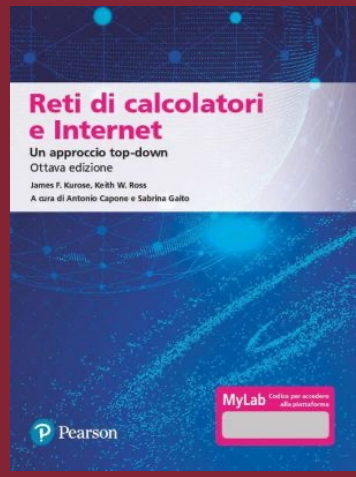
Reti di Elaboratori

Livello di Applicazione, HTTP, cookie



SAPIENZA
UNIVERSITÀ DI ROMA

Alessandro Checco
alessandro.checco@uniroma1.it

Capitolo 2

Livello di applicazione: sommario

- Principi delle applicazioni di rete
- Web e HTTP
- Posta elettronica, SMTP, IMAP
- Domain Name System: DNS
- Applicazioni P2P
- streaming video e content distribution networks
- programmazione socket con UDP e TCP

Livello di applicazione: panoramica

I nostri obiettivi:

- Capire gli aspetti concettuali *e* implementativi dei protocolli a livello applicativo
  - modelli di servizio del livello di trasporto
  - paradigma client-server
  - paradigma peer-to-peer
- esaminare i protocolli più diffusi a livello di applicazione
  - HTTP
  - SMTP, IMAP
  - DNS
- programmare applicazioni di rete
  - socket API

Alcune app di rete

- reti sociali
- Web
- messaggistica testuale
- e-mail
- giochi di rete multiutente
- streaming di video (YouTube, Hulu, Netflix)
- Condivisione file P2P
- voice over IP (es. Skype)
- videoconferenza in tempo reale
- ricerca Internet
- accesso remoto
- …

D: i tuoi preferiti?

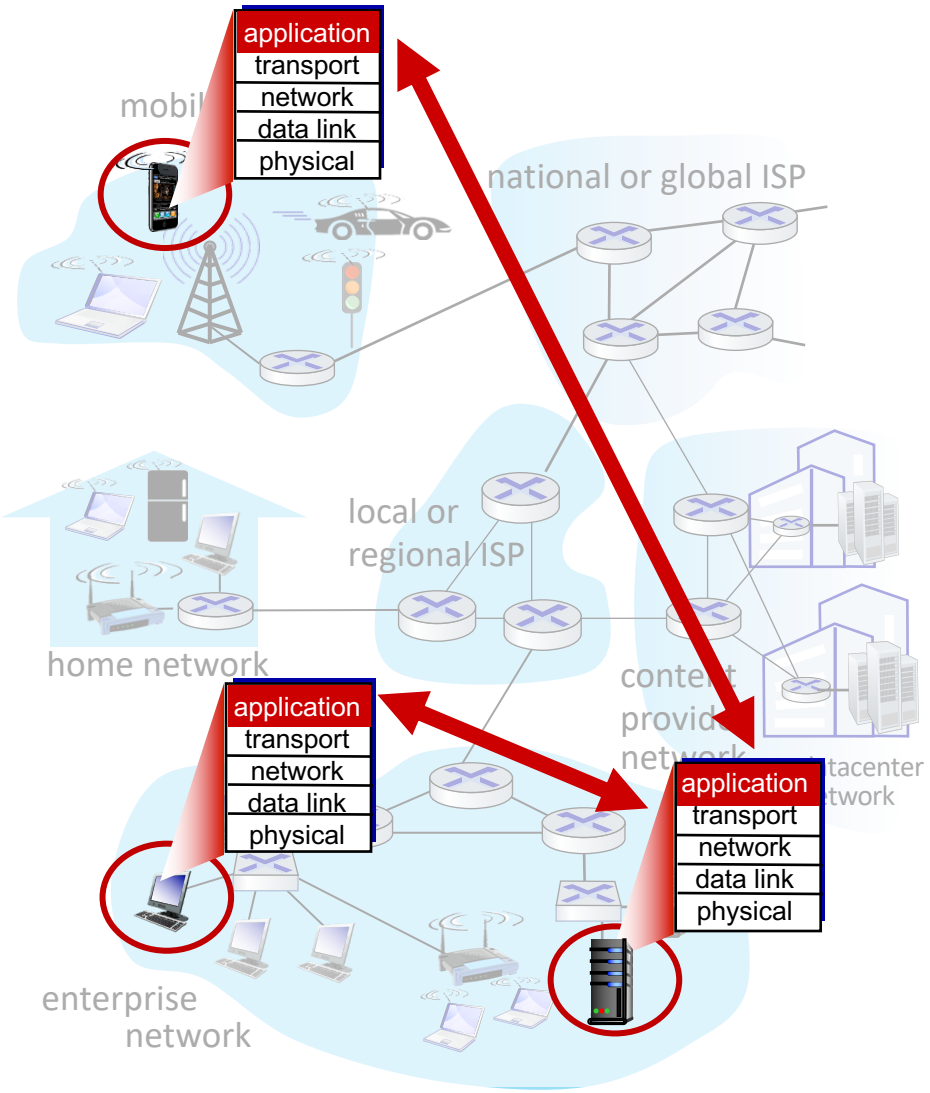
Creazione di una app di rete

scrivere programmi che:

- girano su (diversi) terminali
- comunicano attraverso la rete
- ad esempio, il software del Web server comunica con il software del browser

non è necessario scrivere software per dispositivi core di rete

- i dispositivi core di rete non eseguono le applicazioni
- le applicazioni sui terminali consentono un rapido sviluppo e diffusione delle app



