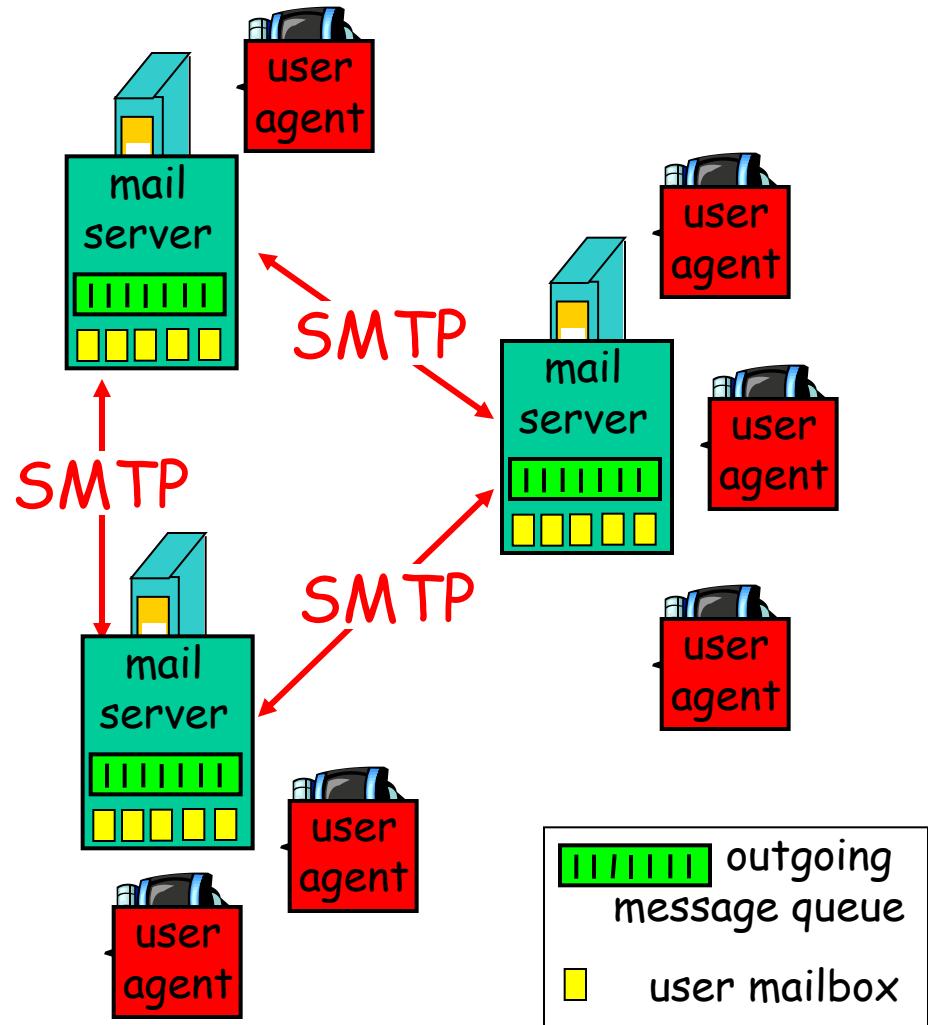
# Le entità in gioco

## Tre entità principali

- user agent
- mail server
- protocollo SMTP

### User Agent

- anche detto mail reader
- composizione, modifica, lettura di messaggi
- es.: Mail, Outlook, Mozilla Thunderbird, Evolution, Kmail...
- messaggi in uscita ed in entrata immagazzinati sul server
- il protocollo SMTP viene utilizzato anche tra user-agent e server durante l'invio di una mail

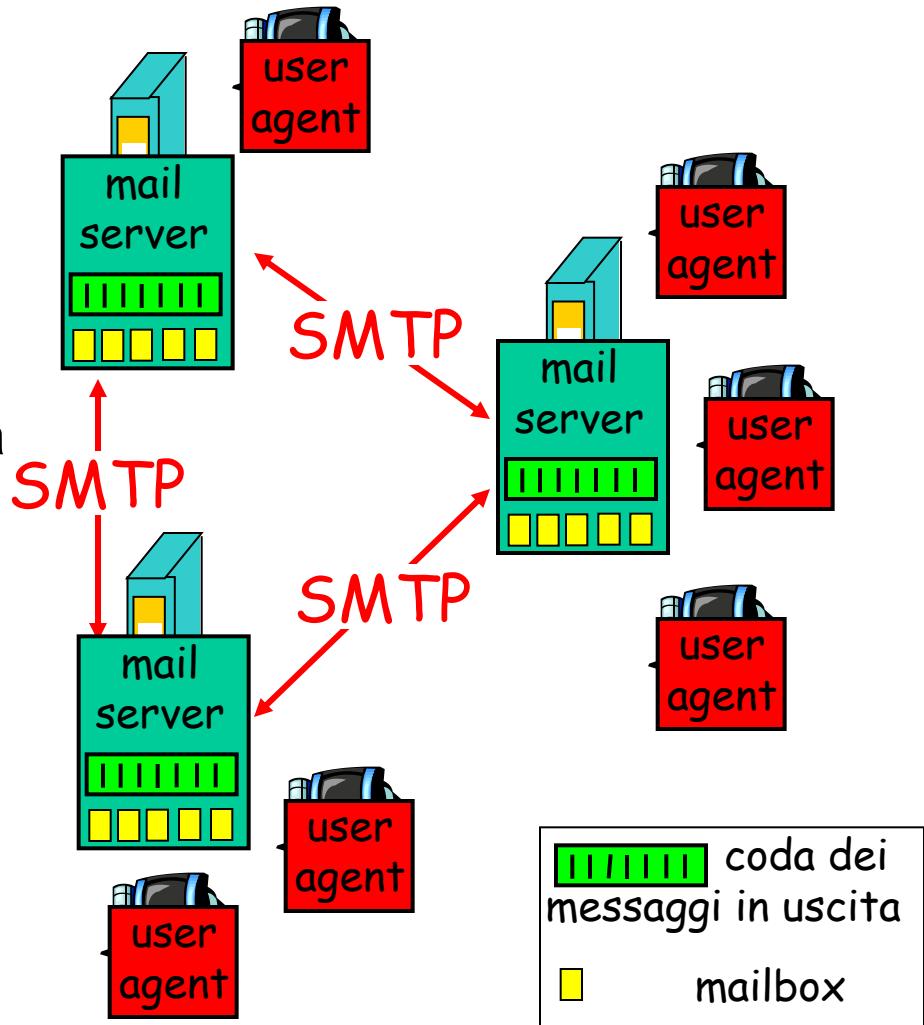




# I mail server

## Mail Server

- **mailbox** contenente messaggi in entrata (non letti) per l'utente
- **coda dei messaggi in uscita** contenente i messaggi non ancora recapitati
- **protocollo SMTP** per l'interazione tra due mail server
  - “client”: mail server mittente
  - “server”: mail server destinatario
- Un “mail server” funge in momenti diversi da client o da server a seconda del ruolo che ricopre nello scambio del messaggio





# Caratteristiche di SMTP (1)

- Usa il protocollo TCP (porto 25) per consegnare in modo affidabile messaggi dal client al server
- Trasferimento diretto dal server mittente al server destinatario
- Tre fasi durante il trasferimento via SMTP
  - handshaking (“stretta di mano”)
  - trasferimento del messaggio
  - chiusura della connessione
- Interazione comando/risposta (command/response)
  - **comandi:** testo ASCII
  - **risposta:** codice di stato e descrizione (facoltativa)
- Messaggi codificati con caratteri ASCII a 7-bit



## Caratteristiche di SMTP (2)

- Usa una connessione persistente
- Richiede che il messaggio, comprensivo del contenuto, sia codificato in caratteri ASCII a 7 bit
- Alcune combinazioni di caratteri non sono ammesse (p.es., CRLF . CRLF). Quando queste combinazioni si presentano il messaggio deve essere opportunamente codificato.
- SMTP usa CRLF . CRLF per determinare la fine di un messaggio



# Esempio di interazione client → server

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```



# SMTP in pratica: uso con telnet (1/2)

- **telnet servername 25**
- Si osservi il codice 220 di risposta dal server
- Si inseriscano i comandi HELO, MAIL FROM, RCPT TO, DATA, QUIT
- In questo modo è possibile inviare un'e-mail senza servirsi dello user agent



# SMTP in pratica: uso con telnet (2/2)

## Prompt dei comandi

```
220 smtp1.unina.it ESMTP Sendmail 8.14.0/8.14.0; Wed, 14 Oct 2009 09:10:36 +0200
HELO crepes.fr

250 smtp1.unina.it Hello host57-57-dynamic.53-79-r.retail.telecomitalia.it [79.5
3.57.57], pleased to meet you
500 5.5.1 Command unrecognized: ""
MAIL FROM: <alice@crepes.fr>
250 2.1.0 <alice@crepes.fr>... Sender ok
RCPT TO: <r canonico@unina.it>
250 2.1.5 <r canonico@unina.it>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
ciao

.
250 2.0.0 n9E7AaxZ016057 Message accepted for delivery
QUIT
221 2.0.0 smtp1.unina.it closing connection

Connessione all'host perduta.

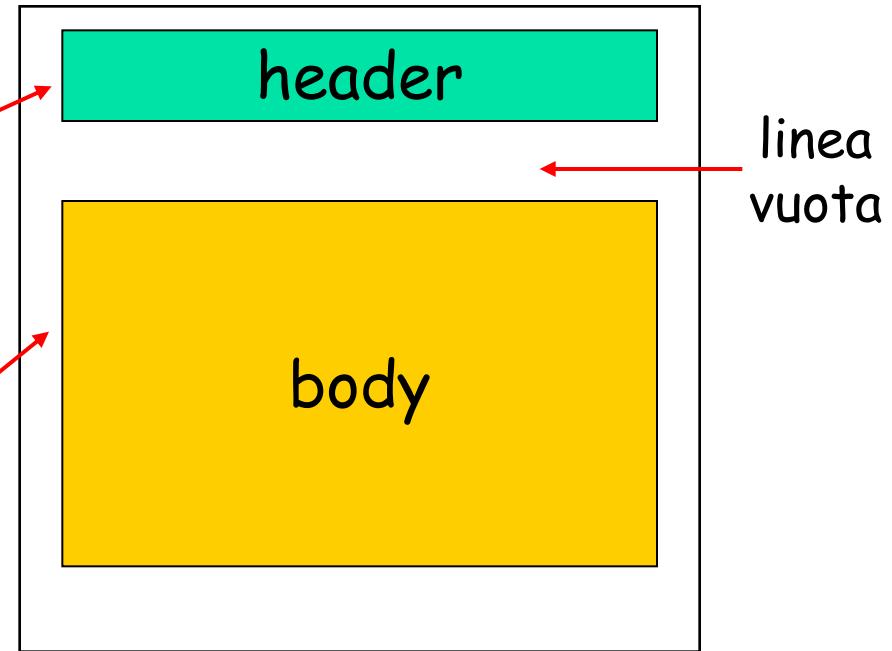
C:\Documents and Settings\User>
```



# Formato del messaggio SMTP

- Linee di intestazione (header):
  - To:
  - From:
  - Subject:
  - ...

*differenti da comandi smtp!*
- corpo (body):
  - il “messaggio” vero e proprio
    - solo caratteri ASCII





# L'estensione MIME

- MIME: Multipurpose Internet Mail Extensions, RFC 2045-2056
- righe aggiuntive nell'intestazione informano della presenza di un body MIME

versione MIME  
metodo utilizzato per codificare i dati  
tipo, sottotipo e parametri del contenuto  
dati codificati

From: alice@crepes.fr  
To: bob@hamburger.edu  
Subject: Picture of yummy crepe.  
MIME-Version: 1.0  
Content-Transfer-Encoding: base64  
Content-Type: image/jpeg

base64 encoded data .....  
.....  
.....base64 encoded data

# Messaggi SMTP: un esempio (sniffer Ethereal)



0190	74	0d	0a	43	6f	6e	74	65	6e	74	2d	54	79	70	65	3a	t.	Content-Type:
01a0	20	74	65	78	74	2f	70	6c	61	69	6e	3b	20	63	68	61		text/plain; charset=ISO-8859-1;
01b0	72	73	65	74	3d	49	53	4f	2d	38	38	35	39	2d	31	3b		format=flowed..
01c0	20	66	6f	72	6d	61	74	3d	66	6c	6f	77	65	64	0d	0a		



# Esempi di tipi MIME

Content-Type: type/subtype; parameters

## Text

- sottotipi: **plain**, **html**

## Video

- sottotipi: **mpeg**,  
**quicktime**

## Image

- sottotipi: **jpeg**, **gif**

## Application

## Audio

- sottotipi:  
**basic** (8-bit mu-law encoded), **32kadpcm**  
**(32 kbps coding)**

- altri dati che devono essere processati da specifiche applicazioni
- sottotipi: **msword**,  
**octet-stream**



# Esempio di mail “multipart”

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789
```

--98766789

```
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain
```

Dear Bob,  
Please find a picture of a crepe.  
--98766789  
Content-Transfer-Encoding: base64  
Content-Type: image/jpeg

```
base64 encoded data .....
.....
.....base64 encoded data
--98766789--
```



# POP/IMAP

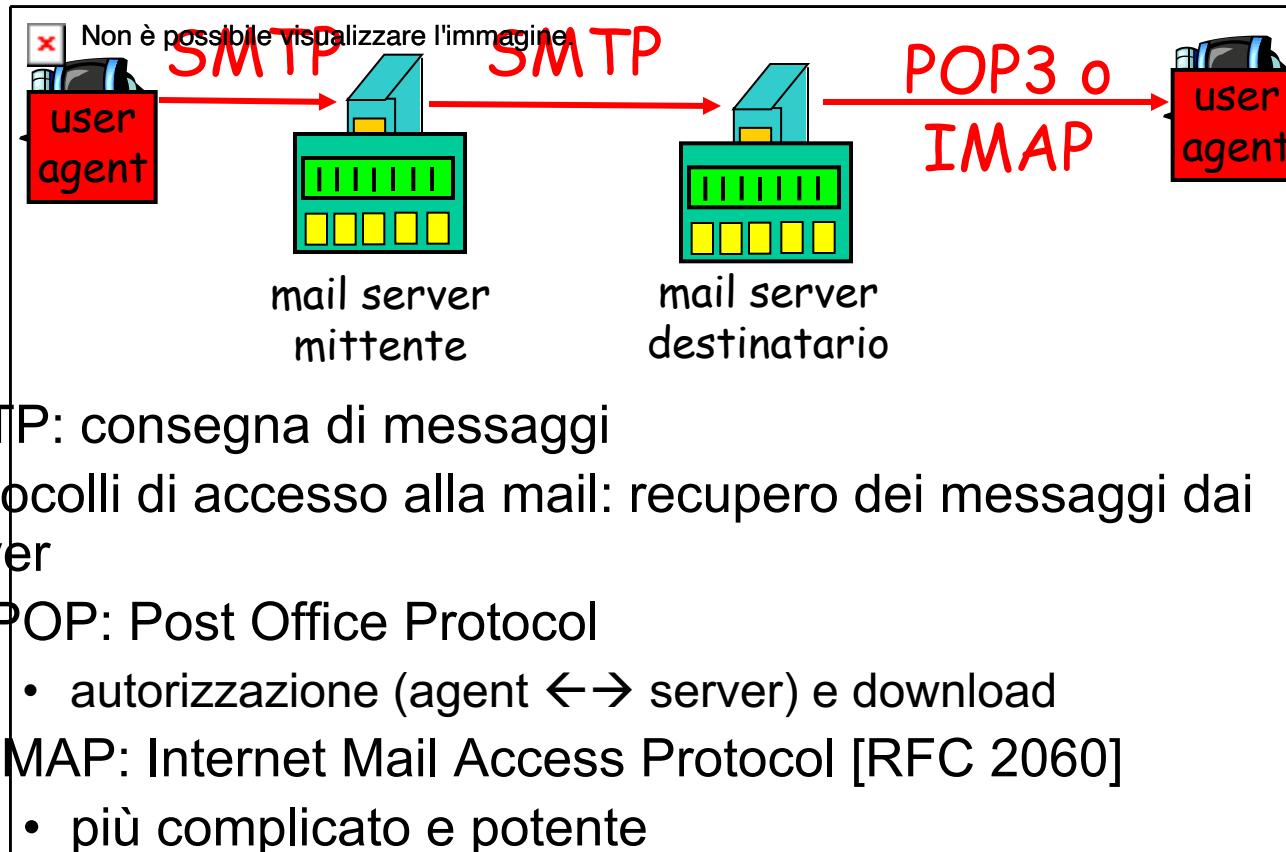


# Prelievo della posta: Post Office Protocol (POP3)

- Fino ad ora abbiamo visto come sia possibile trasferire messaggi tra i vari mail server
- Non abbiamo però ancora parlato di come un utente possa, in un momento qualsiasi, accedere alla propria casella di posta elettronica per leggere i propri messaggi
- Per questa operazione è previsto un ulteriore protocollo
- Esso è chiamato POP3 (Post Office Protocol – versione 3) ed è definito in RFC 1939
- Si tratta sempre di un protocollo client server:
  - lo user agent ancora una volta gioca il ruolo di client POP
  - il mail server gioca il ruolo di server POP



# La catena dei protocolli per la posta



- SMTP: consegna di messaggi
- Protocolli di accesso alla mail: recupero dei messaggi dai server
  - POP: Post Office Protocol
    - autorizzazione (agent  $\leftrightarrow$  server) e download
  - IMAP: Internet Mail Access Protocol [RFC 2060]
    - più complicato e potente
      - manipolazione avanzata dei messaggi sul server
- HTTP: gmail, Hotmail, Yahoo! Mail, ecc.



# Esempio di dialogo POP3

## autorizzazione

- comandi del client:
  - **user**: specifica la username
  - **pass**: specifica la password
- il server risponde
  - +OK
  - -ERR

## fase di scambio

- comandi del client:
  - **list**: visualizza la lista dei messaggi
  - **retr**: preleva il messaggio per numero
  - **dele**: elimina il messaggio dal server
  - **quit**: chiude la sessione

```
S: +OK POP3 server ready
C: user alice
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```



# POP3 in pratica (1/2): telnet cds.unina.it 110

Telnet cds.unina.it

```
+OK cds.unina.it Cyrus POP3 v2.1.10 server ready
user spromano
+OK Name is a valid mailbox
pass [REDACTED]
+OK Maildrop locked and ready
list_
```

Qui c'era la password!

...



# POP3 in pratica (2/2): telnet cds.unina.it 110

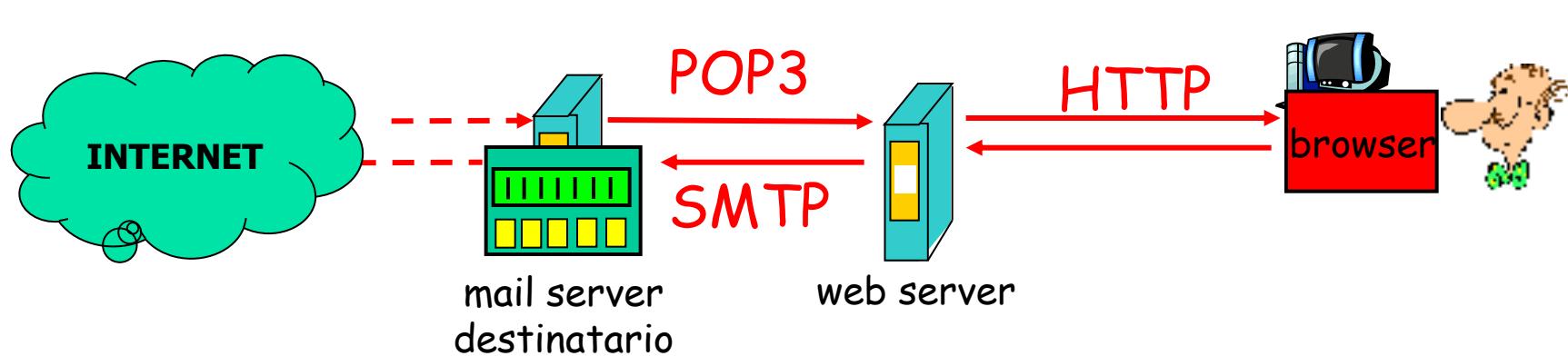
```
6292 1867
6293 2683
6294 913
.
retr 6294
+OK Message follows
Return-Path: <spromano@unina.it>
Received: from cds.unina.it ([unix socket])
    by cds.unina.it (Cyrus v2.1.10) with LMTP; Mon, 15 Mar 2004 13:13:33 +0100
X-Sieve: CMU Sieve 2.2
Received: from mail.unina.it (mail.unina.it [192.132.34.73])
    by cds.unina.it (8.12.11/8.12.9) with ESMTP id i2FCDX6G016069
    for <spromano@unina.it>; Mon, 15 Mar 2004 13:13:33 +0100 (CET)
Received: (from root@localhost)
    by mail.unina.it (8.12.11/8.12.11) id i2FCBCfe021235
    for spromano@unina.it; Mon, 15 Mar 2004 13:11:12 +0100
Received: from grid.unina.it ([143.225.229.172])
    by mail.unina.it (8.12.11/8.12.11) with SMTP id i2FC5bZc018823
    for spromano@unina.it; Mon, 15 Mar 2004 13:10:24 +0100
Date: Mon, 15 Mar 2004 13:10:24 +0100
From: spromano@unina.it
Message-Id: <200403151210.i2FC5bZc018823@mail.unina.it>
X-scanner: scanned by Inflex 1.0.12.7

Ciao giovane, come stai?
Fammi sapere asap...
.
```



# L'accesso alla posta via WEB

- Molti siti web forniscono accesso alle proprie caselle di posta (gmail, hotmail, Yahoo!, ecc.)
- In questo caso non serve avere uno user agent installato e correttamente configurato per ricevere ed inviare posta.
- È sufficiente disporre di un qualsiasi browser





# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

### **Protocolli applicativi: DNS**

**I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso**

# Nota di copyright per le slide COMICS



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta



# Domain Name System (DNS)

- Tutti noi siamo oggi abituati a raggiungere un servizio (e quindi il calcolatore che lo offre) utilizzando nomi simbolici di facile memorizzazione
  - www.google.com
  - www.rai.it
  - pippo@unina.it
- Questi nomi non sono immediatamente adatti ad essere compresi dai dispositivi che costituiscono la rete Internet
- Un nome di questo tipo, infatti, non dà informazioni esatte sulla dislocazione sul territorio della macchina che si desidera contattare
- I router, di conseguenza, non saprebbero come instradare i dati in maniera tale da raggiungere la destinazione



# Nomi simbolici vs Indirizzi IP

- La rete Internet è stata progettata invece per lavorare con indirizzi di diversa natura. Per es.
  - 143.225.229.3
  - 217.9.64.225
- Questi indirizzi, detti indirizzi IP, nella versione IPv4 sono rappresentati con 4 numeri che vanno da 0 a 255 separati da punti
- Ogni dispositivo nella rete Internet ha un tale indirizzo; esso permette *l'identificazione* univoca a livello globale e la *localizzazione* (topologicamente) nella rete
- A differenza dei nomi simbolici, essendo gli indirizzi IP di lunghezza fissa, sono più facilmente gestibili dalle macchine



# Il servizio DNS

- Non volendo rinunciare alla comodità di lavorare con nomi simbolici, è stato necessario progettare un servizio di risoluzione dei nomi simbolici in indirizzi IP
- Tale servizio associa ad un nome simbolico univoco ([www.grid.unina.it](http://www.grid.unina.it)) un indirizzo IP (143.225.229.3) permettendo così di raggiungere la macchina
- Questo servizio si chiama Domain Name System (DNS) ed è definito in RFC1034 e RFC1035
- Esso si basa sullo scambio di messaggi UDP sul porto 53 (anche TCP può essere usato, a richiesta)



# Altre funzionalità offerte

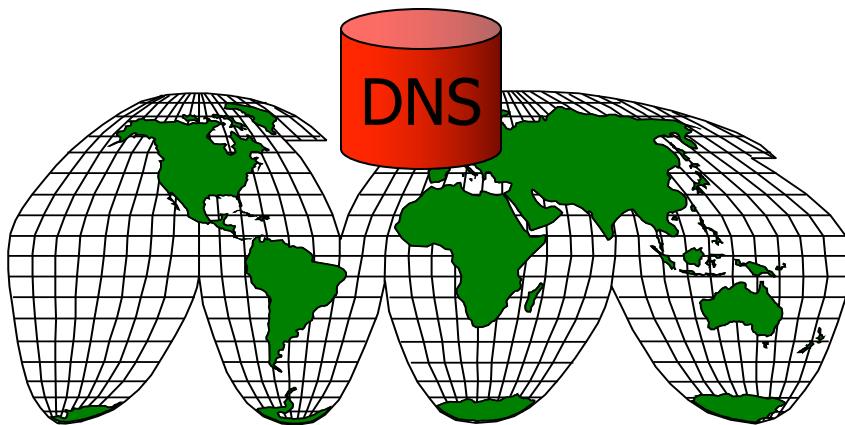
- Alias degli hostname
  - ad una macchina con un nome complicato può essere associato un “soprannome” più piccolo e semplice da ricordare  
P.es.: rcsn1.roma.rai.it → www.rai.it
- Alias dei server di posta
  - permette di associare un server di posta al nome di un dominio per facilitare la memorizzazione dell'indirizzo di posta  
P. es.: **pippo@unina.it** identifica l'utente **pippo** sulla macchina **mailsrv1.cds.unina.it**. L'associazione @unina.it → mailsrv1.cds.unina.it è a carico del servizio DNS
- Distribuzione del carico
  - quando un server gestisce un carico troppo elevato si suole replicare il suo contenuto su molte macchine differenti. Il servizio DNS distribuisce il carico tra le macchine rilasciando ciclicamente indirizzi appartenenti all'intero pool, senza che gli utenti si accorgano di nulla

www.domain.com  
–1.2.3.4  
–1.2.3.15  
–1.2.4.200  
–1.2.15.121  
–1.5.34.12



# DNS centralizzato?

- Si potrebbe pensare di risolvere il problema piazzando in un unico punto della terra una macchina che realizzi la risoluzione di tutti i nomi



- Questa soluzione, sebbene teoricamente realizzabile, ha così tanti svantaggi da risultare impraticabile
  - Single Point of Failure
  - Volume di traffico
  - Database distante
  - Manutenzione

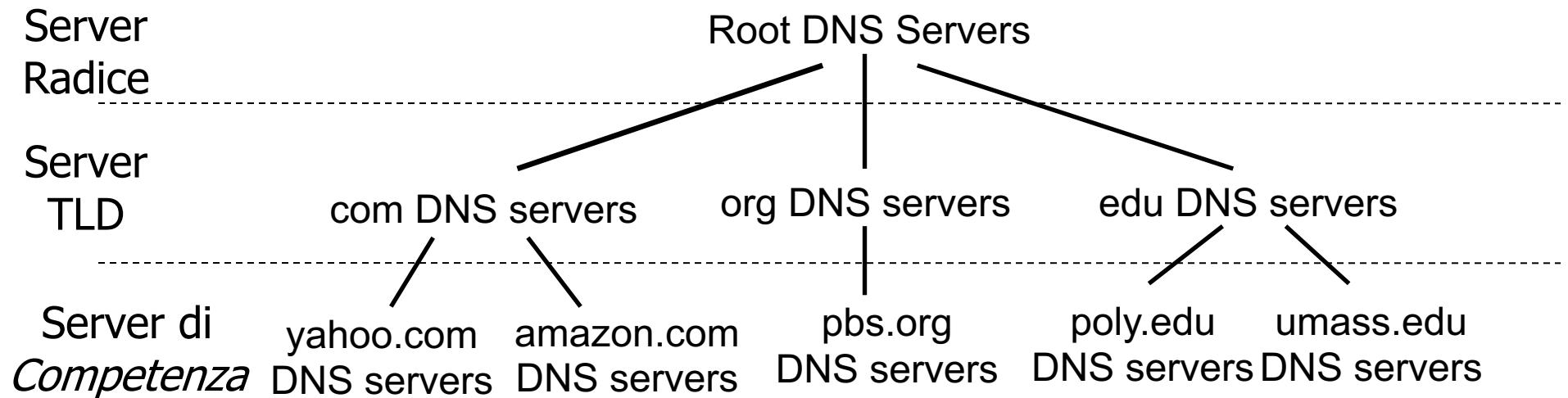


# DNS distribuito!

- Quello che si fa è **distribuire** le **informazioni** sul territorio
- Si **distribuisce** anche la **responsabilità** di raccogliere, gestire, aggiornare e divulgare le informazioni che lo riguardano
- In particolare l'approccio è di tipo gerarchico
  - gli elementi più **alti** nella gerarchia contengono **molte** informazioni **non dettagliate**
  - gli elementi più **bassi** nella gerarchia contengono **poche** informazioni **dettagliate**
- Attraverso un colloquio concertato tra le entità (di cui gli utenti non hanno percezione) si riesce a fornire il servizio di risoluzione



# DNS: un database gerarchico e distribuito



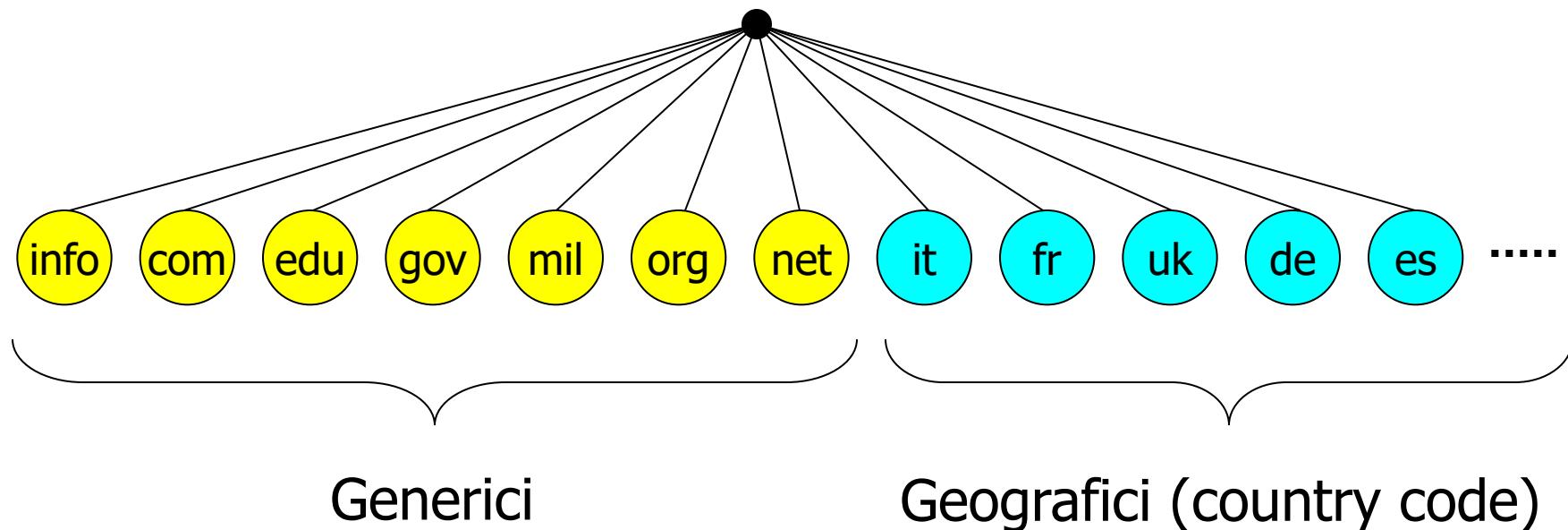
## Un Client richiede l'IP di [www.amazon.com](http://www.amazon.com) (1<sup>st</sup> approx)

- Il client dapprima contatta uno dei root server per avere la lista degli indirizzi IP dei TLD per il dominio com
- Il client contatta uno dei TLD server che gli restituisce l'indirizzo IP del server autorizzato per amazon.com
- Infine il client contatta il server autorizzato per amazon.com che gli restituisce l'indirizzo IP di www.amazon.com



# I “top-level domain” server (TLD)

- Questi server si occupano dei domini di alto livello (generici e geografici). Domini “storici”



Domini TLD: >1500 ad oggi, divisi in molte categorie



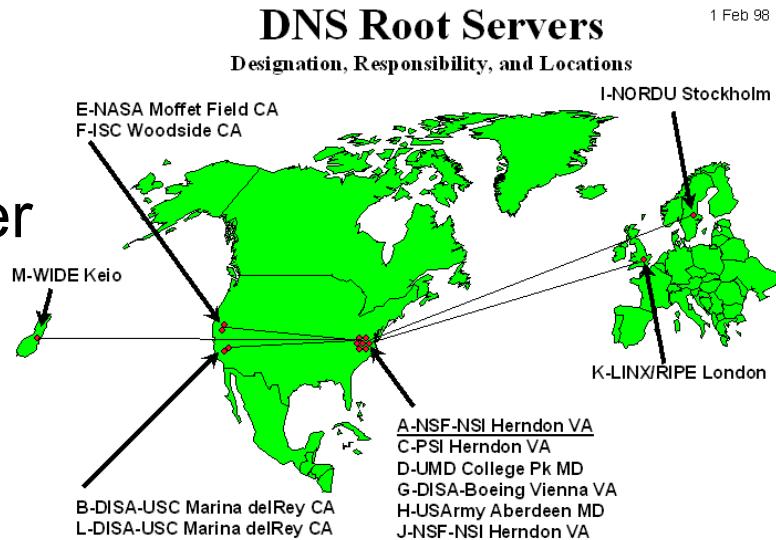
# I nuovi “top-level domain” (2005 – current)

- Nuovi TLD possono essere richiesti (e venduti); aggiornati al 2016:
  - Geografici (country-code): 303
  - Generici: 971
  - Generici-limitati: 3 (.biz, .name, .pro)
  - test: 11 (.испытание, .測試, ...)
  - sponsorizzati (ristretti a comunità, gestiti da ente): 15 (.xxx, .mil, .mobi, .coop, .cat, ...)
  - infrastruttura: 1 (.arpa)
  - ...



# Tipologie di server DNS (Root)

- Root Name Server
  - 13 root server logici in Internet (etichettati da A ad M) i cui indirizzi IP sono ben noti alla comunità
  - In realtà si tratta di 376 diversi server fisici (vedi <http://www.root-servers.org/>)
  - Ad essi si riferiscono i Local Name Server che non possono soddisfare immediatamente una richiesta di risoluzione
  - Il Local Name Server si comporta come client DNS ed invia una richiesta di risoluzione al Root Name Server





# Tipologie di server DNS (Authoritative)

- Authoritative Name Server (Assoluto)
  - È un server dei nomi capace di risolvere tutti i nomi all'interno di un determinato dominio/organizzazione
    - Es.: un server dei nomi assoluto per il dominio unina.it deve essere capace di risolvere tutti i nomi del tipo xyz.unina.it
  - Ad essi si riferiscono i Name Server TLD quando, interpellati dai Local Name Server, devono risolvere un indirizzo
  - Può essere mantenuto dall'organizzazione o da un provider



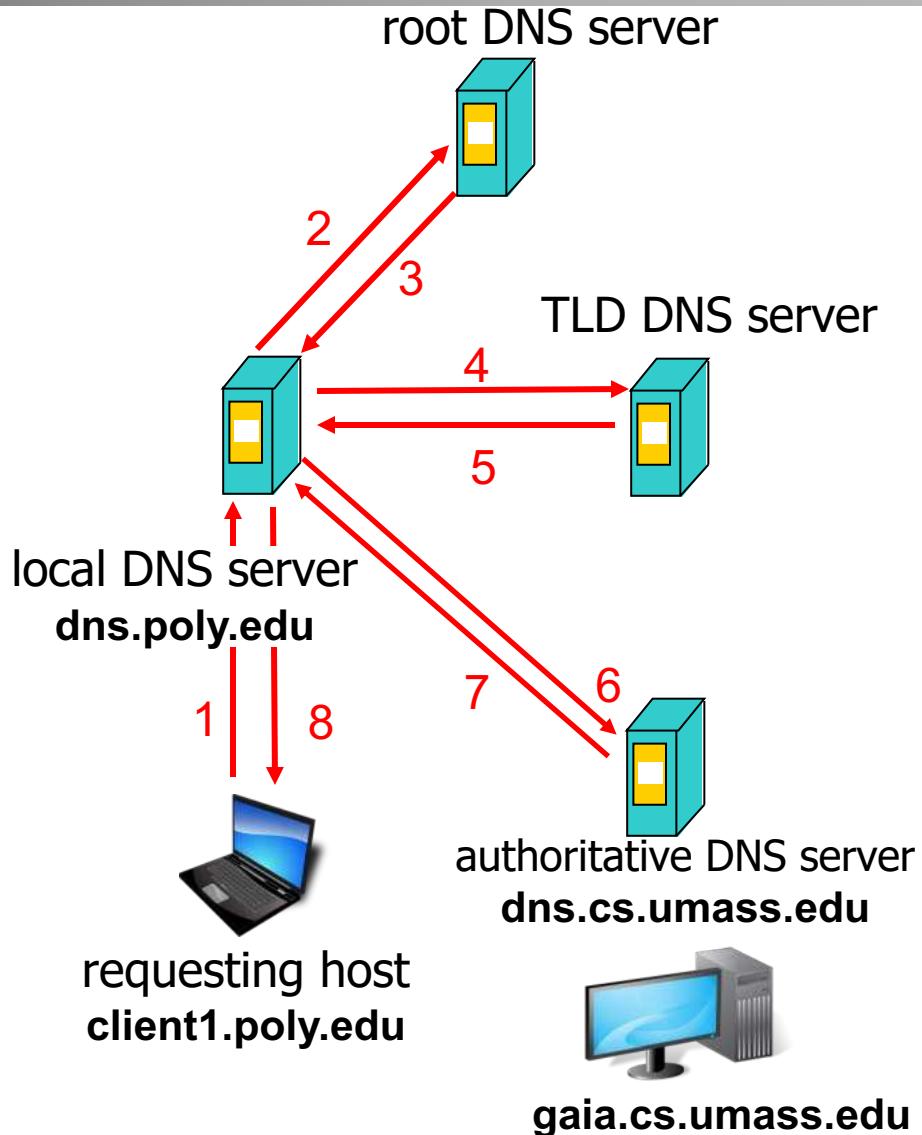
# Tipologie di server DNS (Local)

- Local Name Server (Locale)
  - Non appartengono strettamente alla **gerarchia** di server
  - Ciascun ente (università, società, etc...) ne installa uno nel **proprio dominio**
  - Tutti gli host nel dominio richiedono a questo server il servizio di risoluzione
  - Quando un host effettua una richiesta DNS, la query viene inviata al server **DNS locale che opera da proxy** e, tipicamente, invia la query attraverso la gerarchia di server
  - Ciascun host deve essere **configurato con l'indirizzo del DNS server locale** per il dominio. Questa configurazione spesso avviene manualmente, ma in certi casi può avvenire anche in maniera automatica
  - L'uso di un server DNS locale consente ai singoli host di fare una sola query DNS verso di essi: sarà poi il local DNS server a fare la sequenza di interrogazioni descritta in precedenza



# Un semplice esempio di risoluzione

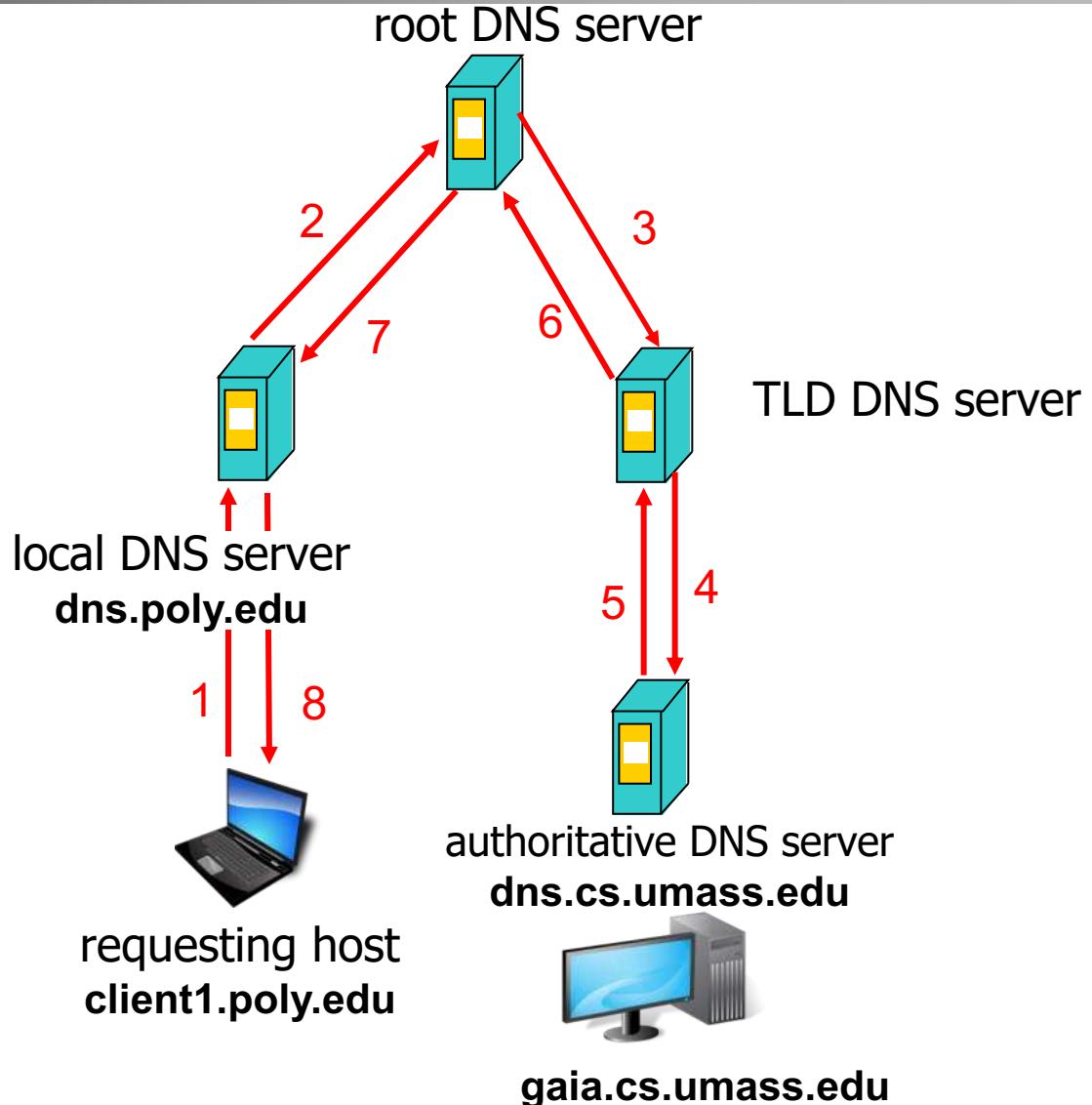
- Un host di `cis.poly.edu` richiede l'indirizzo IP di ***gaia.cs.umass.edu***
- **Query Iterative:**
  - Il server contattato risponde con il nome del server da contattare
  - La logica: non conosco questo nome, ma conosco il nome di qualcuno a cui poter chiedere





