

Il protocollo HTTP

Valeria Cardellini
Università di Roma Tor Vergata

Caratteristiche del protocollo HTTP

- Scambio di messaggi di richiesta e risposta
 - Transazione HTTP o Web
- Protocollo stateless
- Basato sul meccanismo di naming degli URI per identificare le risorse Web
- Metadati sulla risorsa
 - Informazioni *sulla* risorsa (ma *non* parte della risorsa) incluse nei trasferimenti; ad esempio:
 - Dimensione della risorsa
 - Tipo della risorsa (ad es. text/html)
 - MIME per classificare il formato dei dati
 - Data dell'ultima modifica della risorsa

Versioni del protocollo

- HTTP/1.0
 - Versione (quasi) definitiva nel 1996
 - Riferimento: RFC1945 (HTTP/1.0)
 - In precedenza HTTP/0.9 (implementato nel 1990, descritto nel 1992)
- HTTP/1.1
 - Versione (quasi) definitiva nel 1999
 - Riferimento: RFC2616 (HTTP/1.1)

Messaggi HTTP

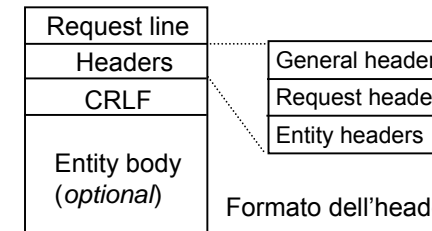
- Due tipologie di messaggi:
 - messaggi di **richiesta** HTTP
 - messaggi di **risposta** HTTP
- Messaggi composti da:
 - Header o intestazione
 - In formato ASCII (leggibile dagli esseri umani)
 - Corpo (*opzionale*)

Richiesta HTTP

- Una *richiesta HTTP* comprende
 - Metodo
 - URL
 - Identificativo della versione del protocollo HTTP
 - Ulteriori informazioni aggiuntive
- Il metodo specifica il tipo di operazione che il client richiede al server
 - GET è il metodo usato più frequentemente: serve per acquisire una risorsa Web
- URL identifica la risorsa *locale* rispetto al server
- Informazioni aggiuntive, quali:
 - la data e l'ora di generazione della richiesta
 - il tipo di software utilizzato dal client (user agent)
 - i tipi di dato che il browser è in grado di visualizzare
- ... per un totale di oltre 50 tipi di informazioni differenti

Messaggio di richiesta HTTP

- Formato generale



Formato dell'header: <header name>: <value>

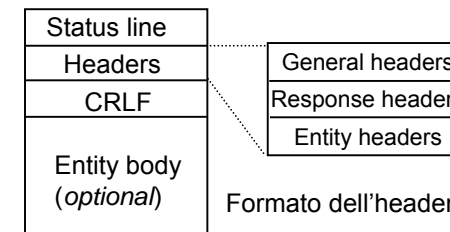
Linea di richiesta → GET /somedir/page.html HTTP/1.1
Intestazioni (generali, di richiesta, di entità) → Connection: close
User-Agent: Mozilla/4.0
Host: www.foo.com
Accept: text/html, image/gif, image/jpeg
Accept-Language: en
Linea vuota (Carriage return, line feed) → CRLF

Risposta HTTP

- Una *risposta HTTP* comprende
 - l'identificativo della versione del protocollo HTTP
 - il codice di stato e l'informazione di stato in forma testuale
 - un insieme di possibili altre informazioni riguardanti la risposta
 - l'eventuale contenuto della risorsa richiesta
- Se la pagina Web richiesta dall'utente è composta da molteplici risorse, ciascuna di esse sarà identificata da un URL differente: **è necessario che il browser invii un esplicito messaggio di richiesta per ognuna delle risorse incorporate nella pagina**

Messaggio di risposta HTTP

- Formato generale



Formato dell'header: <header name>: <value>

Linea di stato → HTTP/1.1 200 OK
Intestazioni (generali, di risposta, di entità) → 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
Carriage return, line feed → CRLF
Corpo del messaggio (es. file HTML richiesto) → data data data data ...

