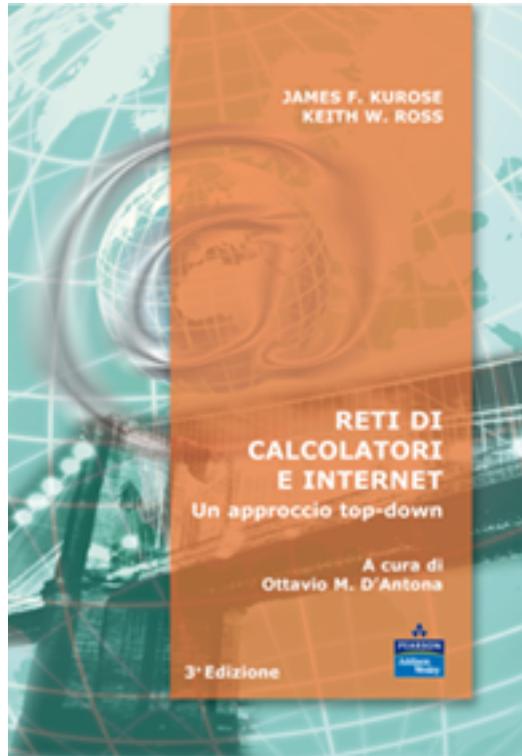


Reti di calcolatori: Livello Applicazione

(Capitolo 2 Kurose-Ross)

Marco Roccetti
1 Marzo 2023

(Capitolo 2 Kurose-Ross)



*Reti di calcolatori e Internet:
Un approccio top-down*

3^a edizione
Jim Kurose, Keith Ross
Pearson Education Italia

©2005

Capitolo 2: Livello di 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
 - ↳ Lega IP e nome del server (poco protetto)
 - ↳ Trasferimento dei dati via posta elettronica
 - ↳ Permette agli utent. di connettersi a un server senza conoscere il suo indirizzo IP
- 2.6 Condivisione di file P2P
- 2.7 Programmazione delle socket con TCP
- 2.8 Programmazione delle socket con UDP
- 2.9 Costruire un semplice server web

Capitolo 2: Livello di 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



*Sbordighe
de la Unreal Engine*

- ❑ Apprendere informazioni sui protocolli esaminando quelli delle più diffuse applicazioni di rete
 - ❖ **HTTP** Protocollo
 - ❖ FTP
 - ❖ SMTP / POP3 / IMAP
 - ❖ DNS
- ❑ Programmare le applicazioni di rete
 - ❖ socket API

Alcune diffuse applicazioni di rete

- Posta elettronica
- Web
- Messaggistica istantanea
- Autenticazione in un calcolatore remoto (Telnet e SSH)
- Condivisione di file P2P
- Giochi multiutente via rete
- Streaming di video-clip memorizzati
- Telefonia via Internet
- Videoconferenza in tempo reale

Quando HTTP era stato inventato non c'erano misure di sicurezza

La sicurezza viene implementata a livello applicazione

I protocolli crittografici devono essere implementati dallo sviluppatore

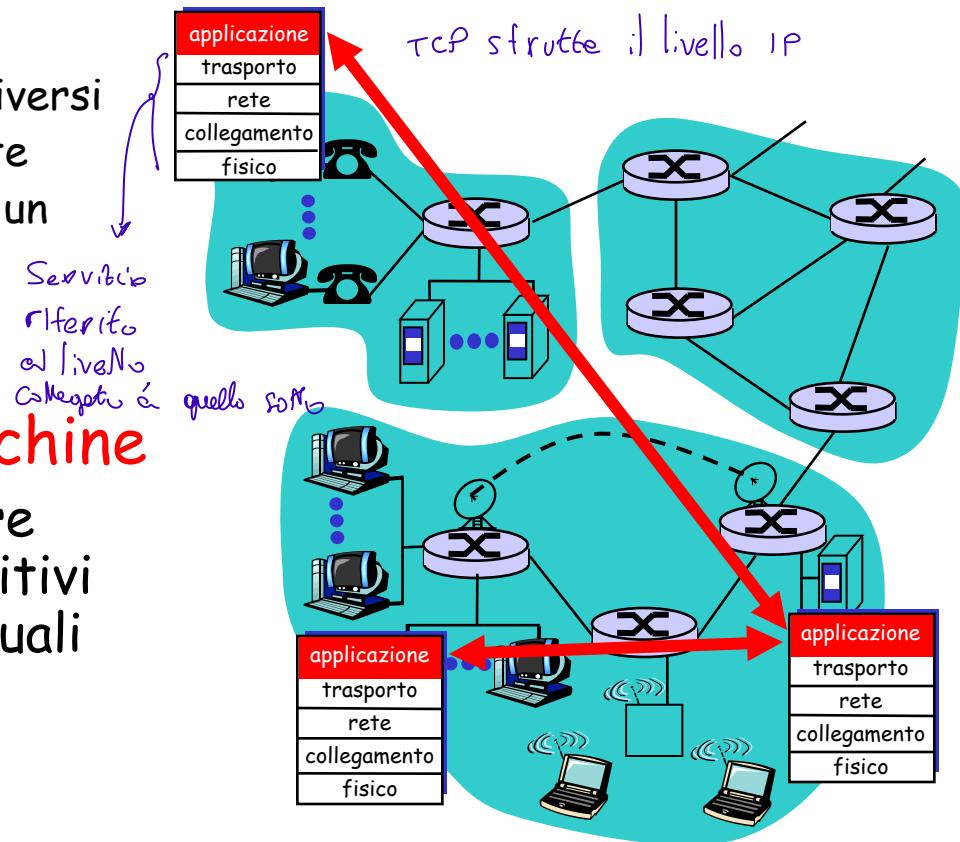
Creare un'applicazione di rete

Scrivere programmi che

- ❖ girano su sistemi terminali diversi
- ❖ comunicano attraverso la rete
- ❖ Ad es. il Web: 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 di 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 Programmazione delle socket con TCP
- 2.8 Programmazione delle socket con UDP
- 2.9 Costruire un semplice server web

Architetture delle applicazioni di rete

Client-server]



Principio architettonico
di tutte le applicazioni di rete

Tutte le applicazioni
usano la rete

Peer-to-peer (P2P)

Architetture ibride (client-server e P2P)

Il server risponde alle richieste del client, e quindi deve essere sempre acceso

→ NAPSTER (P2P) indicizzazione di copie di canzoni → denunciato

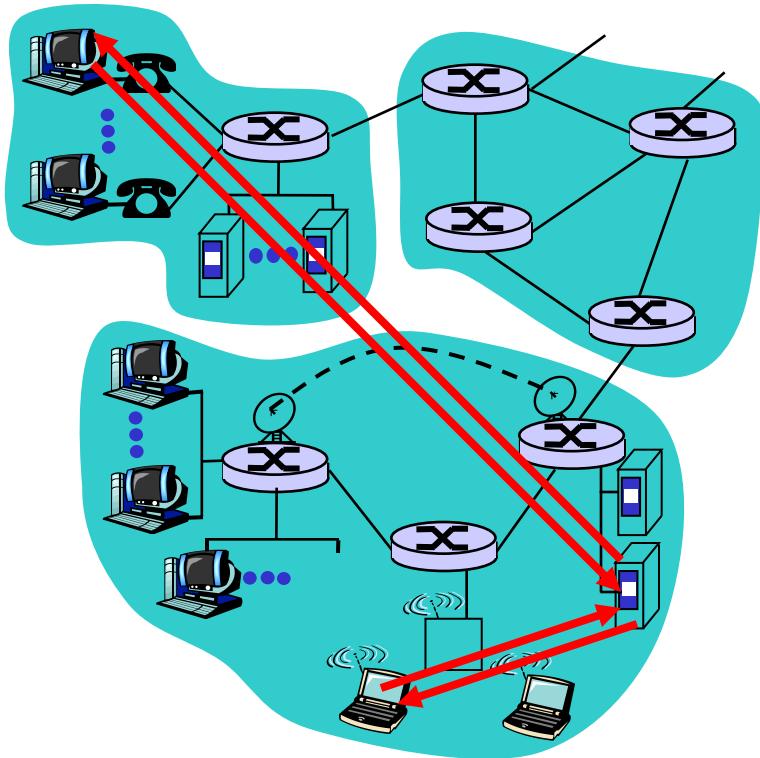
cliente → ^{server} NAPSTER → computer che metteva
la canzone a disposizione

Non fa ricerche, ma è un
database di link

Princípio di neutralità del principio elettronico → non è colpa di chi inventa lo strumento
elettronico, ma di chi lo usa

Napster ha messo su un sistema che infrange copyright, quindi è diventato un servizio che offre un
algoritmo di ricerca (metodo) per trovare canzoni già su internet

Architettura client-server



server:

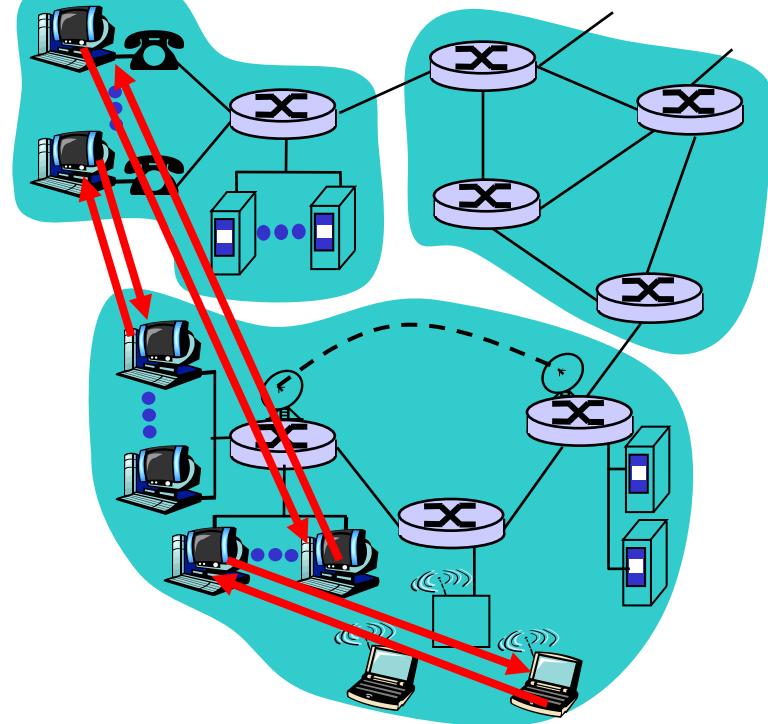
- ❖ host **sempre attivo** (always on)
- ❖ indirizzo IP **fisso**
- ❖ server farm per creare un potente server virtuale

client:

- ❖ comunica con il server
- ❖ può contattare il server in qualunque momento
- ❖ può avere indirizzi IP dinamici
- ❖ non comunica direttamente con gli 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
- Un esempio: Gnutella
 - Facilmente scalabile
 - Difficile da gestire



Ibridi (client-server e P2P)

Napster

- ❖ Scambio di file secondo la logica P2P
- ❖ Ricerca di file centralizzata:
 - i peer registrano il loro contenuto presso un server centrale
 - i peer chiedono allo stesso server centrale di localizzare il contenuto

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

Processi comunicanti

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

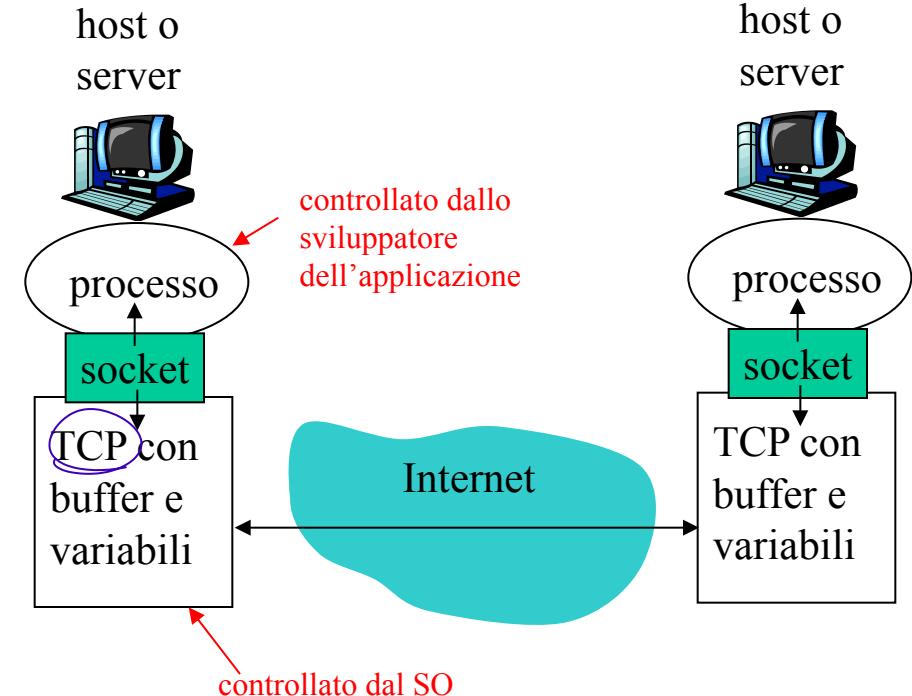
Processo server : processo che attende di essere contattato

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

Socket

→ Punto di ingresso e uscita che invia
o riceve messaggi

- 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)

Processi di 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 A ha un **indirizzo IP univoco a 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
- Approfondiremo questi argomenti più avanti

Nel P2P si usano numeri di porta elevati

Un IP eseguisce tanti processi quanti sono le sue porte

Protocollo a livello di applicazione

IP: Ponte → Processo unico

- 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
- Consente l'interoperabilità
- Ad esempio, HTTP, SMTP

Protocolli proprietari:

- Ad esempio, KaZaA

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%**

Temporizzazione

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

Internet non garantisce la temporizzazione

$$\hookrightarrow + \text{Banda} = -\text{congestione} \approx \text{temporizzazione}$$

Aampiezza di banda

- alcune applicazioni (ad esempio, quelle multimediali) per essere "efficaci" richiedono un'ampiezza di banda minima
- altre applicazioni ("le applicazioni elastiche") utilizzano l'ampiezza di banda che si rende disponibile

Io voglio che un pacchetto sia consegnato entro un tempo limite

Requisiti del servizio di trasporto di alcune applicazioni comuni

Applicazione	Tolleranza alla perdita di dati	Aampiezza di banda	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 Kbps a 1 Mbps Video: da 10 Kbps a 5 Mbps	Sì, centinaia di ms
Audio/video memorizzati	Sì	Come sopra	Sì, pochi secondi
Giochi interattivi	Sì	Fino a pochi Kbps	Sì, centinaia di ms
Messaggistica istantanea	No	Variabile	Sì e no

Servizi dei protocolli di trasporto Internet

Servizio di 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 la rete è sovraccaricata
- non offre:** temporizzazione, ampiezza di banda minima

ci s: può permettere di perdere pacchetti ?

