

Reti (Computer Networks)

Capitolo 2 Il livello applicazione

Docente: Paolo Casari

TA: Andrea Rosani

Capitolo 2: Il livello applicazione

- 2.1 Principi delle applicazioni di rete
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Posta elettronica
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Applicazioni P2P
- 2.7 Cloud Computing
- 2.8 Programmazione delle socket (cenni)

Capitolo 2: Il livello applicazione

Obiettivi:

- Fornire i concetti base e gli aspetti implementativi dei protocolli delle applicazioni di rete
 - ❖ Modelli di servizio del livello di trasporto
 - ❖ Paradigma client-server
 - ❖ Paradigma peer-to-peer
- Apprendere informazioni sui protocolli esaminando quelli delle più diffuse applicazioni di rete
 - ❖ HTTP (1.0, 1.1, 2.0)
 - ❖ FTP
 - ❖ SMTP / POP3 / IMAP
 - ❖ DNS
- Programmare le applicazioni di rete
 - ❖ Socket API

Internet e le Applicazioni

2021 *This Is What Happens In An Internet Minute*



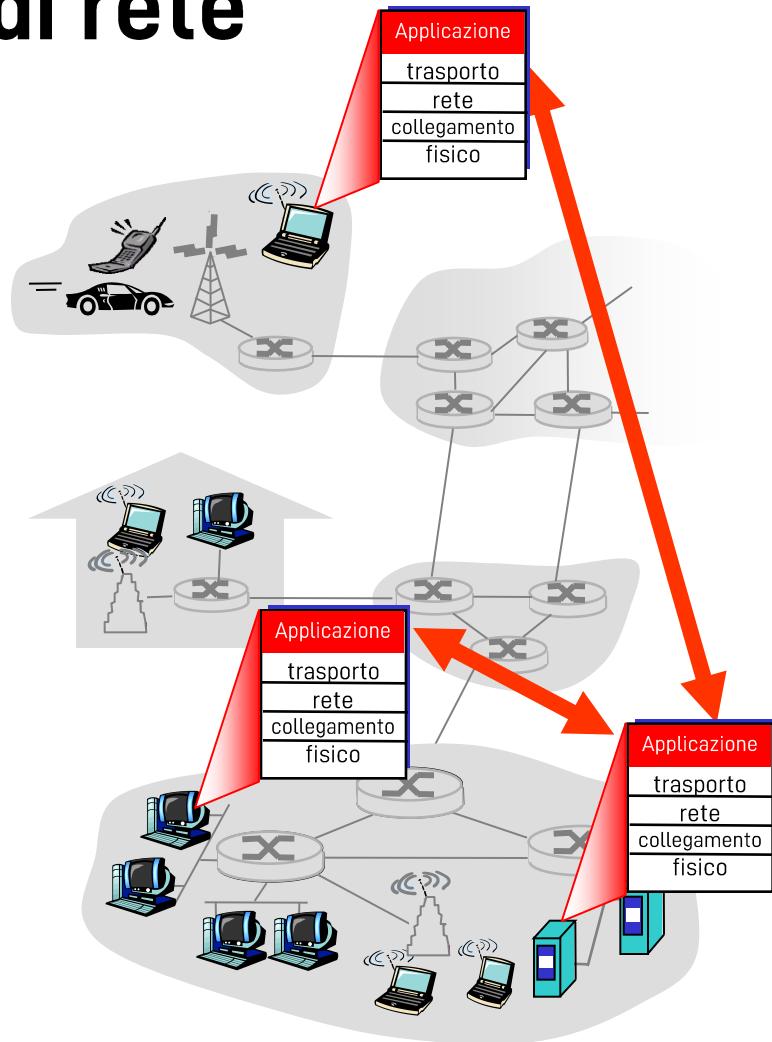
Creare un'applicazione di rete

Scrivere programmi che

- ❖ Girano su sistemi terminali diversi
- ❖ Comunicano attraverso la rete
- ❖ Ad es. il software di un server Web comunica con il software di un browser

Software in grado di funzionare
su più macchine

- ❖ Non occorre predisporre
programmi per i dispositivi del
nucleo della rete, quali router o
commutatori Ethernet



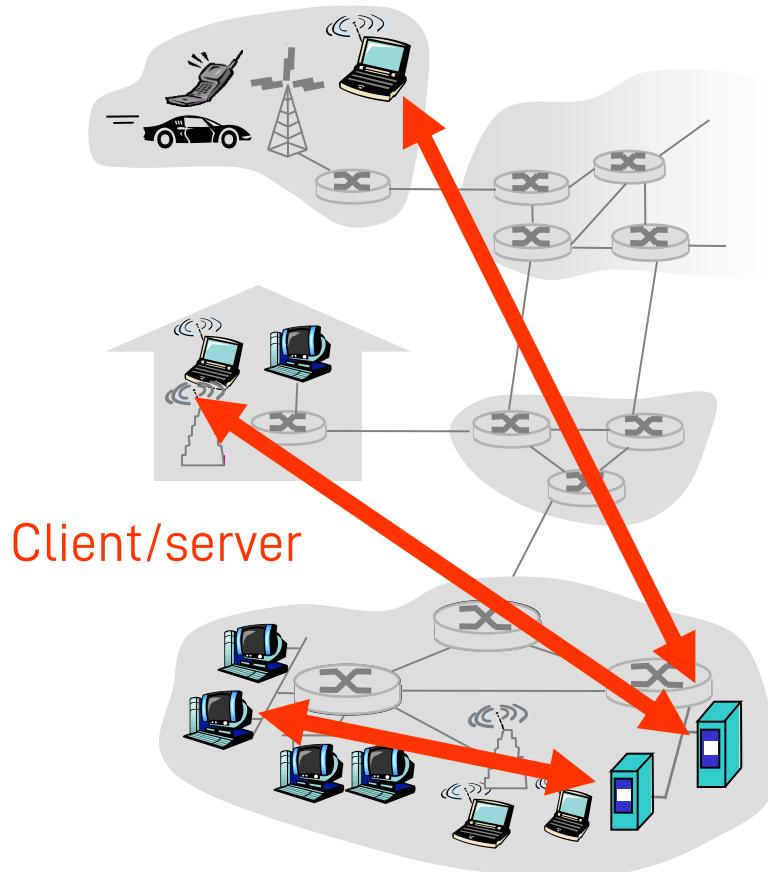
Capitolo 2: Livello applicazione

- 2.1 Principi delle applicazioni di rete
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Posta elettronica
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Applicazioni P2P
- 2.7 Cloud Computing
- 2.8 Programmazione delle socket

Architetture delle applicazioni di rete

- Client-server
- Peer-to-peer (P2P)
- Architetture ibride (client-server e P2P)
- Cloud computing

Architettura client-server



□ Server

- Host sempre attivo
- Indirizzo permanente
- Come scalare?

□ Client

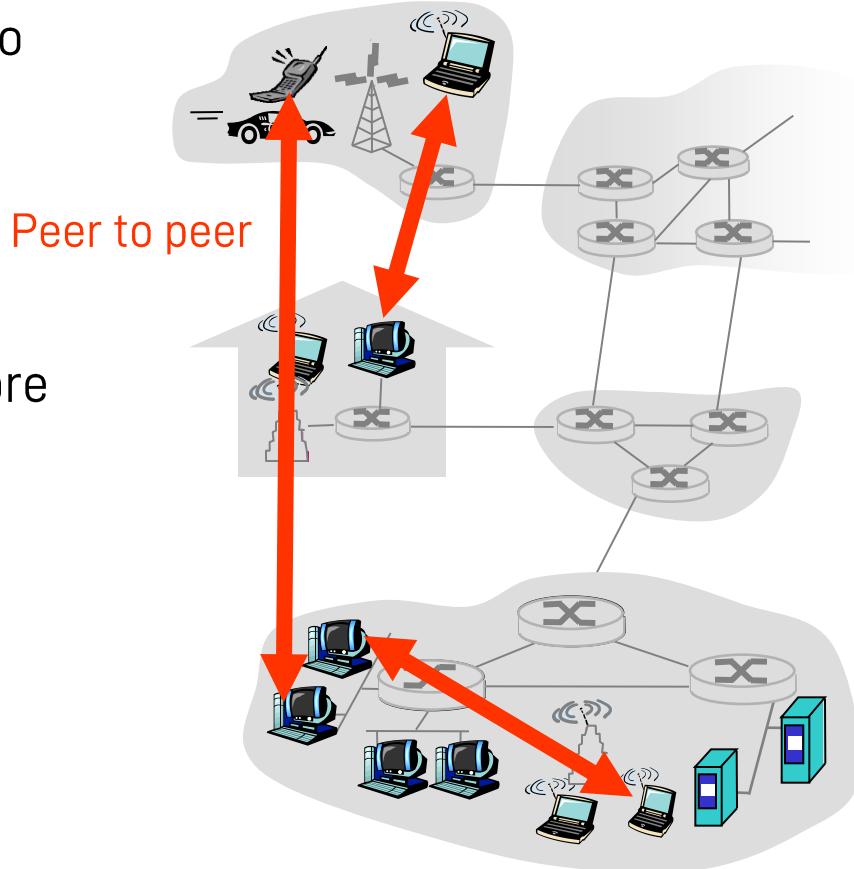
- Comunica con il server
- Può disconnettersi temporaneamente
- Può avere un indirizzo dinamico
- Non comunica direttamente con altri client

Architettura P2P pura

- Non c'è un server sempre attivo
- Coppie arbitrarie di host (peer) comunicano direttamente tra loro
- I peer non devono necessariamente essere sempre attivi, e cambiano indirizzo IP

Facilmente scalabile

Difficile da gestire



Ibridi (client-server e P2P)

Skype

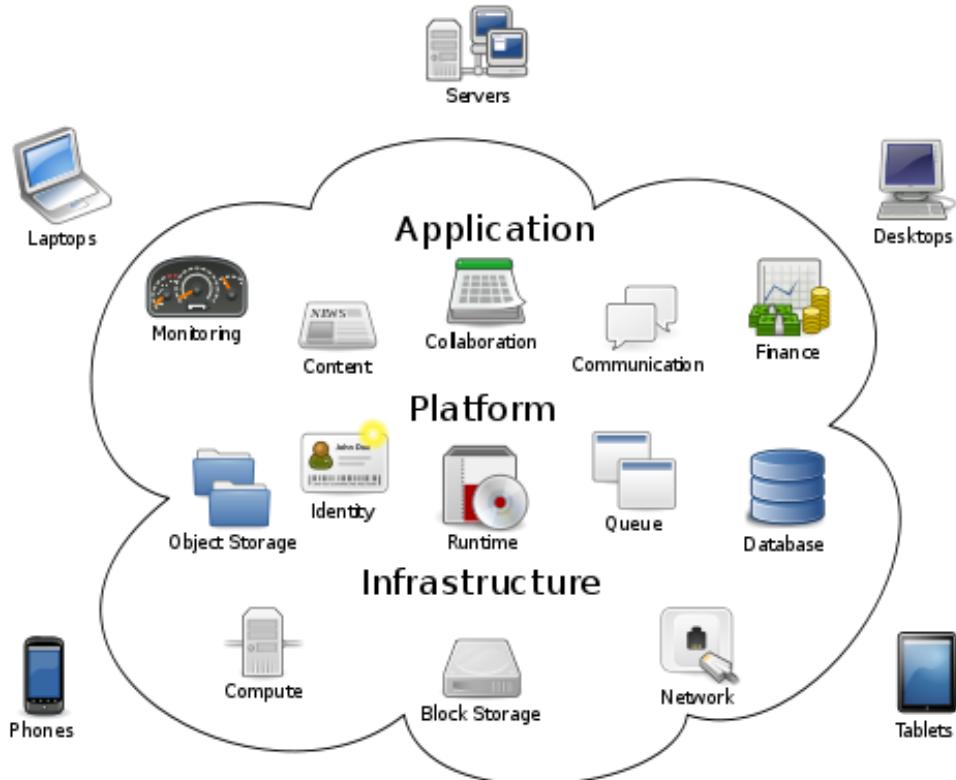
- ❖ Applicazione P2P di Voice over IP
- ❖ Server centralizzato: ricerca indirizzi della parte remota
- ❖ Connessione client-client: diretta (non attraverso il server)

Messaggistica istantanea

- ❖ La chat tra due utenti è del tipo P2P
- ❖ Individuazione della presenza/location centralizzata:
 - l'utente registra il suo indirizzo IP sul server centrale quando è disponibile online
 - l'utente contatta il server centrale per conoscere gli indirizzi IP dei suoi amici

Cloud computing

- Insieme di tecnologie che permettono **di memorizzare, archiviare e/o elaborare dati** (con CPU o software) tramite l'utilizzo di risorse distribuite e virtualizzate in rete
- La creazione di una copia di sicurezza (backup) è automatica e l'operatività si trasferisce tutta **online**
- I dati sono memorizzati in *server farm* generalmente localizzate nei Paesi di origine del service provider





There is no cloud
it's just someone else's computer

Processi del sistema operativo in comunicazione

Processo: programma in esecuzione su di un host

- All'interno dello stesso host, due processi comunicano utilizzando **schemi interprocesso** (definiti dal SO)
- Processi su host differenti comunicano attraverso lo scambio di **messaggi**

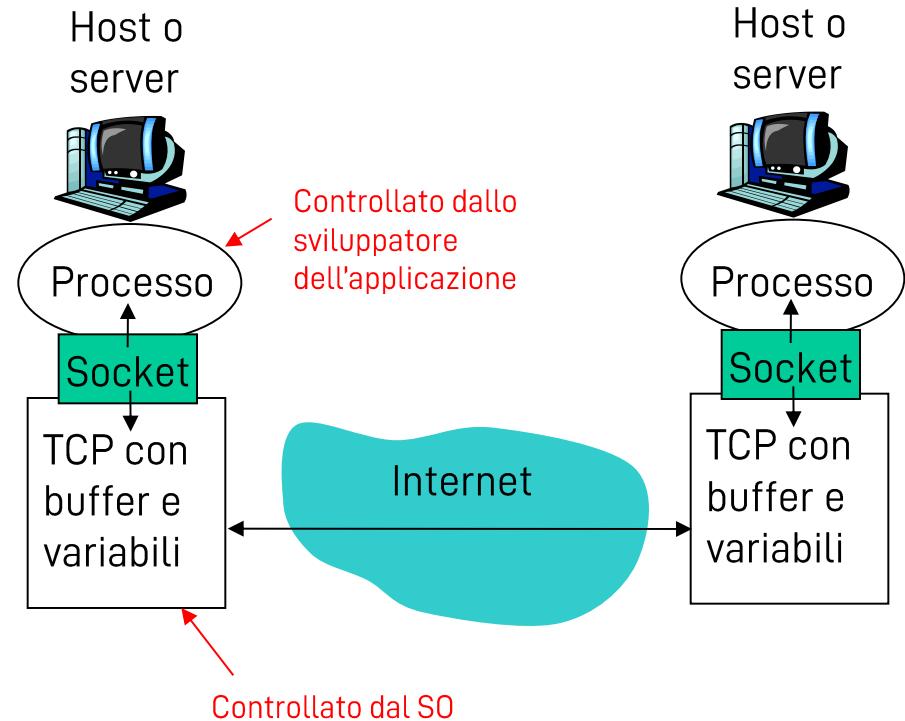
Processo client: processo che dà inizio alla comunicazione

Processo server: processo che attende di essere contattato

- Nota: le applicazioni con architetture P2P hanno processi client e processi server

Socket

- Un processo invia/riceve messaggi a/da la sua **socket**
- Una socket è analoga a una porta
 - ❖ Un processo che vuole inviare un messaggio, lo fa uscire dalla propria "porta" (socket)
 - ❖ Il processo presuppone l'esistenza di un'infrastruttura esterna che trasporterà il messaggio attraverso la rete fino alla "porta" del processo di destinazione
- API: (1) scelta del protocollo di trasporto; (2) capacità di determinare alcuni parametri (**approfondiremo questo aspetto più avanti**)



Indirizzamento

- ❑ Affinché un processo su un host invii un messaggio a un processo su un altro host, il mittente deve identificare il processo destinatario
- ❑ Un host ha un indirizzo (**detto indirizzo IP**) univoco di 32 bit
- ❑ **D:** È sufficiente conoscere l'indirizzo IP dell'host su cui è in esecuzione il processo per identificare il processo stesso?
- ❑ **Risposta:** No, sullo stesso host possono essere in esecuzione molti processi.
- ❑ L'identificatore comprende sia l'indirizzo IP che i **numeri di porta** associati al processo in esecuzione su un host
- ❑ Esempi di numeri di porta:
 - ❖ HTTP server: 80
 - ❖ Mail server: 25
- ❑ Per inviare un messaggio HTTP al server **gaia.cs.umass.edu**:
 - ❖ **Indirizzo IP:** 128.119.245.12
 - ❖ **Numero di porta:** 80

Protocolli a livello applicazione

- Tipi di messaggi scambiati, ad esempio messaggi di richiesta e di risposta
- Sintassi dei tipi di messaggio: quali sono i campi nel messaggio e come sono descritti
- Semantica dei campi, ovvero significato delle informazioni nei campi
- Regole per determinare quando e come un processo invia e risponde ai messaggi

Protocolli di pubblico dominio:

- Definiti nelle RFC della IETF
- Consentono l'interoperabilità
- Ad esempio, HTTP, SMTP

Protocolli proprietari:

- Ad esempio, Skype

Quale servizio di trasporto richiede un'applicazione?

Perdita di dati

- alcune applicazioni (ad esempio, audio) possono tollerare qualche perdita
- altre applicazioni (ad esempio, trasferimento di file, telnet) richiedono un trasferimento dati affidabile al 100%

Throughput

- alcune applicazioni (ad esempio, quelle multimediali) per essere "efficaci" richiedono un certo throughput minimo per funzionare
- altre applicazioni (le "applicazioni elastiche") utilizzano il throughput a disposizione

Temporizzazione

- alcune applicazioni (ad esempio, telefonia Internet, giochi interattivi) per essere "realistiche" richiedono piccoli ritardi

Sicurezza

- Cifratura, integrità dei dati, ...

Requisiti del servizio di trasporto di alcune applicazioni comuni

Applicazione	Tolleranza alla perdita di dati	Throughput	Sensibilità al tempo
Trasferimento file	No	Variabile	No
Posta elettronica	No	Variabile	No
Documenti Web	No	Variabile	No
Audio/video in tempo reale	Sì	Audio: da 5 kbit/s a 1 Mbit/s Video: da 10 kbit/s a 5 Mbit/s	Sì, centinaia di ms
Audio/video memorizzati	Sì	Come sopra	Sì, pochi secondi
Giochi interattivi	Sì	Fino a pochi kbit/s	Sì, centinaia di ms
Messaggistica istantanea	No	Variabile	Sì e no

Servizi dei protocolli di trasporto Internet

Servizio di Transport Control Protocol (TCP):

- orientato alla connessione*: è richiesto un setup fra i processi client e server
- trasporto affidabile* fra i processi d'invio e di ricezione
- controllo di flusso*: il mittente non vuole sovraccaricare il destinatario
- controllo della congestione*: "strozza" il processo d'invio quando le rete è sovraccaricata
- non offre*: temporizzazione, garanzie su un'ampiezza di banda minima, sicurezza

Servizio di User Datagram Protocol (UDP):

- trasferimento dati inaffidabile fra i processi d'invio e di ricezione
- non offre*: setup della connessione, affidabilità, controllo di flusso, controllo della congestione, temporizzazione né ampiezza di banda minima e sicurezza

D: Perché preoccuparsi? Perché esiste UDP?

Applicazioni Internet: protocollo a livello applicazione e protocollo di trasporto, RFC

Applicazione	Protocollo a livello applicazione	Protocollo di trasporto sottostante
Posta elettronica	SMTP [RFC 2821]	TCP
Accesso a terminali remoti	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Trasferimento file	FTP [RFC 959]	TCP
Multimedia in streaming	HTTP (es. YouTube) RTP [RFC 1889]	TCP o UDP
Telefonia Internet	SIP, RTP, proprietario (es. Skype)	Tipicamente UDP

Capitolo 2: Livello applicazione

- 2.1 Principi delle applicazioni di rete
 - ❖ Architetture delle applicazioni
 - ❖ Servizi richiesti dalle applicazioni
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Posta elettronica
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Applicazioni P2P
- 2.7 Cloud computing
- 2.8 Programmazione delle socket (cenni)

Tim Berners-Lee



- ❑ Inventore del World Wide Web
- ❑ Ha ricevuto l'ACM Turing Award
 - ❖ "For inventing the World Wide Web, the first web browser, and the fundamental protocols and algorithms [that allowed] the web to scale"
- ❑ https://amturing.acm.org/award_winners/berners-lee_8087960.cfm
- ❑ Sir Tim Berners-Lee 2016 ACM A.M. Turing Lecture "[What is the World Wide Web & what is its future?](#)"

Web e HTTP

Terminologia

- Una **pagina web** è costituita da **oggetti**
- Un oggetto può essere un file HTML, un'immagine JPEG, un'applet Java, un file audio, ...
- Una pagina web è formata da un **file base** scritto tramite l'**Hypertext Markup Language (HTML)**, che solitamente include diversi oggetti referenziati
- Ogni oggetto è referenziato da uno Uniform Resource Locator (**URL**)

Esempio di URL:

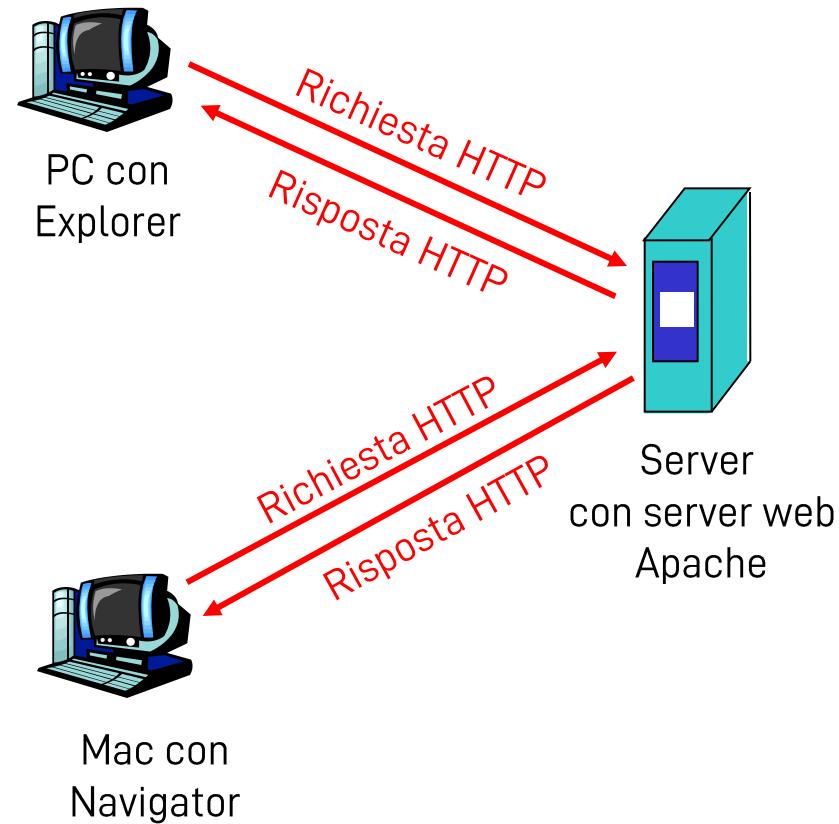
www . someschool . edu / someDept / home . html

The URL "www . someschool . edu / someDept / home . html" is shown with two horizontal curly braces underneath it. The first brace spans from "www . someschool . edu" to the first slash, and is labeled "nome dell'host". The second brace spans from the first slash to "home . html", and is labeled "nome del percorso".

Panoramica su HTTP

HTTP: HyperText Transfer Protocol

- ❑ Protocollo a livello di applicazione del Web
- ❑ Modello client/server
 - ❖ *client*: il browser che richiede, riceve, "visualizza" gli oggetti del Web
 - ❖ *server*: il server web invia oggetti in risposta a una richiesta



Panoramica su HTTP (continua)

Usa TCP:

- Il client inizializza la connessione TCP (crea una socket) con il server, la porta 80
- Il server accetta la connessione TCP dal client
- Messaggi HTTP scambiati fra browser (client HTTP) e server web (server HTTP)
- Connessione TCP chiusa

HTTP è un protocollo "senza stato" (stateless)

- Il server non mantiene informazioni sulle richieste fatte dal client

nota

I protocolli che mantengono lo "stato" sono complessi!

- La storia passata (stato) deve essere memorizzata
- Se il server e/o il client si bloccano, le loro viste dello "stato" potrebbero essere contrastanti e dovrebbero essere riconciliate

Connessioni HTTP

Connessioni non persistenti

- Almeno un oggetto viene trasmesso su una connessione TCP

Connessioni persistenti

- Più oggetti possono essere trasmessi su una singola connessione TCP tra client e server

Connessioni non persistenti

Supponiamo che l'utente immetta l'URL

`www.someSchool.edu/someDepartment/home.html`

(contiene testo, riferimenti a 10 immagini jpeg)

1a. Il client HTTP inizializza una connessione TCP con il server HTTP (processo) a `www.someSchool.edu` sulla porta 80

2. Il client HTTP trasmette un *messaggio di richiesta* (con l'URL) nella socket della connessione TCP. Il messaggio indica che il client vuole l'oggetto `someDepartment/home.index`

1b. Il server HTTP all'host `www.someSchool.edu` in attesa di una connessione TCP alla porta 80 "accetta" la connessione e avvisa il client

3. Il server HTTP riceve il messaggio di richiesta, forma il *messaggio di risposta* che contiene l'oggetto richiesto e invia il messaggio nella sua socket

Connessioni non persistenti (continua)

tempo

-
- The diagram illustrates a non-persistent HTTP connection. A vertical blue line on the left is labeled "tempo" (time) with a downward arrow. To its right, two red numbered steps are shown: step 4 above step 5. Step 4 points to the end of a red arrow pointing from step 5. Step 5 contains descriptive text about the client receiving a response and visualizing the HTML. Step 6 is positioned below step 5, indicating the repetition of the process for each of the 10 JPEG objects.
4. Il server HTTP chiude la connessione TCP
 5. Il client HTTP riceve il messaggio di risposta che contiene il file html e visualizza il documento html.
Esamina il file html, trova i riferimenti a 10 oggetti jpeg
 6. I passi 1-5 sono ripetuti per ciascuno dei 10 oggetti jpeg

Schema del tempo di risposta

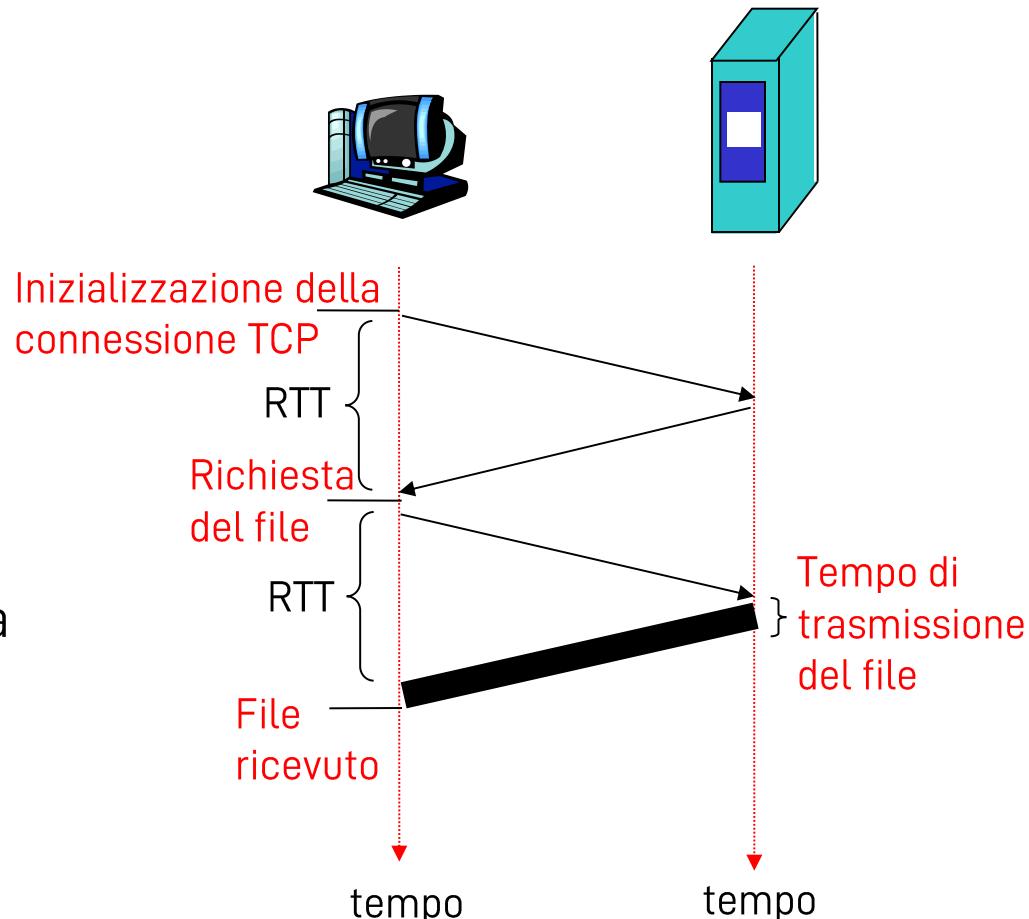
Definizione di Round-Trip Time (RTT):