I metodi della richiesta

Metodo	Versione HTTP	Modifiche in HTTP/1.1
GET	1.0 e 1.1	Richiesta di parti di entità
HEAD	1.0 e 1.1	Uso di header di richiesta condizionali
POST	1.0 e 1.1	Gestione di connessione, trasmissione di messaggio
PUT	1.1 (1.0 non standard)	
DELETE	1.1 (1.0 non standard)	
OPTIONS	1.1	Estensibilità
TRACE	1.1	Estensibilità
CONNECT	1.1	Uso futuro

I metodi della richiesta (2)

- Analizziamo i metodi seguenti:
 - GET
 - POST
 - HEAD
 - PUT
- Cenni sui metodi
 - DELETE
 - OPTIONS
 - TRACE
 - CONNECT
- Un metodo HTTP può essere:
 - **sicuro**: non altera lo stato della risorsa
 - **idempotente**: l'effetto di più richieste identiche è lo stesso di quello di una sola richiesta

Il metodo GET

- E' il più importante (ed unico in v. 0.9) metodo di HTTP
- Richiede una risorsa ad un server (no corpo della richiesta)
- E' il metodo più frequente
 - Ad es., attivato cliccando sul link di un documento HTML o specificando un URL nell'apposito campo del browser
- GET per risorsa statica
GET /foo.html
- GET per risorsa dinamica (es. risorsa generata con CGI)
GET /cgi-bin/query?q=foo
- E' sicuro ed idempotente
- Può essere:
 - **Assoluto**: normalmente, ossia quando la risorsa viene richiesta senza altre specificazioni
 - **Condizionale**: se la risorsa corrisponde ad un criterio indicato negli header If-Match, If-Modified-Since, If-Range, ...
 - **Parziale**: se la risorsa richiesta è una sottoparte di una risorsa memorizzata

Il metodo HEAD

- Variante di GET usata principalmente per scopi di controllo e debugging
- Il server deve rispondere soltanto con i metadati associati alla risorsa richiesta (solo header) senza il corpo della risorsa
- E' sicuro ed idempotente
- Usato per verificare:
 - **Validità di un URI**: la risorsa esiste ed è di lunghezza non nulla
 - **Accessibilità di un URI**: la risorsa è accessibile presso il server e non sono richieste procedure di autenticazione
 - **Coerenza di cache di un URI**: la risorsa non è stata modificata rispetto a quella in cache, non ha cambiato lunghezza, valore hash o data di modifica

Il metodo POST

- Permette di trasmettere delle informazioni dal client al server
 - Aggiornare una risorsa esistente o fornire dati di ingresso
 - Dati forniti nel corpo della richiesta (GET: dati codificati nell'URI di richiesta)
 - Ad es. usato per esempio per sottomettere i dati di un form HTML ad un'applicazione sul server identificata dall'URI specificata nella richiesta
- Non è né sicuro, né idempotente
- Il server può rispondere positivamente in tre modi:
 - 200 Ok: dati ricevuti e sottomessi alla risorsa specificata; è stata data risposta
 - 201 Created: dati ricevuti, la risorsa non esisteva ed è stata creata
 - 204 No content: dati ricevuti e sottomessi alla risorsa specificata; non è stata data risposta

Il metodo PUT

- Serve per trasmettere delle informazioni dal client al server, creando o sostituendo la risorsa specificata
- Differenza tra PUT e POST:
 - l'URI di POST identifica la risorsa che gestirà l'informazione inviata nel corpo della richiesta
 - l'URI di PUT identifica la risorsa inviata nel corpo della richiesta: è la risorsa che ci si aspetta di ottenere facendo un GET in seguito con lo stesso URI
- Non è sicuro, ma è idempotente
- Non offre nessuna garanzia di controllo degli accessi o locking
 - Estensione WebDAV (Web-based Distributed Authoring and Versioning) del protocollo HTTP: fornisce (tra le altre cose) una semantica sicura e collaborativa per il metodo PUT
 - Riferimento: RFC 2518

Altri metodi

- DELETE: per cancellare la risorsa specificata dall'URI
- OPTIONS: permette al client di conoscere le capacità del server (ad es., i metodi supportati)
- TRACE: permette al client di conoscere il contenuto del messaggio di richiesta effettivamente ricevuto dal server
 - Loopback di livello applicativo
- CONNECT: per usi futuri