Real time

No

? si

+ Bande

Servizio di 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

meno probabilità di perdere dati

D: Perché preoccuparsi?
Perché esiste UDP?

Algoritmi usati da TCP per non congestionare la rete e il mittente

Applicazioni Internet: protocollo a livello applicazione e protocollo di trasporto

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	Proprietario (ad esempio, RealNetworks)	TCP o UDP
Telefonia Internet	Proprietario (ad esempio, Vonage, Dialpad)	Tipicamente UDP

Capitolo 2: Livello di 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 Condivisione di file P2P
- 2.7 Programmazione delle socket con TCP
- 2.8 Programmazione delle socket con UDP
- 2.9 Costruire un semplice server web

Web e HTTP

Trasferire dal web al client un oggetto

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 HTML** che include diversi oggetti referenziati
- Ogni oggetto è referenziato da un **URL** Universal resource locator
- Esempio di URL:

www.someschool.edu/someDept/pic.gif

nome dell'host

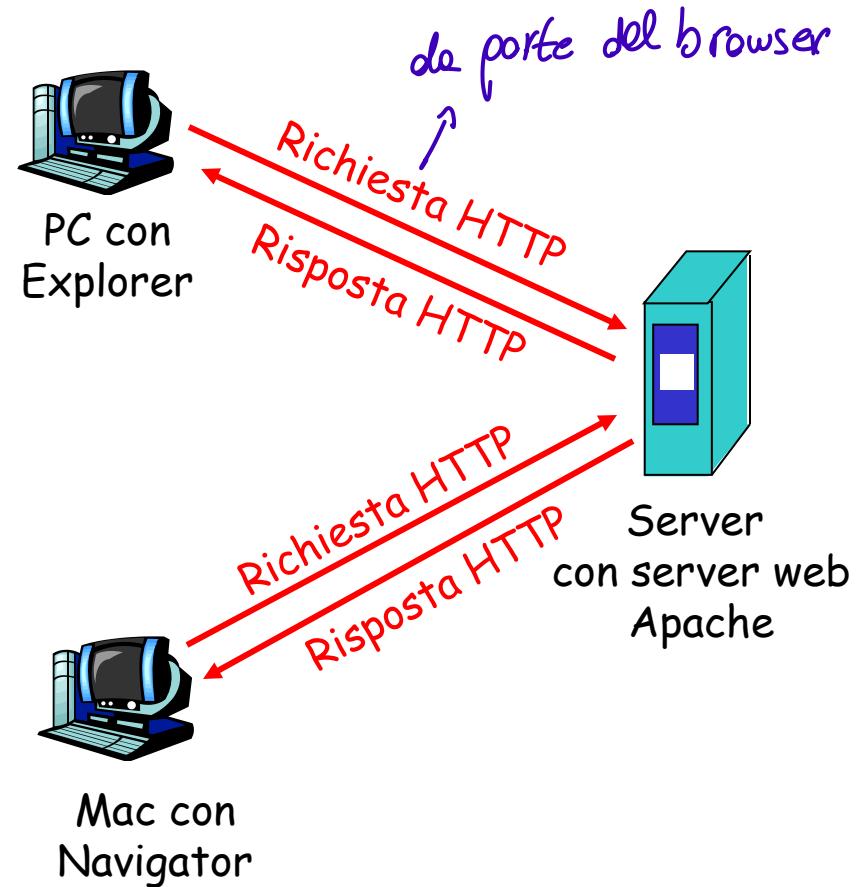
nome del percorso

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
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068

→ Servizi offerti completamente diversi



Panoramica su HTTP (continua)

Usa TCP:

La rete funziona
Meglio se non deve
memorizzare

- 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 (e anche quella HTTP)

HTTP è un protocollo
"senza stato" (stateless)

- Il server non mantiene informazioni sulle richieste fatte dal client

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

nota

Connessioni HTTP

Connessioni non persistenti

- Ogni oggetto viene trasmesso su una connessione TCP
- HTTP/1.0 usa connessioni non persistenti

No stato = comunicazione efficiente

Connessioni persistenti

- Più oggetti possono essere trasmessi su una singola connessione TCP tra client e server
- HTTP/1.1 usa connessioni persistenti nella modalità di default

Connessioni non persistenti

Supponiamo che l'utente immetta l'URL

www.someSchool.edu/someDepartment/home.index

(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

tempo
↓

Connessioni non persistenti (continua)

tempo
↓

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

Formato predefinito per il protocollo HTTP

Schema del tempo di risposta

me mi interessa:

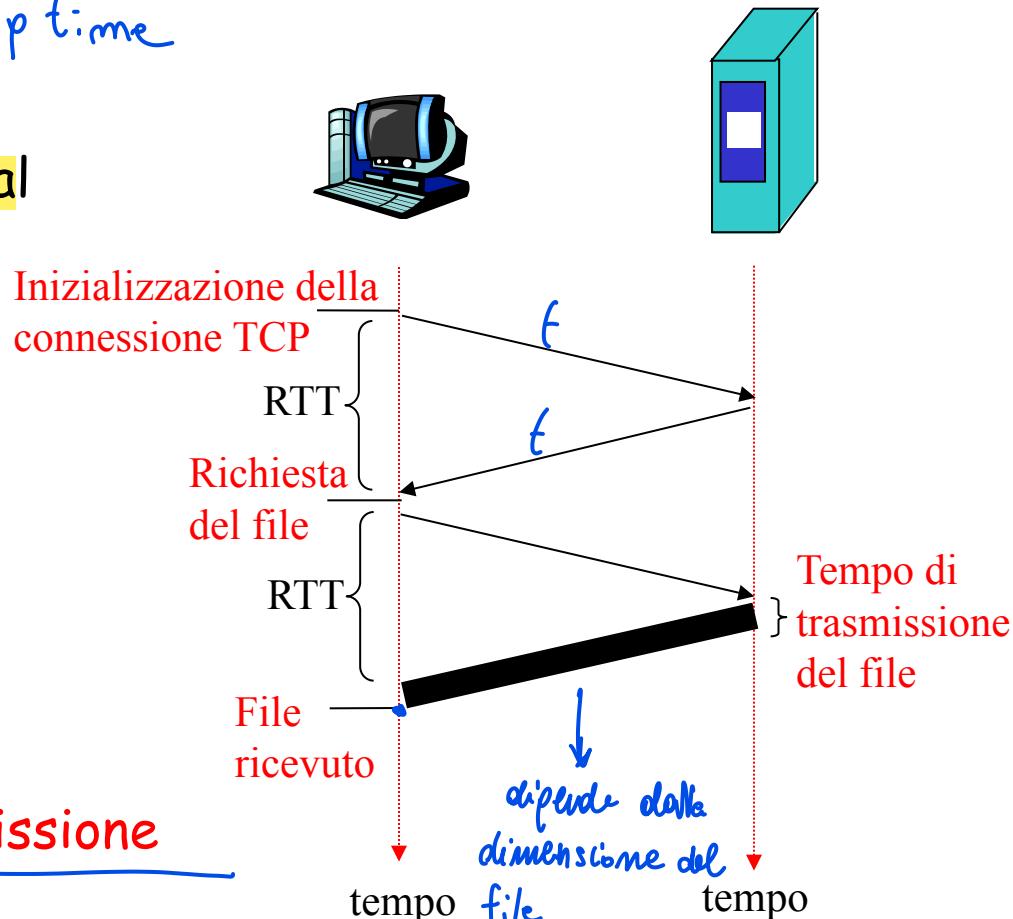
tempo di trasmissione \rightarrow round trip time
Definizione di RTT: tempo

impiegato da un pacchetto
(piccolo) 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

VARIA RTT

Svantaggi delle connessioni **non** persistenti:

- richiede 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 ^{HTTP 1.1} connessione **TCP aperta** dopo l'invio di una risposta
- i successivi messaggi tra gli stessi client/server vengono trasmessi sulla connessione aperta

Connessione persistente senza pipelining:

- il client invia una **nuova richiesta solo quando ha ricevuto la risposta precedente**
- un RTT per ogni oggetto referenziato

Connessione persistente con pipelining:

- è la modalità di default in **HTTP/1.1** *per solo x aprire*
- il client invia le richieste non appena incontra un oggetto referenziato
- un solo RTT per tutti gli oggetti referenziati

Pipeline: il client può spedire pacchetti persistenti

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)

GET /somedir/page.html HTTP/1.1

risultato

Host: www.someschool.edu

User-agent: Mozilla/4.0

Connection: close

Accept-language: fr

vedere se un
server è online

Righe di
intestazione

Tra i righe
completate

Un carriage return

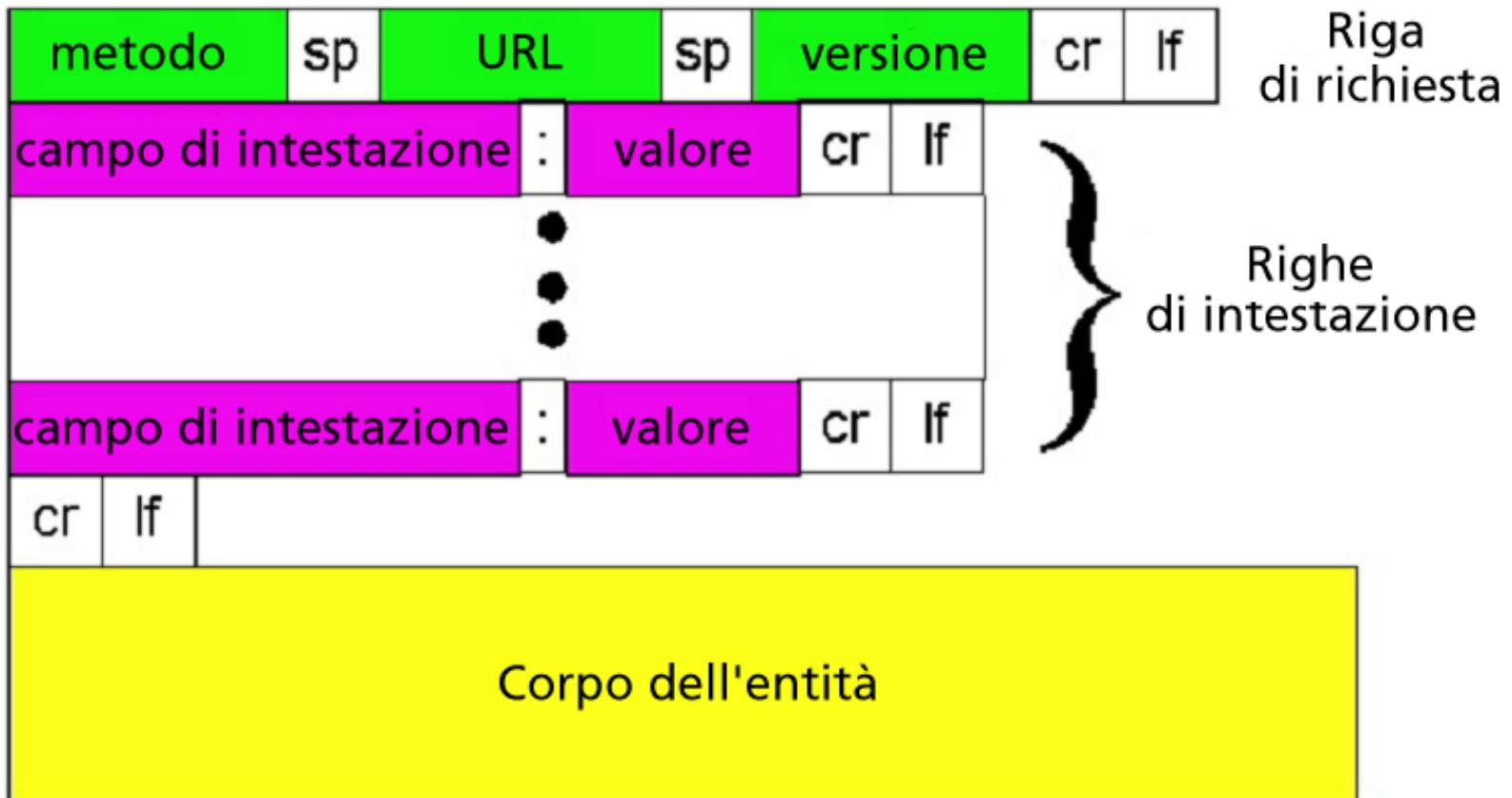
e un line feed

indicano la fine
del messaggio

(carriage return e line feed extra)

TUTTI i messaggi
viaggiano in chiaro
sono leggibili dall'uomo
(ASCII)

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

dare possibilità al client di scrivere qualcosa

10-15 anni

pagina da ricuperare che risiede su un server

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 *scrivere sulle form*
- HEAD *Verificare che il server sia online*
 - ❖ chiede al server di escludere l'oggetto richiesto dalla risposta, utile per il debugging

HTTP/1.1

- GET, POST, HEAD
 - ❖ *Scorico info*
- PUT *scrive un intero file*
 - ❖ include il file nel corpo dell'entità e lo invia al percorso specificato nel campo URL
- DELETE
 - ❖ cancella il file specificato nel campo URL

con queste 5 operazioni, gestisce un intero database

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

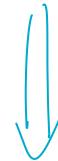
- ❖ Il messaggio di richiesta non è stato compreso dal server

404 Not Found

- ❖ Il documento richiesto non si trova su questo server

505 HTTP Version Not Supported

- ❖ Il server non ha la versione di protocollo HTTP



Nel RFC

Ci sono le possibili risposte

Provatate HTTP (lato client)