Paradigma client-server

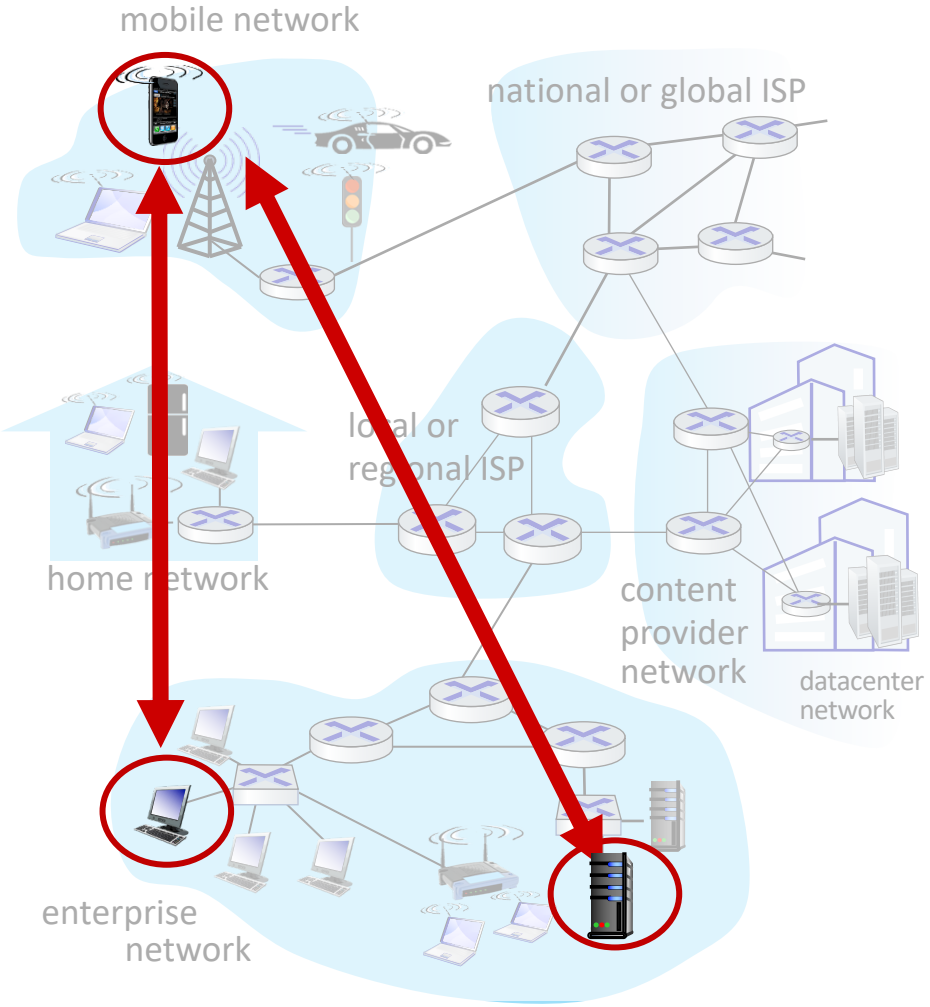
server:

- host sempre attivo
- indirizzo IP permanente
- spesso nei data center, per scalare

clients:

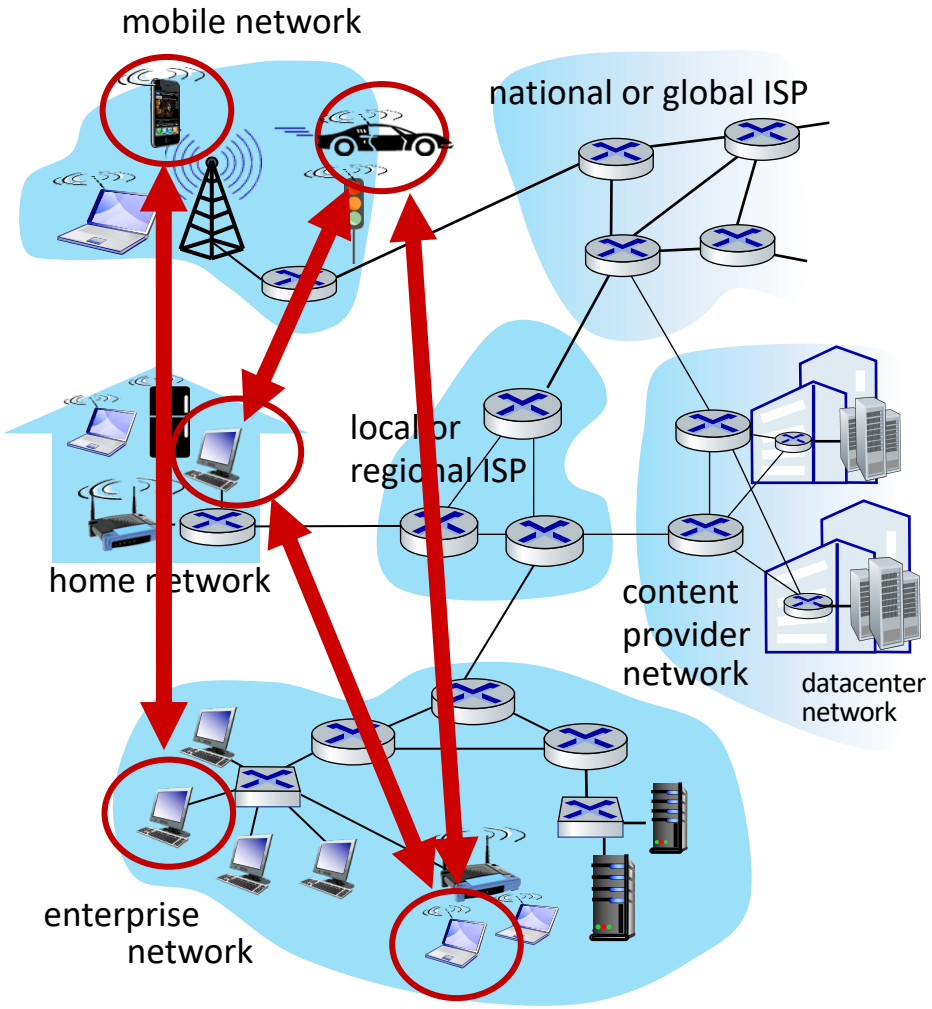
- contatta e comunica con il server
- può essere collegato in modo intermittente
- può avere indirizzi IP dinamici
- ***non comunicano*** direttamente tra loro

- esempi: HTTP, IMAP, FTP



Paradigma peer-to-peer

- *nessun* server è sempre attivo
- qualsiasi peer può comunicare direttamente
- i peer richiedono (e forniscono) servizi da altri peer
  - *autoscalabilità*: nuovi peer incrementano la capacità di servizio, nonché nuove richieste di servizio
- i peer sono connessi in modo intermittente con indirizzi IP variabili
  - gestione complessa
- esempio: condivisione di file P2P



Processi che comunicano fra loro

processo: programma in esecuzione all'interno di un host

- all'interno dello stesso host, due processi comunicano utilizzando la comunicazione inter-process (definita dal sistema operativo)
- i processi in host diversi comunicano scambiandosi messaggi

client, server

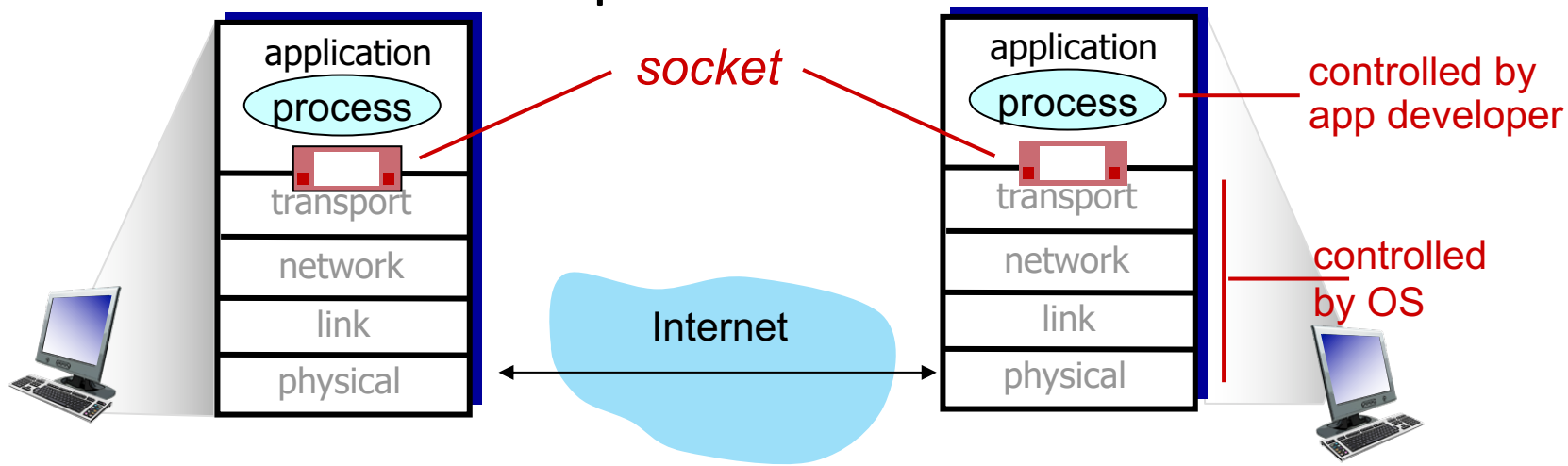
processo cliente: processo che avvia la comunicazione