# Un semplice esempio di risoluzione

- **Query Ricorsive:**
  - Sposta il carico della risoluzione dei nomi sul server contattato, delegando al NS contattato la responsabilità di risolvere l'indirizzo
  - Troppo carico
  - Root server sempre iterativi





# Un esempio a più livelli

- Il TLD potrebbe non contattare necessariamente l'Authoritative Name Server finale, ma un Authoritative Name Server intermediario
- Sarà il server intermediario a fornire il nome del server di competenza
- In questi casi il numero di messaggi DNS aumenta



## Il caching dei nomi

- Per esigenze di efficienza un server DNS memorizza localmente un certo numero di corrispondenze
- Per evitare che informazioni non aggiornate restino nella rete, dopo un certo tempo (circa un giorno), le associazioni vengono eliminate dalla cache
- Ad es. un server locale può memorizzare associazioni IP/nomi di non sua competenza e/o gli indirizzi dei server TLD in modo da aggirare i server root



# Altri livelli di caching

- Applicazione (es. browser web)
  - Firefox
- Sistema operativo
  - windows: dnscache
  - linux: dnsmasq



# Cosa memorizza un DNS

## Resource records (RR)

**Formato RR:** (**Nome, Valore, Tipo, TTL**)

- **TTL**: tempo di vita residuo di un record scaduto il quale viene eliminato dalla cache
- Il significato di **Nome** e **Valore** dipende da **Tipo**
  - **Tipo=A**
    - nome=hostname
    - valore: indirizzo IP
  - **Tipo=NS**
    - nome=dominio (p.es.: unina.it)
    - valore=ind. IP dell'Authoritative NS
  - **Type=CNAME**
    - **nome**=alias per il nome canonico (reale)
    - **valore**=nome canonico
  - **Type=MX**
    - **nome**=dominio di posta (p.es. libero.it)
    - **valore**=nome dell'host mailserver associato a **nome**



# Esempi di RR

- Type A
    - relay.bar.foo.com, 145.37.93.126, A
  - Type NS
    - foo.com, dns.foo.com, NS
  - Type CNAME
    - foo.com, relay.bar.foo.com, CNAME
  - Type MX
    - foo.com, mail.bar.foo.com, MX
- \* Negli esempi non è mostrato il valore del campo TTL

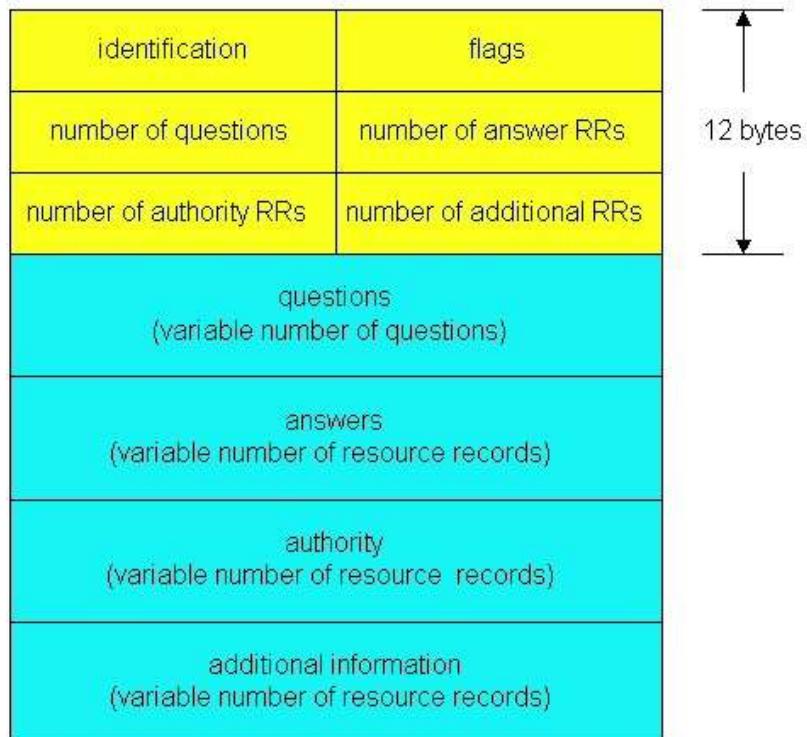


# Il formato dei messaggi (1)

Protocollo DNS : *richieste e risposte*, entrambe con lo stesso formato di messaggio

## Header del messaggio

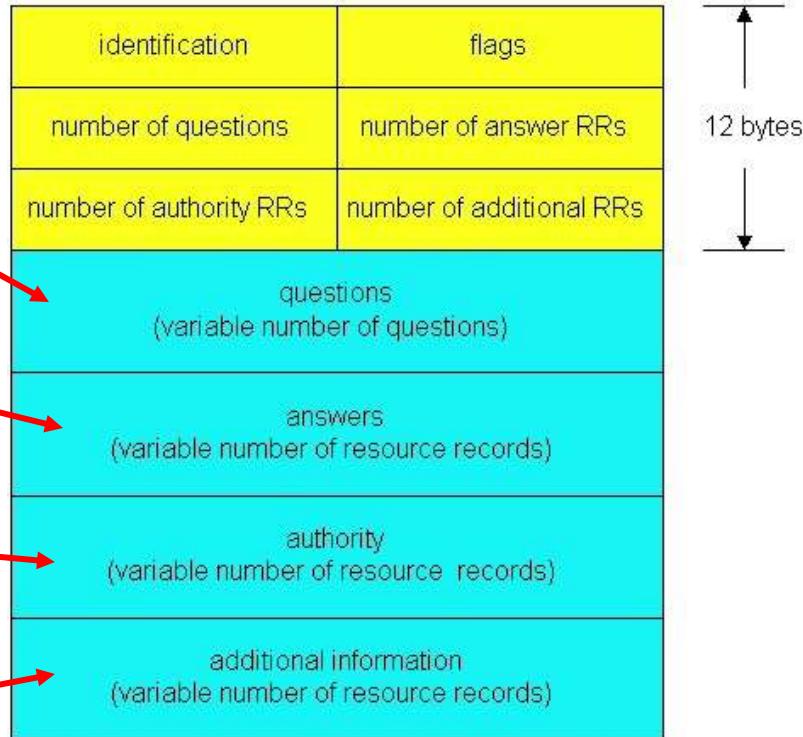
- **identification:** diverso numero di 16bit per ogni richiesta. Le risposte usano lo stesso identificativo
- **flags:**
  - risposta a richiesta
  - ricorsione desiderata
  - ricorsione disponibile
  - risposta authoritative





## Il formato dei messaggi (2)

- Nome e tipo per una richiesta
- RR in risposta ad una richiesta
- records per server authoritative
- informazioni addizionali utili che possono essere utilizzate





# Note di sicurezza

- Il protocollo DNS non offre autenticazione, né integrità, né riservatezza
  - chiunque può **leggere** le richieste (es: [www.pornocamini.it](http://www.pornocamini.it))
  - chiunque può **rispondere** al posto del vero DNS resolver
  - chiunque può **alterare** in volo le richieste/risposte
- mancanza di riservatezza:
  - sorveglianza, blocchi
- mancanza di autenticazione ed integrità:
  - censura (ed altri attacchi informatici)
- Soluzioni
  - per autenticazione ed integrità: **DNSSec** (nuovi RR)  
<https://tools.ietf.org/html/rfc4034>
  - per la riservatezza: al 2016 in discussione **DNS-PRIVe WG**  
<https://datatracker.ietf.org/doc/charter-ietf-dprivate/>



# Il servizio BIND

- BIND (Berkeley Internet Name Domain) è una implementazione dei protocolli Domain Name System (DNS)
- È liberamente re-distribuibile
- È costituito dai seguenti componenti
  - Un server DNS (named)
  - Una libreria per la risoluzione dei nomi di dominio
  - Strumenti di diagnostica
- Questa implementazione è di gran lunga la più utilizzata su Internet su sistemi Unix-like



# Configurazione di BIND: un esempio di file di zona

```
$TTL 3600
@       IN SOA grid.grid.unina.it. root.grid.grid.unina.it. (
                  2004020901      ; Serial
                  10800          ; Refresh
                  3600           ; Retry
                  604800         ; Expire
                  86400 )        ; Minimum TTL

; Machine Name
localhost     A      127.0.0.1

vesuvio        A      143.225.229.1
grid           A      143.225.229.3
honolulu       A      143.225.229.111
comicserver    A      143.225.229.112
...
; Aliases
www            CNAME  grid
ftp             CNAME  grid
news            CNAME  grid
tesisti         CNAME  vesuvio
www.tesisti    CNAME  vesuvio

; MX Record
MX              10     grid.grid.unina.it.
```

\* SOA: Start of Authority



# Significato di alcuni parametri

- **Serial**: numero seriale progressivo utilizzato per rilevare aggiornamenti del file. Di solito usa il formato: aaaammmgxx
- **Refresh**: intervallo in secondi tra due successivi prelievi del file di zone da parte di un DNS server
- **Retry**: intervallo in secondi tra tentativi successivi di recuperare una zona in caso di fallimento
- **Expire**: tempo in secondi che deve trascorrere per ritenere scadute le informazioni di una zona che non si riesce ad aggiornare
- **Minimum TTL**: tempo di durata di default delle singole entry del file di zona



# Un esempio: configurazione del Reverse DNS

\$TTL 3600

```
@ IN SOA grid.grid.unina.it. root.grid.grid.unina.it. (
    2004020901      ; Serial
    10800           ; Refresh
    3600            ; Retry
    604800          ; Expire
    86400 )         ; Minimum TTL
; DNS Servers
NS      grid.grid.unina.it.

; Machine Name
1       PTR     vesuvio.grid.unina.it.
3       PTR     grid.grid.unina.it.
111     PTR     honolulu.grid.unina.it.
112     PTR     comicserver.grid.unina.it.
```



## II file named.root

```
.          3600000  IN  NS      A.ROOT-SERVERS.NET.  
A.ROOT-SERVERS.NET.    3600000          A      198.41.0.4  
;  
; formerly NS1.ISI.EDU  
;  
.          3600000          NS      B.ROOT-SERVERS.NET.  
B.ROOT-SERVERS.NET.    3600000          A      128.9.0.107  
;  
; formerly C.PSI.NET  
  
...  
.          3600000          NS      L.ROOT-SERVERS.NET.  
L.ROOT-SERVERS.NET.    3600000          A      198.32.64.12  
;  
; housed in Japan, operated by WIDE  
;  
.          3600000          NS      M.ROOT-SERVERS.NET.  
M.ROOT-SERVERS.NET.    3600000          A      202.12.27.33  
; End of File
```



# Un esempio di uso di nslookup

```
> nslookup  
> www.cisco.com  
Server:      143.225.229.3  
Address:     143.225.229.3#53  
Non-authoritative answer:  
Name:   www.cisco.com  
Address: 198.133.219.25  
> set ty=ns  
> cisco.com  
Server:      143.225.229.3  
Address:     143.225.229.3#53  
Non-authoritative answer:  
cisco.com    nameserver = ns1.cisco.com.  
cisco.com    nameserver = ns2.cisco.com.  
Authoritative answers can be found from:  
ns1.cisco.com  internet address = 128.107.241.185  
ns2.cisco.com  internet address = 64.102.255.44  
> server ns1.cisco.com  
Default server: ns1.cisco.com  
Address: 128.107.241.185#53  
> set ty=a  
> www.cisco.com  
Server:      ns1.cisco.com  
Address:     128.107.241.185#53  
Name:   www.cisco.com  
Address: 198.133.219.25
```

Chiedo di risolvere l'host

Indirizzo del local DNS che serve la richiesta

Ecco la risposta, che non proviene da un server

Imposto nslookup per l'invio di query di tipo NS: restituirà i name server authoritative di un dominio specificato

Chiedo il Name Server authoritative per il dominio cisco.com

Eccoli: sono due

E questi sono i loro indirizzi IP

Imposto nslookup per l'invio delle successive query al NS ns1.cisco.com

Reimposto nslookup per l'invio di query di tipo A (risoluzione di nomi di host) e richiedo la risoluzione del nome di host www.cisco.com

Questa volta la risposta è autoritative.  
La entry non-authoritative memorizzata in cache era valida.



# Un esempio di uso di dig (1/2)

```
> dig www.cisco.com

; <>> DiG 9.3.1 <>> www.cisco.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39786
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
www.cisco.com.          IN      A

;; ANSWER SECTION:
www.cisco.com.      85619    IN      A      198.133.219.25

;; AUTHORITY SECTION:
cisco.com.           81328    IN      NS      ns2.cisco.com.
cisco.com.           81328    IN      NS      ns1.cisco.com.

;; ADDITIONAL SECTION:
ns2.cisco.com.       86175    IN      A      64.102.255.44
ns1.cisco.com.       81857    IN      A      128.107.241.185

;; Query time: 1 msec
;; SERVER: 143.225.229.3#53(143.225.229.3)
;; WHEN: Tue Oct 10 19:54:08 2006
;; MSG SIZE  rcvd: 115
```



# Un esempio di uso di dig (2/2)

```
> dig www.cisco.com @ns1.cisco.com

; <>> DiG 9.3.1 <>> www.cisco.com @ns1.cisco.com
; (1 server found)
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7291
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.cisco.com.           IN      A

;; ANSWER SECTION:
www.cisco.com.        86400    IN      A      198.133.219.25

;; AUTHORITY SECTION:
cisco.com.            86400    IN      NS      ns1.cisco.com.
cisco.com.            86400    IN      NS      ns2.cisco.com.

;; ADDITIONAL SECTION:
ns1.cisco.com.        86400    IN      A      128.107.241.185
ns2.cisco.com.        86400    IN      A      64.102.255.44

;; Query time: 224 msec
;; SERVER: 128.107.241.185#53(128.107.241.185)
;; WHEN: Tue Oct 10 19:55:10 2006
;; MSG SIZE  rcvd: 115
```



# MX server con nslookup

```
> nslookup  
> set ty=mx  
> unina.it  
Server:      143.225.229.3  
Address:     143.225.229.3#53
```

Imposto nslookup per l'invio di query di tipo MX (Mail eXchanger): server SMTP di dominio

Chiedo i mail server del dominio "@unina.it"

Non-authoritative answer:

```
unina.it      mail exchanger = 10 pmx1.unina.it.  
unina.it      mail exchanger = 10 pmx2.unina.it.
```

Authoritative answers can be found from:

```
unina.it      nameserver = dscna1.unina.it.  
unina.it      nameserver = dscna2.unina.it.  
pmx1.unina.it internet address = 192.132.34.28  
pmx2.unina.it internet address = 192.132.34.29  
dscna1.unina.it internet address = 192.133.28.1  
dscna2.unina.it internet address = 192.133.28.7
```



# MX server con dig

```
> dig unina.it MX

; <>> DiG 9.3.1 <>> unina.it MX
;; global options:  printcmd
;; Got answer:
;; ->>>HEADER<<- opcode: QUERY, status: NOERROR, id: 433
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 4

;; QUESTION SECTION:
;unina.it.          IN      MX

;; ANSWER SECTION:
unina.it.        26684    IN      MX      10 pmx1.unina.it.
unina.it.        26684    IN      MX      10 pmx2.unina.it.

;; AUTHORITY SECTION:
unina.it.        55400    IN      NS      dscna1.unina.it.
unina.it.        55400    IN      NS      dscna2.unina.it.

;; ADDITIONAL SECTION:
pmx1.unina.it.   26684    IN      A       192.132.34.28
pmx2.unina.it.   8220     IN      A       192.132.34.29
dscna1.unina.it. 55400    IN      A       192.133.28.1
dscna2.unina.it. 55400    IN      A       192.133.28.7

;; Query time: 1 msec
;; SERVER: 143.225.229.3#53(143.225.229.3)
;; WHEN: Tue Oct 10 20:00:10 2006
;; MSG SIZE  rcvd: 174
```



# Nslookup: esempio (1)

```
C:\Documents and Settings\User>nslookup  
*** Impossibile trovare nome server per l'indirizzo 85.37.17.11: Non-existent domain  
Server predefinito: host69-28-static.38-85-b.business.telecomitalia.it  
Address: 85.38.28.69
```

```
> www.cisco.com  
Server: host69-28-static.38-85-b.business.telecomitalia.it  
Address: 85.38.28.69
```

Risposta da un server non di fiducia:

```
Nome: e144.cd.akamaiedge.net  
Address: 88.221.28.170  
Aliases: www.cisco.com, www.cisco.com.akadns.net  
          geoprod.cisco.com.akadns.net, www.cisco.com.edgekey.net  
          www.cisco.com.edgekey.net.globalredir.akadns.net
```

```
> set ty=ns  
> cisco.com  
Server: host69-28-static.38-85-b.business.telecomitalia.it  
Address: 85.38.28.69
```

Risposta da un server non di fiducia:

```
cisco.com      nameserver = ns2.cisco.com  
cisco.com      nameserver = ns1.cisco.com
```



# Nslookup: esempio (2)

```
> server ns1.cisco.com  
Server predefinito: ns1.cisco.com  
Address: 128.107.241.185
```

```
> set ty=a  
> www.cisco.com  
Server: ns1.cisco.com  
Address: 128.107.241.185  
  
Nome: origin-www.cisco.com  
Address: 198.133.219.25  
Aliases: www.cisco.com, www.cisco.com.akadns.net
```

L'indirizzo IP  
non corrisponde con  
quello dato prima:  
88.221.28.170



# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

---

**Il livello rete in Internet  
Il protocollo IP**

I lucidi presentati al corso sono uno strumento didattico  
che **NON** sostituisce i testi indicati nel programma del corso

# Nota di copyright per le slide COMICS



## Nota di Copyright

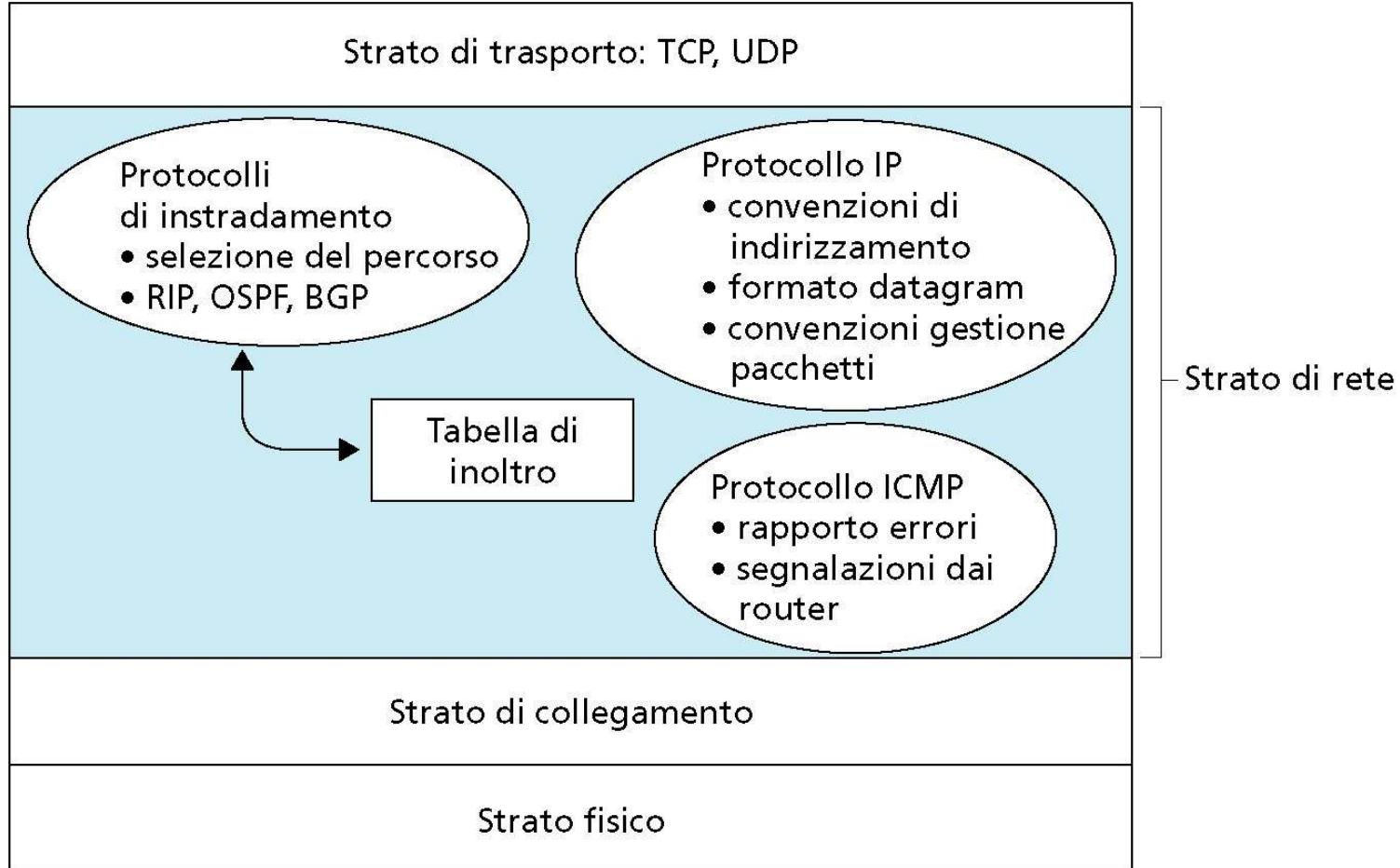
Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta



# Lo strato di rete di Internet





# IP (Internet Protocol)

- IP è un protocollo di livello rete usato per lo scambio di dati tra reti di calcolatori
- I dati sono trasportati con la tecnica dei datagrammi
- Offre un servizio di comunicazione *connection-less*
- Gestisce indirizzamento, frammentazione, riassemblaggio e multiplexing dei protocolli
- Costituisce la base sulla quale si basano tutti gli altri protocolli, collettivamente noti come *TCP/IP suite*
  - TCP, UDP, ICMP, ARP
- È responsabile dell'instradamento dei pacchetti



# Il datagramma IP

- Un pacchetto IP è anche chiamato *datagramma*
- È costituito da un *header* e un'area dati
- I datagrammi possono avere dimensioni diverse
- La dimensione dell'header è solitamente fissata (**20 byte**) a meno che non siano presenti opzioni
- Un datagramma può contenere fino a un massimo di 65535 byte ( $2^{16} - 1$ )

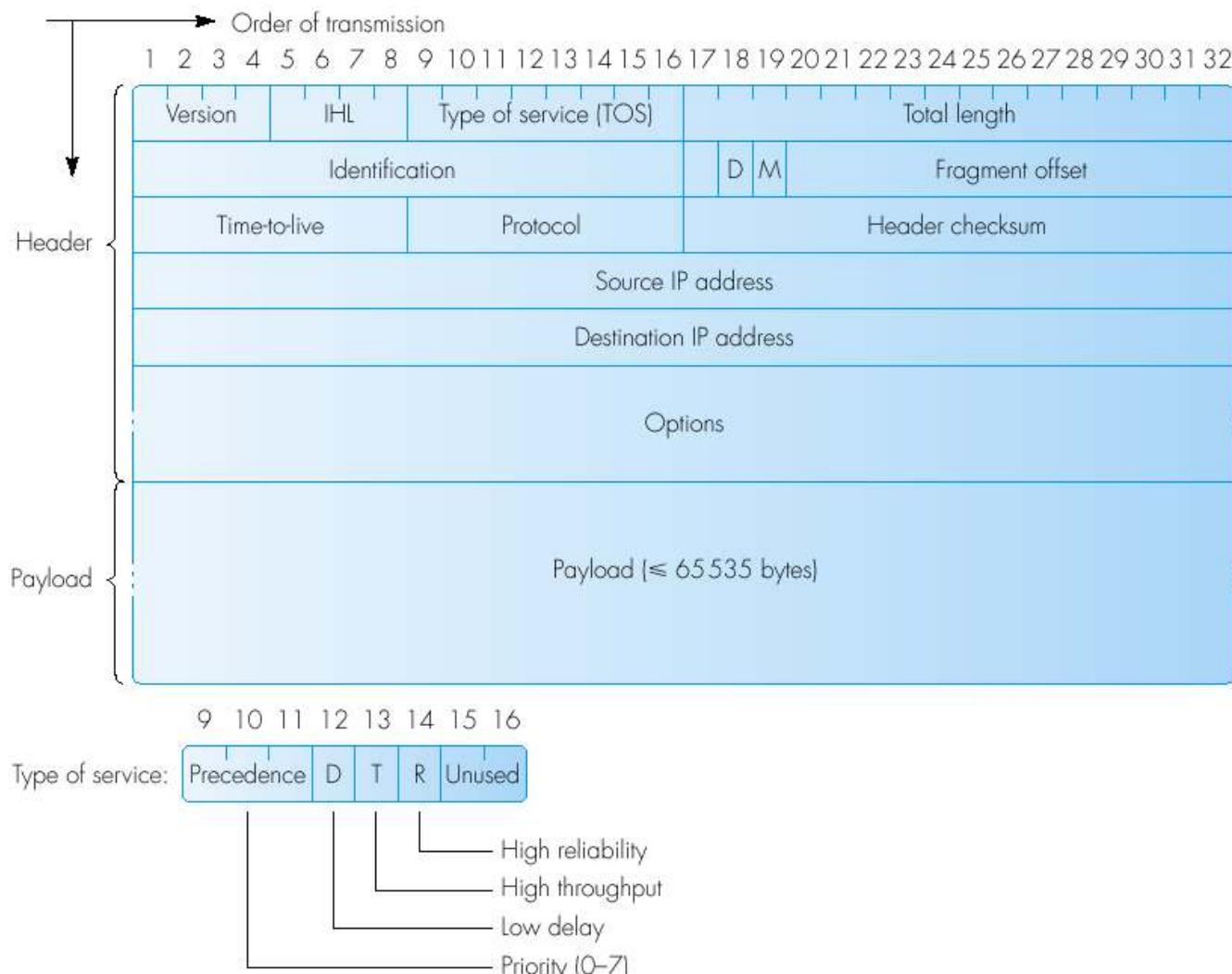


# L'header IP

- L'header contiene tutte le informazioni necessarie per la consegna del datagramma alla destinazione
  - Indirizzo destinazione
  - Indirizzo sorgente
  - Identificativo
  - Ed altro ancora...
- I router esaminano l'header di ogni datagramma e inoltrano il pacchetto lungo il percorso verso la destinazione
  - Usano tavole di routing per calcolare il *next hop*
  - Aggiornano tali tavole usando protocolli di routing dinamici



# Formato del pacchetto IP





# Formato del pacchetto IP

- **Version**
  - 4 bit, versione del protocollo IP cui il pacchetto è conforme
- **IP header length (IHL)**
  - 4 bit, lunghezza dell'header, in multipli di 32 bit (max 60 byte)
- **Type-of-Service (ToS)**
  - 8 bit, specifica come un protocollo di livello superiore vorrebbe che il pacchetto fosse trattato
- **Total length**
  - 16 bit, specifica la lunghezza in byte dell'intero pacchetto (header + dati)
  - ...max 64kB, cioè 65535 byte ( $2^{16} - 1$ )



# Formato del pacchetto IP

- **Time-to-live (TTL)**
  - 8 bit, contatore che viene gradualmente decrementato fino a zero, punto in cui il pacchetto viene scartato. Serve ad evitare che un pacchetto resti perennemente in circolo
- **Protocol**
  - 8 bit, indica il protocollo di livello superiore che riceve il pacchetto dopo che l'elaborazione IP è terminata
    - Analogo al numero di porto di livello trasporto: 6 indica TCP, 17 indica UDP
- **Header checksum**
  - 16 bit, aiuta a garantire l'integrità dell'header IP
- **Source Address**
  - 32 bit, specifica il nodo mittente



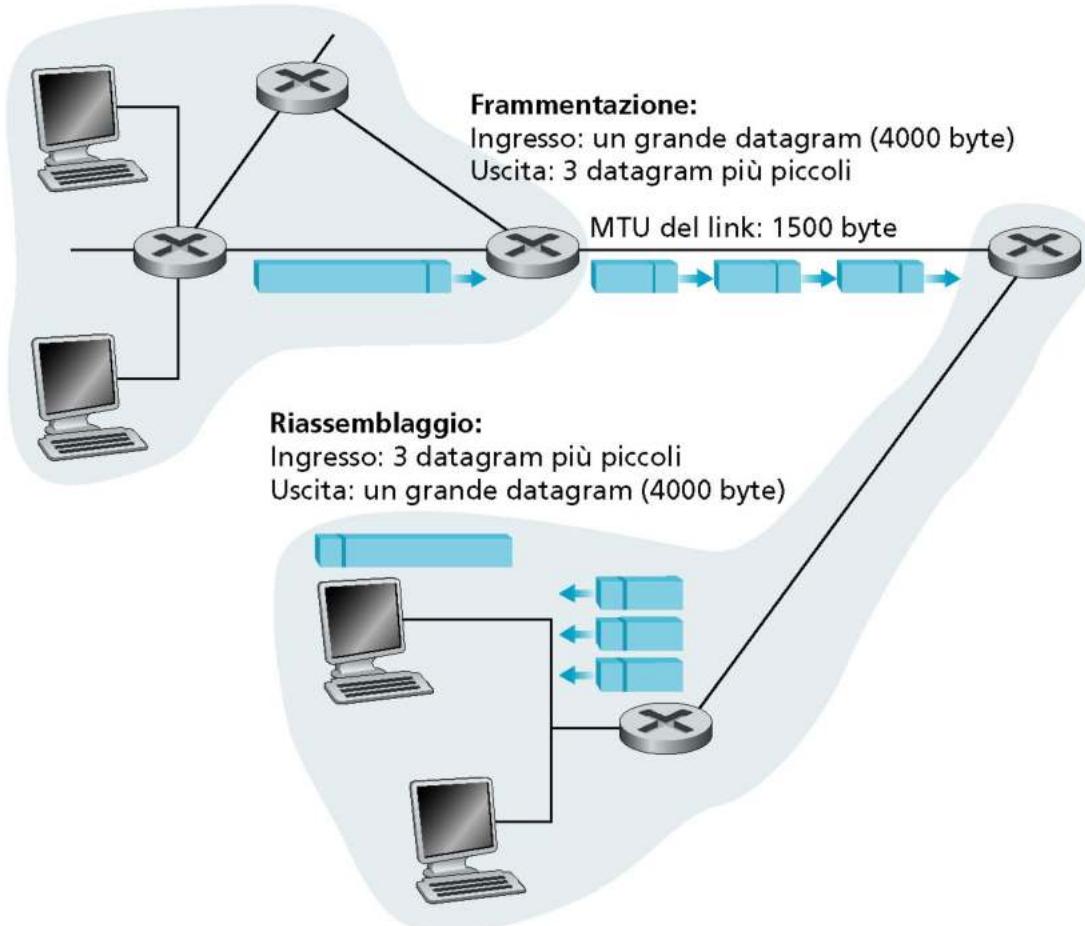
# Formato del pacchetto IP

- **Destination Address**
  - 32 bit, specifica il nodo ricevente
- **Identification**
  - I pacchetti possono essere frammentati lungo il percorso
  - Questo campo (16 bit) è un identificativo del datagramma
- **Flags**
  - Il bit D indica se il pacchetto può essere frammentato
  - Il bit M indica se il pacchetto è l'ultimo frammento
- **Fragment offset**
  - 13 bit, identifica la posizione del frammento all'interno del pacchetto



# Frammentazione e riassemblaggio IP

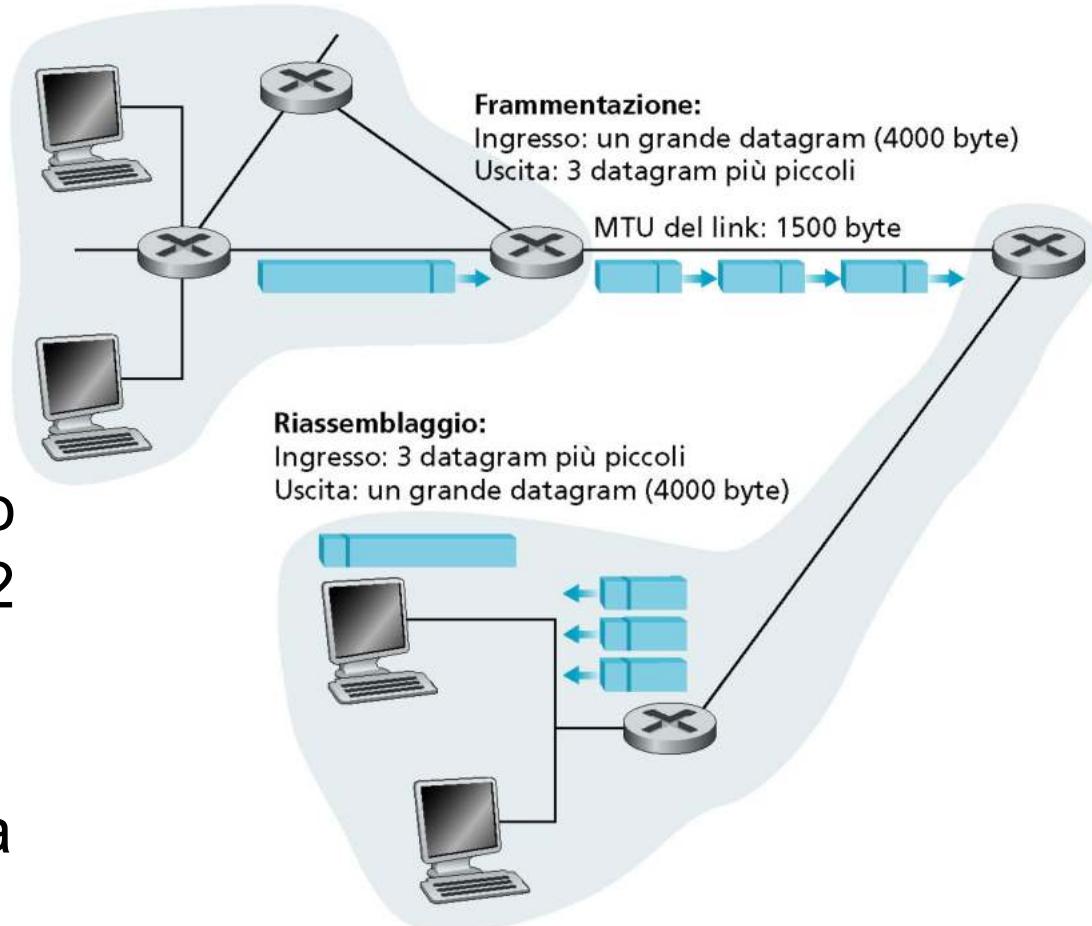
- Se un pacchetto di dimensione  $N$  arriva ad un router e deve essere trasmesso su un link di uscita con MTU  $M < N$ , il pacchetto è **frammentato**
- Ogni frammento è trasmesso come singolo pacchetto IP
- La dimensione di ogni frammento sarà un multiplo di 8 byte
- Tutti i frammenti hanno lo stesso ID number





# Frammentazione e riassemblaggio IP (2)

- Tutti i frammenti tranne l'ultimo devono essere multipli di 8 byte
- 8 byte è la dimensione del frammento elementare
- Avendo 13 bit a disposizione, ci possono essere al massimo 8192 frammenti per ogni datagramma
- La dimensione massima di un datagramma è 65535 byte

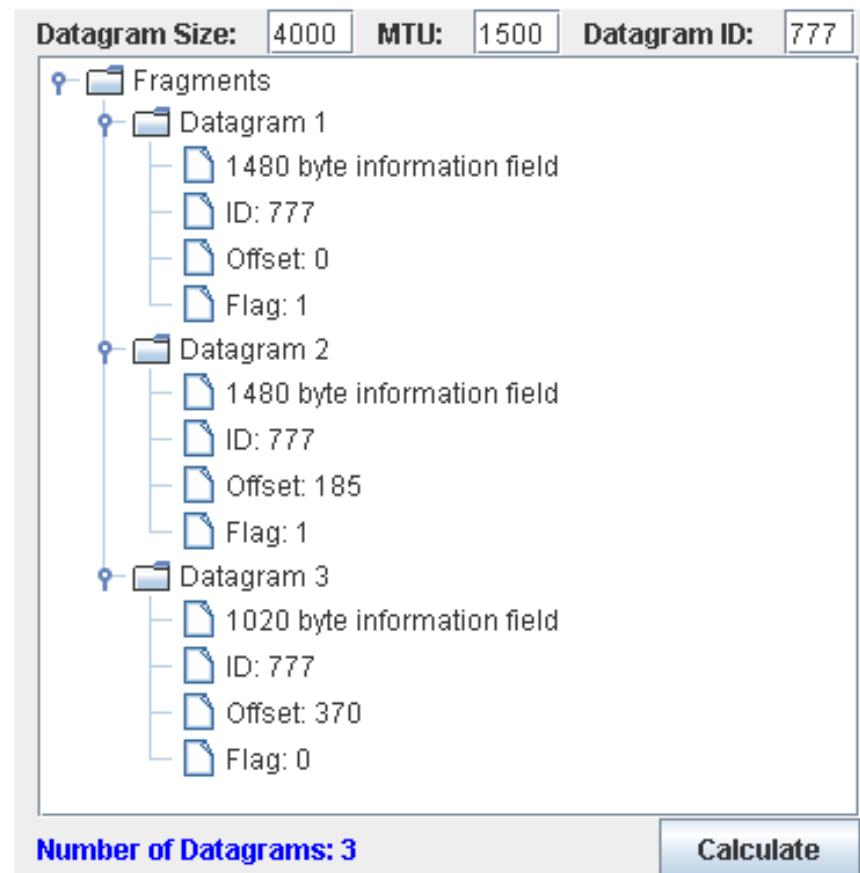




# Frammentazione IP: esempio 1

- $N = 4000$ ,  $MTU = 1500$
- Tre frammenti, ciascuno con header 20 byte
- Frammento 1:  
payload 1480  
offset 0
- Frammento 2:  
payload 1480  
offset  $(1480/8)=185$
- Frammento 3:  
payload 1020  
offset  $(1480+1480)/8=370$

Note: Datagram size includes an IP header of 20 bytes.  
MTU and Datagram size must be greater than 30, and all values must be less than  $2^{16} - 1$  (65535).



NOTA:  
 $20 + 1480 + 1480 + 1020 = 4000$

This applet was coded by Ryan Gilbert in 2008, a student at Arizona State University.  
It replaces an applet coded by Albert Huang in 1997 as part of course work at the University of Pennsylvania.



# Frammentazione IP: esempio 2

- Il pacchetto IP raffigurato di seguito deve attraversare un link avente Maximum Transfer Unit (MTU) pari a 1500 bytes. Come verrà trattato?