1. Collegatevi via Telnet al vostro server web preferito:

telnet cis.poly.edu 80

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

2. Digitate una richiesta GET:

**GET /~ross/ HTTP/1.1
Host: cis.poly.edu**

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

3. Guardate il messaggio di risposta trasmesso dal server HTTP!

Osserviamo HTTP in azione

- Esempio Telnet
- Esempio Ethereal

Interazione utente-server: i cookie

Molti dei più importanti siti web usano i cookie

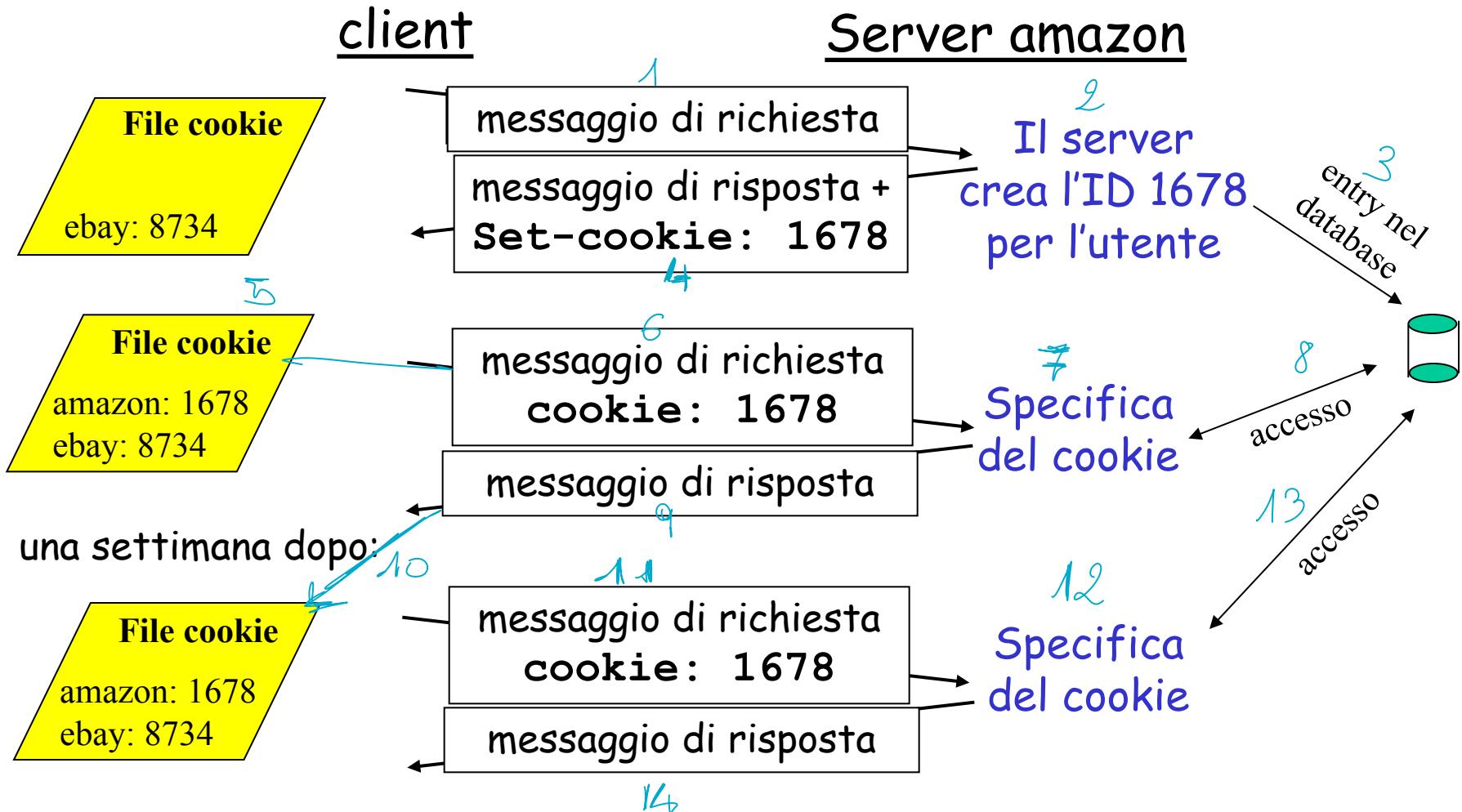
Quattro componenti:

- 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)



Cookie (continua)

nota

Cosa possono contenere i cookie:

- autorizzazione
- carta per acquisti
- raccomandazioni
- stato della sessione dell'utente (e-mail)

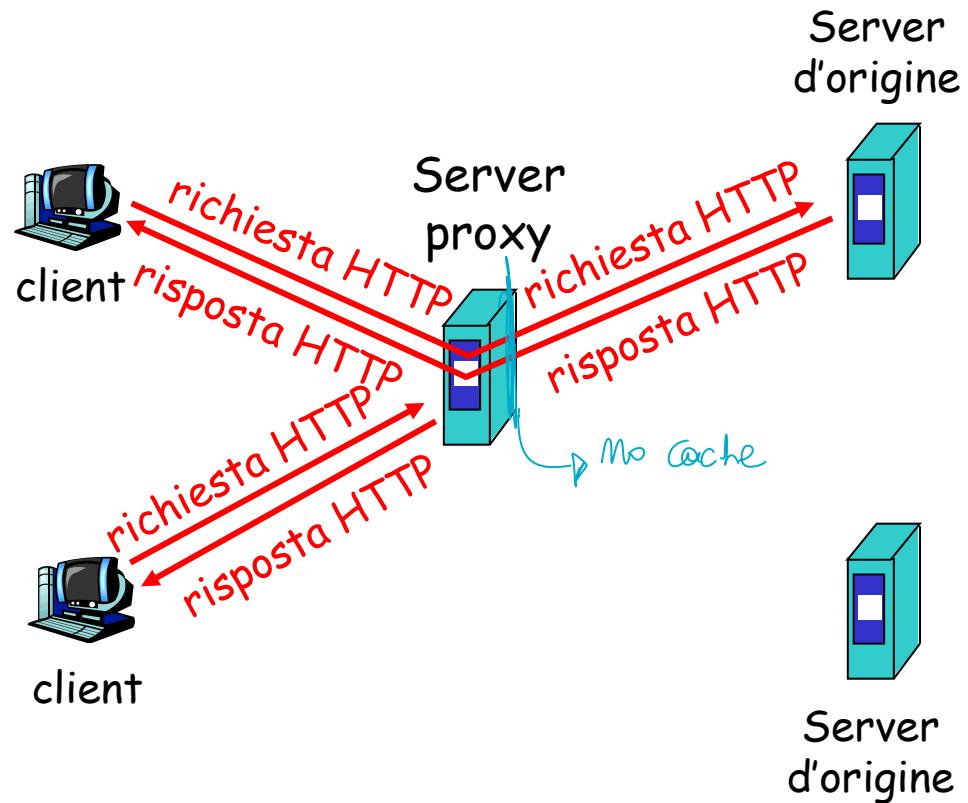
Cookie e privacy:

- i cookie permettono ai siti di imparare molte cose sugli utenti
- l'utente può fornire al sito il nome e l'indirizzo e-mail
- i motori di ricerca usano il reindirizzamento e i cookie per sapere ancora di più
- le agenzie pubblicitarie ottengono informazioni dai siti (attraverso gestione banner)
- Uso controverso

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 la condivisione di file P2P)

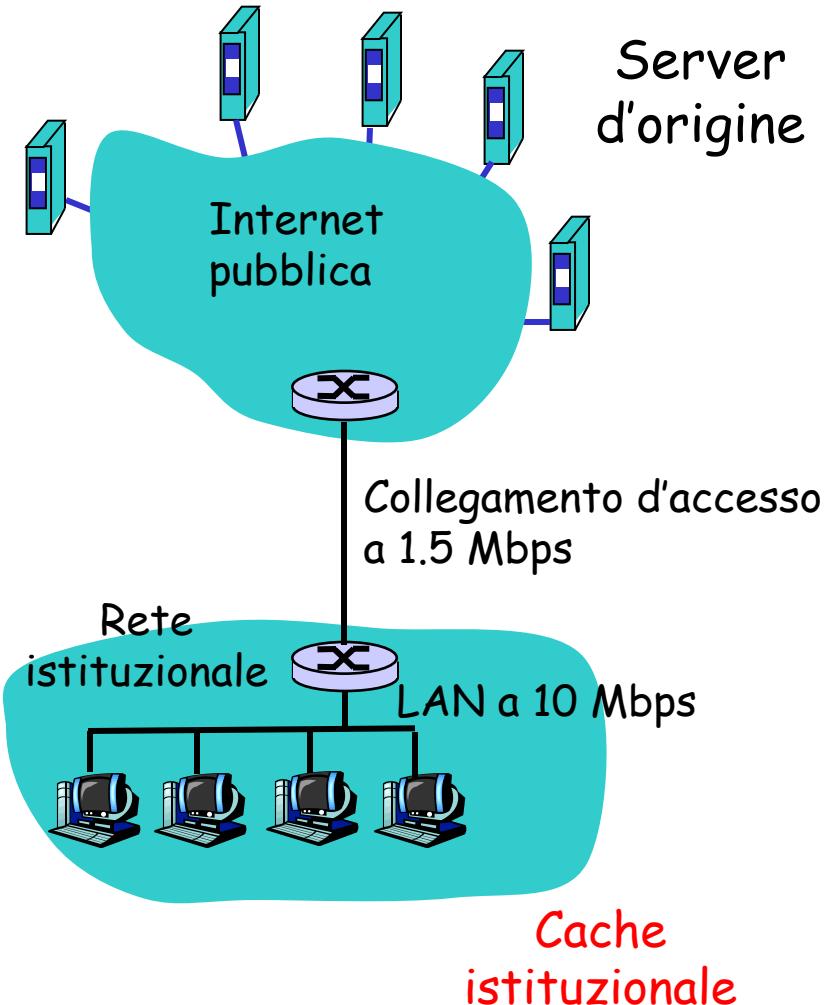
Esempio di caching

Ipotesi

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

Conseguenze

- utilizzazione sulla LAN = $(15 \text{ rps} \times 100 \text{ Kbit/r}) / 10 \text{ Mbps} = 0.15 = 15\%$
- utilizzazione sul collegamento d'accesso = 100%
- ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN
= 2 sec + minuti + millisecondi



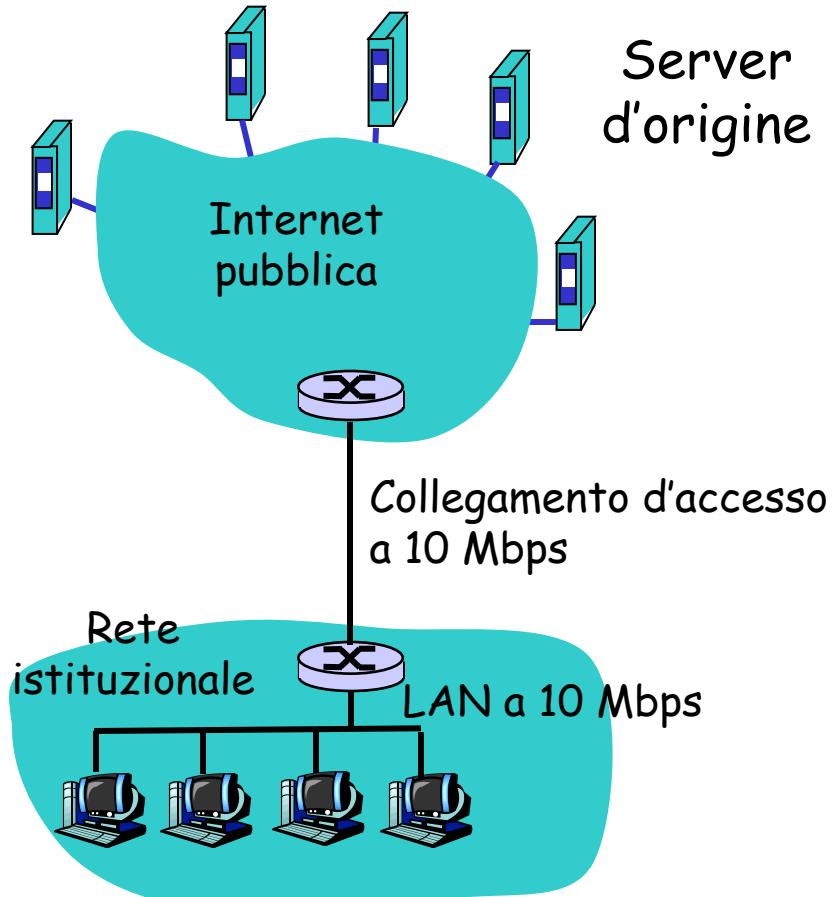
Esempio di caching (continua)

Soluzione possibile

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

Conseguenze

- utilizzazione sulla LAN = 15%
- utilizzazione sul collegamento d'accesso = 15%
- ritardo totale = ritardo di Internet + ritardo di accesso + ritardo della LAN
= 2 sec + msec + msec
- l'aggiornamento spesso è molto costoso



Esempio di caching (continua)

