



POLITECNICO
MILANO 1863



Fondamenti di TELECOMUNICAZIONI

Pierpaolo Boffi

Fondamenti di TELECOMUNICAZIONI

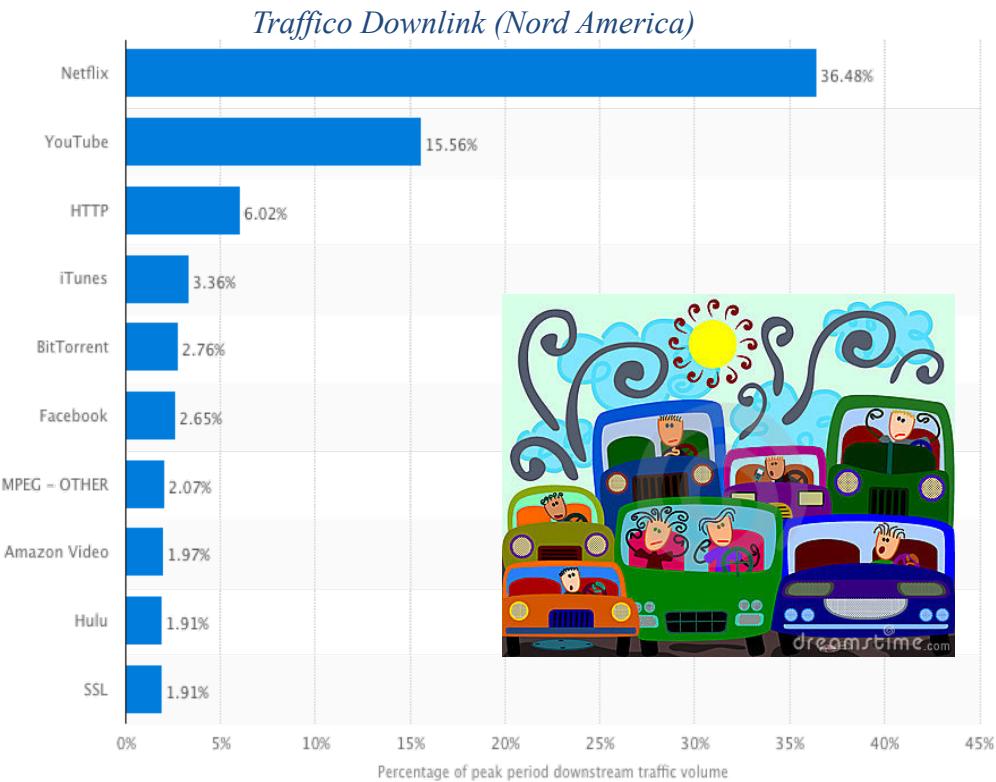


6 - Il Livello Applicativo

Applicazioni e Architetture, Web
Browsing, Posta Elettronica

Alcune applicazioni di rete

- **World Wide Web**
 - Firefox, Safari, Chrome, Apache
- **Posta elettronica:**
 - Mail, Gmail
- **Social networking:**
 - Facebook, Twitter, Instagram, Snapchat, ecc.. (social networking)
- **P2P file sharing:** BitTorrent, eMule, ecc..
- **Video streaming:**
 - NetFlix, YouTube, Hulu
- **Telefonia:**
 - Skype, Hangout, ecc..
- **Network games**
- **Video conference**
- **Massive parallel computing**
- **Instant messaging**
- **Remote login:**
 - TELNET
- ...

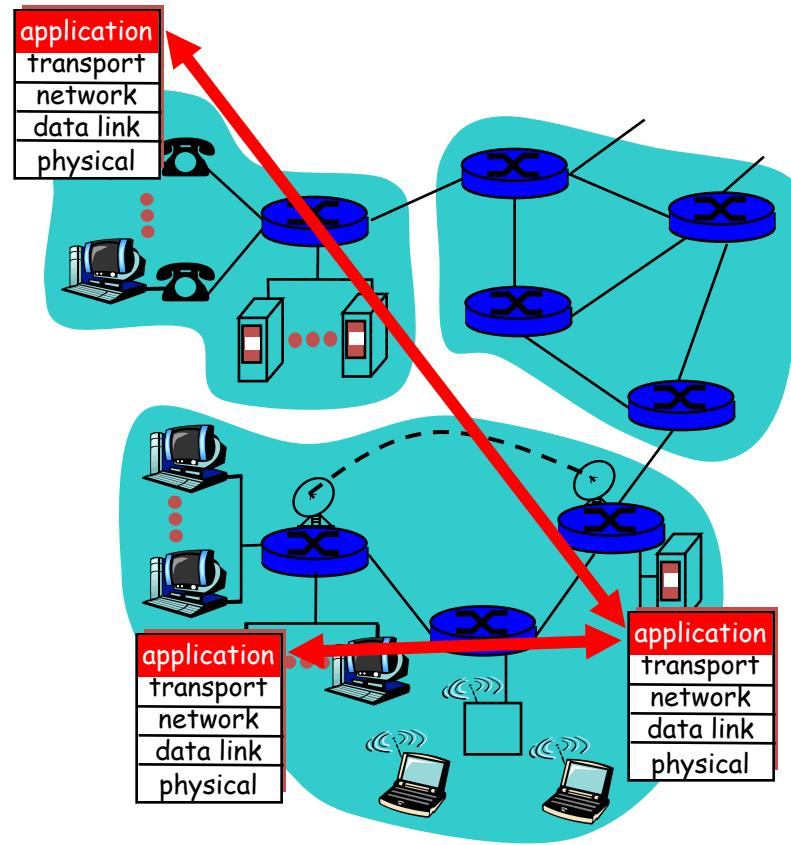


Source: www.statista.com (2016)



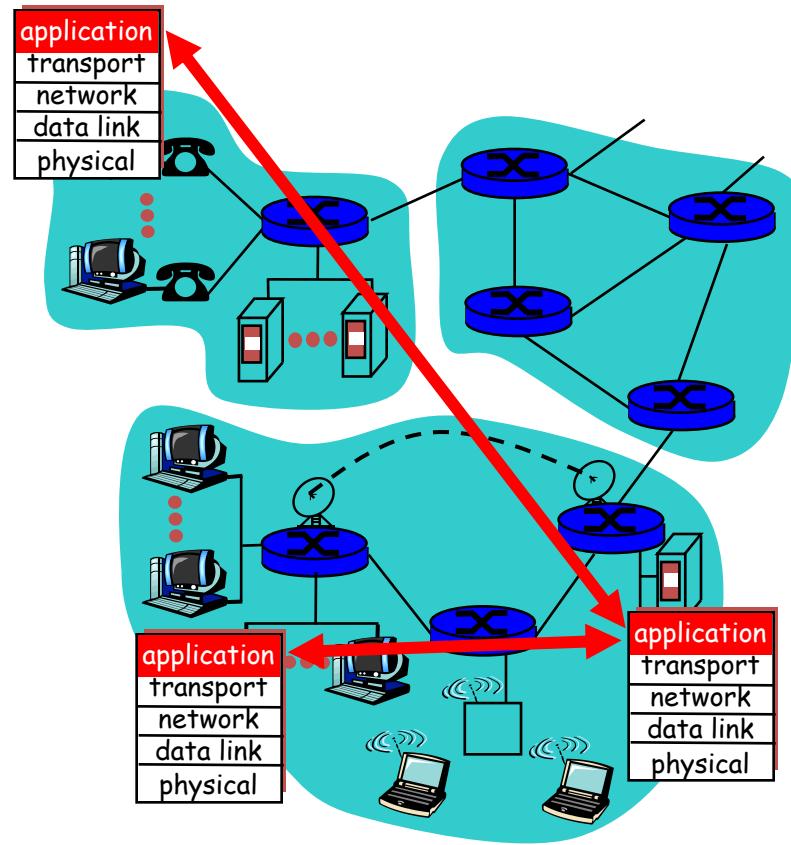
Applicazione di rete

- L'applicazione è un software che:
 - possa essere eseguito su diversi terminali e
 - possa comunicare tramite la rete
- Esempio: il browser web (*FireFox, Safari, Chrome, ecc..*) è un software “in esecuzione” su un dispositivo che comunica con un software in esecuzione su un server web (www.google.com, www.amazon.com, ecc..)



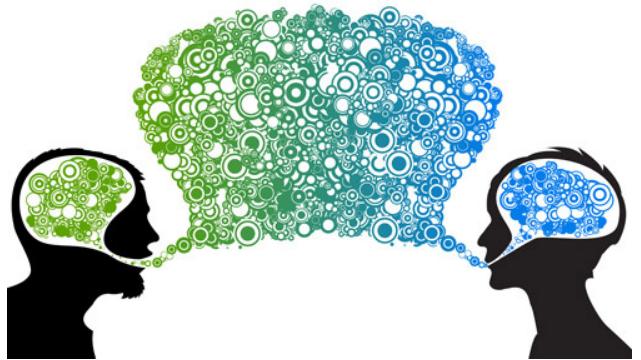
Applicazione di rete

- Inventare una nuova applicazione non richiede di cambiare il software della rete
- I nodi della rete non hanno software applicativo
- Le applicazioni sono solo nei terminali e possono essere facilmente sviluppate e diffuse



Comunicazione tra processi

- **Host:** dispositivo d'utente
 - Laptop, smartphone, desktop
- **Processo:** programma software in esecuzione su un *host*
 - Molti processi possono essere in esecuzione simultaneamente sullo stesso *host*



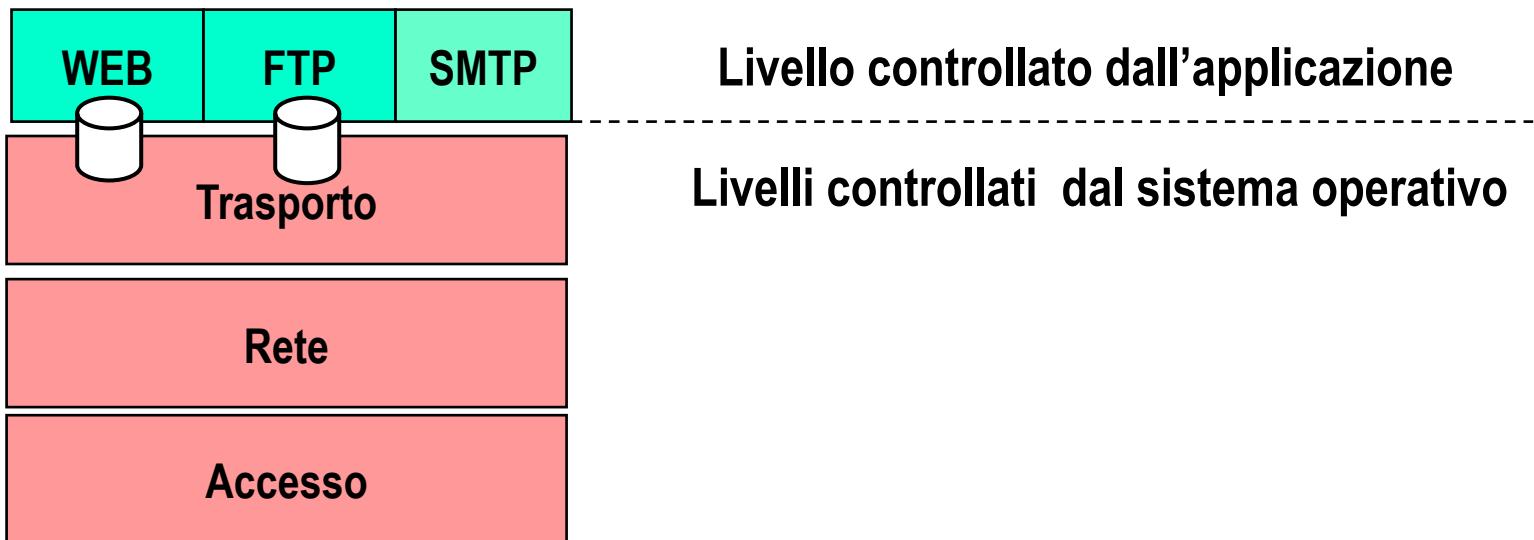
Ingredienti per la comunicazione tra processi remoti