Tempo di propagazione di andata e ritorno tra due host

(Es.: tempo impiegato da un piccolo pacchetto per andare dal client al server e ritornare al client)

Tempo di risposta:

- un RTT per inizializzare la connessione TCP
- un RTT perché ritornino la richiesta HTTP e i primi byte della risposta HTTP
- tempo di trasmissione del file



$$\text{totale} = 2 \text{ RTT} + \text{tempo di trasmissione}$$

Connessioni persistenti

Svantaggi delle connessioni non persistenti:

- Richiedono 2 RTT per oggetto
- Overhead del sistema operativo per *ogni* connessione TCP
- I browser spesso aprono connessioni TCP parallele per caricare gli oggetti referenziati

Connessioni persistenti

- Il server lascia la connessione TCP aperta dopo l'invio di una risposta
- I successivi messaggi tra gli stessi client/server vengono trasmessi sulla connessione aperta
- Il client invia le richieste non appena incontra un oggetto referenziato
- Un solo RTT per tutti gli oggetti referenziati

Messaggi HTTP

- Due tipi di messaggi HTTP: *richiesta, risposta*
- **Messaggio di richiesta HTTP:**
 - ❖ ASCII (formato leggibile dall'utente)

Riga di richiesta
(comandi GET,
POST, HEAD)

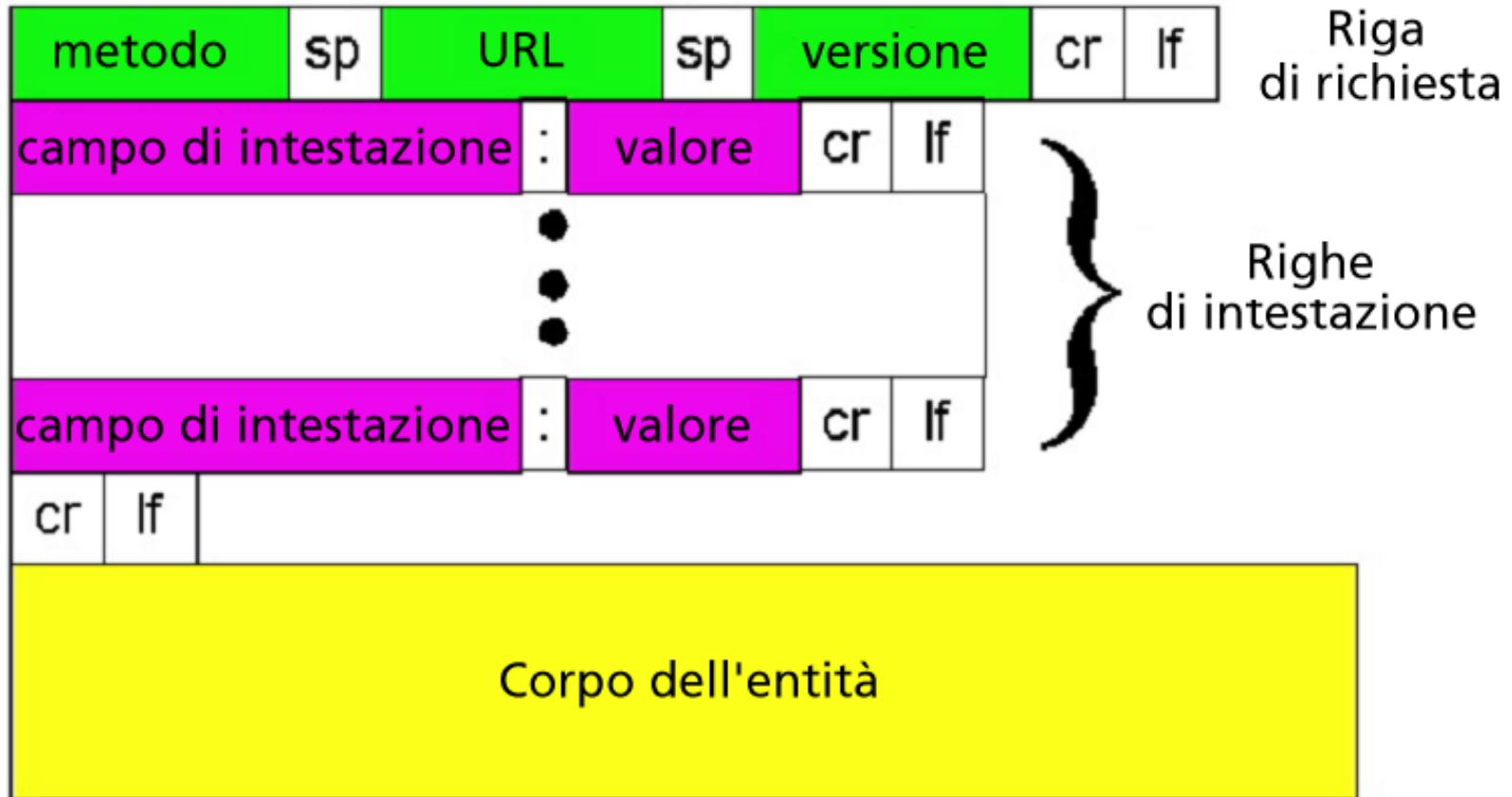
Righe di
intestazione

Un carriage return
e un line feed
indicano la fine
del messaggio

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

→ (carriage return e line feed extra)

Messaggio di richiesta HTTP: formato generale



Upload dell'input di un form

Metodo Post:

- La pagina web spesso include un form per l'input dell'utente
- L'input arriva al server nel corpo dell'entità

Metodo URL:

- Usa il metodo GET
- L'input arriva al server nel campo URL della riga di richiesta:

`www.somesite.com/animalsearch?monkeys&banana`

Tipi di metodi

HTTP/1.0

- GET
- POST
- HEAD
 - ❖ Chiede al server di escludere l'oggetto richiesto dalla risposta

HTTP/1.1

- GET, POST, HEAD
- PUT
 - ❖ Include il file nel corpo dell'entità e lo invia al percorso specificato nel campo URL
- DELETE
 - ❖ Cancella il file specificato nel campo URL

Messaggio di risposta HTTP

Riga di stato
(protocollo
+ codice di stato
+ espressione di stato)

Righe di intestazione

Dati, ad esempio
il file HTML
richiesto

HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ...
Content-Length: 6821
Content-Type: text/html

dati dati dati dati dati ...

Codici di stato della risposta HTTP

Nella prima riga nel messaggio di risposta server->client.

Alcuni codici di stato e relative espressioni:

200 OK

- ❖ La richiesta ha avuto successo; l'oggetto richiesto viene inviato nella risposta

301 Moved Permanently

- ❖ L'oggetto richiesto è stato trasferito; la nuova posizione è specificata nell'intestazione **Location:** della risposta

400 Bad Request

- ❖ Il messaggio di richiesta non è stato compreso dal server

404 Not Found

- ❖ Il documento richiesto non si trova su questo server

505 HTTP Version Not Supported

- ❖ Il server non supporta la versione richiesta del protocollo HTTP

Provatate HTTP (lato client)

- Collegatevi via Telnet o simili al vostro server web preferito:

```
telnet www.unitn.it 80
```

```
rlwrap nc -C www.unitn.it 80  
(oppure via putty)
```

Apre una connessione TCP alla porta 80 (porta di default per un server HTTP) dell'host www.unitn.it
Tutto ciò che digitate viene trasmesso alla porta 80 di www.unitn.it

- Digitate una richiesta GET:

```
GET / HTTP/1.1  
Host: www.unitn.it
```

Digitando questo (premete due volte il tasto Invio), trasmettete una richiesta GET minima (ma completa) al server HTTP

- Guardate il messaggio di risposta trasmesso dal server HTTP
- Inviare quanto sotto a gaia.cs.umass.edu: cosa cambia?

```
GET /kurose_ross/interactive/index.php HTTP/1.1  
Host: gaia.cs.umass.edu
```

Interazione utente-server: i cookie

Molti dei più importanti siti web usano i cookie

Quattro componenti:

- 1) Una riga di intestazione nel messaggio di *risposta* HTTP
- 2) Una riga di intestazione nel messaggio di *richiesta* HTTP
- 3) Un file cookie mantenuto sul sistema terminale dell'utente e gestito dal browser dell'utente
- 4) Un database sul sito

Esempio:

- ❖ Susan accede sempre a Internet dallo stesso PC
- ❖ Visita per la prima volta un particolare sito di commercio elettronico
- ❖ Quando la richiesta HTTP iniziale giunge al sito, il sito crea un identificativo unico (ID) e una *entry* nel database per ID

Cookie (continua)

Client (Susan)



cookie



ebay 8734
amazon 1678

messaggio di richiesta

messaggio di risposta +
Set-cookie: 1678

messaggio di richiesta
cookie: 1678

messaggio di risposta

una settimana dopo:



ebay 8734
amazon 1678

messaggio di richiesta
cookie: 1678

messaggio di risposta

Server

Il server
crea l'ID 1678
per l'utente

entry nel
database



Specifica
del cookie

accesso

accesso

Specifica
del cookie

Cookie (continua)

Cosa possono contenere i cookie:

- Autorizzazione
- Carta per acquisti
- Raccomandazioni
- Stato della sessione dell'utente (e-mail)

Lo "stato"

- Mantengono lo stato del mittente e del ricevente per più transazioni
- I messaggi http trasportano lo stato

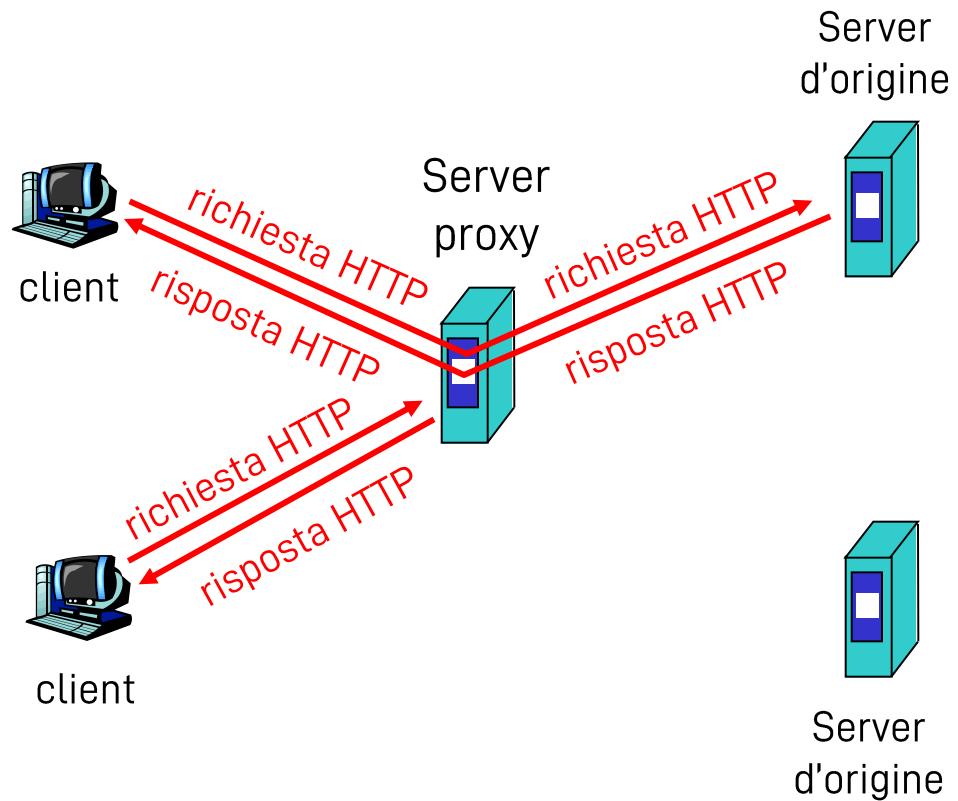
nota
Cookie e privacy:

- I cookie permettono ai siti di imparare molte cose sugli utenti
- L'utente può fornire al sito il nome e l'indirizzo e-mail

Cache web (server proxy)

Obiettivo: soddisfare la richiesta del client senza coinvolgere il server d'origine

- L'utente configura il browser: accesso al Web tramite la cache
- Il browser trasmette tutte le richieste HTTP alla cache
 - ❖ Oggetto nella cache: la cache fornisce l'oggetto
 - ❖ Altrimenti la cache richiede l'oggetto al server d'origine e poi lo inoltra al client



Cache web (continua)

- La cache opera come client e come server
- Tipicamente la cache è installata da un ISP (università, aziende o ISP residenziali)

Perché il caching web?

- Riduce i tempi di risposta alle richieste dei client
- Riduce il traffico sul collegamento di accesso a Internet
- Internet arricchita di cache consente ai provider "scadenti" di fornire dati con efficacia (ma così fa anche la condivisione di file P2P)

Esempio di caching

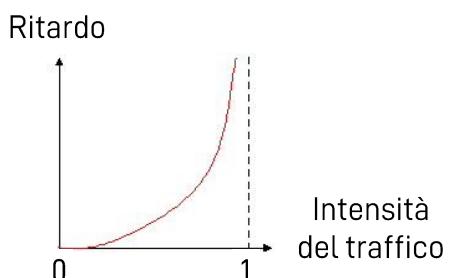
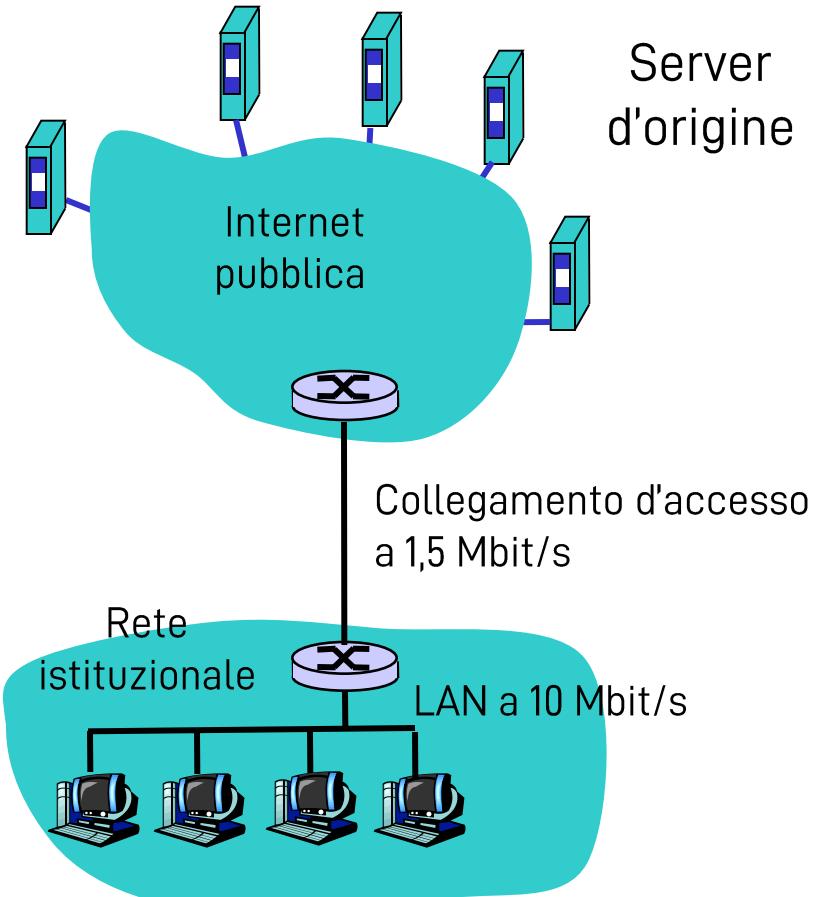
Ipotesi

- Dimensione media di un oggetto = 100.000 bit
- Frequenza media di richieste dai browser istituzionali ai server d'origine = 15/secondo
- Ritardo dal router istituzionale a qualsiasi server d'origine e ritorno al router = 2 s

Conseguenze

- Intensità di utilizzo della LAN = 15%
- Intensità di utilizzo del collegamento d'accesso = 100%
- Ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN
 - Intensità traffico su LAN:
 $(15 \text{ richieste/s}) * (0,1 \text{ Mbit/richiesta}) / (10 \text{ Mbit/s}) = 0,15$
 - Intensità traffico sul link di accesso:
 $(15 \text{ richieste/s}) * (0,1 \text{ Mbit/richiesta}) / (1,5 \text{ Mbit/s}) = 1$

= 2 s + **minuti** + qualche ms



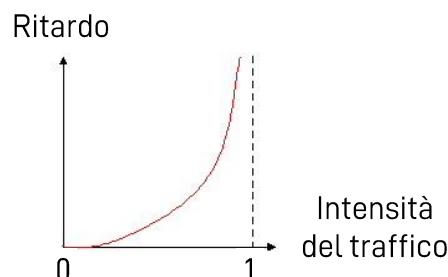
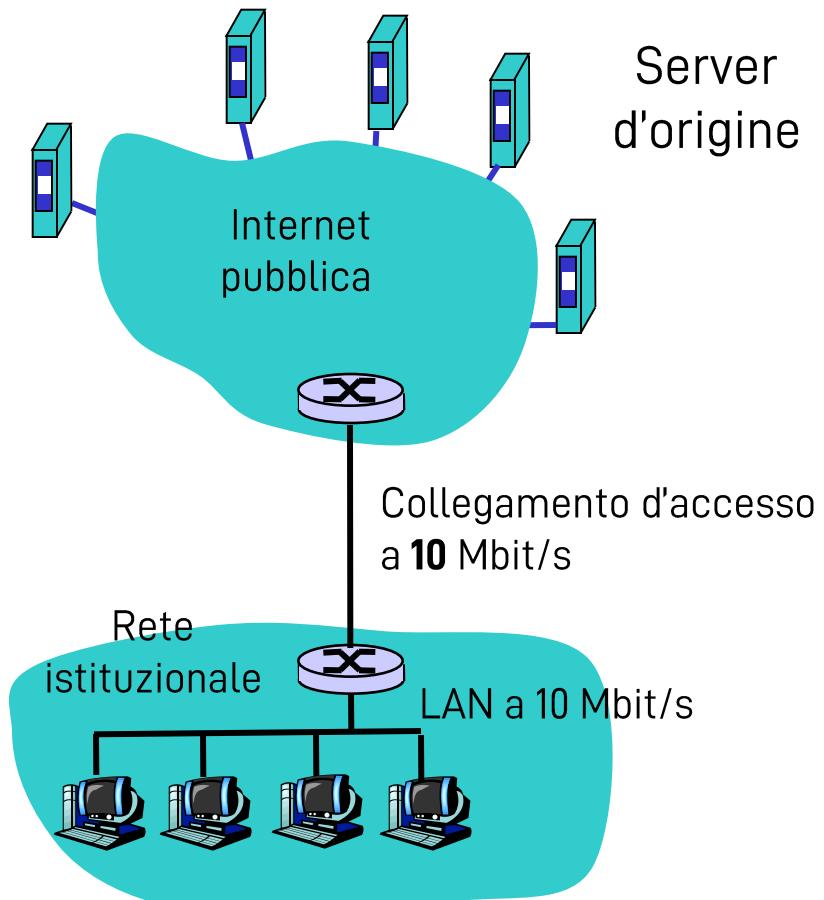
Esempio di caching (continua)

Soluzione possibile

- ☐ Aumentare l'ampiezza di banda del collegamento d'accesso a 10 Mbps, per esempio

Conseguenze

- ☐ Intensità di utilizzo della LAN = 15%
- ☐ Intensità di utilizzo del collegamento di accesso = $15\% = 0.15$
 - ☐ Quindi il ritardo non arriva fino a minuti, al più fino a qualche millisecondo
- ☐ Ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN
= 2 s + qualche ms + qualche ms
- ☐ Tuttavia:
 - ☐ Cambiare infrastruttura è spesso molto costoso



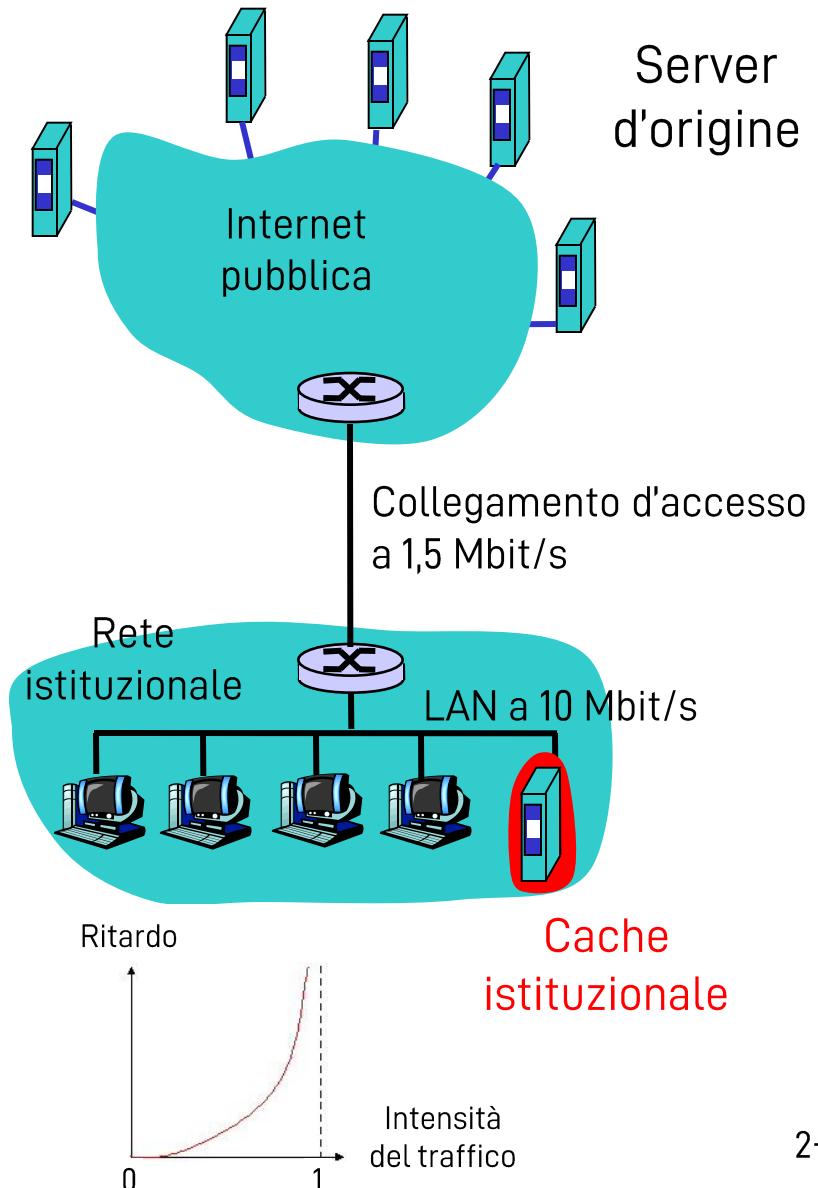
Esempio di caching (continua)

Soluzione possibile: installare la cache

- Supponiamo una percentuale di successo (*hit rate*) pari a 40%

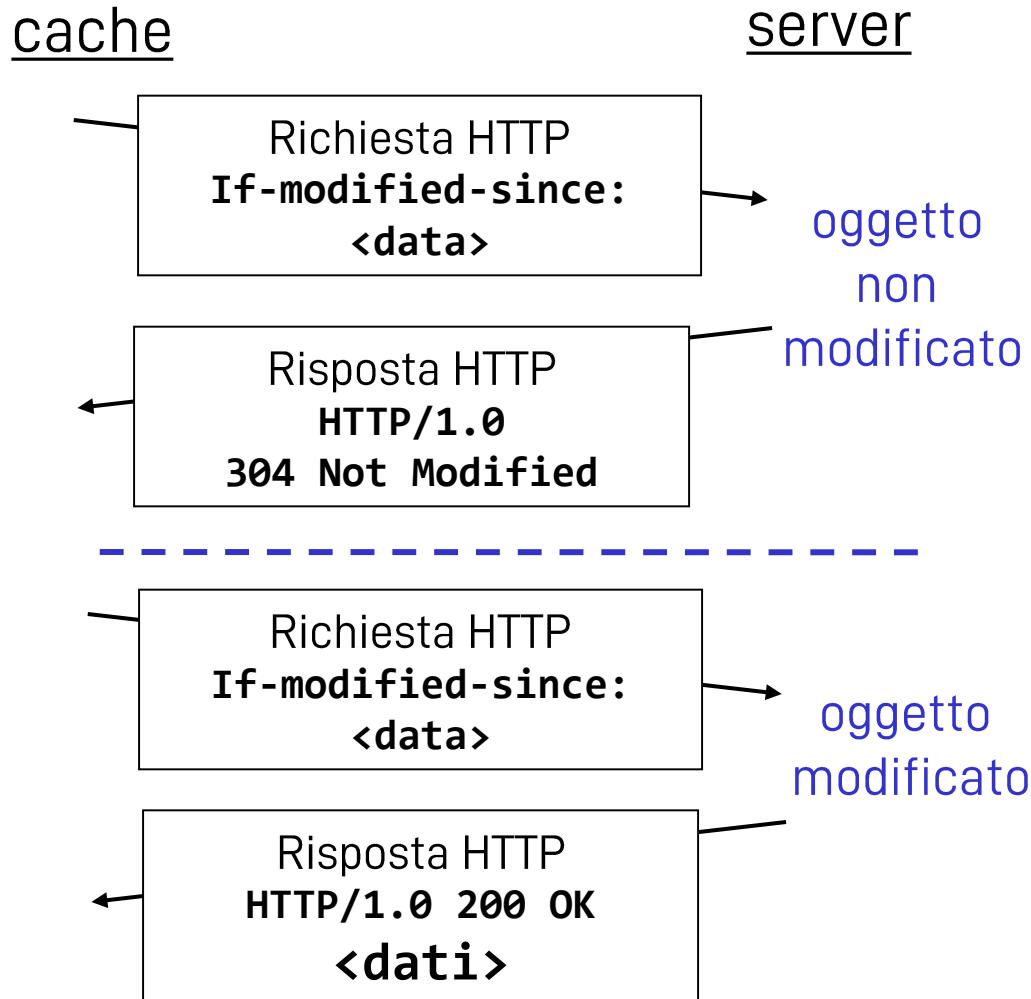
Conseguenze

- Il 40% delle richieste sarà soddisfatto quasi immediatamente (diciamo in 0,01 s)
- Il 60% delle richieste sarà soddisfatto dal server d'origine
- L'utilizzo del collegamento d'accesso si è ridotta al 60%, determinando ritardi trascurabili (circa 10 ms)
 - $(0,6 * 15 \text{ richieste/s}) * (0,1 \text{ Mbit/richiesta}) / (1,5 \text{ Mbit/s}) = 0,6 < 1 \rightarrow \text{molto meglio!}$
 - Ritardo totale: 2,01 s (ovvero Internet + 10 ms)
- Quindi qual è il ritardo medio adesso?
 - $0,6 * (2,01) \text{ s} + 0,4 * 0,01 \text{ s} = 1,21 \text{ s}$