processo del server: processo che attende di essere contattato

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

Socket

- il processo invia/riceve messaggi al/dal suo socket
- socket analogo a una porta
  - il processo di invio spinge il messaggio fuori dalla porta
  - il processo di invio si basa sull'infrastruttura di trasporto dall'altra parte della porta per consegnare il messaggio al socket del processo del destinatario
  - abbiamo **due socket**: uno per lato



Indirizzamento dei processi

- per ricevere messaggi, il processo deve avere un *identifier*
- dispositivo host ha un indirizzo IP univoco a 32 bit
- *D:* l'indirizzo IP dell'host su cui è in esecuzione il processo è sufficiente per identificare il processo?
  - *R:* no, *molti* processi possono essere eseguiti sullo stesso host
- *identifier* include sia l'indirizzo IP che i numeri di porta associati al processo sull'host.
- esempi di numeri di porta:
  - Server HTTP: 80
  - server di posta: 25
- per inviare un messaggio HTTP al server web gaia.cs.umass.edu:
  - Indirizzo IP: 128.119.245.12
  - numero di porta: 80
- di più a breve…

Un protocollo a livello di applicazione definisce:

- i tipi di messaggi scambiati,
  - ad esempio, richiesta, risposta
- sintassi del messaggio:
  - quali campi nei messaggi e come sono delineati i campi
- semantica del messaggio
  - significato delle informazioni nei campi
- regole per quando e come i processi inviano e rispondono ai messaggi

- protocolli aperti:
  - definito nelle RFC, tutti hanno accesso alla definizione del protocollo
  - consente interoperabilità
  - ad esempio, HTTP, SMTP
- protocolli proprietari:
  - ad esempio Skype

Di quali servizi di trasporto ha bisogno un'app?

integrità dei dati

- alcune app (ad es. trasferimento file, transazioni web) richiedono un trasferimento dati affidabile al 100%.
- altre app (ad es. audio) possono tollerare alcune perdite

garanzie temporali

- alcune app (es. telefonia via Internet, giochi interattivi) richiedono un basso ritardo per essere "efficaci"

throughput

- alcune app (ad es. multimediali) richiedono una quantità minima di throughput per essere "efficaci"
- altre app ("app elastiche") possono utilizzare qualsiasi velocità effettiva che riescono a ottenere

sicurezza

- crittografia, integrità dei dati rispetto a manomissione, …

Di quali servizi di trasporto ha bisogno un'app? Esempi

application	data loss	throughput	time sensitive?
file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's ms
streaming audio/video	loss-tolerant	same as above	yes, few s
interactive games	loss-tolerant	Kbps+	yes, 10's ms
text messaging	no loss	elastic	yes and no

Servizi del protocolli di trasporto Internet

Servizio TCP:

- *trasporto affidabile* tra il processo di invio e quello di ricezione
- *controllo del flusso:* il mittente non ingolfa il destinatario
- *controllo della congestione:* limita il mittente quando la rete è sovraccarica
- *non prevede:* timing, throughput minimo garantito, sicurezza
- *orientato alla connessione:* configurazione richiesta tra i processi client e server

Servizio UDP:

- *trasferimento di dati non affidabile* tra il processo di invio e quello di ricezione
- *non fornisce:* affidabilità, controllo del flusso, controllo della congestione, temporizzazione, garanzia di throughput, sicurezza o stabilimento di connessione

D: A che serve? *Perché* c'è un UDP?

Servizi del protocollo di trasporto Internet

applicazione	protocollo del livello applicazione	protocollo di trasporto
trasferimento di file	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Documenti web	HTTP 1.1 [RFC 7320]	TCP
Telefonia Internet	SIP [RFC 3261], RTP [RFC 3550] o proprietario	TCP o UDP
streaming audio/video	HTTP [RFC 7320], DASH	TCP
giochi interattivi	WOW, FPS (proprietario)	UDP o TCP

Protezione del TCP

Socket TCP e UDP standard:

- nessuna crittografia
- le password in chiaro inviate al socket attraversano Internet in chiaro (!)

Transport Layer Security (TLS)

- fornisce connessioni TCP crittografate
- integrità dei dati
- autenticazione dell'end-point

TLS implementato a livello dell'applicazione

- le app utilizzano librerie TSL, che a loro volta utilizzano TCP

API del socket TLS

- testo in chiaro inviato al socket TLS viene *crittografato*
- vedere il capitolo 8

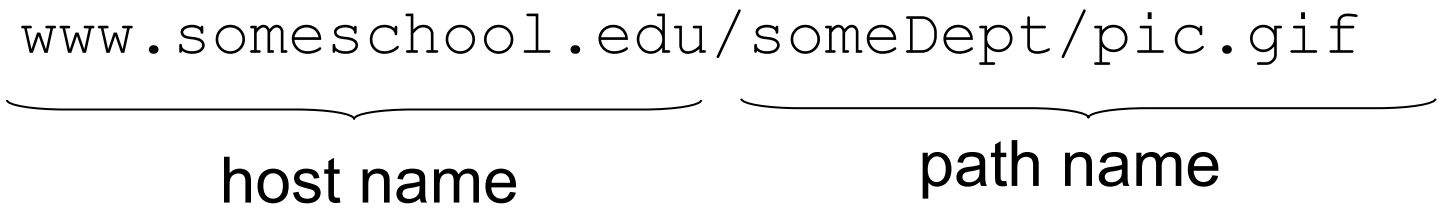
Livello di applicazione: sommario

- Principi delle applicazioni di rete
- Web e HTTP
- Posta elettronica, SMTP, IMAP
- Domain Name System: DNS
- Applicazioni P2P
- streaming video e content distribution networks
- programmazione socket con UDP e TCP