- Indirizzamento dei processi (conoscere il “numero di telefono” dell’interlocutore)
- Protocollo di scambio dati (decidere la “lingua” con cui si parla”
 - Tipi di messaggi scambiati: Richieste, risposte
 - Sintassi dei messaggi: Campi del messaggio e delimitatori
 - Semantica dei messaggi: Significato dei campi
 - Regole su come e quando inviare e ricevere i messaggi



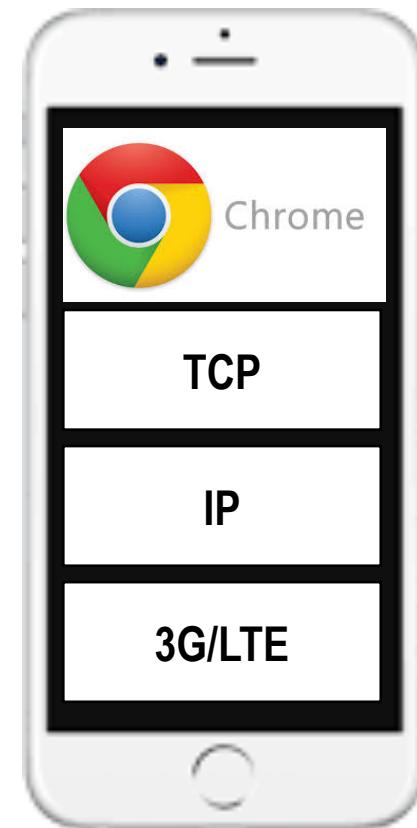
Indirizzamento di processi applicativi

- Lo scambio di messaggi fra i processi applicativi avviene utilizzando i servizi dei livelli inferiori attraverso i SAP (*Service Access Point*)
- Ogni processo è associato ad un SAP



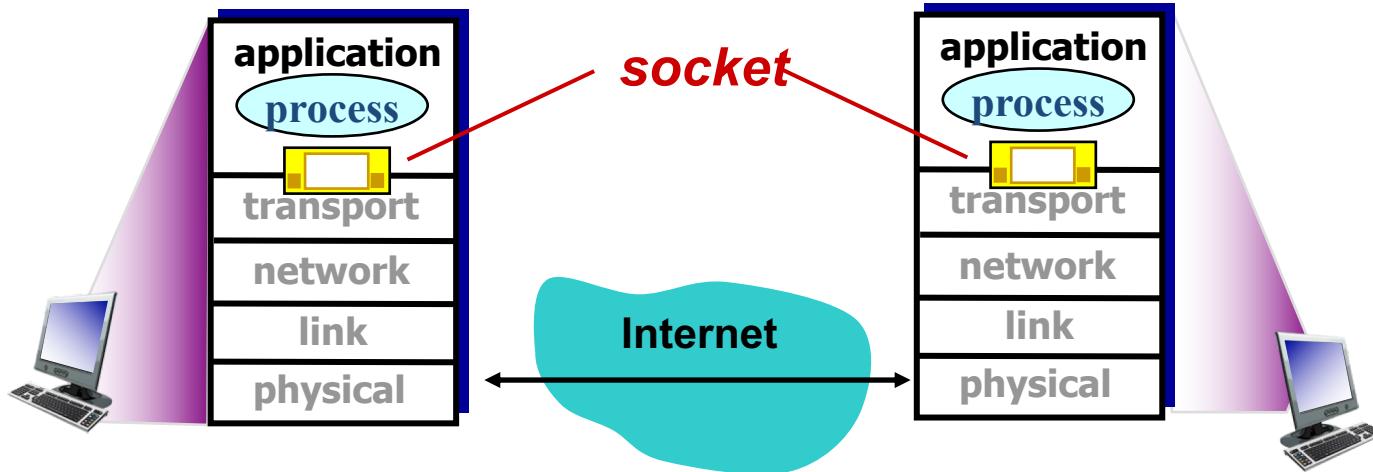
Indirizzamento di processi applicativi

- Cosa serve per identificare il mio browser *Chrome* in esecuzione sul mio *host*?
 - Indirizzo del mio *host*
 - Indirizzo IP univoco di 32 bit (di cui parleremo ampiamente in seguito)
 - Indirizzo del SAP del processo in esecuzione sul mio *host*
 - numero di porta



Indirizzamento di processi applicativi

- Indirizzo di un processo in esecuzione = indirizzo IP + numero di porta = **socket**
- Esempio:
 - Il server web del Politecnico www.polimi.it è raggiungibile a: 131.175.12.34/80
- Le **socket** sono delle porte di comunicazione
 - Il processo trasmittente mette il messaggio fuori dalla porta
 - La rete raccoglie il messaggio e lo trasporta fino alla porta del destinatario



Requisiti delle applicazioni

- Perdita di dati
 - Alcune applicazioni possono tollerare perdite parziali (ad es. audio)
 - Altre applicazioni richiedono a completa affidabilità (ad es. file transfer)
- Ritardo
 - Alcune applicazioni richiedono basso ritardo (ad es. Internet telephony, interactive games)
- Banda
 - Alcune applicazioni richiedono un minimo di velocità di trasferimento (ad es. appl. multimediali)
 - Altre applicazioni si adattano alla velocità disponibile (“appl. elastiche”)



Requisiti delle 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: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100msec
instant messaging	no loss	elastic	yes and no

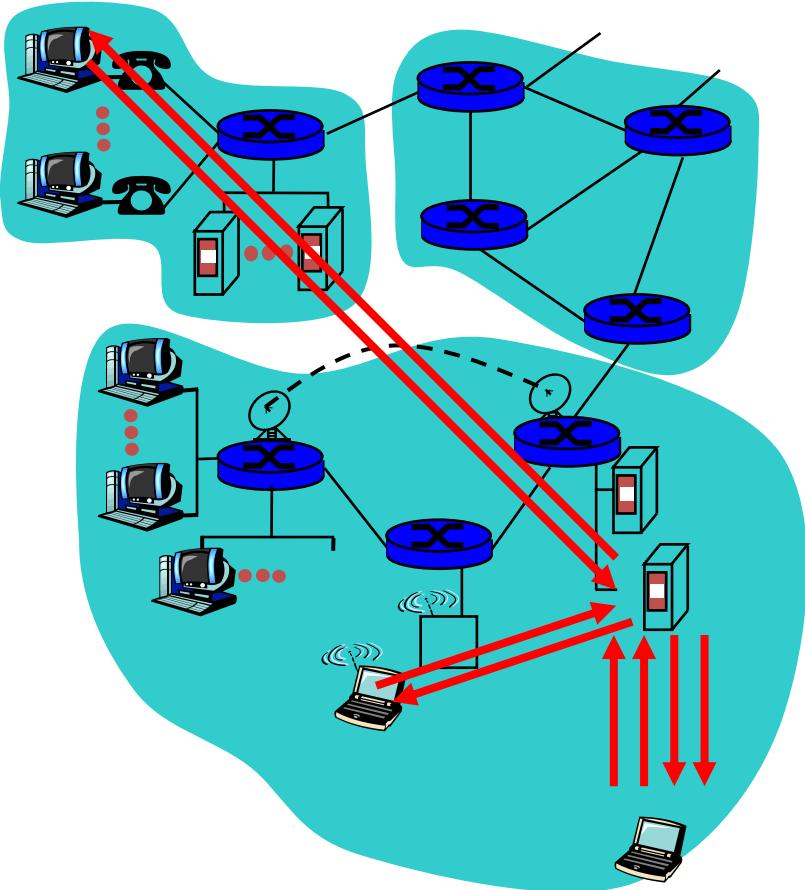


Architetture applicative

- ***Client-server***
 - I dispositivi coinvolti nella comunicazione implementano o solo il processo *client* o solo il processo *server*
 - I dispositivi *client* e *server* hanno caratteristiche diverse
 - I *client* possono solo eseguire richieste
 - I *server* possono solo rispondere a richieste ricevute
- ***Peer-to-peer (P2P)***
 - I dispositivi implementano tutti sia il processo *client* che quello *server*
- ***Irida***



Architettura *client-server*



Server:

- Host sempre attivo
- Indirizzo IP permanente
- Possibilità di utilizzo di macchine in *cluster*
- Possono ricevere richieste da molti *client*

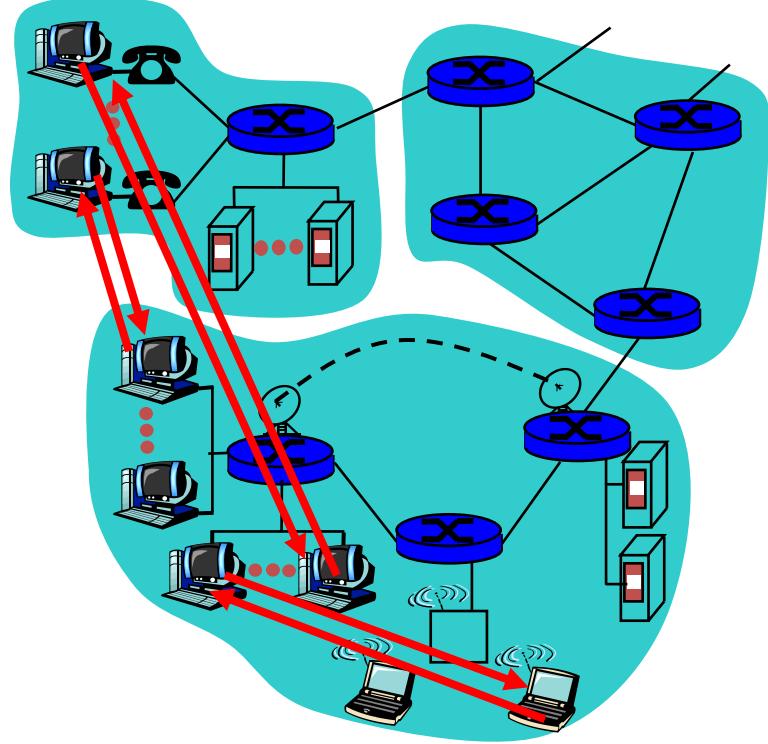
Client:

- Comunicano con il server
- Possono essere connessi in modo discontinuo
- Possono cambiare indirizzo IP
- Non comunicano con altri client
- Possono inviare molte richieste allo stesso server



Architettura P2P (pura)

- Non ci sono server sempre connessi
- Terminali (*peers*) comunicano direttamente
- I *peers* sono collegati in modo intermittente e possono cambiare indirizzo IP
- Esempio: *BitTorrent*



Fortemente scalabile ma difficile da gestire





Il servizio di Web Browsing

Hyper Text Transfer Protocol (HTTP)
RFC 1945, 2616

Cosa contengono i messaggi HTTP – le Pagine Web

- Breve ripasso:
 - le *pagine web* sono fatte di *oggetti*
 - gli *oggetti* possono essere file HTML file, immagini JPEG, applet Java, file audio file, file video, collegamenti ad altre pagine web..
 - generalmente le pagine web hanno un file HTML (o php) base che “chiama” gli altri *oggetti*
 - ogni *oggetto* è indirizzato da una *Uniform Resource Locator (URL)*, es:

http://www.polimi.it:80/index.html

↑
Indica il protocollo applicativo

↑
Indica l'indirizzo di rete del server

↑
Indica il numero di porta (opzionale)

↑
Indica la pagina web richieste



La comunicazione HTTP

- Architettura *client/server*:
 - **client**: browser che effettua richieste HTTP di pagine web (oggetti), le riceve e le mostra all'utente finale
 - **server**: Web server inviano gli oggetti richiesti tramite risposte HTTP
- Nessuna memoria sulle richieste viene mantenuta nei server sulle richieste passate ricevute da un client (protocollo *stateless*)



La comunicazione HTTP

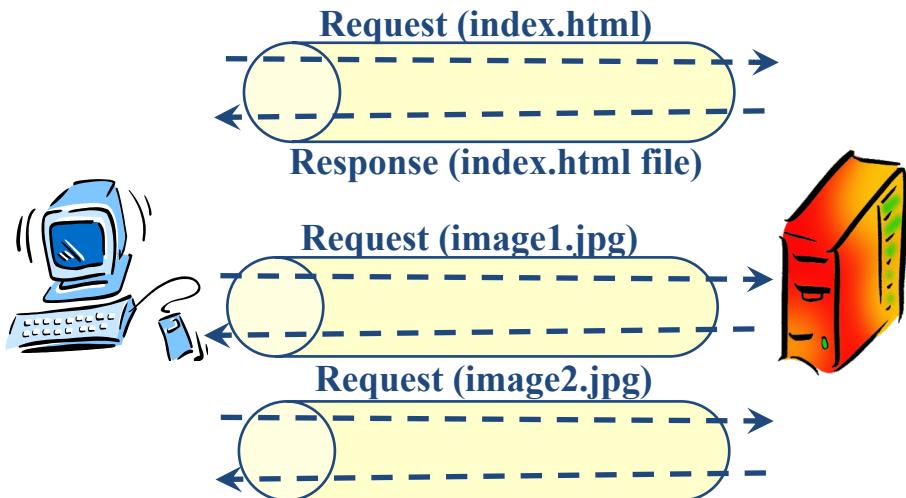
- HTTP si appoggia su TCP a livello di trasporto:
 1. Il client HTTP inizia una connessione TCP verso il server (porta 80)
 2. Il server HTTP accetta connessioni TCP da client HTTP
 3. ***Client e Server HTTP si scambiano informazioni (pagine web e messaggi di controllo)***
 4. La connessione TCP tra client e server HTTP viene chiusa



Modalità di connessione tra *client* e *server* HTTP

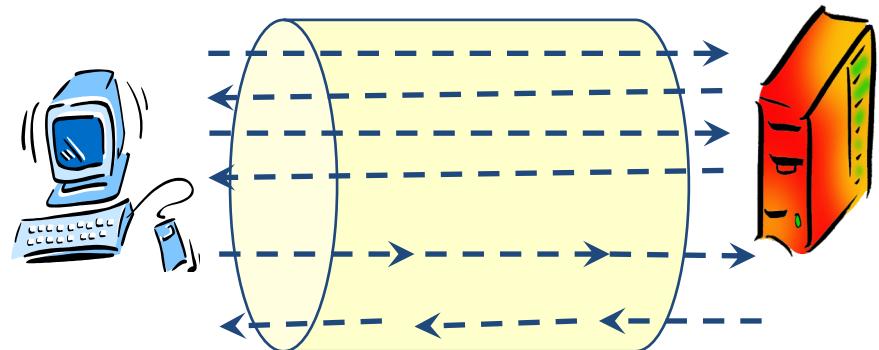
Connessione non persistente

- Una connessione TCP per una sola sessione richiesta-risposta; inviato l'oggetto il server chiude la connessione TCP
- La procedura viene ripetuta per tutti i file collegati al documento HTML base
- Le connessioni TCP per più oggetti possono essere aperte in parallelo per minimizzare il ritardo



Connessione persistente

- La connessione TCP rimane aperta e può essere usata per trasferire più oggetti della stessa pagina web o più pagine web
 - *without pipelining*: richieste HTTP inviate in serie
 - *with pipelining*: richieste HTTP inviate in parallelo



Esempio di connessione non persistente

L'utente digita nel browser la URL:

www.polimi.it/home/index.html

(l'HTML contiene testo e riferimenti a 10
immagini jpeg)

1a. Il client HTTP inizia una connessione TCP verso il server HTTP www.polimi.it sulla porta 80

2. il client HTTP invia una richiesta HTTP (contenente la URL) tramite la connessione TCP. La richiesta indica che il client vuole l'oggetto /home/index.html

0. il server HTTP in esecuzione su www.polimi.it in attesa sulla porta 80

1b. accetta la connessione e notifica il client

3. Il server HTTP riceve la richiesta HTTP ed invia una risposta HTTP contenente il file HTML

tempo



POLITECNICO MILANO 1863

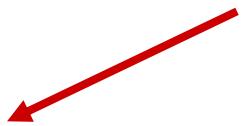
Fondamenti di TELECOMUNICAZIONI

Esempio di connessione non persistente

tempo
↓

4. Il server HTTP chiude la connessione TCP.

5. Il client HTTP riceve il messaggio di risposta contenente il file html e lo visualizza. Si scopre dei collegamenti a 10 oggetti JPEG



I passi da 1 a 5 sono ripetuti per ognuna delle 10 immagini JPEG indicate dal file HTML

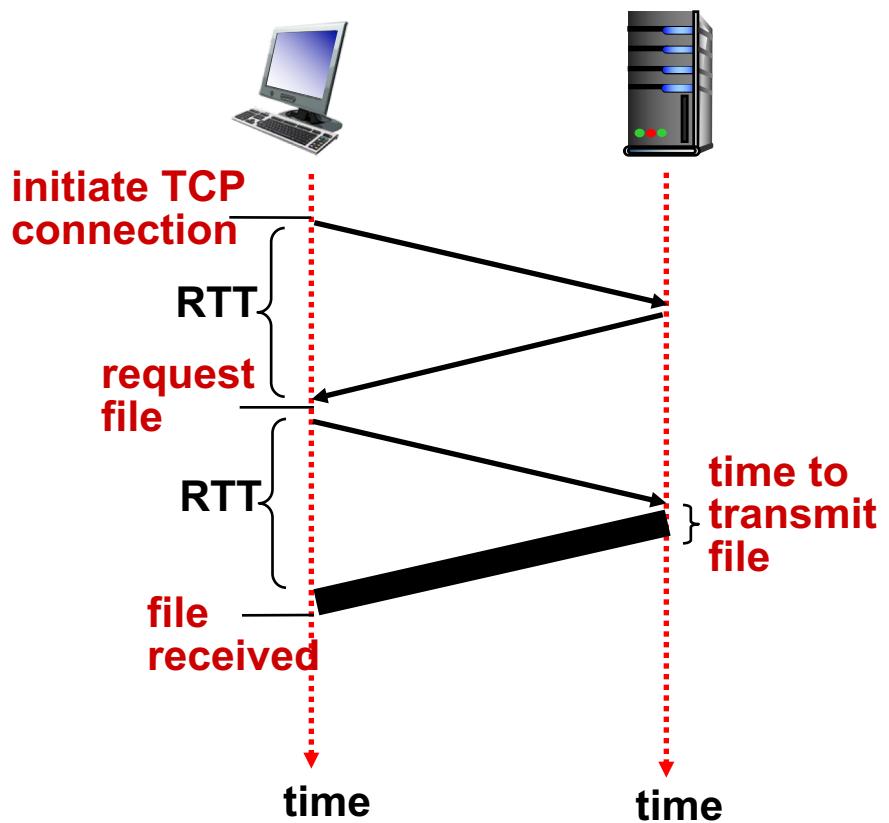


Stima del tempo di trasferimento in HTTP

- *Round trip Time (RTT)*: tempo per trasferire un messaggio “piccolo” dal *client* al *server* e ritorno
- Tempo di risposta di HTTP:
 - un RTT per iniziare la connessione TCP
 - un RTT per inviare i primi byte della richiesta HTTP e ricevere i primi byte di risposta
 - tempo di trasmissione dell’oggetto (file HTML, immagine, ecc..)
- Supponendo che la pagina web sia composta da 11 oggetti (un file HTML e 10 immagini JPEG), il tempo di download dell’intera pagina web è:

$$T_{nonpers} = \sum_{i=0}^{10} (2RTT + T_i)$$

$$T_{pers} = RTT + \sum_{i=0}^{10} (RTT + T_i)$$



Mantenere uno “stato” in HTTP: i cookies

Ingredienti dei cookies:

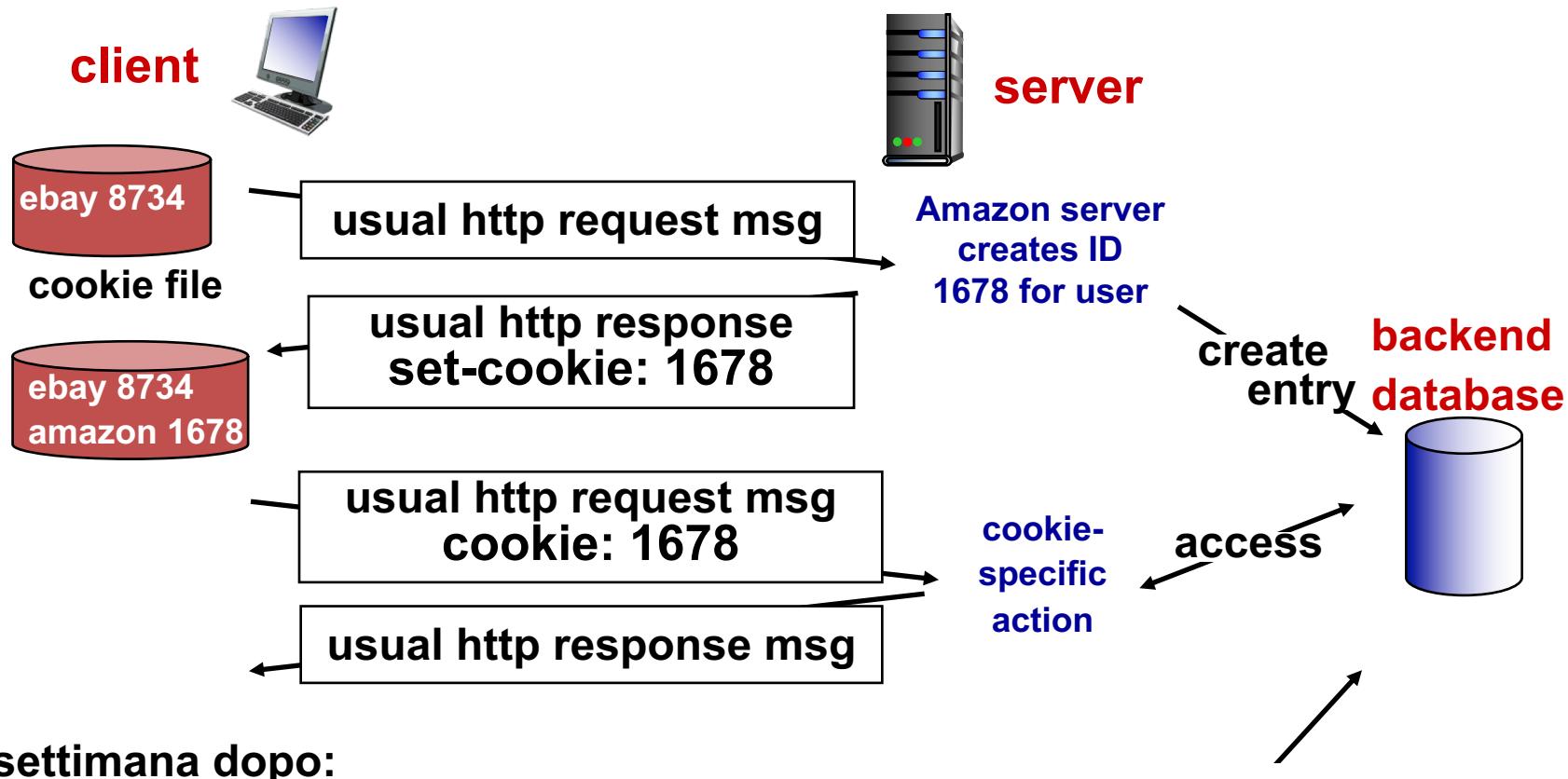
- 1) Un header “cookie” nelle risposte HTTP
- 2) Un header “cookie” nelle successive richieste HTTP
- 3) Una lista di cookie mantenuta sull’host (dal browser)
- 4) Un data base di cookie mantenuto dal sito web

esempio:

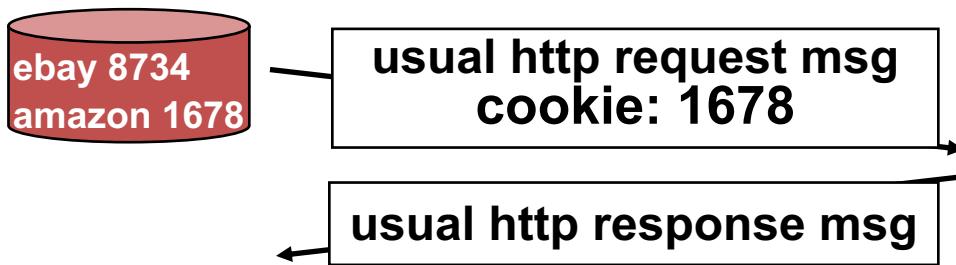
- Matteo visita un sito di e-commerce per la prima volta
- Quando il sito riceve la prima richiesta HTTP, il sito genera:
 - un ID unico
 - una entry nel data base locale che corrisponde all’ID creato



Esempio di uso dei cookie

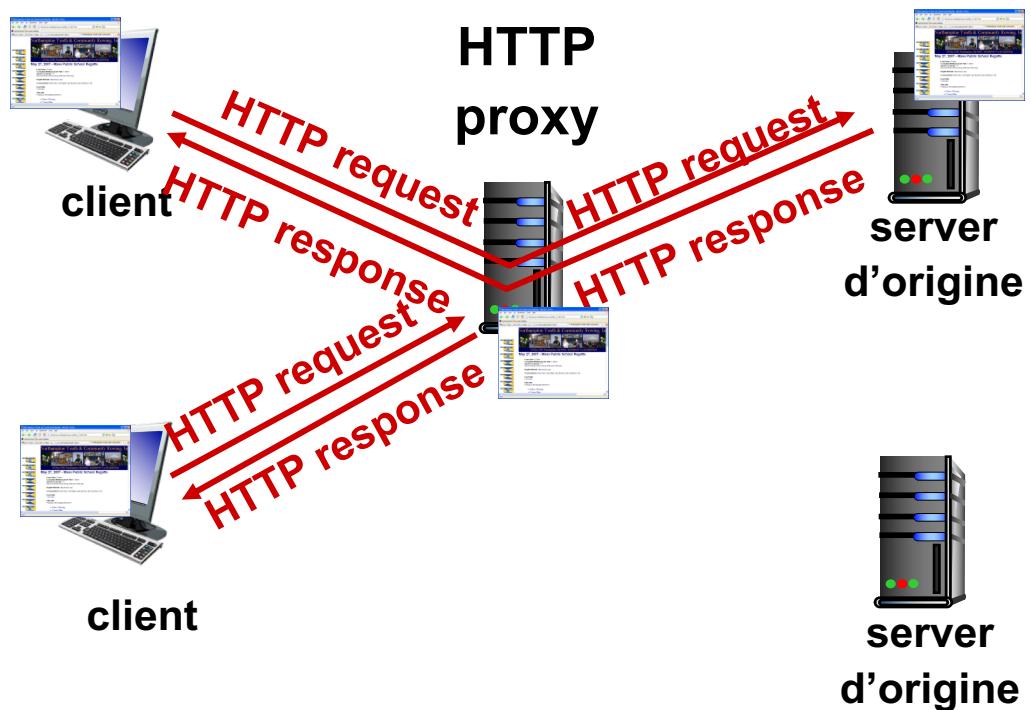


una settimana dopo:



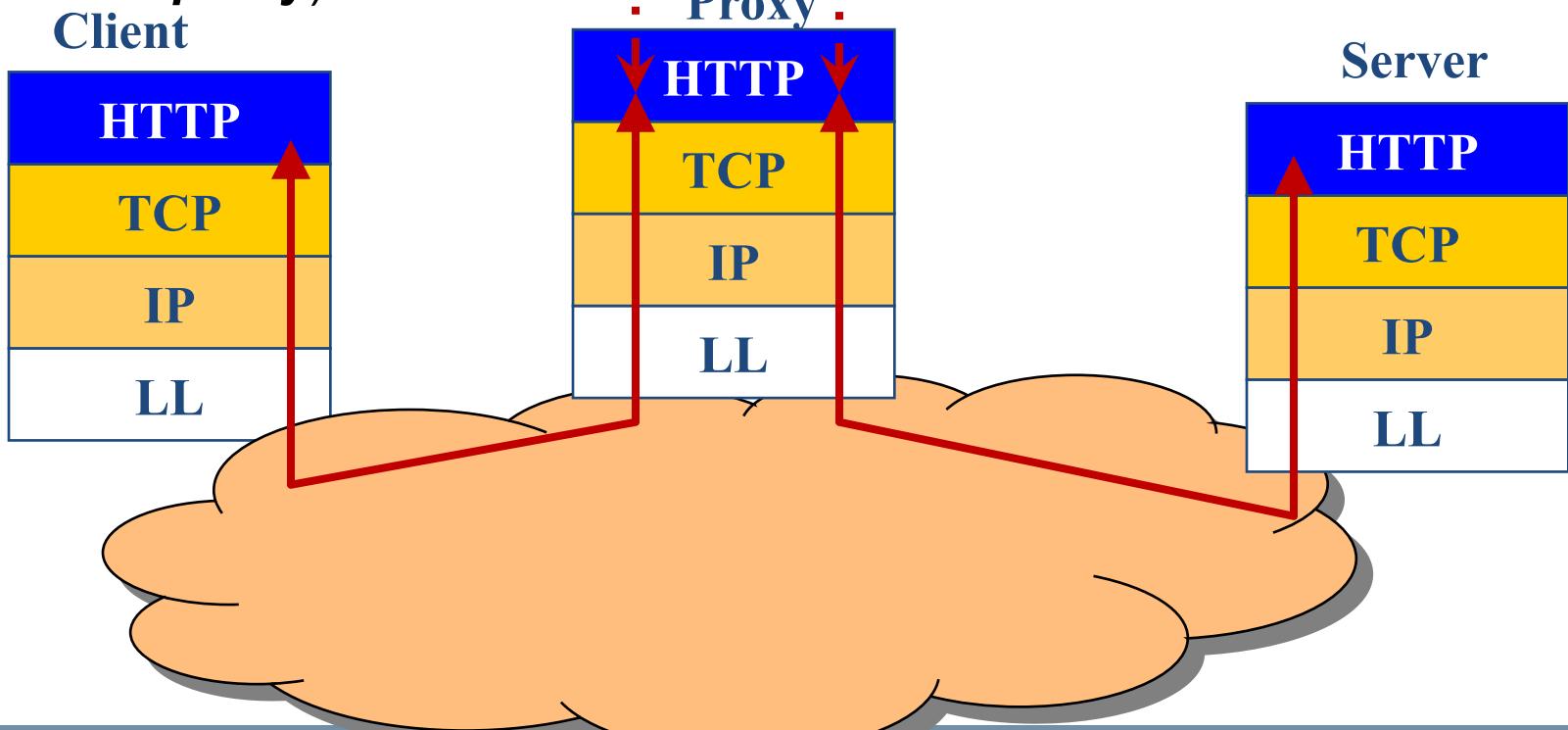
I proxy HTTP: Cache di rete

- Obiettivo: **rispondere alle richieste HTTP senza coinvolgere il server HTTP**
 - Il client HTTP invia **tutte** le richieste HTTP ad un *proxy* HTTP:
 - se l'oggetto richiesto è disponibile nella cache del proxy server, il proxy server risponde con l'oggetto
 - altrimenti il proxy recupera l'oggetto dal server d'origine e lo restituisce al client



