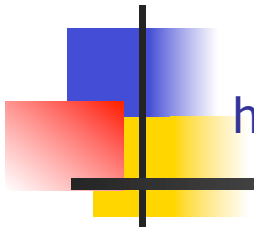
Reti di Calcolatori
AA 2011/2012

UNIVERSITÀ DEGLI STUDI DI TRENTO



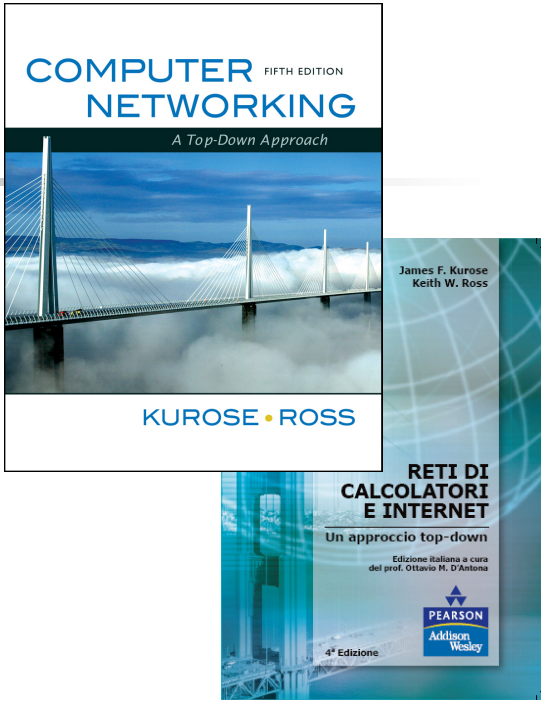
http://disi.unitn.it/locigno/index.php/teaching-duties/computer-networks

Protocolli di applicazione

Csaba Kiraly

Renato Lo Cigno

Livello di applicazione



A note on the use of these slides:

These slides are an adaptation from the freely available version provided by the book authors to all (faculty, students, readers). The originals are in PowerPoint and English.

The Italian translation is originally form
Gianluca Torta, Stefano Leonardi, Francesco Di Tria

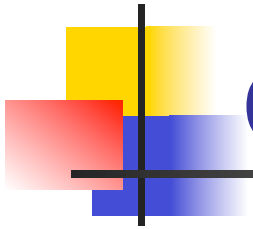
Adaptation is by Csaba Kiraly and Renato Lo Cigno

Computer Networking: A Top Down Approach,
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.

Reti di calcolatori e Internet: Un approccio top-down
4ª edizione
Pearson Paravia Bruno Mondadori Spa
©2008

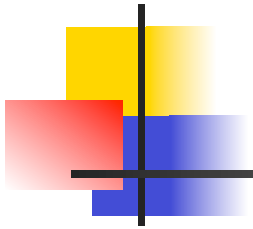
{kiraly,locigno}@disi.unitn.it

UNIVERSITÀ DEGLI STUDI DI TRENTO



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



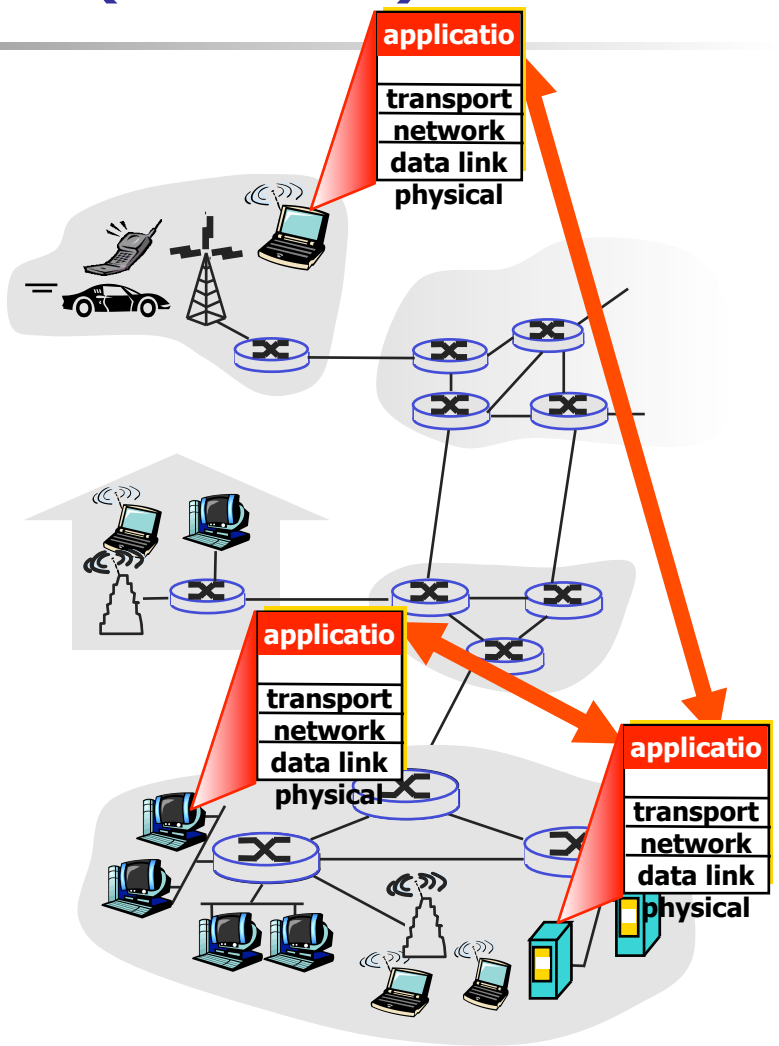
Applicazioni comuni (in rete)

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

Creare applicazioni (in rete)

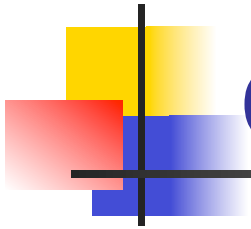
Scrivere programmi che

- girano su *end systems*
- comunicano sulla rete
- implementano un **protocollo a livello applicazione** (non l'applicazione stessa)



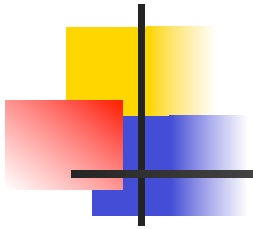
Non è necessario scrivere software per dispositivi interni alla rete

- I dispositivi di rete non eseguono applicazioni utente
- Rapido sviluppo di applicazioni



Capitolo 2: Livello di applicazione

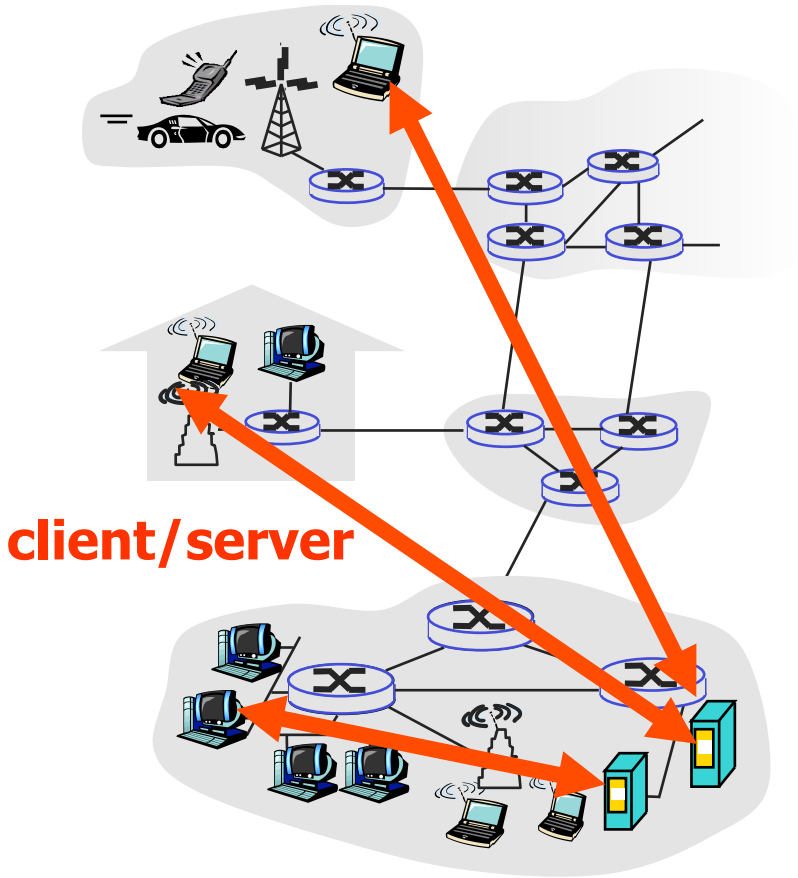
- 2.1 Principi delle applicazioni di rete
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Posta Elettronica
  SMTP, POP3, IMAP
- 2.5 DNS