https://www.rfc-editor.org/rfc/rfc2616

Web e HTTP

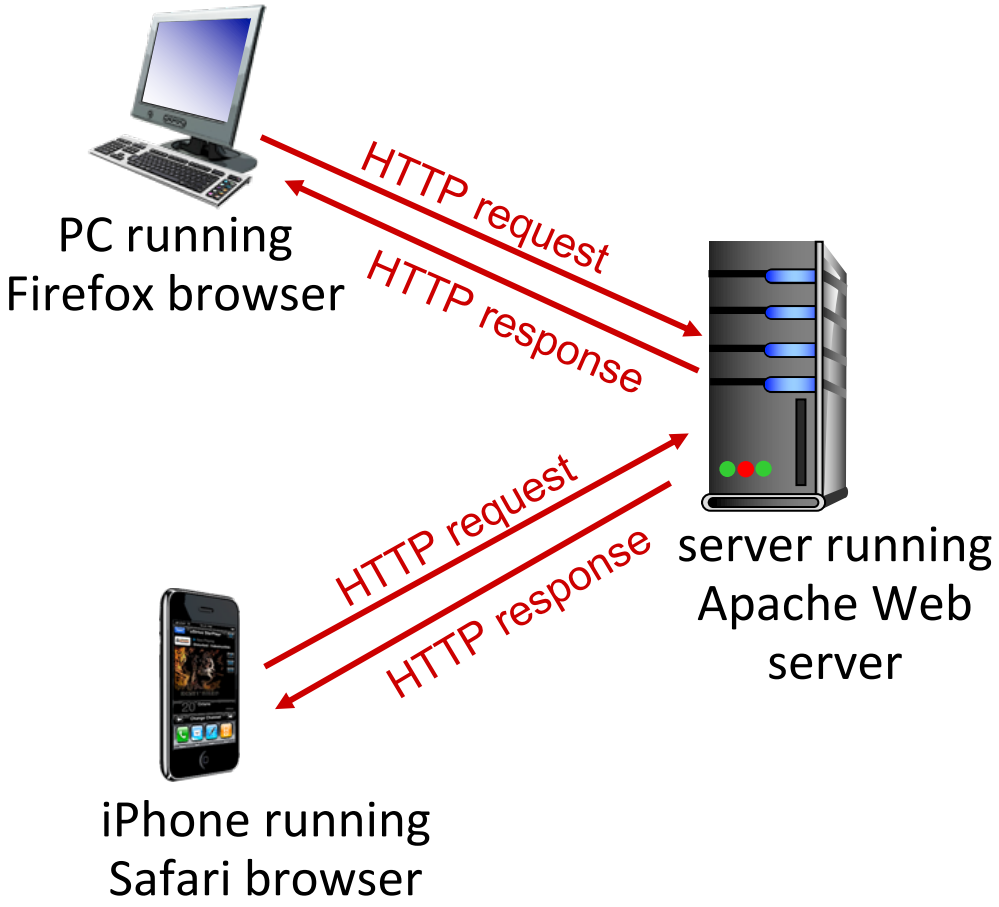
- Una pagina web è composta da *oggetti,* ciascuno dei quali può essere archiviato su un diverso server Web
- l'oggetto può essere un file HTML, un'immagine JPEG, applet Java, file audio,...
- la pagina web consiste *in un file HTML di base* che include *diversi oggetti referenziati, ciascuno* indirizzabile da un *URL,* ad es.



Panoramica HTTP

HTTP: hypertext transfer protocol

- Protocollo a livello di applicazione del Web
- modello client/server:
  - *client:* browser che richiede, riceve (utilizzando il protocollo HTTP) e "visualizza" oggetti Web
  - *server: il server* Web invia (utilizzando il protocollo HTTP) oggetti in risposta alle richieste



Panoramica HTTP (2)

HTTP utilizza TCP:

- il client avvia una connessione TCP con il server, porta 80
- il server accetta la connessione TCP dal client
- Messaggi HTTP (messaggi di protocollo a livello di applicazione) scambiati tra il browser (client HTTP) e il server Web (server HTTP)
- Connessione TCP chiusa

HTTP è stateless

- non conserva alcuna informazione sulle richieste client passate

i protocolli che mantengono lo "stato" sono complessi!

- la storia passata (stato) deve essere mantenuta
- se il server/client si arresta in modo anomalo, i rispettivi punti di vista sullo "stato" potrebbero essere incoerenti e devono essere riconciliati

Connessioni HTTP: due tipi

HTTP non persistente

- Connessione TCP aperta
- al massimo un oggetto inviato su connessione TCP
- Connessione TCP chiusa

il download di più oggetti richiedeva più connessioni

HTTP persistente

- Connessione TCP aperta a un server
- più oggetti possono essere inviati su *una singola* connessione TCP tra client e server
- Connessione TCP chiusa

HTTP non persistente: esempio

L'utente inserisce l'URL: **`www.someSchool.edu/someDepartment/home.index`**
(contenente testo + riferimenti a 10 immagini jpeg)

1a . Il client HTTP avvia la connessione TCP al server HTTP (processo) su www.someSchool.edu sulla porta 80

1b . Server HTTP sull'host www.someSchool.edu in attesa della connessione TCP alla porta 80 "accetta" la connessione, notificando il client

2 . Il client HTTP invia *un messaggio di richiesta HTTP* (contenente l'URL) nel socket di connessione TCP. Il messaggio indica che il client desidera l'oggetto someDepartment /home.index

3 . Il server HTTP riceve il messaggio di richiesta, forma *il messaggio di risposta* contenente l'oggetto richiesto e invia il messaggio nel suo socket

tempo

HTTP non persistente: esempio (2)

L'utente inserisce l'URL: **`www.someSchool.edu/someDepartment/home.index`**
(contenente testo + riferimenti a 10 immagini jpeg)

4. Il server HTTP chiude la connessione TCP

5 . Il client HTTP riceve un messaggio di risposta contenente il file html, visualizza html. Analizzando il file html, trova 10 oggetti jpeg referenziati