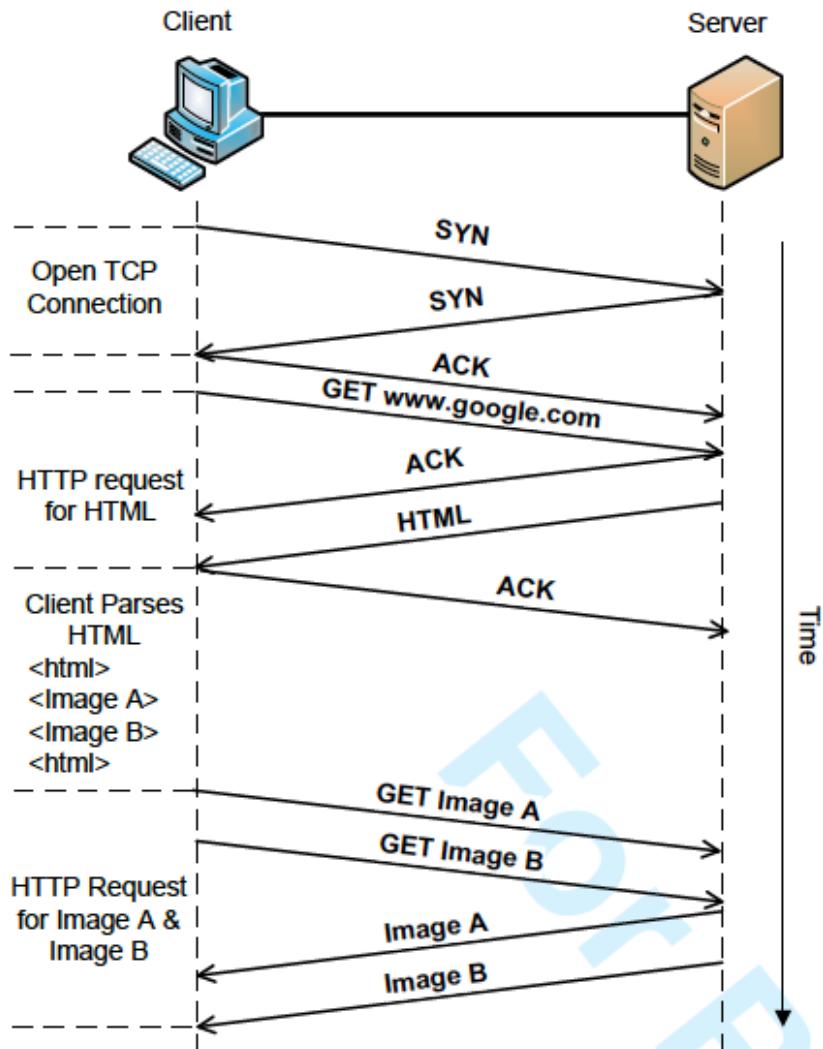
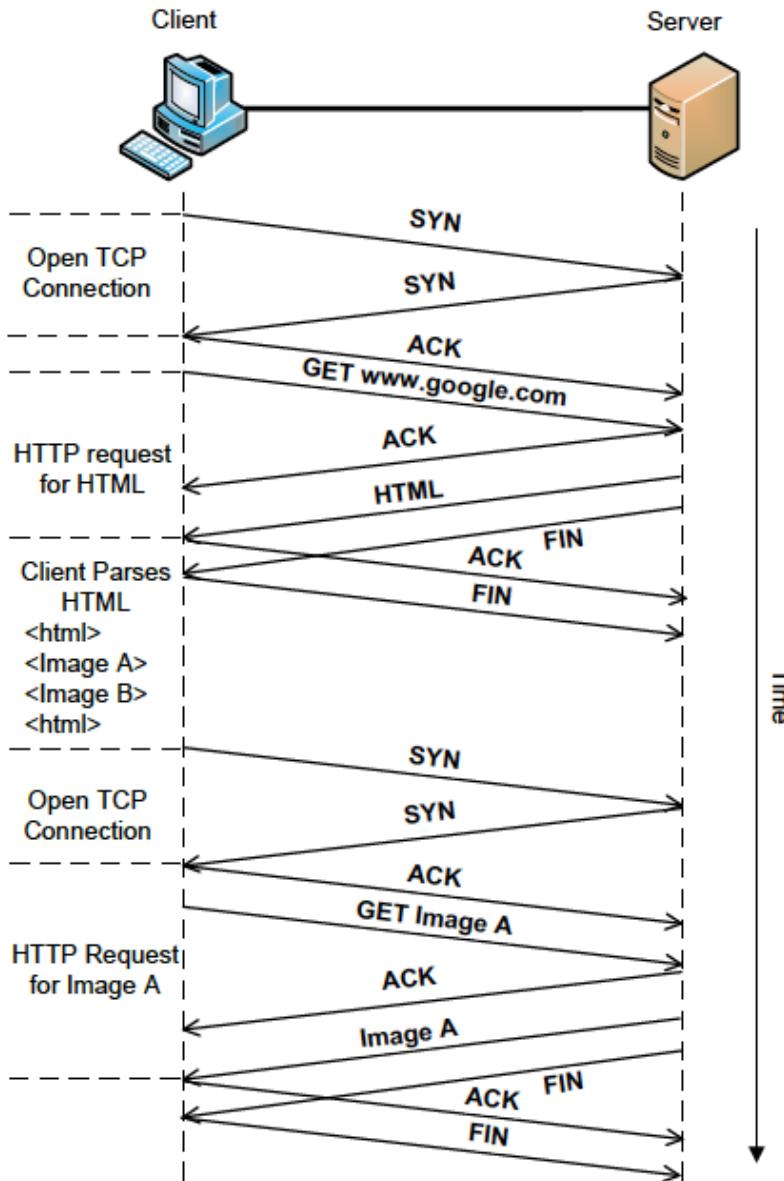


Supporto HTTP per le cache: GET condizionale

- **Obiettivo:** non inviare un oggetto se la cache ha una copia aggiornata dell'oggetto
- Cache: specifica la data della copia dell'oggetto nella richiesta HTTP
 - ❖ **If-modified-since:** <data>
- Server: la risposta non contiene l'oggetto se la copia nella cache è aggiornata:
 - ❖ **HTTP/1.0 304 Not Modified**



HTTP/1.0 e HTTP/1.1

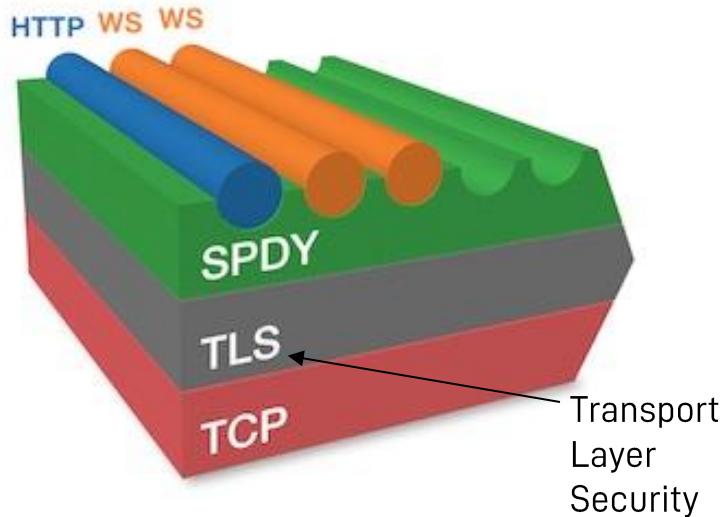


HTTP/2.0

- HTTP/2 rappresenta un'evoluzione di HTTP, cioè mantiene i metodi HTTP, i codici di stato e la semantica
- Il protocollo è focalizzato sulle prestazioni, specificamente sulla latenza percepita dall'utente e l'utilizzo delle risorse di rete e dei server
- L'obiettivo è utilizzare un'unica connessione dai browser ad un sito web

HTTP/2.0

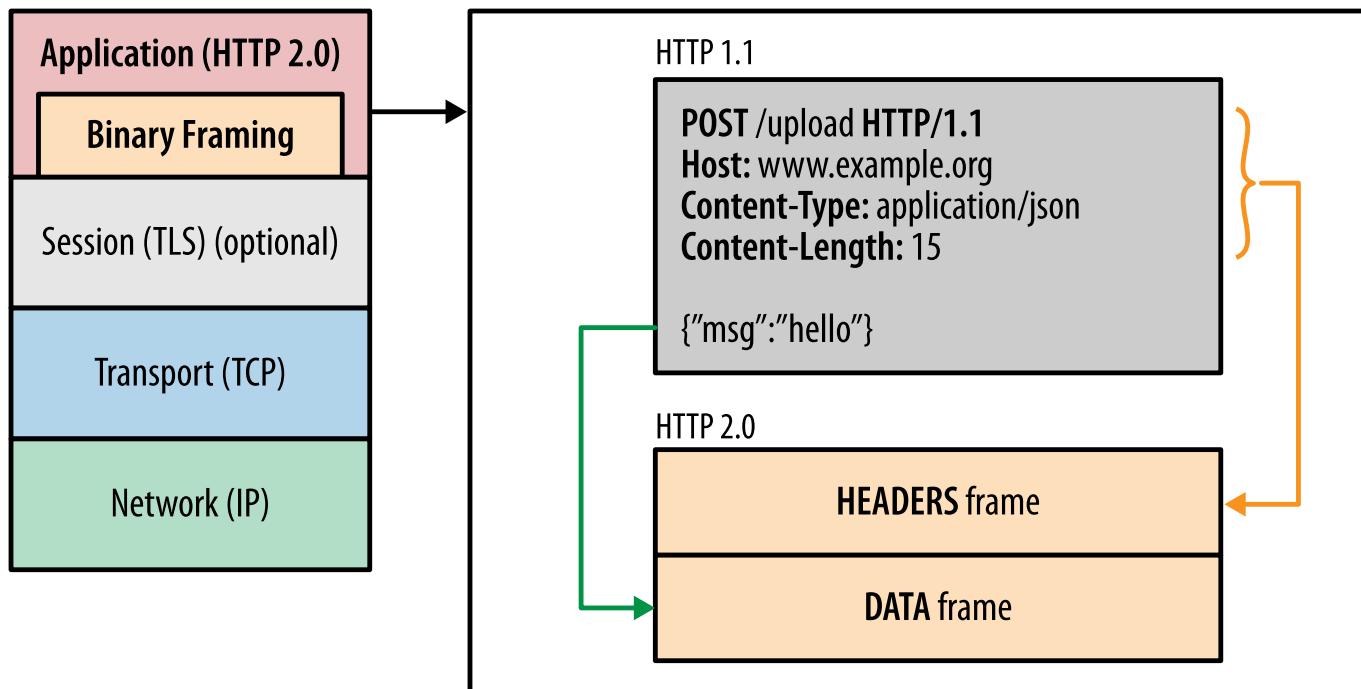
- HTTP/2 si basa su SPDY, protocollo di livello applicazione per trasportare contenuti sul web con minima latenza, tramite:
 - ❖ Multiplexing di flussi: supporta un numero illimitato di flussi su una singola connessione TCP ("allunga la connessione" aumentandone l'efficienza)
 - ❖ Priorità delle richieste: il client può inviare quante richieste desidera, assegnando a ciascuna una priorità (evita la congestione)
 - ❖ Compressione dell'header HTTP: diminuisce i byte trasmessi



Transport
Layer
Security

Framing binario

- Nuovo livello di framing binario, responsabile della modalità di encapsulamento e trasferimento dei messaggi HTTP



Framing binario

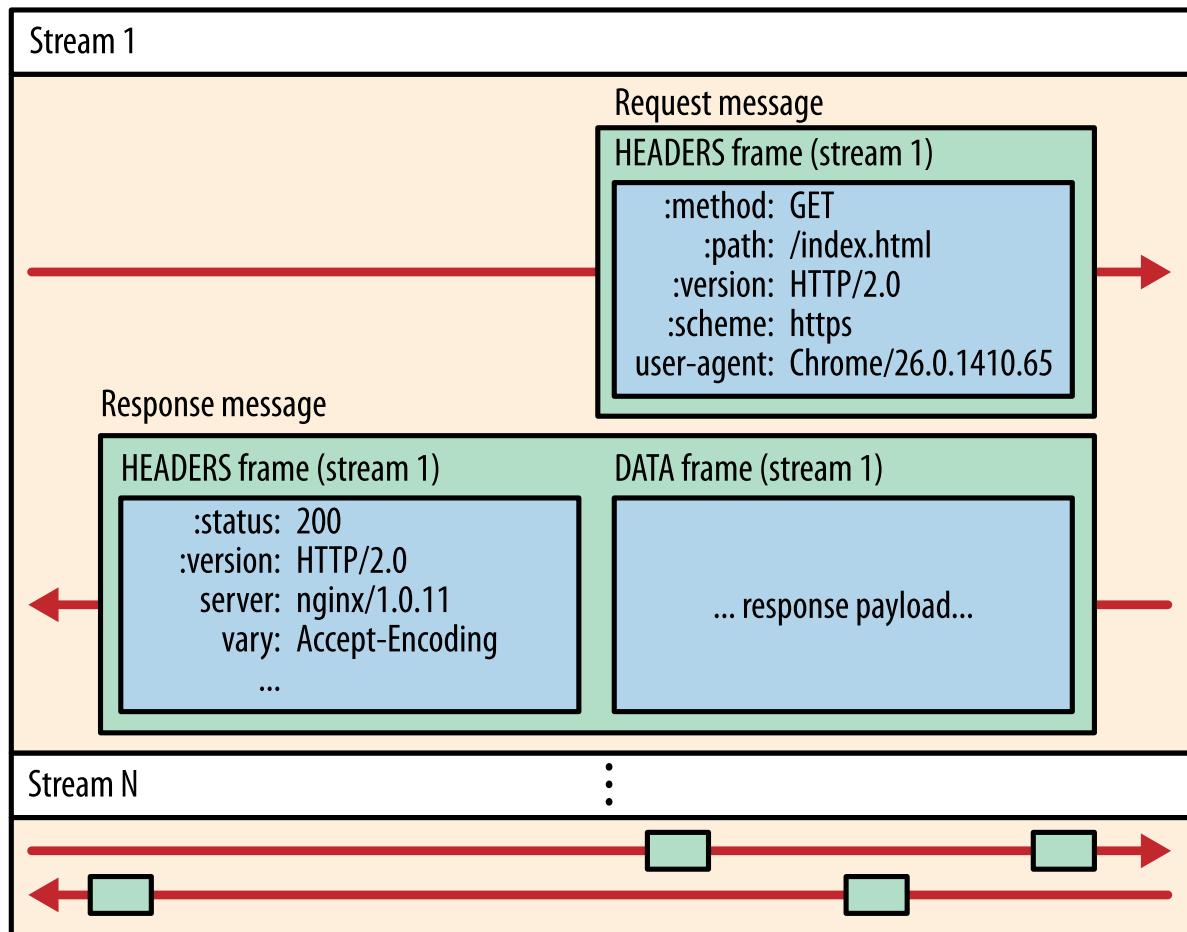
- La semantica HTTP è invariata
- La codifica in transito è differente
- Tutte le comunicazioni HTTP/2 sono suddivise in messaggi e frame più piccoli, ognuno dei quali codificato in formato binario
- Sia il client che il server devono utilizzare il nuovo meccanismo di codifica binario per capirsi tra loro: un client HTTP/1.x non comprende un server HTTP/2 e viceversa

Stream, messaggi e frame

- Tutte le comunicazioni vengono eseguite all'interno di una singola connessione TCP che può portare qualsiasi numero di stream bidirezionali di byte
- Ogni stream ha un identificativo univoco e le informazioni di priorità opzionali utilizzate per il trasporto di messaggi
- Ogni messaggio è un messaggio HTTP logico, come una richiesta o una risposta, che consiste di uno o più frame
- Il frame è la più piccola unità di comunicazione che porta un tipo specifico di dati (intestazioni HTTP, payload del messaggio)
- I frame di diversi stream possono essere interposti e quindi riassemblati tramite l'identificatore di flusso incorporato nell'intestazione di ciascun frame

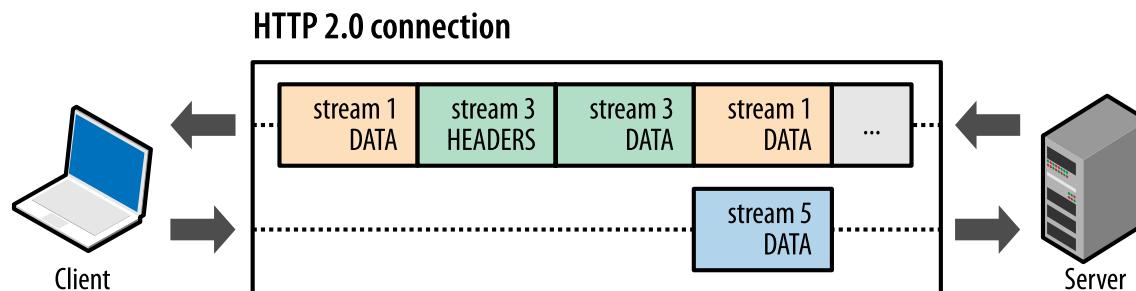
Stream, messaggi e frame

Connection



Multiplexing di richieste e risposte

- In HTTP/1.x, se il client desiderava fare più richieste parallele per migliorare le prestazioni, doveva utilizzare più connessioni TCP
- Il nuovo livello di framing binario in HTTP/2 rimuove queste limitazioni e consente il multiplexing di richieste e risposte
- Il client e il server possono dividere un messaggio HTTP in frame indipendenti, intervallarli e ricomporli all'altro capo



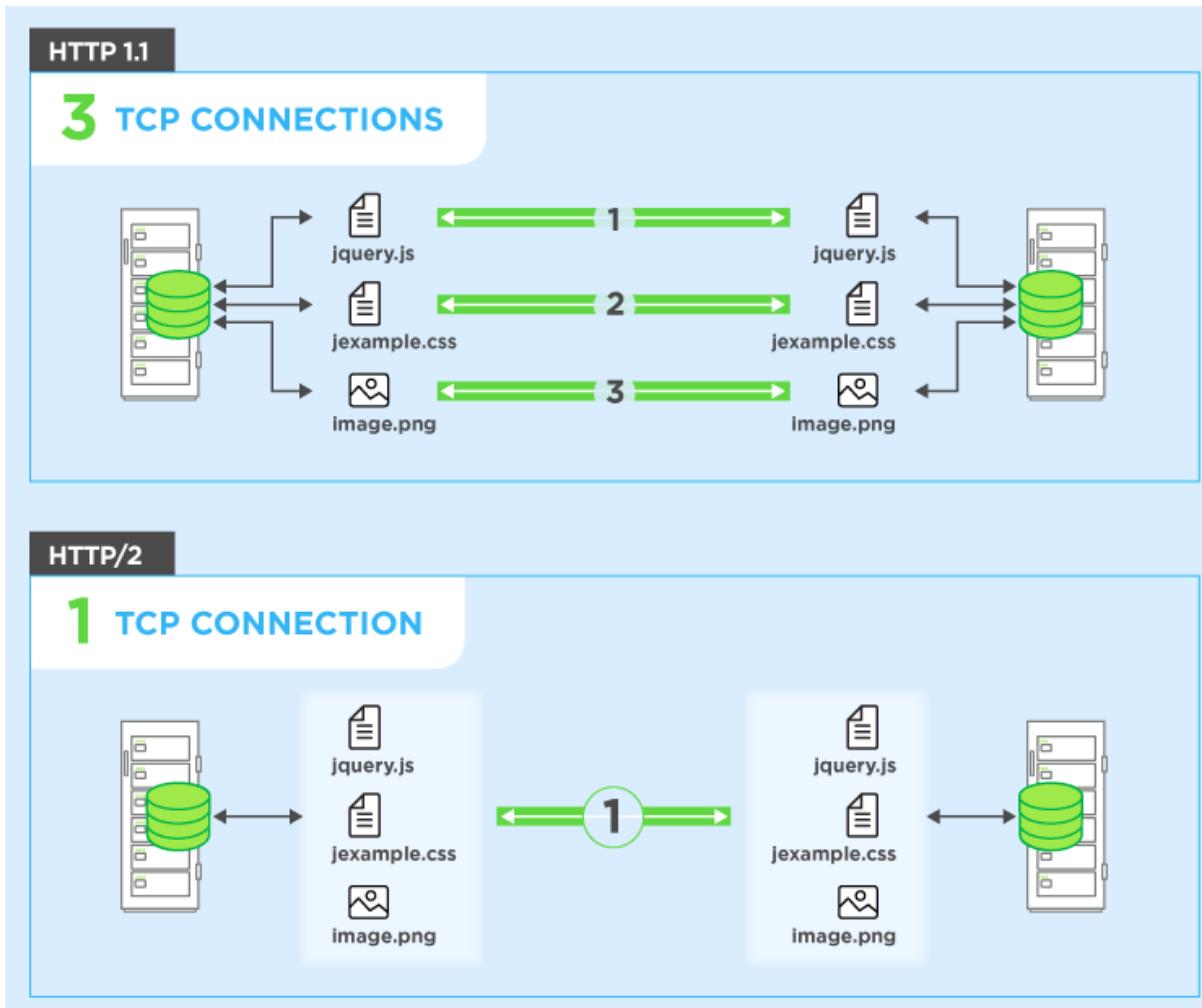
Priorità degli stream

- L'ordine in cui i frame vengono interposti e consegnati sia dal client che dal server influenza le prestazioni
- HTTP/2 consente a ciascun stream di avere peso e dipendenza associati:
 - ❖ Ogni stream può avere assegnato un peso intero compreso tra 1 (più leggero) e 256 (più pesante)
 - ❖ Ogni stream può avere una dipendenza esplicita su un altro stream
- Il client può costruire e comunicare un "albero di priorità" che esprime come preferirebbe ricevere risposte. A sua volta, il server può utilizzare queste informazioni per dare priorità all'elaborazione dello stream

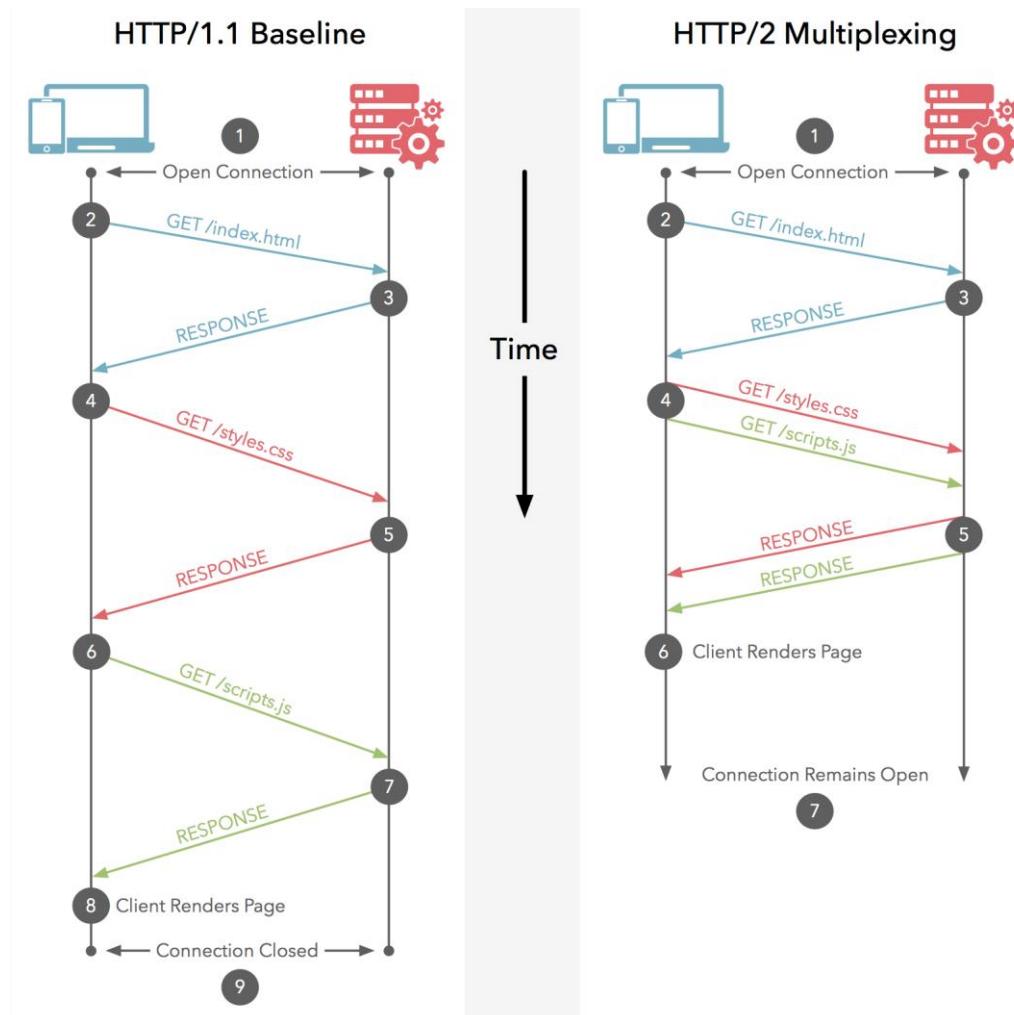
Connessione HTTP/2

- HTTP/2 non ha bisogno di connessioni TCP multiple per eseguire multiplex di stream in parallelo
- Ogni stream è suddiviso in molti frame, che possono essere intervallati e gestiti con priorità diverse
- Tutte le connessioni HTTP/2 sono persistenti, e si rende necessaria solo una connessione per origine

Connessione HTTP/2



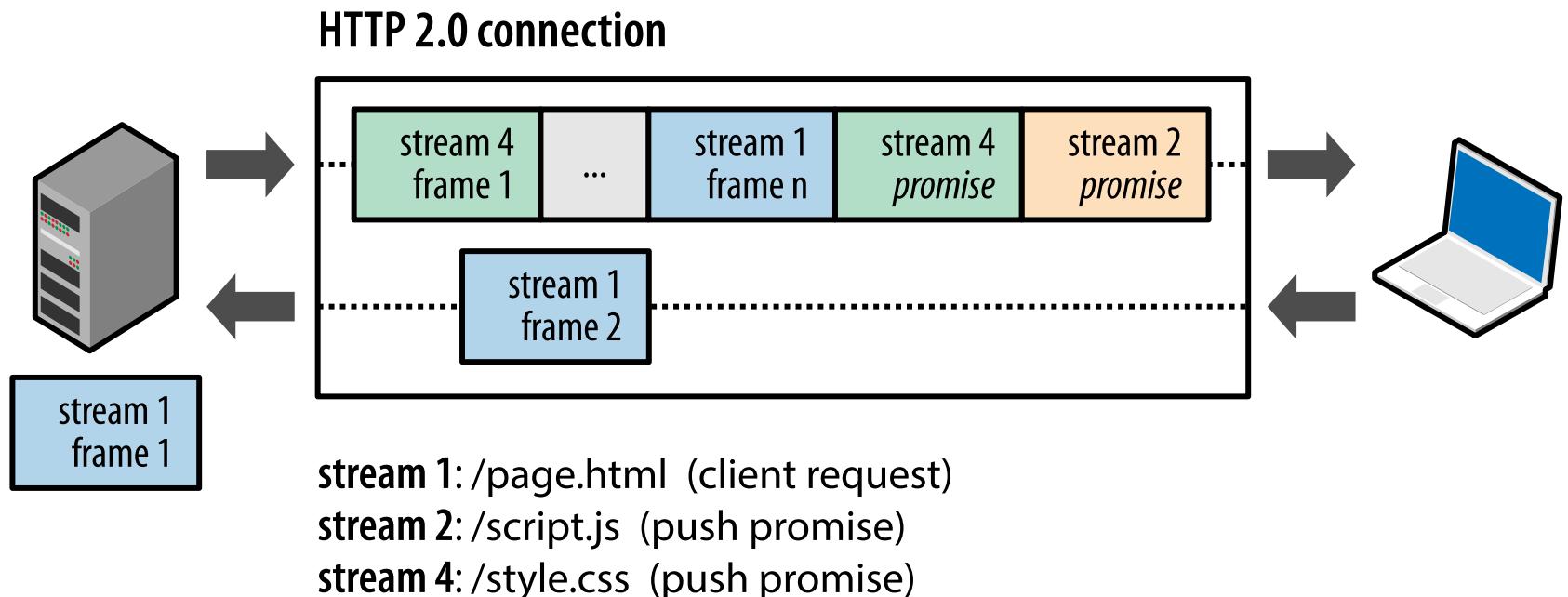
Connessione HTTP/2



Server push

- Il server può inviare più risposte per una singola richiesta del client:
 - ❖ Oltre alla risposta alla richiesta originale, il server può inviare risorse aggiuntive senza che il client debba richiederle esplicitamente
- Una tipica applicazione web è costituita da decine di risorse, tutte scoperte dal cliente esaminando il documento fornito dal server
 - ❖ Perché non eliminare la latenza supplementare e lasciare che il server invii le risorse associate in anticipo?