Architetture delle applicazioni di rete

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

Architettura client-server



server:

- host sempre attivo
- indirizzo IP fisso e noto al client
- server farm (=un hostname con più indirizzi IP) per creare un potente server virtuale

client:

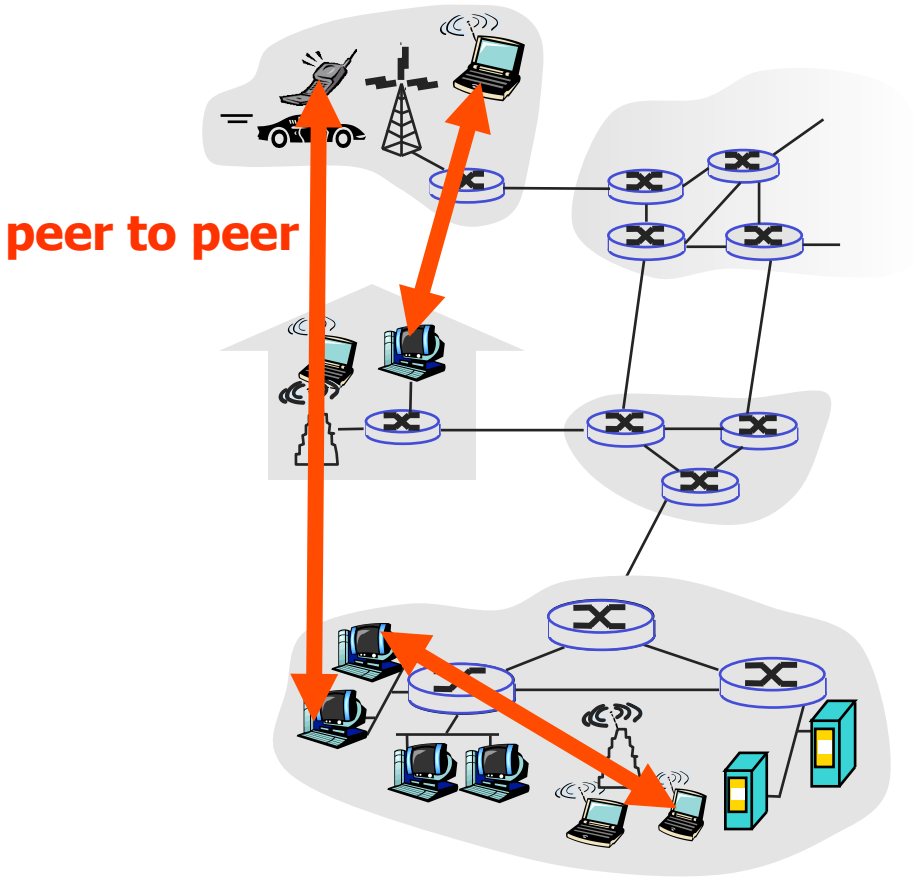
- comunica con il server
- può contattare il server in qualunque momento
- può avere indirizzi IP dinamici
- non comunica direttamente con gli altri client

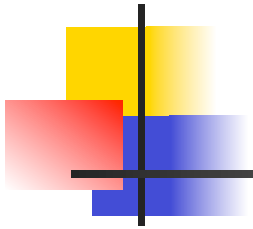
Architettura P2P pura

- 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

Facilmente scalabile

Difficile da gestire





Ibridi (client-server e P2P)

Skype

- Applicazione P2P di Voice over IP
- Server centralizzato: Autenticazione
- Ricerca utenti e indirizzi (Rubrica telefonica): P2P, con l'aiuto di SuperPeer che normalmente hanno indirizzi pubblici
- Connessione client-client: diretta o attraverso SuperPeer (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

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

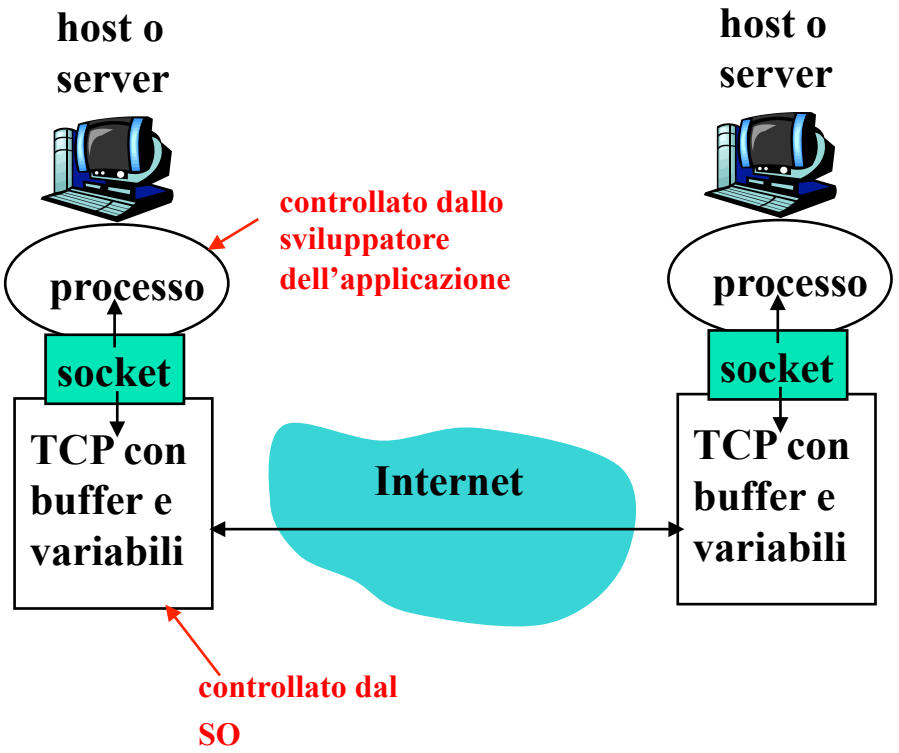
Processo client: processo che dà inizio alla comunicazione

Processo server : processo che attende di essere contattato

le applicazioni con architetture P2P hanno processi client e processi server

Socket

- un processo invia/riceve messaggi a/da la sua socket
- un socket è analogo a un punto di accesso/uscita
  - un processo che vuole inviare un messaggio, lo fa uscire dalla propria "interfaccia" (socket)
  - il processo presuppone l'esistenza di un'infrastruttura esterna che trasporterà il messaggio attraverso la rete fino alla "interfaccia" del processo di destinazione
- Si usano API che consentono:
  - scelta del protocollo di trasporto
  - capacità di determinare alcuni parametri



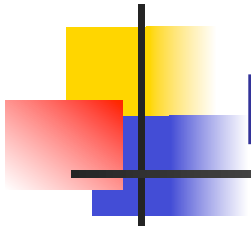
Le chiamate ai socket sono le "primitive" del protocollo

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 IP univoco a 32 bit

- Domanda: 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 di 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
- 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%

Temporizzazione

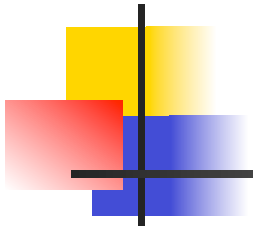
- alcune applicazioni (ad esempio, telefonia Internet, giochi interattivi) per essere "realistiche" richiedono piccoli ritardi

Throughput

- **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**

Sicurezza

- **Cifratura, integrità dei dati, …**



Requisiti del servizio di trasporto di alcune applicazioni comuni

Applicazione	Tolleranza alla perdita di dati	Throughput	Sensibilità al tempo e tolleranza ai ritardi
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:

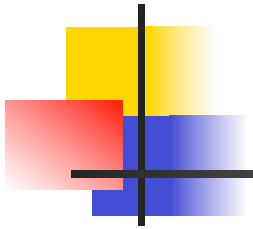
- *orientato alla connessione:* è richiesto un setup fra i processi client e server (handshaking)
- *trasporto affidabile* fra i processi d'invio e di ricezione
- *controllo di flusso:* il mittente non vuole sovraccaricare il destinatario
- *controllo della congestione:* "strozza" il processo d'invio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

Servizio di 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
- Usato per applicazioni in tempo reale (tollerano perdita di dati ma richiedono una frequenza minima di trasmissione)

Applicazioni Internet: protocollo a livello applicazione e protocollo di trasporto

Applicazione	Protocollo di applicazione	Protocollo di trasporto
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 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

Web: HTML e HTTP

Terminologia HTML (da non confondere con HTTP!)