6. Passaggi 1-5 ripetuti per ognuno dei 10 oggetti jpeg

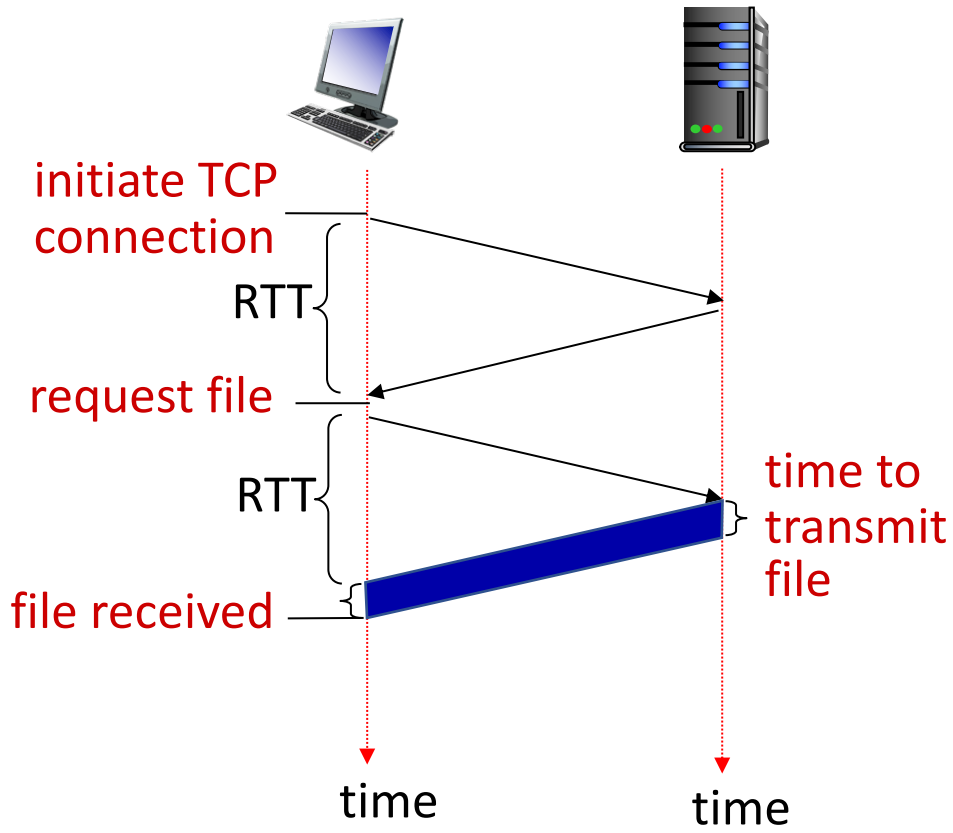
tempo

HTTP non persistente: tempo di risposta

RTT (definizione): tempo di viaggio di un piccolo pacchetto dal client al server e ritorno

Tempo di risposta HTTP (per oggetto):

- un RTT per avviare la connessione TCP
- un RTT per la richiesta HTTP e i primi byte della risposta HTTP da restituire
- oggetto /file



Tempo di risposta HTTP non persistente = 2RTT+ tempo di trasmissione file

HTTP persistente (HTTP 1.1)

Problemi HTTP non persistente:

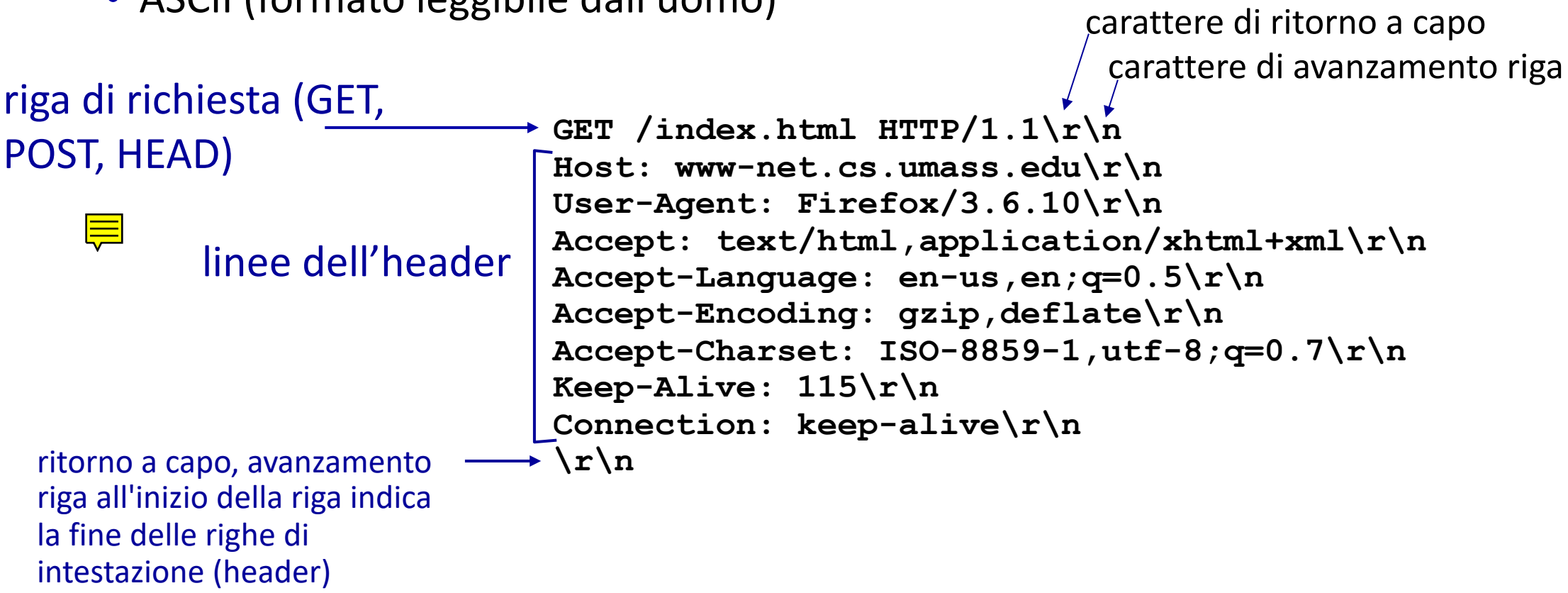
- richiede 2 RTT per oggetto
- Overhead del sistema operativo per *ogni* connessione TCP
- i browser spesso aprono più connessioni TCP parallele per recuperare gli oggetti referenziati in parallelo (ma non risolve il problema di latenza)

HTTP persistente (HTTP1.1):