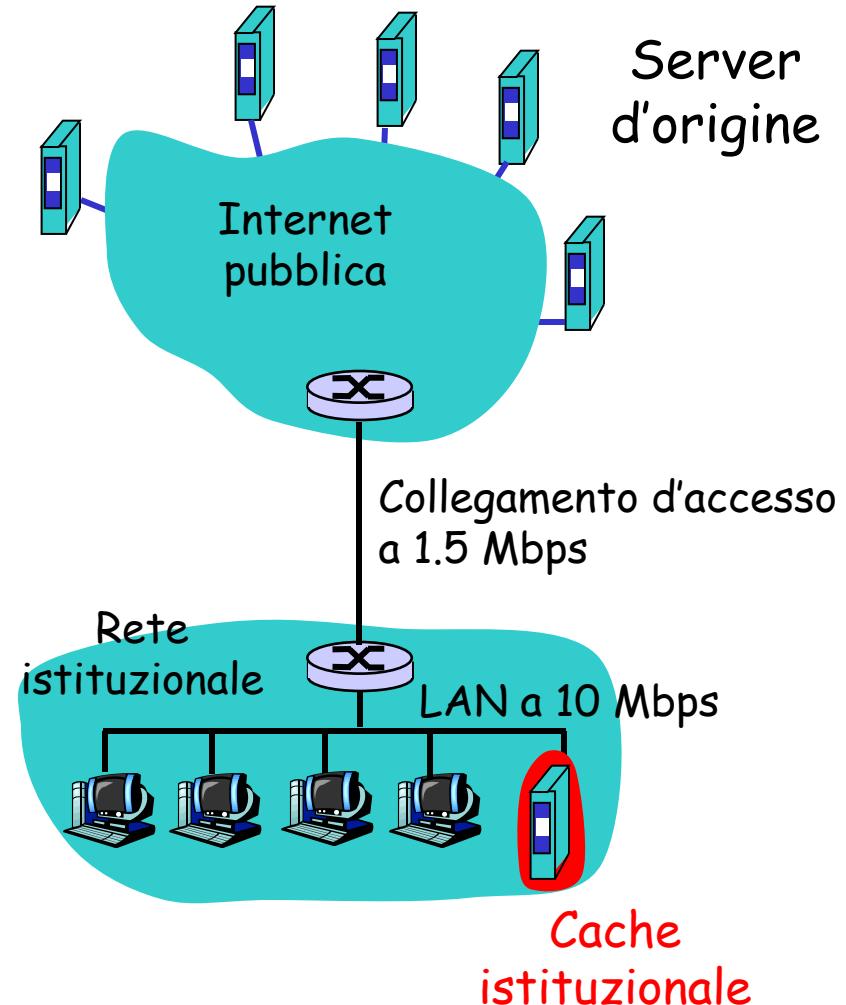
Installare la cache

- supponiamo una percentuale di successo (*hit rate*) pari a 0,4

Conseguenze

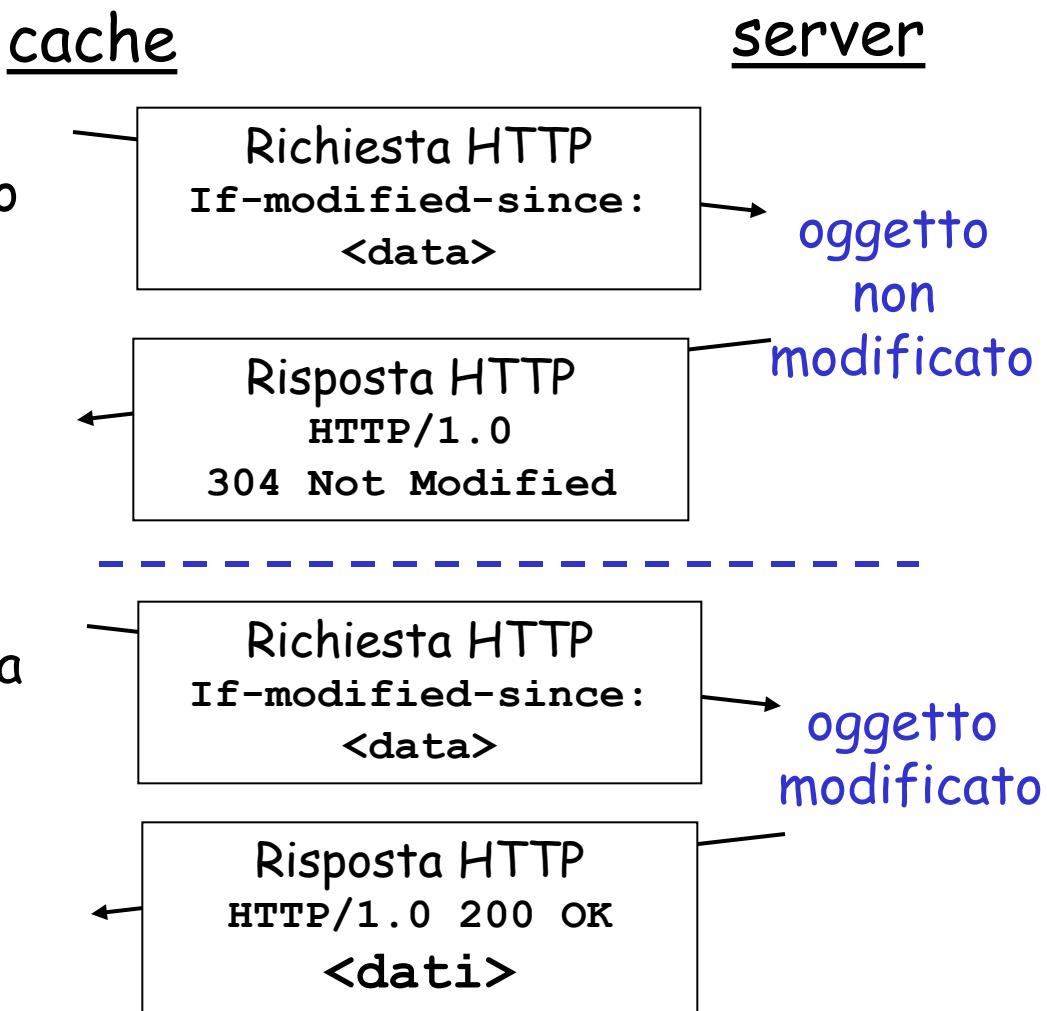
- il 40% delle richieste sarà soddisfatto quasi immediatamente
- il 60% delle richieste sarà soddisfatto dal server d'origine
- l'utilizzazione del collegamento d'accesso si è ridotta al 60%, determinando ritardi trascurabili (circa 10 msec)
- ritardo totale medio = ritardo di Internet + ritardo di accesso + ritardo della LAN =

$$0,6 * (2,01) \text{ sec} + 0,4 * (0,001) \text{ sec} < 1,4 \text{ sec}$$



GET condizionale (oggetti in cache scaduti)

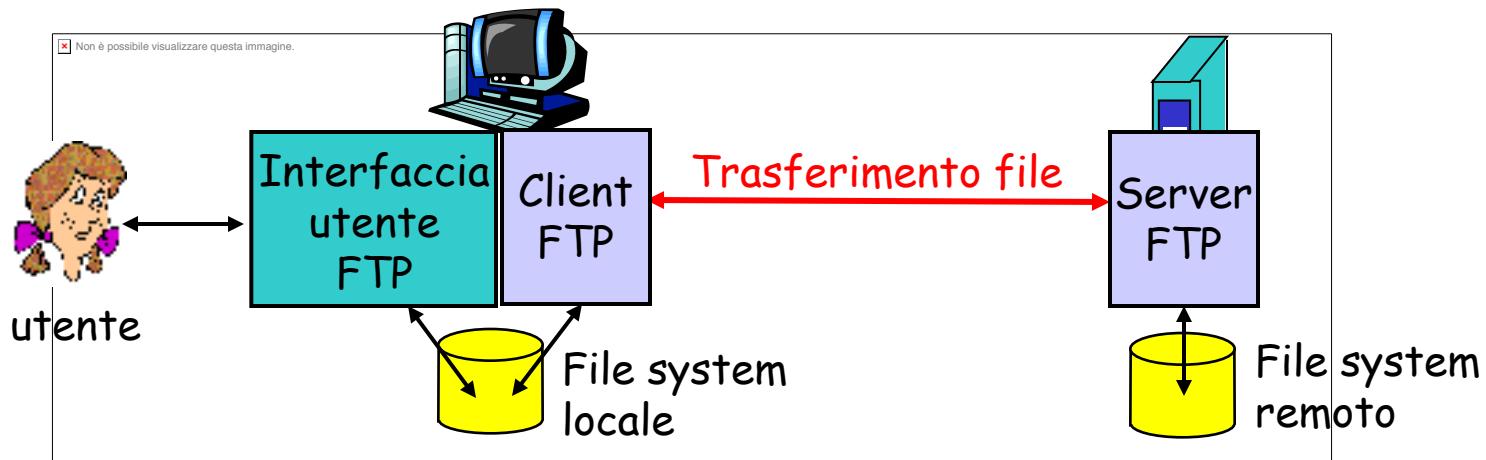
- **Obiettivo:** non inviare un oggetto se la cache ha una copia aggiornata dell'oggetto
- **cache:** specifica la data della copia dell'oggetto nella richiesta HTTP
If-modified-since:
 <data>
- **server:** la risposta non contiene l'oggetto se la copia nella cache è aggiornata:
HTTP/1.0 304 Not Modified



Capitolo 2: Livello di 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 Programmazione delle socket con TCP
- 2.8 Programmazione delle socket con UDP
- 2.9 Costruire un semplice server web

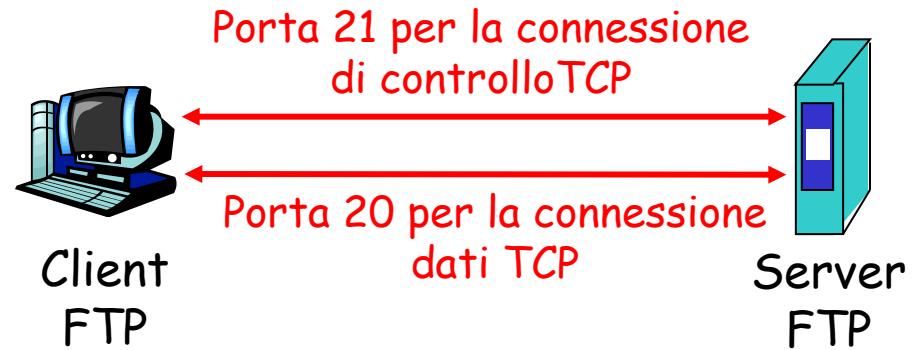
FTP: file transfer protocol



- 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: RFC 959
- server ftp: porta 21

FTP: connessione di controllo, connessione dati, connessioni parallele

- 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*, HTTP fa tutto in-band invece
- Il server FTP mantiene lo "stato": directory corrente, autenticazione precedente, meno performante di HTTP

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 di 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 Programmazione delle socket con TCP
- 2.8 Programmazione delle socket con UDP
- 2.9 Costruire un semplice server web

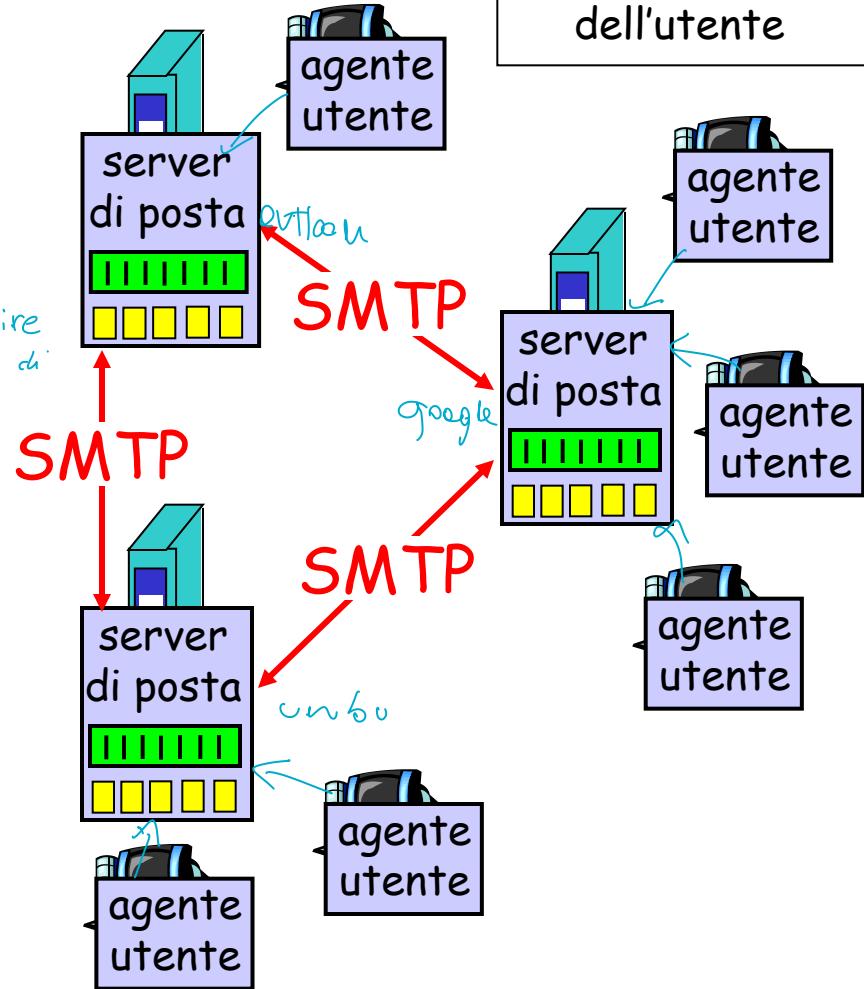
Posta elettronica (asincrona)

Tre componenti principali:

- agente utente
- server di posta (gmail, unibo, ...)
- simple mail transfer protocol: SMTP

Agente utente
Programma di Posta elettronica

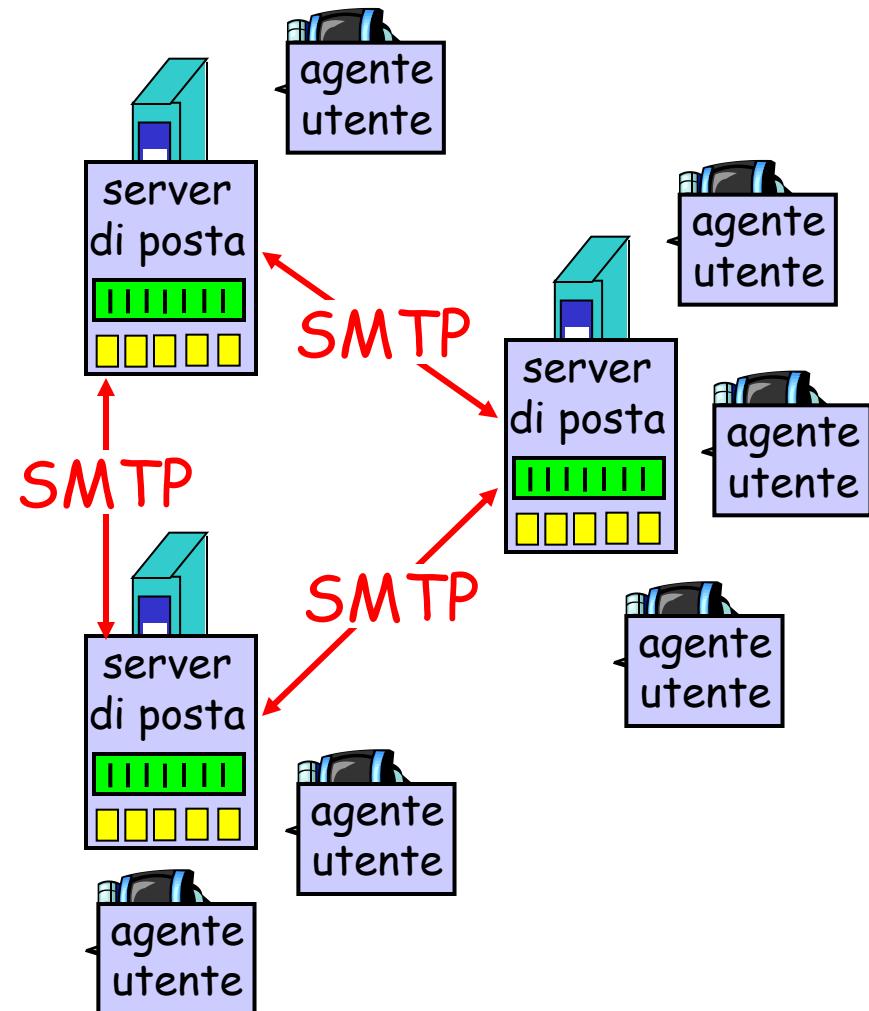
- detto anche "mail reader"
- composizione, editing, lettura dei messaggi di posta elettronica
- esempi: Eudora, Outlook, elm, pine, Netscape Messenger
- i messaggi in uscita o in arrivo sono memorizzati sul server