- 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
- Esempio di URL:

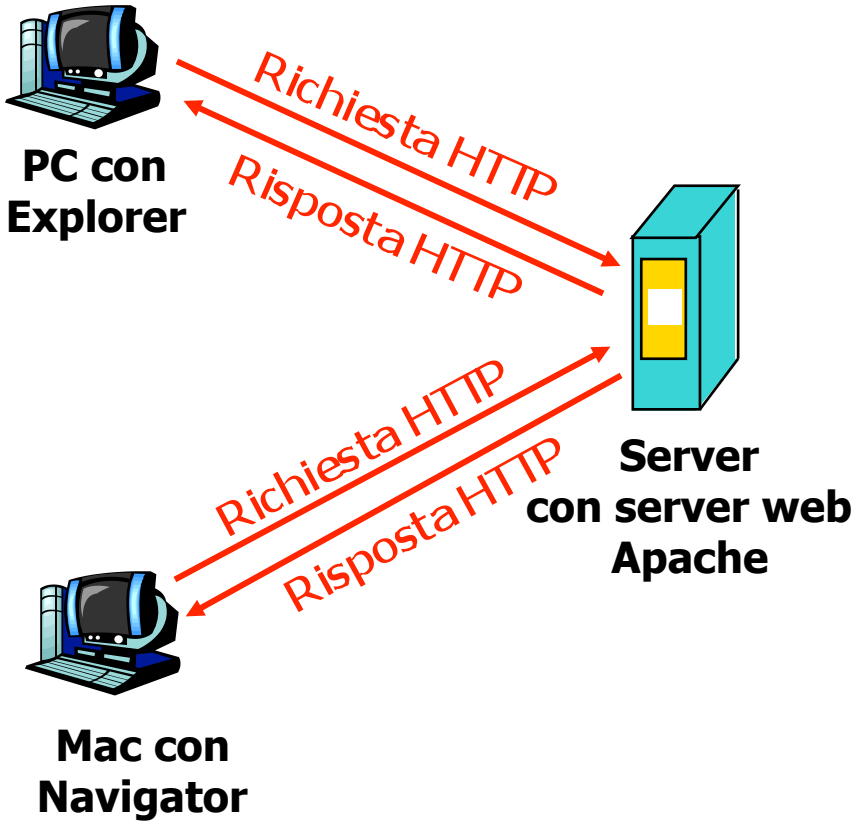
`http://www.someschool.edu/someDept/pic.gif`

protocol **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



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

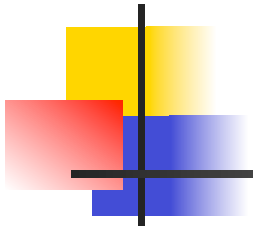
HTTP è un protocollo "senza stato" (stateless)

- 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

- Un singolo oggetto per volta 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.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

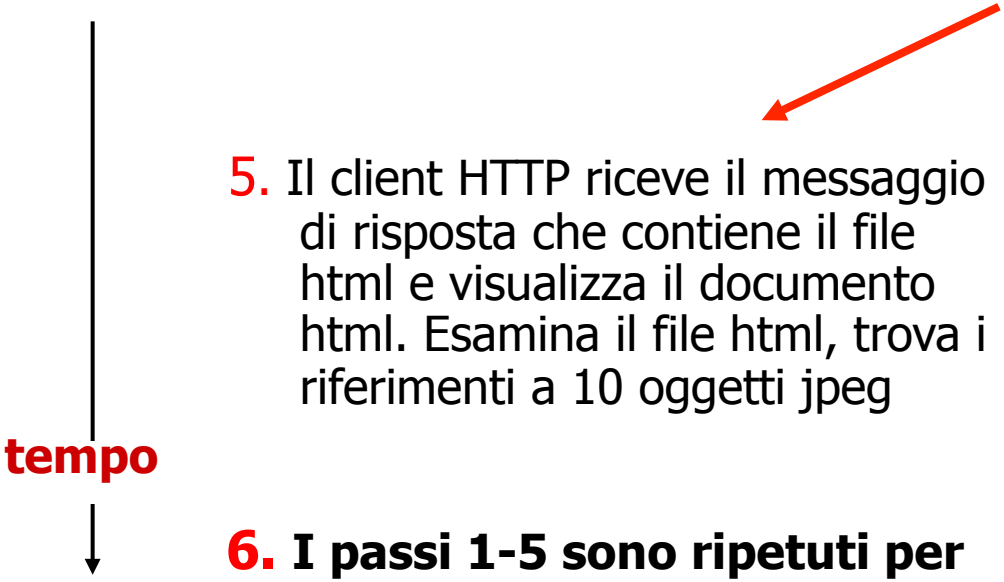
1b. Il server HTTP all'host www.someSchool.edu in attesa di una connessione TCP alla porta 80 "accetta" la connessione e avvisa il client

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

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 (cont.)



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

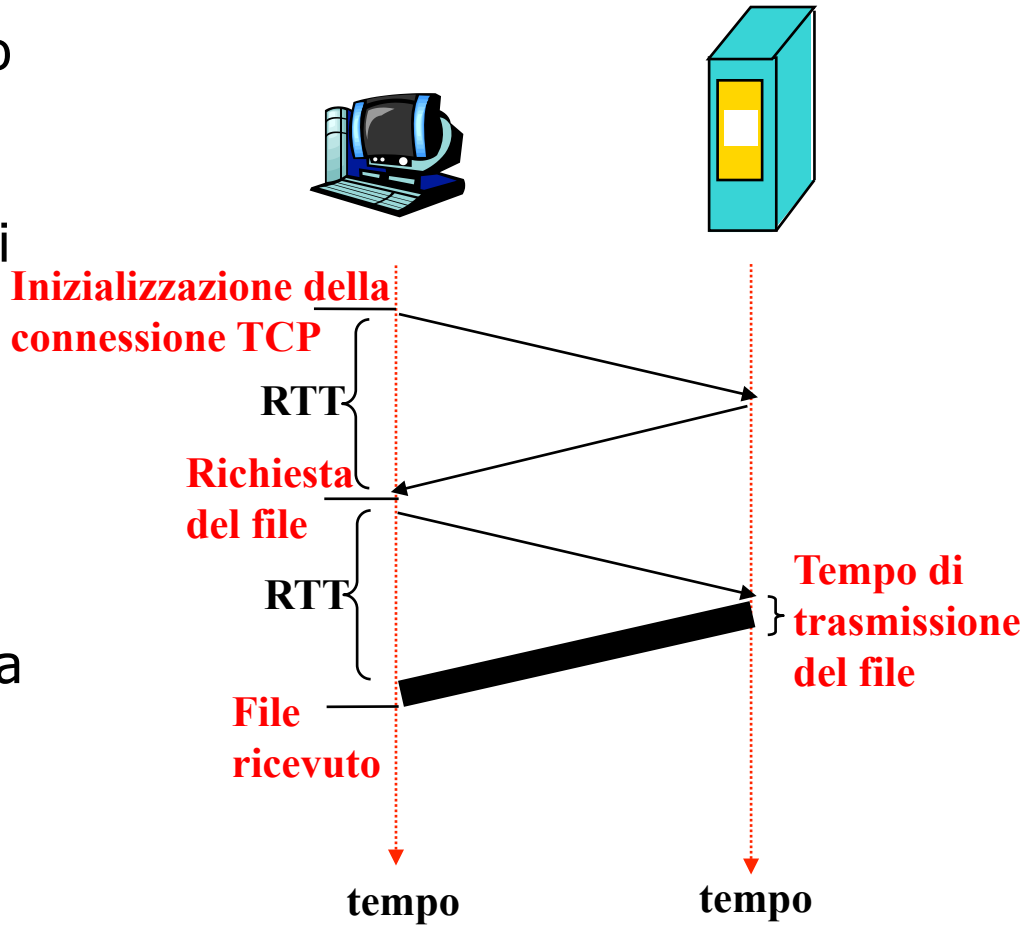
6. I passi 1-5 sono ripetuti per ciascuno dei 10 oggetti jpeg