Header HTTP

- Gli header sono righe testuali free-format che specificano caratteristiche
 - generali della trasmissione (**header generali**)
 - dell'entità trasmessa (**header dell'entità**)
 - della richiesta effettuata (**header della richiesta**)
 - della risposta generata (**header della risposta**)
- Formato dell'header:
`<header name>: <value>`

Header HTTP (2)

- Header generali
 - Si applicano solo al messaggio trasmesso e si applicano sia ad una richiesta che ad una risposta, ma non necessariamente alla risorsa trasmessa
 - Ad es., **Date**: data ed ora della trasmissione
 - RFC 1123 per il formato della data (possibili anche altri formati)
 - Ad es., **Pragma**: **no-cache**: risposta direttamente dall'origin server (no copia in cache di qualche proxy)
- Header dell'entità
 - Forniscono informazioni sul corpo del messaggio, o, se non vi è corpo, sulla risorsa specificata
 - Ad es., **Content-Type**: il tipo MIME dell'entità acclusa
 - Header obbligatorio in ogni messaggio che abbia un corpo
 - Ad es., **Content-Length**: la lunghezza in byte del corpo

Header HTTP (3)

- Header della richiesta
 - Impostati dal client per specificare informazioni sulla richiesta e su se stesso al server
 - Ad es., **User-Agent**: stringa che descrive il client che origina la richiesta; tipicamente: tipo, versione e sistema operativo del client
 - Ad es., **Referer**: l'URL di provenienza (utile per user profiling e debugging)
- Header della risposta
 - Impostati dal server per specificare informazioni sulla risposta e su se stesso al client
 - Ad es., **Server**: stringa che descrive il server che origina la risposta; tipicamente: tipo, versione e sistema operativo del server

Header di richiesta in HTTP/1.0

- Cinque header di richiesta in HTTP/1.0
- From
 - Es.: **From**: gorby@moskvar.com
- User-Agent
 - Es.: **User-Agent**: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
- If-Modified-Since
 - Es.: **If-Modified-Since**: Thu, 01 Apr 2002 16:00:00 GMT
- Referer
 - Es.: **Referer**: http://www.ce.uniroma2.it/index.html
- Authorization
 - Es.: **Authorization**: Basic QWxhZGRpbjpvYVUyIHNlc2FtZQ==
Codifica in base 64 di userid "Alladin" e password "open sesame"

Header di risposta in HTTP/1.0

- Tre header di risposta in HTTP/1.0
- Server
 - Es.: **Server**: Apache 1.3.20
- Location
 - Es.: **Location**: http://www.uniroma2.it/newindex.html
- WWW-Authenticate
 - Es.: **WWW-Authenticate**: Basic realm="Area Privata"

Header di entità in HTTP/1.0

- Sei header di entità in HTTP/1.0
- Content-Type
 - Es.: `Content-Type: text/html`
- Content-Length
 - Es.: `Content-Length: 650`
 - Tipicamente omissso in risposte contenenti risorse generate dinamicamente
- Content-Encoding
 - Tipo di compressione applicato alla risorsa
 - Es.: `Content-Encoding: gzip`
- Allow
 - Metodi supportati dalla risorsa specificata dall'URI
 - Es.: `Allow: GET`
- Last-Modified
 - Es.: `Last-Modified: Thu, 01 Apr 2002 18:16:45 GMT`
- Expires
 - Es.: `Expires: Fri, 02 Apr 2002 21:00:00 GMT`

Codice di stato della risposta

- E' un numero di tre cifre, di cui la prima indica la classe della risposta e le altre due la risposta specifica
- Esistono le seguenti classi:
 - **1xx: Informational** una risposta temporanea alla richiesta, durante il suo svolgimento
 - **2xx: Successful** il server ha ricevuto, capito e accettato la richiesta
 - **3xx: Redirection** il server ha ricevuto e capito la richiesta, ma possono essere necessarie altre azioni da parte del client per portare a termine la richiesta
 - **4xx: Client error** la richiesta del client non può essere soddisfatta per un errore da parte del client (errore sintattico o richiesta non autorizzata)
 - **5xx: Server error** la richiesta può anche essere corretta, ma il server non è in grado di soddisfare la richiesta per un problema interno (suo o di applicazioni invocate per generare dinamicamente risorse)