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, agisce sia da
 - ❖ client: server di posta
trasmittente (mittente),
che da
 - ❖ "server": server di posta
ricevente

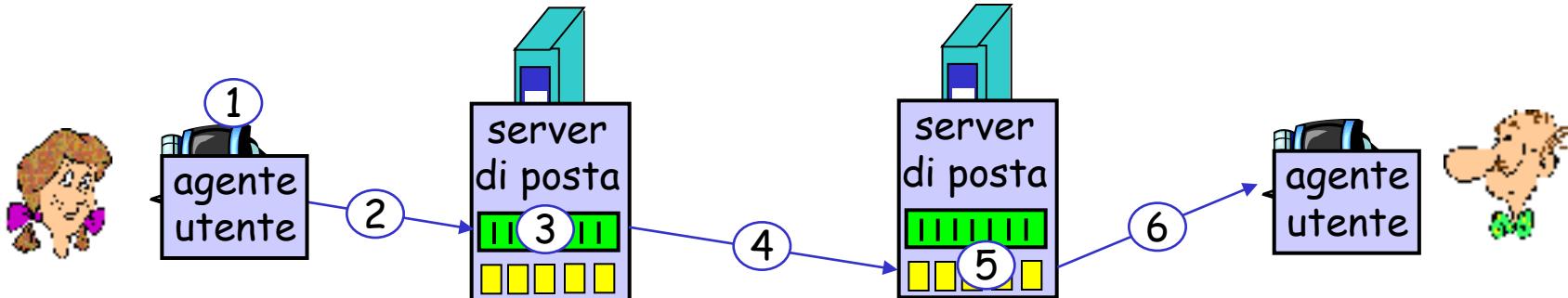


Posta elettronica: SMTP [RFC 2821]

- usa TCP per trasferire in modo affidabile i messaggi di posta elettronica dal client al server, porta 25
- trasferimento diretto: il server trasmittente al server ricevente (se fallisce riprova dopo 30 min, per giorni e se non ce la fa notifica l'utente dell'insuccesso)
- tre espressioni 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 (limitazione arcaica, costringe a ricodificare in ASCII attachment multimediali, HTTP è più furbo)

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



Esempio di interazione (handshaking) SMTP (comandi in maiuscolo)

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

Provate un'interazione SMTP:

- telnet servername 25**
- Riceverete la risposta 220 dal server
- Digitate i comandi HELO, MAIL FROM, RCPT TO, DATA, QUIT

Questo vi consente di inviare messaggi di posta elettronica senza usare il client di posta (lettore)

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 CRLF.CRLF per determinare la fine del messaggio

Confronto con HTTP:

- HTTP: pull (da server a client, server passivo)
- SMTP: push (tra server, ruolo attivo)
- Entrambi hanno un'interazione comando/risposta in ASCII, codici di stato
- HTTP: ogni oggetto è encapsulato 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 (tra server)

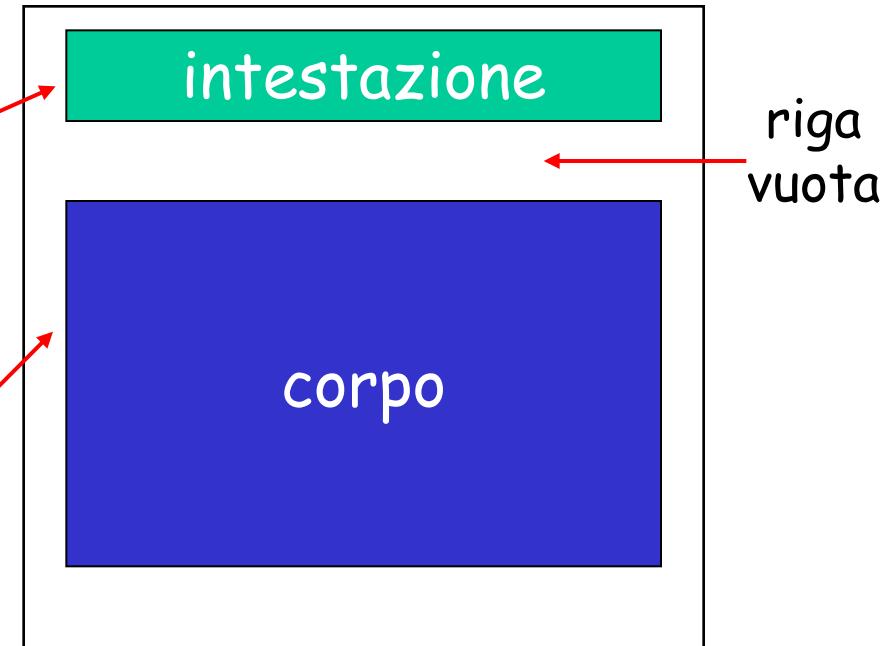
RFC 822: standard per il formato dei messaggi di testo:

- Righe di intestazione, per esempio

- ❖ To:
 - ❖ From:
 - ❖ Subject:

differenti dai comandi SMTP!

- corpo
 - ❖ il "messaggio", soltanto caratteri ASCII



Formato del messaggio: estensioni di messaggi multimediali

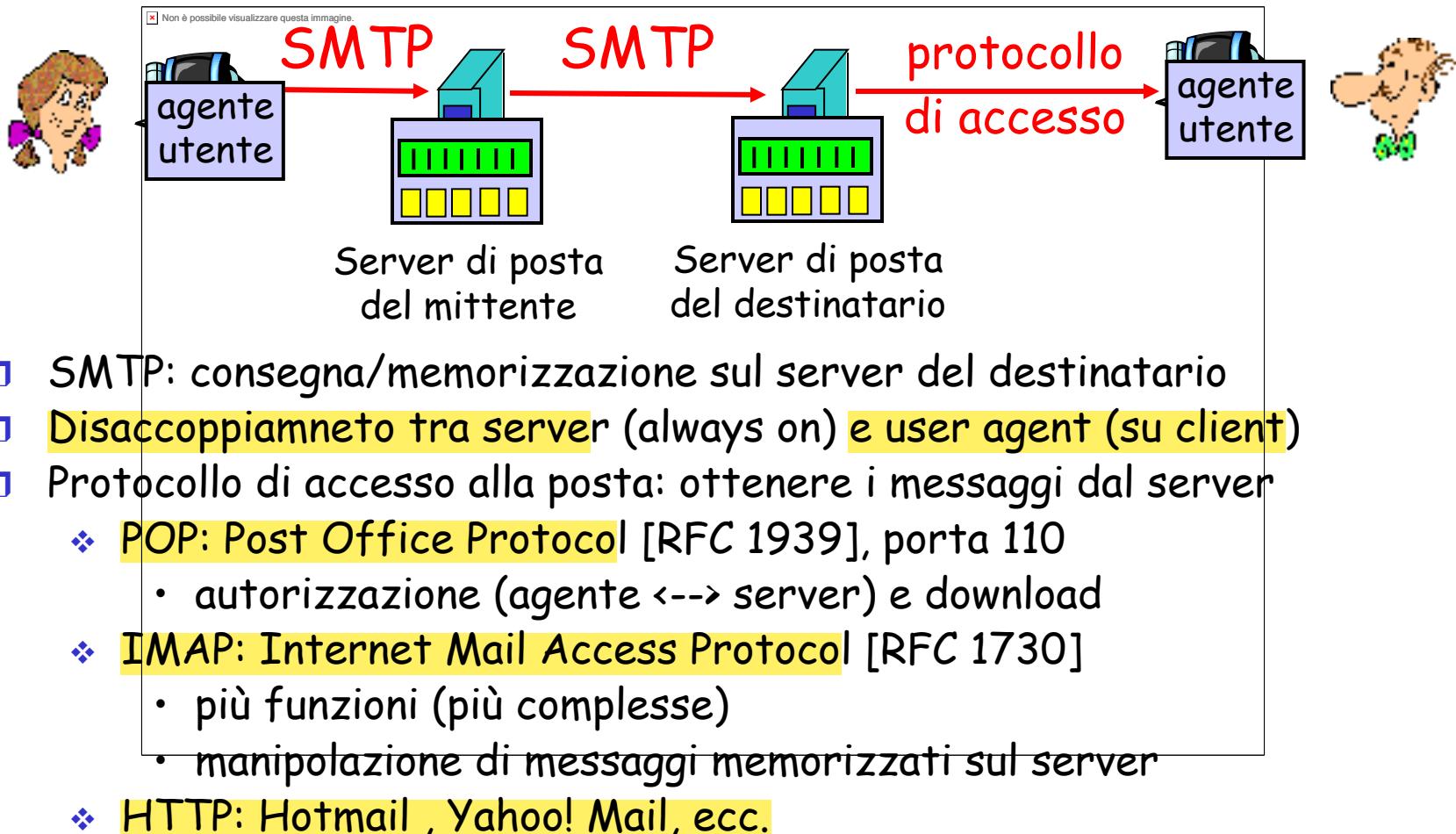
- ❑ MIME: estensioni di messaggi di posta multimediali, RFC 2045, 2056 Multipurpose Internet Mail Extensions
- ❑ Alcune righe aggiuntive nell'intestazione dei messaggi dichiarano il tipo di contenuto MIME (menzionare Received:)

Versione MIME
metodo usato per codificare i dati
Tipo di dati multimediali, sottotipo, dichiarazione dei parametri
Dati codificati

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

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

Protocolli di accesso alla posta



Protocollo POP3

(tre fasi: aut, trans e agg, dopo quit)

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 per numero
- **dele**: cancella
- **quit**

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

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

POP3 (altro) e IMAP

Leggere

Ancora su POP3

- Il precedente esempio usa la modalità "scarica e cancella"
- Bob non può rileggere le e-mail se cambia client perché le ha già scaricate
- Modalità "scarica e mantieni": i messaggi sono leggibili su più client perché li lascia sul server
- POP3 è un protocollo senza stato tra le varie sessioni, non consente di organizzarsi i msg in cartelle se non su ogni singolo client

IMAP

- Mantiene tutti i messaggi in un unico posto: il server
- Sistema di cartelle su server remoto
- Consente all'utente di organizzare i messaggi in cartelle, gestendo informazioni di stato che permettono di muoversi su diverse cartelle da piu' client
- 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
 - ❖ HTTP per leggere posta col browser e' alternativa a POP e IMAP (piu' simile a IMAP perché permette gestione cartelle remote)

Capitolo 2: Livello di 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 Programmazione delle socket con TCP
- 2.8 Programmazione delle socket con UDP
- 2.9 Costruire un semplice server web



DNS: Domain Name System

Persone: molti identificatori:

- ❖ nome, codice fiscale,
carta d'identità

Host e router di Internet:

- ❖ **indirizzo IP (32 bit)** - usato per indirizzare i datagrammi
- ❖ "**nome**", ad esempio, **www.yahoo.com** - usato dagli esseri umani

D: Come associare un indirizzo IP a 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)
 - ❖ nota: **funzioni critiche** di Internet **implementate** come protocollo a livello di **applicazione**
 - ❖ complessità nelle parti periferiche della rete



DNS: a cosa serve?

- Da un dato host con un browser HTTP ci si vuole collegarsi all'host di nome www.someschool.edu
- L'host utente fa girare il lato client dell'applicazione DNS
- Il browser estrae il nome www.someschool.edu e lo passa al client DNS
- Il client DNS interroga il server DNS
- Il client DNS riceve la risposta con l'indirizzo IP di www.someschool.edu
- Il browser inizia una connessione TCP verso il processo server HTTP presso quell'indirizzo IP

DNS

Servizi DNS

- Traduzione degli hostname in indirizzi IP**
livello applicazione, non si piani
boss: delle architetture
- Host aliasing
 - ❖ un host può avere più nomi
- Mail server aliasing
- Distribuzione locale
 - ❖ server web replicati:
insieme di indirizzi IP per
un nome canonico

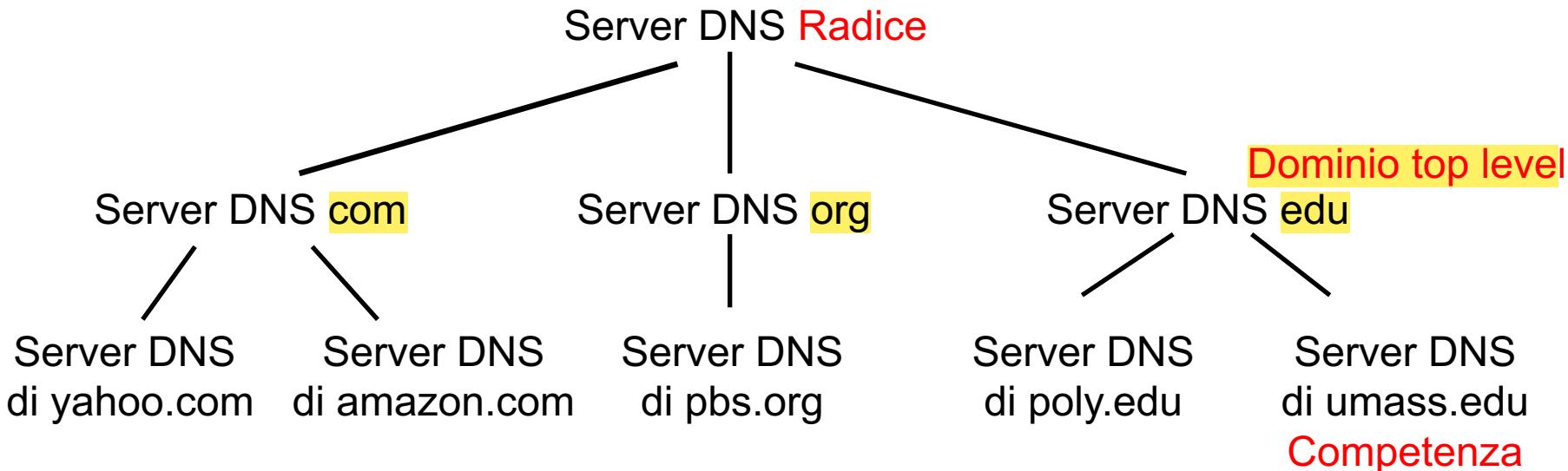
Database numero - nome
Non è scalabile,
ma può processare
numero di richieste

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 vuole l'IP di www.amazon.com; 1^a approssimazione:

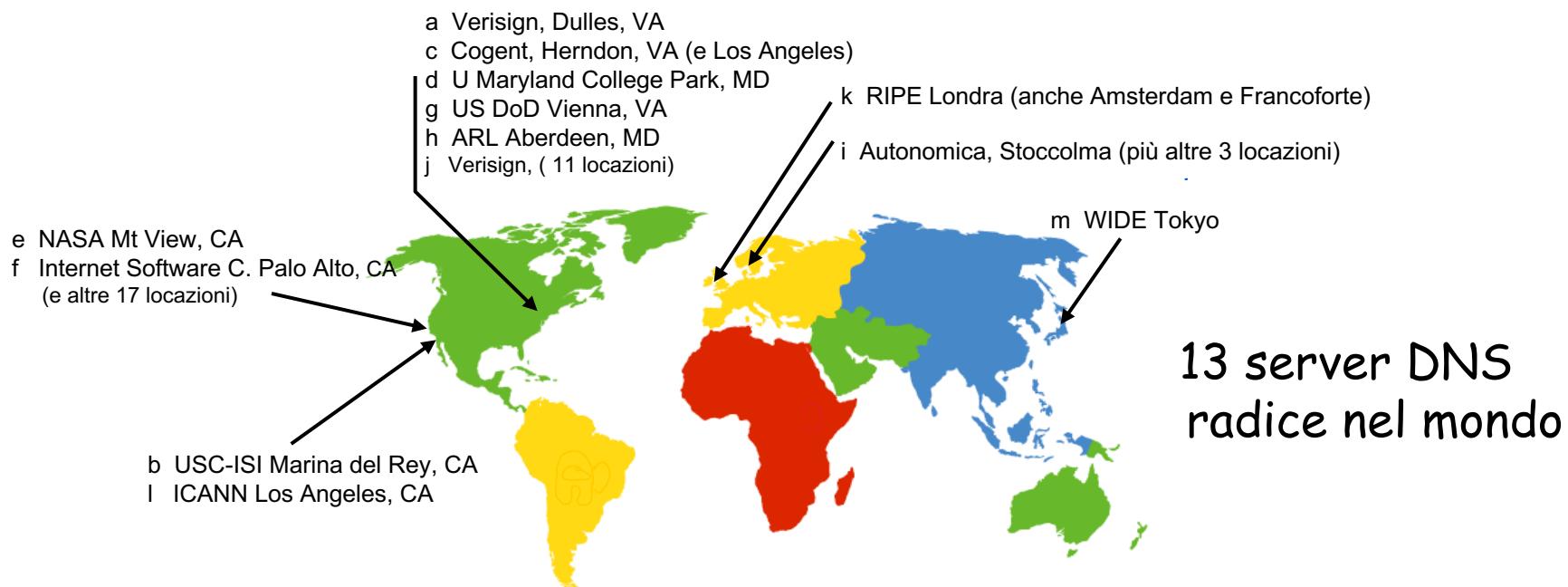
- 1 Il client interroga il server radice per trovare il server DNS com
- 2 Il client interroga il server DNS com per ottenere il server DNS amazon.com
client → root → .com → www.amazon.com → ip di www.amazon.com
- 3 Il client interroga il server DNS amazon.com per ottenere l'indirizzo IP di www.amazon.com

① Il dns locale interroga la radice e le chiede l'ip del server che gestisce il dominio .com

② Il proxy fa riferimento a

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**



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 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

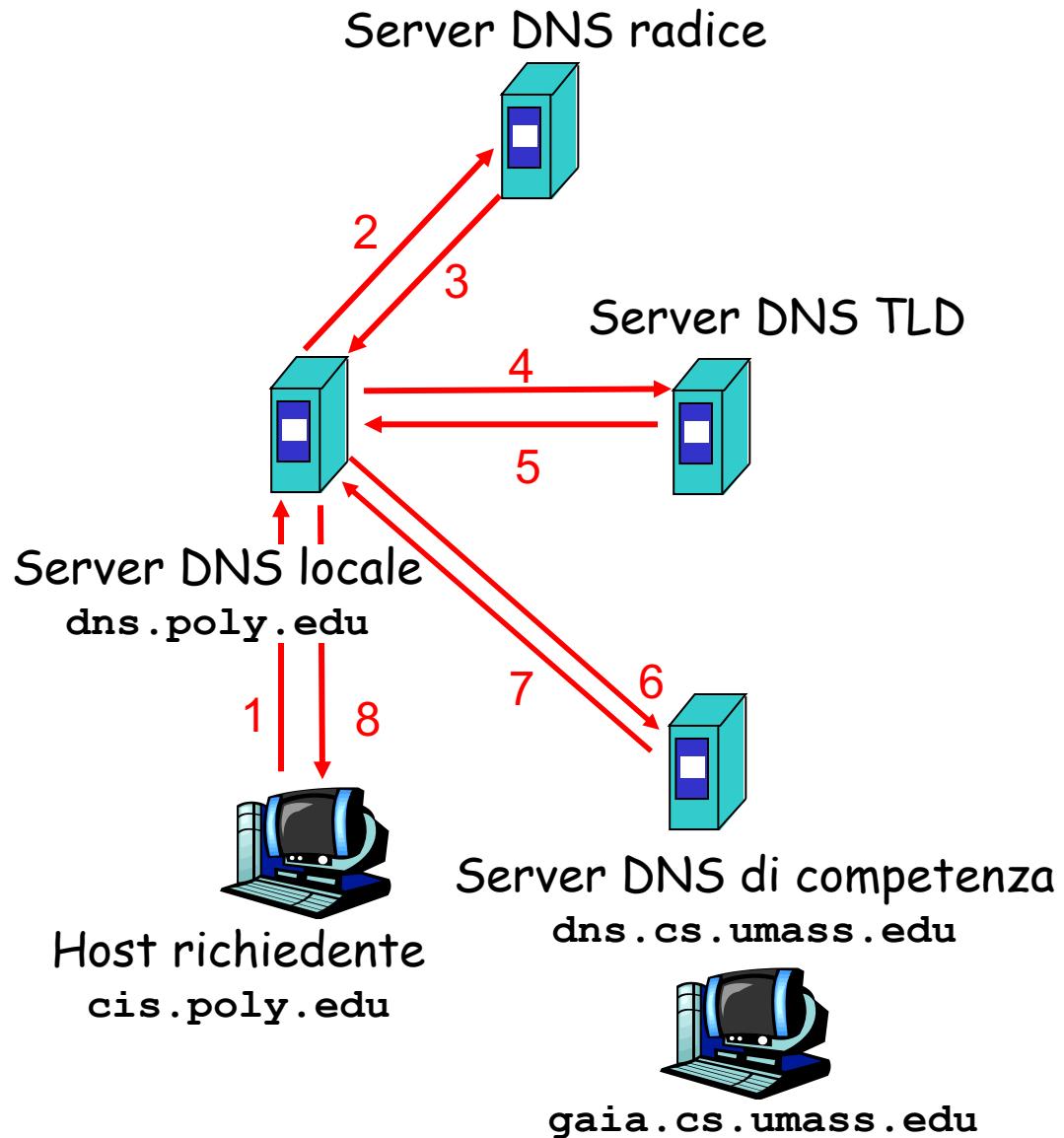
Server DNS locale

↳ caching di una certa richiesta (29.0m)

- 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



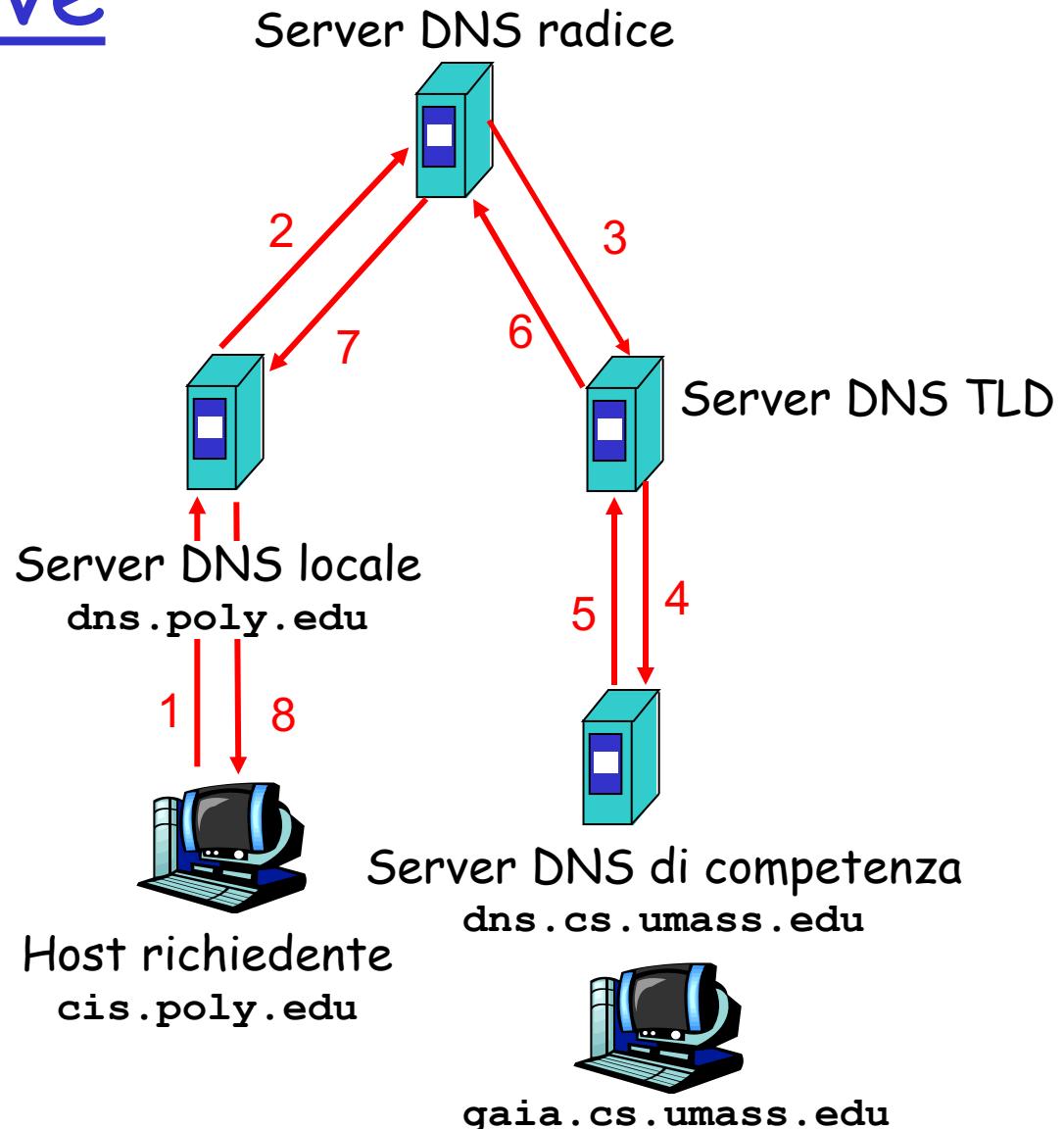
Query ricorsive

Query ricorsiva:

- ❑ Affida il compito di tradurre il nome al server DNS contattato
- ❑ Compito difficile?

Query iterativa:

- ❑ Il server contattato risponde con il nome del server da contattare
- ❑ "Non conosco questo nome, ma chiedi a questo server"



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 (2 giorni)
 - ❖ 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>

Record DNS

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

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

Time to leave

- Type=A
 - ❖ 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

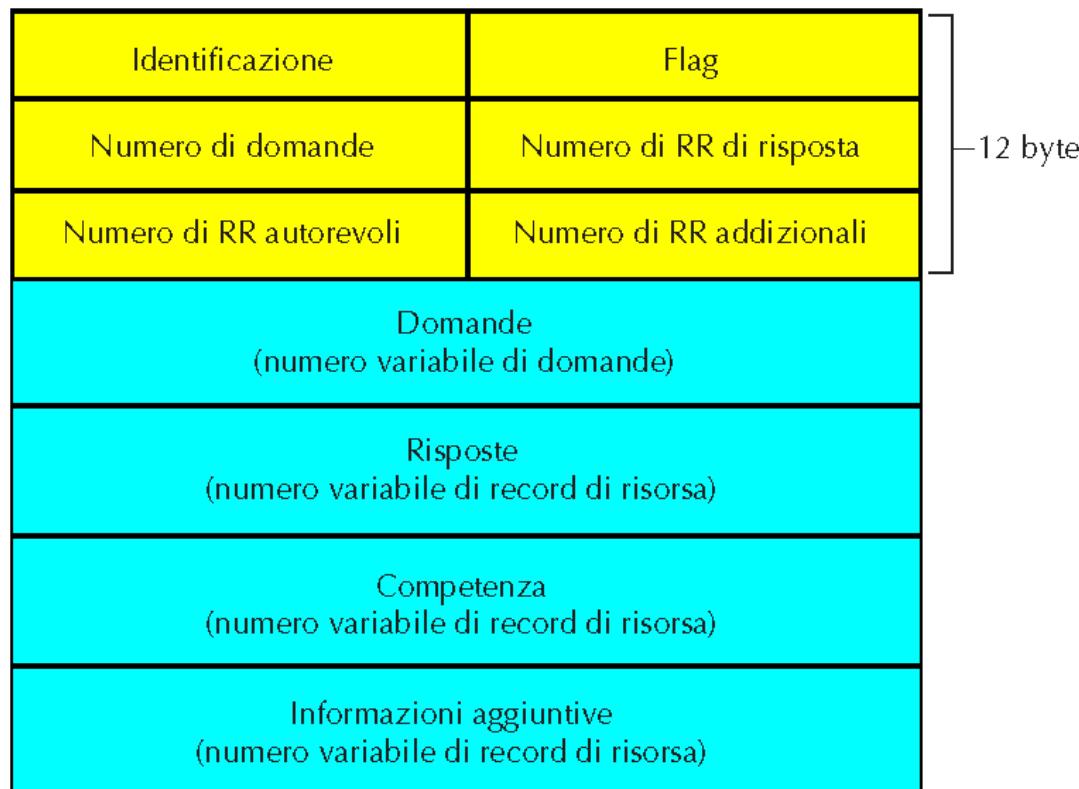
Messaggi DNS

NO

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 (b)
 - ❖ richiesta di ricors. (b)
 - ❖ Ricors. disponibile (b)
 - ❖ risposta di competenza
 - ❖ Numero di occorrenze dei campi azzurri



Messaggi DNS

Campi per il nome richiesto e tipo (A, MX) di domanda
Record Risorsa (RR) nella risposta alla domanda
Record per i server di competenza
Informazioni extra che possono essere usate (es. Mailserver, MX)
Provate a interrogare un DNS con nslookup! Disponibili anche via motori di ricerca