Calcolo del tempo di risposta

Definizione di RTT: tempo impiegato da un piccolo pacchetto per andare dal client al server e per una eventuale risposta (breve) di 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



totale = 2RTT + tempo di trasmissione

Connessioni persistenti

Connessioni non persistenti:

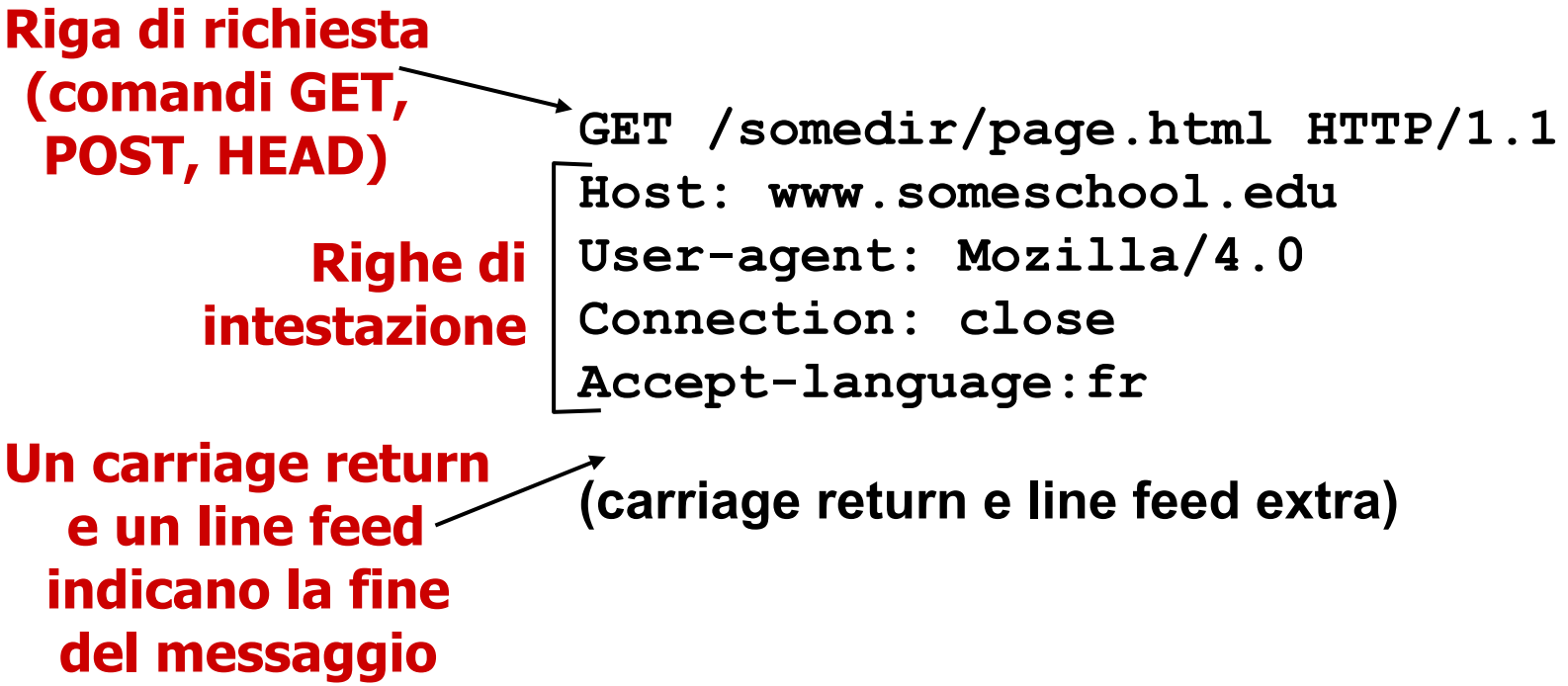
- richiedono 2 RTT per oggetto
- overhead del sistema operativo per *ogni* connessione TCP
- i browser spesso aprono connessioni TCP parallele per caricare gli oggetti referenziati

Connessioni persistenti

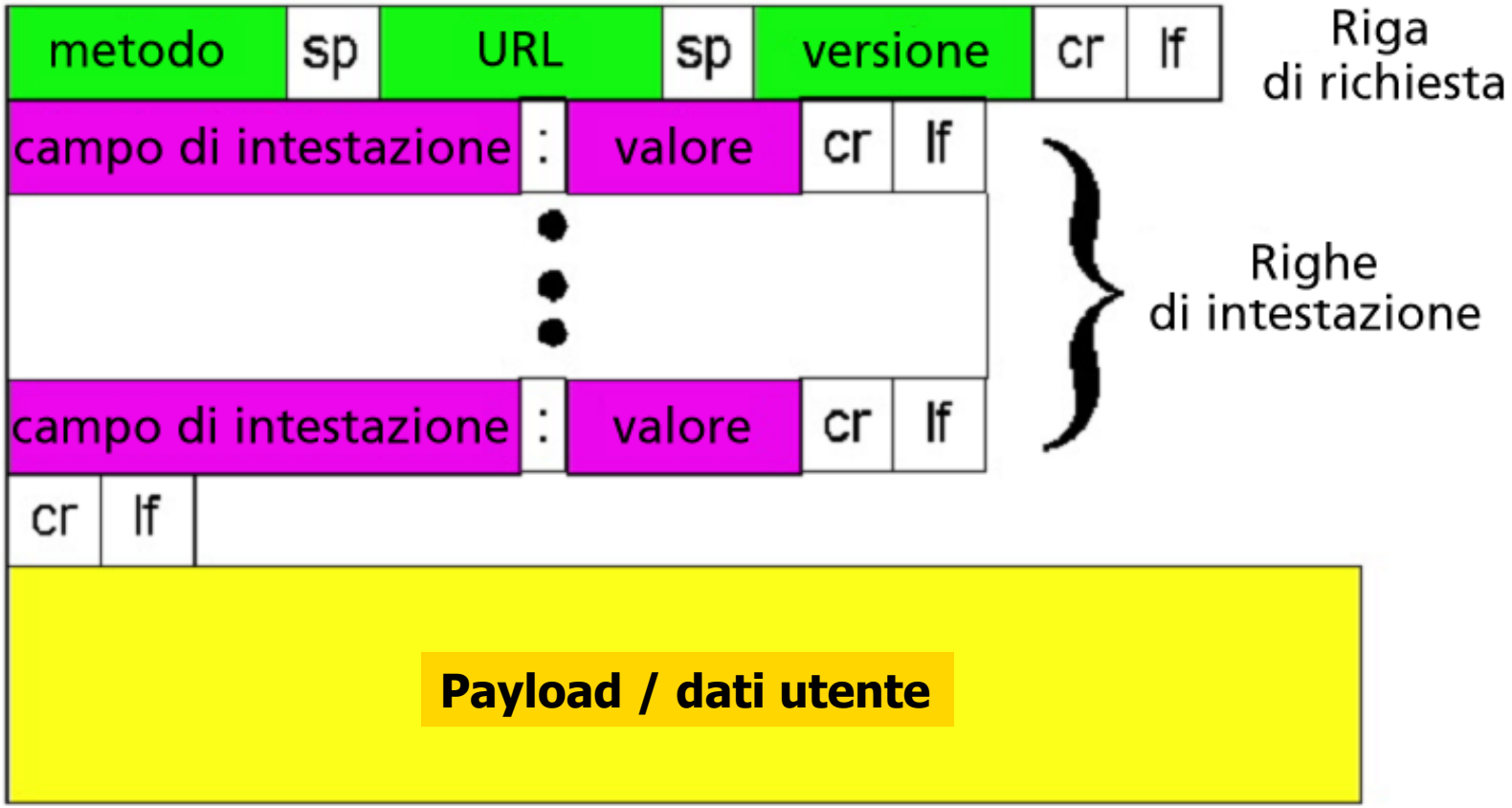
- il server lascia la connessione TCP aperta dopo l'invio di una risposta
- i successivi messaggi tra gli stessi client/server vengono trasmessi sulla connessione aperta
- il client invia le richieste non appena incontra un oggetto referenziato
- un solo RTT per ogni oggetto richiesto
- Con pipelining:
  - Il client invia le richieste a raffica senza aspettare i precedenti oggetti
  - Un solo RTT di attesa per tutti gli oggetti