Server push



Transport Layer Security (TLS)

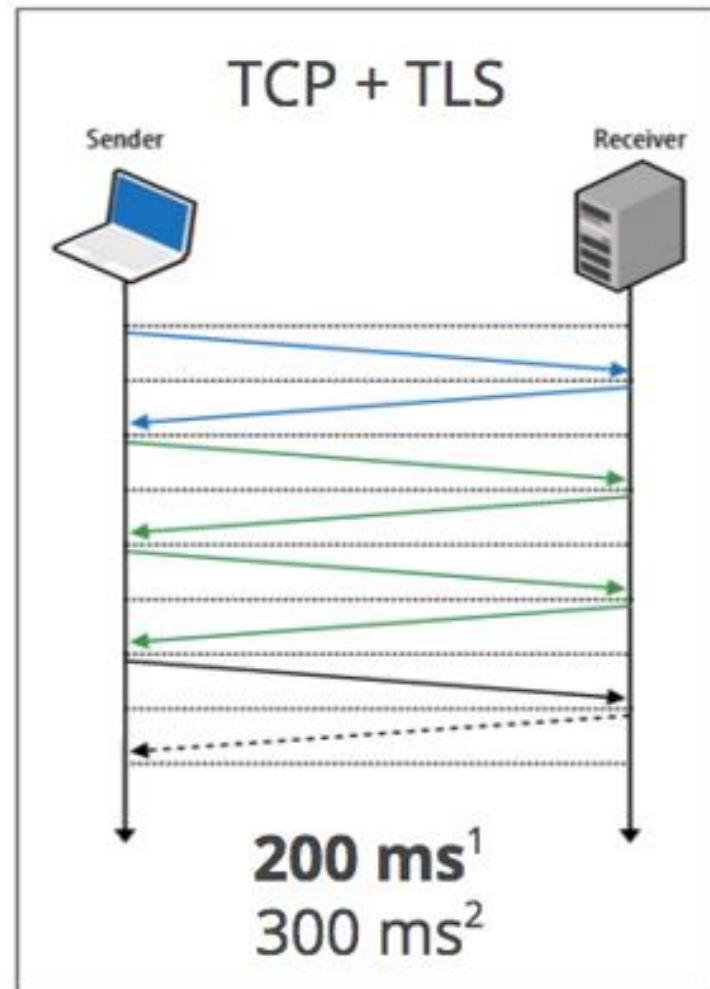
- Protocollo crittografico che permette una comunicazione sicura dalla sorgente al destinatario fornendo:
 - ❖ Autenticazione
 - ❖ Integrità dei dati
 - ❖ Confidenzialità
- Il funzionamento del protocollo TLS può essere suddiviso in tre fasi principali:
 - ❖ Negoziazione fra le parti dell'algoritmo da utilizzare
 - ❖ Scambio delle chiavi e autenticazione
 - ❖ Cifratura simmetrica e autenticazione dei messaggi

HTTPS

- Estende HTTP inviando tutte le richieste e risposte attraverso una comunicazione cifrata (tramite TLS) sulla porta 443
- Es., <https://www.unitn.it>
 - ❖ Il browser capisce automaticamente che deve usare TLS
 - ❖ Inizializza la connessione
 - ❖ Invia richieste criptate e deve decriptare le risposte che riceve
- Nota: questo critta solo le informazioni del protocollo HTTP...
 - ❖ ...non gli header dei protocolli inferiori
 - ❖ ...quindi le porte TCP usate e gli indirizzi IP usati sono in chiaro
- Recentemente, > 50-70% dei siti usa HTTPS (fonte: Wikipedia)

La «S» aumenta il ritardo di connessione

- L'inizializzazione di una sessione TLS richiede tempo
- 2 RTT
- +1 RTT per stabilire una connessione TCP
- Soluzione: protocolli più moderni (HTTP/3, QUIC)
 - ❖ Capitolo 3 sul livello di trasporto

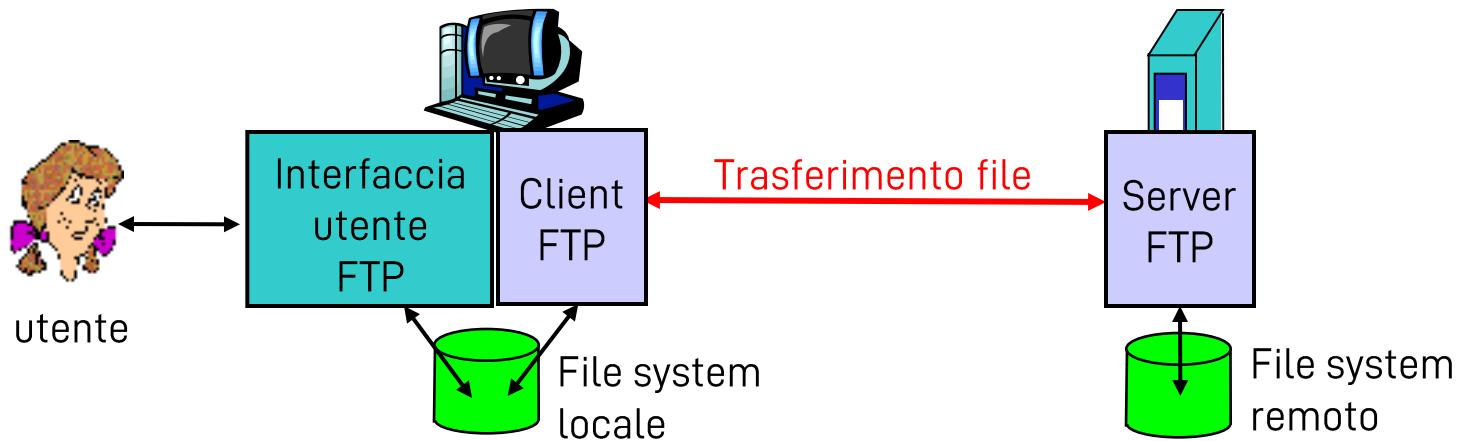


1. Repeat connection
2. Never talked to server before

Capitolo 2: Livello applicazione

- 2.1 Principi delle applicazioni di rete
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Posta elettronica
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Applicazioni P2P
- 2.7 Cloud computing
- 2.8 Programmazione delle socket (cenni)

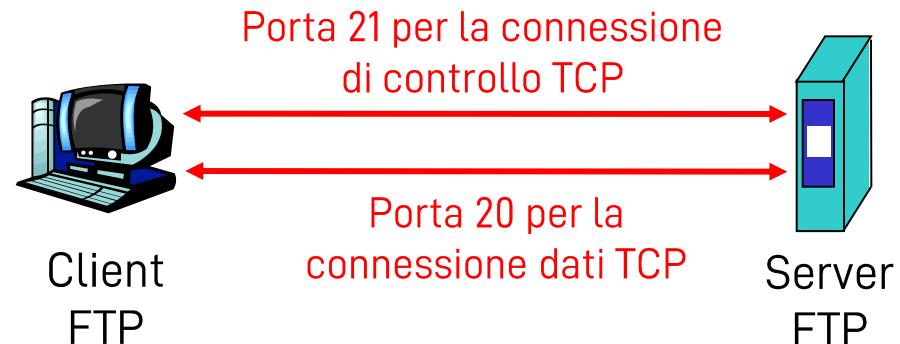
FTP: file transfer protocol



- Trasferimento file a/da un host remoto
- Modello client/server
 - ❖ *client*: il lato che inizia il trasferimento (a/da un host remoto)
 - ❖ *server*: host remoto
- FTP: definito nell'RFC 959
- Server FTP: porta 21

FTP: connessione di controllo, connessione dati

- Il client FTP contatta il server FTP alla porta 21, specificando TCP come protocollo di trasporto
- Il client ottiene l'autorizzazione sulla connessione di controllo
- Il client cambia la directory remota inviando i comandi sulla connessione di controllo
- Quando il server riceve un comando per trasferire un file, apre una connessione dati TCP con il client
- Dopo il trasferimento di un file, il server chiude la connessione



- Il server apre una seconda connessione dati TCP per trasferire un altro file.
- Connessione di controllo: **"fuori banda"** (*out-of-band*)
- Il server FTP mantiene lo "stato": directory corrente, autenticazione precedente

Comandi e risposte FTP

Comandi comuni:

- Inviati come testo ASCII sulla connessione di controllo
- USER *username***
- PASS *password***
- LIST**
elenca i file della directory corrente
- RETR *filename***
recupera (*get*) un file dalla directory corrente
- STOR *filename*** memorizza (*put*) un file nell'host remoto

Codici di ritorno comuni:

- Codice di stato ed espressione (come in HTTP)
- 331 Username OK, password required**
- 125 data connection already open; transfer starting**
- 425 Can't open data connection**
- 452 Error writing file**

Capitolo 2: Livello applicazione

- 2.1 Principi delle applicazioni di rete
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Posta elettronica
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Applicazioni P2P
- 2.7 Cloud computing
- 2.8 Programmazione delle socket (cenni)

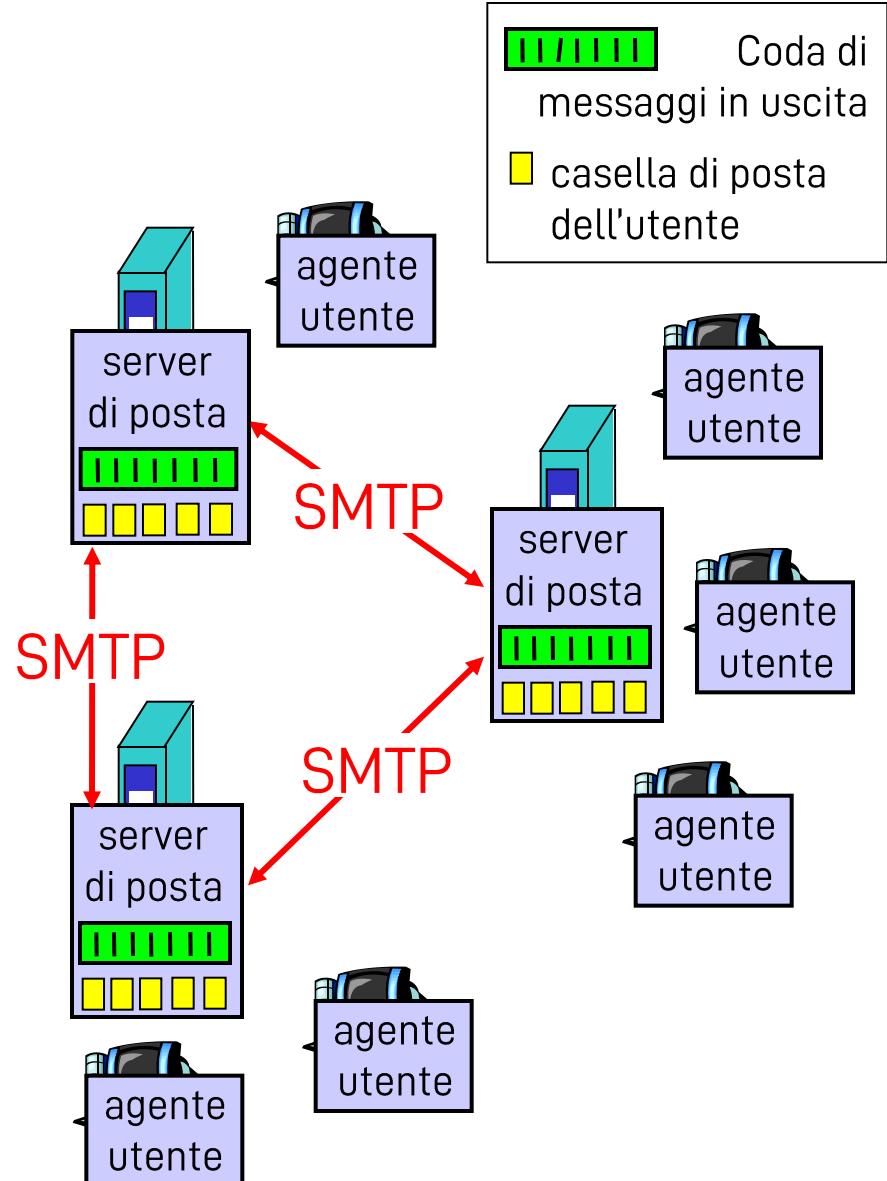
Posta elettronica

Tre componenti principali:

- Agente utente
- Server di posta
- Simple mail transfer protocol:
SMTP

Agente utente

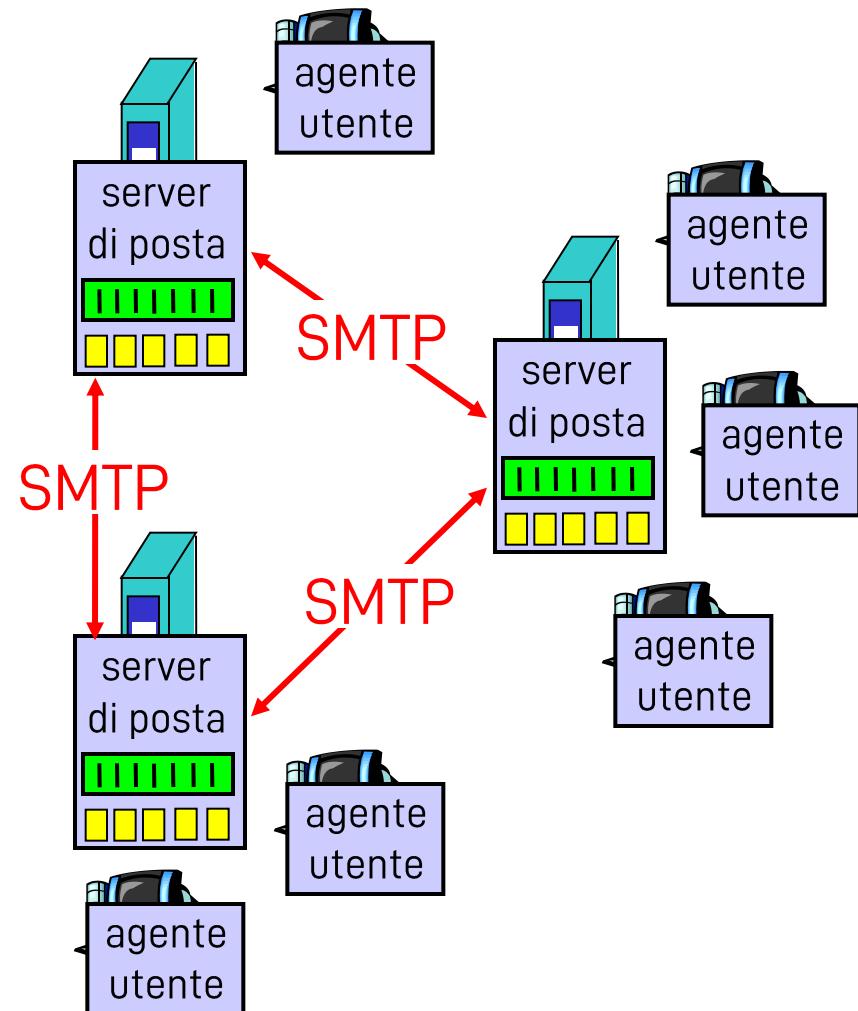
- Detto anche "mail reader"
- Composizione, editing, lettura dei messaggi di posta elettronica
- Esempi: Eudora, Outlook, elm, Mozilla, Thunderbird
- I messaggi in uscita o in arrivo sono memorizzati sul server



Posta elettronica: server di posta

Server di posta

- Casella di posta (*mailbox*)
contiene i messaggi in arrivo per l'utente
- Coda di messaggi da trasmettere
- Protocollo SMTP tra server di posta per inviare messaggi di posta elettronica
 - ❖ Client: server di posta trasmittente
 - ❖ "Server": server di posta ricevente



Posta elettronica: SMTP [RFC 2821]

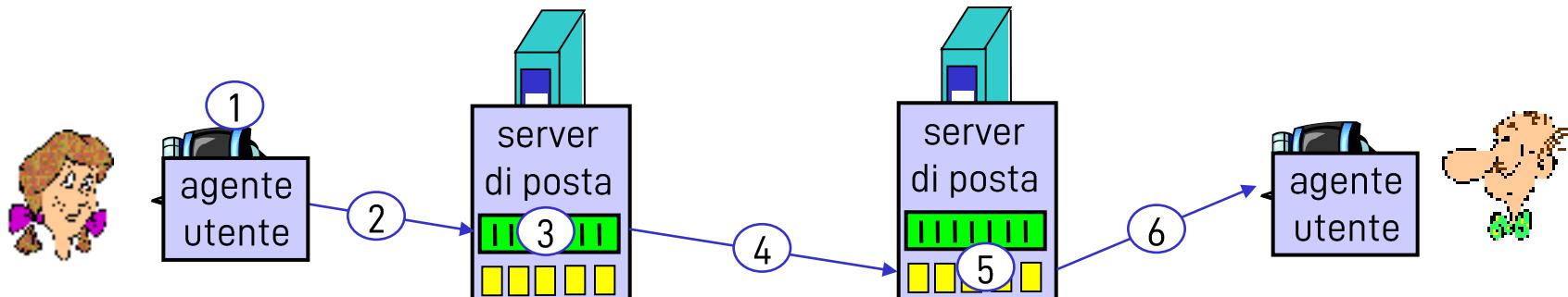
- ❑ Usa TCP per trasferire in modo affidabile i messaggi di posta elettronica dal client al server, porta 25
- ❑ Trasferimento diretto:
dal server SMTP trasmittente al server SMTP ricevente
- ❑ Tre fasi per il trasferimento
 - ❖ handshaking (saluto)
 - ❖ trasferimento di messaggi
 - ❖ chiusura
- ❑ Interazione comando/risposta
 - ❖ **comandi:** testo ASCII
 - ❖ **risposta:** codice di stato ed espressione
- ❑ I messaggi devono essere nel formato **ASCII a 7 bit**

Codici ASCII a 7 bit

Hex	Value																
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	'	70	p		
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q		
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r		
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s		
04	EOT	14	DC4	24	\$	34	4	44	D	54	T	64	d	74	t		
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u		
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v		
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w		
08	BS	18	CAN	28	(38	8	48	H	58	X	68	h	78	x		
09	HT	19	EM	29)	39	9	49	I	59	Y	69	i	79	y		
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z		
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[6B	k	7B	{		
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C		7C			
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D]	6D	m	7D	}		
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~		
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL		

Scenario: Alice invia un messaggio a Bob

- 1) Alice usa il suo agente utente per comporre il messaggio da inviare a `bob@someschool.edu`
- 2) L'agente utente di Alice invia un messaggio al server di posta di Alice; il messaggio è posto nella coda di messaggi
- 3) Il lato client di SMTP apre una connessione TCP con il server di posta di Bob
- 4) Il client SMTP invia il messaggio di Alice sulla connessione TCP
- 5) Il server di posta di Bob pone il messaggio nella casella di posta di Bob
- 6) Bob invoca il suo agente utente per leggere il messaggio



SMTP: note finali

- SMTP usa connessioni persistenti
- SMTP richiede che il messaggio (intestazione e corpo) sia nel formato ASCII a 7 bit
- Il server SMTP usa <CR><LF>. <CR><LF> per determinare la fine del messaggio

Confronto con HTTP:

- HTTP: pull
- SMTP: push
- Entrambi hanno un'interazione comando/risposta in ASCII, codici di stato
- HTTP: ciascun oggetto è incapsulato nel suo messaggio di risposta
- SMTP: più oggetti vengono trasmessi in un unico messaggio

Formato dei messaggi di posta elettronica

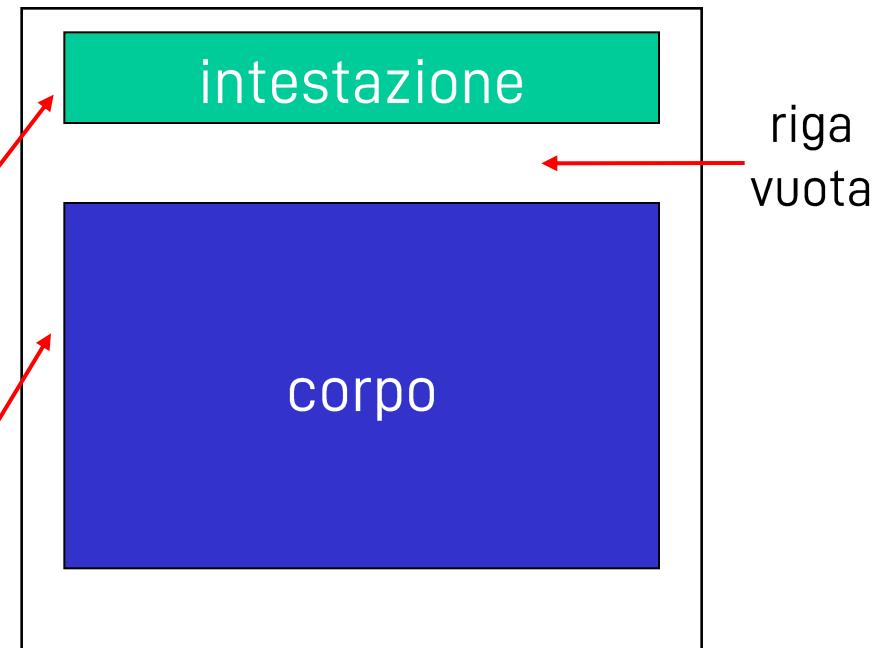
SMTP: protocollo per scambiare messaggi di posta elettronica

RFC 822: standard per il formato dei messaggi di testo:

- Righe di intestazione, per esempio
 - ❖ From/Da:
 - ❖ To/A:
 - ❖ Subject/Oggetto:

*** *Diversi dai comandi SMTP !* ***

- Riga vuota
- Corpo
 - ❖ il "messaggio", soltanto caratteri ASCII



Esempio di interazione SMTP

```
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: <rob@hamburger.edu>
S: 250 rob@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
```

Provate un'interazione SMTP:

- Aprite una connessione con il server, ad esempio usate:
 - `telnet servername 25` (Linux)
 - oppure `r1wrap nc -C servername 25` (Linux)
 - oppure usando l'applicazione `putty` (Windows)
- Riceverete la risposta con codice **220** dal server
- Digitate i comandi
 - `HELO, MAIL FROM, RCPT TO, DATA, QUIT`
 - Ricordate di inserire i campi intestazione (`From: / To: / Subject:`), poi una linea vuota, poi il corpo del messaggio
- Questo vi consente di inviare messaggi di posta elettronica anche senza usare un client per la posta elettronica

Altri comandi: <https://blog.mailtrap.io/smtp-commands-and-responses/>

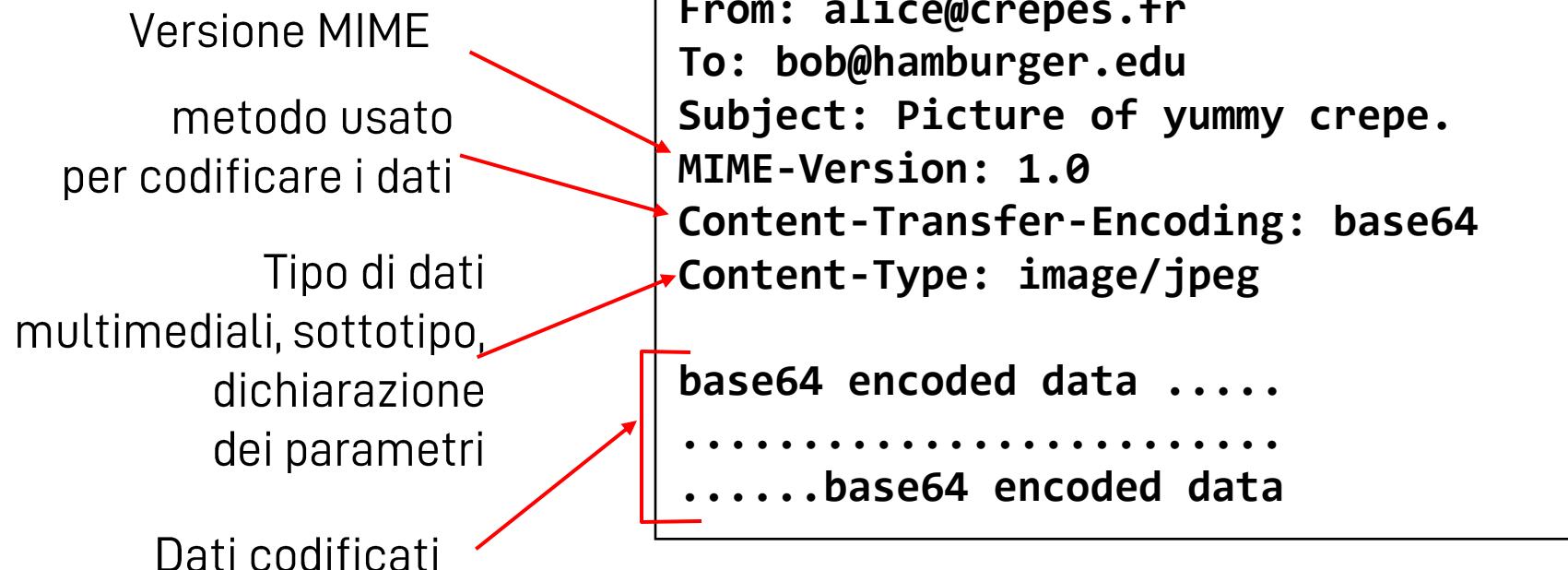
Per chi vuole far girare il container docker con il server SMTP (comandi Linux):

```
sudo docker pull reachfive/fake-smtp-server
```

```
sudo docker run --name fake-smtp -p 1080:1080 -p 1025:1025 reachfive/fake-smtp-server
```

Formato del messaggio: estensioni di messaggi multimediali

- ❑ MIME: estensioni di messaggi di posta multimediali, RFC 2045, 2056
- ❑ Alcune righe aggiuntive nell'intestazione dei messaggi dichiarano il tipo di contenuto MIME



Codifica base-64

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	ø
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

Hex	Value
77	W

Inefficiente per definizione:
ogni gruppo di 6 bit viene codificato nell'email usando 8 bit

Un esempio reale

```
Date: Fri, 19 May 2000 10:23:16 -0400 (EDT)
From: Doug Sauder <doug@penguin.example.com>
To: =?iso-8859-1?Q?Heinz_M=FCller?= <mueller@example.com>
Subject: PNG graphic
Message-ID: <Pine.LNX.4.21.0005191019440.8452-101000@penguin.example.com>
MIME-Version: 1.0
Content-Type: MULTIPART/MIXED; BOUNDARY="-1463757054-170444605-958746196=:8452"
```

This message is in MIME format. The first part should be readable text, while the remaining parts are likely unreadable without MIME-aware tools. Send mail to mime@docserver.cac.washington.edu for more info.