Original IP Datagram

Sequence	Identifier	Total Length	DF May / Don't	MF Last / More	Fragment Offset
0	345	5140	0	0	0

IP Fragments (Ethernet)

Sequence	Identifier	Total Length	DF May / Don't	MF Last / More	Fragment Offset
0-0	345	1500	0	1	0
0-1	345	1500	0	1	185
0-2	345	1500	0	1	370
0-3	345	700	0	0	555



# Opzioni

- È il modo per estendere IP con un numero variabile di opzioni
  - Security
  - Source routing
  - Route recording
  - Stream identification
  - Timestamping
- A causa delle opzioni, l'header può essere di lunghezza variabile
  - Questo è il motivo della presenza del campo IHL
  - Se l'opzione non occupa 4 byte (o un suo multiplo), vengono inseriti dei bit di riempimento (tutti zero)
  - Le presenza opzionale di questi campi rende difficile la gestione in implementazioni hw-based



# IP: servizio Best Effort

- IP *non* garantisce di prevenire:
  - Datagrammi duplicati
  - Consegna ritardata o fuori ordine
  - Corruzione di dati
  - Perdita di pacchetti (e di frammenti)
- La consegna affidabile dei pacchetti può avvenire grazie a meccanismi di controllo da parte di protocolli di livello superiore



# Corso di Laurea in Informatica

## Corso di Reti di Calcolatori 1

Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))

---

Il protocollo IP: indirizzamento  
Classi di indirizzi e subnetting

I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso

# Nota di copyright per le slide COMICS



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta



# Indirizzi IP

- Ad ogni host è assegnato un indirizzo IP o indirizzo Internet
  - È un numero di 32 bit = 4 byte
  - Unico in tutta Internet
- Ogni indirizzo IP è diviso in un prefisso e un suffisso
  - Il prefisso indica la rete alla quale l'host è collegato
    - Due reti differenti hanno numero di rete differente
  - Il suffisso identifica l'host all'interno della rete
    - Due host sulla stessa rete non possono avere lo stesso suffisso, ma host su reti diverse possono avere lo stesso suffisso





# Notazione *dotted decimal*

- La notazione dotted decimal rappresenta gli indirizzi IP come 4 numeri decimali separati da punto
- Ogni numero decimale, poiché rappresenta un byte, è compreso tra 0 e 255

32-bit Binary Number	Equivalent Dotted Decimal
10000001 00110100 00000110 00000000	129 . 52 . 6 . 0
11000000 00000101 00110000 00000011	192 . 5 . 48 . 3
00001010 00000010 00000000 00100101	10 . 2 . 0 . 37
10000000 00001010 00000010 00000011	128 . 10 . 2 . 3
10000000 10000000 11111111 00000000	128 . 128 . 255 . 0



# Sistema di numerazione decimale (cenni)

- La base è 10. Questo vuol dire che
  - Si utilizzano 10 simboli (le cifre da 0 a 9)
  - Ogni cifra è pesata secondo una potenza di 10 con esponente crescente da destra verso sinistra
- es.

$$(2536)_{10} = 2*10^3 + 5*10^2 + 3*10^1 + 6*10^0$$



# Sistema di numerazione binaria (cenni)

- La base è 2. Questo vuol dire che
  - Si utilizzano 2 simboli (le cifre 0 e 1)
  - Ogni cifra è pesata secondo una potenza di 2 con esponente crescente da destra verso sinistra
- es.  
$$(11010)_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (26)_{10}$$
- Abbiamo così visto anche il modo per convertire un numero da base 2 a base 10

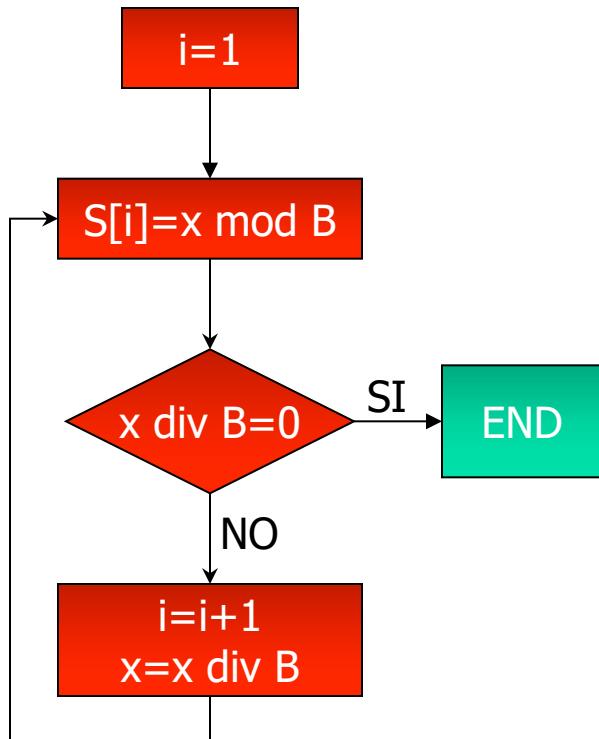


# Sistema di numerazione esadecimale (cenni)

- La base è 16. Questo vuol dire che
  - Si utilizzano 16 simboli (le cifre da 0 a 9 e le lettere da A a F)
  - Ogni cifra è pesata secondo una potenza di 16 con esponente crescente da destra verso sinistra
- es.  
$$(1E5)_{16} = 1*16^2 + 14*16^1 + 5*16^0 = (485)_{10}$$
- Abbiamo così visto anche il modo per convertire un numero da base 16 a base 10



# Conversione da decimale (cenni)



## Esempio

Vogliamo convertire 25 in binario:

$$25 \bmod 2 = 1 \Rightarrow S=1$$

$$25 \div 2 = 12$$

$$12 \bmod 2 = 0 \Rightarrow S=01$$

$$12 \div 2 = 6$$

$$6 \bmod 2 = 0 \Rightarrow S=001$$

$$6 \div 2 = 3$$

$$3 \bmod 2 = 1 \Rightarrow S=1001$$

$$3 \div 2 = 1$$

$$1 \bmod 2 = 1 \Rightarrow S=11001$$

$$1 \div 2 = 0$$

END

$$(25)_{10} = (11001)_2$$

i: contatore

S: stringa risultato

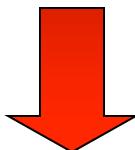
x: numero da convertire

B: base

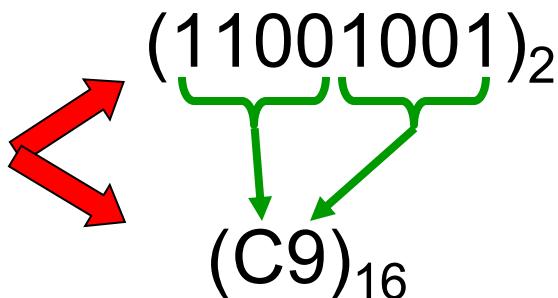


# Relazione tra numeri binari ed esadecimali

- Una stringa di 4 bit può assumere  $2^4=16$  diversi valori
- Se consideriamo ogni stringa come numero binario, essa rappresenta un numero compreso tra 0 e 15
- In base 16 abbiamo 16 cifre che assumono un valore compreso tra 0 e 15



- 1 cifra esadecimale “rappresenta” 4 cifre decimali



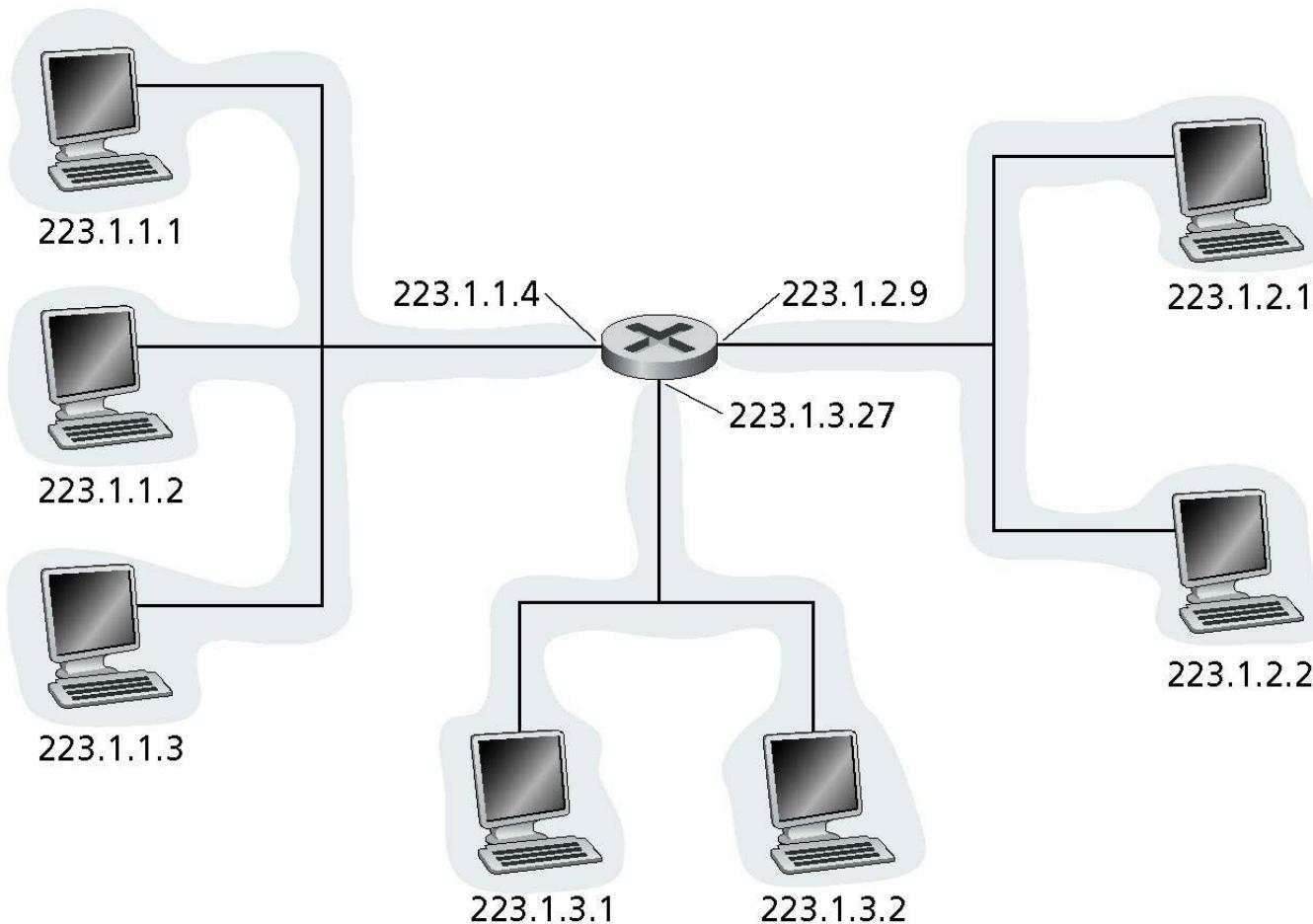


# Chi assegna gli indirizzi IP?

- **ICANN:**
  - Internet Corporation for Assigned Names and Numbers
- Assegna gli indirizzi (pubblici)
- Gestisce il DNS
- Assegna i nomi dei domini
- Risolve eventuali dispute (conflitti di nomi e/o indirizzi)
- NON PIU' sotto il controllo del governo statunitense dal 1/10/2016

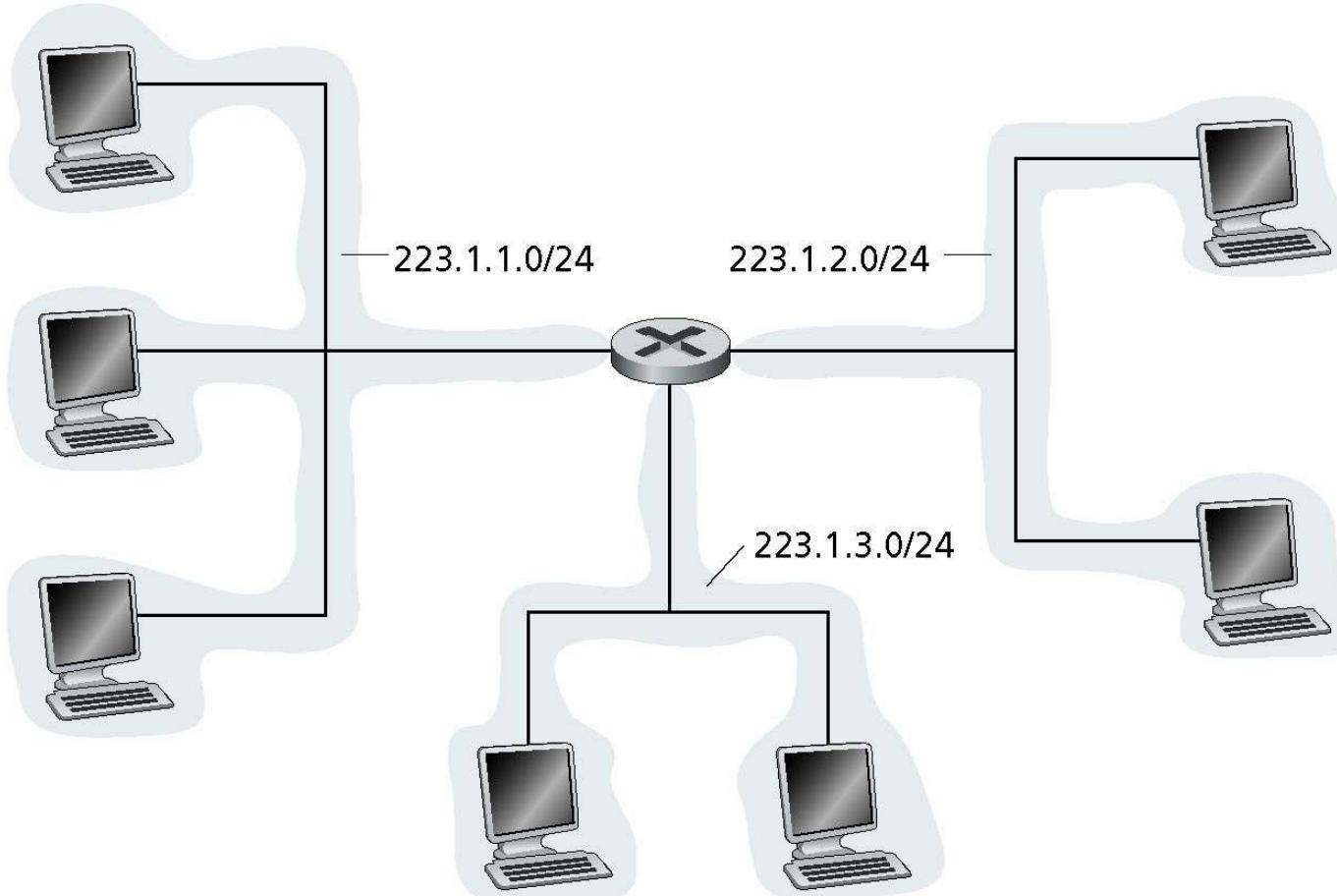


# Indirizzi delle interfacce...





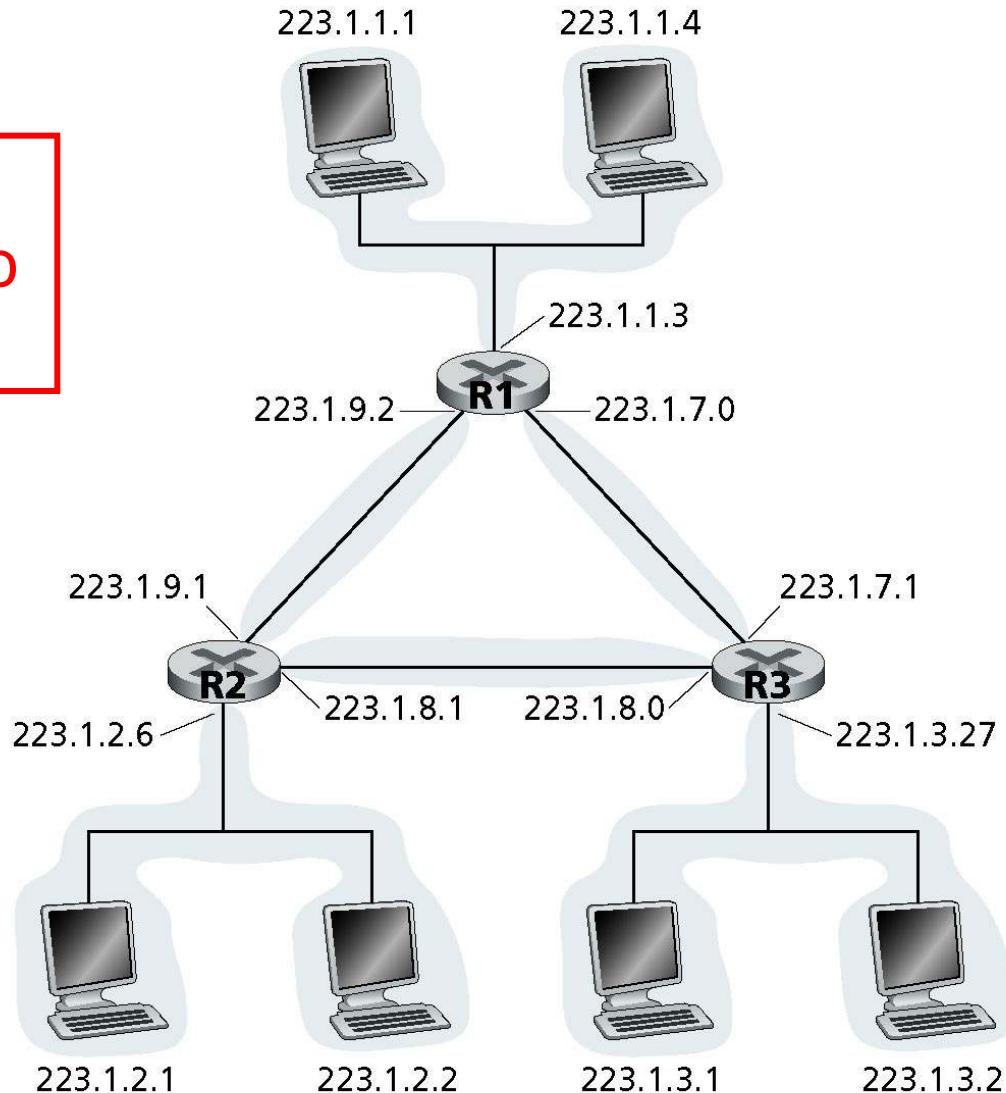
# ...indirizzi delle reti





# Tre router che interconnettono sei host

Domanda:  
Quante sono  
le reti?



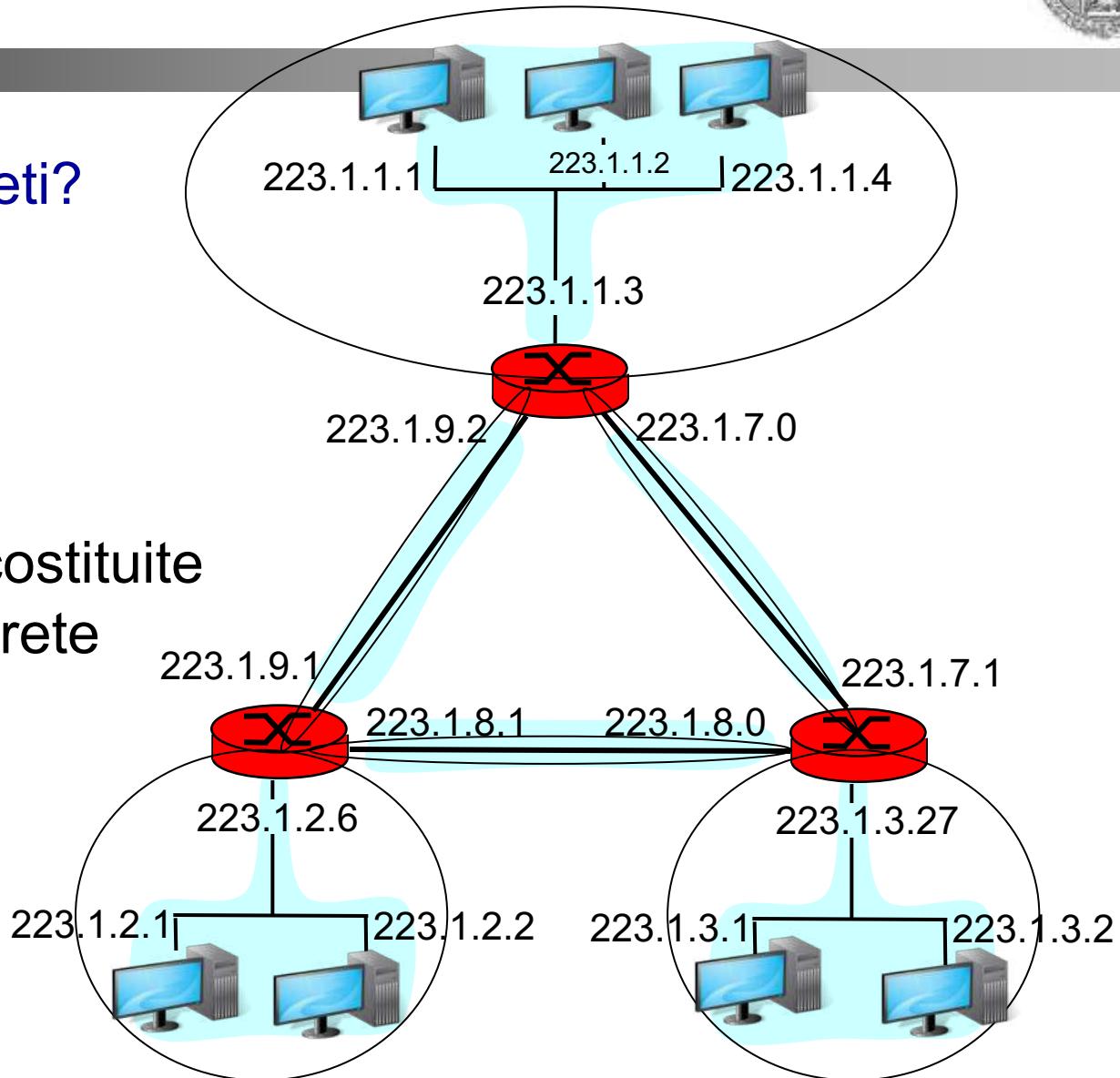


# Soluzione...

Come trovare le reti?

- ‘Staccare’ ogni interfaccia dal corrispondente router/host
- Creare “isole” costituite da segmenti di rete disgiunti

Nell'esempio:  
sistema  
interconnesso  
costituito da sei  
reti





# Classi di indirizzi

- La parte di indirizzo che specifica la rete e quella che specifica l'host non hanno lunghezza fissa, ma variano a seconda della *classe* a cui appartiene l'indirizzo
- Sono state originariamente definite 5 classi
  - 3 (A, B, C) sono usate per gli indirizzi degli host e si differenziano per la lunghezza della parte rete/host
  - 1 (D) è usata per il *multicast*
  - 1 (E) è riservata per usi futuri



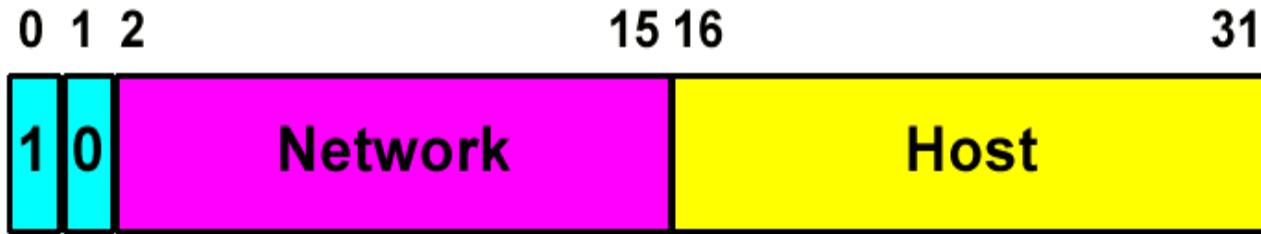
# Indirizzi di classe A



- Campo rete
  - 7 bit
  - Massimo 128 reti
  - Il primo byte è compreso tra 0 e 127
- Campo host
  - 24 bit
  - Massimo  $2^{24} \approx 16M$  host



# Indirizzi di classe B



- Campo rete
  - 14 bit
  - Massimo 16k reti
  - Il primo byte è compreso tra 128 e 191
- Campo host
  - 16 bit
  - Massimo  $2^{16} \approx 64k$  host



# Indirizzi di classe C



- Campo rete
  - 21 bit
  - Massimo 2M reti
  - Il primo byte è compreso tra 192 e 223
- Campo host
  - 8 bit
  - Massimo 256 host



# Indirizzi di classe D e E





# Classless Inter-Domain Routing

- Detto CIDR
- Standard di IETF a partire dal 1993
  - Ultima versione RFC 4632 del 2006
- Introduce la notazione a.b.c.d/n per definire la sottorete
- Supera il limite dell'allocazione degli indirizzi a classi (detto anche classfull)
  - La sottorete può essere di dimensione arbitraria, non più solo /8, /16, o /24
  - E' per questo detto classless
- Consente un utilizzo migliore dello spazio degli indirizzi



# Indirizzi IP “speciali”

- **Network address**
  - La rete stessa ha un indirizzo, il cui suffisso è costituito da tutti ‘0’
  - Nessun host può quindi avere tutti ‘0’ nel suffisso
- **Directed broadcast address**
  - Per mandare un messaggio in broadcast ad una rete il suffisso è costituito da tutti ‘1’: il pacchetto è inviato a tutti gli host di una specifica rete
- **Limited broadcast address**
  - L'intero indirizzo (sia la parte rete che la parte host) è costituito da tutti ‘1’: 255.255.255.255
  - Broadcast sulla LAN locale



# Indirizzi IP “speciali”

- **This computer address**

- L'intero indirizzo è costituito da tutti '0'
- Per ottenere un indirizzo automaticamente all'avvio, si potrebbe usare IP per comunicare...
- ...ma non abbiamo ancora un indirizzo
  - vedremo che per l'assegnazione di tale indirizzo si utilizzano protocolli quali DHCP (Dynamic Host Configuration Protocol) e BOOTP (Boot Protocol)

- **Loopback address**

- Ogni indirizzo che comincia con 127 indica il computer locale
- 127.0.0.1 è il più comune, ma va bene anche 127.0.44.53
- Usato per test, nessun pacchetto esce sulla rete
- Utile quando il computer non ha schede di rete



# Indirizzi IP

- La suddivisione degli indirizzi IP in classi non è efficiente perché comporta lo spreco di indirizzi
- 32 bit  $\Rightarrow 2^{32} \approx 4$  miliardi di indirizzi diversi, ma non tutti vengono usati
- Rimedi
  - Indirizzi privati
  - Sottoreti



# Indirizzi IP privati

- L'RFC 1918 (1996) + RFC 6598 (2012) riservano i seguenti blocchi di indirizzi per uso privato
  - 10.0.0.0 - 10.255.255.255 /8, 1 Classe A
  - 172.16.0.0 - 172.31.255.255 /12, 32 Classe B
  - 192.168.0.0 - 192.168.255.255 /16, 256 Classe C
  - (new) 100.64.0.0 - 100.127.255.255 /10,  $\frac{1}{4}$  Classe A
- I router di Internet generalmente non inoltrano pacchetti aventi indirizzo sorgente o destinazione compreso in uno di questi blocchi
- Vedremo dopo il modo per tradurre tali indirizzi in indirizzi Internet pubblici (NAT)
- Gli indirizzi 100.64.0.0/10 sono per Carrier-Grade NAT



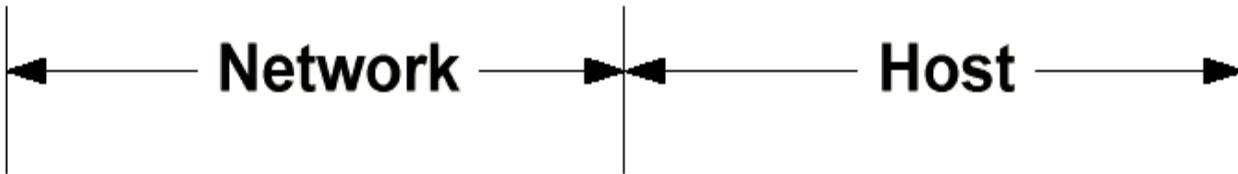
# Sottoreti

- Gli indirizzi IP sono assegnati in modo che tutti gli host sulla stessa rete locale appartengono alla stessa sottorete
- Una sottorete è individuata dai bit del prefisso più alcuni bit presi in prestito dal suffisso, come specificato dalla **subnet mask**
- Una subnet mask è una stringa di 32 bit associata ad ogni host
  - Gli ‘1’ definiscono la porzione di indirizzo che identifica la sottorete
  - Gli ‘0’ definiscono la porzione di indirizzo che identifica l’host
- **L’indirizzo della sottorete si ottiene mediante un AND bit a bit tra l’indirizzo dell’host e la netmask**
- Esempio: vogliamo utilizzare un unico indirizzo di classe B avendo diverse reti fisiche. Se prendiamo in prestito 8 bit dal suffisso avremo a disposizione 256 sottoreti, ognuna delle quali potrà avere 254 host
  - L’host 128.192.56.50 con netmask 255.255.255.0 appartiene alla sottorete 128.192.56.0

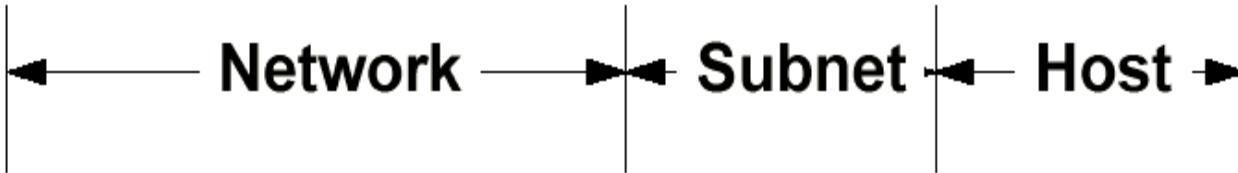


# Sottoreti

Indirizzo di classe B prima del subnetting

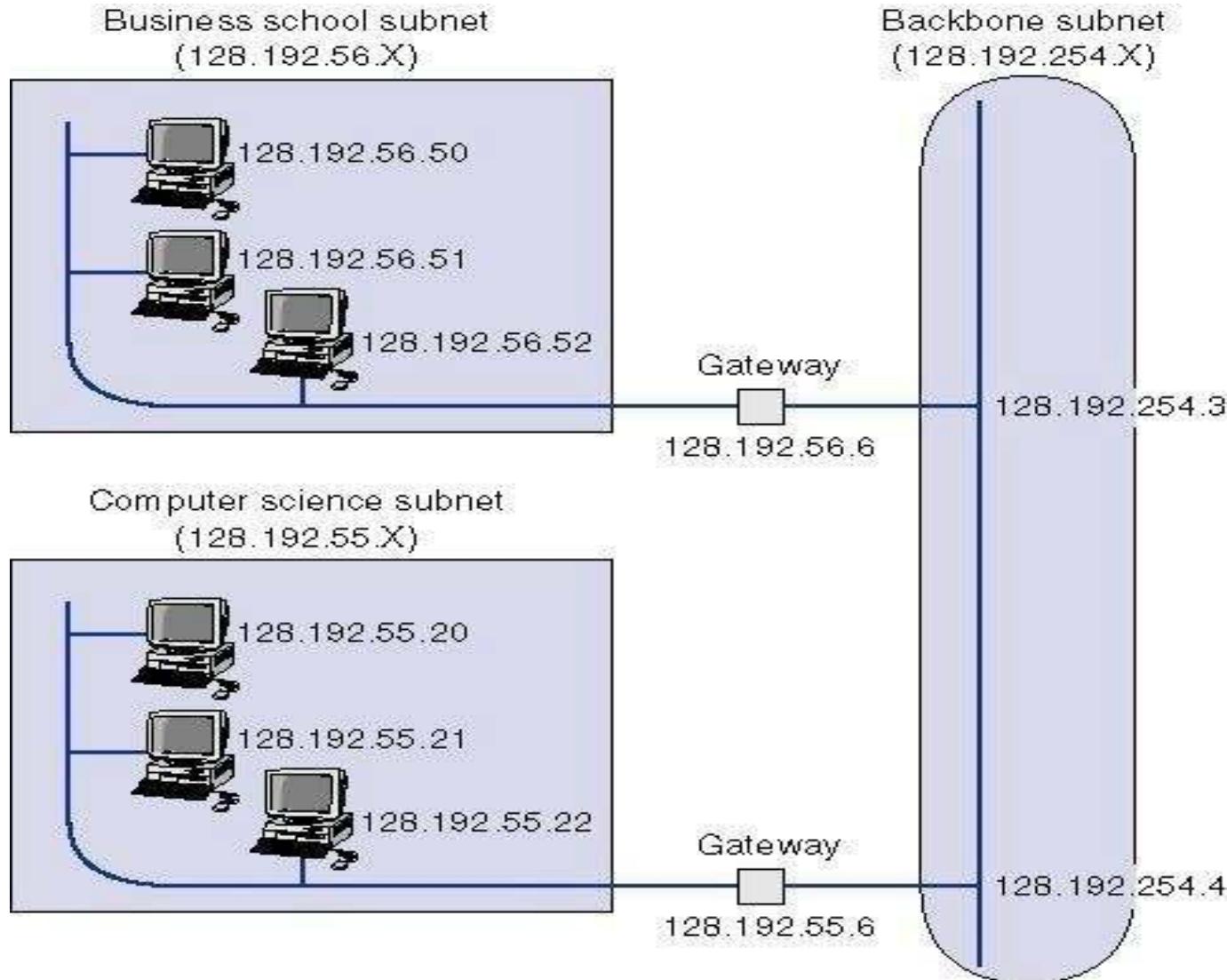


Indirizzo di classe B dopo il subnetting





# Sottoreti: un esempio





# Sottoreti

- Suddividere una rete in sottoreti ci consente di allocare in maniera efficiente gli indirizzi, migliorando al tempo stesso le prestazioni (il traffico relativo ad una sottorete non viene introdotto nelle altre)
- Come viene utilizzata una subnet mask?
  - Da un host che deve trasmettere un pacchetto
    - Confronta la destinazione con la propria subnet mask
    - Se la dest è sulla stessa sottorete, invia sulla LAN
    - Altrimenti, invia al gateway
  - Da un router all'interno della rete suddivisa in sottorete
    - Utilizza la subnet mask con l'indirizzo di rete delle reti collegate per determinare la giusta destinazione



# Come viene utilizzata la subnet mask da un host

- RFC 950 (Internet Standard Subnetting Procedure)

```
IF bitwise_and(dg.ip_dest, my_ip_mask) ==
    bitwise_and(my_ip_addr, my_ip_mask)
```

THEN

```
    send_dg_locally(dg, dg.ip_dest)
```

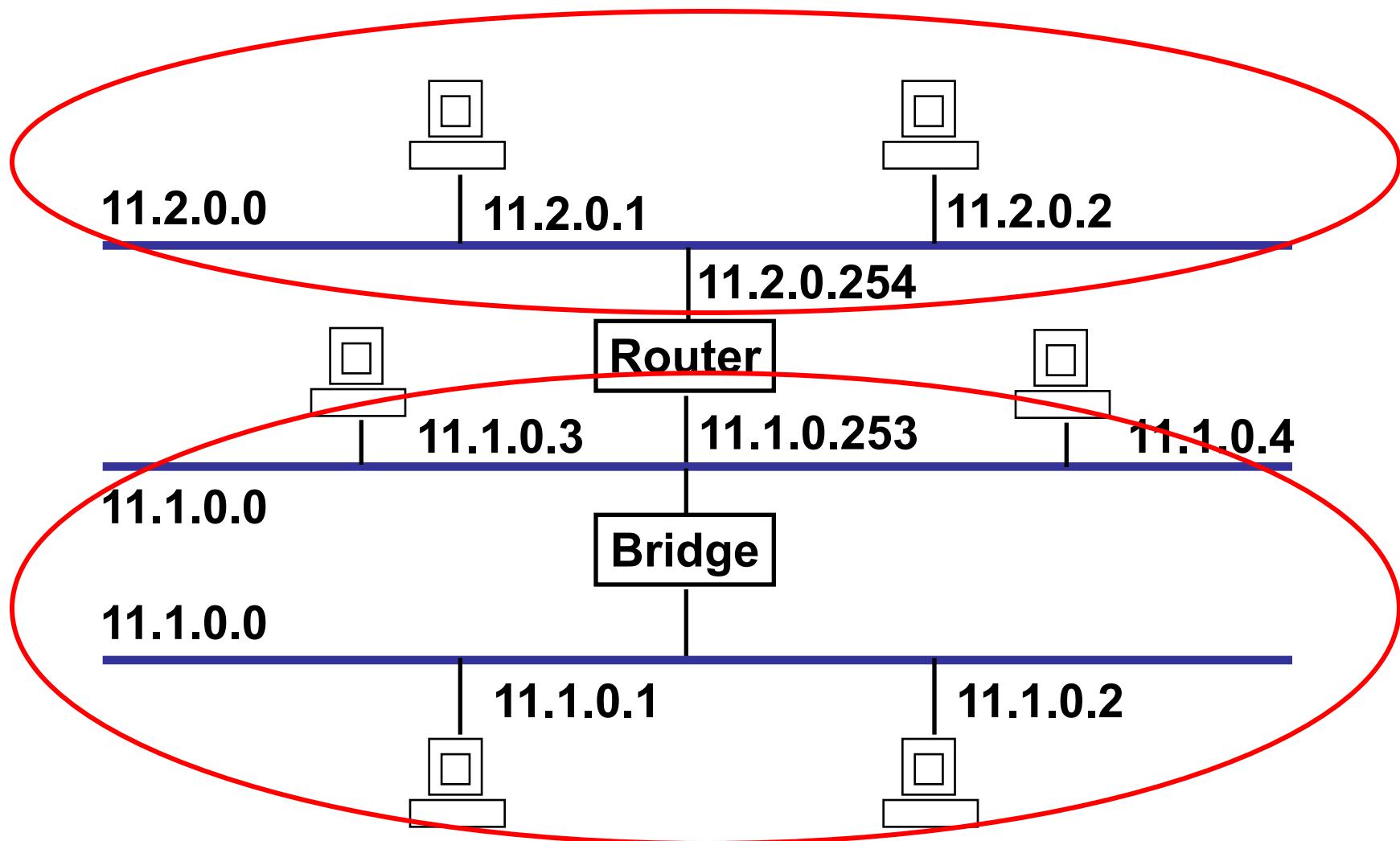
ELSE

```
    send_dg_locally(dg, gateway_to(bitwise_and(dg.ip_dest, my_ip_mask)))
```

- dg sta per datagram
- gateway\_to è l'indirizzo del next hop router ottenuto dalla routing table



# Reti logiche e fisiche





# Netmask

- Parametro che specifica il subnetting
  - bit a 1 in corrispondenza dei campi network e subnetwork
  - bit a 0 in corrispondenza del campo host
- Esempio: si supponga di voler partizionare una rete di classe B in 16 subnet da 4096 host
  - Netmask 11111111 11111111 11110000 00000000
  - Netmask esadecimale ff ff f0 00
  - Netmask decimale 255.255.240.0
  - /20



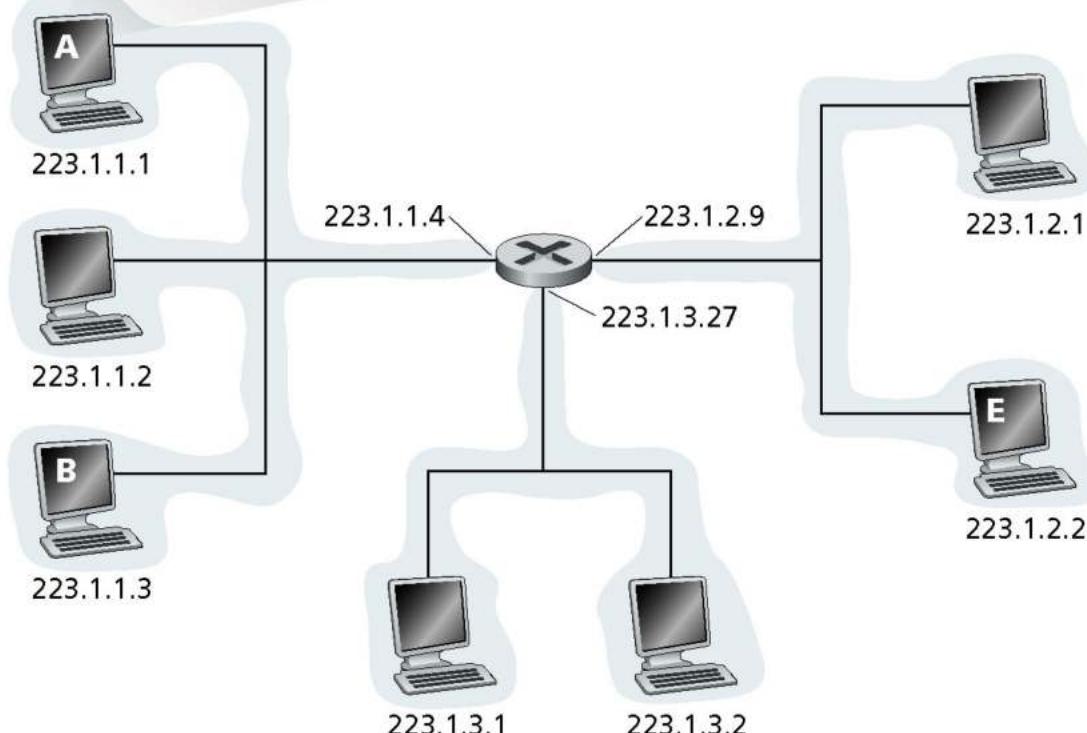
# Subnet e reti fisiche

- IP assume una corrispondenza biunivoca tra reti fisiche e subnet
  - routing implicito all'interno di una subnet
  - Il routing tra subnet diverse è esplicito
    - gestito dai router tramite tabelle di instradamento



# Es: tabella di routing nell'host A

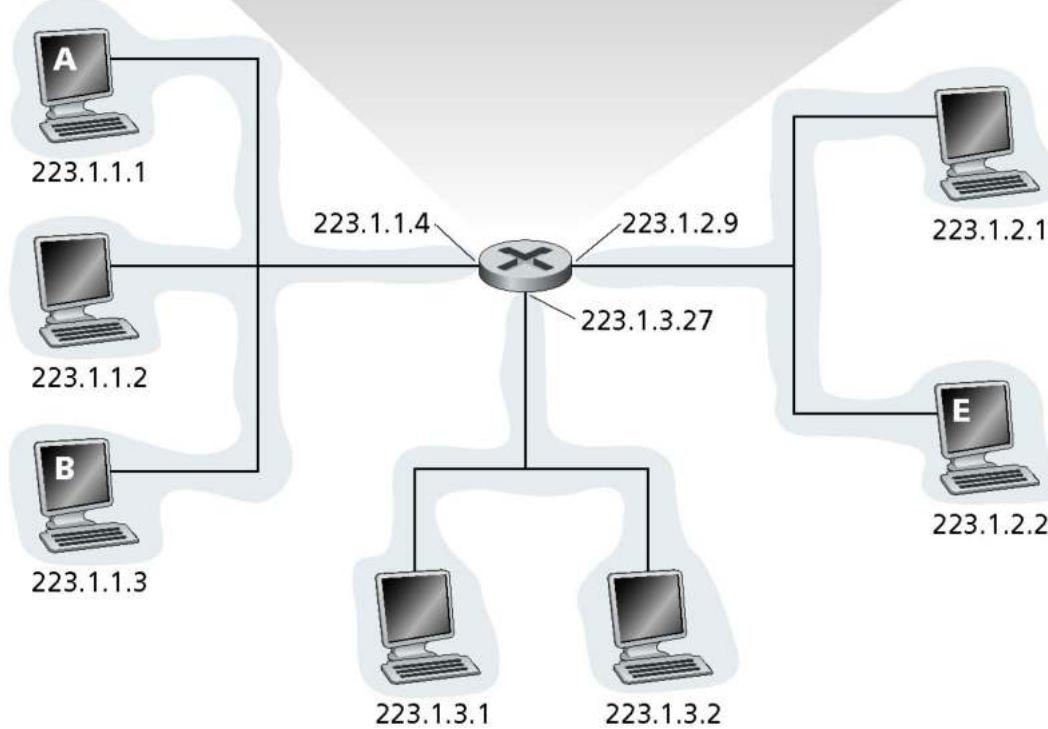
Tabella di rilancio in A		
Rete di destinazione	Router successivo	Numero salti
223.1.1.0/24		1
223.1.2.0/24	223.1.1.4	2
223.1.3.0/24	223.1.1.4	2





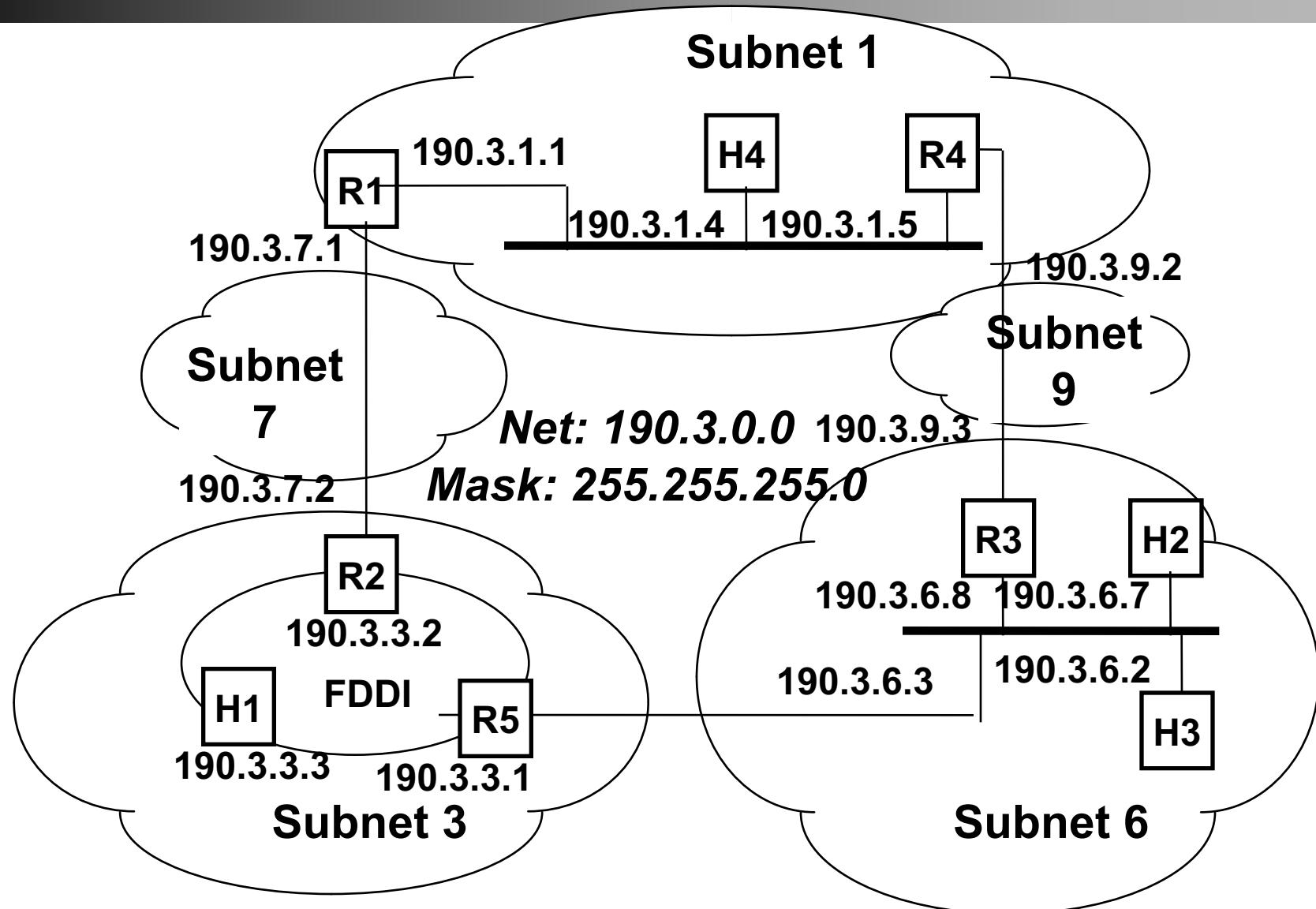
# Es: tabella di routing nel router

Tabella di rilancio nel router			
Rete di destinaz.	Router success.	N. hop	Interfaccia
223.1.1.0/24	—	1	223.1.1.4
223.1.2.0/24	—	1	223.1.2.9
223.1.3.0/24	—	1	223.1.3.27





# Esempio





# Subnet: instradamento

- All'interno della subnet l'instradamento deve essere fornito dalla rete fisica
- Corrispondenza tra gli indirizzi di subnet (indirizzi IP) e gli indirizzi di livello 2 gestita da ARP (Address Resolution Protocol)
- Indirizzi di livello 2
  - Indirizzi MAC sulle LAN
  - Indirizzi di DTE in X.25
  - Identificatori di LCI in Frame Relay
  - .....