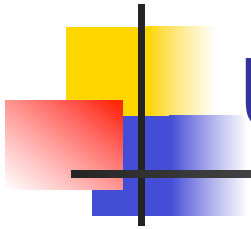
Messaggi HTTP

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



Messaggio di richiesta HTTP: formato generale





Upload dell'input di un form

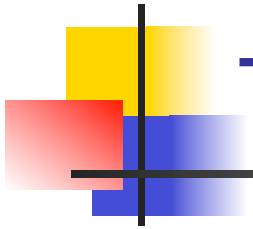
Metodo Post:

- Una pagina web a volte può includere spazi e campi per consentire "input" di dati da parte dell'utente
- I dati di input arrivano al server nel payload

Metodo GET:

- Non richiede in genere dati utente e arriva al server nel campo URL della riga di richiesta:

```
www.somesite.com/search?a=2&b=5
```



Tipi di metodi

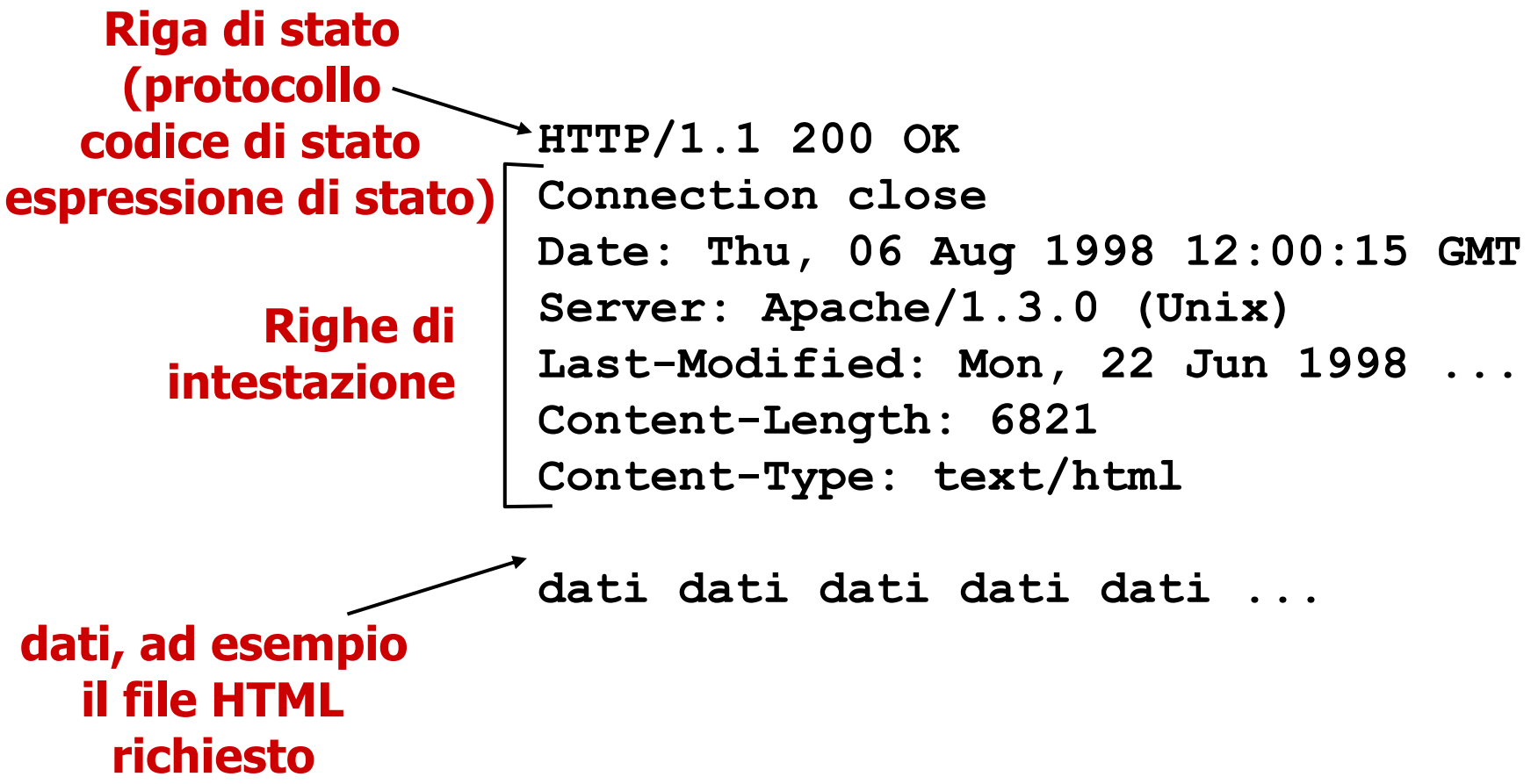
HTTP/1.0

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

HTTP/1.1

- GET, POST, HEAD
- PUT
  - include il file (o oggetto) specificato nel payload e lo invia al percorso specificato nel campo URL del messaggio
- DELETE
  - cancella il file specificato nel campo URL

Messaggio di risposta HTTP



Codici di stato della risposta HTTP

Sono sempre il contenuto della 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

Esempio di richieste HTTP

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

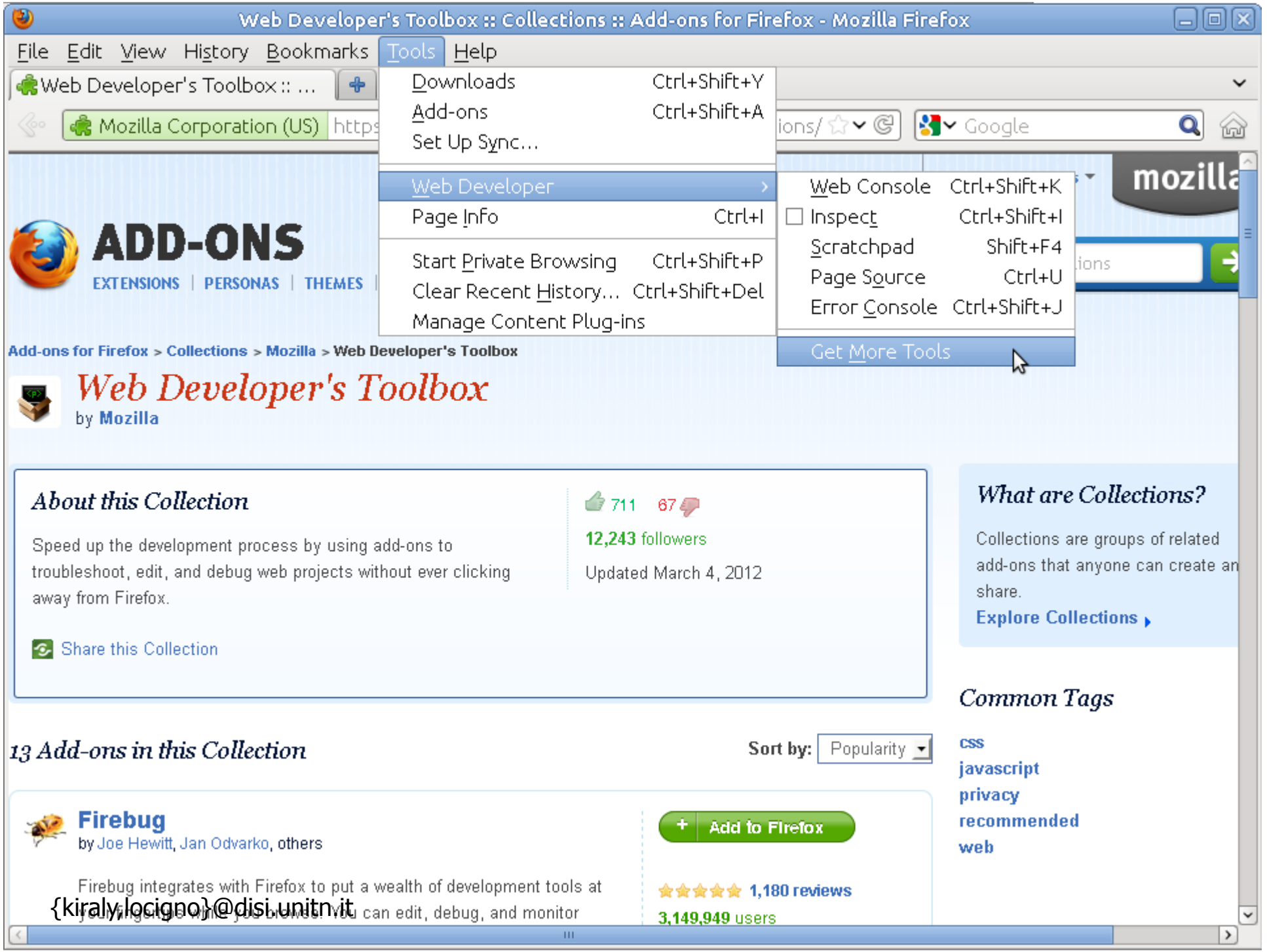
2. Digitate una richiesta GET:

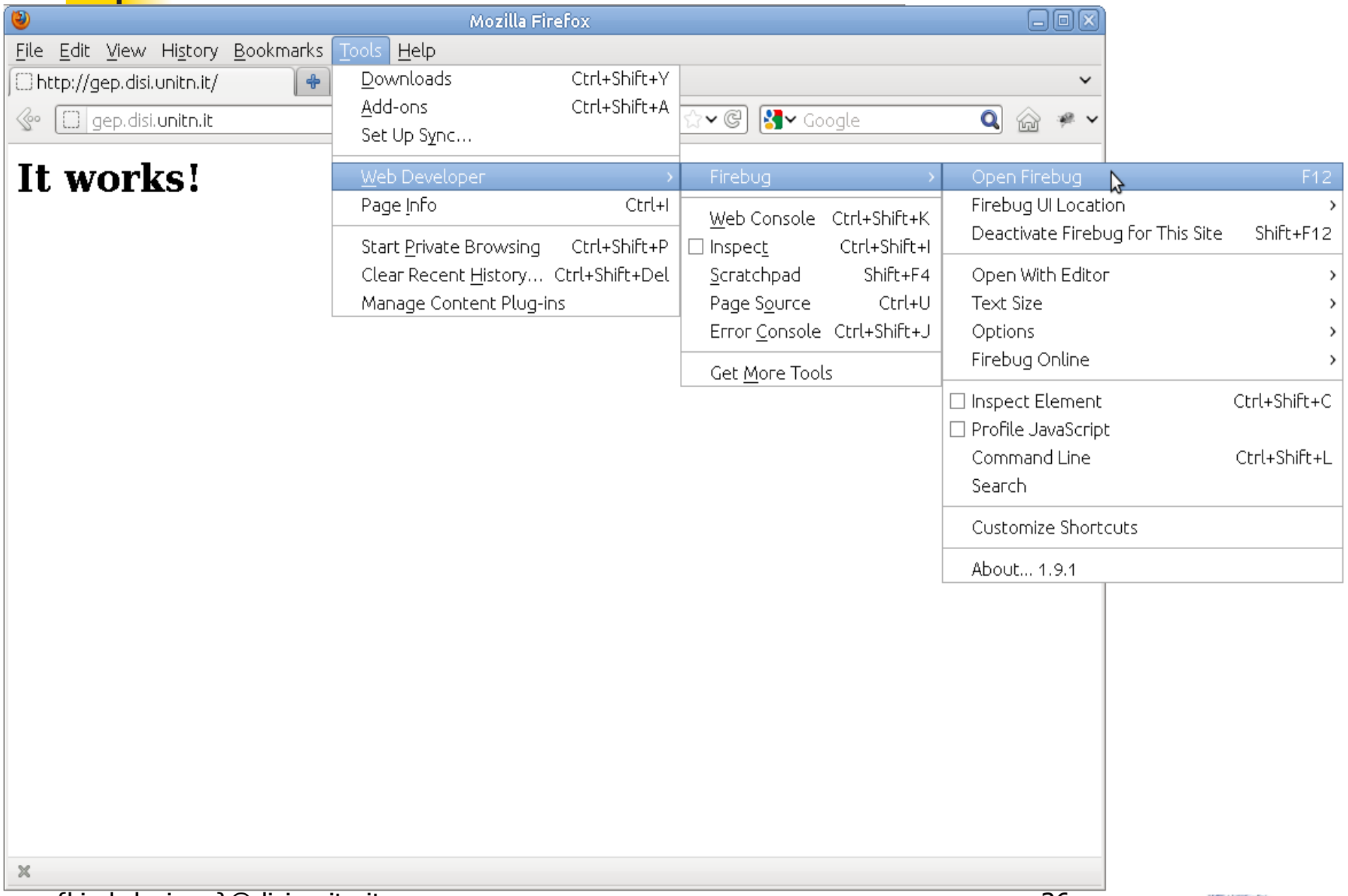
```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

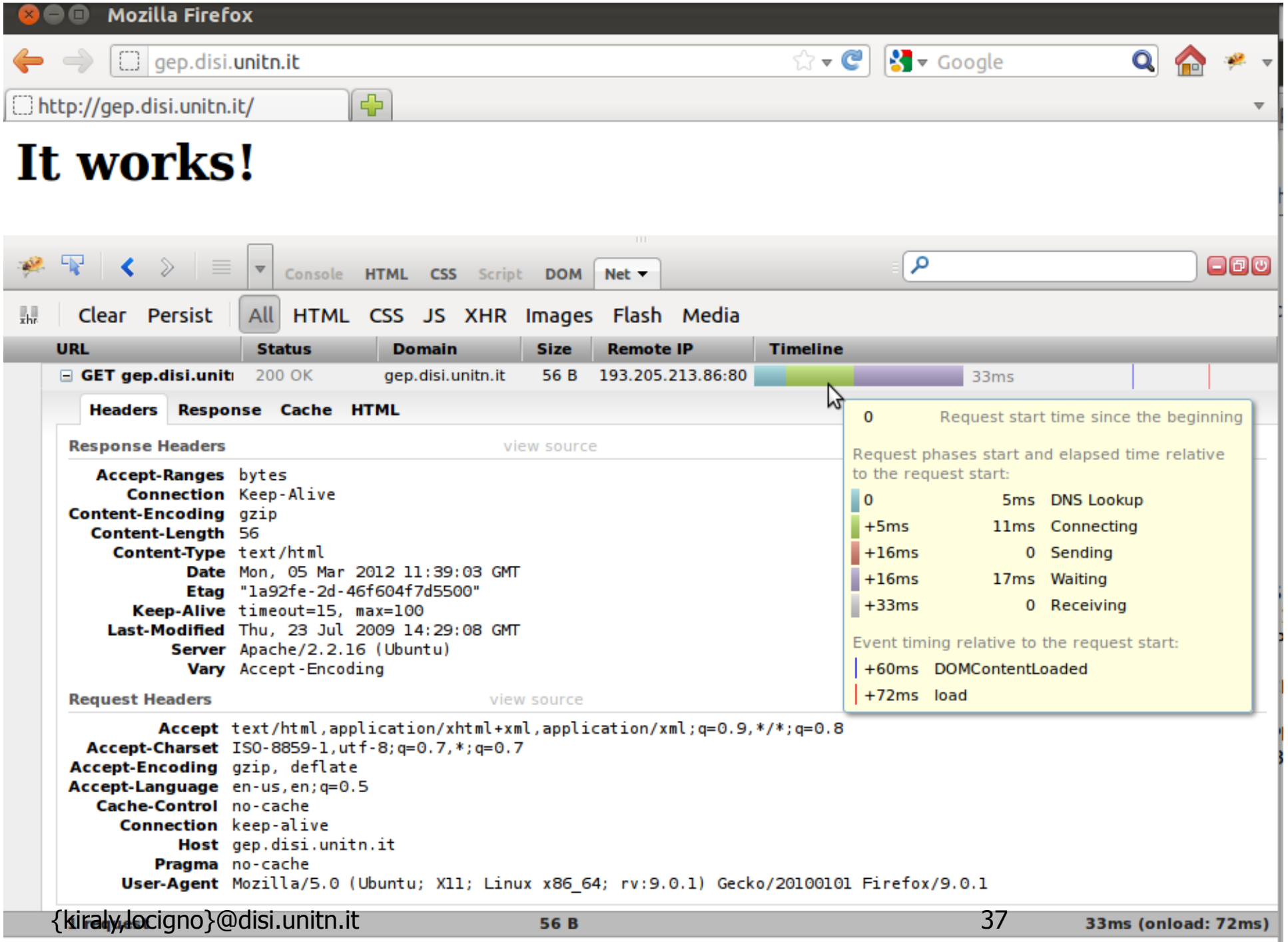
Digitando questo (premete due volte il tasto Invio), trasmettete una richiesta GET minima (ma completa) al server HTTP

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

Bell'esempio ... ma non funziona perchè gli amministratori di rete non consentono queste operazioni per questioni di sicurezza (giustamente!!)




Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://gep.disi.unitn.it/
gep.disi.unitn.it
Google
It works!
Downloads Ctrl+Shift+Y
Add-ons Ctrl+Shift+A
Set Up Sync...
Web Developer
Page Info Ctrl+I
Start Private Browsing Ctrl+Shift+P
Clear Recent History... Ctrl+Shift+Del
Manage Content Plug-ins
Firebug
Web Console Ctrl+Shift+K
Inspect Ctrl+Shift+I
Scratchpad Shift+F4
Page Source Ctrl+U
Error Console Ctrl+Shift+J
Get More Tools
Open Firebug F12
Firebug UI Location
Deactivate Firebug For This Site Shift+F12
Open With Editor
Text Size
Options
Firebug Online
Inspect Element Ctrl+Shift+C
Profile JavaScript
Command Line Ctrl+Shift+L
Search
Customize Shortcuts
About... 1.9.1


Mozilla Firefox
gep.disi.unitn.it
Google
http://gep.disi.unitn.it/
It works!
Console HTML CSS Script DOM Net
Clear Persist All HTML CSS JS XHR Images Flash Media