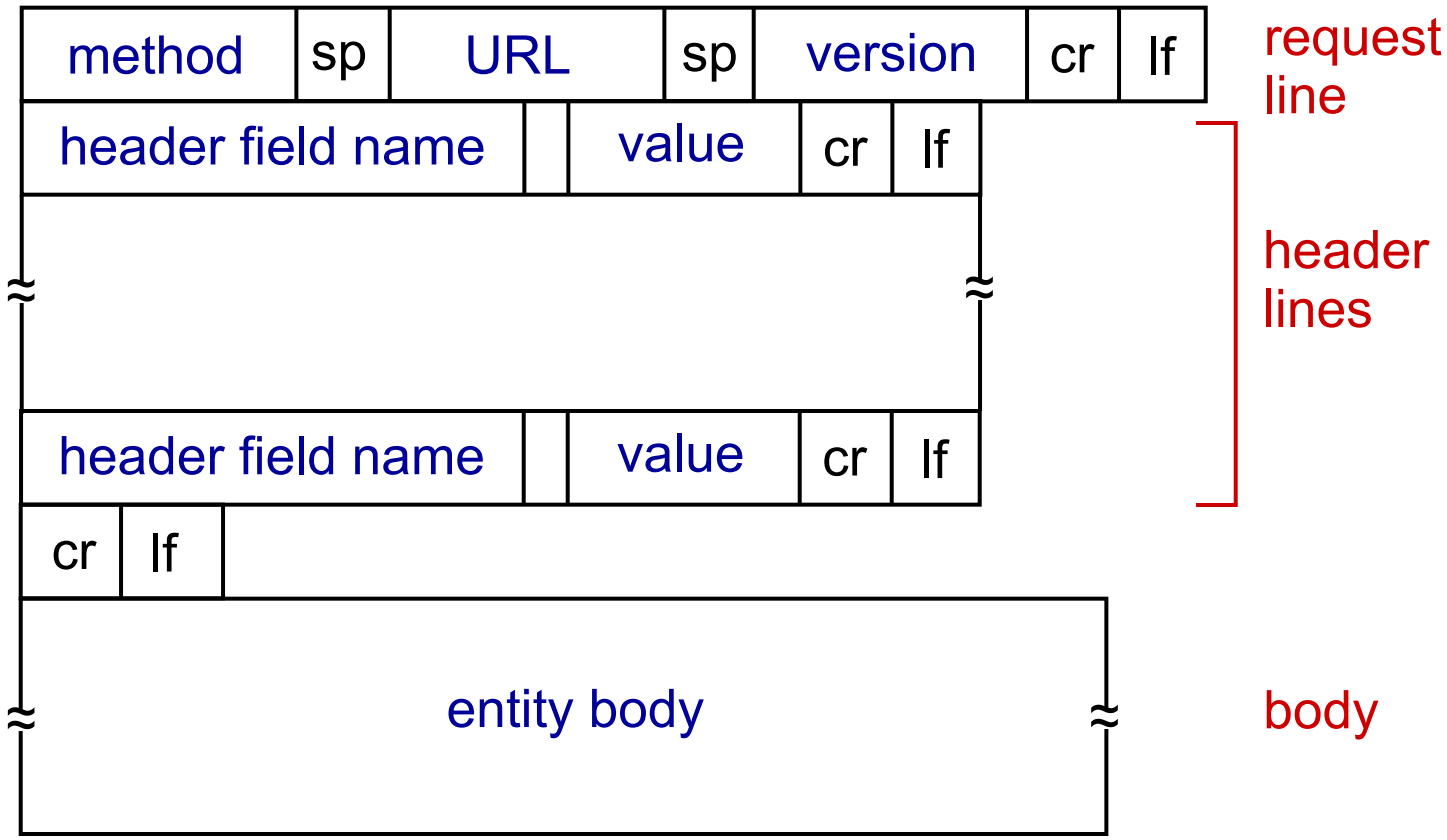
- il server lascia la connessione aperta dopo aver inviato la risposta
- i successivi messaggi HTTP tra lo stesso client/server vengono inviati tramite la connessione aperta
- client invia richieste non appena incontra un oggetto referenziato
- un RTT per tutti gli oggetti referenziati (fino a quasi dimezzare i tempi di risposta)

Messaggio di richiesta HTTP

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



Messaggio di richiesta HTTP: formato generale



Altri messaggi di richiesta HTTP

Metodo POST:

- form in pagine web
- input dell'utente inviato dal client al server nel corpo dell'entità del messaggio di richiesta HTTP POST

Metodo GET con parametri

(per l'invio dei dati al server):

- includi i dati utente nel campo URL del messaggio di richiesta HTTP GET (dopo un '?'):

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

Metodo HEAD:

- richiede (solo) l'header che verrebbe restituito *se l'* URL specificato fosse richiesto con un metodo HTTP GET.

Metodo PUT:

- carica un nuovo file (oggetto) sul server
- sostituisce completamente il file esistente nell'URL specificato con il contenuto nel corpo dell'entità del messaggio di richiesta HTTP POST

Intestazioni nella richiesta

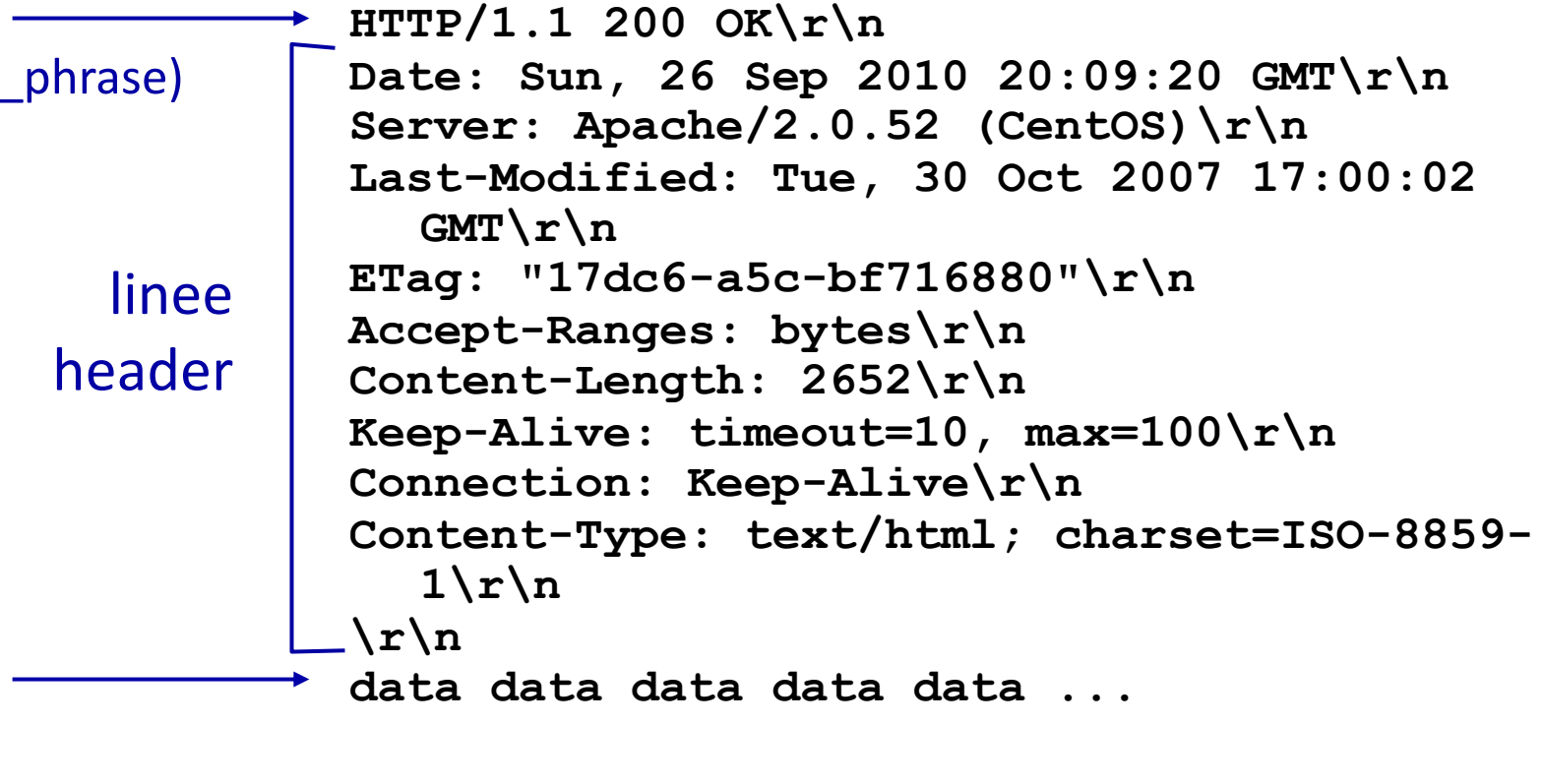
Intestazione	*Descrizione*
User-agent	Indica il programma client utilizzato
Accept	Indica il formato dei contenuti che il client è in grado di accettare
Accept-charset	Famiglia di caratteri che il client è in grado di gestire
Accept-encoding	Schema di codifica supportato dal client
Accept-language	Linguaggio preferito dal client
Authorization	Indica le credenziali possedute dal client
Host	Host e numero di porta del client
Date	Data e ora del messaggio
Upgrade	Specifica il protocollo di comunicazione preferito
Cookie	Comunica il cookie al server (verrà spiegato successivamente)
If-Modified-Since	Invia il documento solo se è più recente della data specificata