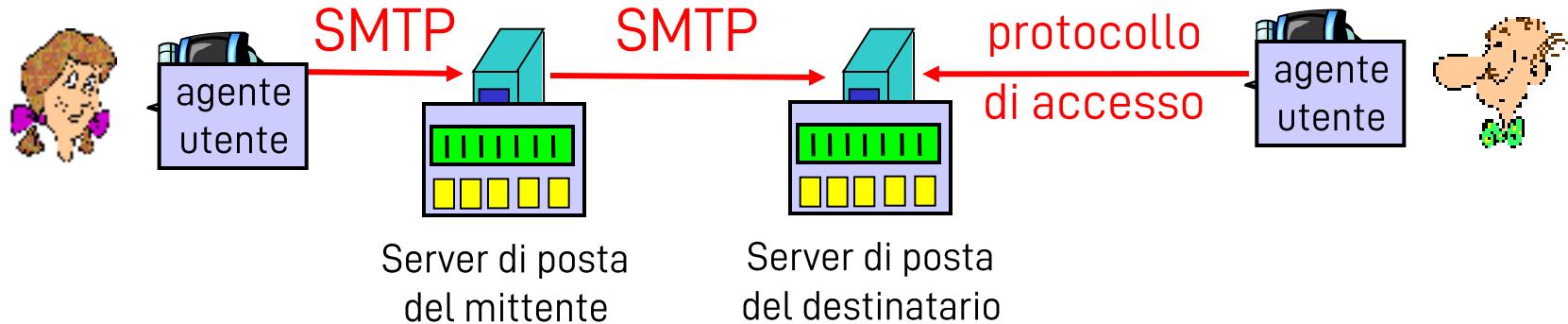
```
--1463757054-170444605-958746196=:8452
Content-Type: TEXT/PLAIN; charset=US-ASCII
```

```
--1463757054-170444605-958746196=:8452
Content-Type: APPLICATION/octet-stream; name="redball.png"
Content-Transfer-Encoding: BASE64
Content-ID: <Pine.LNX.4.21.0005191023160.8452@penguin.example.com>
Content-Description: red ball
Content-Disposition: attachment; filename="redball.png"
```

```
iVBORw0KGgoAAAANSUhEUgAAABsAAAAC0MAAQCj8CgRnAADAFBMVEx//8A
AAABAAALAAAABAAxAAAARAAKAAAADAAcAAAyAABEAABNAABIAAA9AAAj
AAAHAAmABAAB7AACGAACHAAB9AAAB0AA8gAA5AAUAAAAGAAAnAABLAABv
AACQAAC1AAC7AAC/AACrAAChACMAABzAABAAAuuAAIAABMAAB3AACZAAC0
GRnK0djVPt3kSndQW4AACoBPSAAAxAAYAAAEEABFAACaAC7CTRYWhf
hITmf3/mV1bgHx/SAAC5AACjAA8dAABCAAAQAAAJAAbAAC6Dw//OVFTek5P1
rKzpMntzXvJSSXAADBAAcAACCAAAbtAABTA2AA2AbAAFAAFAAeAACBAADL
ICDdZ2fonJzrpqbtiorvUVhvFBTRAADDAAc2AA4AA8eAABAAAiAABXAACS
AAADCAdaGxvoVXseHjvehjVVfVjibhADAOAAc3AACnAAVAHBAAAraAAP
AACDAADFAADhBQxrKCjvPDzvNTXvGxvjADQAAJDAAc1AACXAAEeAABsAAB
AAASAAACAAcBAAAdpAAdvAgLnAADDAAcLAAcGAAACwAAbwAAAATAAKAByADI
ADTAADNAAcAAACDAAcBAAuAAAeAAB+AAAACKAACNAAB//AAbpAABQAAAawAACR
AACpAAC8AACqAACbAA1ABJAAqAAA0AAA0AACSACVAActACmAACJAA86
AABrAAbaAA+AAPAApA8gAACCAACFAAcAACWAACPAABAAAzaAAAHAABVAACO
ACKAAA4AAAQA/AAByAACAAcBAA3AAAABmAAABDAABWAAAeAAAzAAA8
AA6AAAFAAAmAAAAdANAAAAMAAAAAAAAMAAAAAAAAMAAAAAAAAMAAAAAAAAM
AAAAAAAAMAAAAAAAAMAAAAAAAAMAAAAAAAAMAAAAAAAAMAAAAAAAAMAAAAAAAAM
AAAAAAAAMAAAAAAAAMAAAAAAAAMAAAAAAAAMAAAAAAAAMAAAAAAAAMAAAAAAAAM
Lt1FAAAAAXRST1MAQ0byZgAAABZ0RVh0l29mdHhc.mUAZ21mMnBuZyAYlAu
MT1evmgAAIIISURBVHicY2Cg/_80wIA8mJhZWFnZ2Dk4MaU5u5heSh5+Lkf
BD1QJf8zC/Eti4IKiUt18koJcsqgyF5nlpwTV1UU1ZRVVpx4NFk1UjIyghp
6+jq6RsYGHmbKJgk85mzW8Dk/rnASlhZ29ja2Ts40jkrl6Li4urDFNf53N/0w
8Tyv9hV18w8IDAoWDQkNC4+ASP5ni4wKlo6jytPSExTnfWE51F4A69n9G
Z1Z2Tn5efkFHxFySw1z1Ed5RSVY7j+TkGRvdul1txX1DY10zcktpaIt7h2Yn
0Aj+d711tyo79vT29SdnSj445bFVdhIo9xS1HNPuAwqTp1fn5Jrznk0050U1
a/acUG5p1Nz/uXLzV2qm6dxZ584S2WB1cJF15cshZr539xVftnyFKUVTi2T
VjvyhjLXb1m7TqohPt6F/HWbg0Ng63crGqvtwxRtu07Bjihcsw1+zanRw8
Z89eq337RQ/Ip60x03gIE1X/LbiKdm8T36KwbNmRo703zpHkPSzwHbQl//8f
lz1x200kyKJti7aqbzutfuUzI2gIuF8F2lR/05dw2+fZdw18VVOlI+Cj4/9/
jo0yYed50zMvhGqnm2V0Wi1m//D01fxHt6vL1k9w7rx7vQ5SQjbFtsMs
1y9evXid7QZacg0xmSxk1NzdtSwwU+J/VICaCPFIYU3XAjhI0tjf5sfyAAA
JXRFWHRDB21tzK50AGNsaxAyZ21mIHYuMC42IGJ5IF12ZxMgUG1ndwV0NnM
vAAAAABJRU5ErkJgg==
--1463757054-170444605-958746196=:8452-
```

Provate a copiare e incollare
in un file di testo, e a riconvertire
il contenuto dalla codifica
base64 a un file di nome
“redball.png”

Protocolli di accesso alla posta



- SMTP: consegna/memorizzazione sul server del destinatario
- Protocollo di accesso alla posta: ottenere i messaggi dal server
 - ❖ POP: Post Office Protocol [RFC 1939]
 - Autorizzazione (agente \leftrightarrow server) e download
 - ❖ IMAP: Internet Mail Access Protocol [RFC 1730]
 - Più funzioni (più complesse)
 - Manipolazione di messaggi memorizzati sul server
 - ❖ HTTP: gmail, Hotmail , Yahoo! Mail, ecc.

Protocollo POP3

Fase di autorizzazione

- Comandi del client:
 - ❖ **user**: dichiara il nome dell'utente
 - ❖ **pass**: password
- Risposte del server
 - ❖ +OK
 - ❖ -ERR

Fase di transazione, client:

- list**: elenca i numeri dei messaggi
- retr**: ottiene i messaggi in base al numero
- dele**: cancella
- quit**

```
S: +OK POP3 server ready
C: user rob
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 2 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 (altro) e IMAP

Ancora su POP3

- Il precedente esempio usa la modalità "scarica e cancella"
- Bob non può rileggere le e-mail se cambia client
- Modalità "scarica e mantieni": copia i messaggi su più client
- POP3 è un protocollo senza stato tra le varie sessioni

IMAP

- Mantiene tutti i messaggi in un unico posto: il server
- Consente all'utente di organizzare i messaggi in cartelle
- IMAP conserva lo stato dell'utente tra le varie sessioni:
 - ❖ I nomi delle cartelle e l'associazione tra identificatori dei messaggi e nomi delle cartelle

SMTP e Transport Layer Security (TLS)

- Per garantire la sicurezza della comunicazione, si può usare SMTP insieme a TLS
- Le comunicazioni non possono avvenire su porte di comunicazione ove è in ascolto un server di posta che non usa la crittografia
- Sono pertanto utilizzate porte diverse:
 - ❖ SMTP con SSL/TLS su porta 465
 - ❖ POP3 con SSL/TLS su porta 995
 - ❖ IMAP con SSL/TLS su porta 993

STARTTLS

- Evoluzione di TLS
- Consente di cifrare la connessione anche sulle porte originali o standard (110 per POP3, 143 per IMAP, 25 per SMTP)
- Il client che sfrutta tale protocollo:
 - ❖ Chiede in prima istanza al server l'instaurazione di una connessione cifrata
 - ❖ La sessione inizia «in chiaro»
 - ❖ Quindi diventa cifrata prima che venga trasmessa qualunque informazione sensibile o potenzialmente tale
- Viene spesso impiegato tra MTA (Mail Transfer Agent) ovvero durante il trasporto dell'email da un provider all'altro

Capitolo 2: Livello applicazione

- 2.1 Principi delle applicazioni di rete
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Posta elettronica
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Applicazioni P2P
- 2.7 Cloud computing
- 2.8 Programmazione delle socket (cenni)

DNS: Domain Name System

Persone: molti identificatori:

- ❖ Nome, codice fiscale,
numero della carta d'identità

Host e router di Internet:

- ❖ Indirizzo IP (32 bit) - usato per
indirizzare i datagrammi
 - ❖ Es. 121.7.106.83
 - ❖ Ogni campo compreso tra 0 e 255
(8 bit per campo, in totale 32 bit)
- ❖ “Nome”, ad esempio,
`www.yahoo.com` – usato
dagli esseri umani

D: Come associare un indirizzo IP ad
un nome?

Domain Name System:

- *Database distribuito*
implementato in una gerarchia di
server DNS
- *Protocollo a livello di applicazione*
che consente agli host, ai router e
ai server DNS di comunicare per
risolvere i nomi (tradurre
indirizzi/nomi)
 - ❖ Si noti: funzioni critiche di
Internet implementate come
protocollo a livello di applicazione
 - ❖ Complessità nelle parti
periferiche della rete

DNS

Servizi DNS

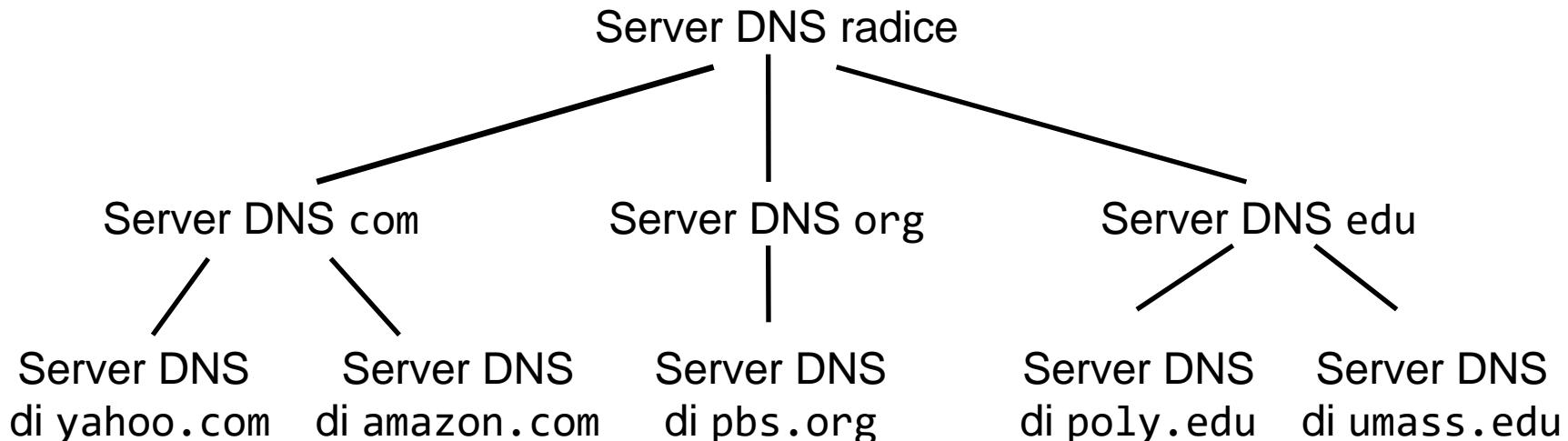
- Traduzione degli hostname in indirizzi IP
- Host aliasing
 - ❖ Un host può avere più nomi
 - ❖ Es. relay1.west.enterprise.com
→ www.enterprise.com
- Mail server aliasing
- Distribuzione del carico
 - ❖ Server web replicati: insieme di indirizzi IP raggruppati sotto lo stesso nome canonico

Perché non centralizzare DNS?

- Singolo punto di guasto
- Volume di traffico
- Database centralizzato distante
- Manutenzione

Un database centralizzato su un singolo server DNS
non è scalabile !

Database distribuiti e gerarchici



Il client cerca l'IP di `www.amazon.com`

Semplificando, succede che:

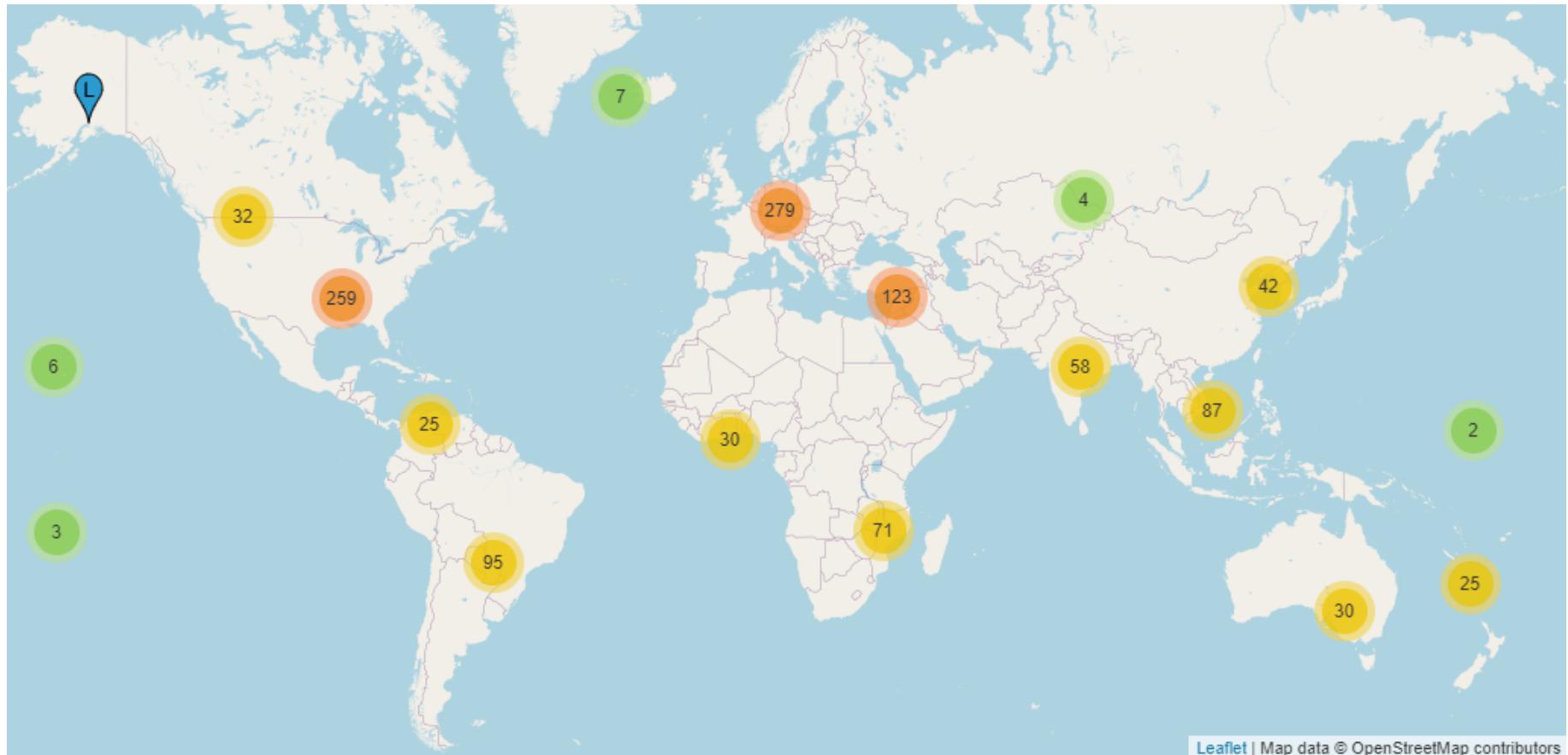
- Il client chiede al root server di trovare il DNS server del dominio `.com`
- Il client chiede al server DNS `.com` di trovare il server DNS di `amazon.com`
- Infine il client chiede al server DNS di `amazon.com` di restituirci l'indirizzo IP del server `www` in quel dominio, cioè `www.amazon.com`

DNS: server DNS radice

- Contattato da un server DNS locale che non può tradurre il nome
- Server DNS radice:
 - ❖ contatta un server DNS autorizzato se non conosce la mappatura
 - ❖ ottiene la mappatura
 - ❖ restituisce la mappatura al server DNS locale



DNS: server DNS radice



Mappa delle istanze DNS radice: <https://root-servers.org/>
Concetto di *anycasting* (<https://it.wikipedia.org/wiki/Anycast>)

Server TLD e server di competenza

- Server TLD (top-level domain):
 - Si occupano dei domini com, org, net, edu, ecc. e di tutti i domini locali di alto livello, quali it, uk, fr, ca e jp
 - Network Solutions gestisce i server TLD per il dominio com
 - Educause gestisce quelli per il dominio edu
- Server di competenza (*authoritative server*)
 - Ogni organizzazione dotata di host Internet pubblicamente accessibili (quali i server web e i server di posta) deve fornire i record DNS di pubblico dominio che mappano i nomi di tali host in indirizzi IP
 - Possono essere mantenuti dall'organizzazione o dal service provider

Server DNS locale

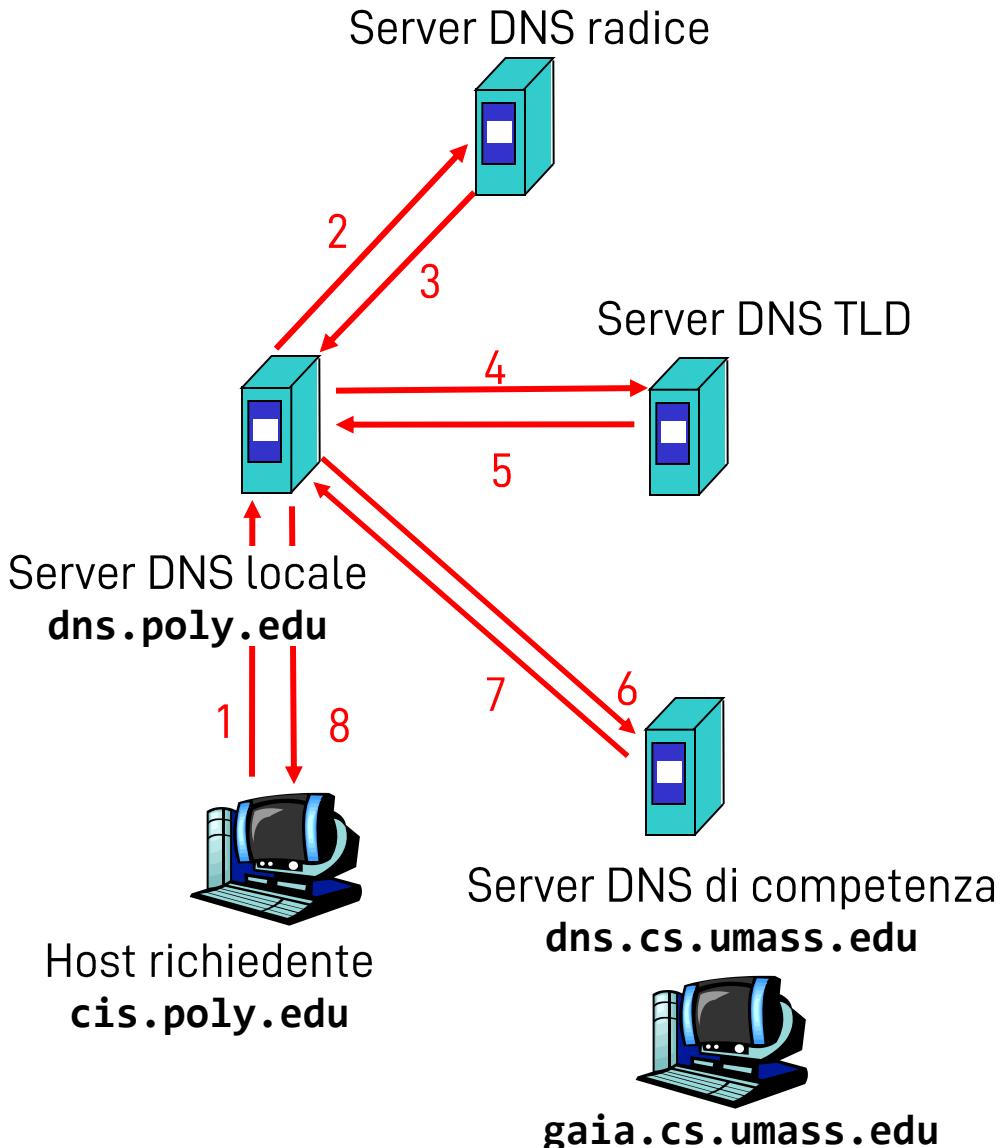
- Non appartiene strettamente alla gerarchia dei server
- Ciascun ISP (università, società, ISP residenziale) ha un server DNS locale
 - ❖ Detto anche "default name server"
- Quando un host effettua una richiesta DNS, la query viene inviata al suo server DNS locale
 - ❖ Il server DNS locale opera da proxy e inoltra la query in una gerarchia di server DNS

Esempio

- L'host **cis.poly.edu** vuole l'indirizzo IP di **gaia.cs.umass.edu**

Query iterativa:

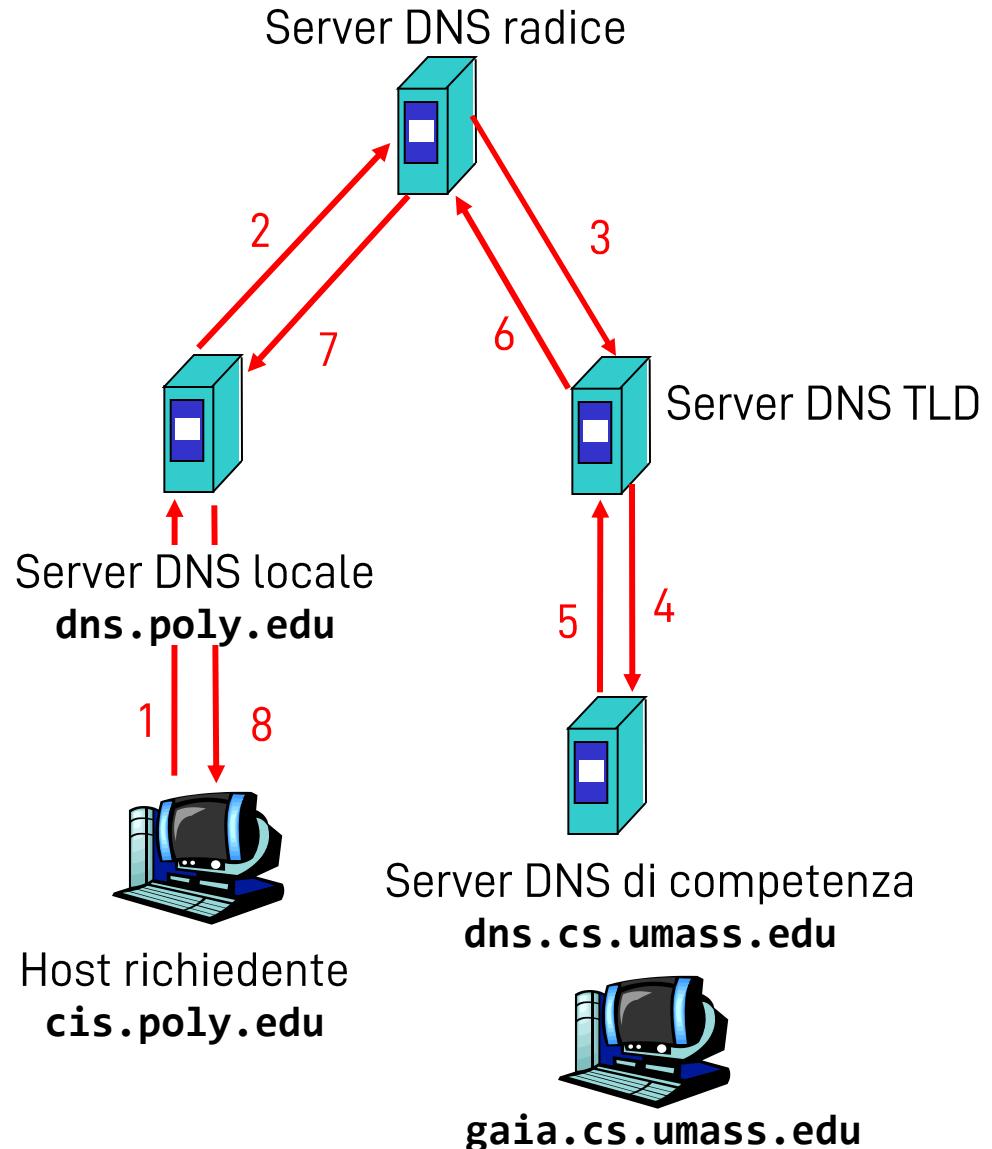
- Il server contattato risponde con il nome del server da contattare
- 8 MESSAGGI
- “Io non conosco questo nome, ma puoi chiederlo a questo server”



Esempio

Query ricorsiva:

- Affida il compito di tradurre il nome al server DNS contattato
- D: E' più efficiente questa maniera o la precedente?

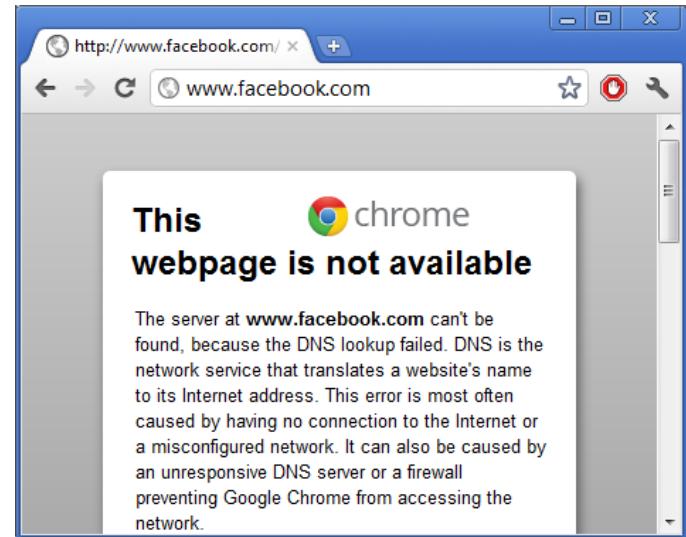


DNS: caching e aggiornamento dei record

- Una volta che un server DNS impara la mappatura, la mette nella *cache*
 - ❖ le informazioni nella cache vengono invalidate (spariscono) dopo un certo periodo di tempo
 - ❖ tipicamente un server DNS locale memorizza nella cache gli indirizzi IP dei server TLD
 - quindi i server DNS radice non vengono visitati spesso
- I meccanismi di aggiornamento/notifica sono progettati da IETF
 - ❖ RFC 2136
 - ❖ <http://www.ietf.org/html.charters/dnsind-charter.html>
- **D:** Quali problemi possono sorgere abilitando una cache DNS?

E se i DNS radice falliscono?

- MOLTO difficile che succeda
 - ❖ Infrastruttura estremamente ridondata
 - ❖ Gli indirizzi IP dei server TLD sono cached quasi ovunque
- Attacco Distributed Denial-of-Service (DDoS) di ottobre 2002
 - ❖ Molto massiccio
 - ❖ ...ma del tutto inefficace (gli utenti quasi non se ne sono accorti)
- Attacchi più mirati hanno maggiori probabilità di successo
 - ❖ Attacco al DNS locale → impedisce l'accesso al DNS
 - ❖ Attacco al DNS autoritativo → impedisce l'accesso al dominio



Resource Record (RR) nel DNS

DNS: database distribuito che memorizza i record di risorsa (**RR**)

Formato RR: **(name, value, type, ttl)**

Type=A

- ❖ Associazione standard tra nome dell'host e indirizzo IP
- ❖ **name** è il nome dell'host
- ❖ **value** è l'indirizzo IP

Type=NS

- ❖ **name** è il dominio (ad esempio `foo.com`)
- ❖ **value** è il nome dell'host del server di competenza di questo dominio

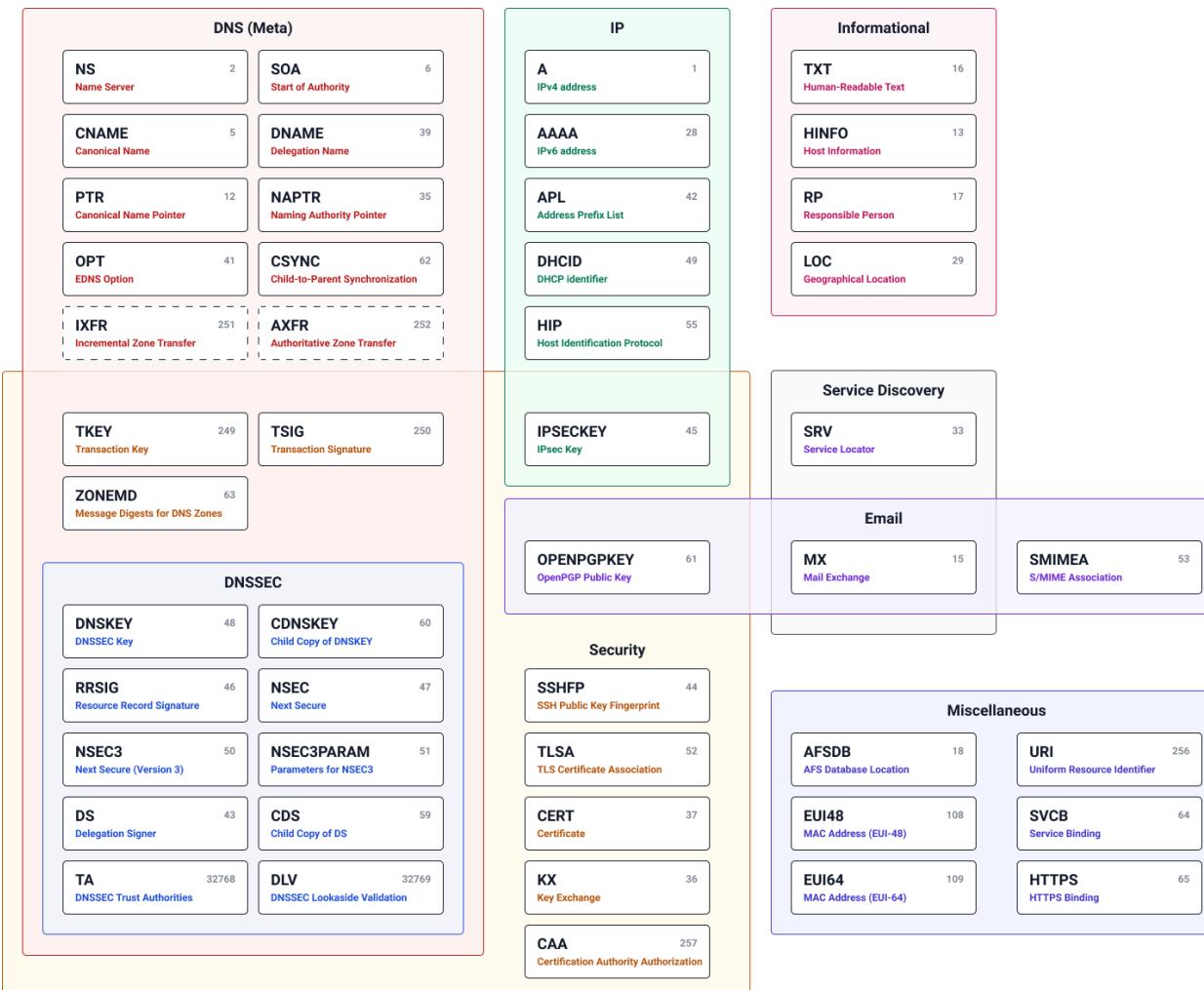
Type=CNAME

- ❖ **name** è il nome alias di qualche nome "canonico" (nome vero)
`www.ibm.com` è in realtà
`servereast.backup2.ibm.com`
- ❖ **value** è il nome canonico

Type=MX

- ❖ **value** è il nome del server di posta associato a **name**

DNS Types



Messaggi DNS

Protocollo DNS: domande (query) e messaggi di *risposta*, entrambi con lo stesso *formato*

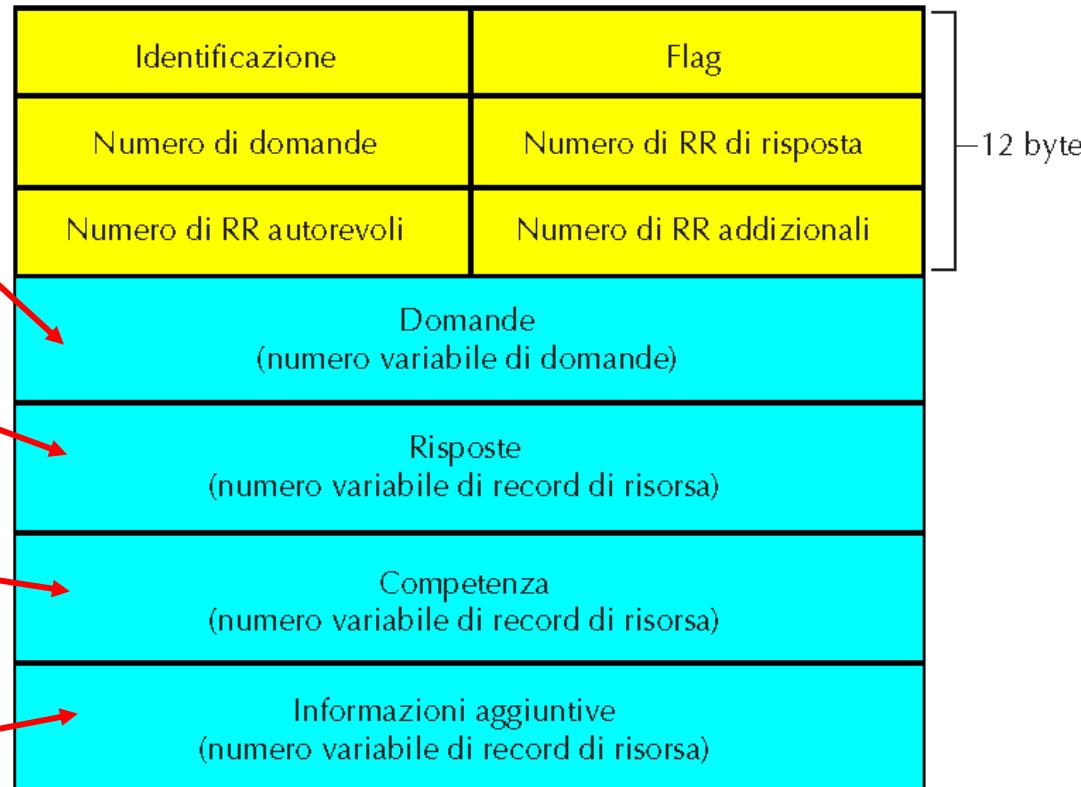
Intestazione del messaggio

- **Identificazione:** numero di 16 bit per la domanda; la risposta alla domanda usa lo stesso numero
- **Flag:**
 - ❖ domanda o risposta
 - ❖ richiesta di ricorsione
 - ❖ ricorsione disponibile
 - ❖ risposta di competenza



Messaggi DNS

- Campi per il nome richiesto e il tipo di domanda
- RR nella risposta alla domanda
- Record per i server di competenza
- Informazioni extra che possono essere usate



Inserire record nel database DNS

- Esempio: abbiamo appena avviato la nuova società "Network Utopia"
- Registriamo il nome **networkutopia.com** presso un **registrar** (ad esempio, Network Solutions)
- Registrar: società che controlla l'unicità del nome, lo inserisce nel database DNS come sotto, e vi tariffa un costo per il servizio
 - ❖ Forniamo al registrar i nomi e gli indirizzi IP dei server DNS di competenza (primario e secondario)
 - ❖ Il registrar inserisce due RR nel server **TLD** per il dominio **com**:
 - ❖ (networkutopia.com, dns1.networkutopia.com, NS)
 - ❖ (dns1.networkutopia.com, 212.212.212.1, A)
- Inseriamo nel server **di competenza** un record tipo A per www.networkutopia.com e un record tipo MX per networkutopia.com
- In che modo gli utenti otterranno l'indirizzo IP del nostro sito web?

Cosa succede quando si cerca di accedere a www.networkutopia.com

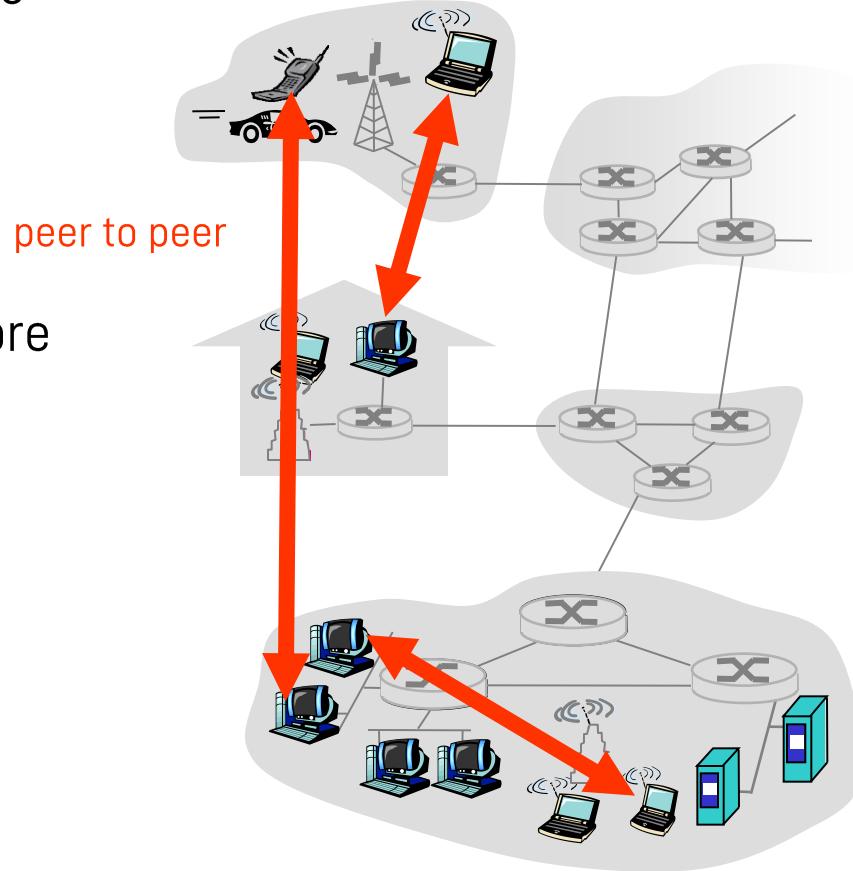
- Es. un utente in Australia vuol vedere www.networkutopia.com
 1. Query al server DNS locale
 2. Il DNS locale contatta il server TLD del dominio com
(e anche un root server se il server TLD di com non è in cache)
 3. Il server TLD trova il record (precedentemente inserito dal registrar)
 4. Il server TLD risponde con il record al DNS locale
 5. Il DNS locale manda query a 212.212.212.1 chiedendo il record Type A corrispondente a www.networkutopia.com
 6. Il DNS locale manda la risposta ad Alice, diciamo 212.212.71.4
 7. Alice può finalmente aprire una connessione HTTP verso 212.212.71.4
(aka www.networkutopia.com) e scaricare la pagina

Capitolo 2: Livello applicazione

- 2.1 Principi delle applicazioni di rete
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Posta elettronica
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Condivisione di file P2P
- 2.7 Cloud computing
- 2.8 Programmazione delle socket (cenni)

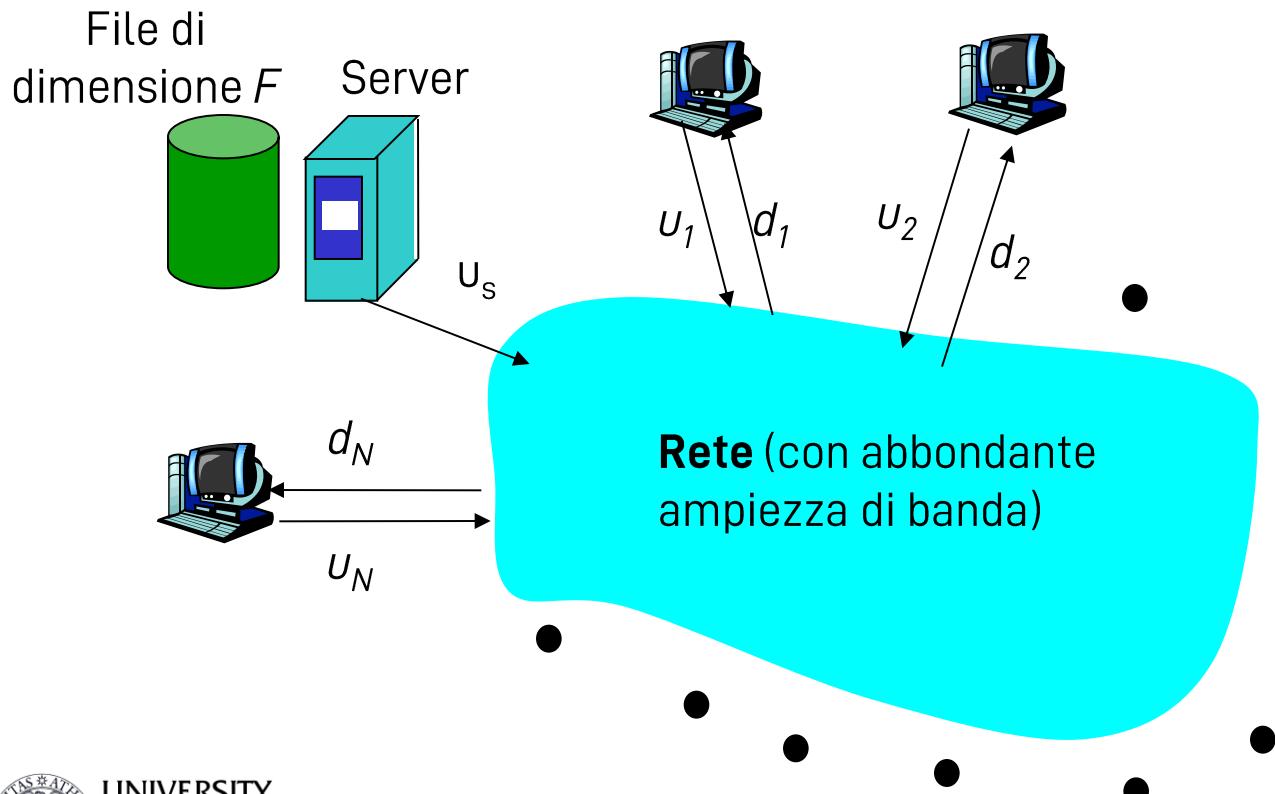
Architettura P2P pura

- Non c'è un server sempre attivo
- Coppie arbitrarie di host (peer) comunicano direttamente tra loro
- I peer non devono necessariamente essere sempre attivi, e cambiano indirizzo IP
- Tre argomenti chiave:
 - Distribuzione di file
 - Ricerca informazioni
 - Caso di studio: Skype



Distribuzione di file: confronto tra Server-Client e P2P

Domanda: Quanto tempo ci vuole per distribuire file da un server a N peer?



U_s : bit rate di upload del collegamento di accesso del server

U_i : bit rate di upload del collegamento di accesso dell' i -esimo peer

d_i : bit rate di download del collegamento di accesso dell' i -esimo peer

•

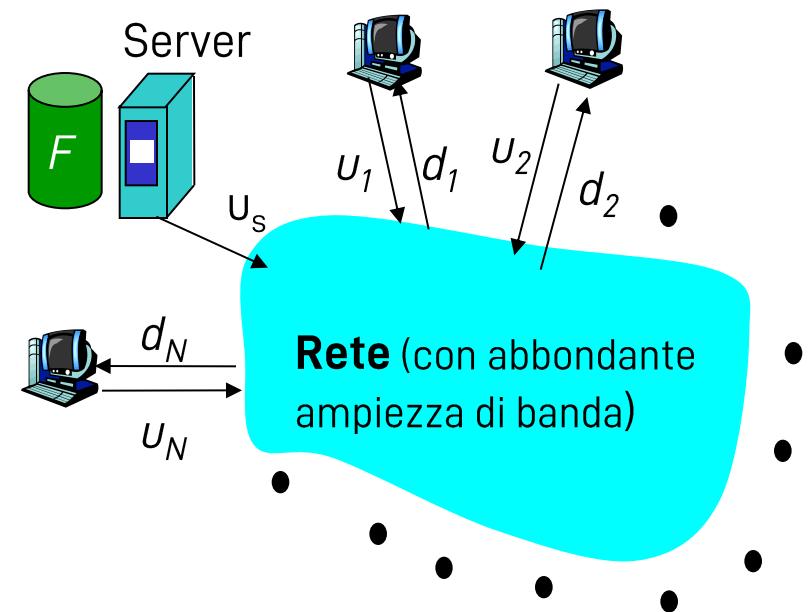
•

•

•

Distribuzione di file: server-client

- Il server invia in sequenza N copie:
 - ❖ $\text{Tempo} = NF/u_s$
- Il client i impiega il tempo F/d_i per scaricare



Tempo per distribuire F

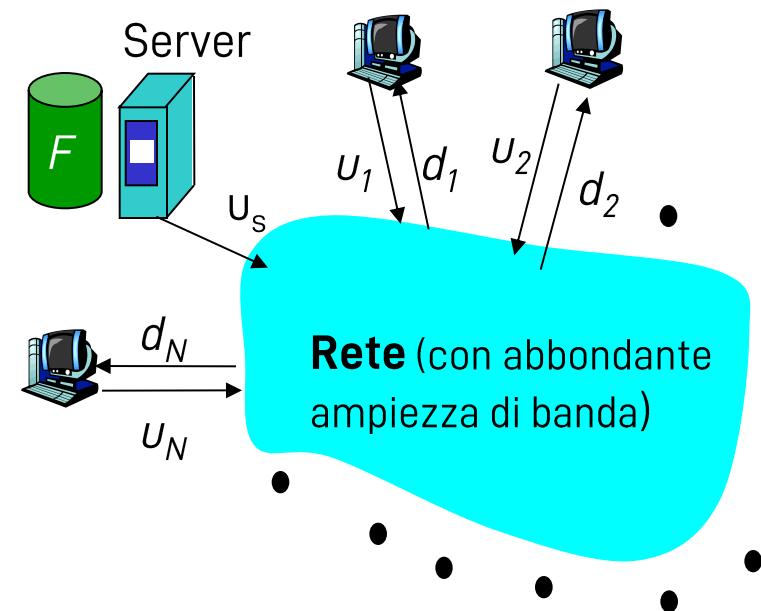
a N client usando
l'approccio client/server

$$= d_{cs} = \max \left\{ NF/u_s, \frac{F}{\min_i(d_i)} \right\}$$

aumenta linearmente
con N peer

Distribuzione di file: P2P

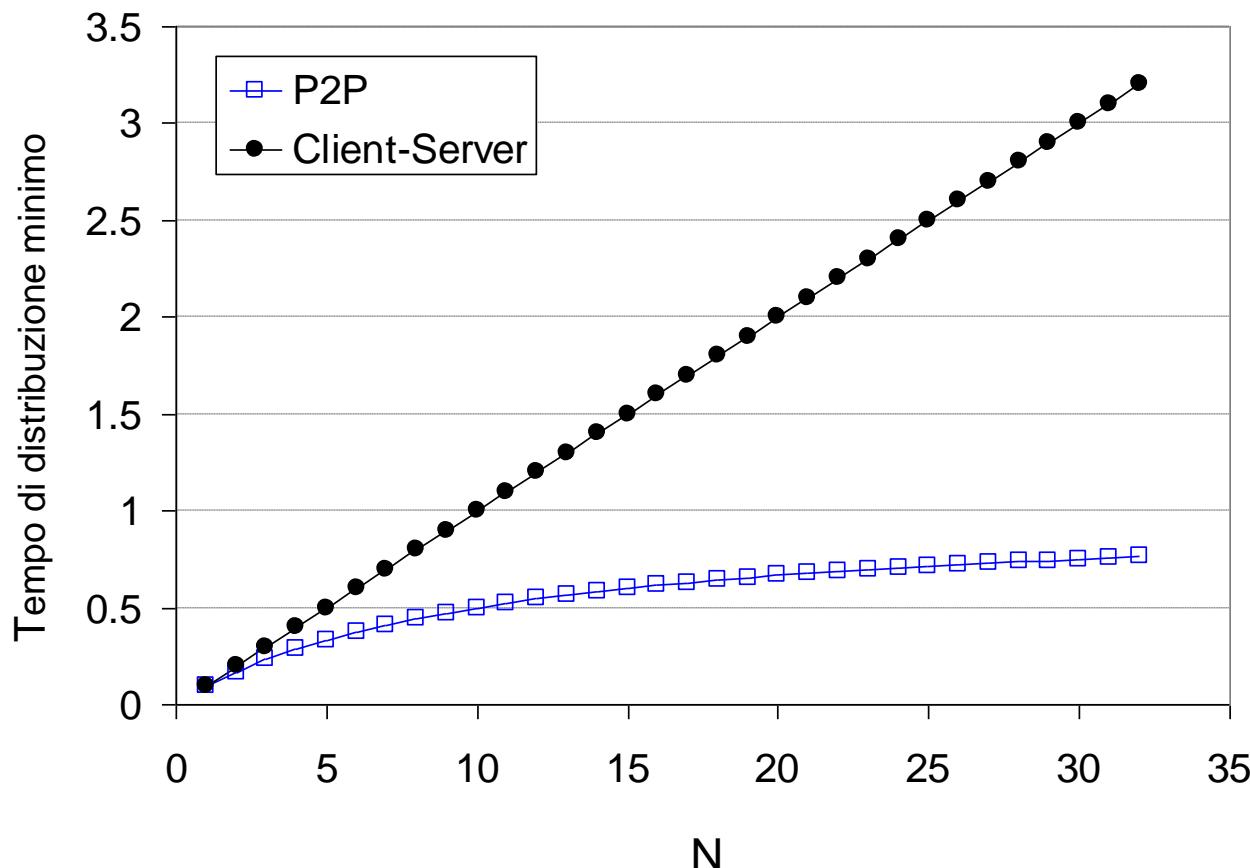
- il server deve inviare una copia nel tempo F/u_s
- il client i impiega un tempo F/d_i per il download
- Devono essere scaricati NF bit
- Il più veloce tasso di upload è:
 $u_s + \sum_i u_i$



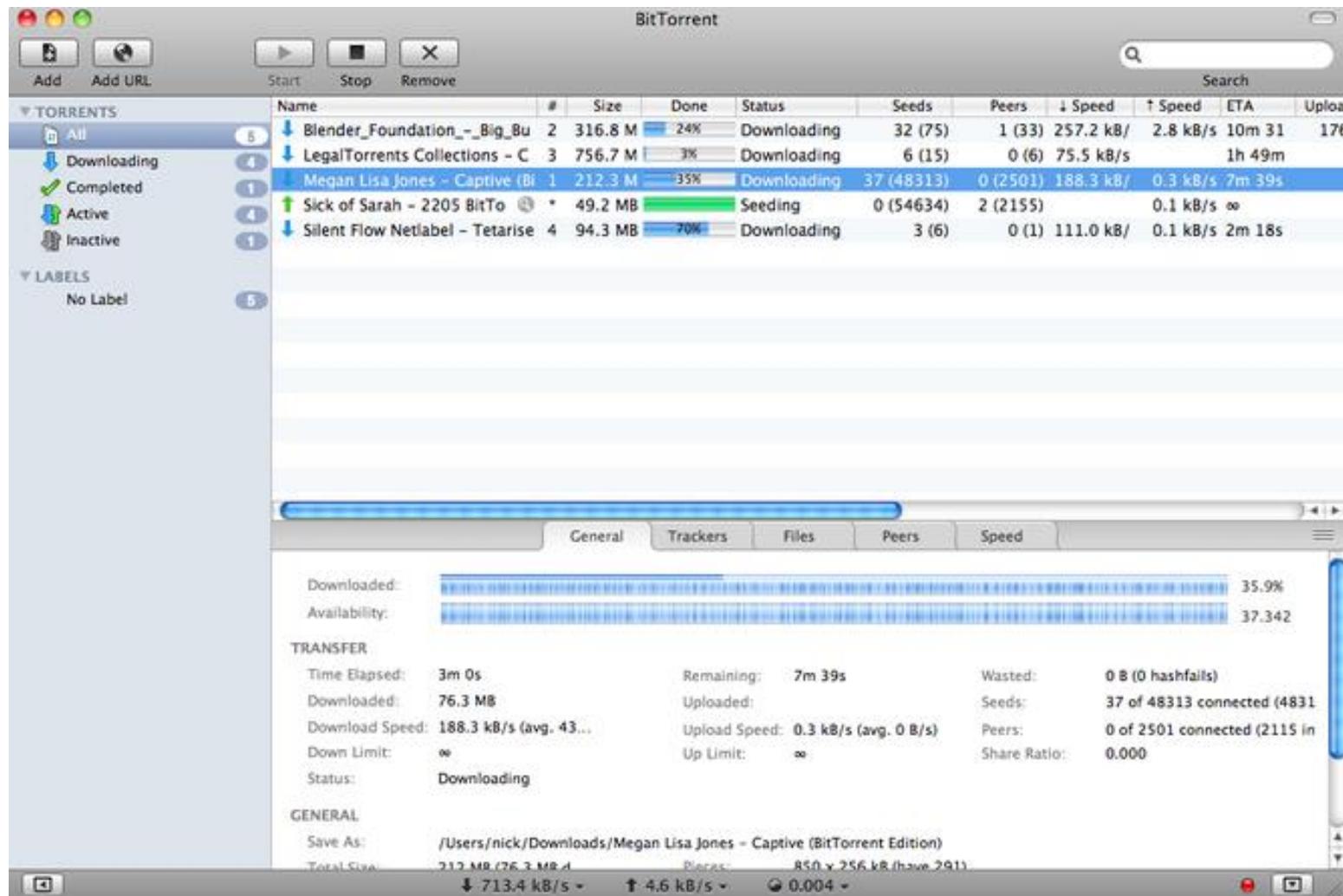
$$d_{P2P} = \max \left\{ \frac{F}{u_s}, \frac{F}{\min(d_i)}, \frac{NF}{u_s + \sum_i u_i} \right\}$$

Confronto tra server-client e P2P: un esempio

Tasso di upload del client = u , $F/u = 1$ ora, $u_s = 10u$, $\min_i(d_i) \geq u_s$



Distribuzione di file: BitTorrent

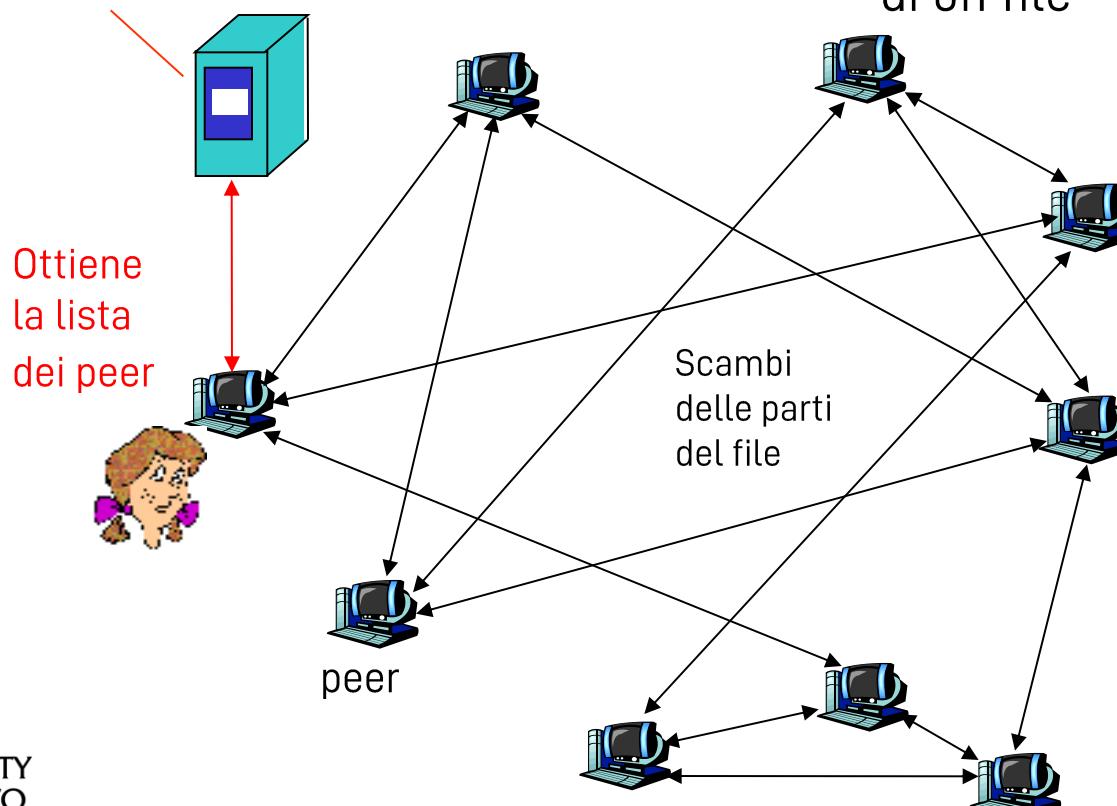


Distribuzione di file: BitTorrent

□ Distribuzione di file P2P

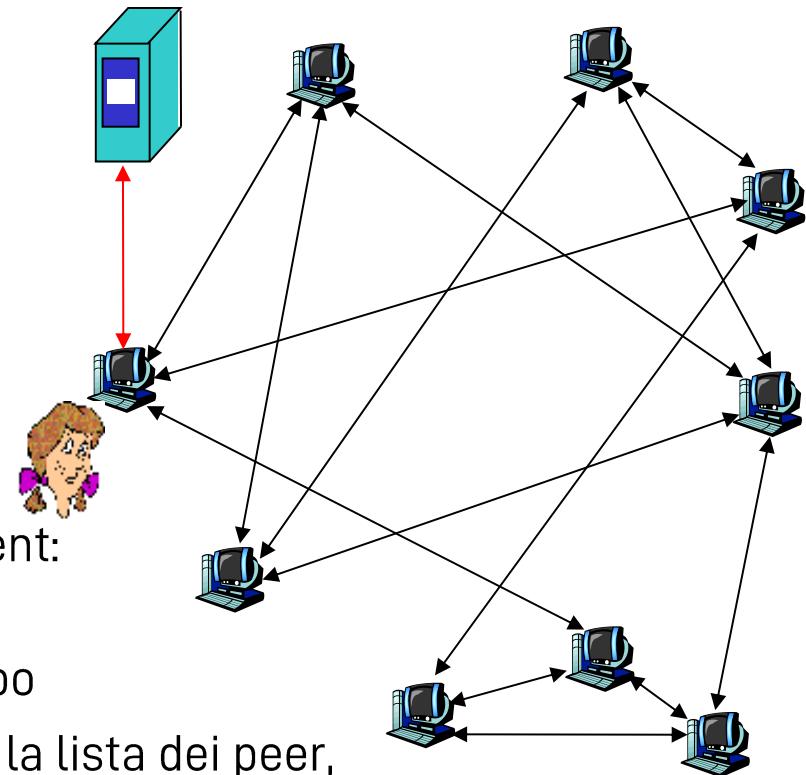
tracker: tiene traccia dei peer che partecipano

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



BitTorrent

- Il file viene diviso in parti (*chunk*), tipicamente da 256 kByte
- Quando un peer entra a far parte del torrent:
 - ❖ Non possiede nessuna parte del file, ma le accumula col passare del tempo