Alcuni codici di stato della risposta

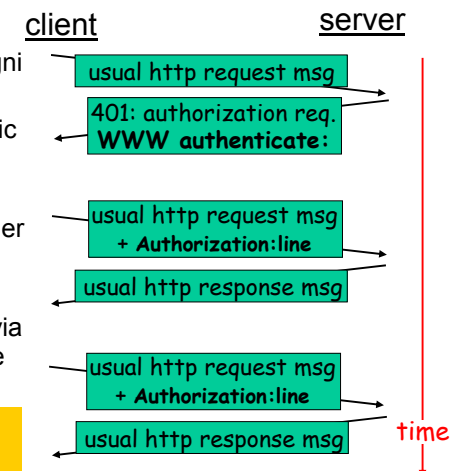
- 200 OK
 - Risorsa nel corpo del messaggio
- 301 Moved Permanently
 - Redirezione: risorsa spostata
- 304 Not Modified
 - Risorsa non modificata
- 401 Unauthorized
 - La risorsa richiede autenticazione dell'utente
- 403 Forbidden
 - Accesso vietato
- 404 Not Found
 - Risorsa non esistente
- 500 Internal Server Error
 - Errore non atteso che impedisce il servizio richiesto
- 501 Not Implemented
 - Server non supporta la funzionalità richiesta (es. metodo non implementato)

Autenticazione in HTTP/1.0

Scopo dell'autenticazione: controllare gli accessi alle risorse del server

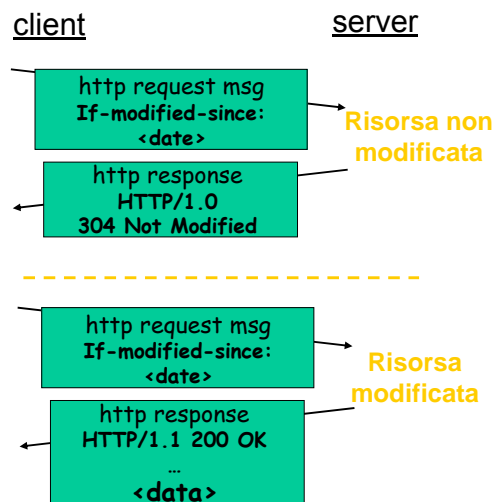
- HTTP stateless: il client deve presentare l'autorizzazione in ogni richiesta
- Autorizzazione: solitamente Basic (userid e password codificati in base64)
 - Autorizzazione: linea di header nella richiesta
 - Se non c'è autorizzazione, il server rifiuta l'accesso ed invia `WWW authenticate:` come linea di header nella risposta

Attenzione: login e password viaggiano in chiaro (non criptate!)



GET condizionale in HTTP/1.0

- Scopo: non inviare un oggetto se il client ha una versione aggiornata già memorizzata nella sua cache
- Il client specifica la data della copia nella cache nella richiesta HTTP
If-modified-since: <data>
- Il server invia una risposta che non contiene l'oggetto se la copia posseduta dal client è aggiornata:
HTTP/1.0 304 Not Modified



Principali problemi di HTTP/1.0

- Lentezza e congestione nelle connessioni → connessioni multiple (“hack”)
- Limitatezza nel numero di indirizzi IP
 - Un indirizzo IP per ciascun server Web
- Limiti del meccanismo di autenticazione
 - Password in chiaro
- Limiti nel controllo dei meccanismi di caching

Principali novità in HTTP/1.1

- Meccanismo hop-by-hop
- Connessioni persistenti e pipelining
- Hosting virtuale
- Autenticazione crittografata (“digest”)
- Nuovi metodi di accesso
 - Aggiornamento delle risorse sul server e diagnostica
- Miglioramento dei meccanismi di caching
 - Gestione molto più sofisticata delle cache
 - Maggiore accuratezza nella specifica di validità
 - Header Cache-Control per direttive di caching
- Chunked encoding
 - La risposta può essere inviata al client suddivisa in sottoparti, anche prima di conoscerne la dimensione totale

Header in HTTP/1.1

- Header generali
 - Date
 - Pragma
 - Cache-Control
 - Connection
 - Trailer
 - Transfer-Encoding
 - Upgrade
 - Via
 - Warning
- Header di entità
 - Allow
 - Content-Encoding
 - Content-Length
 - Content-Type
 - Expires
 - Last-Modified
 - Content-Language
 - Content-Location
 - Content-MD5
 - Content-Range