# Default Route

- Gli host devono conoscere almeno un router presente sulla loro rete fisica
- Il protocollo ICMP permette di ottimizzare dinamicamente il routing
- Ad esempio sull'host H4
  - `route add default gw 190.3.1.5`



# Tabelle di Instradamento

- L'instradamento tra subnet diverse viene gestito da tavelle di instradamento presenti sui router
- Esempio
  - tavelle di instradamento del router R5
    - 3 subnet non raggiungibili direttamente

<b>Subnet di Destinazione</b>	<b>Indirizzo del router</b>
190.3.1.0	190.3.3.2
190.3.7.0	190.3.3.2
190.3.9.0	190.3.6.8



# **Corso di Laurea in Informatica**

## **Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

### **Network Address Translation (NAT)**

**I lucidi presentati al corso sono uno strumento didattico  
che NON sostituisce i testi indicati nel programma del corso**



## Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,  
Marcello Esposito, Roberto Canonica, Giorgio Ventre, Alessio Botta



# NAT: Network Address Translation

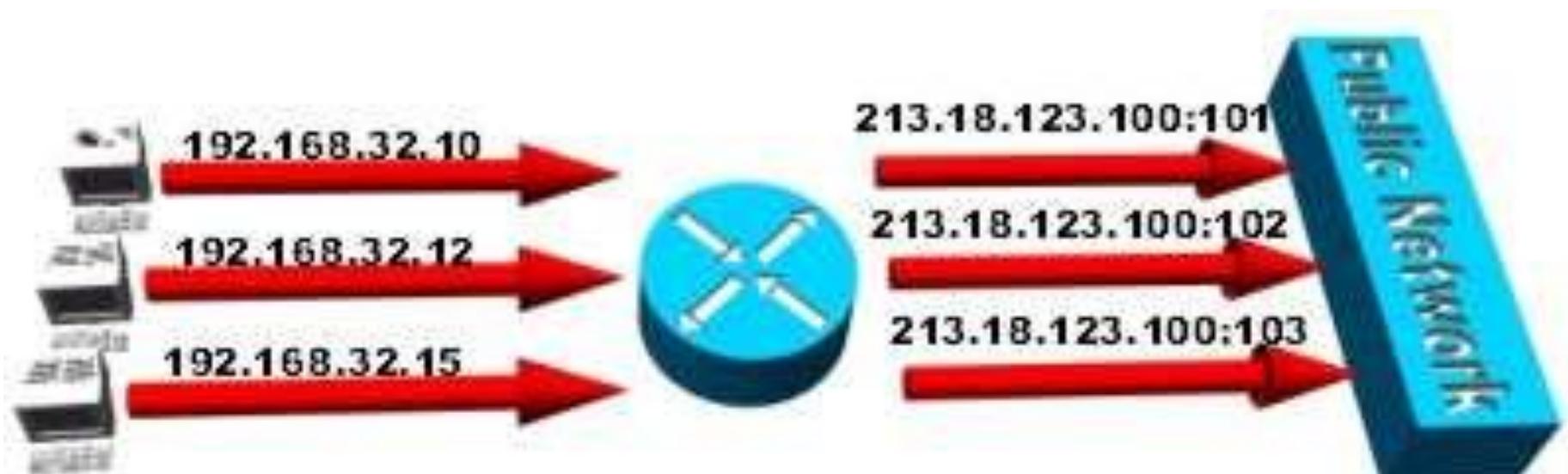
- Network Address Translation (RFC 1631) è una tecnica che consente ad un dispositivo (router) di agire come intermediario tra Internet (rete pubblica) e una rete privata
- In questo modo, un unico indirizzo IP può rappresentare un intero gruppo di computer di una rete privata





# NAT

- L'uso più comune del NAT è quello di mappare un insieme di indirizzi privati su di un unico indirizzo pubblico, utilizzando differenti porti (di livello trasporto) per mantenere traccia dell'indirizzo privato di provenienza





# NAT

- Quando il router riceve un pacchetto inviato da un computer della rete privata ad un computer esterno, salva in una tabella l'indirizzo e il porto del mittente, oltre ai nuovi valori che esso assegna
- Tale tabella viene consultata anche quando il router riceve un pacchetto in entrata susseguente ad un pacchetto n uscita

Source Computer	Source Computer's IP Address	Source Computer's Port	NAT Router's IP Address	NAT Router's Assigned Port Number
A	192.168.32.10	400	215.37.32.203	1
B	192.168.32.13	50	215.37.32.203	2
C	192.168.32.15	3750	215.37.32.203	3
D	192.168.32.18	206	215.37.32.203	4

Tabella  
NAT per  
TCP

# NAT: Network Address Translation



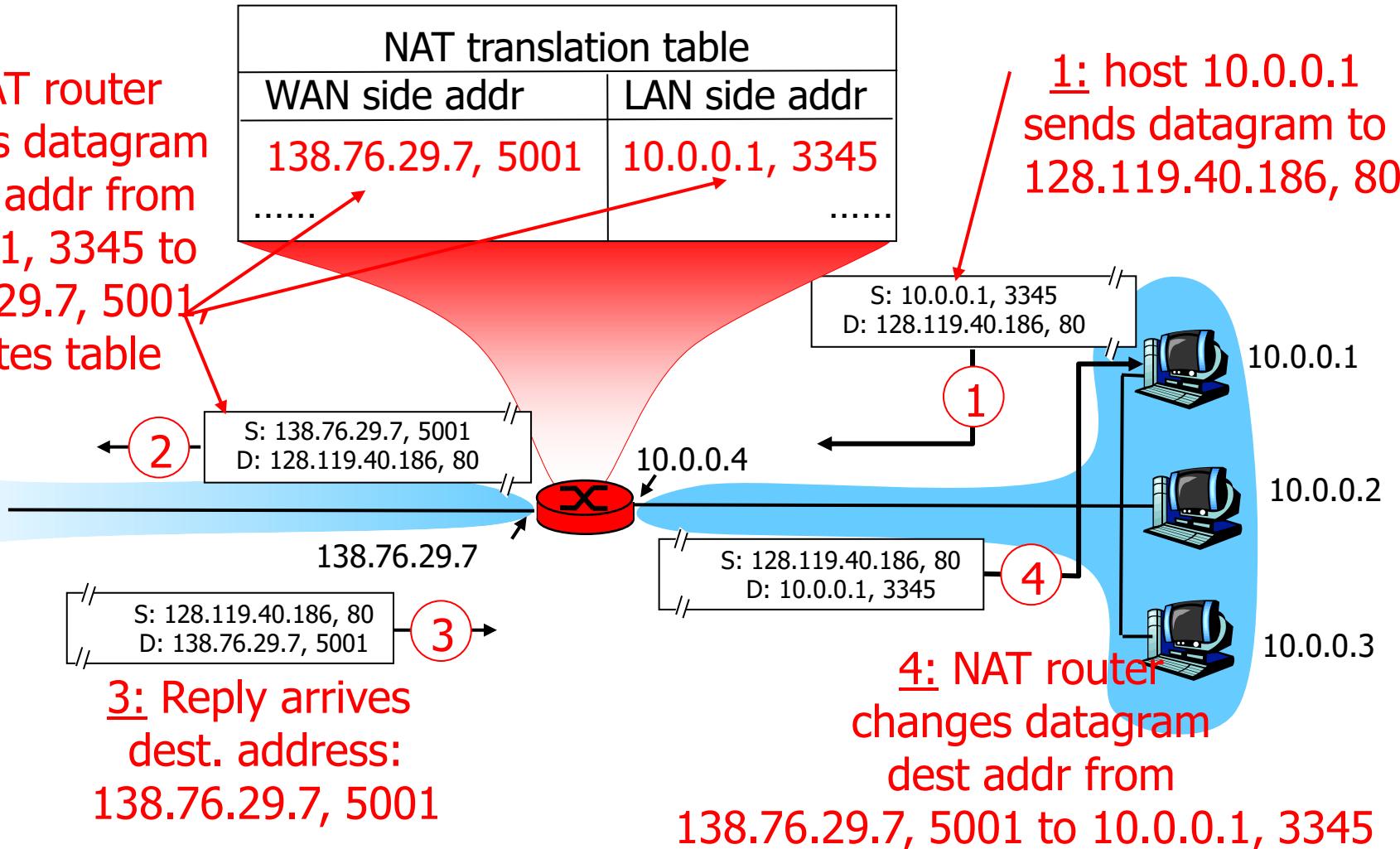
Implementation: NAT router must

- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)  
. . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr.
- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair
- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table



# NAT: un esempio

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001 updates table



3: Reply arrives  
dest. address:  
138.76.29.7, 5001

1: host 10.0.0.1  
sends datagram to  
128.119.40.186, 80

4: NAT router  
changes datagram  
dest addr from  
138.76.29.7, 5001 to 10.0.0.1, 3345



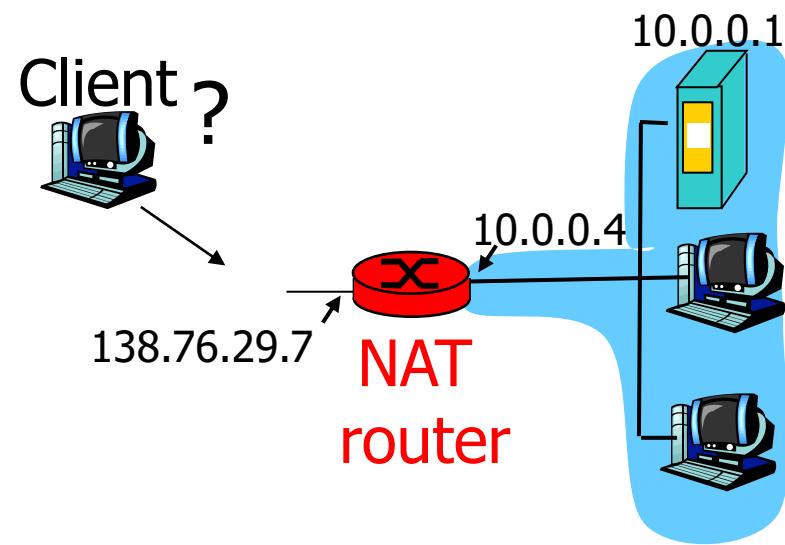
# NAT: Network Address Translation

- 16-bit port-number field
  - 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial
  - routers should only process up to layer 3
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, eg, P2P applications
  - address shortage should instead be solved by IPv6

# NAT traversal problem



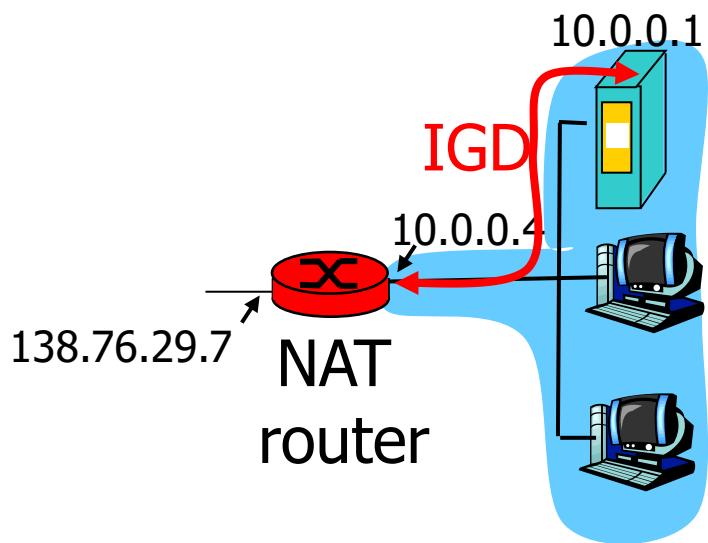
- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can not use it as destination addr)
  - only one externally visible NATted address: 138.76.29.7
- solution 1: statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



# NAT traversal problem



- solution 2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATted host to
  - learn public IP address (138.76.29.7)
  - add/remove port mappings (with lease times)

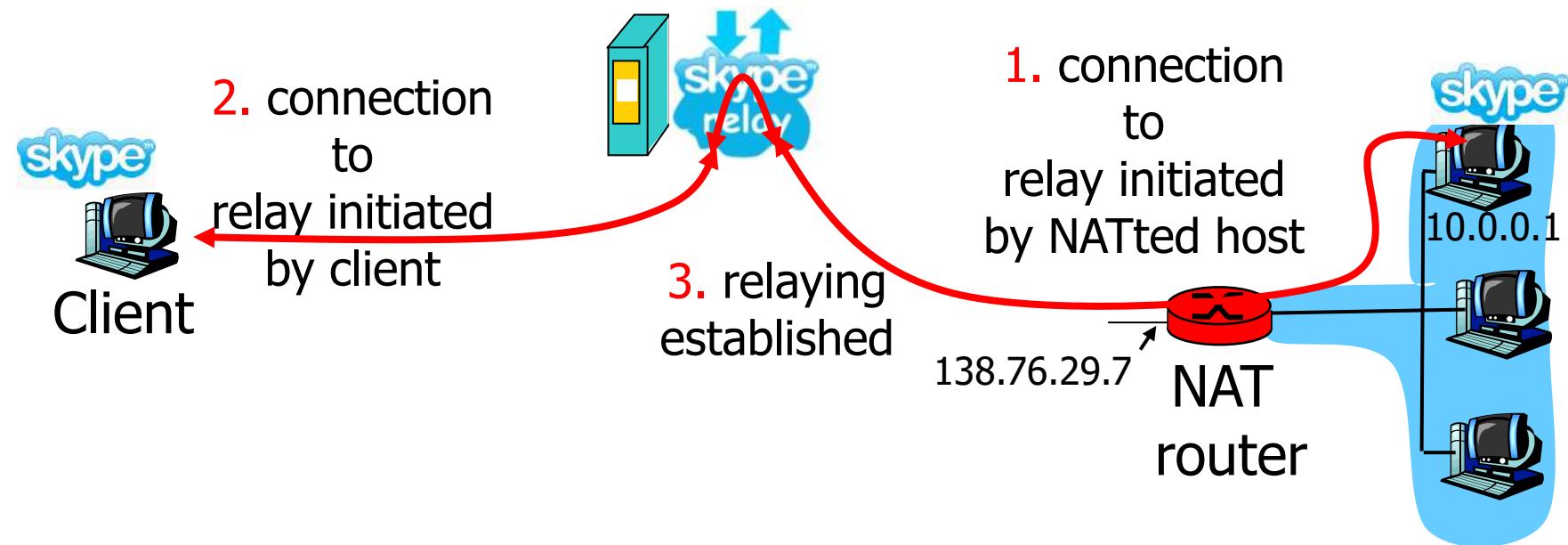


i.e., automate static NAT port map configuration

# NAT traversal problem



- solution 3: relaying (used in Skype)
  - NATed client establishes connection to relay
  - External client connects to relay
  - relay bridges packets between two connections





**Corso di Laurea in Informatica**

**Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

---

**Il livello trasporto:**

**Introduzione e protocollo UDP**

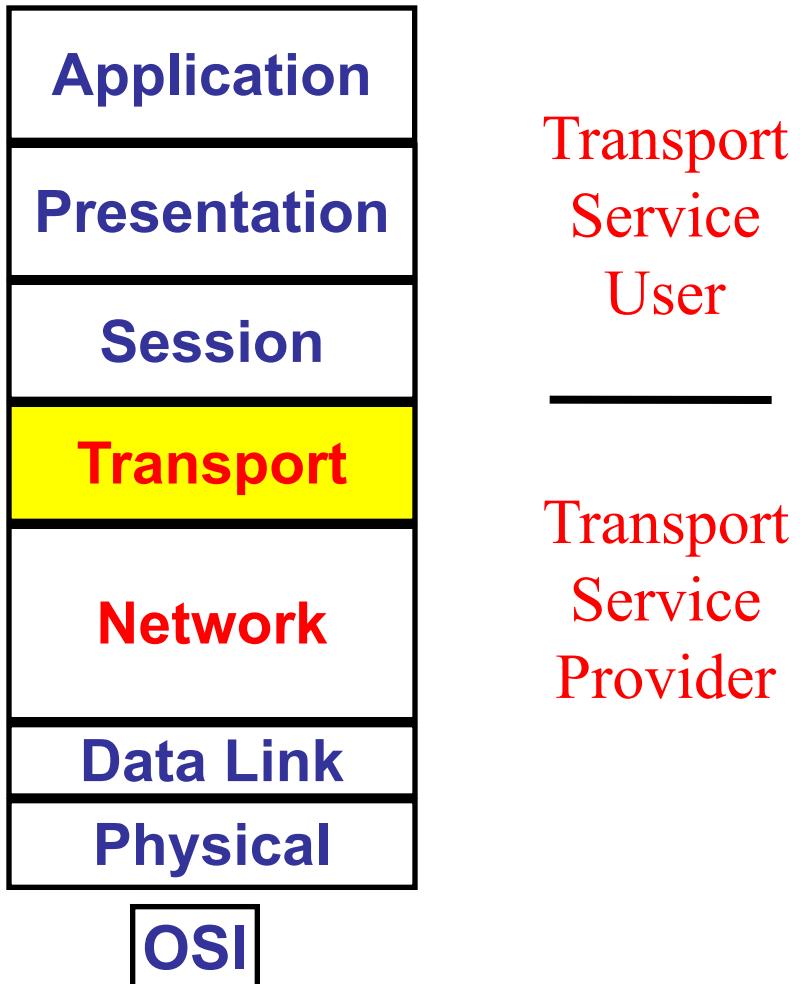


# Nota di Copyright

Quest'insieme di trasparenze è stato realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovrà essere esplicitamente riportata la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

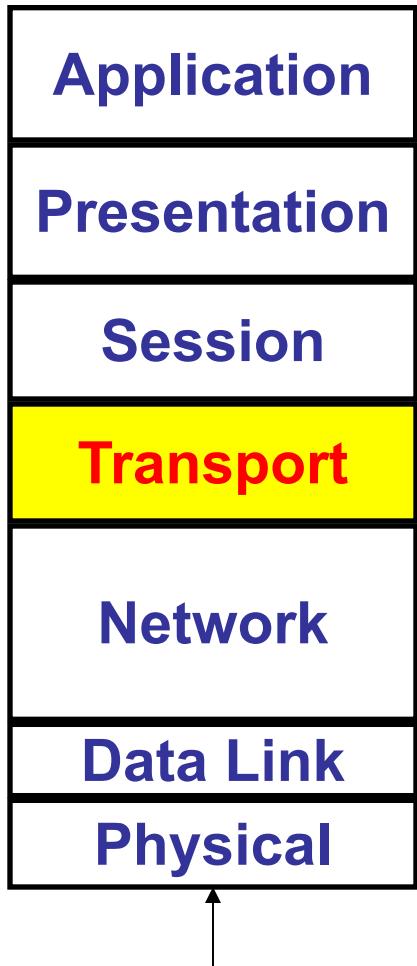


# Livello Trasporto

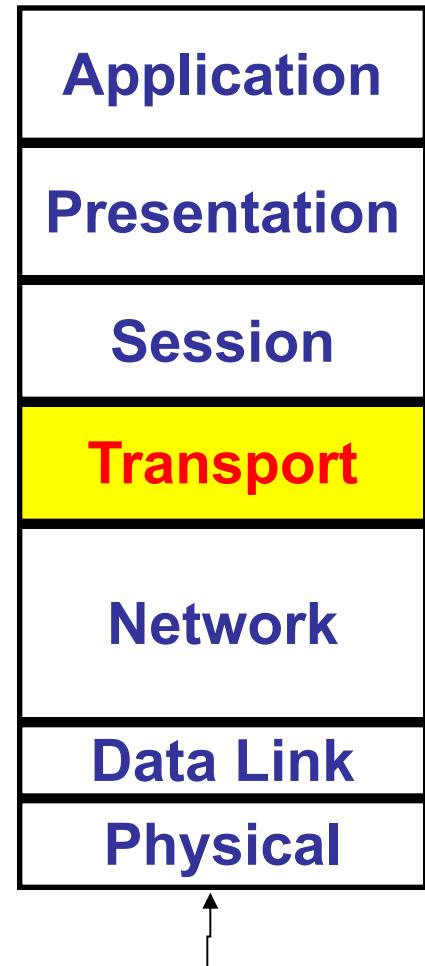
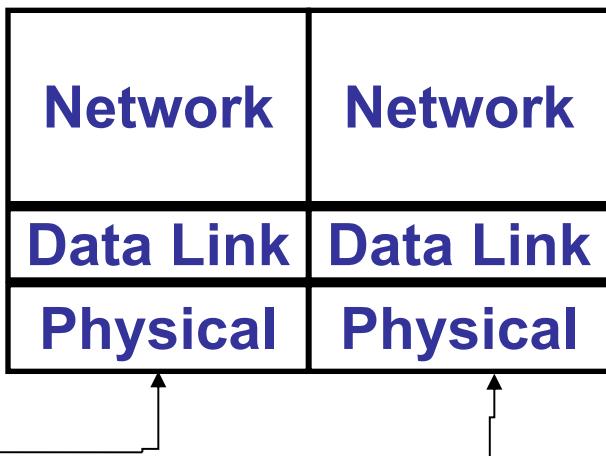




# Livello Trasporto



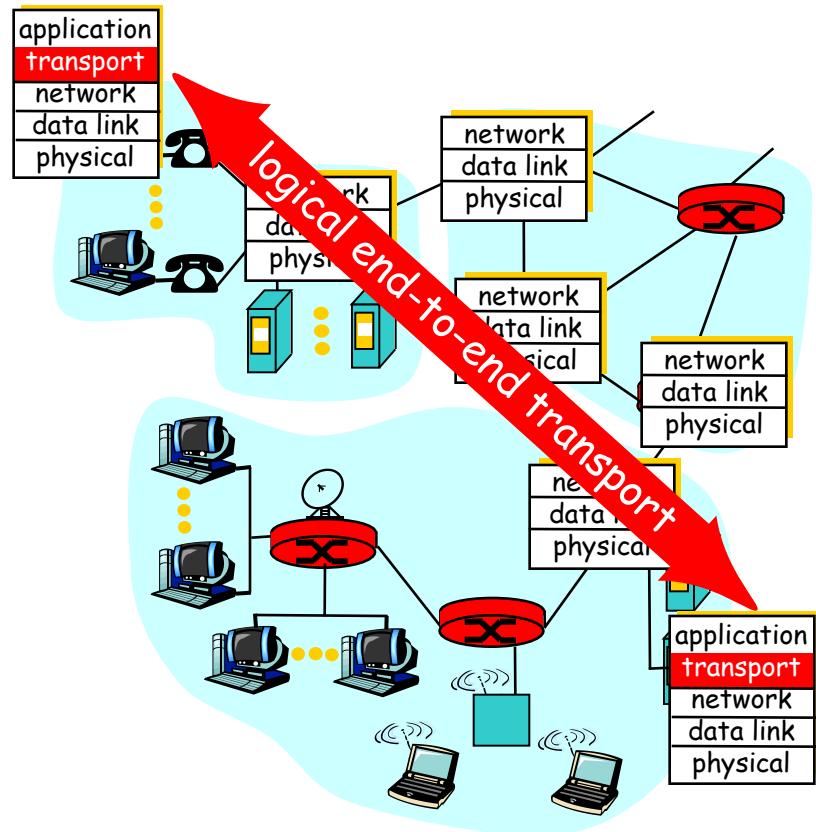
I protocolli di Livello  
Trasporto sono presenti  
**solo negli end-system**





# Servizi e Protocolli del Livello Trasporto

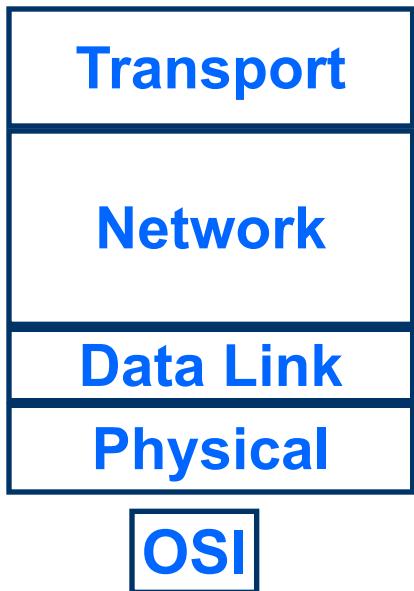
- Offre un canale di **comunicazione logica** tra applicazioni attive su differenti host
- Differenze tra livello trasporto e livello rete
  - **Network-layer** trasferimento dati tra end-system
  - **Transport layer** trasferimento dati tra processi. Naturalmente necessita dei servizi offerti dal livello rete



Da trasporto da *host a host* a  
trasporto da *processo a processo*



# Servizi del Livello Trasporto - 1



- affidabile
- inaffidabile
- affidabile
- inaffidabile

Isolare i livelli superiori dai problemi dovuti all'uso di differenti tecnologie di rete e dalle loro (eventuali) imperfezioni

Il livello rete offre un servizio inaffidabile, quindi

Il livello trasporto deve rimediare

- aumentare l'efficienza
- aumentare l'affidabilità

In particolare

- controllo degli errori
- sequenza ordinata
- controllo di flusso
- controllo di congestione



# Servizi del Livello Trasporto - 2

- I protocolli di Livello Trasporto sono realizzati al di sopra del Livello Rete, quindi è necessario gestire
  - eventuale apertura della connessione (setup)
  - un numero elevato di connessioni ...
    - Multiplexing e Demultiplexing

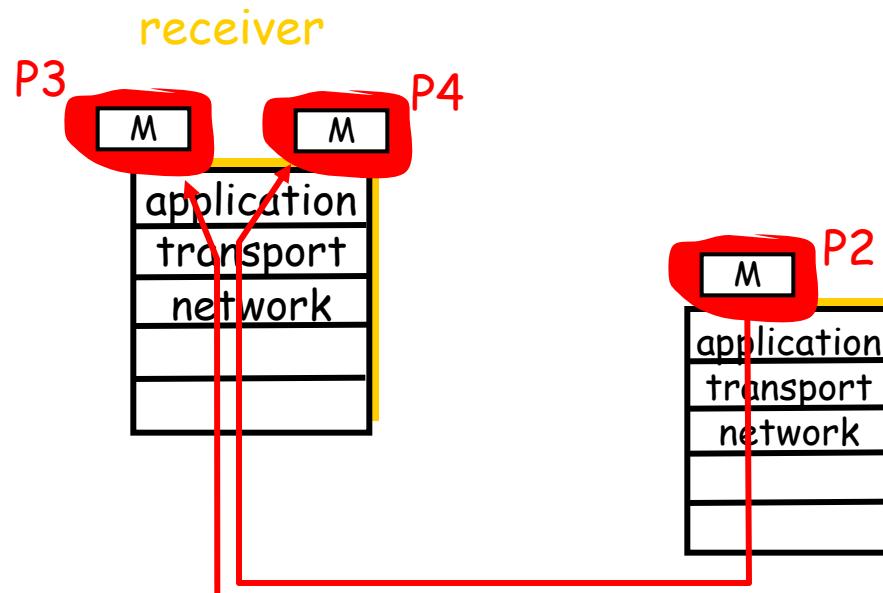
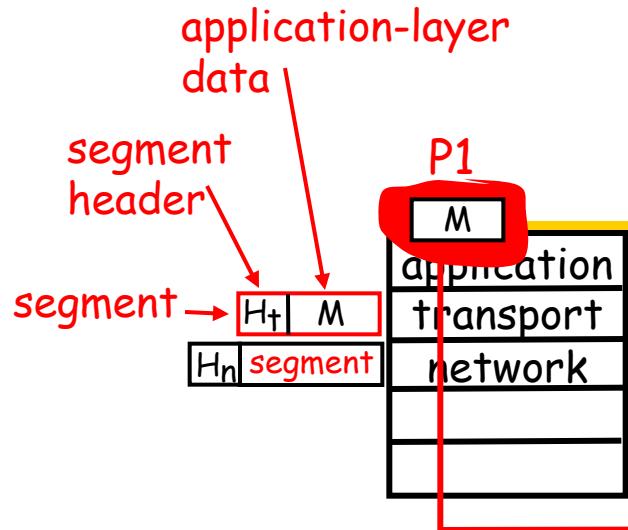


# Multiplexing e Demultiplexing - 1

*segment* : dati che sono scambiati tra processi a livello trasporto

TPDU: transport protocol data unit

**Demultiplexing:** inoltrare i segmenti ricevuti al corretto processo cui i dati sono destinati





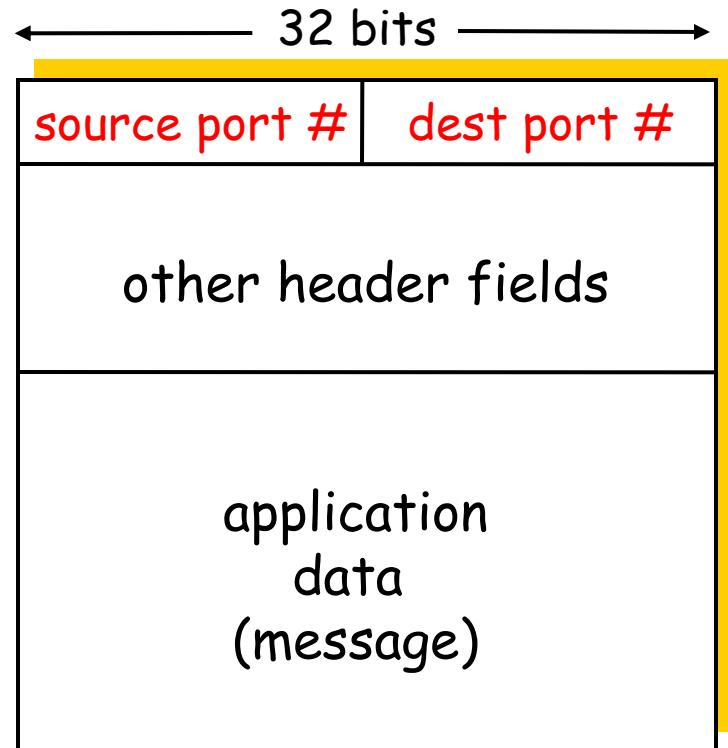
# Multiplexing e Demultiplexing - 2

## Multiplexing

Raccogliere i dati provenienti dalle applicazioni, imbustare i dati con un header appropriato (per il de-multiplexing)

## multiplexing/demultiplexing

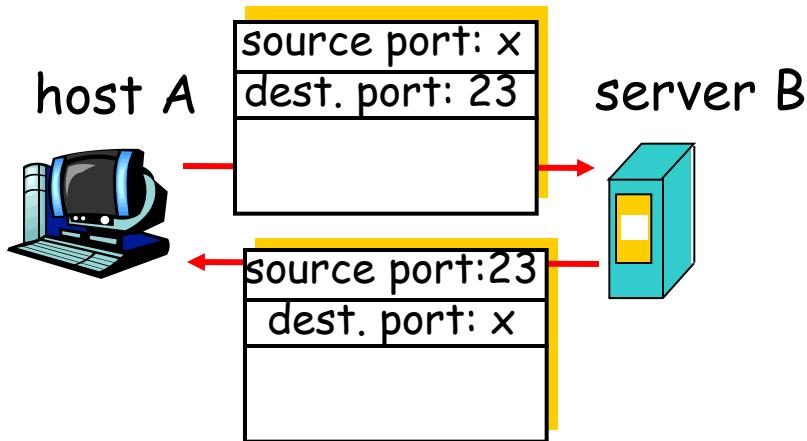
- Realizzato attraverso la coppia **<indirizzo IP, numero di porto>**
  - source, dest port # è presente in ogni segmento
  - numeri di porto “**well-known**” per applicazioni particolari



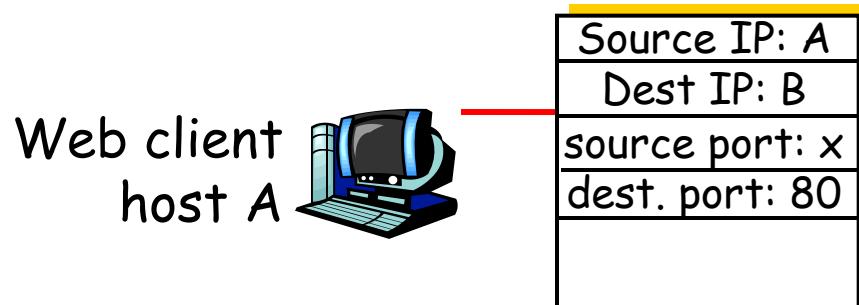
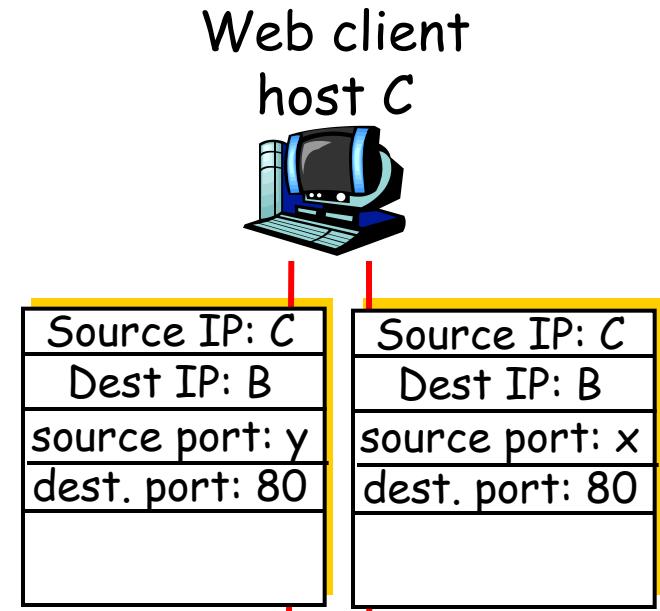
TCP/UDP segment format



# Multiplexing e Demultiplexing: esempi



Uso del concetto di porto:  
una semplice app telnet

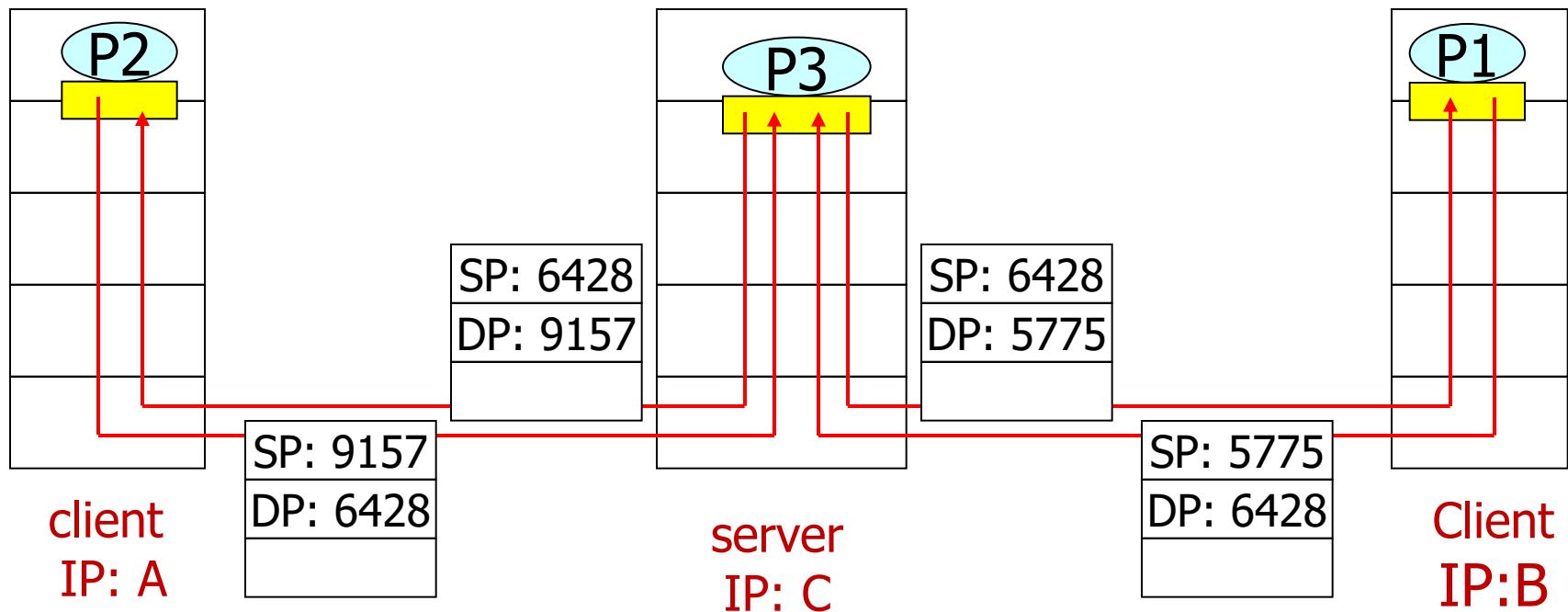


Web  
server B  
Caso del Web server



# Connectionless demux

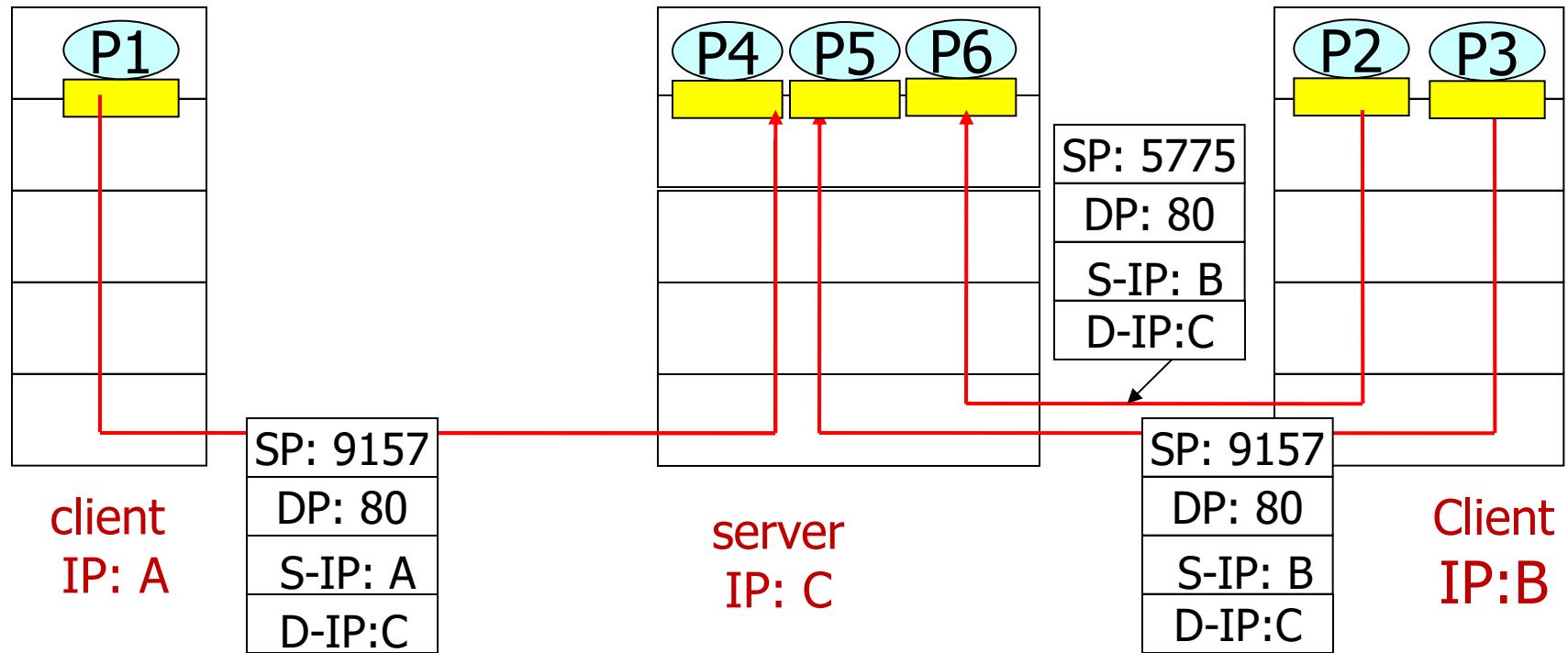
Su P3 c'è un processo in ascolto sul porto UDP 6428