URL Status Domain Size Remote IP Timeline
GET gep.disi.unitn 200 OK gep.disi.unitn.it 56 B 193.205.213.86:80 33ms
Headers Response Cache HTML
Response Headers view source
Accept-Ranges bytes
Connection Keep-Alive
Content-Encoding gzip
Content-Length 56
Content-Type text/html
Date Mon, 05 Mar 2012 11:39:03 GMT
Etag "1a92fe-2d-46f604f7d5500"
Keep-Alive timeout=15, max=100
Last-Modified Thu, 23 Jul 2009 14:29:08 GMT
Server Apache/2.2.16 (Ubuntu)
Vary Accept-Encoding
Request Headers view source
Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding gzip, deflate
Accept-Language en-us,en;q=0.5
Cache-Control no-cache
Connection keep-alive
Host gep.disi.unitn.it
Pragma no-cache
User-Agent Mozilla/5.0 (Ubuntu; X11; Linux x86_64; rv:9.0.1) Gecko/20100101 Firefox/9.0.1
0 Request start time since the beginning
Request phases start and elapsed time relative to the request start:
0 5ms DNS Lookup
+5ms 11ms Connecting
+16ms 0 Sending
+16ms 17ms Waiting
+33ms 0 Receiving
Event timing relative to the request start:
+60ms DOMContentLoaded
+72ms load
56 B
33ms (onload: 72ms)

Interazione utente-server: i cookie

Molti dei più importanti siti web usano i cookie

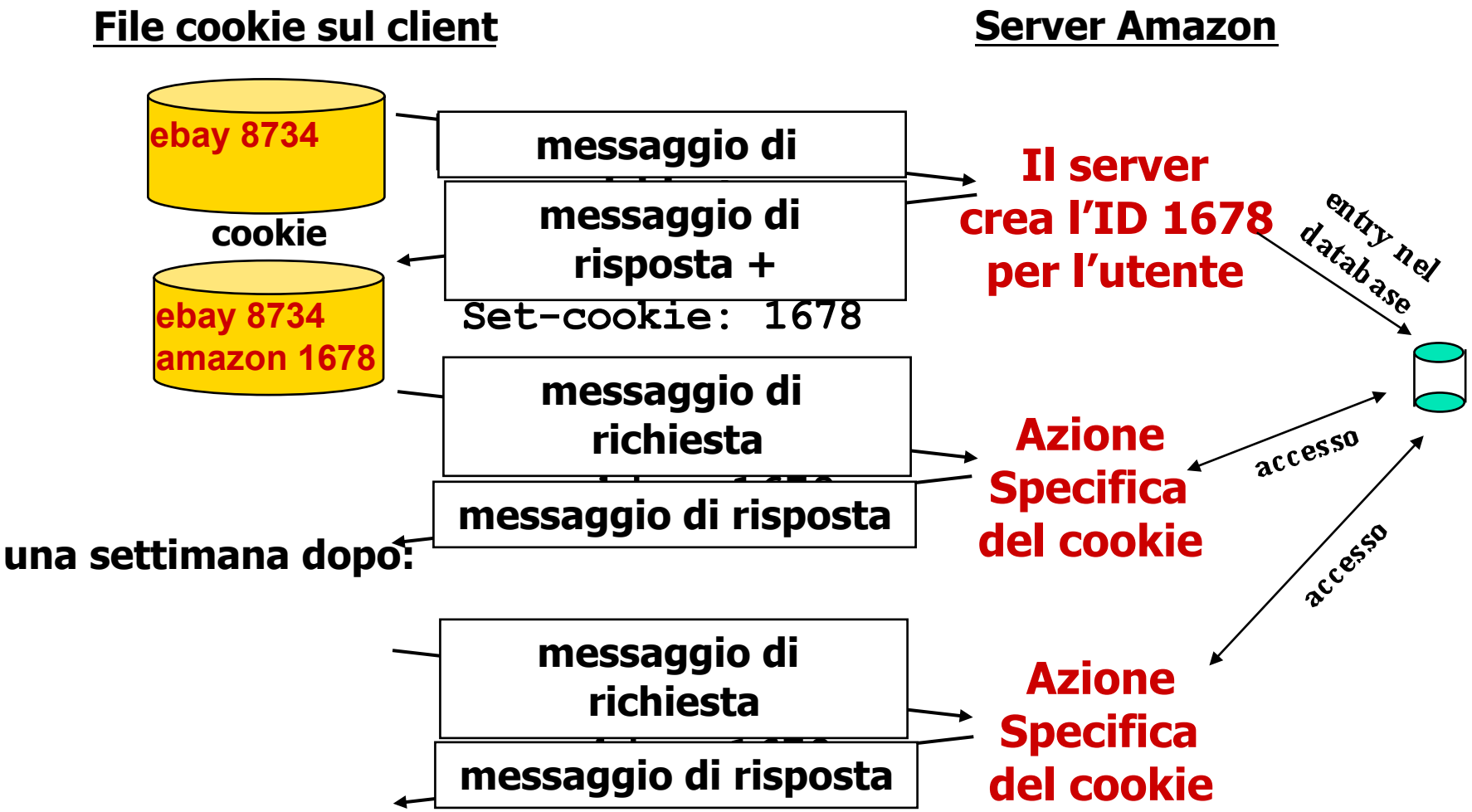
Quattro componenti:

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)