In rosso le novità dell'HTTP/1.1

Header in HTTP/1.1 (2)

- Header di richiesta
 - Authorization
 - From
 - Referer
 - User-Agent
 - If-Modified-Since
 - Accept
 - Accept-Charset
 - Accept-Encoding
 - Accept-Language
 - TE
 - Proxy-Authorization
 - If-Match
 - If-None-Match
 - If-Range
 - If-Unmodified-Since
 - Expect
 - Host
 - Max-Forwards
 - Range
 - Header di risposta
 - Location
 - Server
 - WWW-Authenticate
 - Proxy-Authenticate
 - Retry-After
 - Accept-Ranges
 - Age
 - ETag
 - Vary
- Tool per sniffing degli header HTTP:
- LiveHTTPHeaders per Mozilla
<http://livehttpheaders.mozdev.org/>
 - Web developer toolbar per Opera
<http://operawiki.info/WebDevToolbar>
 - Fiddler (*HTTP debugging proxy*)
<http://www.fiddlertool.com/fiddler/>
 - <http://web-sniffer.net/>

Connessioni in HTTP/1.0

- La transazione HTTP è composta da uno scambio di richiesta e risposta
- Il client apre una connessione TCP separata per ogni risorsa richiesta
 - La connessione TCP è *non persistente*
- Uso di una connessione TCP in HTTP/1.0
 - Il client apre una connessione TCP con il server
 - Il client invia il messaggio di richiesta HTTP sulla connessione
 - Il server invia il messaggio di risposta HTTP sulla connessione
 - La connessione TCP viene chiusa
- Vantaggio
 - Concorrenza massima

Problemi delle connessioni non persistenti

- Overhead per l'instaurazione e l'abbattimento della connessione TCP
 - Per ogni trasferimento di risorsa, 2 round trip time (RTT) in più
 - Ad es., per una risorsa Web che richiede 10 pacchetti: necessari 17 pacchetti, di cui 7 di overhead
 - Overhead sul SO per allocare risorse per ogni connessione
- Overhead per il meccanismo slow start del TCP
 - La finestra ha dimensione pari ad 1 all'inizio di ogni nuova connessione TCP
- Tre approcci proposti per risolvere in problema
 - Connessioni TCP persistenti (HTTP/1.1)
 - Header `Connection: Keep-Alive` in HTTP/1.0 (no standard)
 - Introduzione di nuovi metodi per ottenere risorse multiple sulla stessa connessione (solo proposta)
 - Uso di connessioni multiple in parallelo

Connessioni multiple (in parallelo)

- Soluzione *parziale* ai problemi delle connessioni non persistenti attuata da alcuni browser
- Connessioni TCP **in parallelo**: apertura simultanea di più connessioni TCP, una per ogni risorsa richiesta
- Vantaggio (apparente)
 - Riduzione del tempo di latenza percepito dall'utente
- Problemi delle connessioni parallele
 - Aumento della congestione sulla rete
 - Il server serve un minor numero di client diversi contemporaneamente
 - Richieste di pagine interrotte dall'utente
 - Non è garantita la riduzione della latenza: ogni richiesta è indipendente dalle altre
- RFC 2616: un client non dovrebbe aprire più di 2 connessioni parallele verso lo stesso server