Il SP fornisce il "return address"



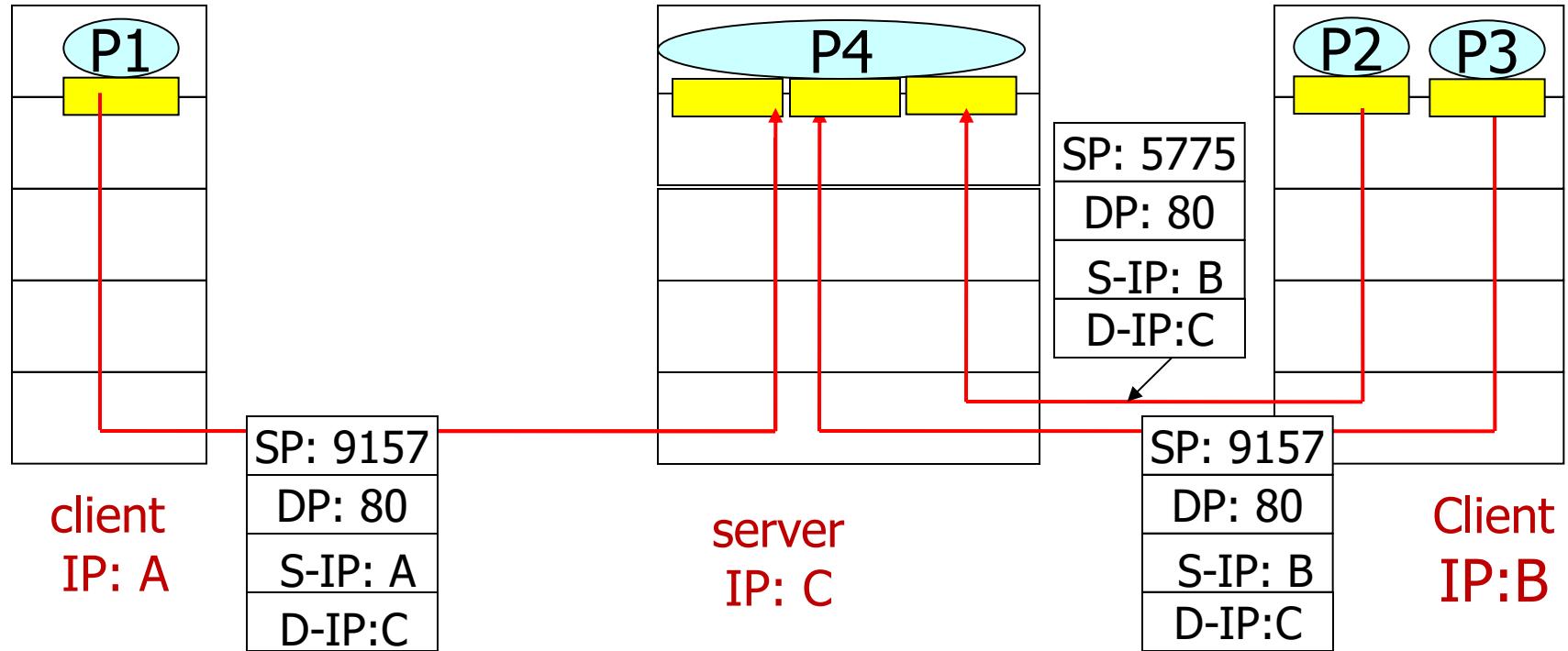
# Connection-oriented demux 1/2



Osservazione su HTTP persistente e non persistente...



# Connection-oriented demux: Threaded Web Server

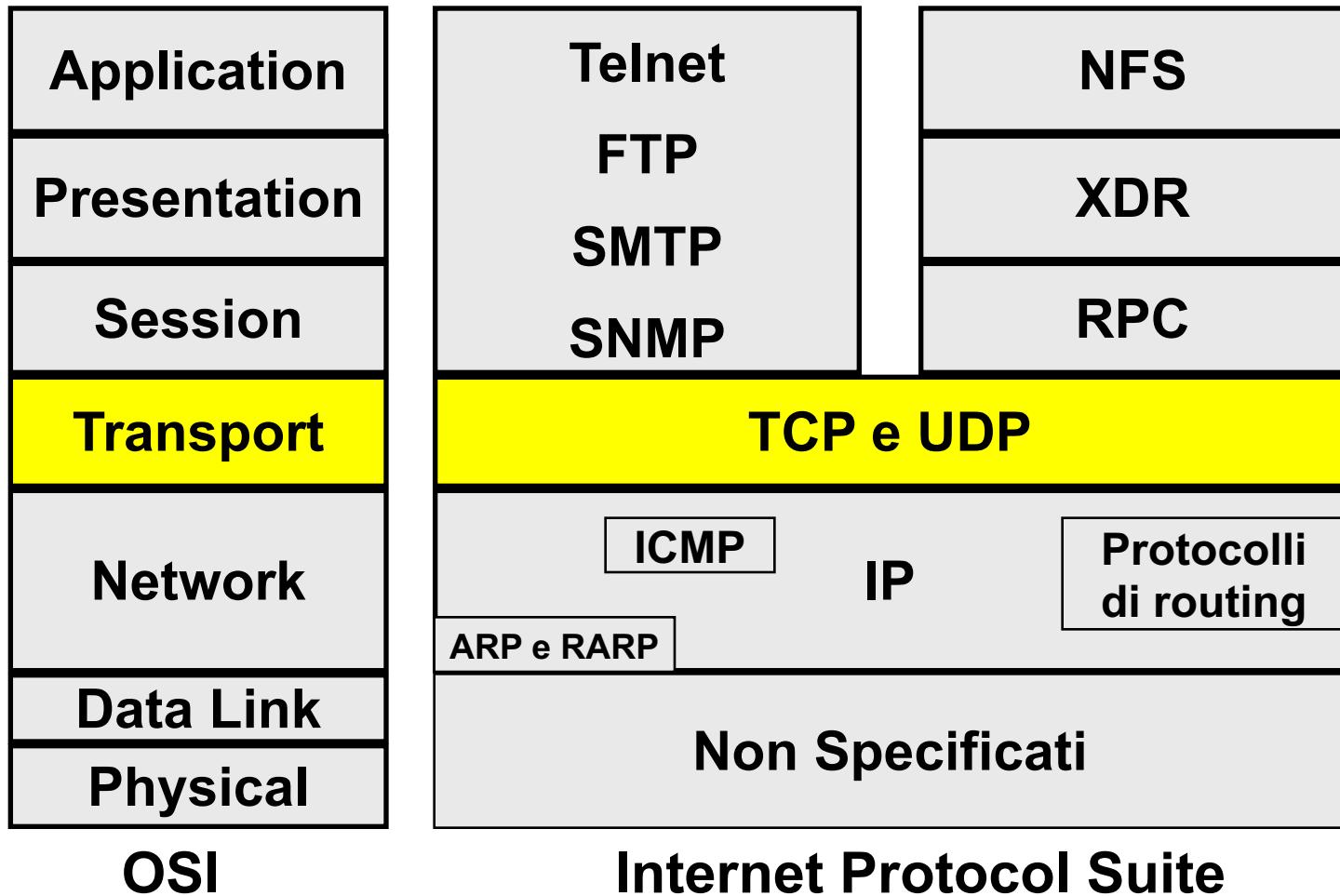




# I protocolli TCP e UDP

Transport  
Service  
User

Transport  
Service  
Provider





# UDP: User Datagram Protocol [RFC 768]

- Aggiunge poco ad IP
  - servizio “best effort”
    - i pacchetti UDP possono
      - subire perdite
      - giungere a destinazione in ritardo, o non arrivare affatto
      - giungere a destinazione non ordinati
    - *servizio connectionless*
      - non è prevista una fase di inizializzazione
      - ogni segmento UDP è inviato indipendentemente dagli altri
        - *Domanda*: C’è qualcuno *in ascolto*?



# UDP: User Datagram Protocol - 2

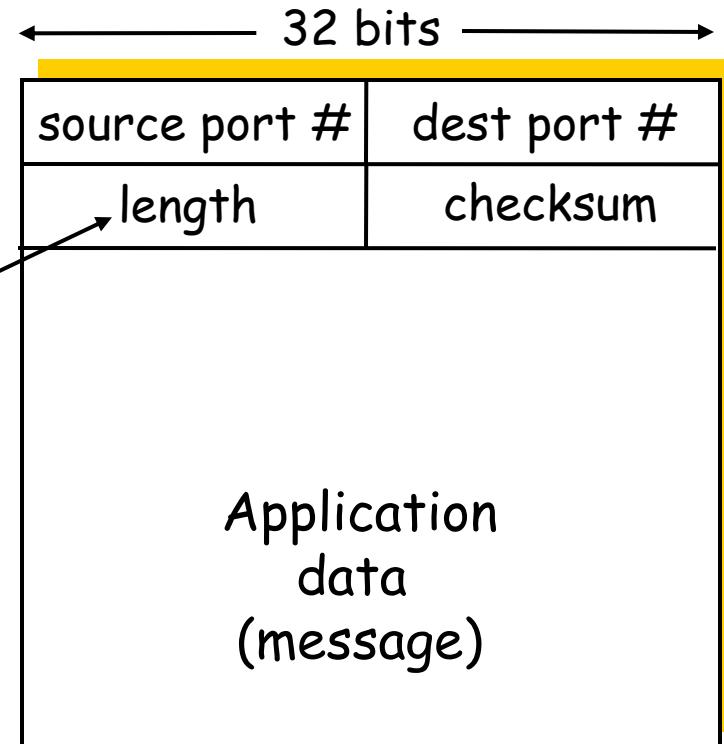
- Perché è stato introdotto UDP?
  - non è necessaria la fase di inizializzazione (setup) che introduce delay
    - Es: DNS è basato su UDP
  - semplice: sender e receiver non devono conservare informazioni di stato
  - intestazione di dimensioni contenute (8 byte)
    - basso overhead
  - controllo della congestione assente
    - le applicazioni possono inviare alla velocità desiderata
      - utile per alcune applicazioni
        - » sia utente che di rete
      - rischioso per la rete



# UDP: User Datagram Protocol - 3

- Ampiamente usato per applicazioni multimediali
  - tolleranti alle perdite di pacchetti
  - sensibili ai ritardi
- Altre applicazioni
  - DNS
  - NFS (Network File System)
  - SNMP (Simple Network Management Protocol)
- **Domanda:** si può rendere UDP affidabile?
  - Gestione degli errori
  - Conferma di avvenuta ricezione

Lunghezza  
in bytes del  
segmento UDP,  
header incluso



Formato di un segmento UDP



# Checksum UDP

Obiettivo: rilevare gli “errori” (bit alterati) nel segmento trasmesso

## Mittente

- Tratta il contenuto del segmento come una sequenza di parole da 16 bit
- **checksum:** complemento ad 1 della somma dei contenuti del segmento
- Il mittente pone il valore della checksum nel campo checksum del segmento UDP

## Ricevente

- calcola la checksum del segmento ricevuto
- controlla se la checksum calcolata è uguale al valore del campo checksum
  - No - errore rilevato
  - Sì - nessun errore rilevato. *Ma potrebbero esserci errori nonostante questo? Altro più avanti ...*



# Esempio di calcolo della checksum

- Example: add two 16-bit integers

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

wraparound

1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

sum	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

- Note
  - When adding numbers, a carryout from the most significant bit needs to be added to the result



**Corso di Laurea in Informatica**

**Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

---

**Il livello trasporto:**

**Tecniche di trasmissione affidabile dei dati**



# Nota di Copyright

Quest'insieme di trasparenze è stato realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovrà essere esplicitamente riportata la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.



# Realizzare una trasmissione affidabile

- Se il livello rete è inaffidabile
  - Presenza di errori
  - Perdita di pacchetti
  - Ordine dei pacchetti non garantito
  - Duplicazione di pacchetti
  - Inoltre bisogna tenere in considerazione
    - Le risorse del computer ricevente
      - Controllo di flusso
    - Le risorse della rete
      - Controllo di congestione

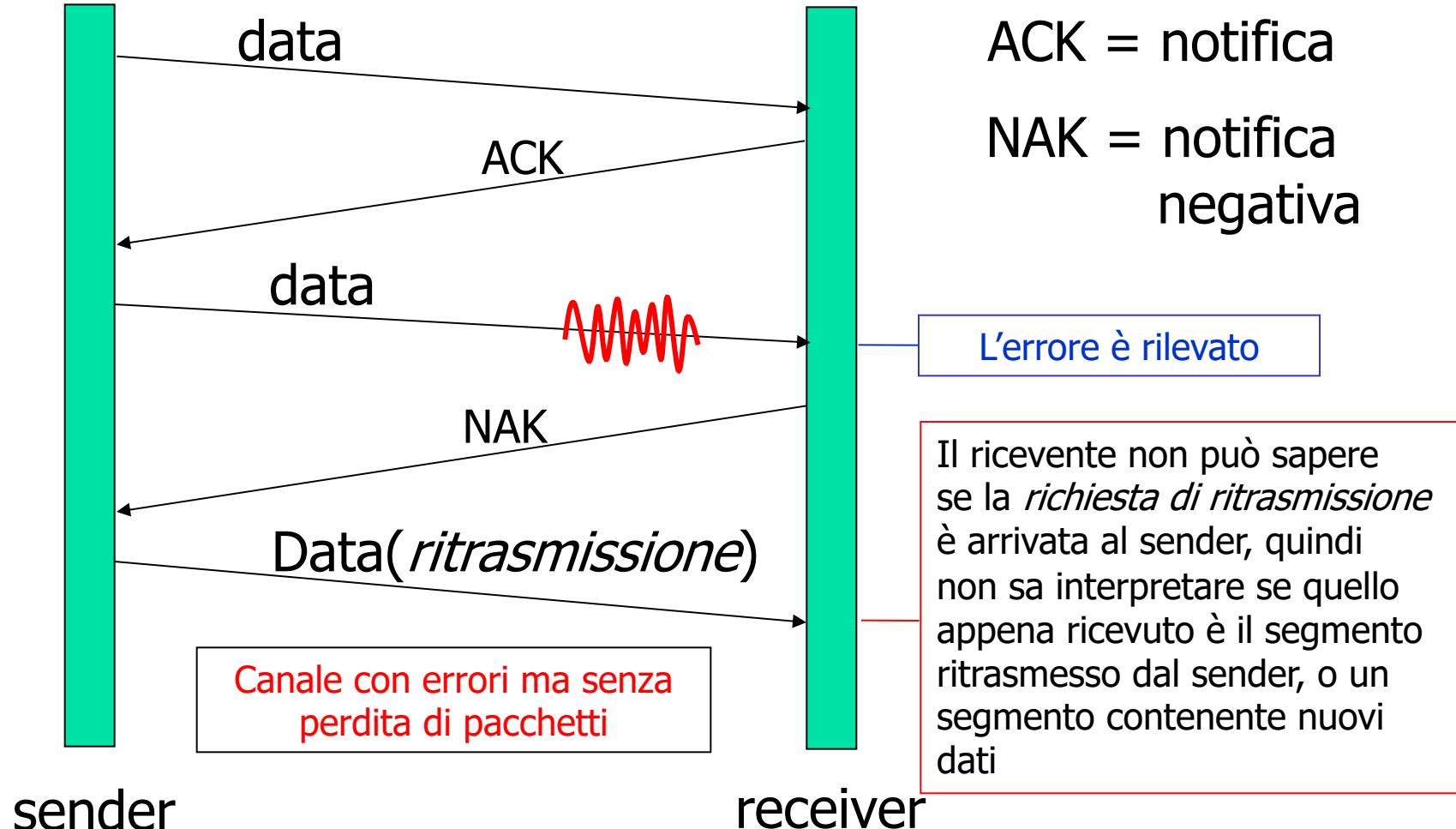


# Realizzare una trasmissione affidabile - 2

- Soluzioni
- Rete che presenta errori di trasmissione
  - In questo caso il ricevente deve effettuare
  - Rilevamento degli errori e:
    - 1) Correzione degli errori  
oppure
    - 2) Notifica al mittente  
Richiesta ritrasmissione
  - La prima soluzione introduce complicazioni, la seconda introduce possibili duplicazioni sulla rete che il ricevente non è in grado di interpretare



# Realizzare una trasmissione affidabile - 3



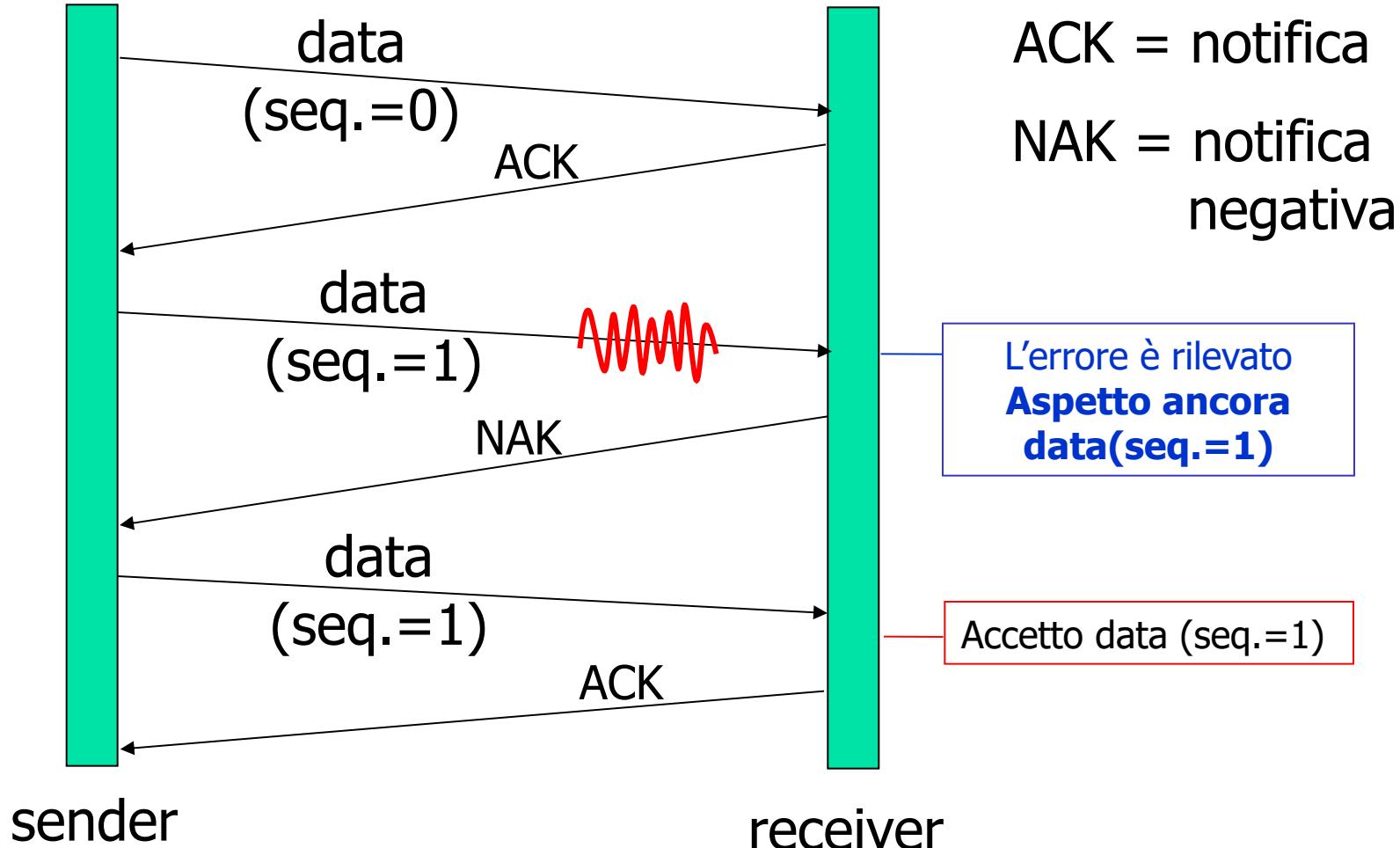


# Realizzare una trasmissione affidabile - 4

- Per risolvere il problema dei duplicati che il ricevente non è in grado di interpretare, occorre inserire nell'header del segmento da inviare un'ulteriore informazione
  - **numero di sequenza**
- Nel caso di protocolli che inviano un messaggio e quindi aspettano un riscontro prima di ritrasmettere un nuovo messaggio (**stop & wait**), è sufficiente un numero di sequenza su un bit (0,1). Vediamo un esempio



# Realizzare una trasmissione affidabile - 5



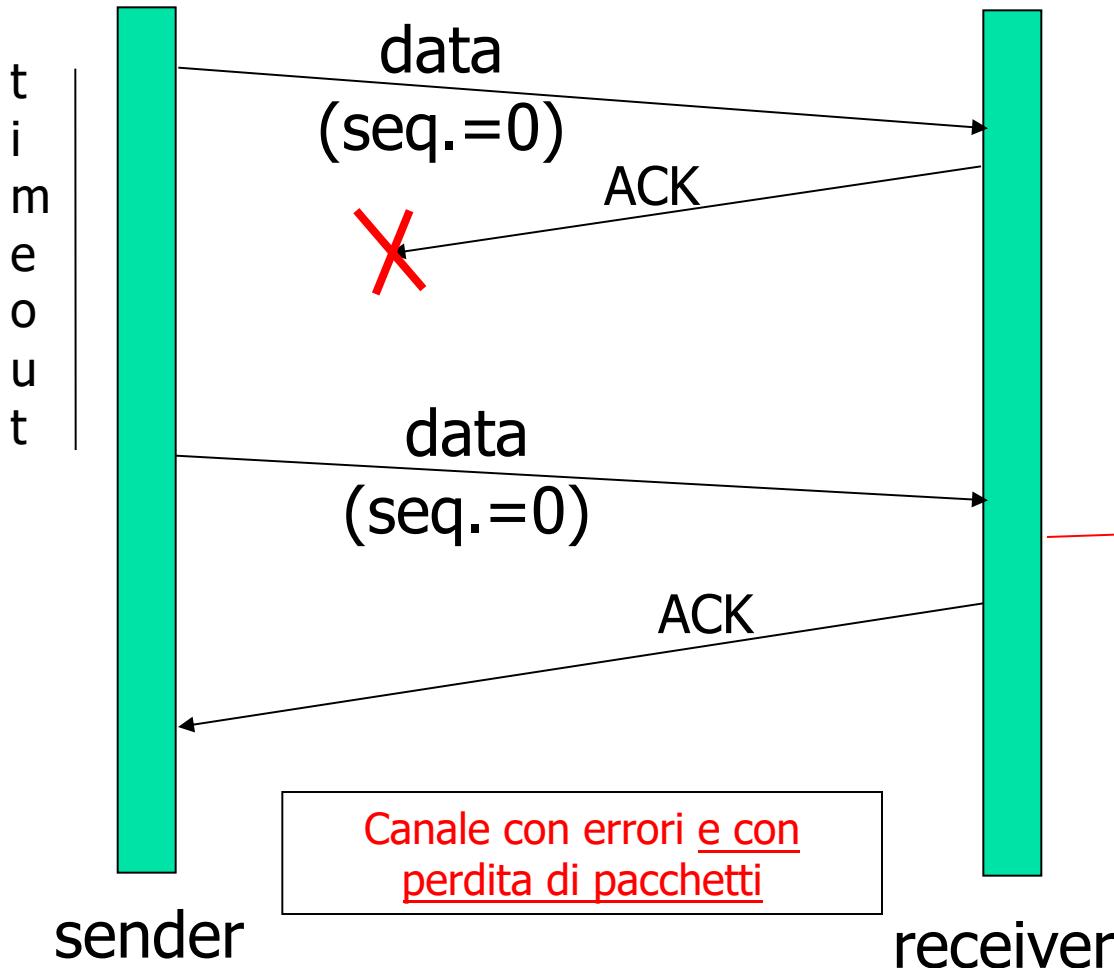


# Un protocollo senza NAK

- Stessa funzionalità del precedente, utilizzando soltanto gli ACK
- Al posto del NAK, il destinatario invia un ACK per l'ultimo pacchetto ricevuto correttamente
  - il destinatario deve includere *esplicitamente* il numero di sequenza del pacchetto con l'ACK
- Un ACK non ricevuto dal mittente determina la stessa azione del NAK: *ritrasmettere il pacchetto corrente*
- Sull'esempio di prima...



# Realizzare una trasmissione affidabile - 6



Nel caso di canale che introduce perdita di pacchetti, è necessario introdurre un altro parametro: il tempo

In particolare un conto alla rovescia: timeout

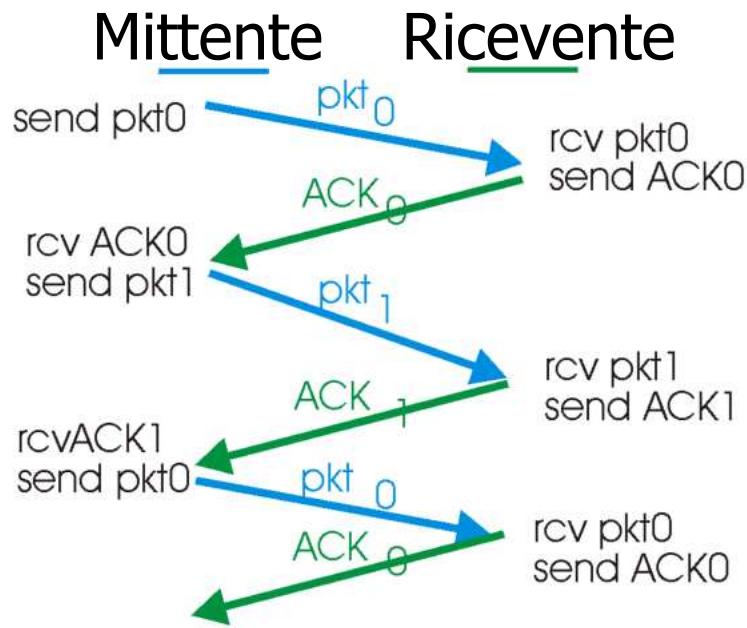
Si può fare a meno dei NAK

Il receiver si accorge di aver già ricevuto data (seq.=0), scarta tale segmento e invia nuovamente l'ACK al mittente

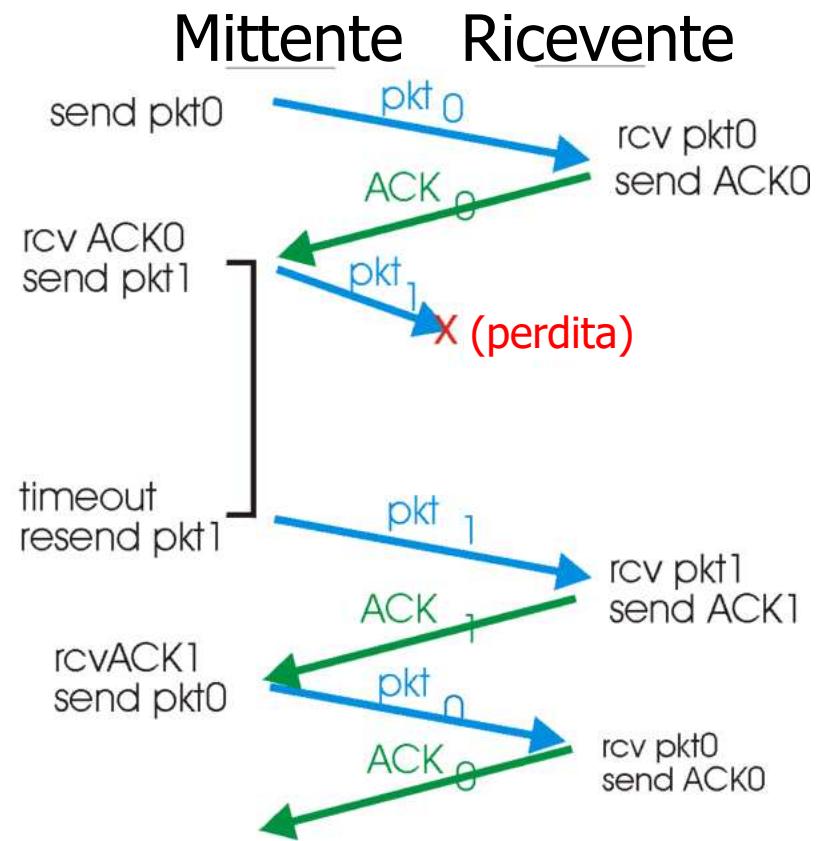
**Problema:** quanto dovrà essere grande il timeout?



# Stop&Wait: Ricapitolando (1/2)



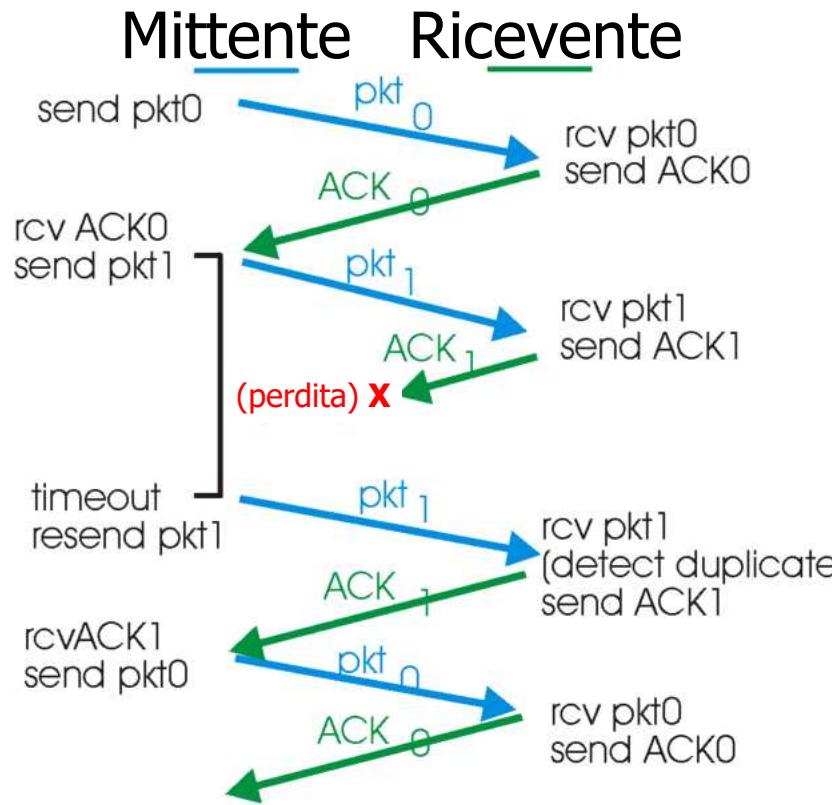
a) Operazioni senza perdite



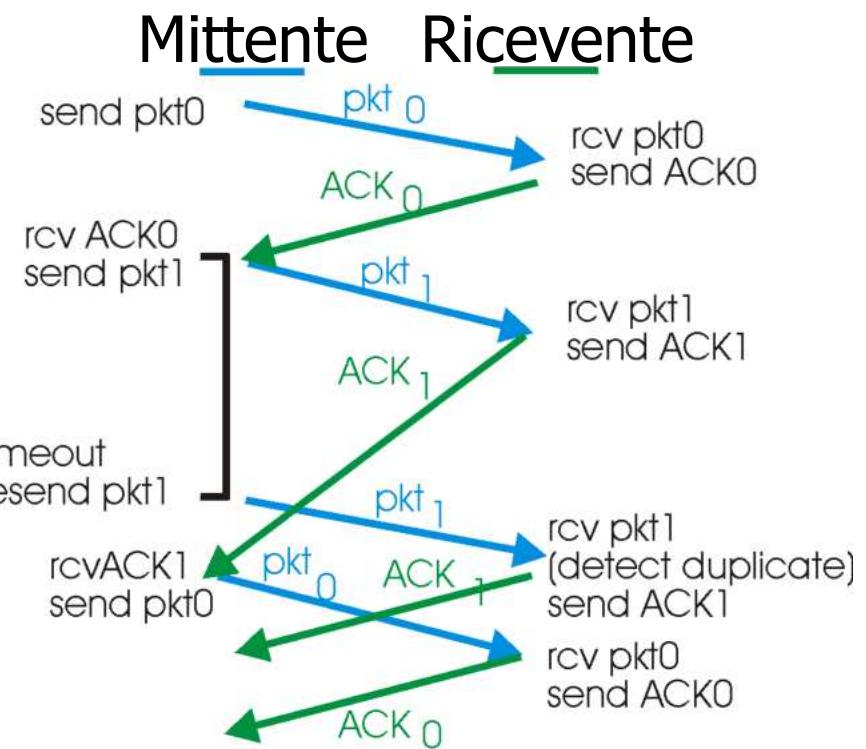
b) Perdita di pacchetto



# Stop&Wait: Ricapitolando (2/2)



c) Perdita di ACK



d) Timeout prematuro

... anche detto "protocollo ad alternanza di bit"

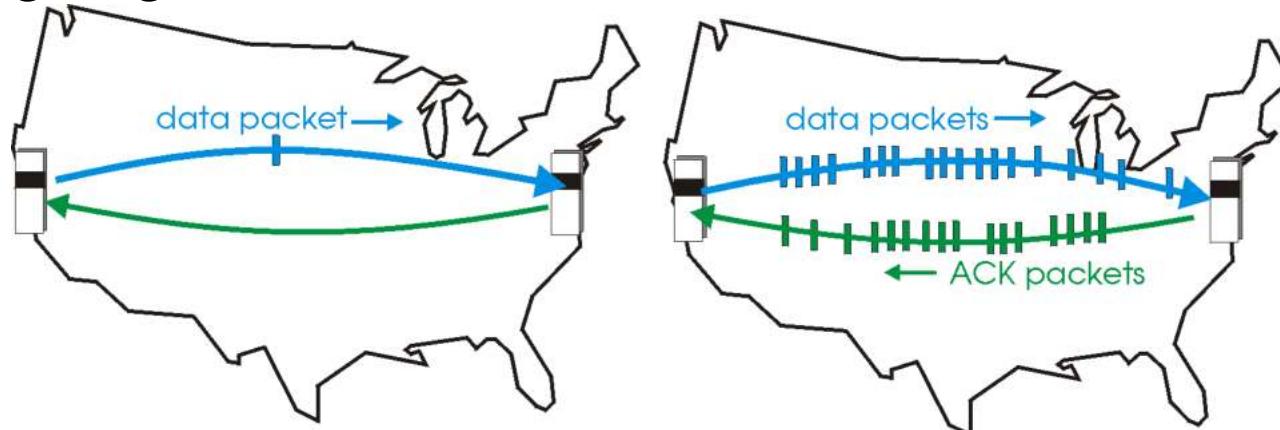


# Aumentare l'efficienza

In alternativa al semplice Stop & Wait...