Inserire record nel database DNS

- Esempio: abbiamo appena avviato la nuova società "Network Utopia"
- Registriamo il nome **networkuptopia.com** presso un **registrar** (ad esempio, Network Solutions, ce ne sono vari autorizzati da ICANN, Internet Corporation for Assigned Names and Numbers)
 - ❖ Forniamo a registrar **i nomi e gli indirizzi IP dei server DNS di competenza** (primario e secondario)
 - ❖ Registrar inserisce due RR nel server TLD 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.networkuptopia.com** e un **record tipo MX** per **mail.networkutopia.com**
- **In che modo gli utenti otterranno l'indirizzo IP del nostro sito web?**

Capitolo 2: Livello di 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 Programmazione delle socket con TCP
- 2.8 Programmazione delle socket con UDP
- 2.9 Costruire un semplice server web

Condivisione di file P2P



Esempio

- Alice esegue un'applicazione di condivisione file P2P sul suo notebook
- Si collega in modo intermittente a Internet; ottiene un nuovo indirizzo IP ogni volta che si collega
- Cerca la canzone intitolata "Hey Jude"
- L'applicazione visualizza altri peer che hanno una copia di "Hey Jude"

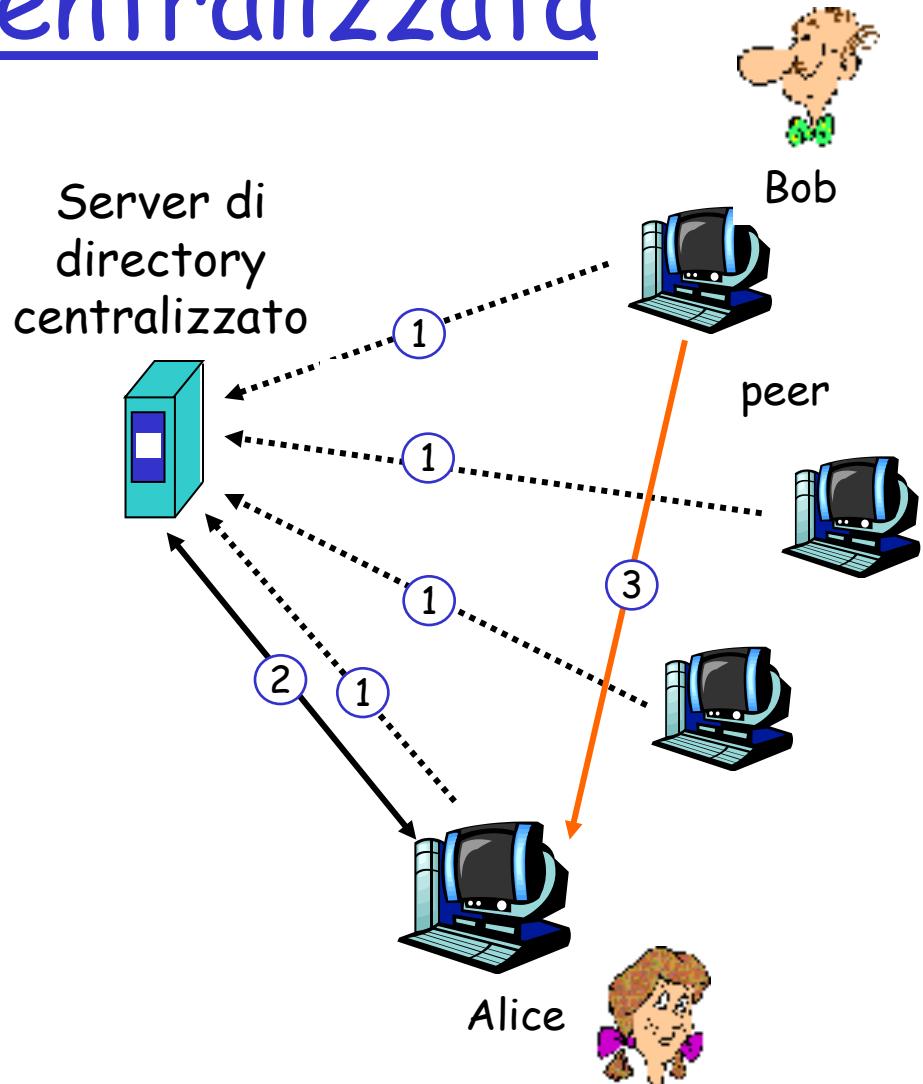
- Alice sceglie uno dei peer, Bob
- Il file viene inviato dal PC di Bob al notebook di Alice: HTTP
- Mentre Alice scarica il file, altri utenti potrebbero scaricare dei file da Alice
- Il peer di Alice è sia client web sia server web transitorio

Tutti i peer sono server = grande scalabilità! Ma come si identificano e trovano?

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) Alice richiede il file a Bob



P2P: problemi con la directory centralizzata

- Unico punto di guasto
- Collo di bottiglia per le prestazioni (database troppo ampio per molti peer)
- Violazione del diritto d'autore (vedi caso Napster)

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

Query flooding: Gnutella

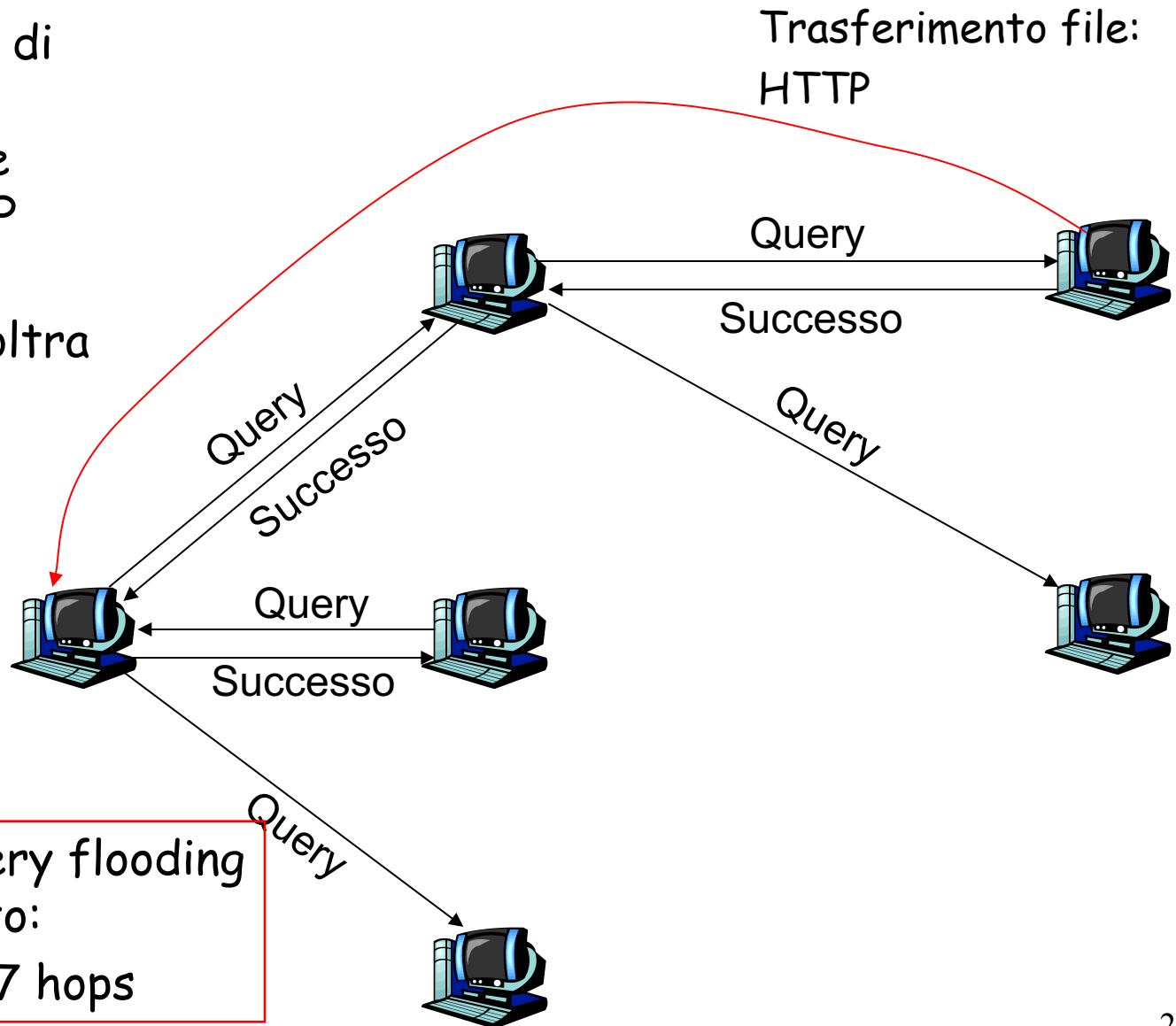
- Completamente distribuito
 - ❖ nessun server centrale
- Protocollo di pubblico dominio
- Molti client Gnutella implementano il protocollo

Rete di copertura: 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 (overlay net)
- Un arco non è un collegamento fisico (ma connessione TCP)
- Un dato peer sarà solit. connesso con meno di 10 peer vicini nella rete di copertura

Gnutella: protocollo

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



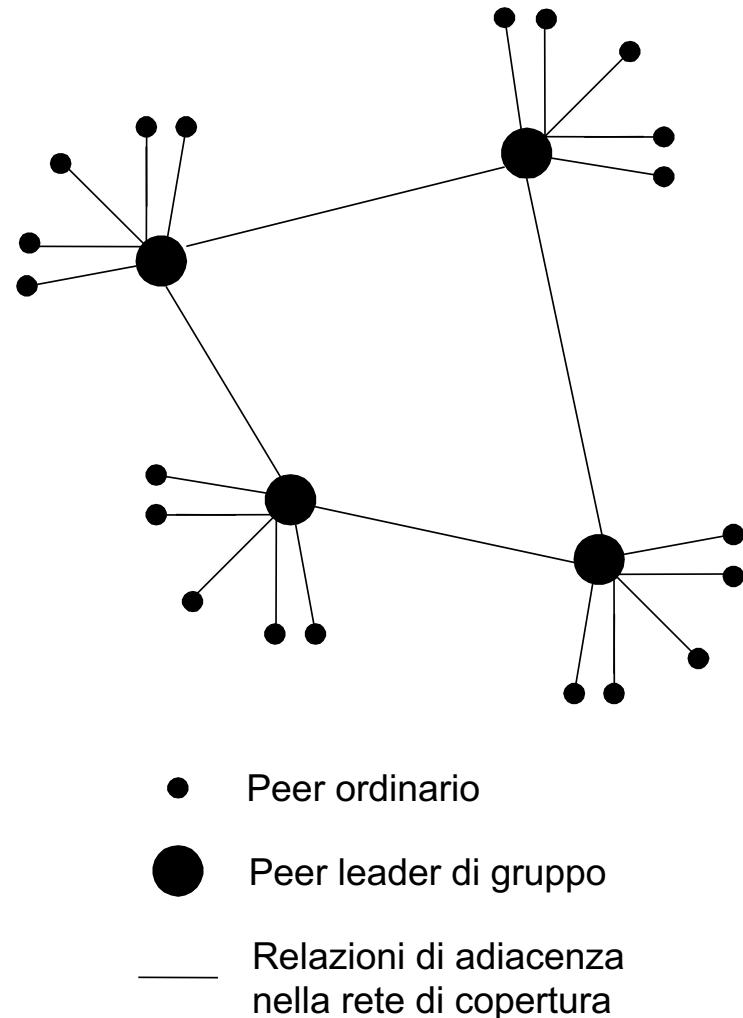
Gnutella: unione di peer

1. Per unire il peer X alla rete, bisogna trovare qualche altro peer della rete Gnutella: es. usare la lista dei peer solitamente connessi o contattare sito Gnutella (bootstrap)
2. X tenta in sequenza di impostare una connessione TCP con i peer della lista finché non stabilisce una connessione con Y
3. X invia un messaggio Ping a Y con contatore; Y inoltra il messaggio Ping ad altri Z finché cont. = 0
4. Tutti i peer che ricevono il messaggio Ping rispondono con un messaggio Pong (IP, file, dimensioni)
5. X riceve molti messaggi Pong. Quindi può impostare delle connessioni TCP addizionali

Distacco dei peer: consultate il problema alla fine del capitolo!

Sfruttare l'eterogeneità: KaZaA

- Ogni peer è un leader di gruppo o è assegnato a un leader di gruppo (stile Napster)
 - ❖ Connessione TCP tra peer e il suo leader di gruppo
 - ❖ Il leader di gruppo tiene traccia del contenuto di tutti i suoi figli (sorta di mini-hub)
- Per connettere hub tra loro, connessioni TCP tra coppie di leader di gruppo con meccanismi stili Gnutella
- È protocollo proprietario e criptato (non nei dati)



KaZaA: query

- Ogni file ha un identificatore hash e un descrittore
- Il client invia al suo leader di gruppo una query con una parola chiave
- Il leader di gruppo risponde con un elenco di peer che condividono i file i cui descrittori corrispondono alle parole chiave:
 - ❖ Per ogni corrispondenza: metadata, hash, indirizzo IP
- Se il leader di gruppo inoltra la query ad altri leader di gruppo, questi rispondono con le corrispondenze
- Il client quindi seleziona i file per il downloading
 - ❖ Le richieste HTTP che usano un identificatore hash sono trasmesse ai peer che hanno il file desiderato

Tecniche KaZaA

- Limitare il numero di upload simultanei (**non lascio scaricare più di 3-7 file simultaneamente, la quarta richiesta si accoda**)
- Accodamento delle richieste
- Priorità di incentivo (**servo dalla coda clienti che hanno scaricato meno in passato**)
- Downloading parallelo (**file recuperato a pezzi da diversi peer mediante l'intestazione byte-range di HTTP**)

No ↑

Capitolo 2: Livello di 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 Programmazione delle socket con TCP
- 2.8 Programmazione delle socket con UDP
- 2.9 Costruire un semplice server web

Programmazione delle socket

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

Socket API

- introdotta in BSD4.1 UNIX, 1981
- esplicitamente creata, usata, distribuita dalle applicazioni
- paradigma client/server
- due tipi di servizio di trasporto tramite una socket API:
 - ❖ datagramma inaffidabile
 - ❖ affidabile, orientata ai byte