 - ❖ Si registra presso il tracker per avere la lista dei peer, e si collega ad un sottoinsieme di vicini (*neighbors, max 4*)
 - ❖ Tit-for-tat: se un vicino mi dà i propri contenuti, si alza la probabilità che io gli invii i miei (e viceversa: metto in «choking» chi mi invia poco o nulla)
- Mentre effettua il download, il peer carica le sue parti su altri peer
- I peer possono entrare e uscire a piacimento dal torrent
- Una volta ottenuto l'intero file, il peer può lasciare il torrent (egoisticamente, *leech*) o rimanere collegato (altruisticamente, *seeder*)



P2P: ricerca di informazioni

Indice nei sistemi P2P: corrispondenza tra le informazioni e la loro posizione negli host, di solito è una Distributed Hash Table

File sharing (es. e-mule)

- L'indice tiene traccia dinamicamente della posizione dei file che i peer condividono.
- I peer comunicano all'indice ciò che possiedono.
- I peer consultano l'indice per determinare dove trovare i file.

Messaggeria istantanea

- L'indice crea la corrispondenza tra utenti e posizione.
- Quando l'utente lancia l'applicazione, informa l'indice della sua posizione
- I peer consultano l'indice per determinare l'indirizzo IP dell'utente.

P2P: directory centralizzata

Progetto originale di "Napster"

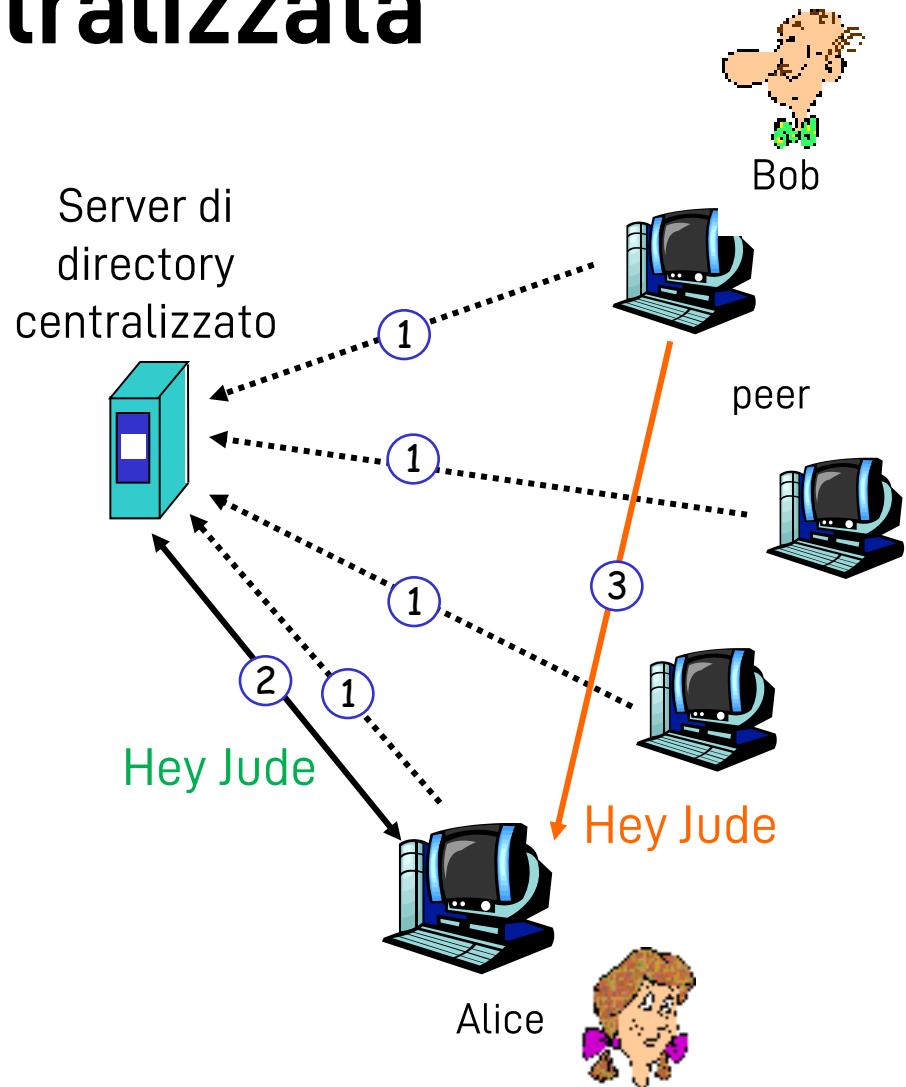
1) quando il peer si collega, informa il server centrale:

- ❖ indirizzo IP
- ❖ contenuto

2) Alice cerca la canzone "Hey Jude"

3) Il server informa Alice che Bob possiede il file

4) Alice richiede il file a Bob



P2P: problemi con la directory centralizzata

- Single point of failure
- Collo di bottiglia per le prestazioni
- Violazione del diritto d'autore

Il trasferimento dei file è distribuito, ma il processo di localizzazione è fortemente centralizzato

Query flooding

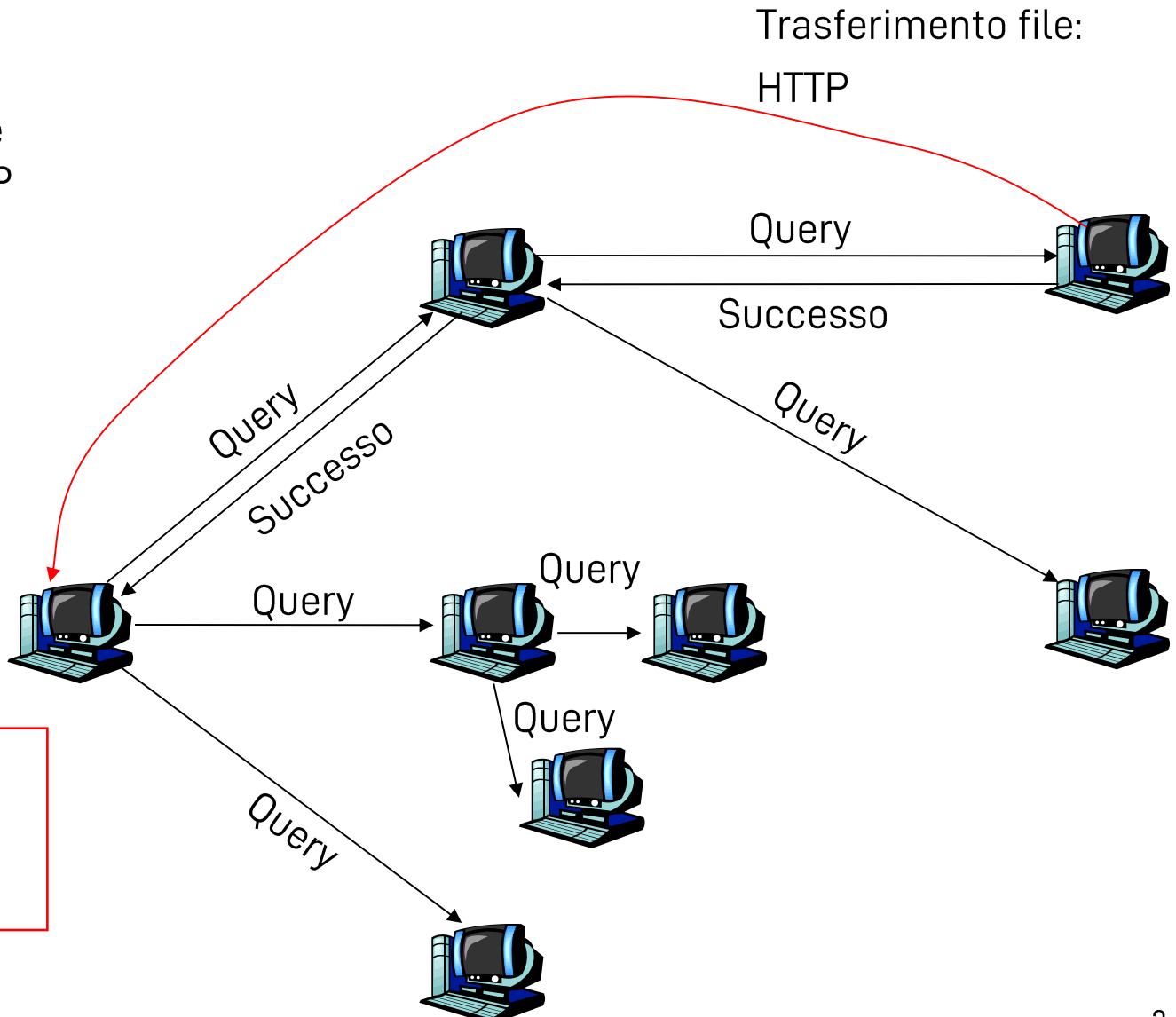
- Completamente distribuito
 - ❖ Nessun server centrale
- Protocollo di pubblico dominio usato da Gnutella
- Ciascun peer indica i file che rende disponibili per la condivisione (e nasconde ogni altro file)

Rete di copertura (overlay network): grafo

- Arco tra i peer X e Y se c'è una connessione (TCP)
- Tutti i peer attivi e gli archi formano la rete di copertura
- Un arco è un collegamento virtuale e *non* fisico
- Un dato peer sarà solitamente connesso con meno di 10 peer vicini nella rete di copertura

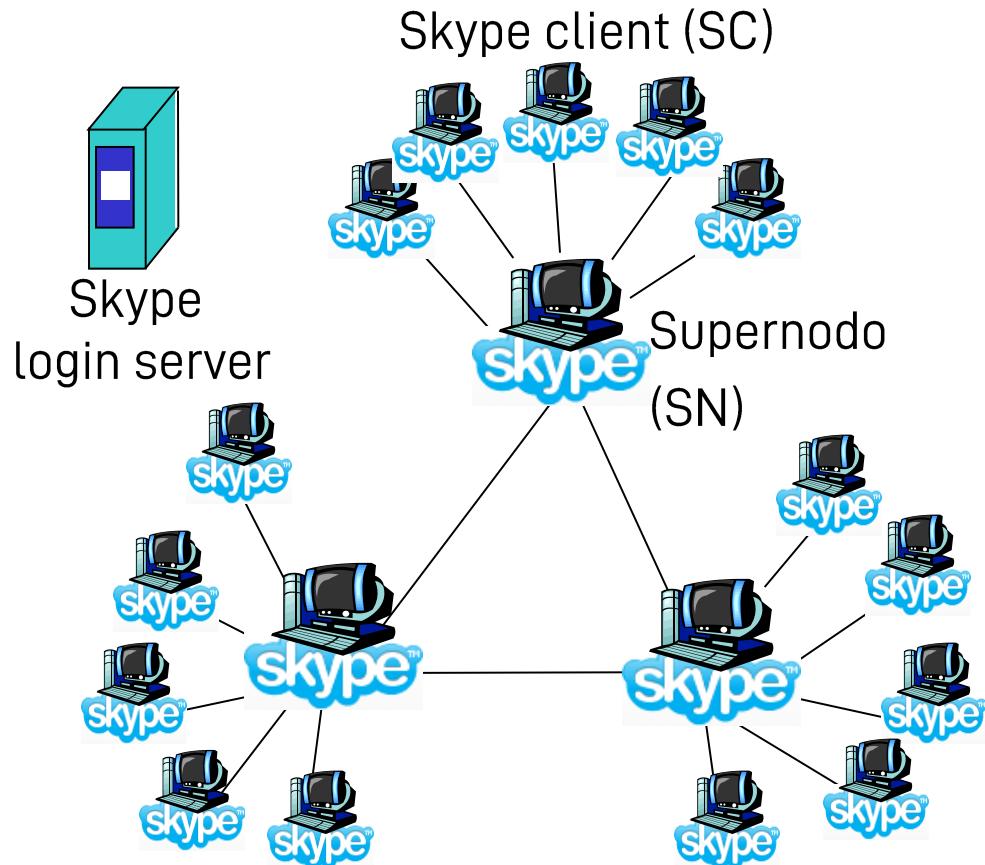
Query flooding

- ☐ Il messaggio di richiesta è trasmesso sulle connessioni TCP esistenti
- ☐ Il peer inoltra il messaggio di richiesta
- ☐ Il messaggio di successo è trasmesso sul percorso inverso



Esempio di sistema ibrido: Skype

- Intrinsecamente P2P:
coppie di utenti
comunicano tra loro
- Protocollo proprietario
(dedotto mediante
reverse engineering)
- Copertura gerarchica
tramite i supernodi
- L'indice crea
corrispondenza tra nomi
utente e indirizzi IP

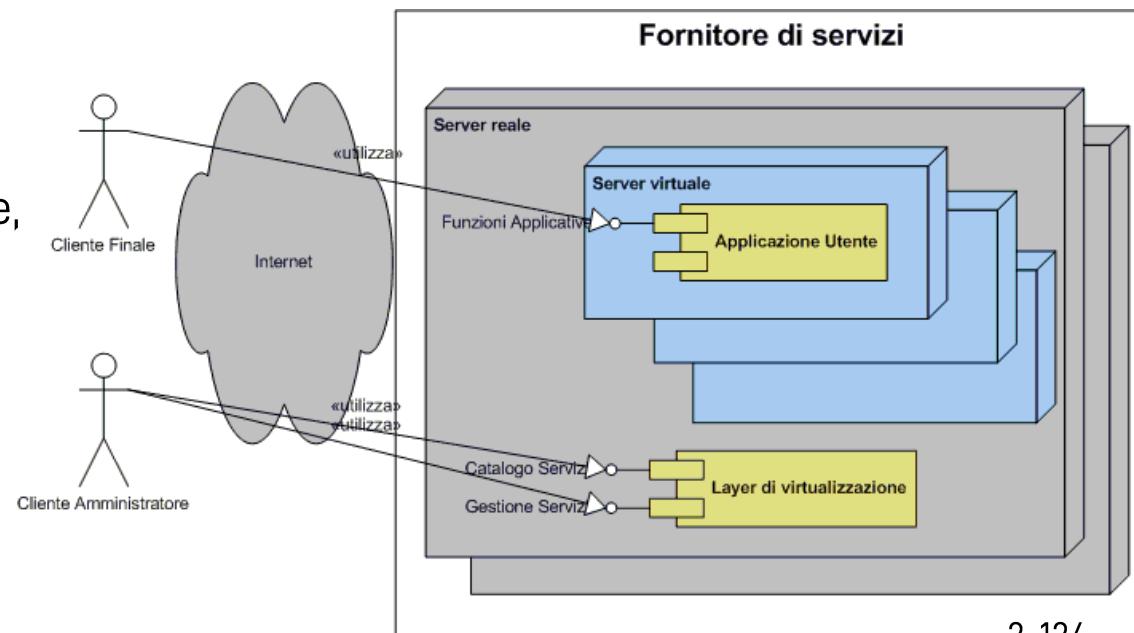


Capitolo 2: Livello applicazione

- 2.1 Principi delle applicazioni di rete
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Posta elettronica
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Condivisione di file P2P
- 2.7 Cloud computing
- 2.8 Programmazione delle socket (cenni)

Cloud computing

- Il cloud computing prevede uno o più server reali, generalmente organizzati in un'architettura ad alta affidabilità e fisicamente collocati presso il data center del fornitore del servizio
 - Il fornitore di servizi espone delle interfacce per elencare e gestire i propri servizi
 - Il cliente amministratore utilizza tali interfacce per selezionare il servizio richiesto (ad es. un server virtuale completo oppure solo storage) e per amministrarlo (configurazione, attivazione, disattivazione)
 - Il cliente finale utilizza il servizio configurato dal cliente amministratore



Cloud computing

- Le caratteristiche fisiche dell'implementazione (server reale, localizzazione del data center) sono irrilevanti
- Criticità:
 - Sicurezza e privacy
 - Problemi internazionali di natura economica e politica
 - Continuità del servizio offerto
 - Difficoltà di migrazione dei dati

“ Rounding off the numbers, twenty-four reels of tape at 170 megabytes each is 4080 megabytes. Three and a half hours is 210 minutes. 4080 megabytes divided by 210 works out to about 19.4 megabytes per minute, or 32.3 kilobytes per second (258.4kilobits per second) - over 100 times faster than a 2400 bps data circuit of the time. Note that the incident above involved only 24 reels - which didn't come anywhere near filling the station wagon, in fact the two boxes of tapes didn't even fill the front passenger seat. (as an aside, a station wagon is known as an estate car or estate in other parts of the world). Incidentally, that conversation was the first time your contributor ever heard the term backhoe fade used to describe accidental massive damage to an underground cable (compare it to the term rain fade used to describe a fade-out of a point-to-point microwave radio path due to the absorptive effect of water in the air).

https://www.reddit.com/r/mildlyinteresting/comments/20jlv3/never_underestimate_the_bandwidth_of_a_station/

La «banda» di FedEx: <https://what-if.xkcd.com/31/>

Cloud computing

- Dove si trovano i data center?

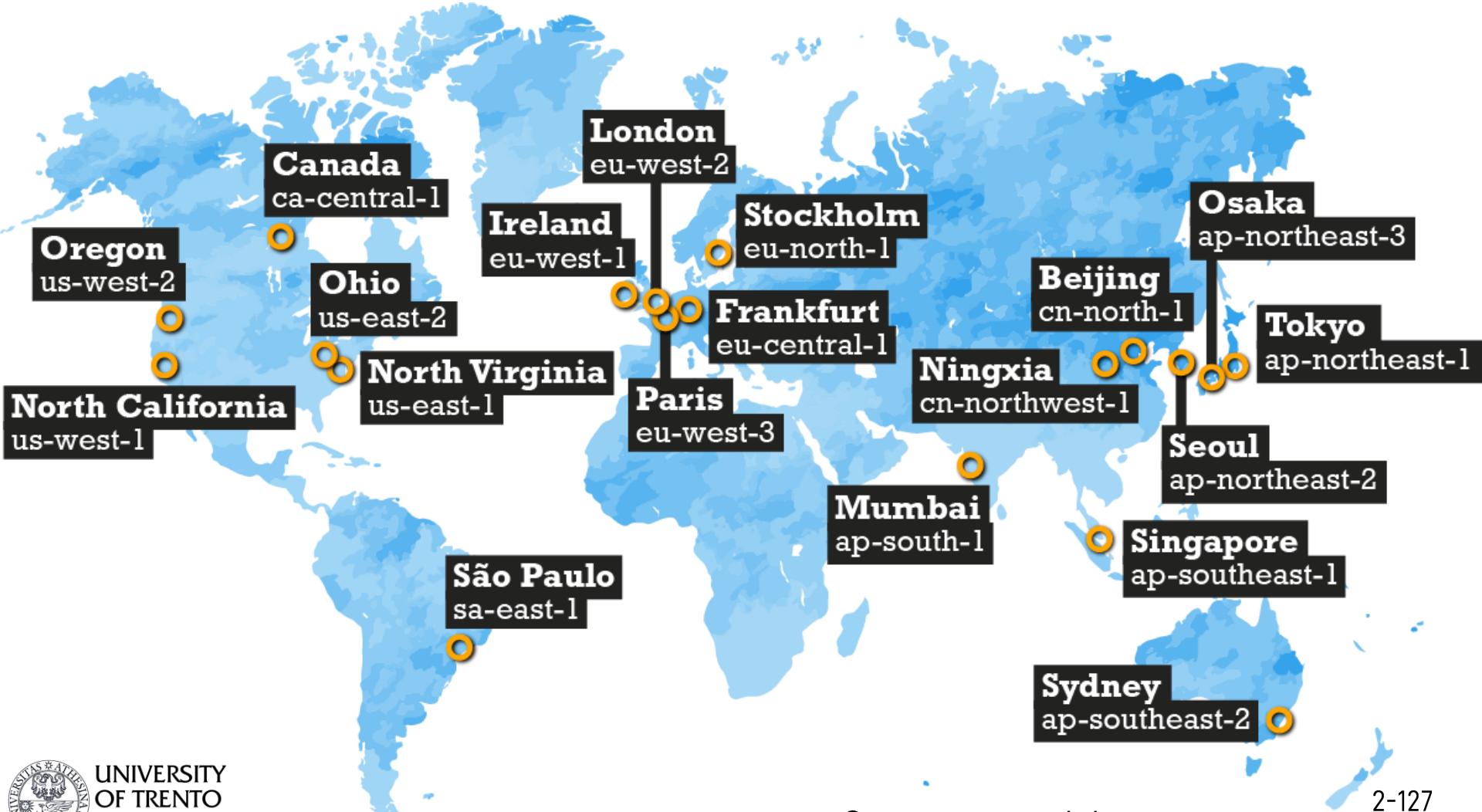
Source: <http://www.vox.com/>



Try also: <https://map.datacente.rs/>

Cloud computing

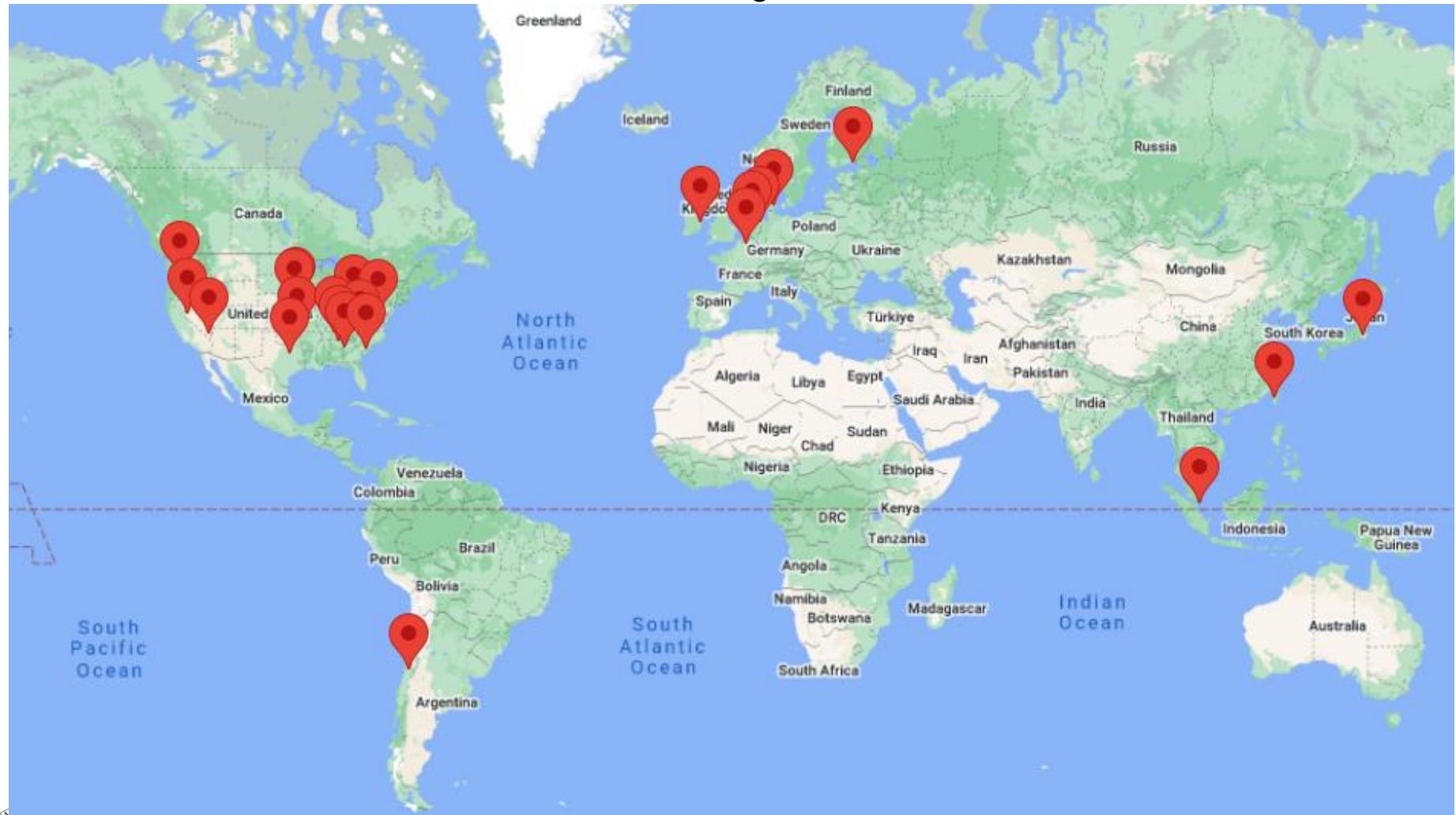
- Dove si trovano i data center (Amazon)?



Source: getcodelove.com

Cloud computing

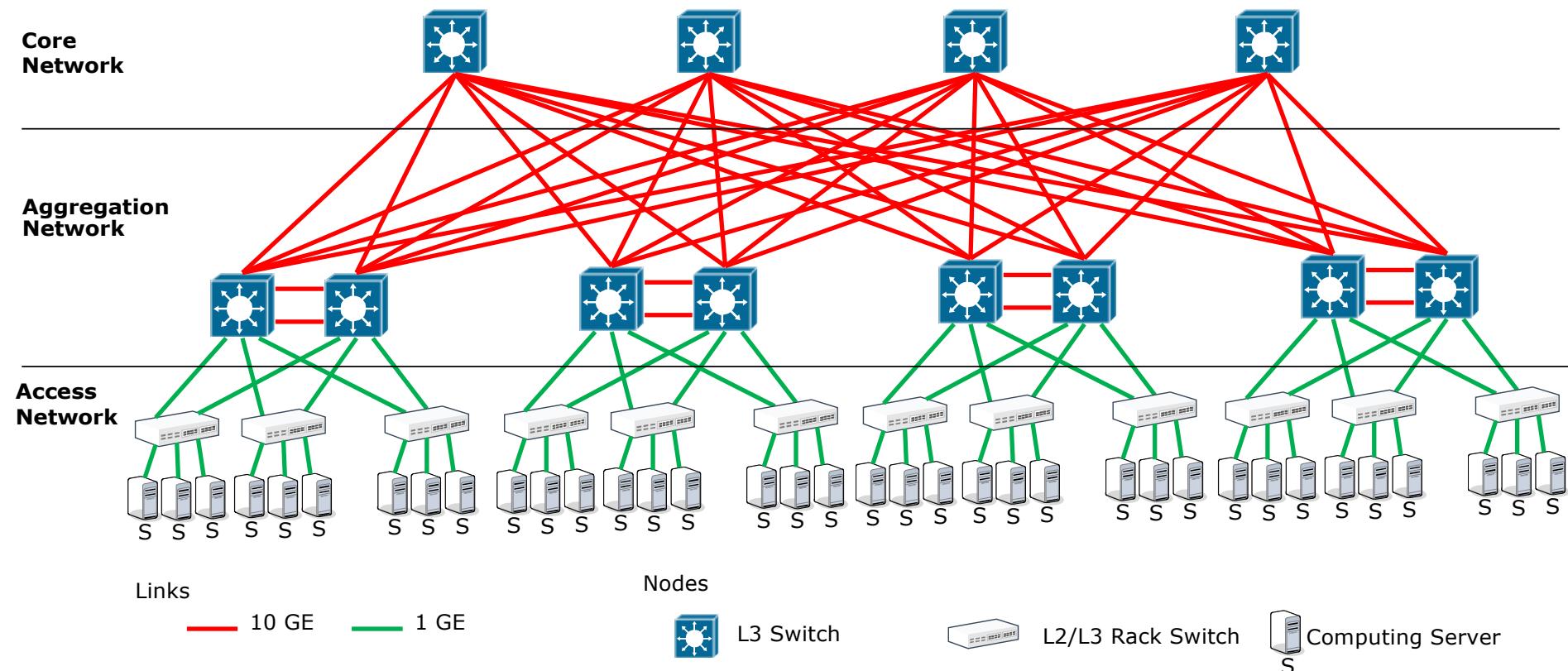
- Dove si trovano i data center (Google)?



Source: <https://www.google.com/about/datacenters/locations/>

Cloud computing

- Come è fatto un (mega) data center dal punto di vista architetturale?



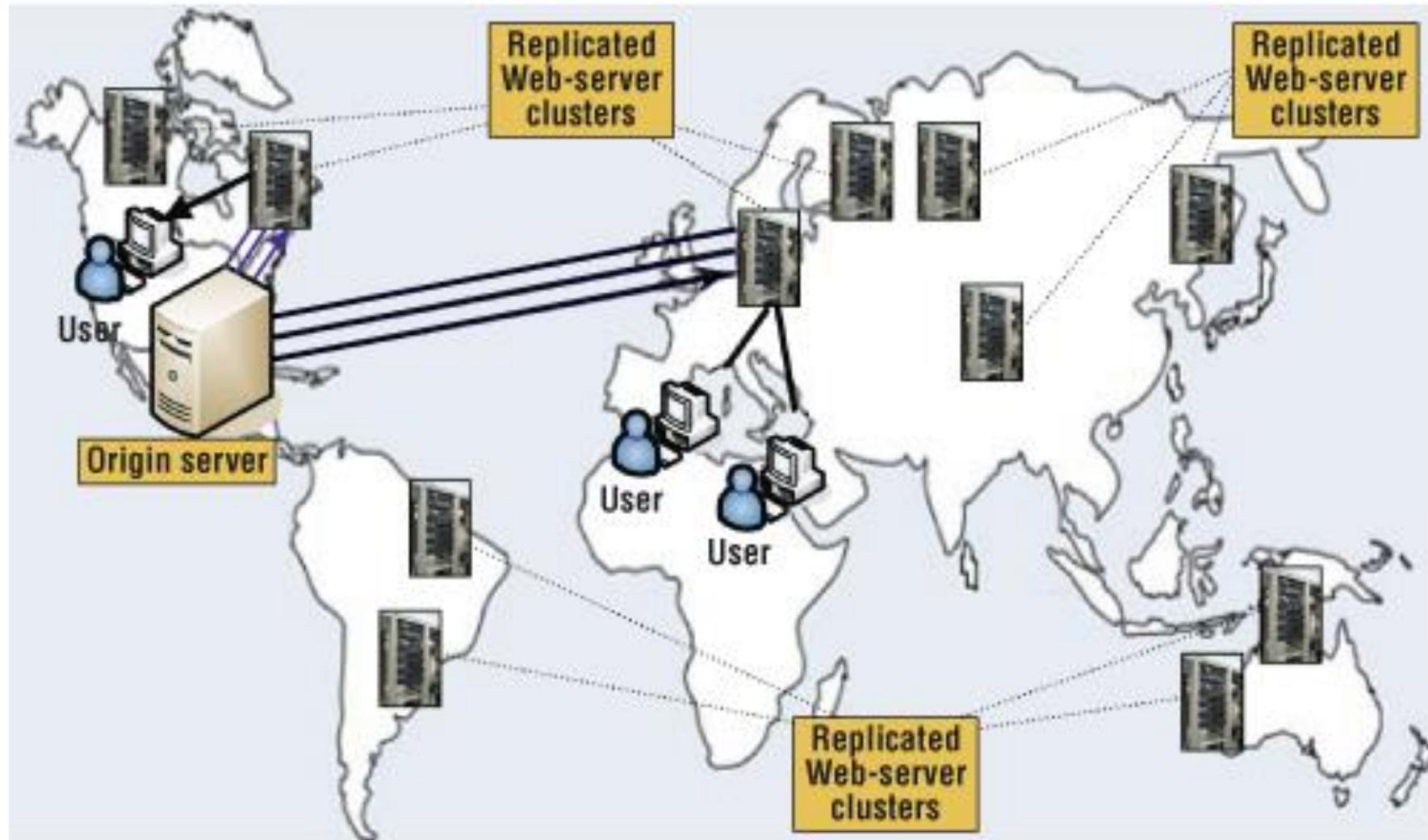
I data center di Google (dati 2016)

- **14** “*mega data-centers*” (8 in nord america, 4 in europa, 4 in asia)
 - ❖ Circa 100.000 server per data center
 - ❖ Servono contenuti dinamici e personalizzati (ad, ricerche, gmail)
- Circa **50** cluster di calcolo “*bring home*” negli IXP
 - ❖ 100-500 server per cluster
 - ❖ Servono contenuti statici, inclusi i video Youtube
- Centinaia di cluster “*enter-deep*” nelle reti di accesso degli ISP
 - ❖ In ogni cluster: 1 rack, qualche decina di server
 - ❖ Servono contenuti statici (es., le parti statiche delle pagine di risposta alle ricerche)
 - Inoltre permettono il *TCP splitting* (ask me about it later)

Content Delivery Networks (CDN)

- Le Content Delivery Network (CDN) rappresentano una soluzione comune per la realizzazione di servizi su Internet
- Una CDN costruisce una rete "overlay" per la distribuzione di contenuti (social media, video, etc.)
- Il concetto è memorizzare i dati il più vicino possibile ai consumatori in modo da:
 - ❖ Ottimizzare le prestazioni di rete
 - ❖ Ridurre la latenza
 - ❖ Evitare i colli di bottiglia
- Due diverse filosofie:
 - ❖ Enter deep (es. *Akamai*): installare i server nelle reti degli ISP in tutto il mondo: migliora ritardo e throughput percepiti dagli utenti
 - ❖ Bring home (es. *Limelight*): meno server, ma installati negli IXP per poter servire le reti di più ISP contemporaneamente con la stessa infrastruttura

Content Delivery Networks (CDN)



Content Delivery Networks (CDN)

- Esempio: Facebook ha la propria CDN
- Aprite una pagina di Facebook ed esamina l'html
- Troverete riferimenti a fcdn.net, che è la CDN di Facebook
- Per scoprire dove si trova il nodo che vi serve i contenuti:
 - ❖ Inviate il comando “ping www.fcdn.net” e otterrete l'indirizzo IP di un nodo della CDN
 - ❖ Localizzate geograficamente l'indirizzo IP
 - Google “where is IP xx.xx.xx.xx?”

Capitolo 2: Livello applicazione

- 2.1 Principi delle applicazioni di rete
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Posta elettronica
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Applicazioni P2P
- 2.7 Cloud computing
- 2.8 Programmazione delle socket (cenni)

Programmazione delle socket

Obiettivo: imparare a costruire un'applicazione client/server che comunica utilizzando le socket

Socket API

- Introdotta in BSD4.1 UNIX, nel 1981
- Esplicitamente creata, usata, distribuita dalle applicazioni
- Esistono diversi tipi di socket
 - ❖ DARPA Internet Address (Internet Socket)
 - ❖ Unix Socket
 - ❖ Indirizzi X.25 (X.25 Socket)
- Paradigma client/server
- Due tipi di servizio di trasporto tramite una socket API:
 - ❖ Datagramma inaffidabile
 - ❖ Affidabile, orientata ai byte

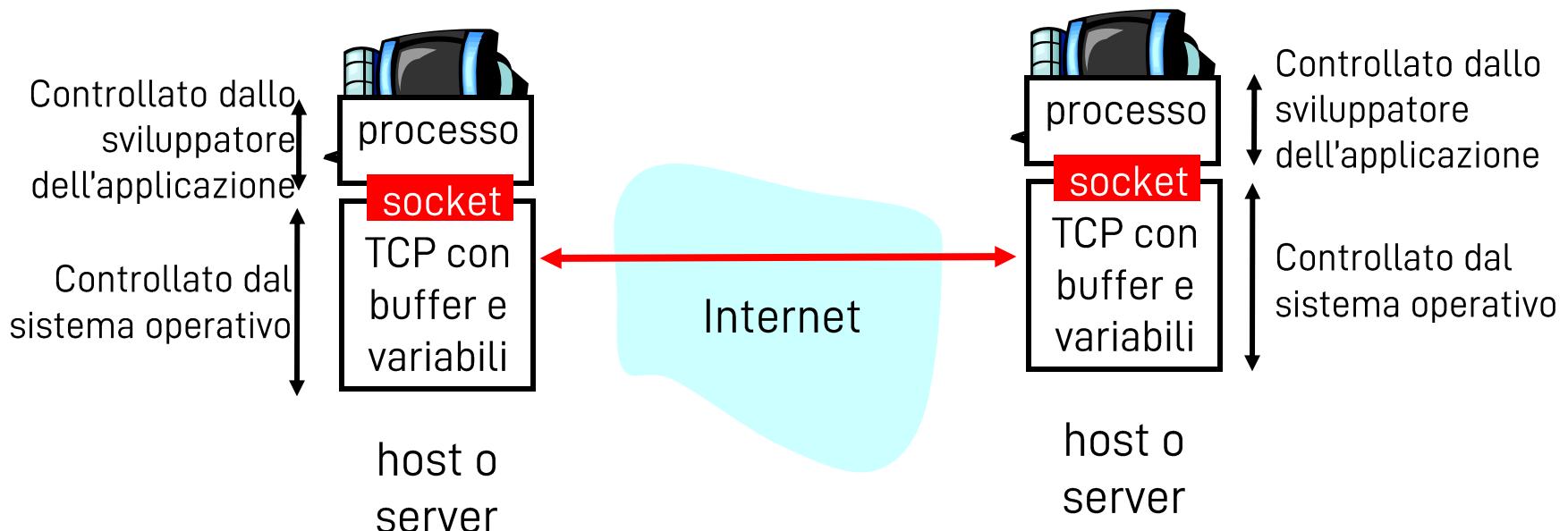
socket

Interfaccia di un *host locale*,
creata dalle applicazioni,
controllata dal SO
(una “porta”) in cui
il processo di un'applicazione può
inviare e ricevere
messaggi al/dal processo di
un'altra applicazione

Programmazione delle socket **con TCP**

Socket: una porta tra il processo di un'applicazione e il protocollo di trasporto end-end (UDP o TCP)

Servizio TCP: trasferimento affidabile di **byte** da un processo all'altro, detto anche stream socket (SOCK_STREAM)



Programmazione delle socket **con TCP**

Il client deve contattare il server

- Il processo server deve essere **in esecuzione**
- Il server deve avere creato una socket (porta) che dà il benvenuto al contatto con il client

Il client contatta il server:

- Creando una socket TCP
- Specificando l'indirizzo IP **e** il numero di porta del processo server
- Quando il **client crea la socket**: il client TCP stabilisce una connessione con il server TCP

- Quando viene contattato dal client, il **server TCP crea una nuova socket** per il processo server per comunicare con il client
 - ❖ consente al server di comunicare con più client
 - ❖ numeri di porta origine usati per distinguere i client
([maggiori informazioni nel Capitolo 3](#))

Punto di vista dell'applicazione

TCP fornisce un trasferimento di byte affidabile e ordinato ("pipe") tra client e server

Interazione delle socket client/server: TCP

Server (gira su **hostid**)

Creazione del socket
Associazione ad un porta
Ascolto sulla porta selezionata

socket()
bind()
listen()

attende la richiesta di
connessione in ingresso

accept()

legge la richiesta da
recv()

scrive la risposta a
send()

Chiusura del socket

Client

Creazione del socket
connesso a **hostid**, port=x

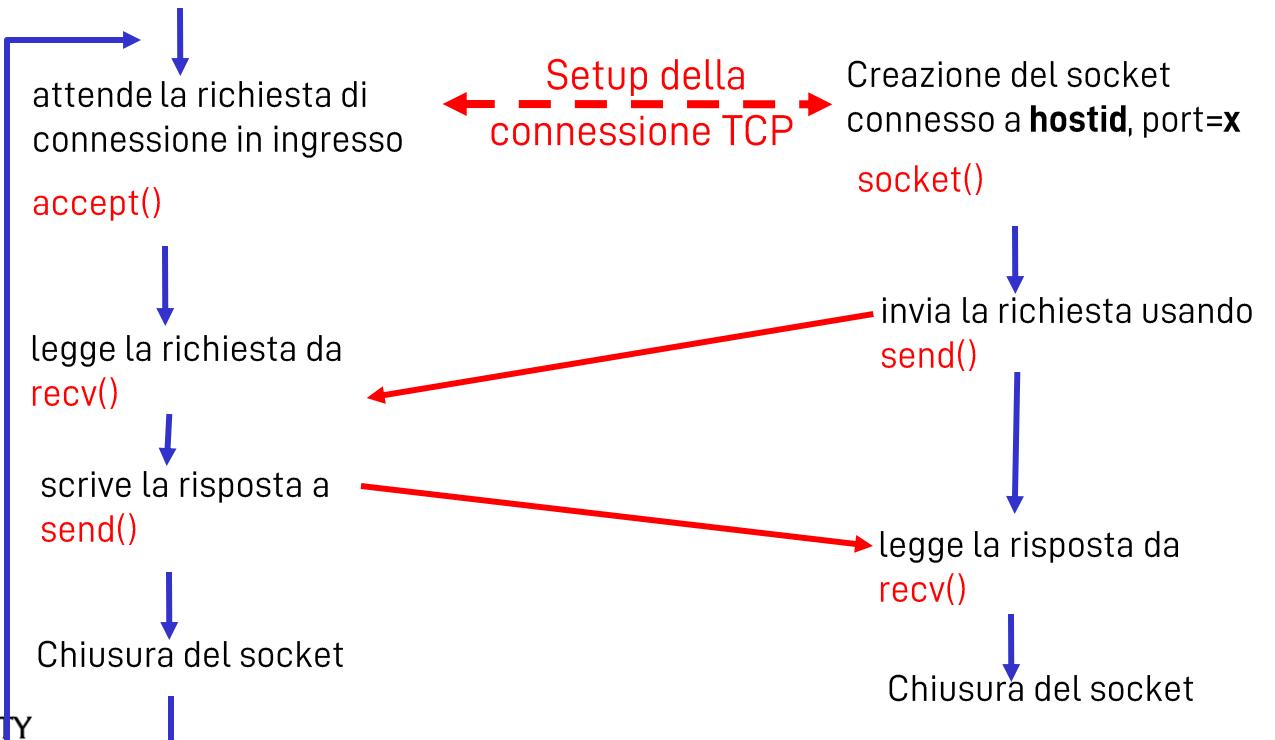
socket()

invia la richiesta usando
send()

legge la risposta da
recv()

Chiusura del socket

Setup della
connessione TCP

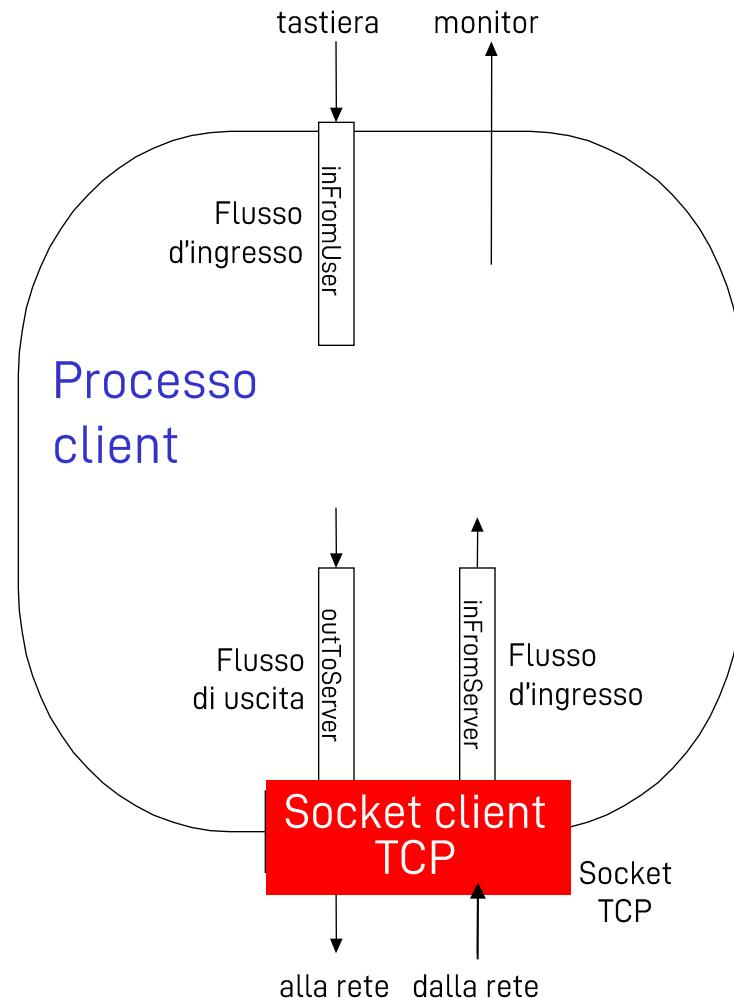


Esempio: Applicazione Client/Server in Linguaggio C (TCP)

Questa parte è fornita come esempio e non fa
parte dei contenuti d'esame

Termini

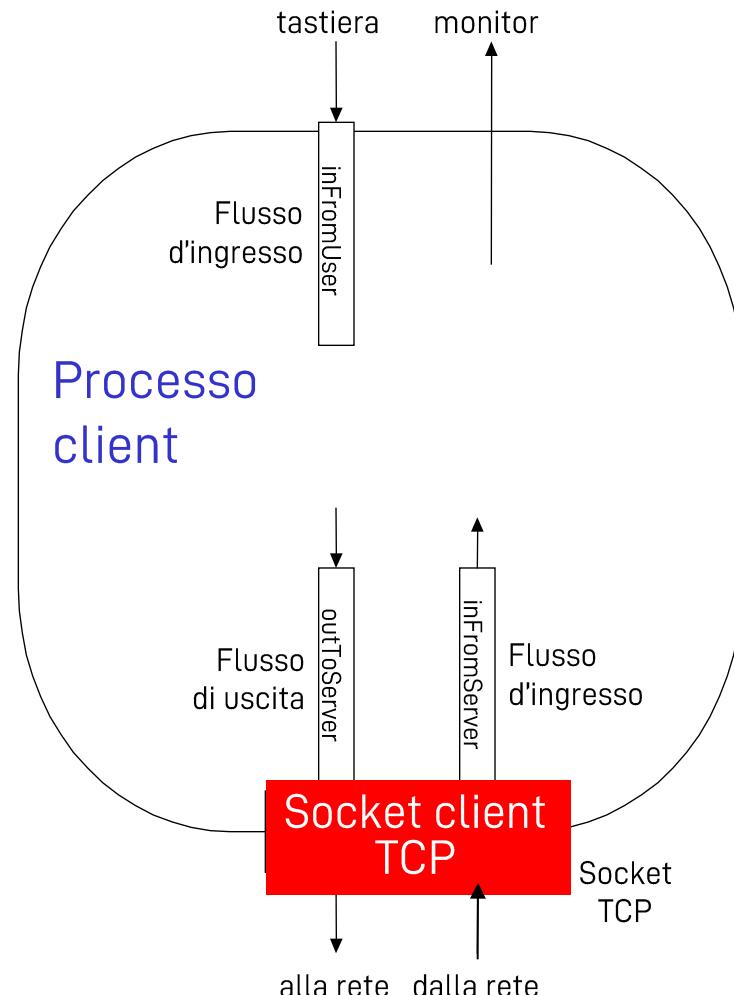
- Un **flusso** (*stream*) è una sequenza di caratteri che fluisce verso/da un processo.
- Un **flusso d'ingresso** (*input stream*) è collegato a un'origine di input per il processo, ad esempio la tastiera o la socket.
- Un **flusso di uscita** (*output stream*) è collegato a un'uscita per il processo, ad esempio il monitor o la socket.



Programmazione delle socket con TCP

Esempio di applicazione client-server:

- 1) Il client legge una riga dall'input standard (flusso **inFromUser**) e la invia al server tramite la socket (flusso **outToServer**)
- 2) Il server legge la riga dalla socket
- 3) Il server converte la riga in lettere maiuscole e la invia al client
- 4) Il client legge nella sua socket la riga modificata e la visualizza (flusso **inFromServer**)



Programmazione delle socket *con UDP*

UDP: non c'è "connessione" tra client e server

- Non c'è handshaking
- Il mittente allega esplicitamente a ogni pacchetto l'indirizzo IP e la porta di destinazione
- Il server deve estrarre l'indirizzo IP e la porta del mittente dal pacchetto ricevuto
- Detto anche datagram socket (SOCK_DGRAM)

Punto di vista dell'applicazione

UDP fornisce un trasferimento inaffidabile di gruppi di byte ("datagrammi") tra client e server

UDP: i dati trasmessi possono perdere o arrivare a destinazione in un ordine diverso da quello d'invio

Indirizzamento (i)

- Questa struct permette di manipolare in maniera semplice i Socket Address
 - ❖ “in” sta per “Internet”
 - ❖ “sin_family” è impostato sempre ad “AF_INET”