**Pipelining:** il mittente invia pacchetti prima di ricevere il riscontro dei precedenti

- Occorre aumentare l'intervallo dei num. sequenza
- Aggiungere buffer nel sender e/o receiver



(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

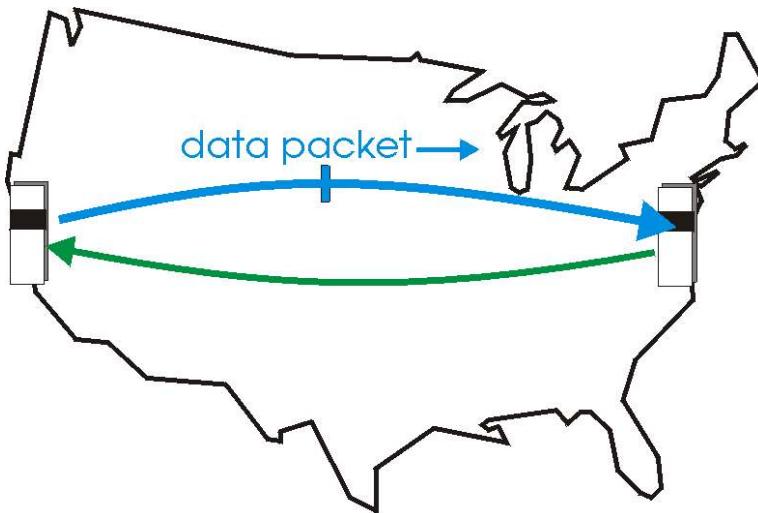
- Due alternative per il pipelining: *go-Back-N*, *selective repeat*



# Performance di Stop&Wait

- Funziona, ma le performance...
- Esempio: 1 Gbps link, 1KB packet,  
15 ms end-to-end propagation delay:

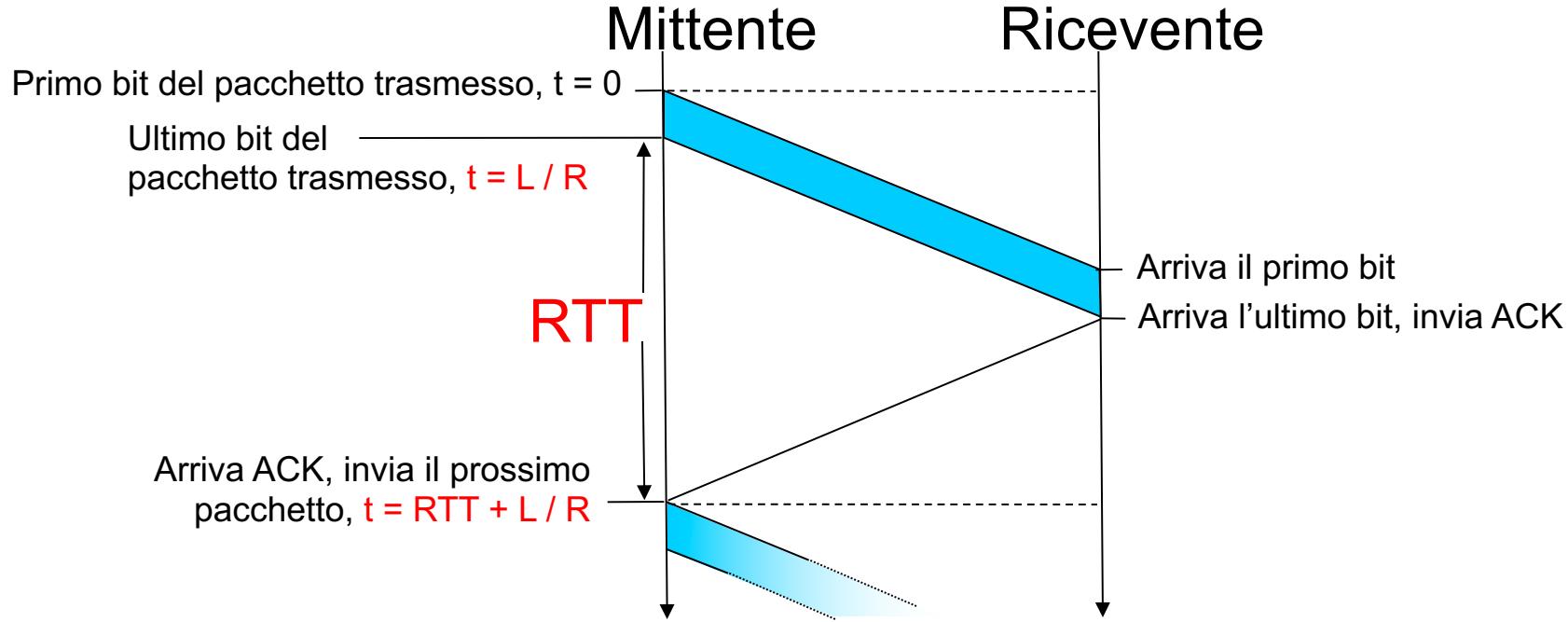
$$T_{\text{transmit}} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate in bps)}} = \frac{8 \text{ kbits}}{10^9 \text{ bps}} = 8 \text{ microsec}$$



(a) a stop-and-wait protocol in operation



# Funzionamento con stop-and-wait

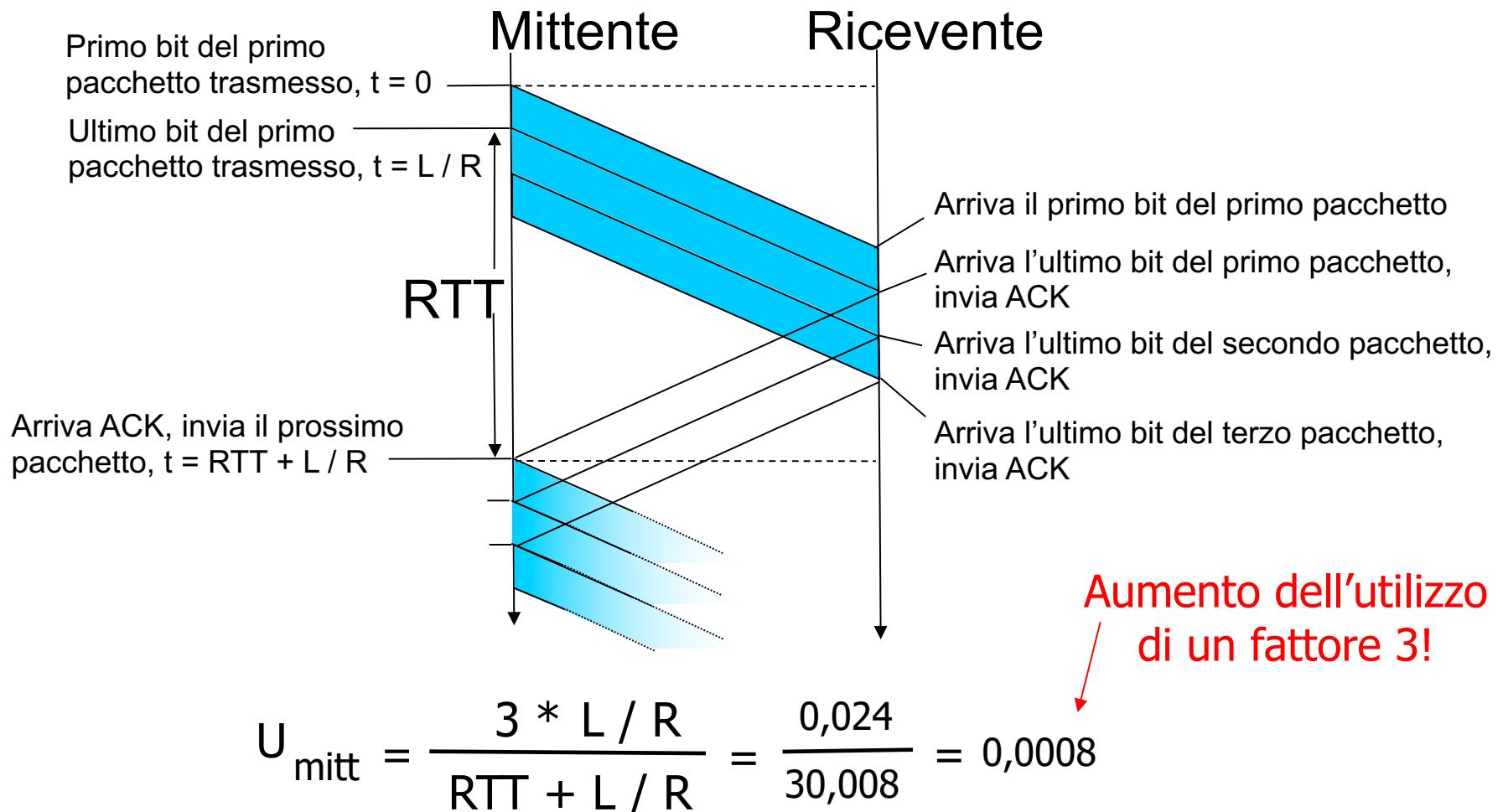


$$U_{\text{mitt}} = \frac{L / R}{RTT + L / R} = \frac{0,008}{30,008} = 0,00027$$

$$R_{\text{eff}} = \frac{1000 \text{ byte}}{30,008 \text{ ms}} = 267 \text{ kbps}$$



# Pipelining: aumento dell'utilizzo

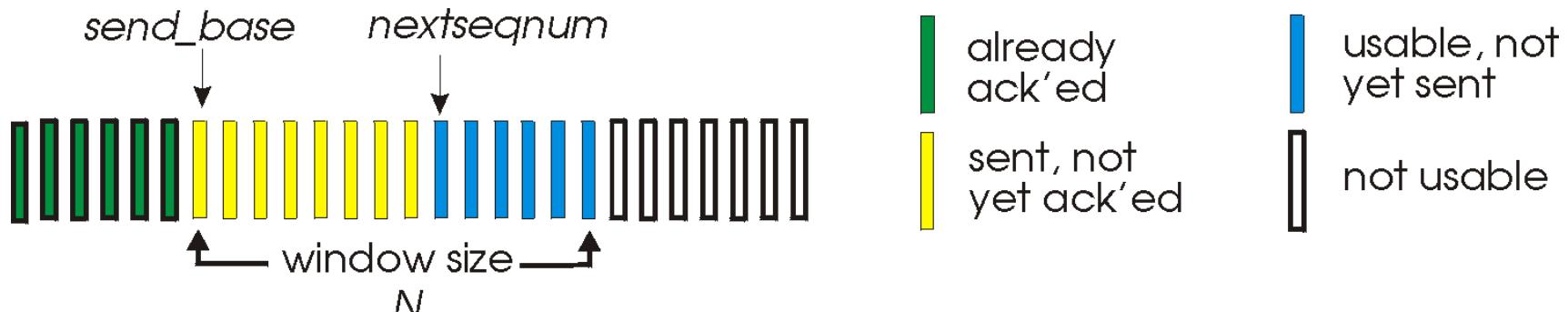




# Go Back-N

## Sender

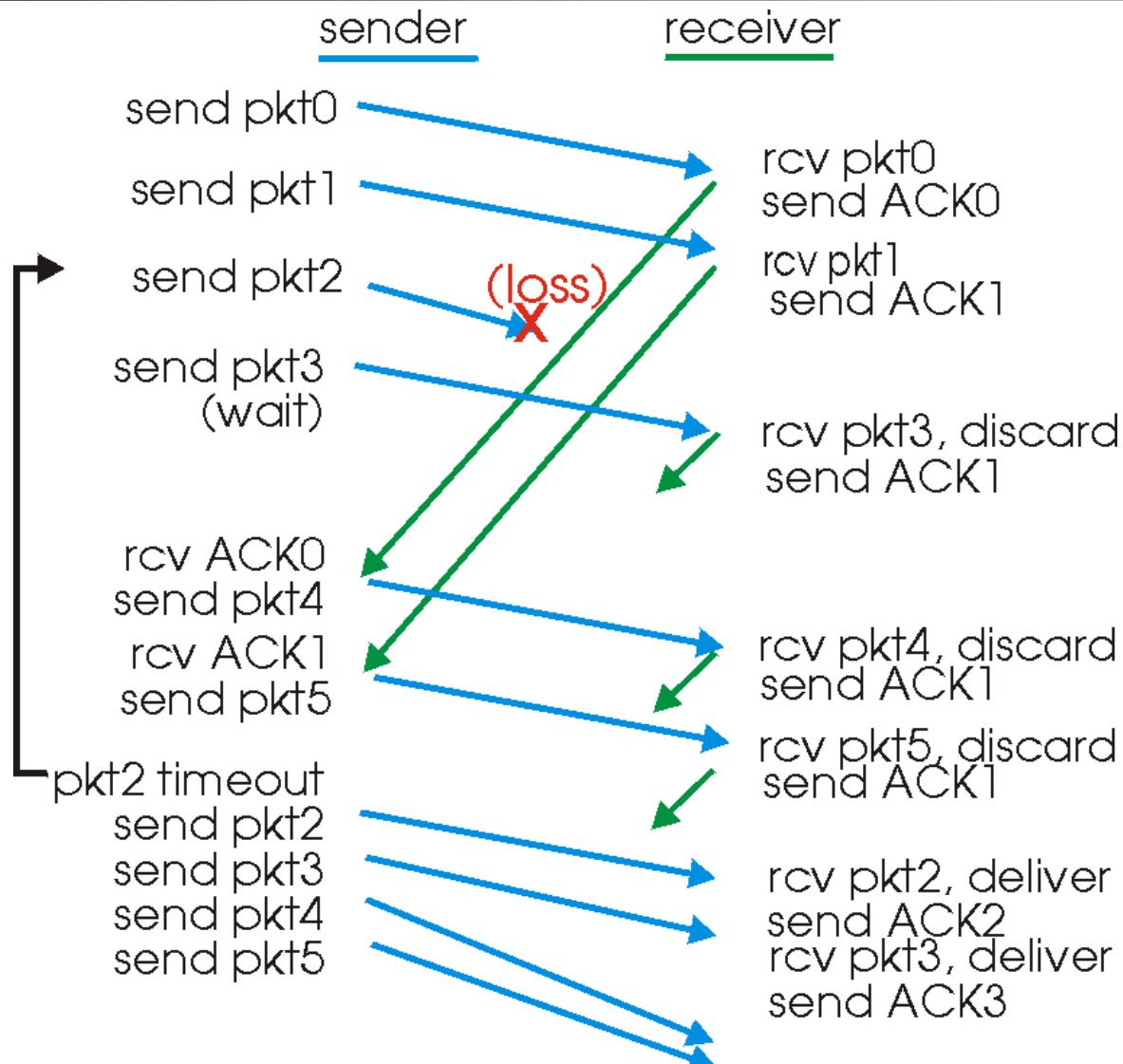
- Nell'header del segmento k-bit per il num. sequenza
- Una finestra di max N pacchetti senza riscontro
- ACK numerati



- ACK cumulativo: ricevere ACK( $n$ ) significa che tutti i pkts precedenti l' $n$ -esimo sono stati ricevuti correttamente
- Un timer per il primo pacchetto trasmesso e non riscontrato
- $timeout(n)$ : ritrasmetti pkt  $n$  e tutti i pacchetti che seguono  $n$
- Considerazione sulla dimensione della finestra ( $N$ )



# Go Back-N in azione



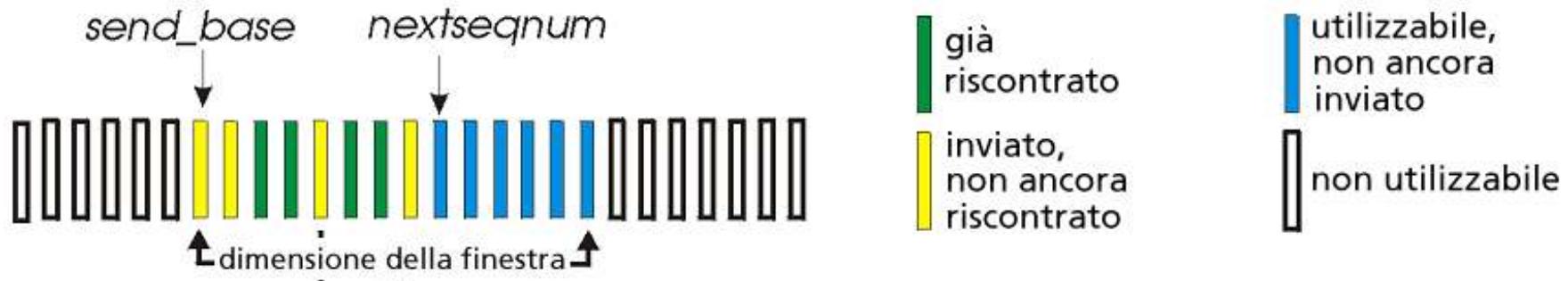


# Ripetizione selettiva

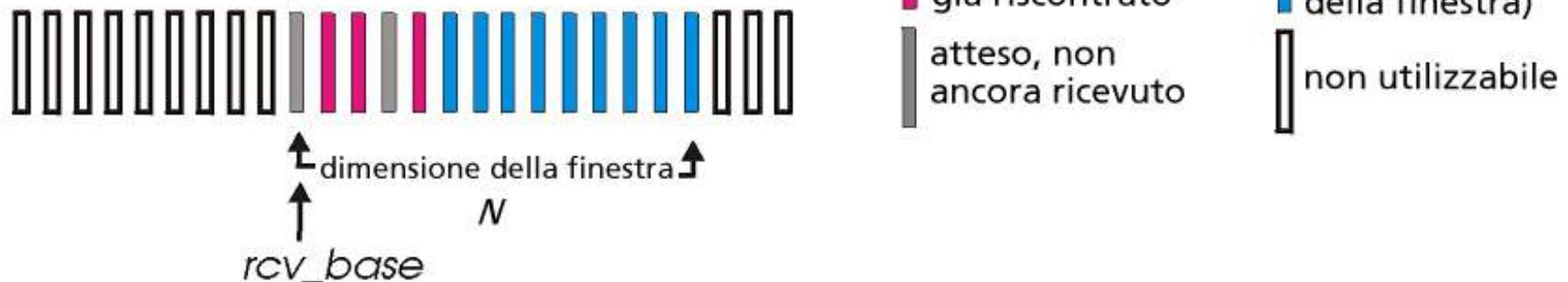
- In Go Back-N un errore su un solo pacchetto può provocare la ritrasmissione di un elevato numero di pacchetti
- Il ricevente invia riscontri *specifici* per tutti i pacchetti ricevuti correttamente
  - Bufferizza i pacchetti per eventuali consegne ***in sequenza*** al livello superiore
- Il mittente ritrasmette soltanto i pacchetti per i quali non ha ricevuto un ACK
  - Timer del mittente per ogni pacchetto non riscontrato
- Finestra del mittente
  - N numeri di sequenza consecutivi
  - Limita ancora i numeri di sequenza dei pacchetti inviati non riscontrati



# Selective Repeat



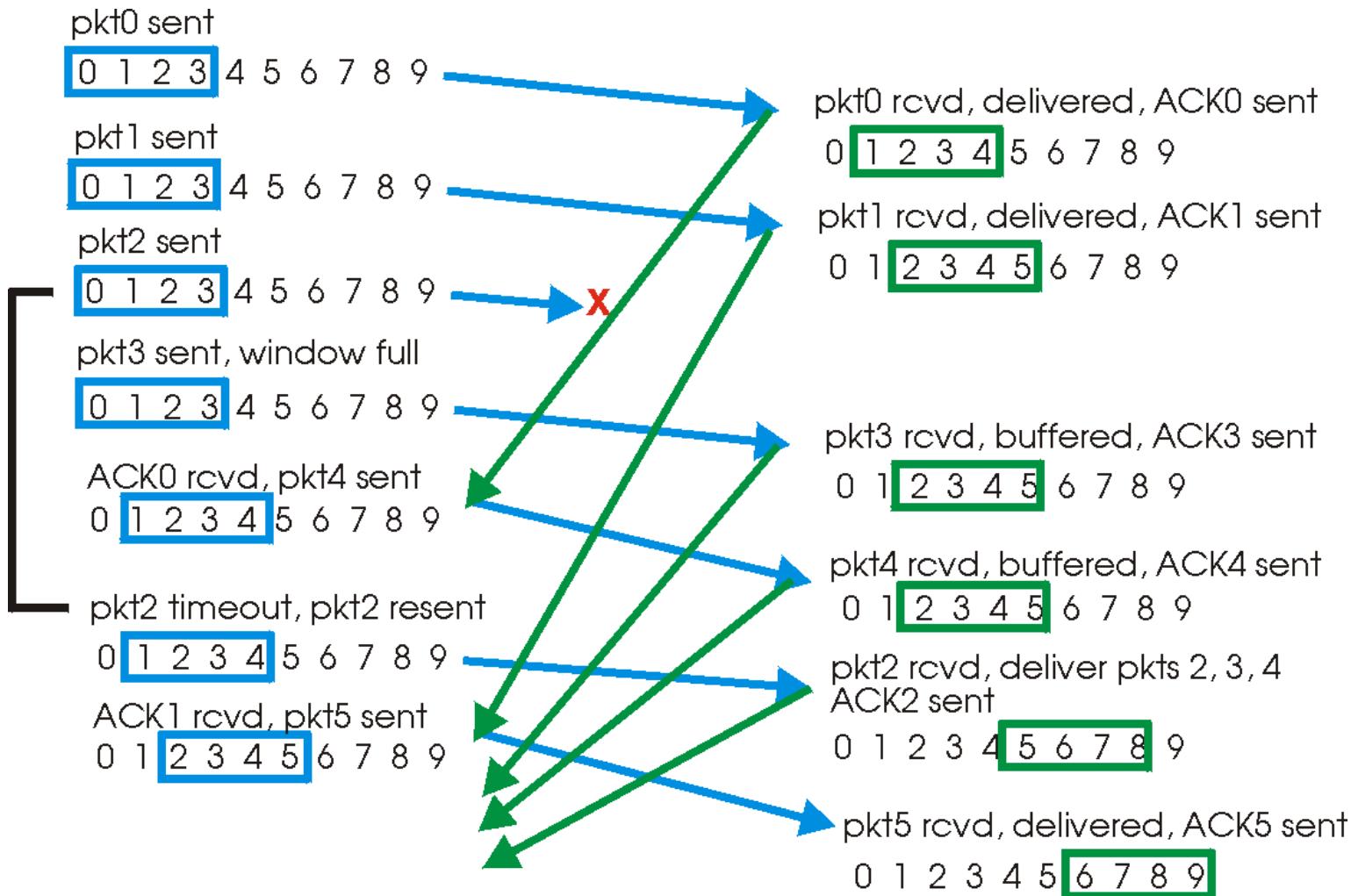
a) Visione del mittente sui numeri di sequenza



b) Visione del ricevente sui numeri di sequenza



# Selective Repeat in azione

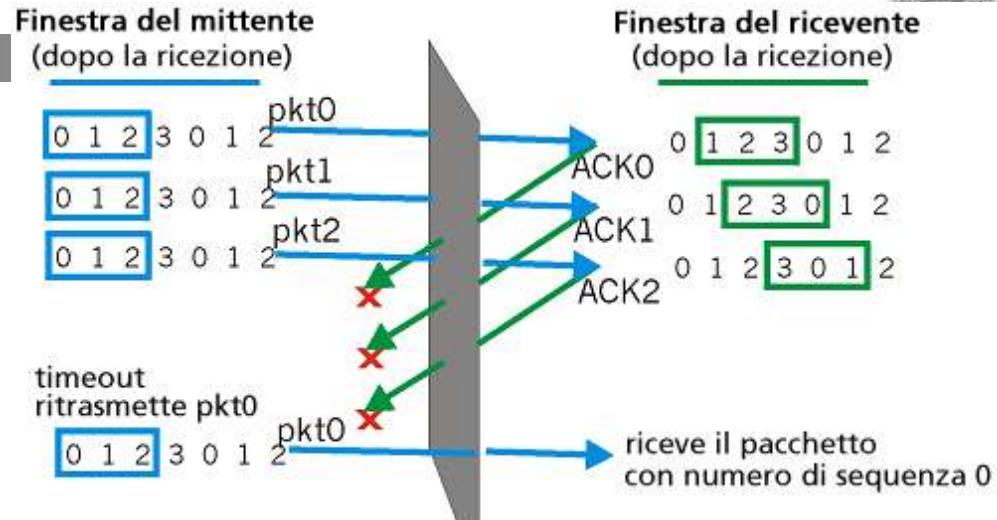




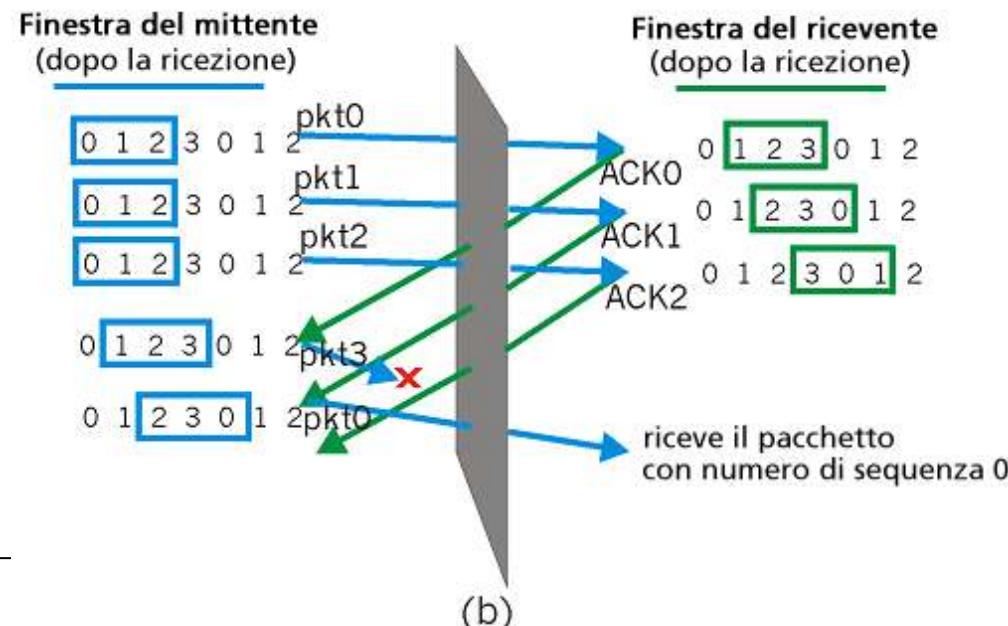
# Ripetizione selettiva: dilemma

## Esempio

- N=Numeri di sequenza: 0, 1, 2, 3
- D= Dimensione della finestra: 3
- Il ricevente non vede alcuna differenza fra i due scenari!
- Passa erroneamente i dati duplicati come nuovi in (a)



(a)



(b)

D: Qual è la relazione fra lo spazio dei numeri di sequenza e la dimensione della finestra?

$$D \leq N/2$$



**Corso di Laurea in Informatica**

**Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

**Il livello trasporto:**

**Il protocollo TCP**



# Nota di Copyright

Quest'insieme di trasparenze è stato realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell' Università di Napoli. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell' uso dovrà essere esplicitamente riportata la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.



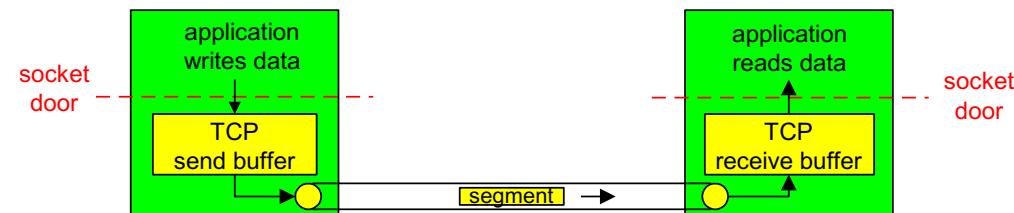
# Agenda

- TCP: caratteristiche
- Intestazione del pacchetto TCP
- Trasmissione affidabile
- Management della connessione
- Diagrammi degli stati



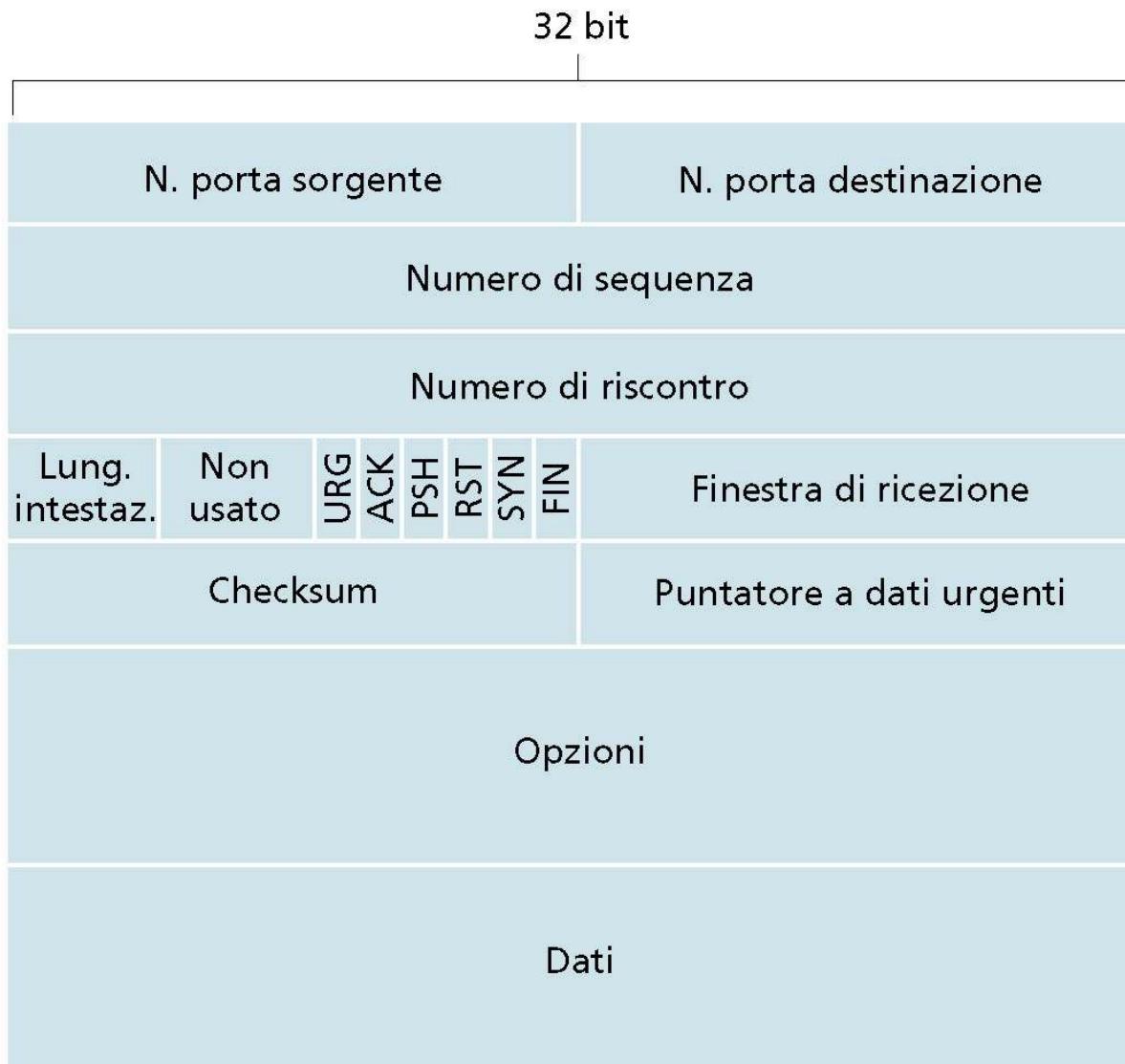
# TCP: Transmission Control Protocol

- End-to-end e punto-punto
  - Una connessione unica tra mittente e ricevente
- Senza errori, sequenza ordinata
- Pipelined
  - Controllo di flusso e di congestione impostano la TCP window
- Buffer su mittente e ricevente
  - Full duplex data
    - Flusso di dati bi-direzionale all'interno della stessa connessione
    - MSS: Maximum Segment Size
  - Connection-oriented
    - handshaking (scambio di msg di controllo a tre vie) prepara mittente e ricevente prima della comunicazione
  - Controllo di flusso
    - Il mittente non invia più di quanto il ricevente non possa accettare





# Struttura del segmento TCP





# TCP: PDU

- **HLEN**
  - 4 bit, contiene un numero intero che indica la lunghezza dell'intestazione TCP del datagramma in parole da 32 bit. Questa informazione è necessaria perché il campo **opzioni** è di lunghezza variabile. Quando il campo opzioni è vuoto, la lunghezza dell'intestazione è 20 byte.
- **Porta (provenienza/destinazione)**
  - Contengono i numeri di porta di protocollo TCP che identificano gli applicativi alle estremità della connessione (mux/demux).



# TCP: PDU

- **Numero di Sequenza**
  - Questo campo identifica, nello stream di byte del trasmittitore, la posizione dei dati nel segmento. Questo valore è riferito allo stream che fluisce nella medesima direzione del segmento, mentre il **Numero di Riscontro** si riferisce allo stream che fluisce nella direzione opposta.
- **Numero di riscontro**
  - Contiene il numero sequenziale del byte successivo a quello correttamente ricevuto dalla destinazione. Tale campo è valido solo nei segmenti di riscontro, o nei segmenti utilizzanti la tecnica trasmissiva **Piggy-backing**, e fa riferimento allo stream di dati che fluisce nella direzione opposta a tale segmento.
  - Nel calcolo del numero di riscontro, oltre a considerare i byte contenuti nel payload TCP, bisogna considerare anche la presenza di byte SYN e FIN inviati, che valgono come un singolo byte.



# TCP: PDU

- Bit di Codice
  - Per identificare il tipo di informazione contenuta nel segmento vengono impiegati i 6 bit di codice
    - ACK: Il campo riscontro è valido
    - RST: Effettua il reset della connessione
    - SYN: Sincronizza i numeri di sequenza
    - FIN: Il trasmettitore ha raggiunto la fine del suo stream di byte
    - PSH: Questo segmento richiede una “spinta” (a destinazione)
    - URG: Il campo puntatore urgente è valido



# TCP: PDU

- **Finestra**
  - Numero intero senza segno di 16 bit che specifica la dimensione del buffer che il TCP ha a disposizione per immagazzinare dati in arrivo. È utilizzato per la gestione dinamica della dimensione della finestra scorrevole.
- **Puntatore urgente**
  - Il TCP permette la trasmissione di dati informativi ad alta priorità. Questi devono essere trasmessi il prima possibile, indipendentemente dalla loro posizione nello stream. Questo campo, se valido, conterrà un puntatore alla posizione, nello stream, dei dati NON urgenti (ultimo byte dei dati urgenti).



# TCP: PDU

- **Checksum**
  - Campo di 16 bit contenente un valore intero utilizzato dal TCP della macchina host di destinazione, per verificare l'integrità dei dati e la correttezza dell'intestazione
    - questa informazione è di essenziale importanza perché il protocollo IP non prevede nessun controllo di errore sulla parte dati del frame
    - per il calcolo del valore checksum il TCP ha bisogno di aggiungere una **pseudointestazione** al datagramma, per effettuare così un controllo anche sugli indirizzi IP di destinazione e provenienza



# TCP: PDU

- La Pseudointestazione viene creata e posta in testa al segmento TCP (header + payload)
- Viene inserito in essa un ulteriore byte di zeri per raggiungere un multiplo di 16 bit
- Successivamente viene calcolata la checksum su tutto il messaggio così formato, viene scartata la pseudointestazione e passato il datagramma al livello IP
- In fase di ricezione, il livello TCP ricrea la pseudointestazione interagendo con l'IP sottostante, calcola la checksum e verifica la correttezza del messaggio ricevuto. In caso di errore il datagramma verrà scartato (e quindi ritrasmesso dal mittente)

0	8	16	31
Indirizzo IP provenienza			
Indirizzo IP destinazione			
Zero	Protocollo	Lunghezza TCP	



# TCP: PDU

- Principali **opzioni** di TCP
  - Maximum TCP payload: durante la fase di connessione, ciascun end-point annuncia la massima dimensione di payload che desidera accettare; la minima tra le due dimensioni annunciate viene selezionata per la trasmissione
  - Window Scale: per negoziare un fattore di scala per la finestra; utile per connessioni a larga banda e/o elevato ritardo di trasmissione
  - Selective Repeat: nel caso in cui un segmento corrotto sia stato seguito da segmenti corretti, introduce i NAK (Not AcKnowledge), per permettere al receiver di richiedere la ritrasmissione di quello specifico segmento; è un' alternativa al “go back n”, che prevede la ritrasmissione di tutti i segmenti



# TCP: Caratteristiche

- Riscontro e ritrasmissione
  - Consiste nella ritrasmissione di un segmento se non è giunta conferma entro un tempo massimo (**time-out**)
- Time-Out
  - Al momento della trasmissione di un segmento, il TCP attiva un timer



# TCP: Numeri di sequenza e numeri di riscontro

- TCP vede i dati come un flusso di byte non strutturati, ma ordinati
- I numeri di sequenza riflettono questa visione: **il numero di sequenza per un segmento** è quindi il numero nel flusso di byte del primo byte nel segmento
  - Flusso lungo 500.000 byte, MSS uguale a 1000 byte, primo byte numerato con 0
  - TCP costruisce 500 segmenti, con numeri di sequenza 0, 1000, 2000 ...
- Per i numeri di riscontro, vista la natura full-duplex della connessione TCP, si ha che ad es.
  - A invia e contemporaneamente riceve da B
  - I segmenti B -> A , contengono un numero di sequenza relativo ai dati B -> A
  - **Il numero di riscontro** che A scrive nei propri segmenti è il numero di sequenza del byte successivo che A attende da B (e lo può mandare anche inviando dati a B)
- TCP invia riscontri cumulativi



# Numeri di sequenza e ACK di TCP

## Numeri di sequenza

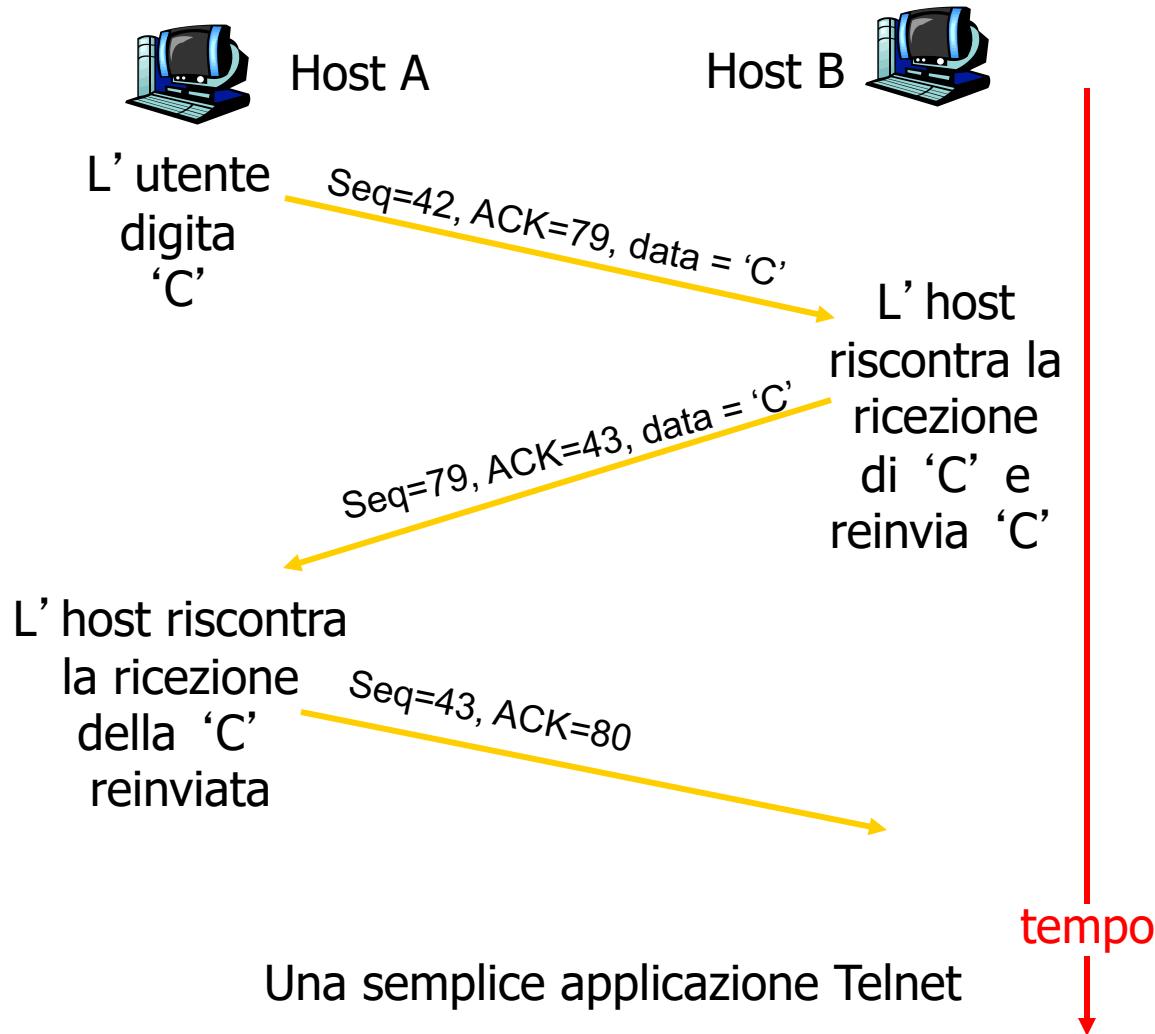
- “numero” del primo byte del segmento nel flusso di byte

## ACK

- numero di sequenza del prossimo byte atteso dall’ altro lato
- ACK cumulativo

D: come gestisce il destinatario i segmenti fuori sequenza?

- R: la specifica TCP non lo dice – dipende dall’ implementatore





# TCP: Caratteristiche del riscontro cumulativo

- Nel datagramma di riscontro la destinazione comunica quale byte dello stream si aspetta di ricevere successivamente
  - I riscontri specificano sempre il numero sequenziale del primo byte non ancora ricevuto
    - » Esempio: in uno stream di 1000 byte segmentato in blocchi di 100 byte, partendo da 0, il primo riscontro conterrà il numero sequenziale 100
  - Con questo metodo di riscontro *cumulativo* si ha il vantaggio che la perdita di un riscontro non blocca la trasmissione se confermato dal riscontro successivo



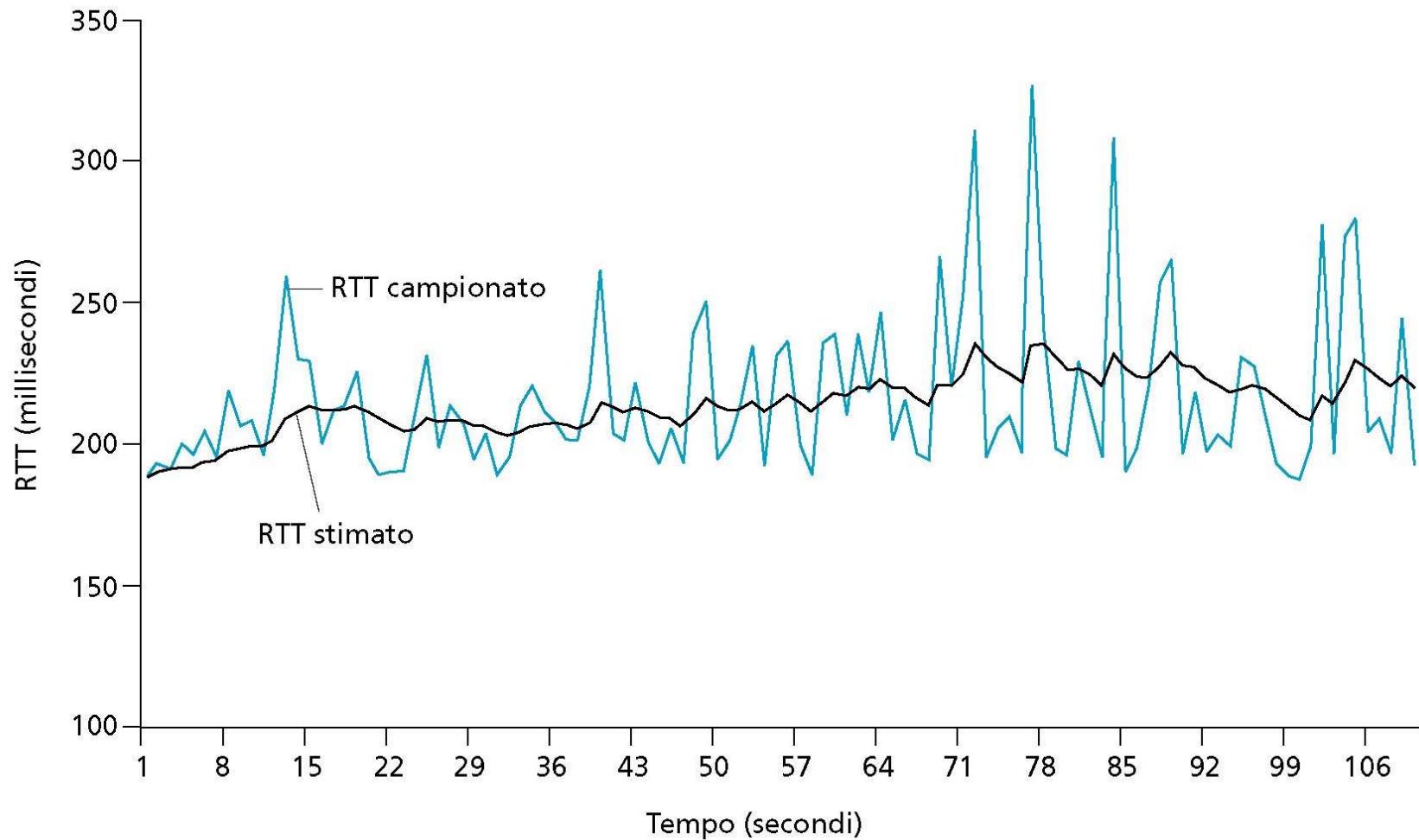
# Round Trip Time e Timeout

Domanda: A che valore deve essere impostato il timeout?