I proxy HTTP: Cache di rete

- I proxy sono degli *application gateway*, ovvero degli instradatori di messaggi di livello applicativo
- Devono essere sia *client* (verso i server d'origine) che *server* (verso i *client*)
- Il server vede arrivare tutte le richieste dal proxy (**mascheramento degli utenti del proxy**)





Il servizio di posta elettronica

Simple Mail Transfer Protocol (SMTP) RFC 5321

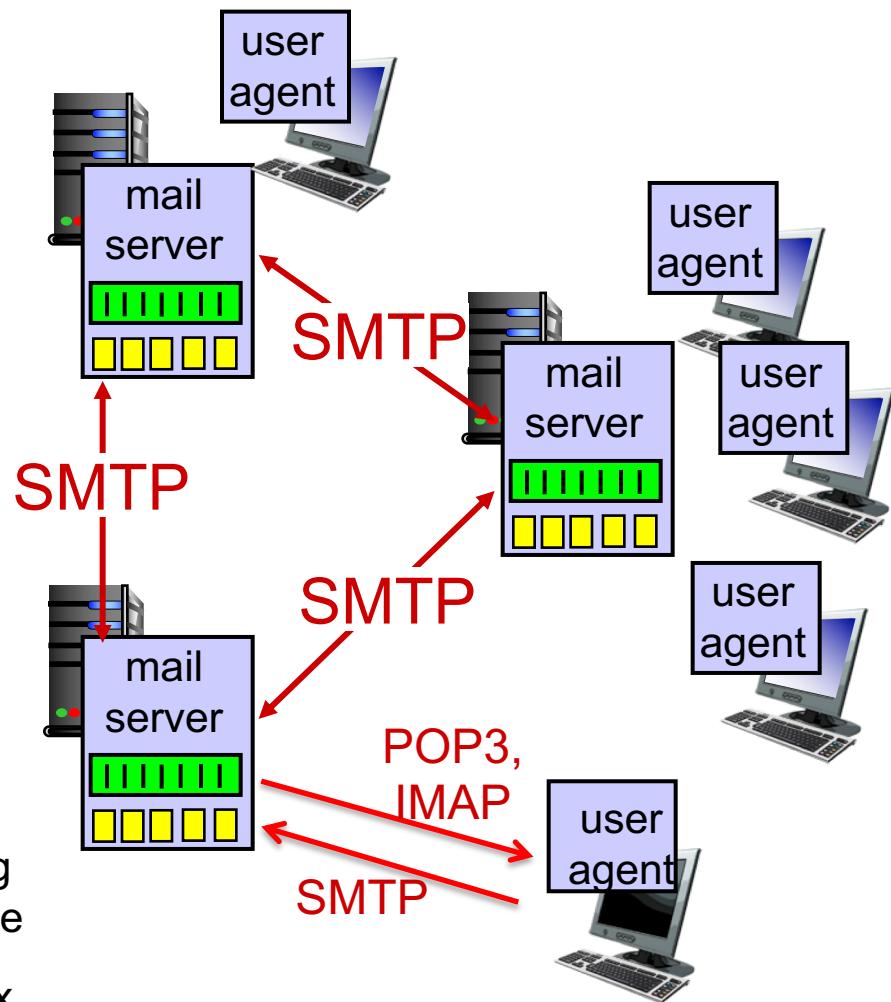
Post Office Protocol (POP3) RFC 1939

Internet Mail Control Protocol (IMAP) RFC 3501

Il servizio di E-mail

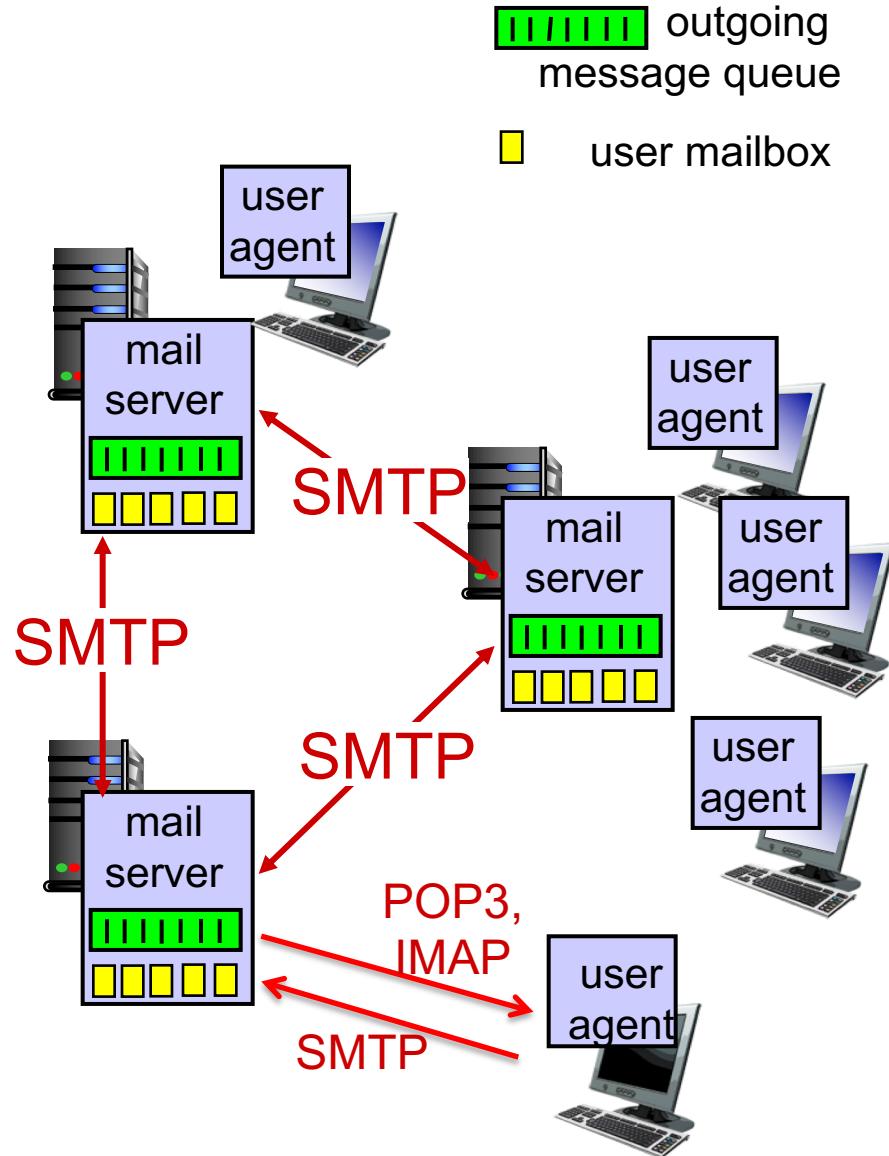
- Client d'utente aka *User Agent* (OutLook, Thunderbird, etc.)
- *Mail Server*
- *Simple Mail Transfer Protocol (SMTP)*: per trasferire email dal client d'utente fino al *mail server* del destinatario
- Protocolli di accesso ai *mail server*: per “scaricare” email dal proprio mail server (**POP3, IMAP**)

 outgoing message queue
 user mailbox



I Mail Server

- I *mail server* contengono per ogni *client* controllato:
 - una coda di email in ingresso (*mailbox*)
 - una coda di email in uscita
- I *mail server*
 - Ricevono le mail in uscita da tutti i *client* d'utente che “controllano”
 - Ricevono da altri *mail server* tutte le *mail* destinate ai *client* d'utente controllati
- I *mail server* “parlano”
 - SMTP con altri *mail server* e con i *client* d'utente in *uplink*
 - POP3/IMAP con i *client* d'utente in *downlink*



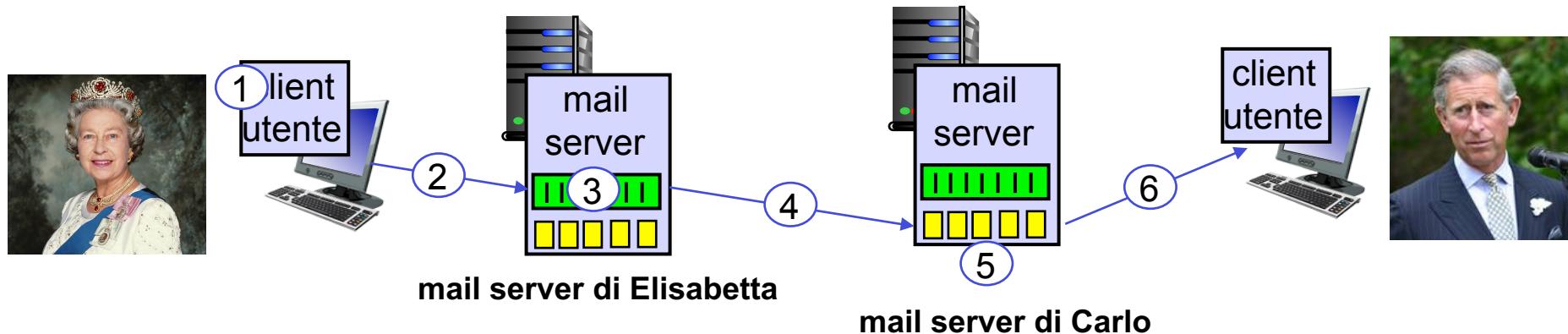
SMTP

- E' un protocollo applicativo *client-server*
- Quando un *mail server* riceve un messaggio da un client d'utente
 - mette il messaggio in una coda
 - apre una connessione TCP con la porta 25 del *mail server* del destinatario
 - trasferisce il messaggio
 - chiude la connessione TCP
- L'interazione tra *client* SMTP e *server* SMTP è di tipo comando/risposta
- Comandi e risposte sono testuali