```
struct sockaddr_in {  
    short int sin_family; // Address family  
    unsigned short int sin_port; // Port number  
    struct in_addr sin_addr; // Internet address  
    unsigned char sin_zero[8];  
};
```

Indirizzamento (ii)

- Esistono diverse funzioni per manipolare gli indirizzi
 - ❖ `inet_aton("10.12.110.57", &(my_addr.sin_addr));`
 - ❖ `ina.sin_addr.s_addr = inet_addr("10.12.110.57");`
 - ❖ `ina.sin_addr.s_addr = INADDR_ANY;`

```
struct sockaddr_in my_addr;  
my_addr.sin_family = AF_INET;  
my_addr.sin_port = htons(MYPORT);  
inet_aton("10.12.110.57", &(my_addr.sin_addr));  
memset(&(my_addr.sin_zero), '\0', 8);
```

Creare un socket

- Un socket è creato utilizzando la seguente system call:
 - ❖ `int socket(int domain, int type, int protocol);`
- Dove:
 - ❖ Domain: `PF_INET`
 - ❖ Type: `SOCK_STREAM` or `SOCK_DGRAM`
 - ❖ Protocol: `0`

Associarsi ad una porta

- Se stiamo implementando un server dobbiamo attendere le nuove connessioni su una specifica porta
 - ❖ `int bind(int sockfd, struct sockaddr *my_addr, int addrlen);`
- Dove:
 - ❖ `sockfd`: file descriptor restituito dalla funzione `socket()`
 - ❖ `my_addr`: puntatore ad una struct `sockaddr` che contiene le informazioni relative al socket address (porta ed indirizzo IP)
 - ❖ `addrlen`: deve essere impostato a `sizeof(struct sockaddr)`

Ascoltare su una porta

- Dopo essersi associato su una porta, il processo server deve rimanere in attesa di nuove connessioni
 - ❖ `int listen(int sockfd, int backlog);`
- Dove:
 - ❖ `sockfd`: socket file descriptor ottenuto da `socket()`
 - ❖ `backlog`: numero massimo di connessioni che possono essere accettate

Accettare una connessione (i)

- Quando qualcuno cerca di connettersi al server sulla porta dove si sta ascoltando, la connessione viene messa in coda in attesa di essere servita
- Nel momento in cui la connessione viene accettata il sistema operativo crea un nuovo file descriptor
- Il file descriptor originale resta ancora in attesa sulla porta originale

Accettare una connessione (ii)

- Sintassi:
 - ❖ `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);`
- Dove:
 - ❖ `sockfd`: file descriptor del socket in ascolto
 - ❖ `addr`: puntatore ad una struct utilizzata per memorizzare le informazioni sulla connessione in arrivo
 - ❖ `addrlen`: da imposta a `sizeof(struct sockaddr_in)`

Trasmettere dati

- Sintassi:
 - ❖ `int send(int sockfd, const void *msg, int len, int flags);`
- Dove:
 - ❖ **sockfd**: file descriptor relativo al socket da utilizzare per la trasmissione
 - ❖ **msg**: puntatore ai dati da trasmettere
 - ❖ **len**: lunghezza della trasmissione in byte
 - ❖ **flags**: da impostare a 0
- Restituisce il numero di byte inviati

Ricevere dati

- Sintassi:
 - ❖ `int recv(int sockfd, void *buf, int len, unsigned int flags);`
- Dove:
 - ❖ **sockfd**: file descriptor del socket da utilizzare per la ricezione
 - ❖ **buf**: buffer di memoria dove scrivere i dati ricevuti
 - ❖ **len**: dimensione massima del buffer
 - ❖ **flags**: da impostare a 0
- Restituisce il numero di byte ricevuti

Esempio: Client (i)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define MAXDATASIZE 100
#define PORT 9001

int main(int argc, char *argv[])
{
    int sockfd, numbytes, len;
    char buf[MAXDATASIZE];
    char *msg;
    struct hostent *he;
    struct sockaddr_in their_addr;

    if (argc != 3) {
        fprintf(stderr,"usage: client hostname message\n");
        exit(1);
    }

    if ((he= gethostbyname(argv[1])) == NULL) {
        perror("gethostbyname");
        exit(1);
    }

    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }
```

Inclusione librerie

Definizione costanti

Dichiarazione variabili

Controllo parametri in ingresso

Traduzione hostname

Creazione socket

Esempio: Client (ii)

```
Creazione Socket Address ]→ their_addr.sin_family = AF_INET;
                           their_addr.sin_port = htons(PORT);
                           their_addr.sin_addr = *((struct in_addr *)he->h_addr);
                           memset(&(their_addr.sin_zero), '\0', 8);

Conessione al socket ]→ if (connect(sockfd, (struct sockaddr *)&their_addr,sizeof(struct
                           sockaddr)) == -1) {
                           perror("connect");
                           exit(1);
}

Trasmissione messaggio ]→ msg = argv[2];
                           len = strlen(msg);

                           if (send(sockfd, msg, len, 0) == -1) {
                           perror("send");
                           exit(1);
}

Ricezione risposta ]→ if ((numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
                           perror("recv");
                           exit(1);
}

Stampa la risposta
E chiude il socket ]→ buf[numbytes] = '\0';
                           printf("Received: %s\n",buf);
                           close(sockfd);
                           return 0;
}
```

Esempio: Server (i)

Inclusione librerie

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
#include <time.h>

#define MAXDATASIZE 100
#define PORT 9001

void sigchld_handler(int s)
{
    while(waitpid(-1, NULL, WNOHANG) > 0);
}

int main()
{
    int sockfd, new_fd, numbytes, len;
    char buf[MAXDATASIZE];
    char *msg;
    struct sockaddr_in my_addr;
    struct sockaddr_in their_addr;
    int sin_size;
    struct sigaction sa;
    int yes=1;

    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }
```

Definizione costanti

Dichiarazione variabili

Creazione socket

Esempio: Server (ii)

```
if (setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1) {
    perror("setsockopt");
    exit(1);
}

my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(PORT);
my_addr.sin_addr.s_addr = INADDR_ANY;
memset(&(my_addr.sin_zero), '\0', 8);

if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) ==
    -1) {
    perror("bind");
    exit(1);
}

if (listen(sockfd, 10) == -1) {
    perror("listen");
    exit(1);
}

sa.sa_handler = sigchld_handler;
sigemptyset(&sa.sa_mask);
sa.sa_flags = SA_RESTART;
if (sigaction(SIGCHLD, &sa, NULL) == -1) {
    perror("sigaction");
    exit(1);
}

while(1) {
    sin_size = sizeof(struct sockaddr_in);
    if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr,
        &sin_size)) == -1) {
        perror("accept");
        exit(1);
    }
}
```

Creazione Socket Address →

Associazione alla porta →

Ascolto sulla porta →

Gestione errori →

Attesa nuove condizioni →

Esempio: Server (iii)

Ricezione messaggio

```
printf("Server: got connection from %s\n",inet_ntoa(their_addr.sin_addr));  
  
if (!fork()) {  
    if ((numbytes=recv(new_fd,buf,MAXDATASIZE-1, 0)) == -1) {  
        perror("recv");  
        exit(1);  
    }  
    buf[numbytes] = '\0';  
    printf("Received: %s\n",buf);  
  
    if (strcmp(buf, "help") == 0) {  
        msg = "Commands supported: time help.";  
    } else if (strcmp(buf, "time") == 0) {  
        time_t t = time(NULL);  
        struct tm* tp = localtime(&t);  
        char datetime[40];  
        sprintf(datetime,"%d-%02d-%02d %02d:%02d:%02d",  
               tp->tm_year + 1900,  
               tp->tm_mon + 1,  
               tp->tm_mday,  
               tp->tm_hour,  
               tp->tm_min,  
               tp->tm_sec);  
        msg = datetime;  
    } else {  
        msg = "Command not recognized.";  
    }  
    len = strlen(msg);  
    if (send(new_fd, msg, len, 0) == -1)  
        perror("send");  
    close(new_fd);  
    exit(0);  
} // end if  
close(new_fd);  
} // end while  
return 0;  
} // end main
```

Gestione richiesta

Invio risposta
E chiusura socket

Capitolo 2: Riassunto

- ❑ Architetture delle applicazioni
 - ❖ Client-server
 - ❖ P2P
 - ❖ Ibride
- ❑ Requisiti dei servizi delle applicazioni:
 - ❖ Affidabilità, ampiezza di banda, ritardo
- ❑ Modello di servizio di trasporto di Internet
 - ❖ Orientato alle connessioni, affidabile: TCP
 - ❖ Inaffidabile, datagrammi: UDP
- ❑ Protocolli specifici:
 - ❖ HTTP
 - ❖ FTP
 - ❖ SMTP, POP, IMAP
 - ❖ DNS
 - ❖ P2P: BitTorrent, Skype
- ❑ Programmazione delle socket (cenni)

Riassunto

Molto importante: conoscere i *protocoli*

- Tipico scambio di messaggi di richiesta/risposta:
 - ❖ Il client richiede informazioni o servizi
 - ❖ Il server risponde con dati e codici di stato
- Formati dei messaggi:
 - ❖ Intestazioni: campi che forniscono informazioni sui dati
 - ❖ Dati: informazioni da comunicare
- Controllo o messaggi di dati
 - ❖ In banda, fuori banda
- Architettura centralizzata o decentralizzata
- Protocollo senza stato o con stato
- Trasferimento di messaggi affidabile o inaffidabile
- Concetto di *complessità nelle parti periferiche della rete*