- Di sicuro sarà maggiore del RTT (Round Trip Time)
- **N.B:** RTT varia nel tempo
  - Se timeout è scelto troppo breve
    - timeout prematuro
    - ritrasmissioni ridondanti
    - scarsa efficienza
  - Se timeout è scelto troppo lungo
    - scarsa efficienza nella gestione delle ritrasmissioni



# RTT campionato vs RTT stimato



**SampleRTT:** tempo misurato dalla trasmissione del segmento fino alla ricezione di ACK, ignorando le ritrasmissioni (se ne sceglie uno ad ogni istante di tempo)



# Calcolo del timeout

$$\text{EstimatedRTT} = (1-\alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- Una media esponenziale pesata (*EWMA: Exponential Weighted Moving Average*) dei campioni
  - L'influenza di un singolo campione sul valore della stima decresce in maniera esponenziale, e si dà più importanza a campioni recenti.
  - Valore tipico per  $\alpha$ : 0.125

## Valore del timeout

- **EstimatedRTT** più un “margin di sicurezza” proporzionale alla variabilità della stima effettuata
  - variazione significativa di **EstimatedRTT** -> margine più ampio

$$\text{Timeout} = \text{EstimatedRTT} + 4 * \text{DevRTT}, \text{ dove}$$

$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

NB: Valore raccomandato per  $\beta$ : 0,25

\* DevRTT è una EWMA della differenza **SampleRTT-EstimatedRTT**



# TCP: trasferimento dati affidabile

- TCP crea un servizio di trasferimento dati affidabile sul servizio inaffidabile di IP
- Pipeline dei segmenti
- ACK cumulativi
- TCP usa un solo timer di ritrasmissione
- Le ritrasmissioni sono avviate da
  - eventi di timeout
  - ACK duplicati
- Inizialmente consideriamo un mittente TCP semplificato
  - ignoriamo gli ACK duplicati
  - ignoriamo il controllo di flusso e il controllo di congestione



# TCP: eventi del mittente

## Dati ricevuti dall' applicazione

- Crea un segmento con il numero di sequenza
- Il numero di sequenza è il numero del primo byte del segmento nel flusso di byte
- Avvia il timer, se non è già in funzione (pensate al timer come se fosse associato al più vecchio segmento non riscontrato)
- Intervallo di scadenza  
TimeOutInterval

## Timeout

- Ritrasmette il segmento che ha causato il timeout
- Riavvia il timer

## ACK ricevuti

- Se riscontra segmenti precedentemente non riscontrati
  - aggiorna ciò che è stato completamente riscontrato
  - avvia il timer se ci sono altri segmenti da completare



# Un sender TCP semplificato

/\* Si è assunto che il sender non sia limitato dal controllo di flusso o di congestione del TCP, che i dati da sopra siano di dimensioni inferiori all'MSS e che il trasferimento dei dati avvenga in una sola direzione. \*/

```
NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) {
    switch(event)

        event: data received from application above
            create TCP segment with sequence number NextSeqNum
            if (timer currently not running)
                start timer
            pass segment to IP
            NextSeqNum=NextSeqNum+length(data)
            break;

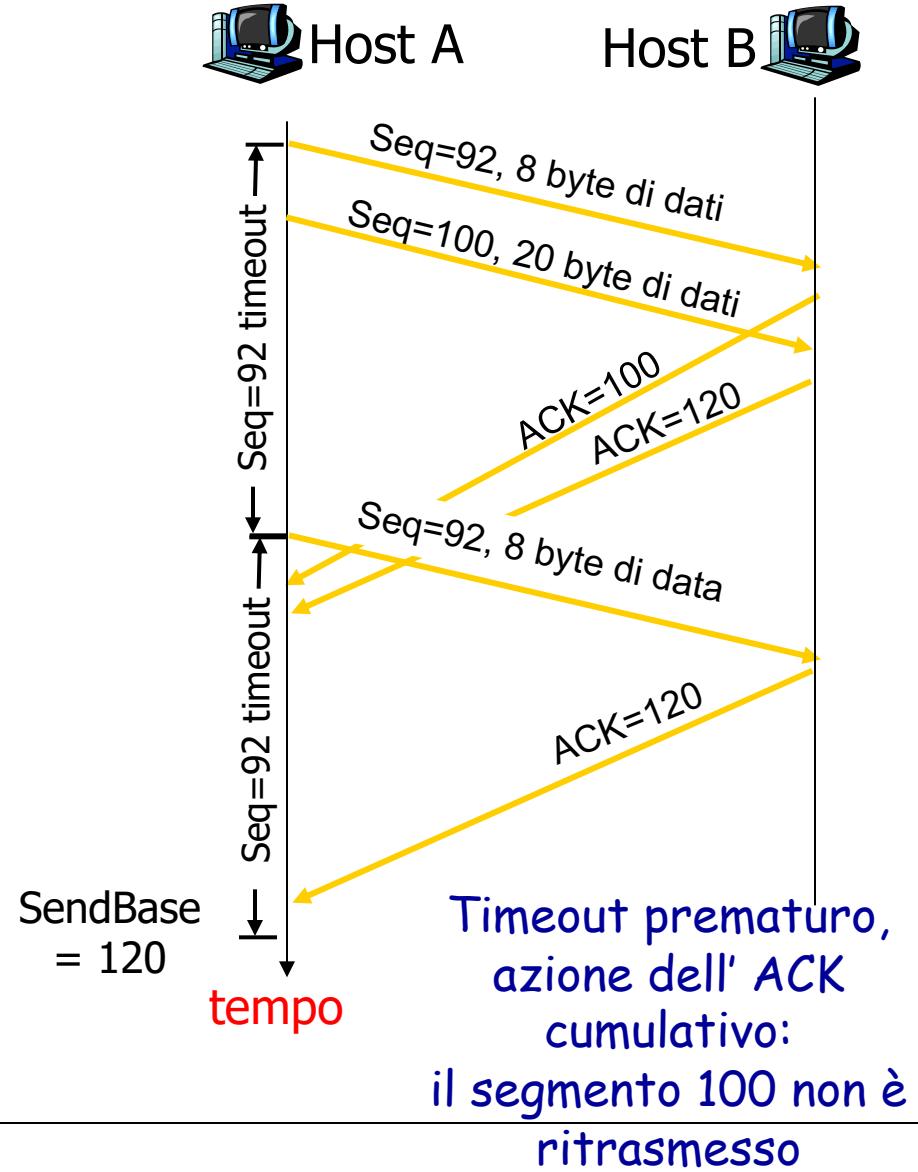
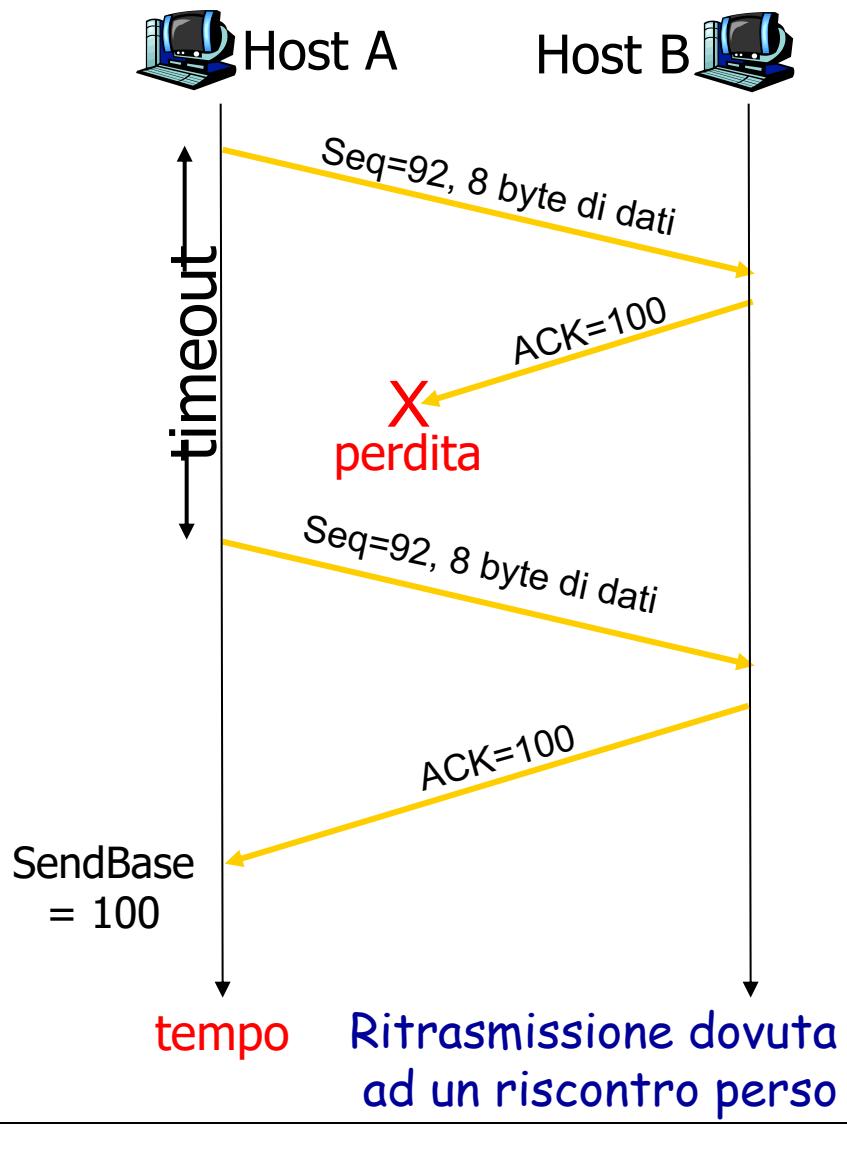
        event: timer timeout
            retransmit not-yet-acknowledged segment with
                smallest sequence number
            start timer
            break;

        event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase=y
                if (there are currently any not-yet-acknowledged
                    segments)
                    start timer
            }
            break;

    } /* end of loop forever */
```

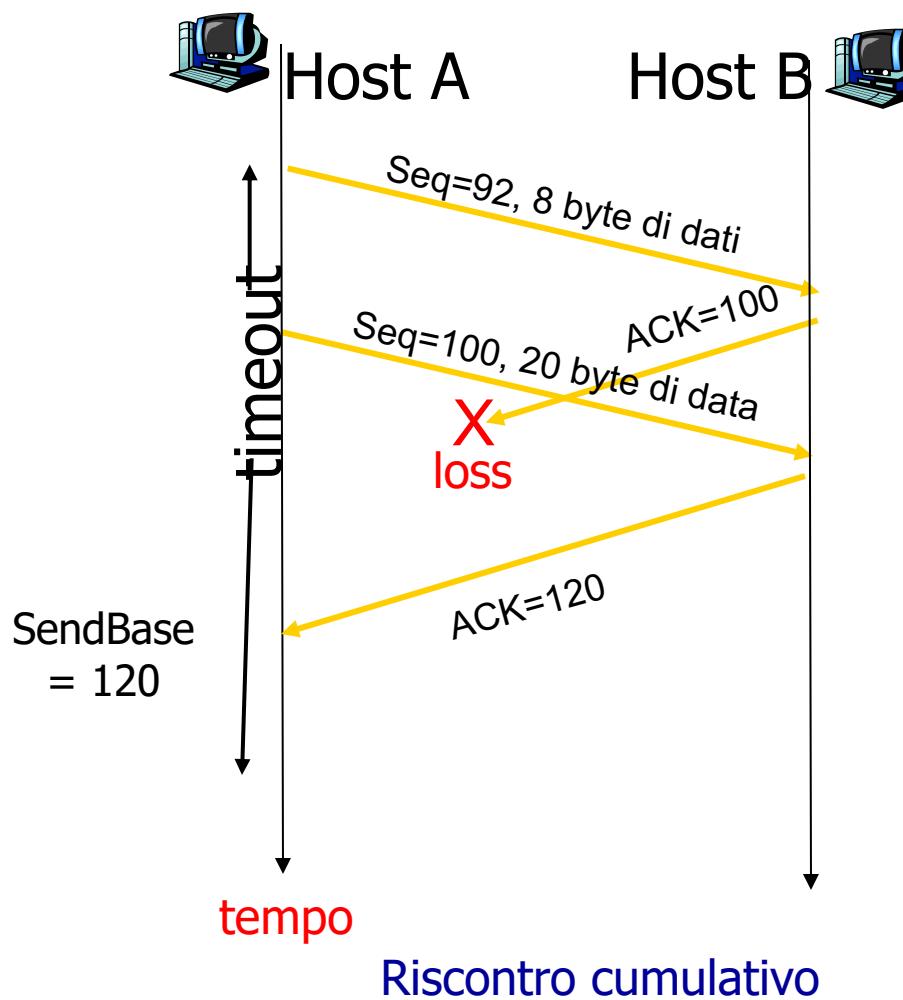


# Alcuni scenari di rilievo - 1





# Alcuni scenari di rilievo - 2



Il riscontro cumulativo  
evita la ritrasmissione del  
primo segmento...



# Modifiche tipiche del TCP - 1

- Raddoppio dell'intervallo di timeout
  - Allo scadere di un timeout
    - si imposta il prossimo intervallo al doppio del valore precedente (invece di usare la stima di RTT)
      - Crescita esponenziale degli intervalli dopo ogni ritrasmissione
    - Quando il timer viene riavviato (ricezione di un ACK o di nuovi dati dall'applicazione)
      - l'intervallo di timeout viene nuovamente configurato in funzione dei valori più recenti di *EstimatedRTT* e *DevRTT*
  - Fornisce una forma limitata di controllo della congestione
    - Il mittente, nel caso supponga una situazione di congestione (perdita di un segmento), ritrasmette ad intervalli sempre più lunghi



# Modifiche tipiche del TCP - 2

- Ritrasmessione veloce
  - ACK duplicati
    - Consentono di rilevare la perdita di un pacchetto prima del timeout
      - un receiver che rileva un “buco” nei segmenti ricevuti (ricezione di un segmento con numero di sequenza maggiore di quello atteso):
        - » invia un nuovo riscontro per l’ultimo byte di dati che ha ricevuto correttamente
      - poiché il mittente spesso manda molti segmenti contigui, se uno di tali segmenti si perde, ci saranno molti ACK duplicati contigui:
        - » un sender che riceve tre ACK duplicati per gli stessi dati assume che il segmento successivo a quello riscontrato tre volte è andato perso ed effettua, quindi, una ritrasmessione prima della scadenza del timeout



# TCP: generazione di ACK [RFC 1122, RFC 2581]

Evento nel destinatario	Azione del ricevente TCP
Arrivo ordinato di un segmento con numero di sequenza atteso. Tutti i dati fino al numero di sequenza atteso sono già stati riscontrati	ACK ritardato. Attende fino a 500 ms l'arrivo del prossimo segmento. Se il segmento non arriva, invia un ACK
Arrivo ordinato di un segmento con numero di sequenza atteso. Un altro segmento è in attesa di trasmissione dell'ACK	Invia immediatamente un singolo ACK cumulativo, riscontrando entrambi i segmenti ordinati
Arrivo non ordinato di un segmento con numero di sequenza superiore a quello atteso. Viene rilevato un buco	Invia immediatamente un ACK (duplicato), indicando il numero di sequenza del prossimo byte atteso
Arrivo di un segmento che colma parzialmente o completamente il buco	Invia immediatamente un ACK, ammesso che il segmento cominci all'estremità inferiore del buco



# TCP Connection Management

Mittente e Ricevente concordano l'apertura della connessione prima di inviare i dati

---

Impostare le variabili del TCP

- Numeri di sequenza
- Allocare i buffer, impostare un valore iniziale della **RcvWindow**

## Three way handshake

Passo 1: client invia segmento di controllo TCP SYN al server

- Specifica il 1° seq #

Passo 2: server riceve SYN, risponde con segmento di controllo SYN/ACK

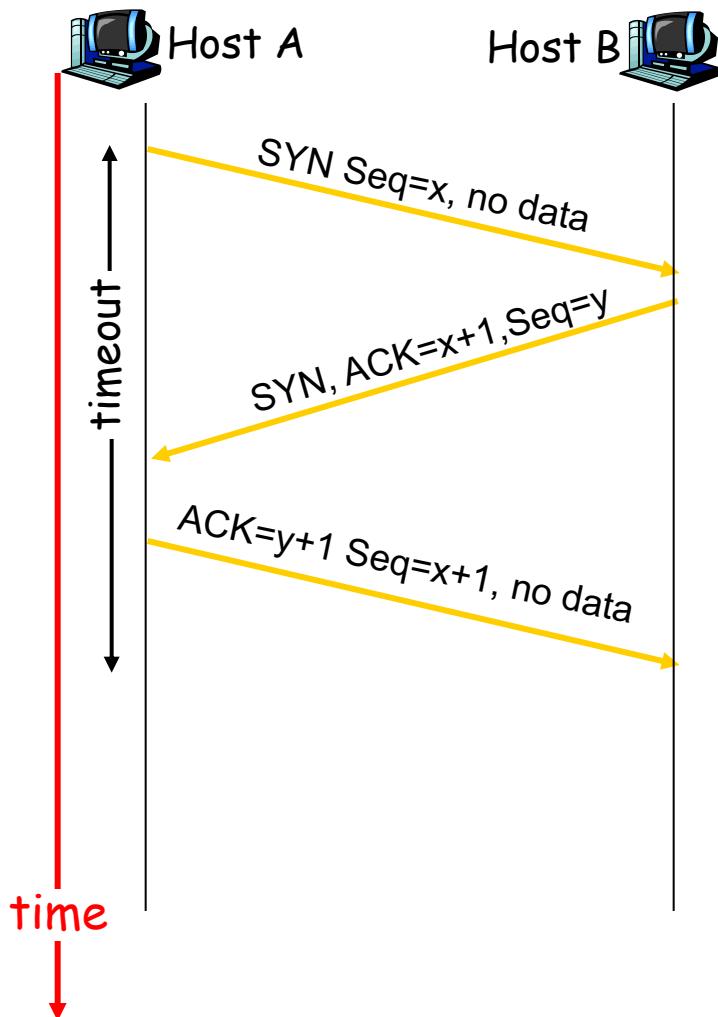
- ACK del SYN ricevuto
- Alloca buffer
- Specifica il 1° seq. # per la connessione server→client

Passo 3: client riceve SYN/ACK, invia ACK al server

- Connessione instaurata



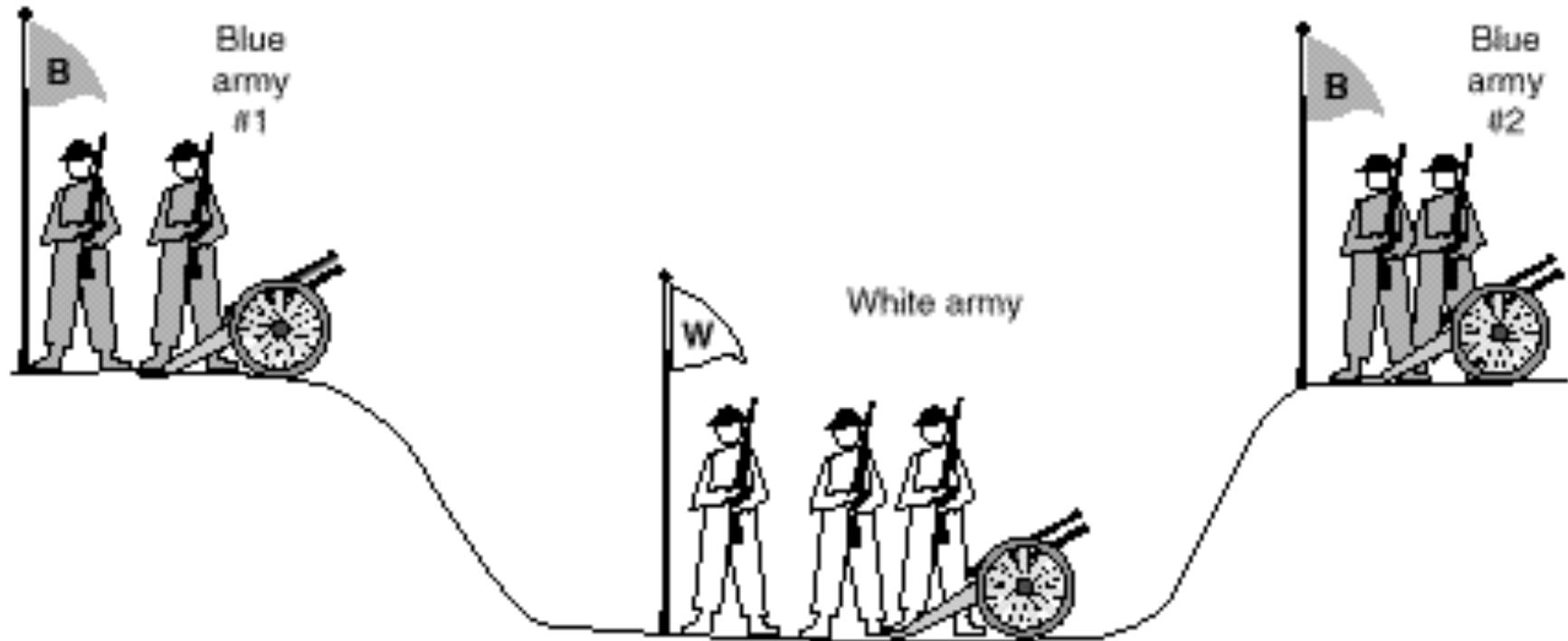
# Three way handshake





# Chiusura di una connessione

## Il problema dei due eserciti



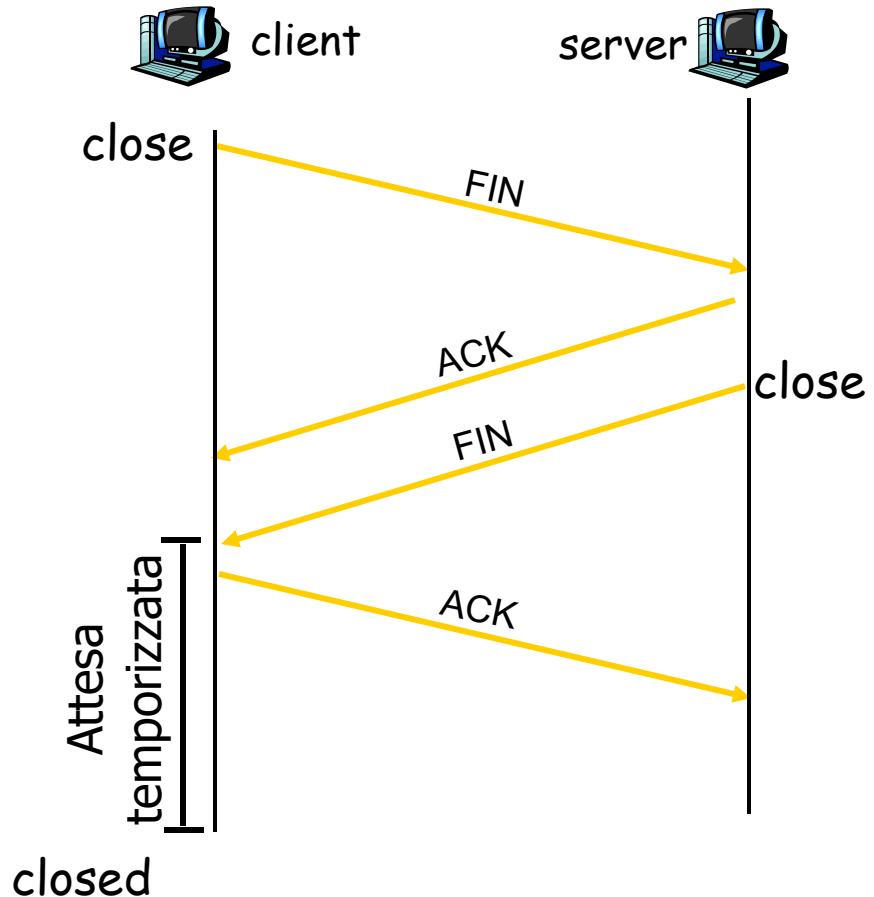


# Chiusura della connessione

Il client decide di chiudere la connessione...

**Passo 1:** client invia un segmento di controllo TCP con FIN alto al server

**Passo 2:** server riceve FIN, risponde con ACK. Quindi chiude la connessione inviando FIN al client





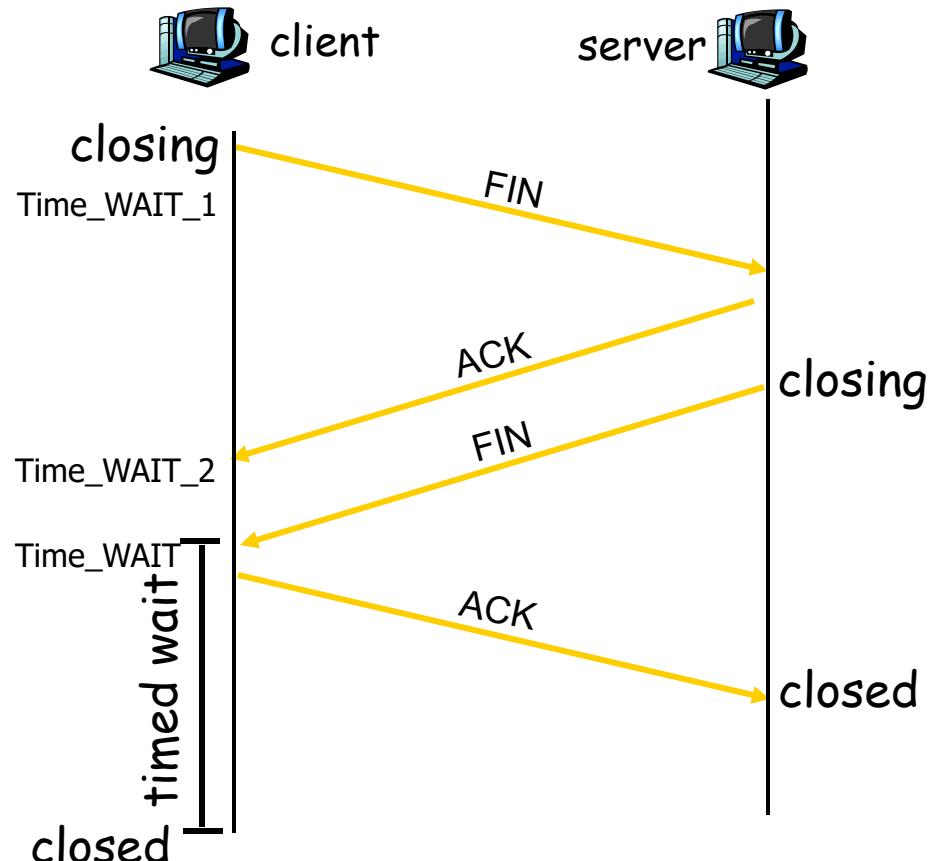
# Chiusura della connessione

**Passo 3:** [client](#) riceve FIN,  
risponde con un ACK

- Attende in uno stato  
TIMED\_WAIT (nel caso in  
cui l'ultimo ACK vada perso,  
e riceva un ulteriore FIN dal  
server)

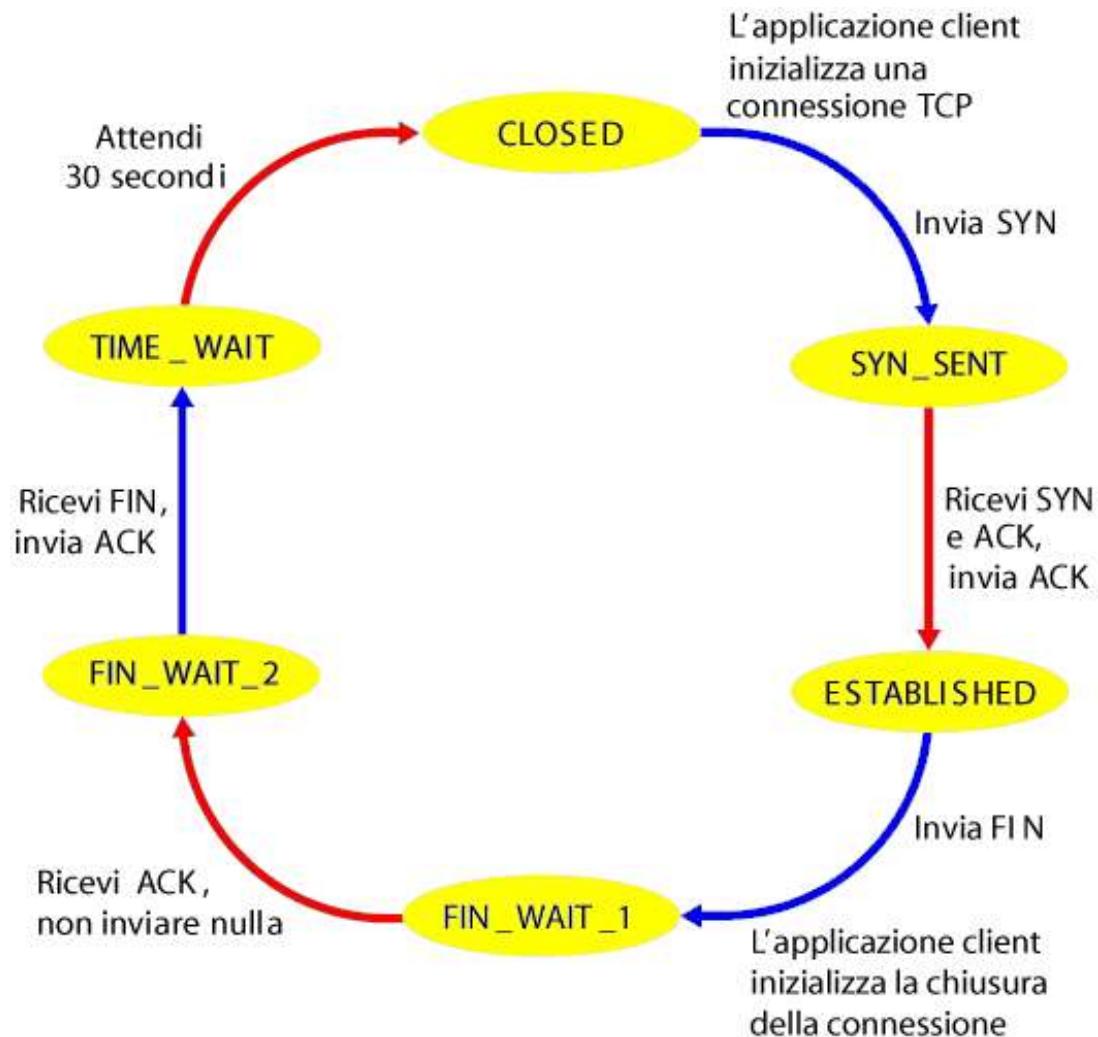
**Passo 4:** [server](#), riceve ACK.  
Chiude la connessione

**N.B.:** una piccola variante al  
Passo 2: il server invia ACK  
e FIN contemporaneamente  
all'interno dello stesso  
segmento



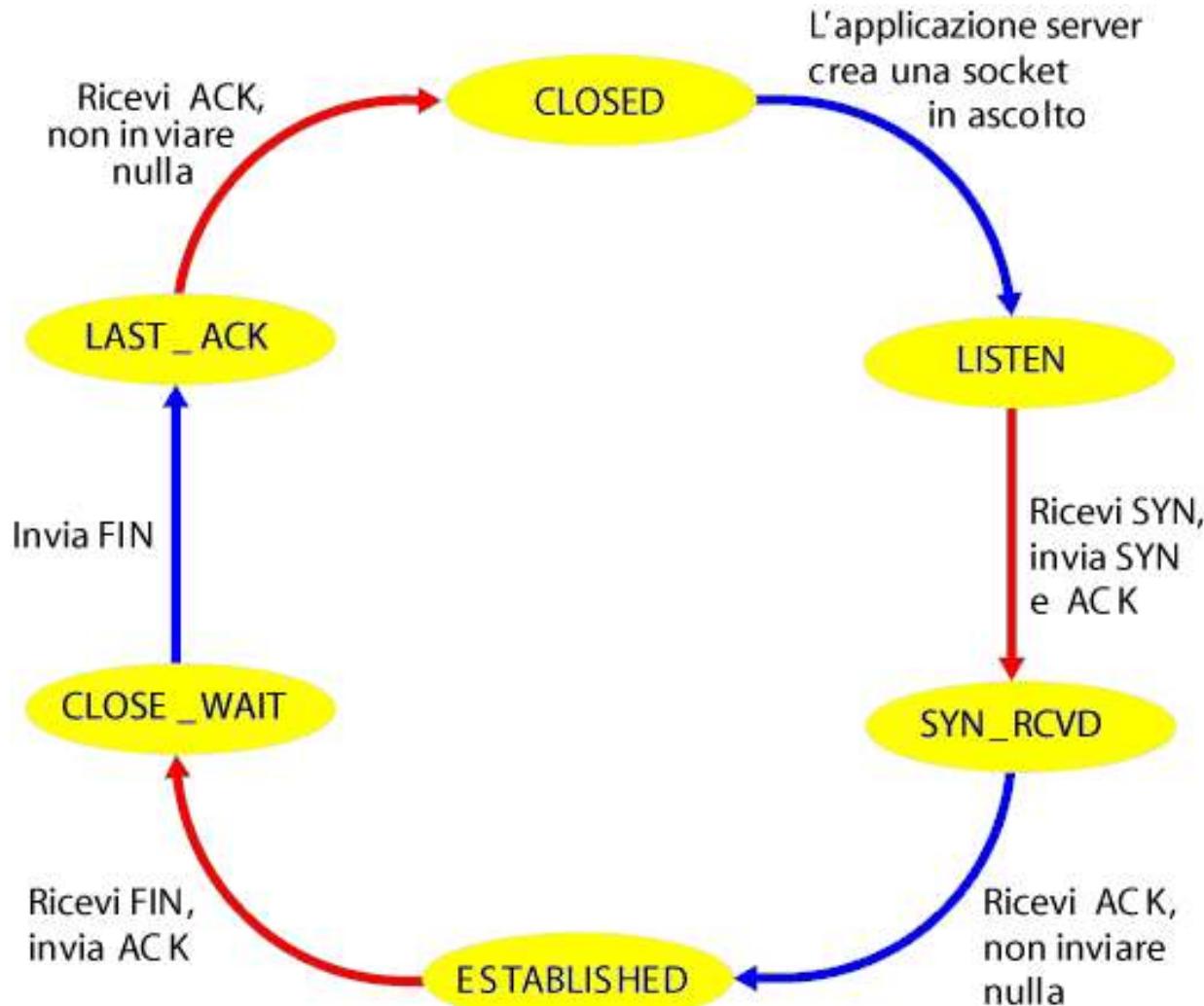


# Sequenza tipica degli stati nel client



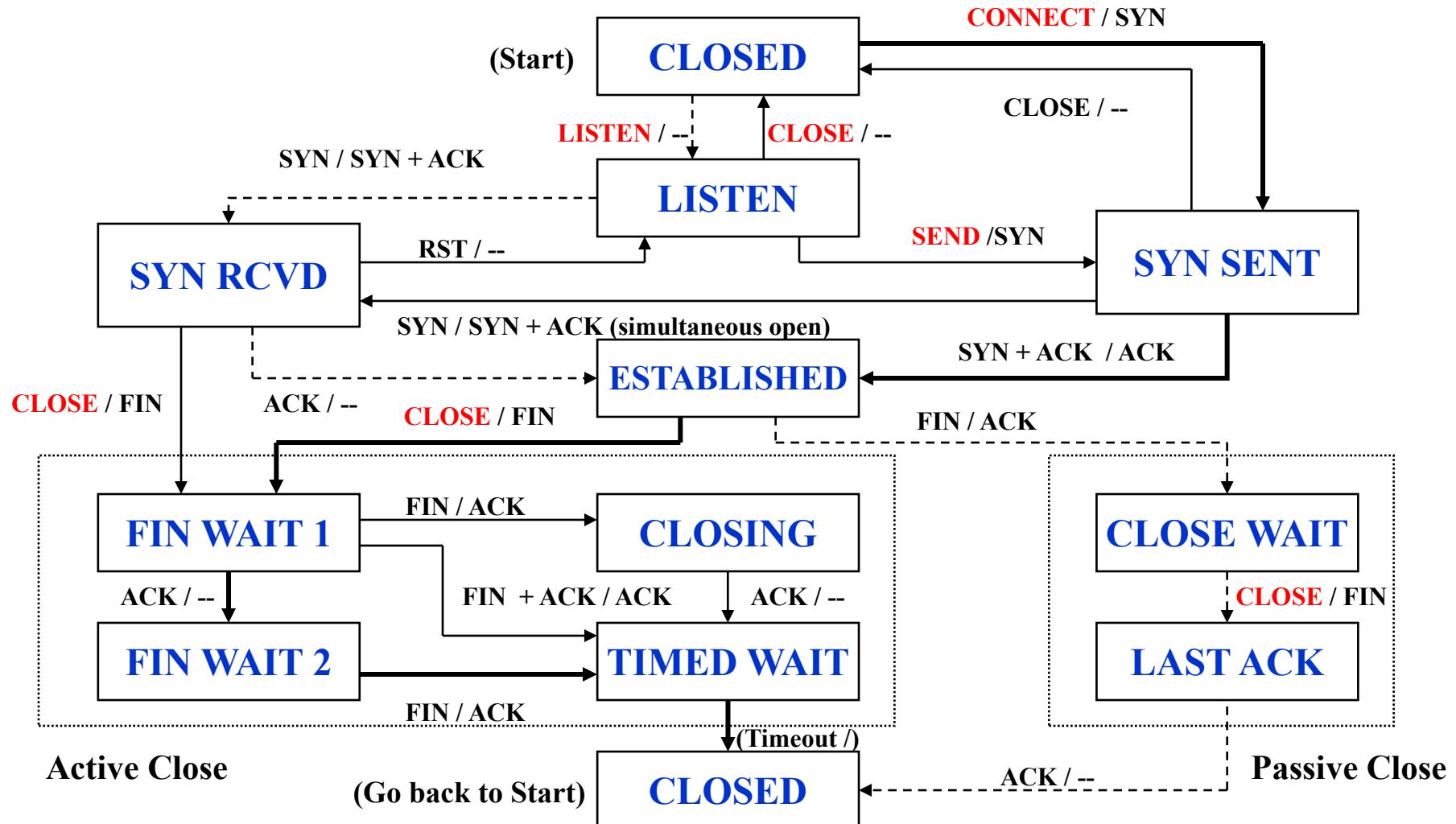


# Sequenza tipica degli stati nel server





# Diagramma degli stati del TCP





**Corso di Laurea in Informatica**

**Corso di Reti di Calcolatori 1**

**Alessio Botta ([a.botta@unina.it](mailto:a.botta@unina.it))**

**Il livello trasporto:**

**TCP: controllo di flusso e controllo della congestione**



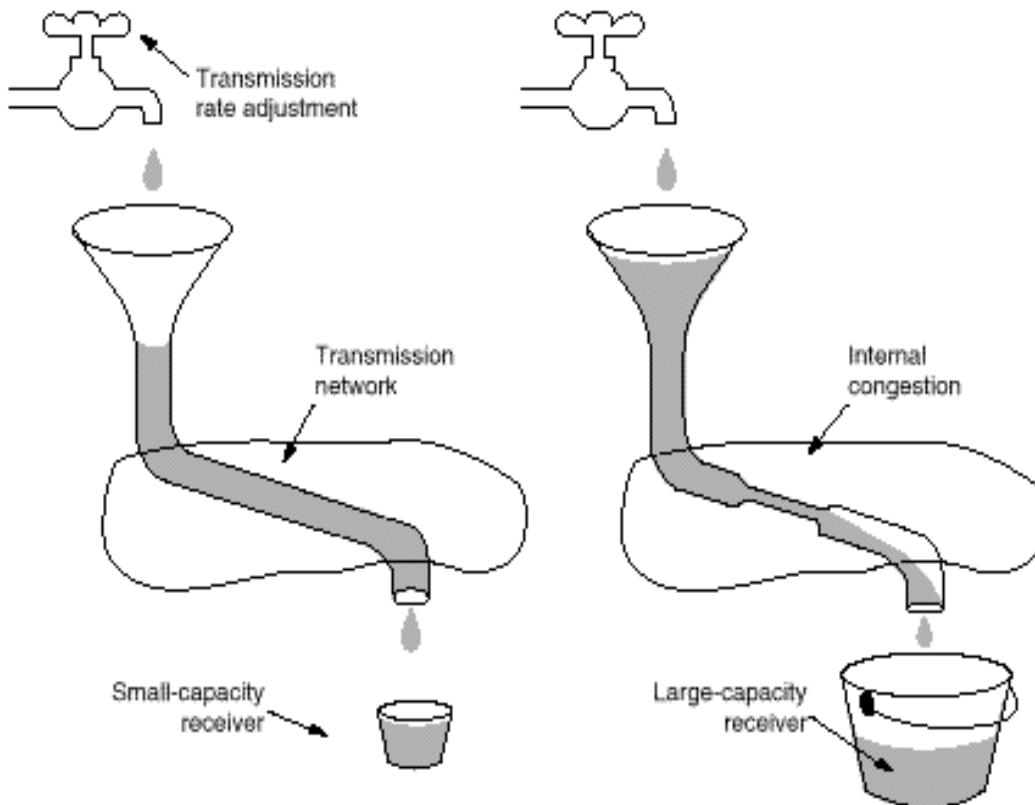
# Nota di Copyright

Quest'insieme di trasparenze è stato realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell' Università di Napoli. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell' uso dovrà essere esplicitamente riportata la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

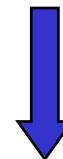


# TCP: Controllo di Flusso e di Congestione

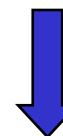
## Come gestire entrambi i tipi di controllo?