Messaggio di risposta HTTP

riga di stato
(protocol status_code status_phrase)

linee
header

dati, ad es.
File HTML



Codici di stato della risposta HTTP

il codice di stato appare nella prima riga nel messaggio di risposta alcuni codici di esempio:

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 di protocollo HTTP

Codici di risposta

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

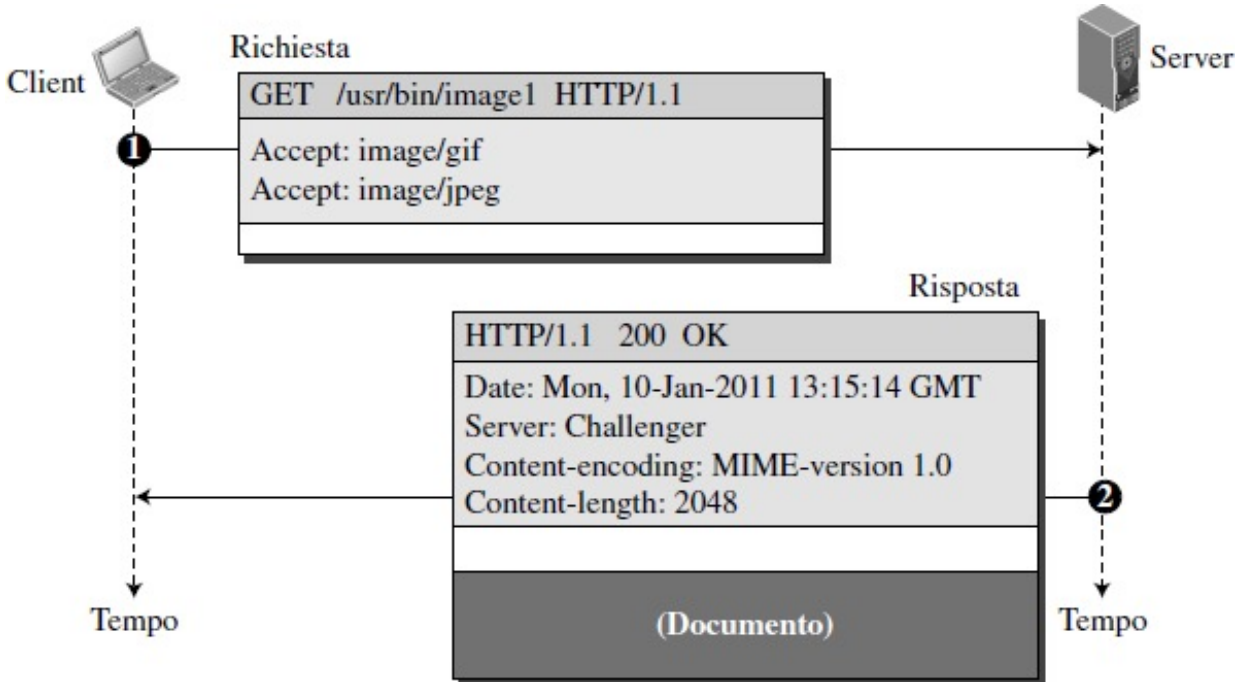
Yes! →

Intestazioni nella risposta

Intestazione	*Descrizione*
Date	Data corrente
Upgrade	Specifica il protocollo preferito
Server	Indica il programma server utilizzato
Set-Cookie	Il server richiede al client di memorizzare un cookie
Content-Encoding	Specifica lo schema di codifica
Content-Language	Specifica la lingua del documento
Content-Length	Indica la lunghezza del documento
Content-Type	Specifica la tipologia di contenuto
Location	Chiede al client di inviare la richiesta a un altro sito
Last-modified	Fornisce data e ora di ultima modifica del documento

Esempio GET

- Il client preleva un documento: viene usato il metodo GET per ottenere l'immagine individuata dal percorso /usr/bin/image1.



- La riga di richiesta contiene il metodo (GET), l'URL e la versione (1.1) del protocollo HTTP. L'intestazione è costituita da due righe in cui si specifica che il client accetta immagini nei formati GIF e JPEG. Il messaggio di richiesta non ha corpo.
- Il messaggio di risposta contiene la riga di stato e quattro righe di intestazione che contengono la data, il server, il metodo di codifica del contenuto (la versione MIME, argomento che verrà descritto nel paragrafo dedicato alla posta elettronica) e la lunghezza del documento.
- Il corpo del messaggio segue l'intestazione.

Provare HTTP (lato client)

1. Telnet al tuo server Web preferito:

```
telnet gaia.cs.umass.edu 80
```

- apre la connessione TCP alla porta 80 (porta del server HTTP predefinita) su gaia.cs.umass.edu
- qualsiasi cosa digitata verrà inviata alla porta 80 su gaia.cs.umass.edu

2. digitare una richiesta HTTP GET:

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

- digitandolo (premendo due volte il ritorno a capo), invii questa richiesta GET minima (ma completa) al server HTTP

3. osserva il messaggio di risposta inviato dal server HTTP!

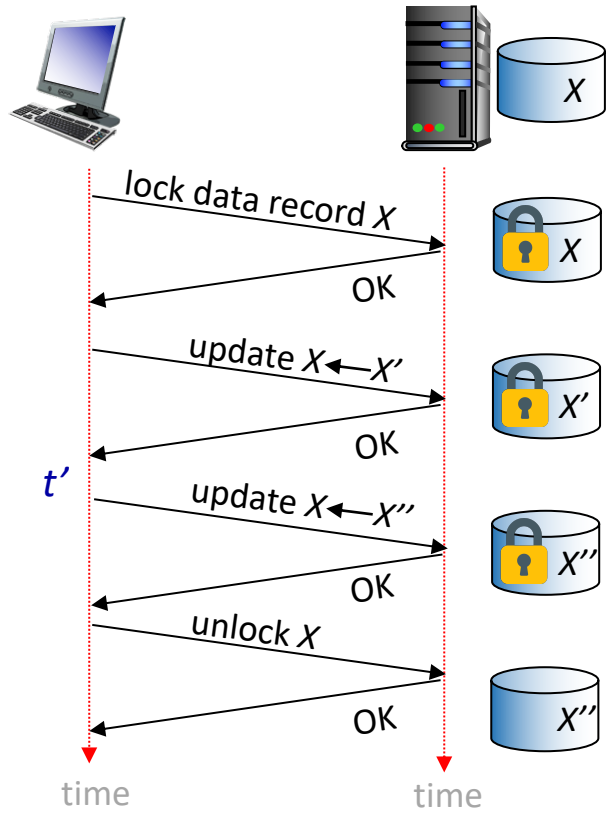
(o usa Wireshark per esaminare la richiesta/risposta HTTP catturata)