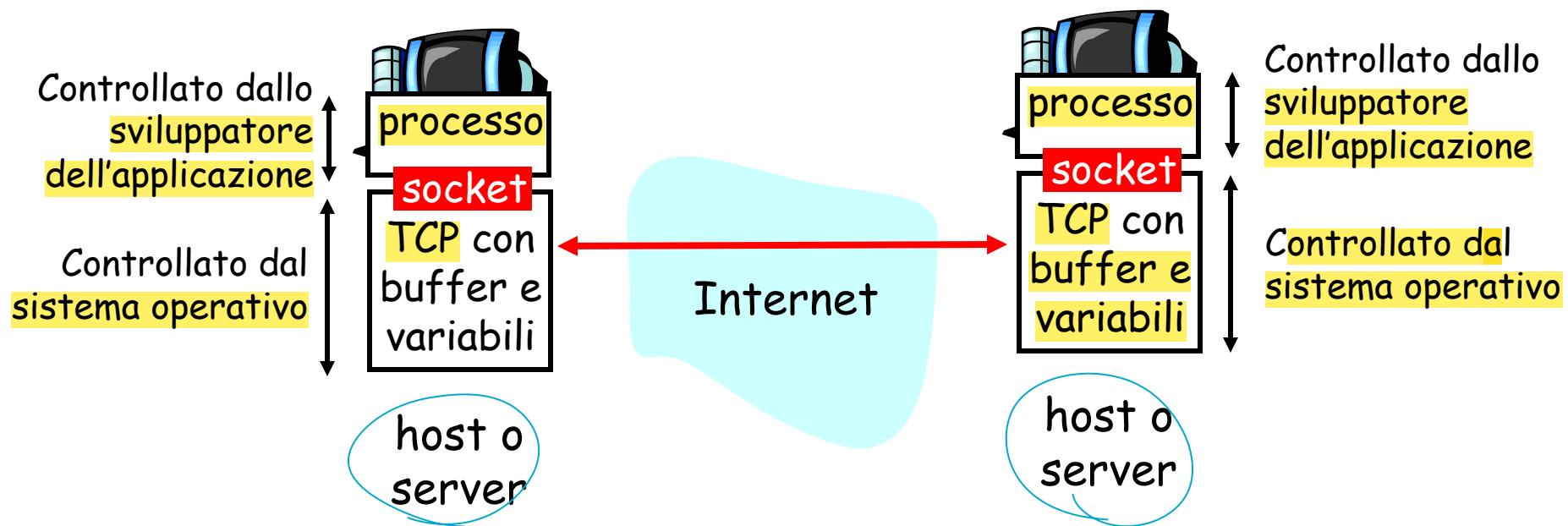
socket

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



Programmazione delle socket con TCP

- 1 Il client deve contattare il server
- Il processo server deve essere in corso di esecuzione
- 2 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, 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

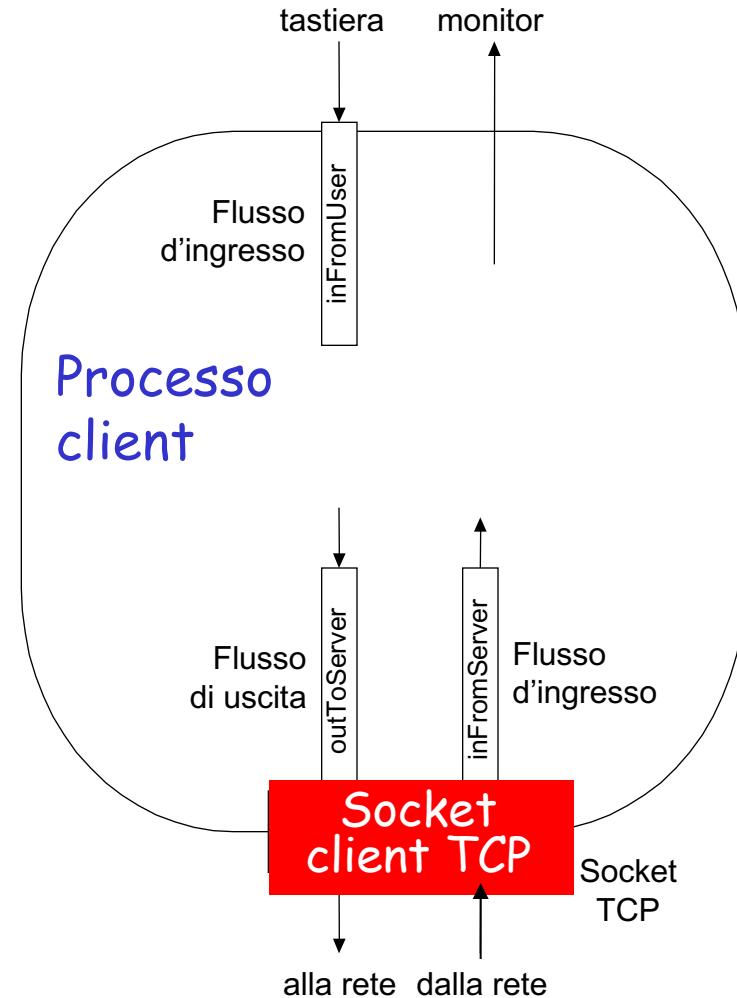
Termini

- 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`)



Interazione delle socket client/server: TCP

Server (gira su hostid)

- 1 crea la socket
port=x per la richiesta
in arrivo:
`welcomeSocket =
ServerSocket()`
- 2 attende la richiesta di connessione in ingresso
`connectionSocket =
welcomeSocket.accept()`
- 3 legge la richiesta da `connectionSocket`
- 4 scrive la risposta a `connectionSocket`
- 5 chiude `connectionSocket`

Client

- 1 crea la socket
connessa a `hostid`, port=x
`clientSocket =
Socket()`
- 2 invia la richiesta usando `clientSocket`
- 3 legge la risposta da `clientSocket`
- 4 chiude `clientSocket`

Setup della connessione TCP

Esempio: client Java (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        Crea un flusso d'ingresso → BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));

        Crea una socket client, connessa al server → Socket clientSocket = new Socket("hostname", 6789);
                                                ↑
                                                ↑ Porba

        Crea un flusso di uscita collegato alla socket → DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
```

Esempio: client Java (TCP), continua

```
    Crea  
    un flusso d'ingresso  
    collegato alla socket } → BufferedReader inFromServer =  
                                new BufferedReader(new  
                                InputStreamReader(clientSocket.getInputStream()));  
  
    Invia una riga  
    al server } → sentence = inFromUser.readLine();  
                outToServer.writeBytes(sentence + '\n');  
  
    Legge la riga  
    dal server } → modifiedSentence = inFromServer.readLine();  
                System.out.println("FROM SERVER: " + modifiedSentence);  
  
                clientSocket.close();  
            }  
        }
```

Esempio: server Java (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

    public static void main(String argv[]) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;

        Crea una socket
        di benvenuto
        sulla porta 6789 }→ ServerSocket welcomeSocket = new ServerSocket(6789);

        Attende, sulla socket
        di benvenuto,
        un contatto dal client }→ while(true) {
                                Socket connectionSocket = welcomeSocket.accept();

        Crea un
        flusso d'ingresso
        collegato alla socket }→ BufferedReader inFromClient =
                                new BufferedReader(new
                                InputStreamReader(connectionSocket.getInputStream()));


```

Esempio: server Java (TCP), continua

Crea un flusso di uscita collegato alla socket

```
DataOutputStream outToClient =  
    new DataOutputStream(connectionSocket.getOutputStream());
```

Legge la riga dalla socket

```
clientSentence = inFromClient.readLine();
```

```
capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Scrive la riga sulla socket

```
outToClient.writeBytes(capitalizedSentence);
```

```
}
```

Fine del ciclo while,
ricomincia il ciclo e attende
un'altra connessione con il client

Capitolo 2: Livello di 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 Programmazione delle socket con TCP
- 2.8 Programmazione delle socket con UDP
- 2.9 Costruire un semplice server web

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

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

Punto di vista dell'applicazione

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

Interazione delle socket client/server: UDP

Server (gira su hostid)

crea la socket
port=x per la
richiesta in arrivo:
`serverSocket =
DatagramSocket()`

legge la richiesta da
`serverSocket`

scrive la risposta a
`serverSocket`
specificando l'indirizzo
dell'host client e
il numero di porta

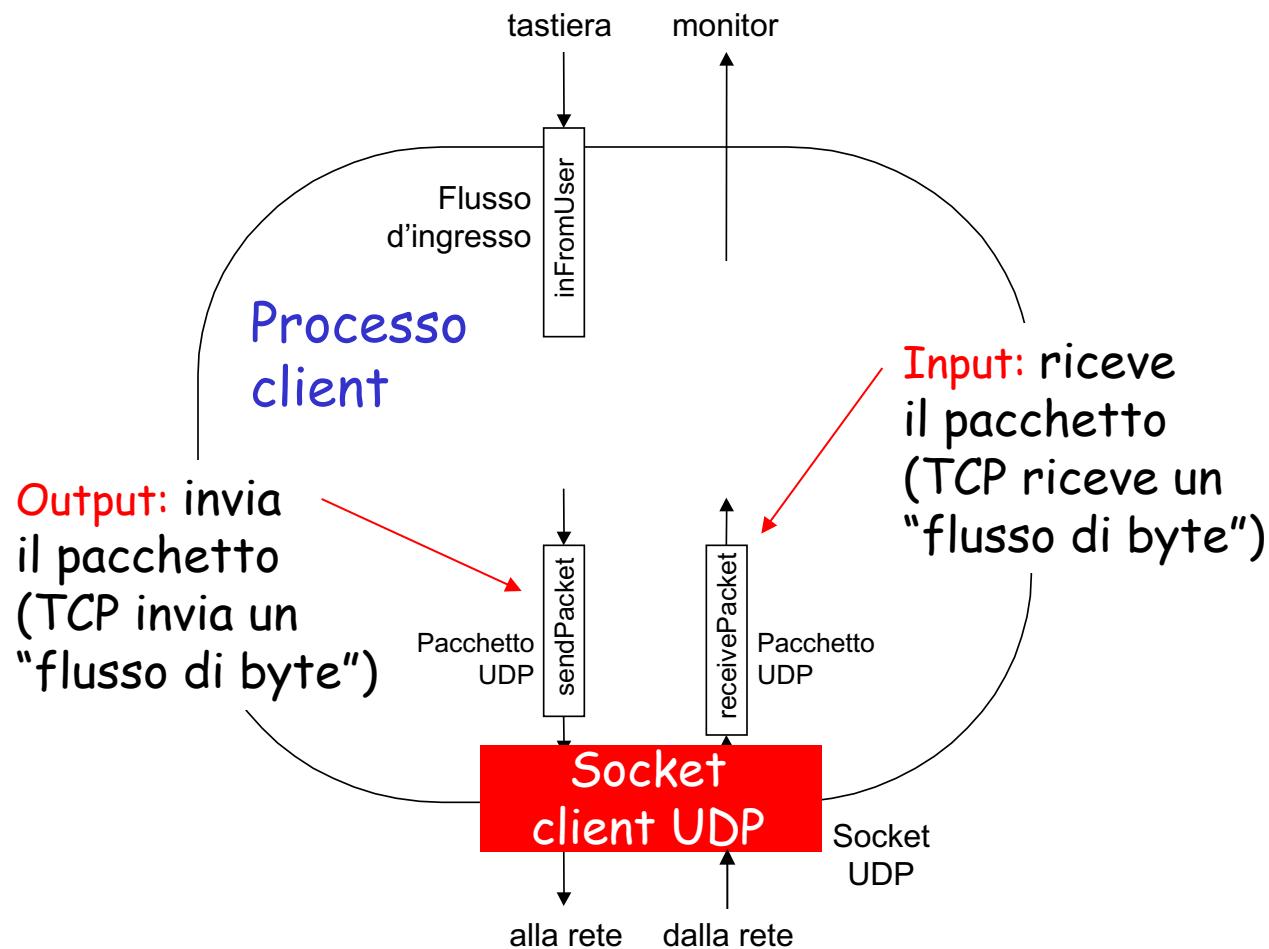
Client

crea la socket
`clientSocket =
DatagramSocket()`

crea l'indirizzo (`hostid, port=x`)
e invia la richiesta di datagrammi
usando `clientSocket`

legge la risposta da
`clientSocket`
chiude
`clientSocket`

Esempio: client Java (UDP)



Esempio: client Java (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("hostname");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
    }
}
```

Crea un flusso d'ingresso

Crea una socket client

Traduce il nome dell'host nell'indirizzo IP usando DNS

Esempio: client Java (UDP), continua

Crea il datagramma
con i dati da
trasmettere,
lunghezza,
indirizzo IP, porta

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
```

Invia
il datagramma
al server

```
clientSocket.send(sendPacket);
```

Legge
il datagramma
dal server

```
DatagramPacket receivePacket =
```

```
    new DatagramPacket(receiveData, receiveData.length);
```

```
clientSocket.receive(receivePacket);
```

```
String modifiedSentence =  
    new String(receivePacket.getData());
```

```
System.out.println("FROM SERVER:" + modifiedSentence);
```

```
clientSocket.close();
```

```
}
```

```
}
```

Esempio: server Java (UDP)

```
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception
    {
        DatagramSocket serverSocket = new DatagramSocket(9876);

        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];

        while(true)
        {

            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);

            serverSocket.receive(receivePacket);
        }
    }
}
```

Crea una socket per
datagrammi
sulla porta 9876

Crea lo spazio per
i datagrammi

Riceve i
datagrammi

Esempio: server Java (UDP), continua

```
String sentence = new String(receivePacket.getData());  
InetAddress IPAddress = receivePacket.getAddress();  
int port = receivePacket.getPort();  
  
String capitalizedSentence = sentence.toUpperCase();  
  
sendData = capitalizedSentence.getBytes();  
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length, IPAddress,  
                      port);  
  
serverSocket.send(sendPacket);  
}  
}  
  
Fine del ciclo while,  
ricomincia il ciclo e attende  
un altro datagramma
```

Ottiene l'indirizzo IP e il numero di porta del mittente

Crea il datagramma da inviare al client

Scrive il datagramma sulla socket

Fine del ciclo while, ricomincia il ciclo e attende un altro datagramma

Capitolo 2: Livello di 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 Programmazione delle socket con TCP
- 2.8 Programmazione delle socket con UDP
- 2.9 *Costruire un semplice server web*

Costruire un semplice server web

- gestisce una richiesta HTTP
- accetta la richiesta
- analizza l'intestazione
- prende il file richiesto dal file system del server
- crea un messaggio di risposta HTTP:
 - ❖ righe di intestazione + file
- invia la risposta al client
- dopo avere creato il server, potete richiedere il file utilizzando un browser (ad esempio, Internet Explorer)
- Vedere il testo per i dettagli

Capitolo 2: Riassunto

Lo studio delle applicazioni di rete adesso è completo!

- 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
- Programmazione delle socket

Riassunto

Molto importante: conoscere i protocolli

- 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
- “Complessità nelle parti periferiche della rete”