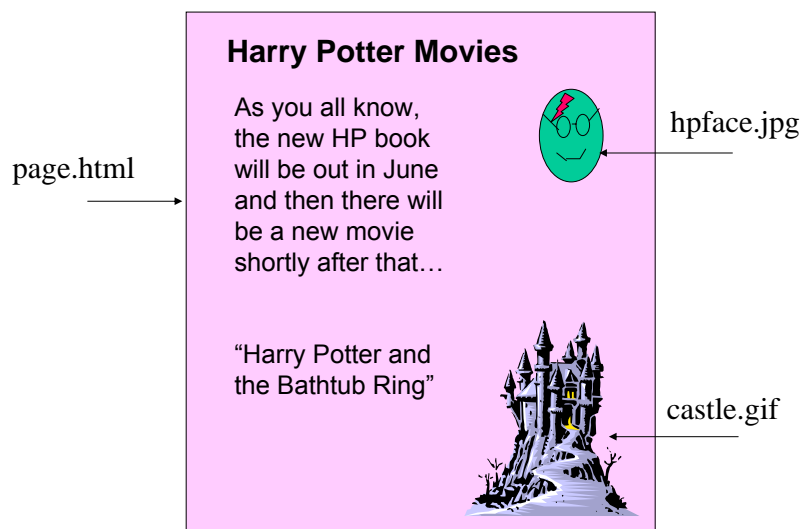
Connessioni in HTTP/1.1

- Per gestire le connessioni, introduzione di **connessioni persistenti** e **pipelining**
- Connessione persistente: possibilità di trasferire coppie multiple di richiesta e risposta entro una stessa connessione TCP
- Vantaggi delle connessioni persistenti
 - Riduzione dei costi (instaurazione ed abbattimento) delle connessioni TCP
 - 3-way handshake del TCP: solo per instaurare la connessione iniziale
 - Riduzione della latenza
 - Si evitano slow-start multipli del TCP
 - Controllo di congestione a regime
 - Migliore gestione degli errori
- Problemi delle connessioni persistenti
 - Concorrenza minore
 - Quando chiudere la connessione TCP?

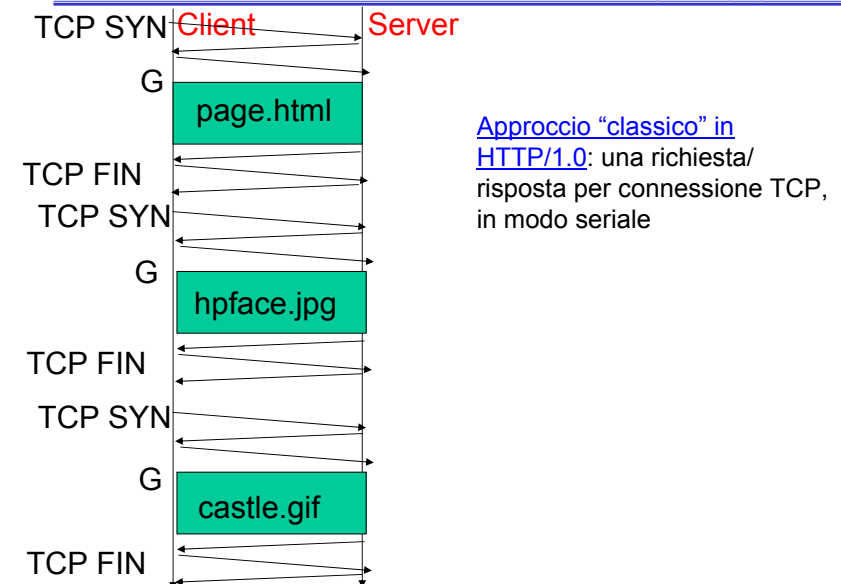
Connessioni in HTTP/1.1 (2)

- Pipelining: trasmissione di più richieste senza attendere l'arrivo della risposta alle richieste precedenti
- Le risposte devono essere date dal server nello stesso ordine in cui sono state fatte le richieste
 - HTTP non fornisce un meccanismo di riordinamento esplicito
 - Il server può tuttavia processare le richieste in un ordine diverso da quello di ricezione
- Vantaggi del pipelining
 - Riduzione ulteriore del tempo di latenza e ottimizzazione del traffico di rete, soprattutto per richieste che riguardano risorse molto diverse per dimensioni o tempi di elaborazione