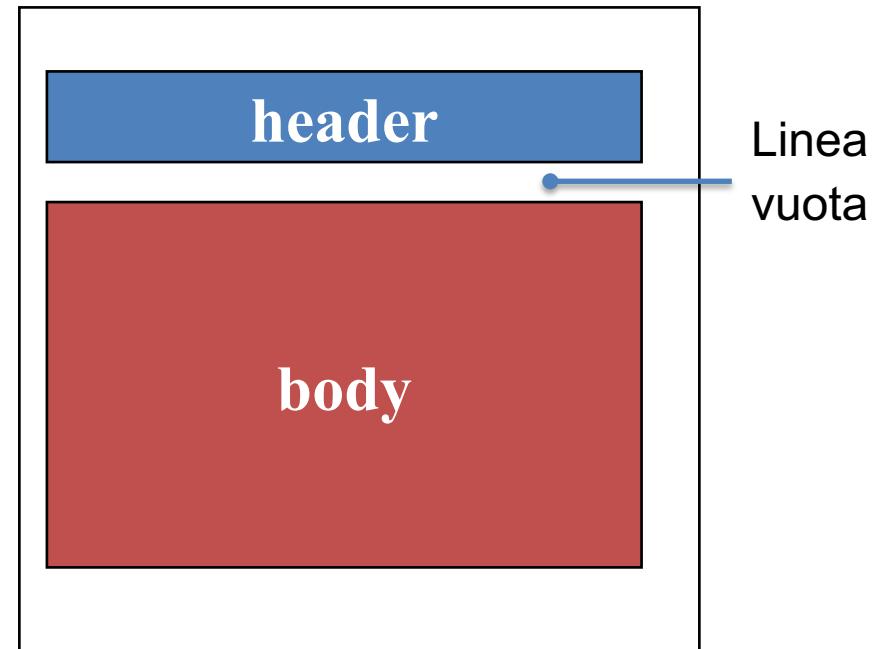
Esempio di trasferimento SMTP

1. Elisabetta compone una *email* destinata a Carlo carlo@miomailserver.com
2. Il *client d'utente* di Elisabetta invia la *mail* al proprio *mail server*
3. Il *mail server* di Elisabetta si comporta come *client SMTP* ed apre una connessione TCP (porta 25) con il *mail server* di Carlo
4. Il *client SMTP* (*mail server* di Elisabetta) invia la *email* sulla connessione TCP
5. Il *mail server* di Carlo memorizza la *mail* nella *mailbox* di Carlo
6. Carlo (in modo asincrono) usa il proprio *client d'utente* per leggere la *mail*



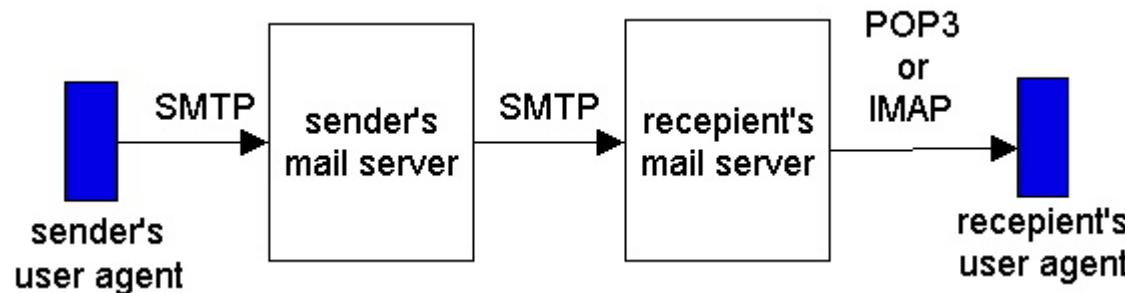
Il formato delle email (RFC 822)

- Il formato dei messaggi inviati (tutto ciò che segue il comando SMTP DATA) è specificato
- *Header:*
 - To:
 - From:
 - Subject:
- *Body:* il contenuto dell'email



Protocolli di accesso al mailbox

- Diversi protocolli sono stati sviluppati per il colloquio tra *user agent* e *server* in fase di lettura dei messaggi presenti nel *mailbox*
 - **POP3** (*Post Office Protocol versione 3, RFC 1939*): download dei messaggi
 - **IMAP** (*Internet Mail Access Protocol, RFC 1730*): download dei messaggi, gestione della *mailbox* sul *mail server*





POLITECNICO
MILANO 1863



Risoluzione di nomi simbolici

Domain Name System (DNS)

- Gli indirizzi IP (32 bit) sono poco adatti ad essere usati dagli applicativi
- E' più comodo utilizzare indirizzi simbolici:
 - E' meglio www.google.com o 74.125.206.99?
- Occorre una mappatura fra indirizzi IP (usati dalle macchine di rete) ed i nomi simbolici (usati dagli esseri umani)

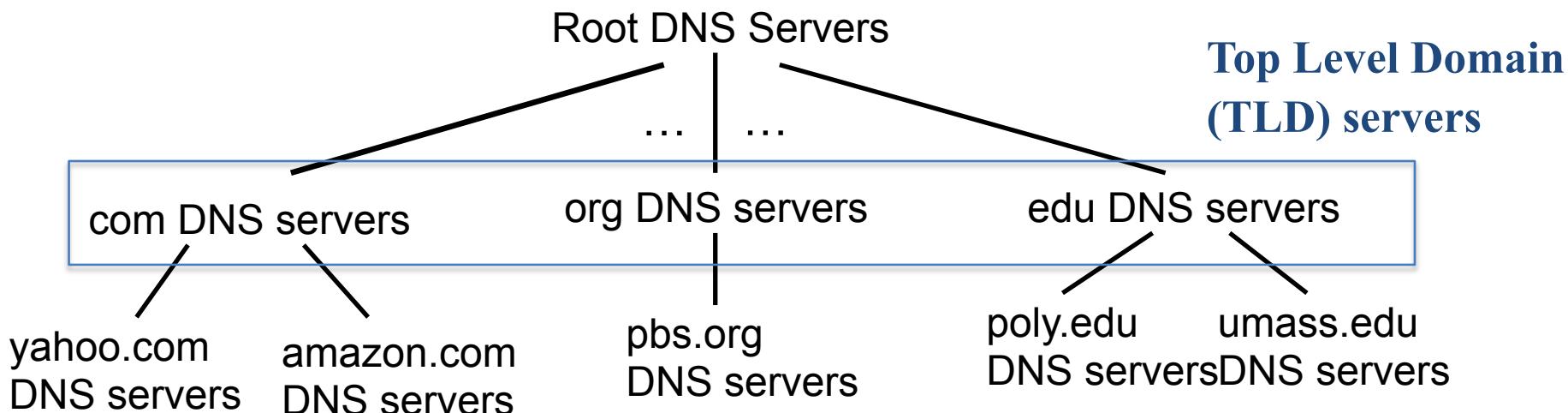


Domain Name System (DNS)

- Ingredienti
 - Database distribuito costituito da molti *name servers* con organizzazione gerarchica
 - Protocollo applicativo tra *name server* e *host* per **risolvere** nomi simbolici (tradurre nomi simbolici in indirizzi IP)



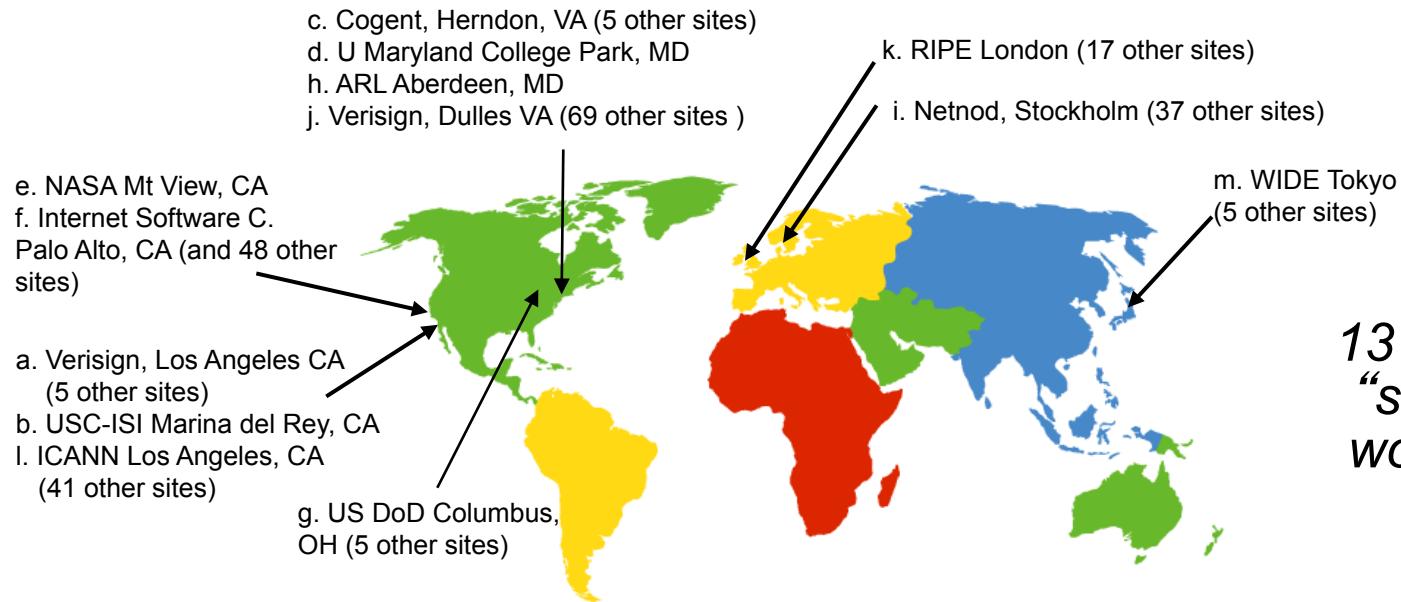
Database distribuito e gerarchico



- Ogni livello nella gerarchia ha diversa “profondità” di informazione
- Esempio: un utente vuole l’IP di `www.google.com`
 - *Root name server sanno come “trovare” i name server di che gestiscono i domini .com*
 - *I name server .com sanno come trovare i name server che gestiscono il dominio google.com*
 - *I name server google.com sanno risolvere il nome simbolico www.google.com*



Root NS



*13 root name
“servers”
worldwide*



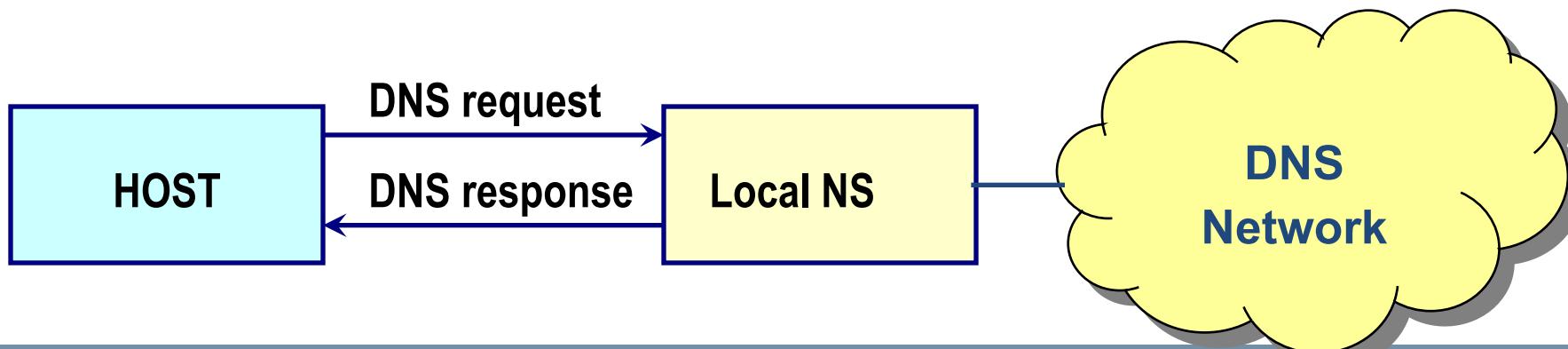
Altri tipi di NS

- *Local Name Servers*
 - Ogni ISP (residenziale, università, compagnia) ha un NS locale
 - Direttamente collegati agli host
 - Tutte le volte che un host deve risolvere un indirizzo simbolico contatta il Local Name Server
 - Il Local Name Server (eventualmente) contatta i Root Name Server nella gerarchia