Mantenimento dello stato utente/server

l'interazione HTTP GET/risposta è *senza stato*

- nessuna nozione di scambi di messaggi HTTP multi-step per completare una "transazione" Web
  - non è necessario che il client/server tenga traccia dello "stato" dello scambio in più fasi
  - tutte le richieste HTTP sono indipendenti l'una dall'altra
  - non è necessario che il client/server "recuperi" da una transazione parzialmente completataata

un protocollo stateful: il client apporta due modifiche a X o nessuna modifica



D: cosa succede se la connessione di rete o il client si bloccano a *t'* ?

Mantenimento dello stato utente/server: cookie

I siti Web e il browser client utilizzano *i cookie* per mantenere uno stato tra le transazioni

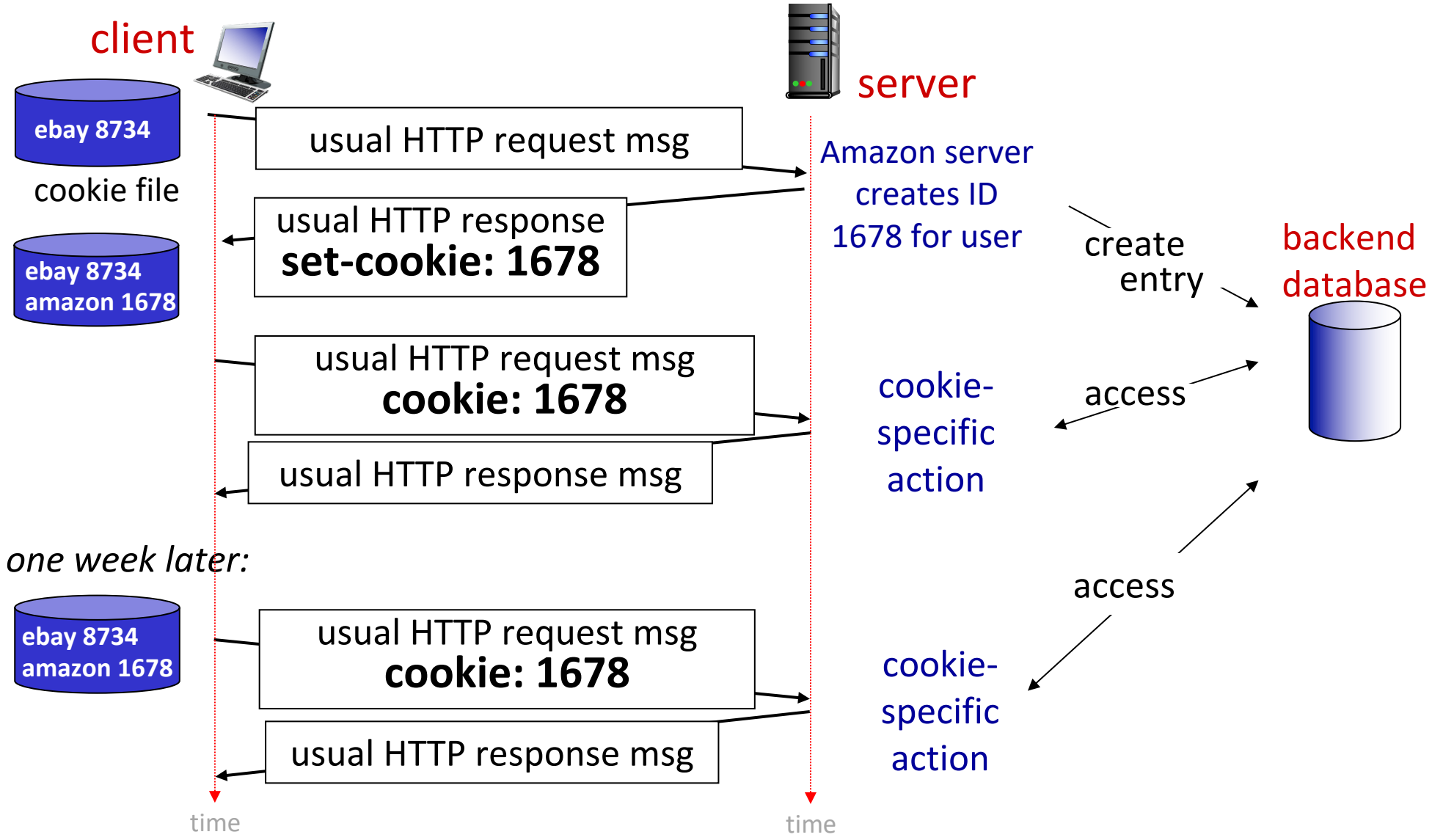
quattro componenti:

1) messaggio di *risposta* HTTP contentente una linea Set-Cookie nell'header
2) messaggio di *richiesta* HTTP include il cookie nell'header
3) file cookie conservato sull'host dell'utente, gestito dal browser dell'utente
4) database back-end sul sito Web

Esempio:

- Susan utilizza il browser sul laptop, visita per la prima volta un sito di e-commerce specifico
- quando le richieste HTTP iniziali arrivano al sito, il sito crea:
  - ID univoco (noto anche come "cookie")
  - voce nel database back-end per l'ID
- le successive richieste HTTP da Susan a questo sito conterranno il valore ID cookie, consentendo al sito di "identificare" Susan

Mantenimento dello stato utente/server: cookie



Durata di un cookie

- Il server chiude una sessione inviando al client una intestazione `Set-Cookie` nel messaggio con

  `Max-Age=0`

- `Max-Age=delta-seconds`

  L'attributo `Max-Age` definisce il tempo di vita in secondi di un cookie. Dopo delta secondi il client dovrebbe rimuovere il cookie. Il valore zero indica che il cookie deve essere rimosso subito.

Cookie HTTP: commenti

A cosa servono i cookie:

- autorizzazione
- carrelli della spesa
- raccomandazioni
- stato della sessione utente (e-mail web)

Risolve il problema di mantenere lo stato in un protocollo *stateless*

- problema: mantenere lo stato al mittente/destinatario su più transazioni
- cookie: i messaggi HTTP trasportano lo stato
- Altre soluzioni: richieste POST o GET con parametri per trasportare lo stato

cookie e privacy:

- i cookie consentono ai siti di *imparare* molto sul client.
- i cookie persistenti di terze parti (cookie di tracciamento) consentono di tracciare un utente (valore del cookie) su più siti web