A cosa possono servire i cookie:

- autorizzazione
- carrello elettronico
- suggerimenti
- stato della sessione dell'utente

Lo "stato"

- Mantengono lo stato del mittente e del ricevente per più transazioni
- Livello di sessione utente al di sopra di HTTP privo di stato

nota

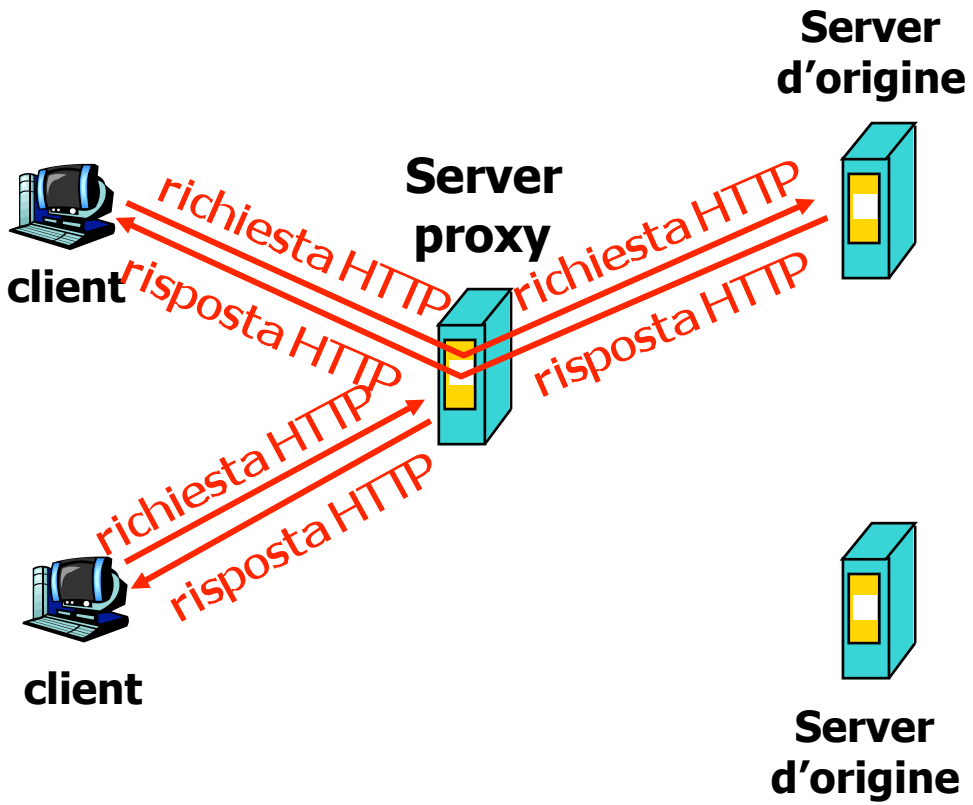
Cookie e privacy:

- **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)

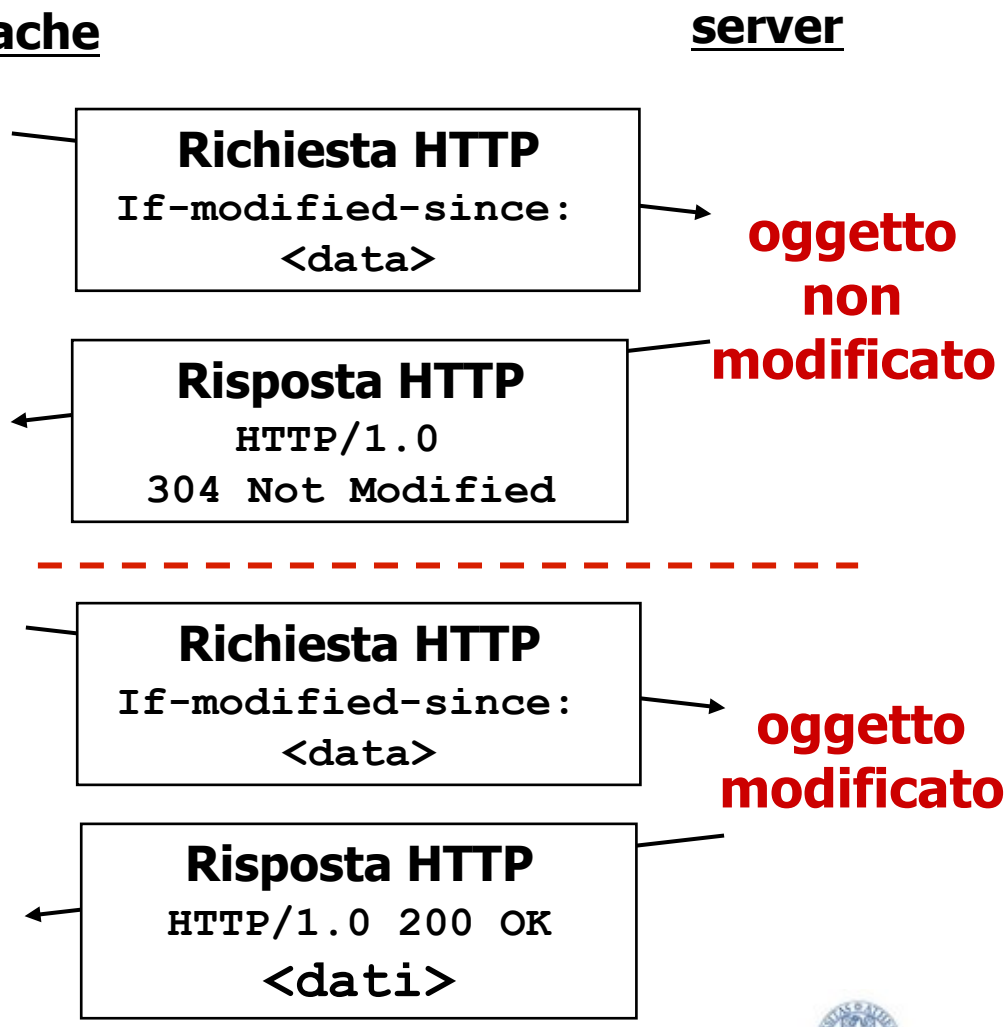
- La cache opera come client e come server
- Tipicamente la cache è installata da un ISP (università, aziende o ISP residenziali)

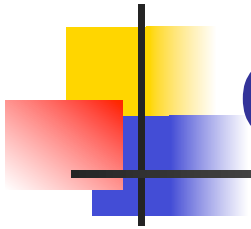
Perché il caching web?

- 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 con bassa ampiezza di banda di fornire dati con efficacia e velocità

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`

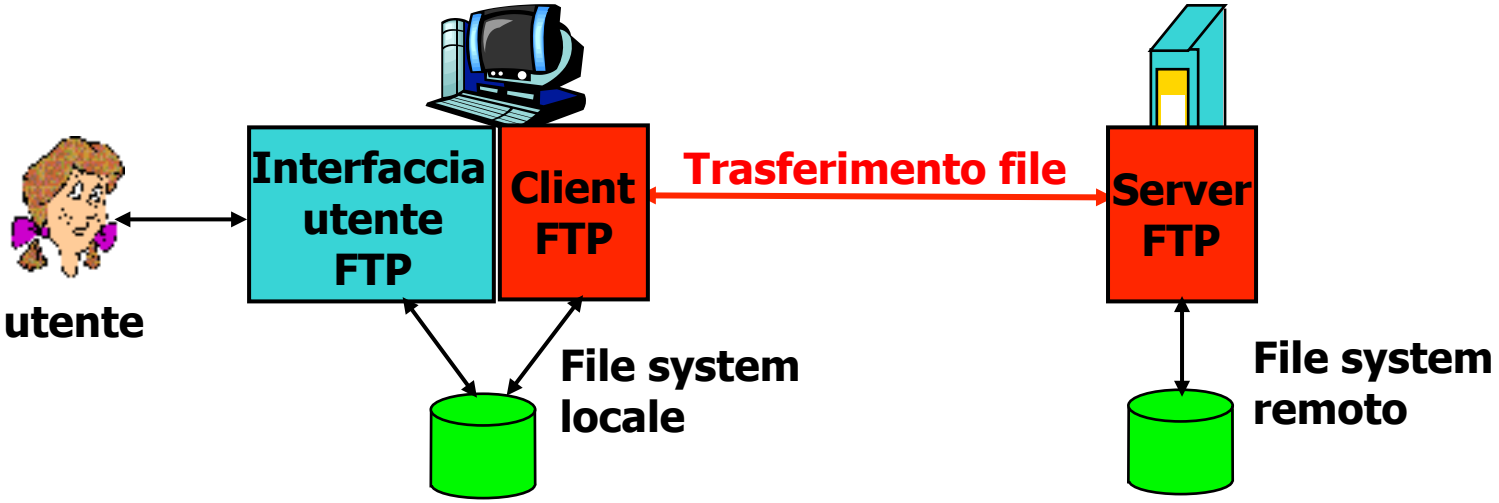




Capitolo 2: Livello di applicazione

- 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

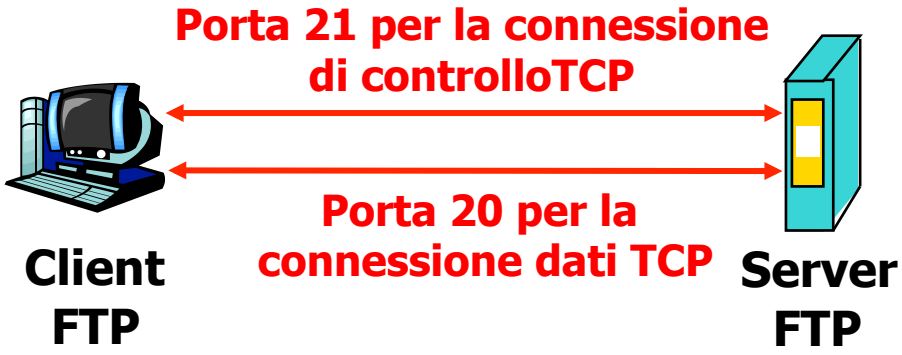
FTP: file transfer protocol



- 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

- 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": associare la connessione di controllo ad un utente e tenere traccia della directory corrente**

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`**