Come ottenere un mappaggio

- Ogni *host* ha configurato l'indirizzo del LNS
- Le applicazioni che richiedono un mappaggio (browser, ftp, etc.) usano le funzioni del DNS
- Una richiesta viene inviata al server DNS
- Il server reperisce l'informazione e restituisce la risposta



Caching

- Un *server*, dopo aver reperito un'informazione su cui non è *authoritative* può memorizzarla temporaneamente
- All'arrivo di una nuova richiesta può fornire l'informazione senza risalire sino al *server authoritative*
- Il TimeToLive (TTL) è deciso dal *server authoritative* ed è un indice di quanto stabile nel tempo è l'informazione relativa
- I TLD *server* sono generalmente “memorizzati” nei *Local Name Servers*
- I *server non-authoritative* usano il TTL per decidere un time-out



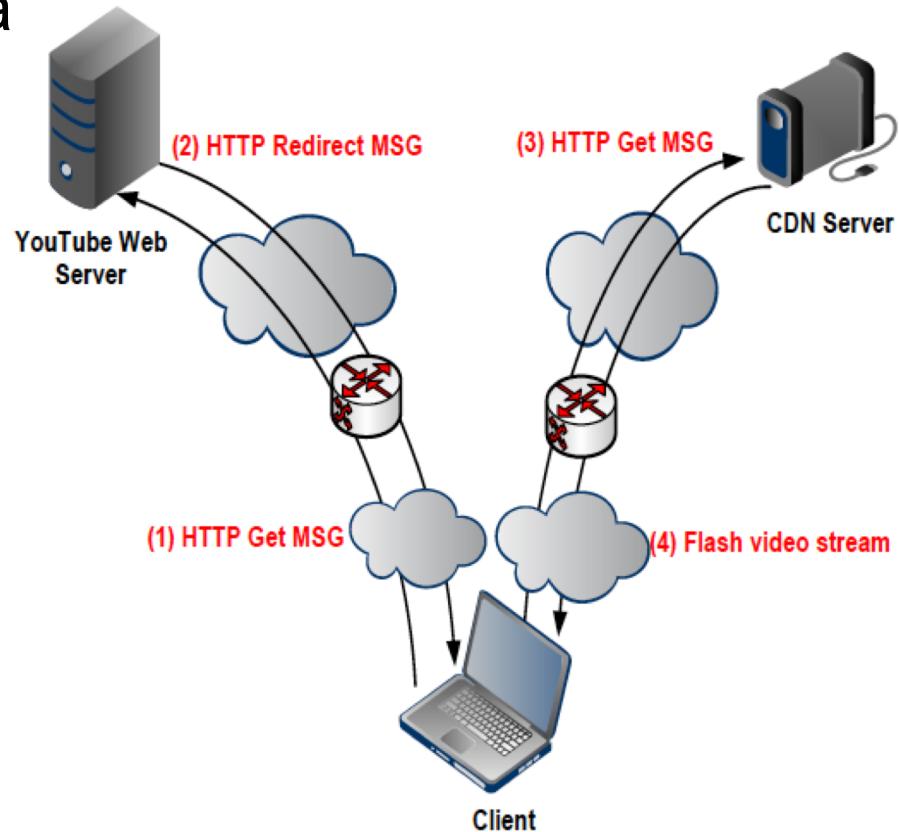
Come aggiungere un dominio alla “rete” DNS

- Una nuova startup *I-Like-Networking* vuole registrare il dominio *I-Like-Networking.com* (supponiamo che il dominio sia disponibile)
- *I-Like-Networking* registra il dominio presso uno dei **DNS Registrars**
 - *I-Like-Networking* deve fornire al **DNS registrar** i nomi simbolici ed i relativi indirizzi IP dei name server autoritativi
 - Il DNS registrar inserisce due record nel server .com
 - I-Like-Networking.com, dns1.I-Like-Networking.com
 - dns1.I-Like-Networking.com, 212.212.212.1



HTTP e *video streaming*: YouTube

- Architettura di YouTube si basa su:
 - HTTP per la distribuzione di contenuti video
 - Una (o più) *Content Distribution Network (CDN)* per lo *storage* distribuito dei contenuti video





POLITECNICO
MILANO 1863



Applicazioni Peer-to-Peer

P2P file sharing

- Gli utenti utilizzano il software P2P sul proprio PC
- Si collegano in modo intermittente a Internet
- Se un utente cerca un file l'applicazione trova altri utenti che lo hanno
- L'utente sceglie da chi scaricarlo
- Il file è scaricato usando un protocollo come HTTP
- Altri utenti potranno (in seguito o in contemporanea) scaricare il file dall'utente
- L'applicazione P2P è sia client che server.

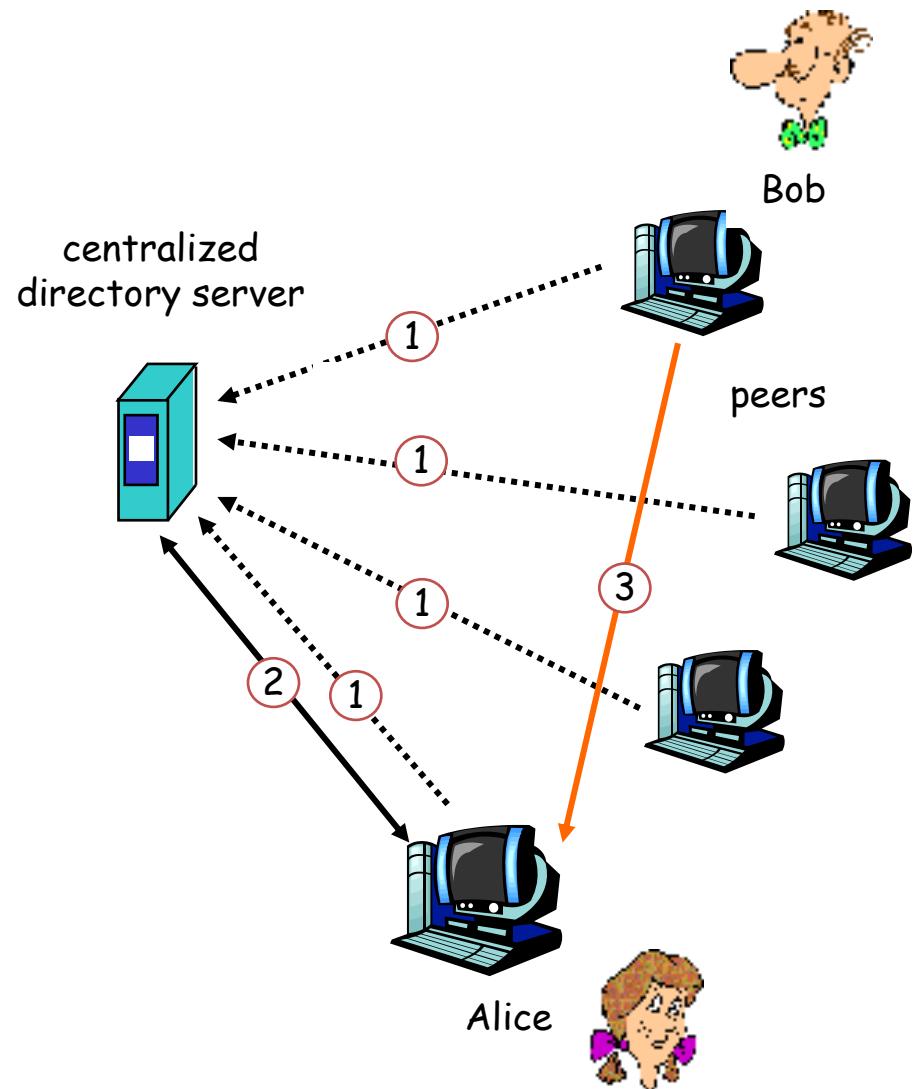
Elevata scalabilità!



P2P: directory centralizzata

Meccanismo di “Napster”

- 1) Quando i peer si connettono, informano il server centrale:
 - Indirizzo IP
 - File condivisi
- 2) Il peer interroga il server centrale per uno specifico file
- 3) Il file viene scaricato direttamente



P2P: directory centralizzata

- Problemi dell'architettura centralizzata
 - Se il server si rompe il sistema si blocca
 - Il server è un collo di bottiglia per il sistema
 - Chi gestisce il server può essere accusato di infrangere le regole sul *copyright*

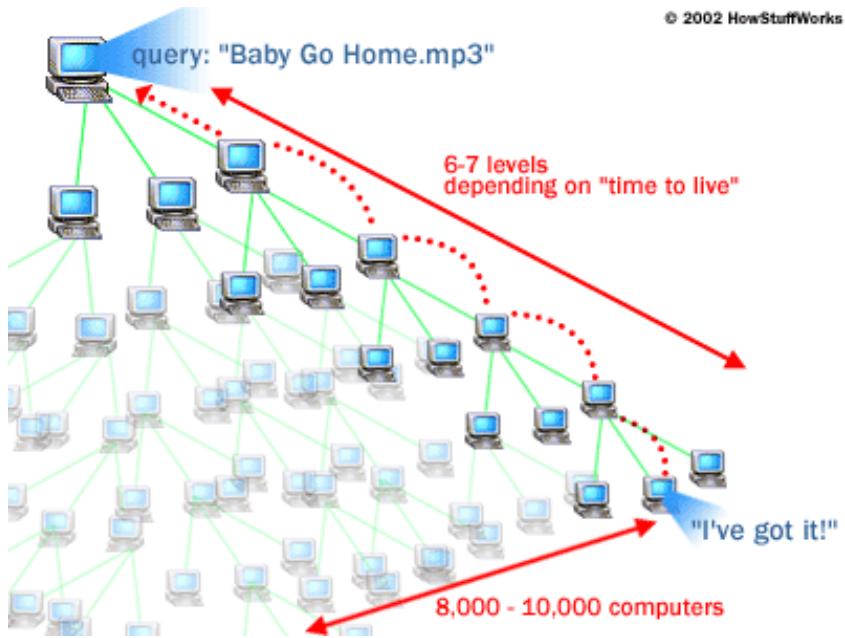
Il trasferimento file è distribuito, ma la ricerca dei contenuti è fortemente centralizzata