- *Receiver window*: dipende dalla dimensione del buffer di ricezione
- *Congestion window*: basata su una stima della capacità della rete



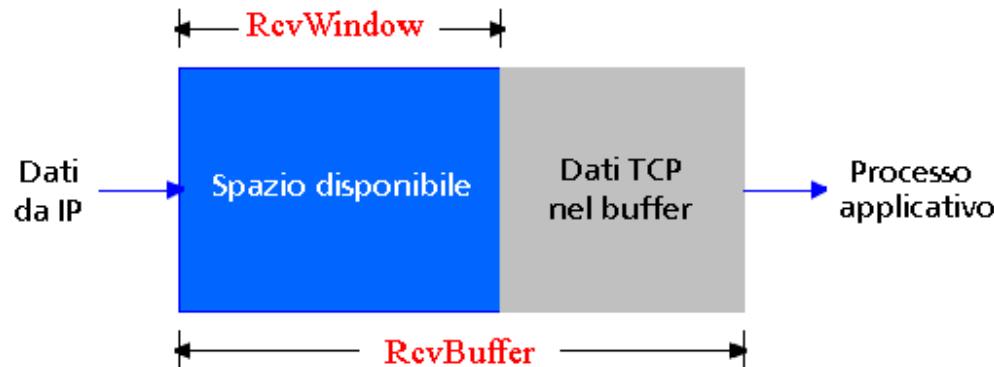
I byte trasmessi corrispondono alla dimensione della finestra più piccola





# TCP: controllo di flusso

- Il lato ricevente della connessione TCP ha un buffer di ricezione



- Il processo applicativo potrebbe essere rallentato dalla lettura nel buffer

## Controllo di flusso

- Servizio di corrispondenza delle velocità: la frequenza di invio deve corrispondere alla frequenza di lettura dell'applicazione ricevente

(supponiamo che il destinatario TCP scarti i segmenti fuori sequenza)



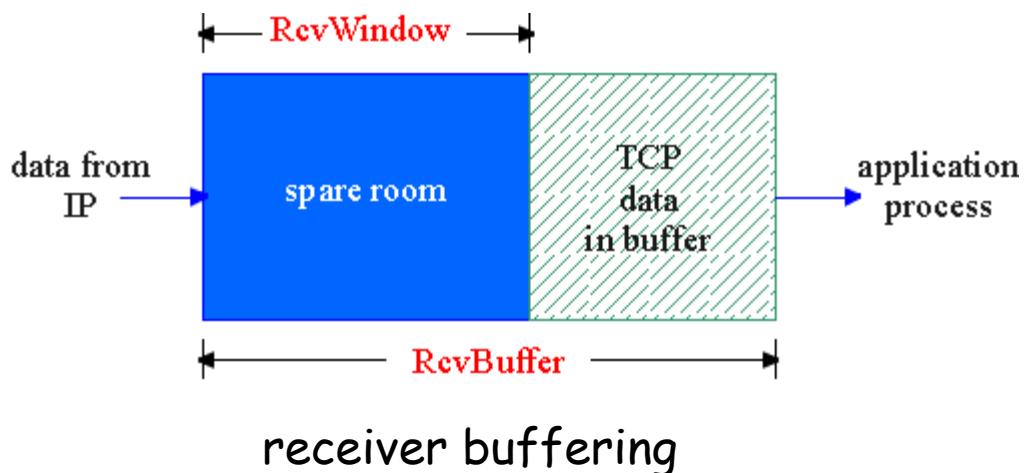
# TCP Flow Control

## flow control

Il mittente non dovrà sovraccaricare il ricevente inviando dati ad una velocità troppo elevata

RcvBuffer = size of TCP Receiver Buffer

RcvWindow = amount of spare room in Buffer



ricevente: comunica dinamicamente al mittente la dimensione corrente del buffer

- campo **RcvWindow** nel segmento TCP

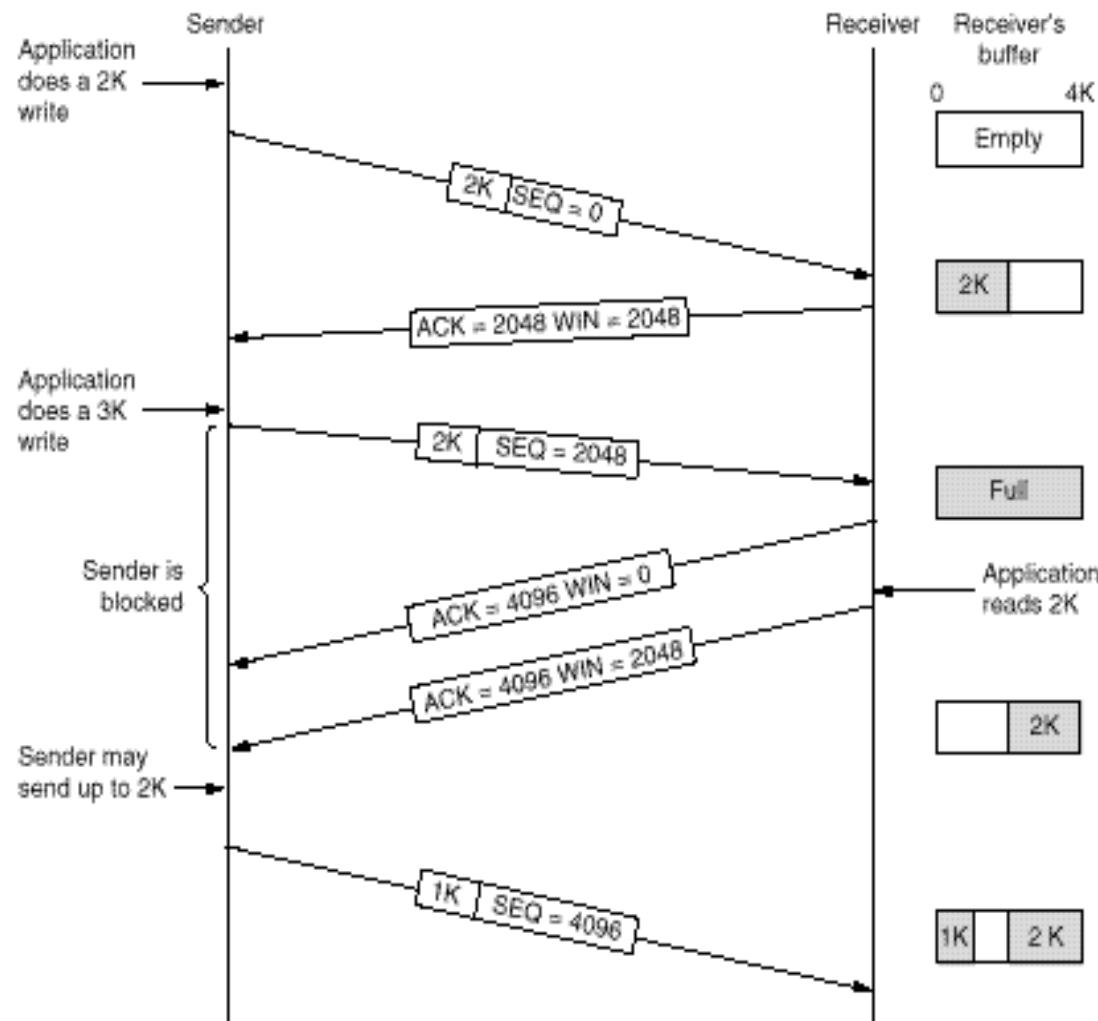
$$\text{RcvWindow} = \text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$$

mittente: conserva i dati già trasmessi ma non riscontrati e limita tale quantità all'ultima **RcvWindow** ricevuta

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{RcvWindow}$$



# TCP: Transmission Policy





# Silly Window Syndrome

- Silly Window Syndrome (ricevitore): il ricevitore svuota lentamente il buffer di ricezione e invia segmenti di ack con dimensione della finestra molto piccola, quindi il trasmettitore invia segmenti corti con molto overhead. La soluzione adottata è l'algoritmo di Clark: il ricevitore indica una finestra nulla finchè il buffer di ricezione non si è svuotato per metà o per una porzione uguale a MSS.
- Silly Window Syndrome (trasmettitore): l' applicazione genera dati lentamente, invia segmenti molto piccoli così come vengono prodotti. La soluzione adottata è l'algoritmo di Nagle: il TCP sorgente invia la prima porzione di dati anche se corta e gli altri vengono inviati solo se o il buffer di uscita contiene almeno MSS byte, oppure se si riceve un ack per il segmento precedente.
- Una volta vengono inviati segmenti corti con molto overhead perché la finestra al ricevitore è grande un byte, un'altra volta perché è l' applicazione che genera un byte alla volta molto lentamente.



# L'algoritmo di Nagle

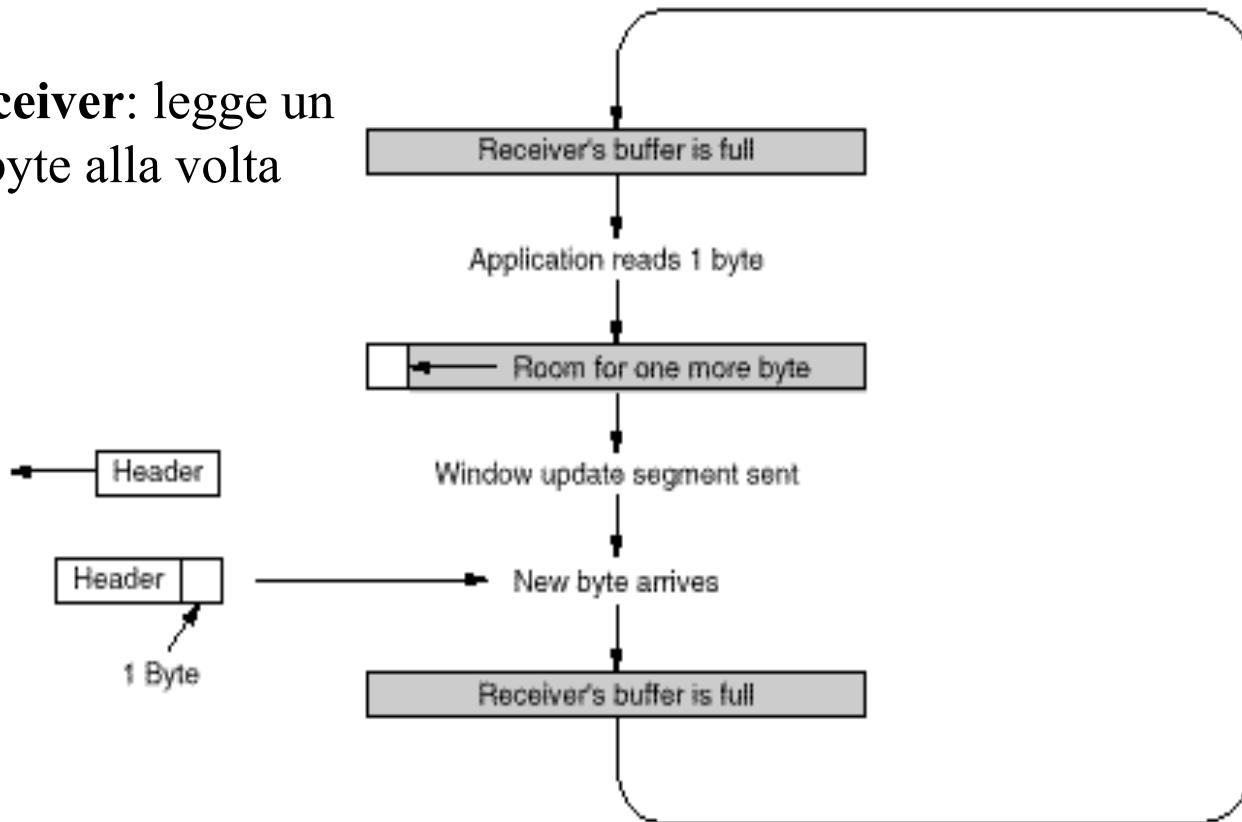
- Per applicazioni che inviano dati un byte alla volta (es: TELNET)
  - invia il primo byte e bufferizza il resto finché non giunge l' ACK
  - in seguito
    - invia, in un unico segmento, tutti i caratteri bufferizzati
    - ricomincia a bufferizzare finché non giunge l' ACK per ognuno di essi
  - Elimina la sindrome della Silly Window al trasmettitore



# La sindrome della Silly Window

**Sender:** invia blocchi grandi

**Receiver:** legge un byte alla volta



**Soluzione di Clark:**  
Impedisce al receiver di aggiornare la finestra un byte alla volta

Il ricevitore indica una finestra nulla finchè il buffer di ricezione non si è svuotato per metà o per una porzione uguale a MSS (Maximum Segment Size)



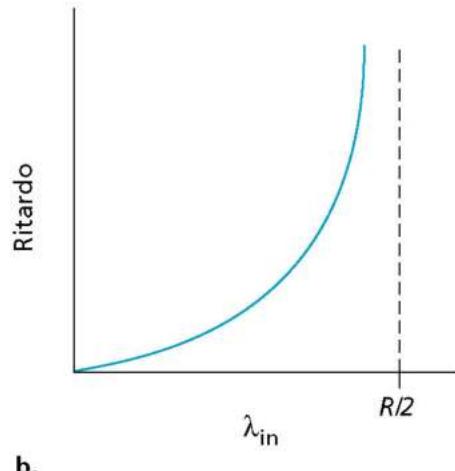
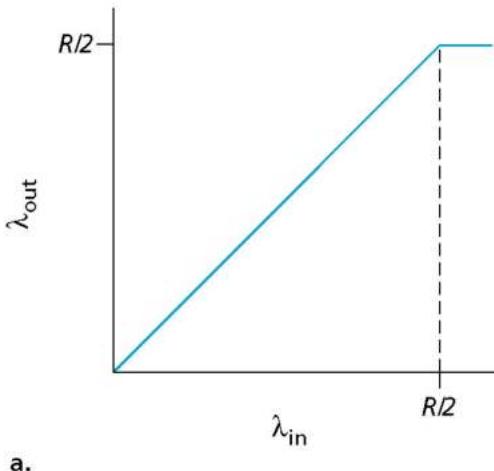
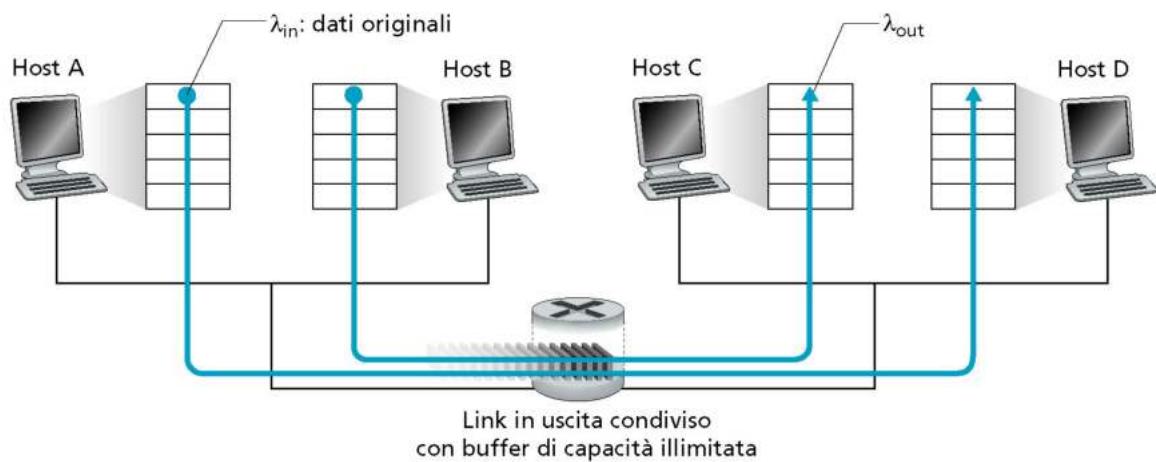
# Controllo della congestione

- Congestione nella rete
  - Tecnicamente dovuta a
    - un numero elevato di sorgenti di traffico
    - sorgenti di traffico che inviano troppi dati
    - traffico inviato ad una frequenza troppo elevata
  - In presenza di questi fenomeni, singoli o concomitanti, la rete è **sovraffollata**
    - Effetti
      - perdita di pacchetti
        - » buffer overflow nei router
      - ritardi nell'inoltro dei pacchetti
        - » accodamenti nei buffer dei router
      - scarso utilizzo delle risorse di rete



# Effetti della congestione: esempi (1/2)

- 2 mittenti
- 2 riceventi
- 1 router con buffer (coda)  $\infty$ :
  - non ci sono ritrasmissioni



- I ritardi aumentano all'avvicinarsi del limite di capacità del canale
- Non si può superare il max throughput



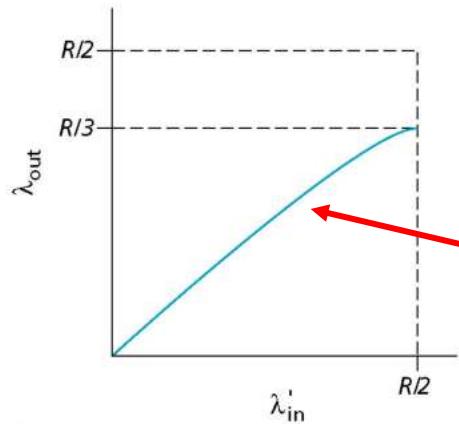
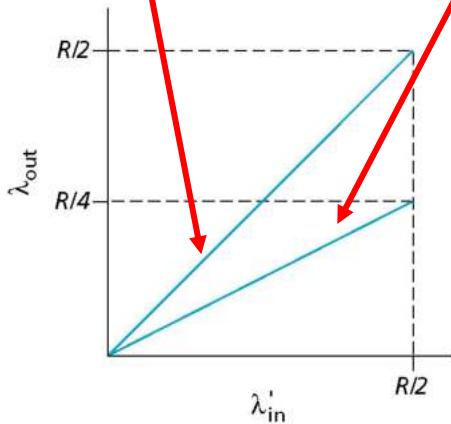
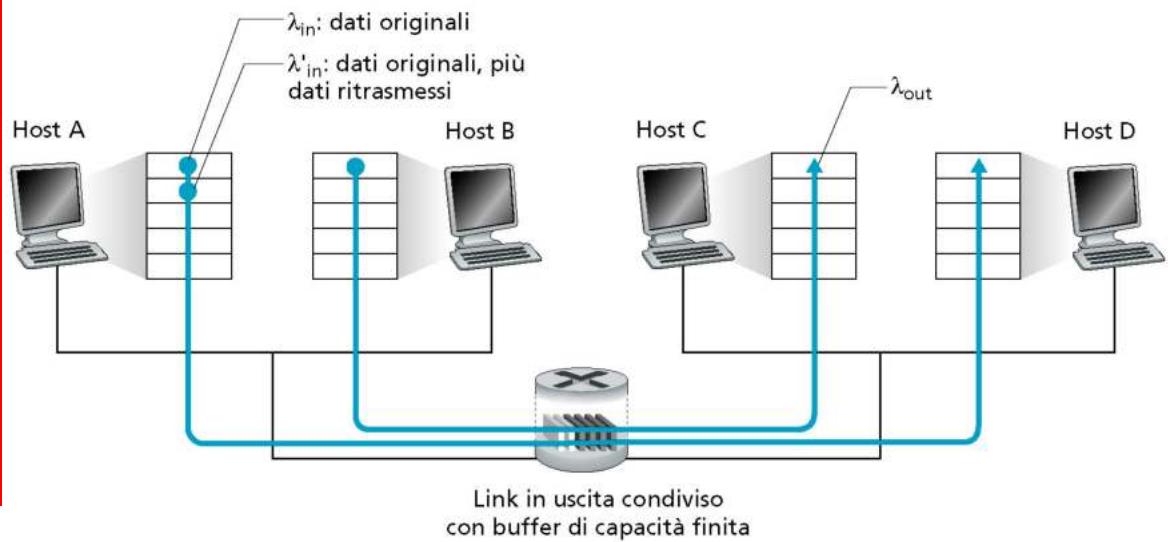
# Effetti della congestione: esempi (2/2)

Il mittente invia dati solo quando il buffer non è pieno:

- caso ideale
  - ✓ no ritrasmissioni
  - ✓ throughput max =  $R/2$

Scadenza prematura del timer del mittente:

- ✓ es: ogni segmento è spedito, in media, due volte
- ✓ throughput max =  $R/4$



Il mittente rispedisce un segmento solo quando è sicuro che sia andato perso:

- ✓ il throughput effettivo è inferiore al carico offerto (trasmissioni dati originali + ritrasmissioni)
- ✓ es: curva in figura



# Tecniche di Controllo della Congestione

## Approccio end-to-end

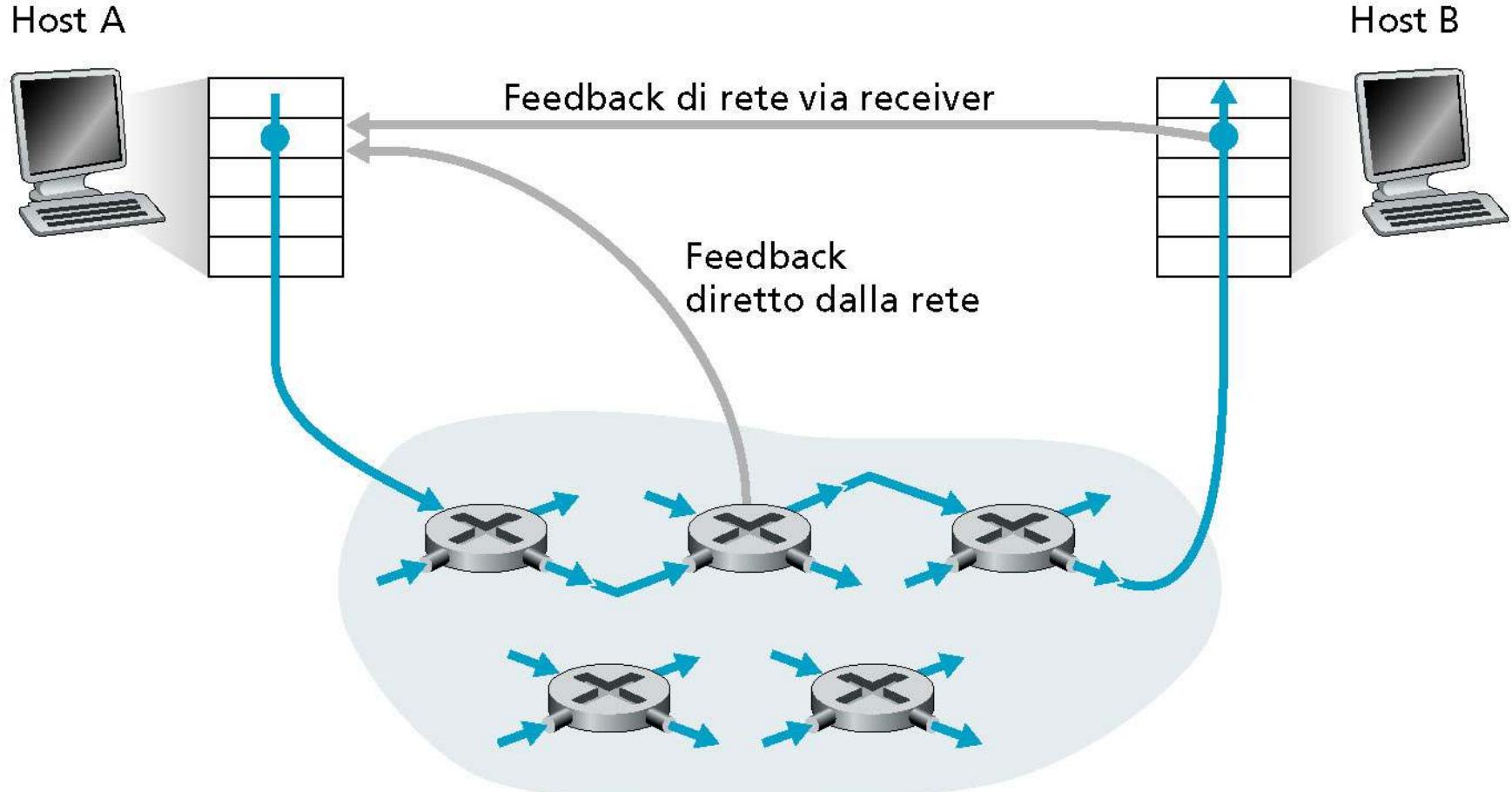
- Nessuna segnalazione esplicita dalla rete
- A partire dall'osservazione di ritardi e perdite di pacchetti gli end-system deducono uno stato di congestione nella rete
- Approccio utilizzato da TCP

## Approccio in base a segnalazione della rete

- I router forniscono informazioni circa lo stato della rete agli end-system
  - l'invio di un singolo bit indica lo stato di congestione
    - SNA, DECbit, TCP/IP ECN, ATM
  - in alternativa, il sender è informato circa la massima frequenza alla quale può trasmettere



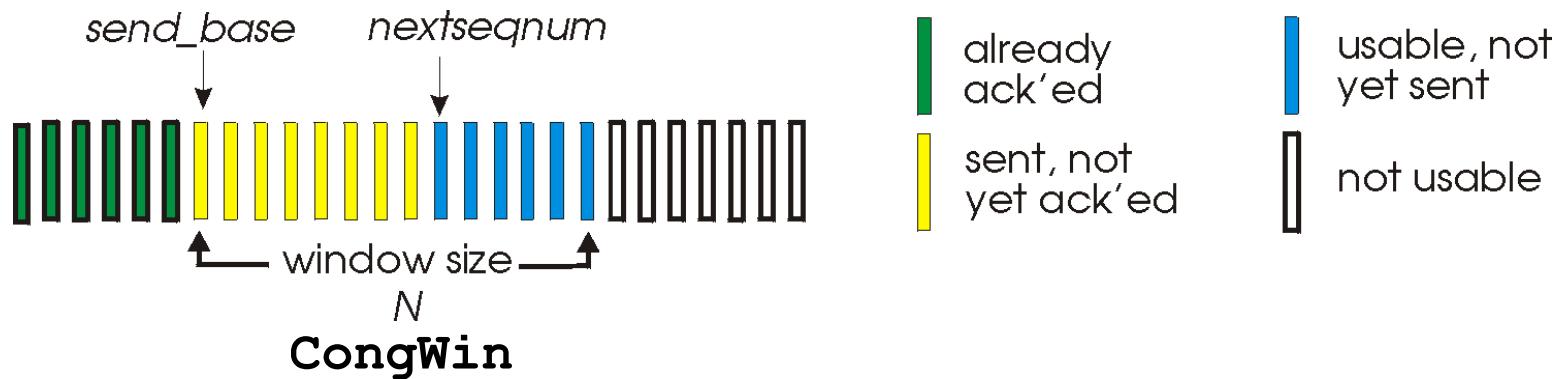
# Feedback di rete: tecniche alternative





# Controllo della congestione in TCP

- Controllo end-to-end: nessun feedback dalla rete
- Frequenza di trasmissione variabile
  - dipendente dalla cosiddetta *finestra di congestione (CongWin)*



**Considerando controllo di flusso e controllo di congestione  
insieme, si ha, dunque**

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{RcvWindow}, \text{CongWin}\}$$



# Controllo della congestione: idea di base

- Si procede **per tentativi**, per stabilire quanto si può trasmettere
  - obiettivo
    - trasmettere alla massima velocità possibile ( $Congwin$  quanto più grande possibile) senza perdite
  - approccio utilizzato
    - incrementare  $Congwin$  finché non si verifica la perdita di un segmento (interpretata come il sopraggiungere dello stato di congestione)
    - in seguito alla perdita di un segmento
      - decrementare  $Congwin$
      - ricominciare daccapo



# Controllo della congestione: fasi

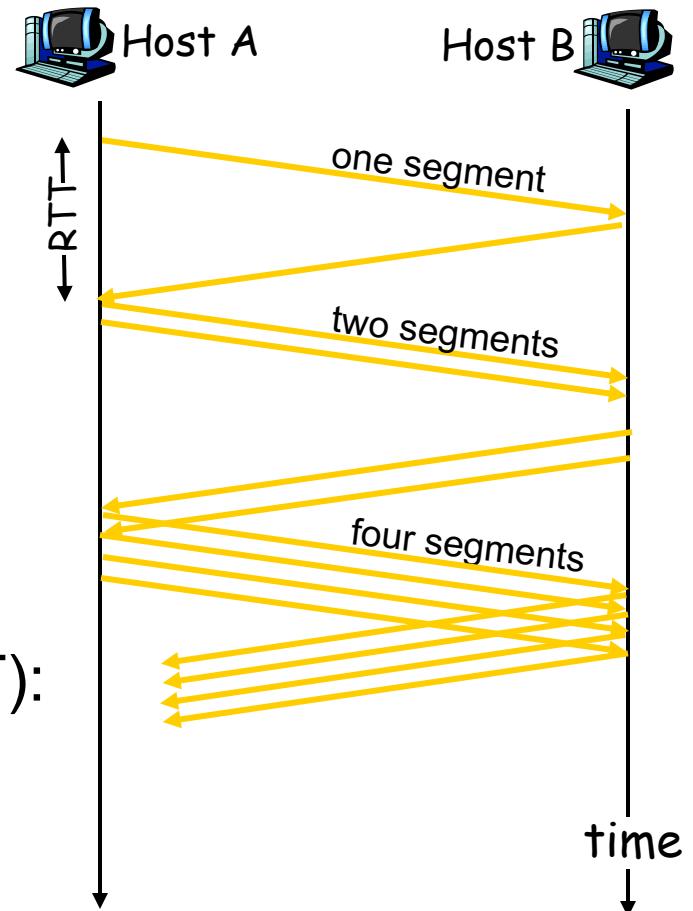
- Slow Start
  - Partenza lenta (per modo di dire!)
- Congestion Avoidance
  - Additive Increase, Multiplicative Decrease (AIMD)
    - incremento additivo, decremento moltiplicativo



# Lo Slow Start in TCP

## Algoritmo Slowstart

```
//initialization  
Congwin = 1 MSS  
  
for (each segment ACKed)  
    Congwin=Congwin+1MSS  
until (loss event OR  
      CongWin > threshold)
```



- Crescita esponenziale della dimensione della finestra (ogni RTT):
  - “Slow start” → termine improprio!
- Evento di *perdita*
  - timeout
  - tre ACK duplicati consecutivi

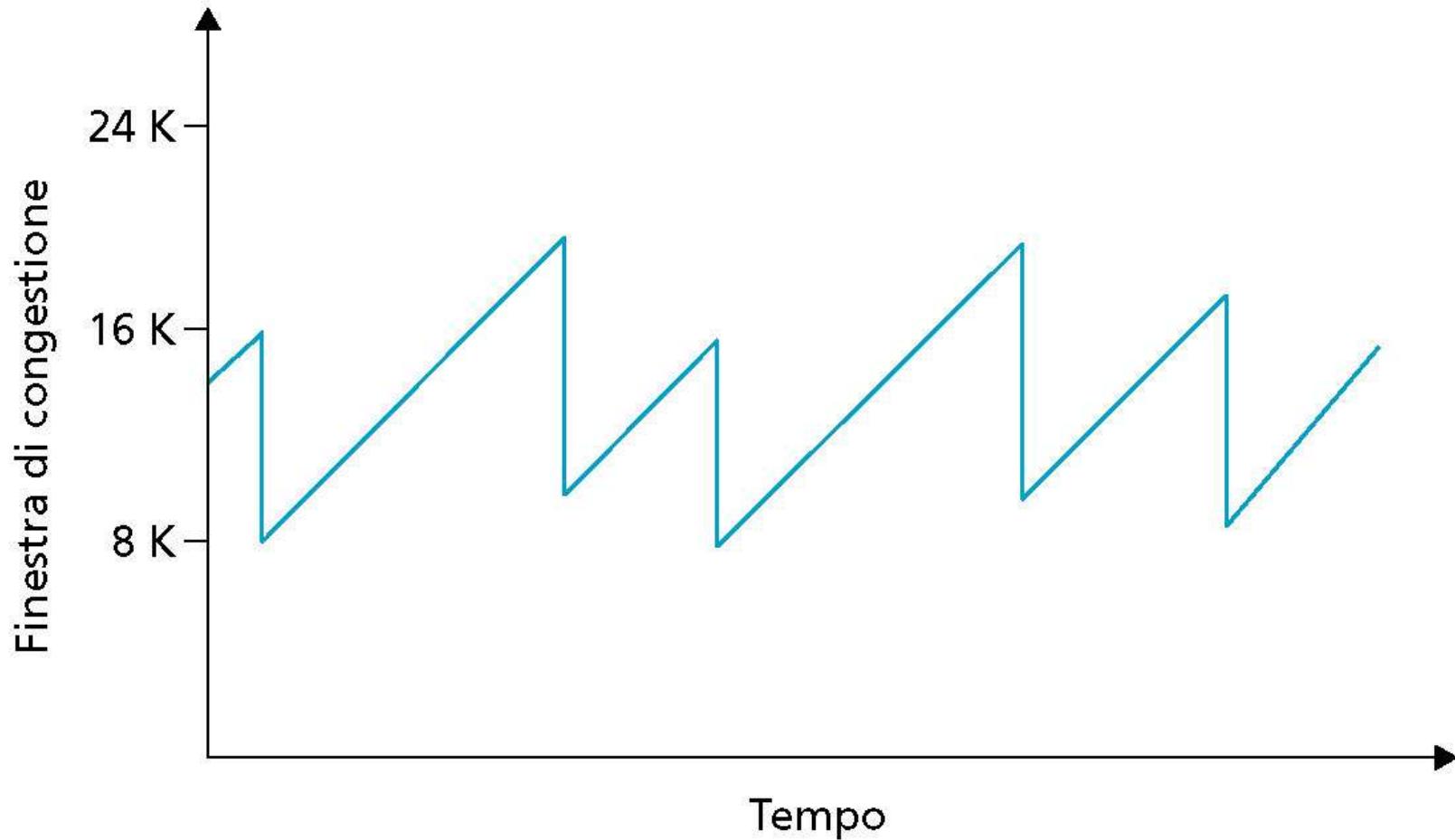


# Dopo lo slow start: AIMD

- **Additive Increase**
  - Una volta raggiunta la soglia
    - ci si avvicina con cautela al valore della banda disponibile tra le due estremità della connessione
    - meccanismo adottato
      - incremento di CongWin di  $\text{MSS} \cdot (\text{MSS}/\text{Congwin})$  alla ricezione di un ACK
    - questa fase è nota come fase di *congestion avoidance*
- **Multiplicative Decrease**
  - Al sopraggiungere della congestione (scadenza di un timeout o ricezione di tre ACK duplicati consecutivi)
    - la finestra di congestione viene dimezzata



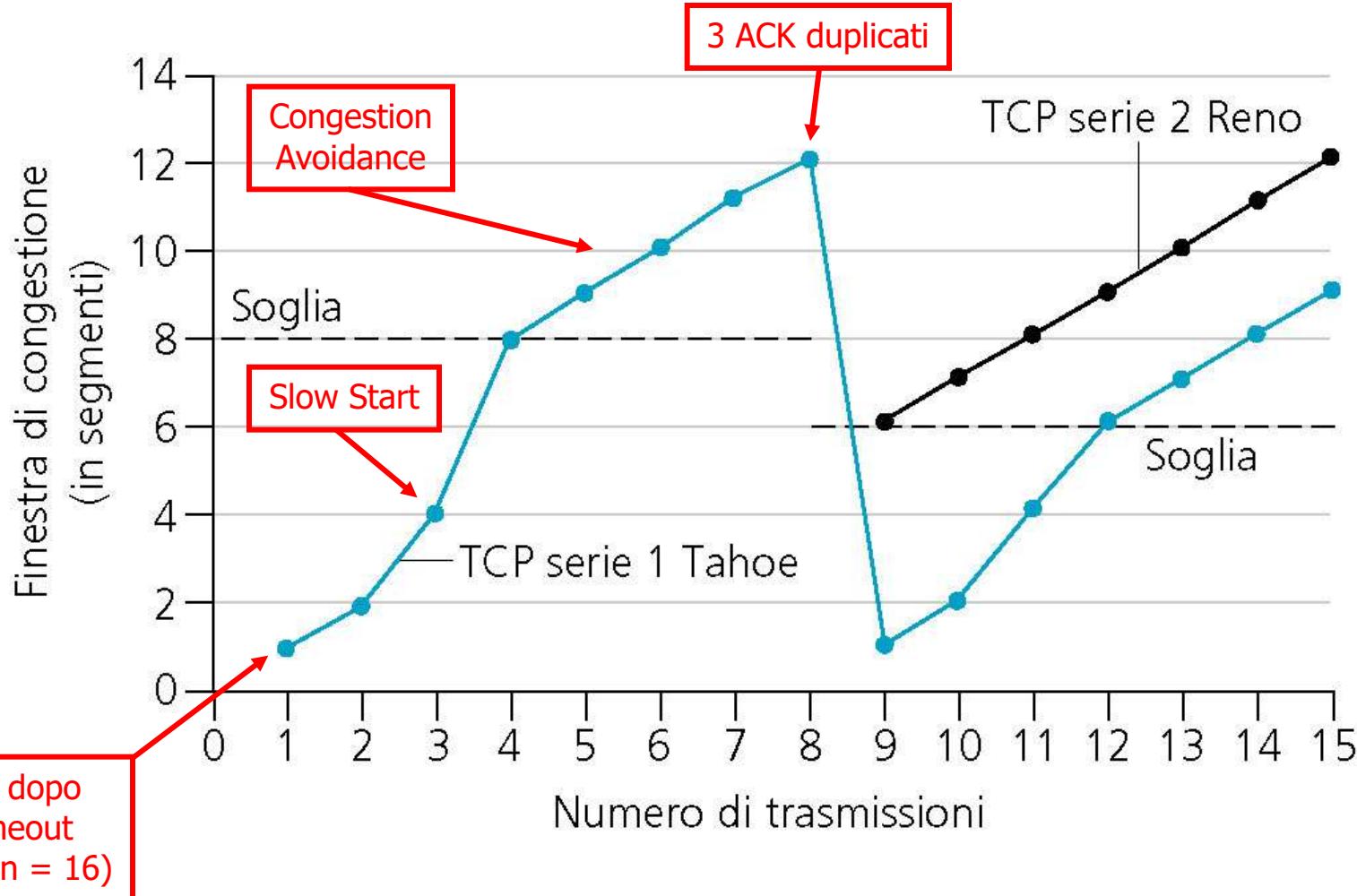
# AIMD: andamento a “dente di sega”





# Controllo della congestione in Internet

È presente un ulteriore parametro: **soglia** (threshold)





# Ricapitolando...

- Finestra di congestione sotto la soglia
  - Slow start
  - Crescita esponenziale della finestra
- Finestra di congestione sopra la soglia
  - Prevenzione della congestione
  - Crescita lineare della finestra
- Evento di perdita dedotto da ACK duplicato 3 volte
  - Soglia posta alla metà del valore attuale della finestra
  - **TCP Reno**
    - Finestra posta pari alla soglia
  - **TCP Tahoe**
    - Finestra posta pari ad un segmento (MSS -- Maximum Segment Size)
- Evento di perdita dedotto da timeout
  - Soglia posta alla metà del valore attuale della finestra
  - Finestra posta pari ad un segmento (MSS -- Maximum Segment Size)



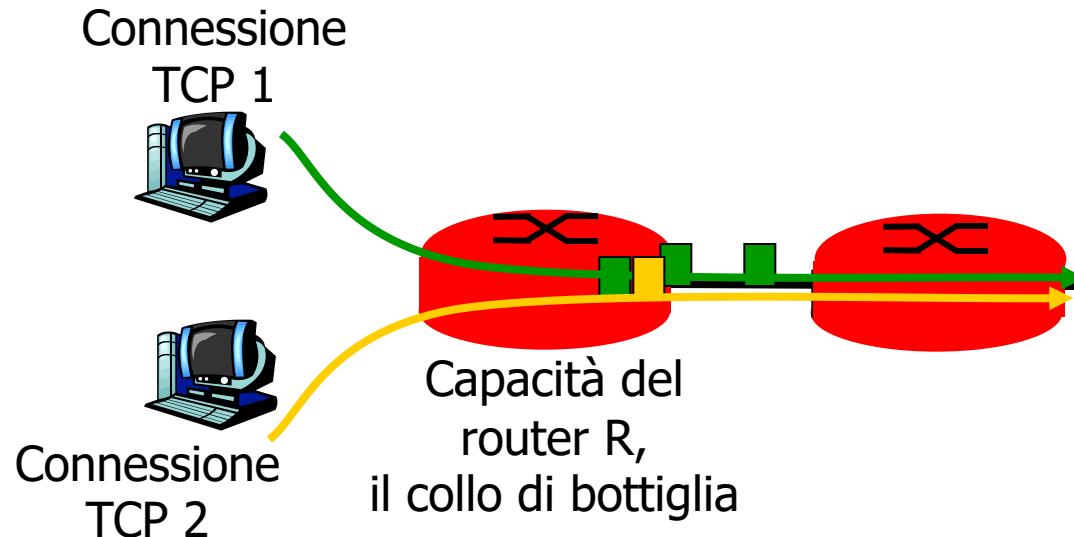
## TCP Reno: “*fast recovery*”

- TCP Reno elimina la fase di partenza lenta dopo un evento di perdita dedotto dalla ricezione di tre ACK duplicati
  - tale evento indica che, nonostante si sia perso un pacchetto, almeno 3 segmenti successivi sono stati ricevuti dal destinatario
    - a differenza del caso di timeout, la rete mostra di essere in grado di consegnare una certa quantità di dati
    - è possibile, quindi, evitare una nuova partenza lenta, ricominciando direttamente dalla fase di prevenzione della congestione



# Equità tra le connessioni TCP (1/2)

**Equità:** se  $K$  sessioni TCP condividono lo stesso collegamento con ampiezza di banda  $R$ , che è un collo di bottiglia per il sistema, ogni sessione dovrà avere una frequenza trasmittiva media pari a  $R/K$



Si può dire che AIMD è equo?



# Equità tra le connessioni TCP

- Ipotesi
  - MSS e RTT uguali per le due connessioni
    - a parità di dimensioni della finestra quindi il throughput è lo stesso
  - Entrambe le connessioni si trovano oltre lo slow start
    - fase di prevenzione della congestione
      - incremento additivo
      - decremento moltiplicativo