P2P: completamente distribuita

Meccanismo di *Gnutella*

- Nessun server centrale
- Protocollo di pubblico dominio
- Molti software diversi basati sullo stesso protocollo



Basato su una rete (grafo) di *overlay*

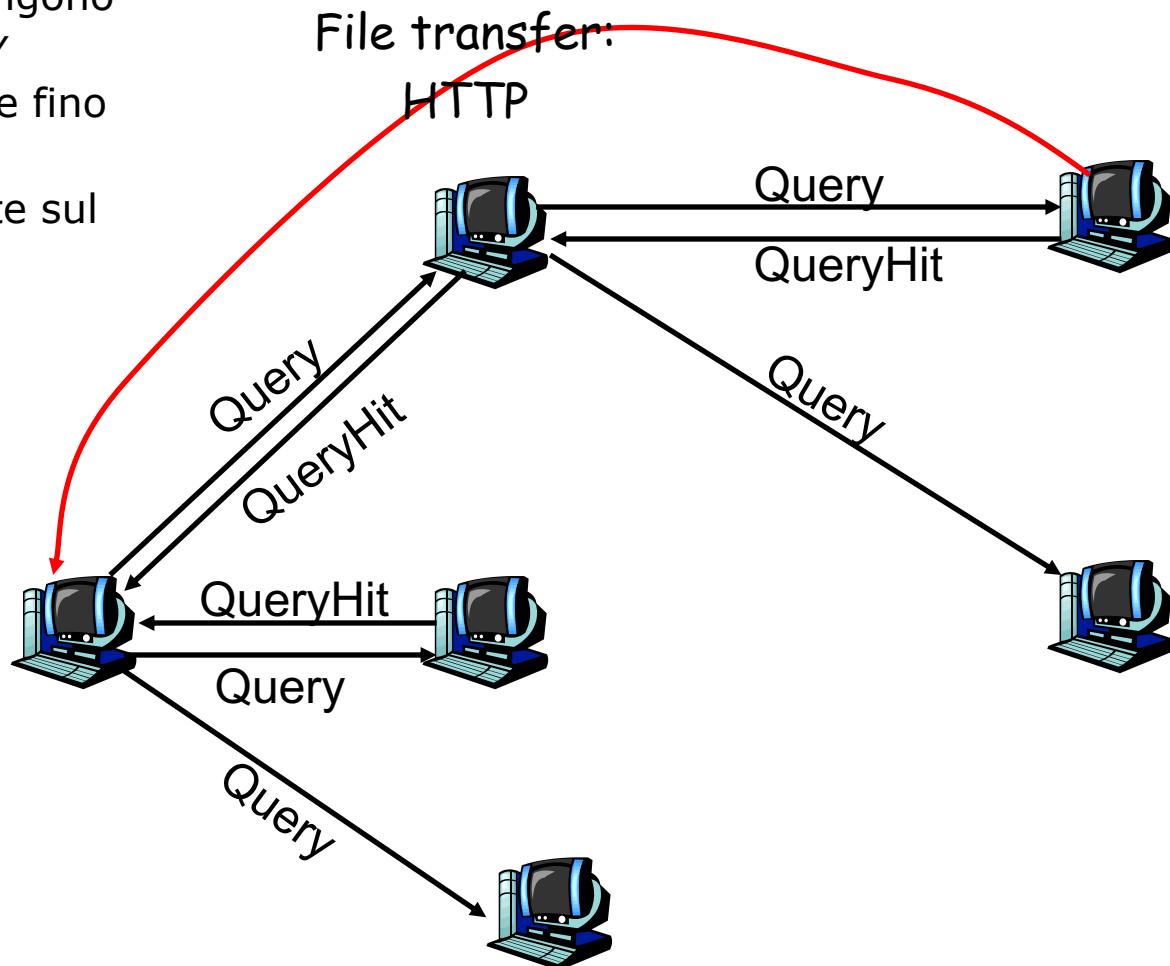
- I *peer* si attivano e si collegano ad un numero (<10) di altri vicini
- La ricerca dei vicini è distribuita
- I vicini nella rete *overlay* possono essere fisicamente distanti



P2P: completamente distribuita

Ricerca di un file

- I messaggi di richiesta vengono diffusi sulla rete di *overlay*
- I *peer* inoltrano le richieste fino a una certa distanza
- Le risposte vengono inviate sul cammino opposto



P2P: completamente distribuita

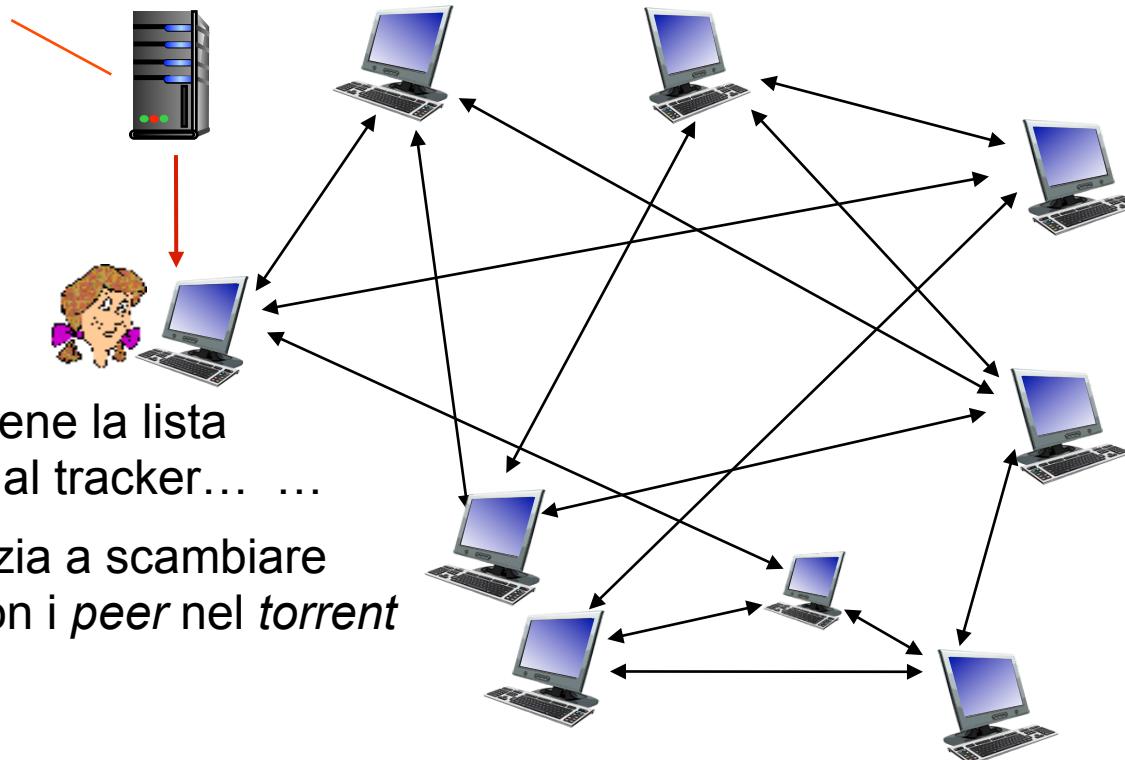
- Accesso alla rete
 - Per iniziare il processo di accesso il *peer* X deve trovare almeno un altro *peer*; la ricerca si basa su liste note
 - X scandisce la lista fino a che un *peer* Y risponde
 - X invia un messaggio di Ping a Y; Y inoltra il Ping nella rete di *overlay*.
 - Tutti i *peer* che ricevono il *Ping* rispondono a X con un messaggio di *Pong*
 - X riceve molti messaggi di *Pong* e può scegliere a chi connettersi aumentando il numero dei suoi vicini nella rete di *overlay*



BitTorrent

- I file sono divisi in *chunk* di 256kbyte

tracker: tiene traccia dei peer che partecipano ad un torrent

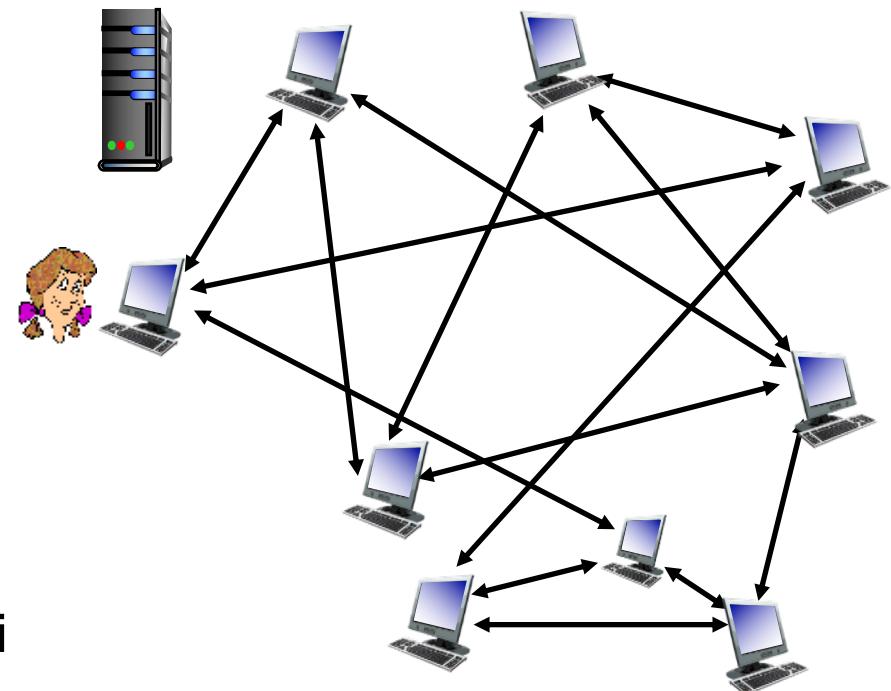


torrent: gruppo di peer che si scambiano chunk di un file



BitTorrent – entrare nel *torrent*

- I *peer* che entrano in un *torrent* si registrano presso un *tracker* per ottenere una lista di peer “attivi”
- Il *tracker* invia una lista di peer attivi su un *torrent* (indirizzi IP)
- Il *peer* entrante stabilisce connessioni TCP con un sottoinsieme dei *peer* nella lista (*neighboring peers*)
- I *neighboring peers* inviano al *peer* entrante la lista dei *chunk* disponibili
- Il *peer* entrante sceglie quale *chunk* scaricare e da quale *peer* scaricare *chunk* secondo meccanismi euristici



Meccanismo di richiesta di *chunk*

- Il principio del *Rarest First*
 - Il peer entrante, tra tutti i *chunk* mancanti, scarica prima i *chunk* più rari nelle liste di *chunk* ricevute da tutti i *neighboring peer*