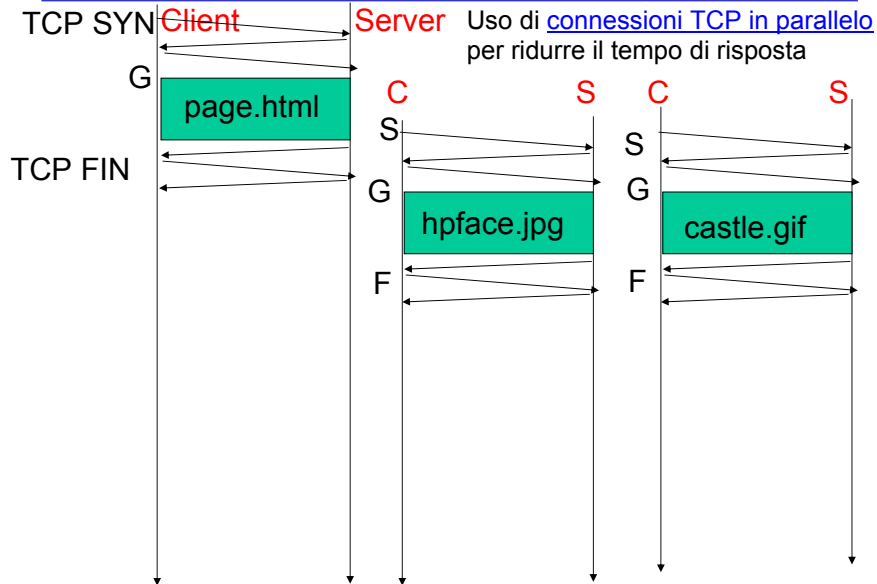
Connessioni in HTTP: esempio



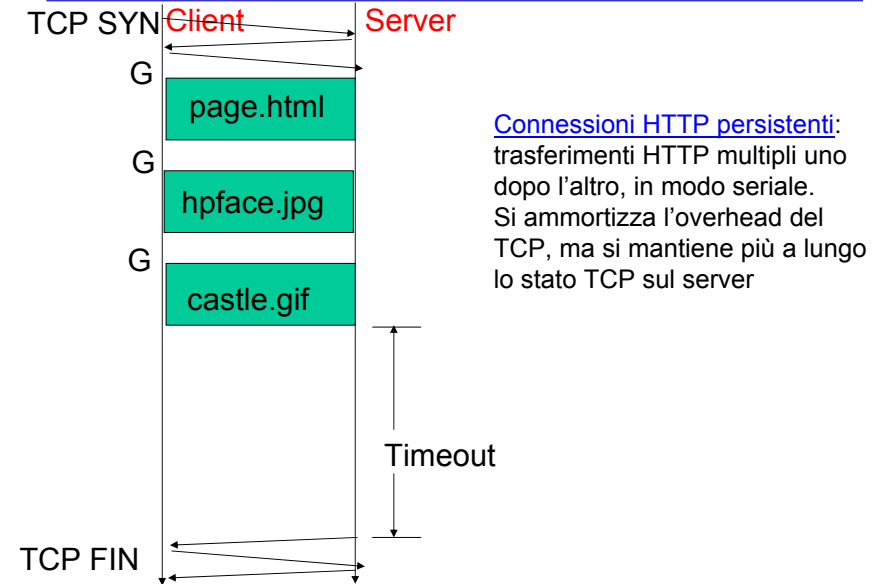
Connessioni in HTTP: esempio (2)



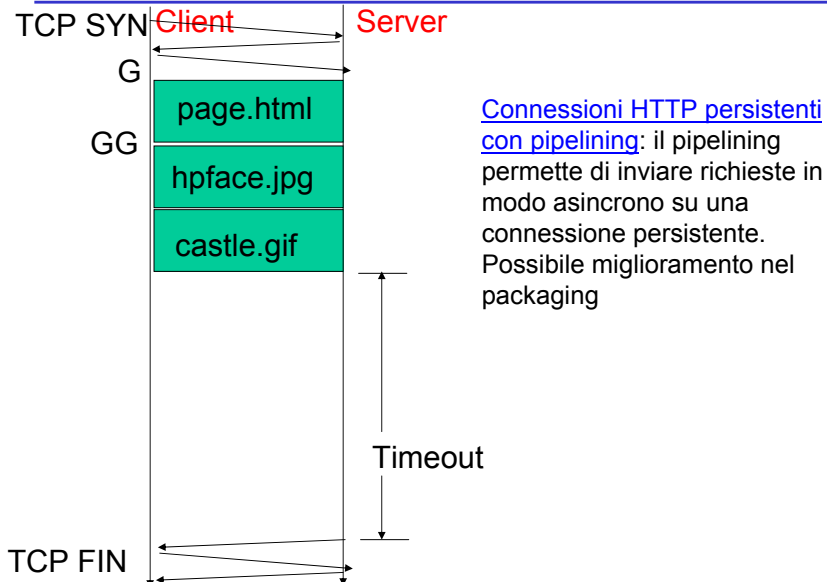
Connessioni in HTTP: esempio (3)



Connessioni in HTTP: esempio (4)



Connessioni in HTTP: esempio (5)



Connessioni persistenti in HTTP/1.1

- Modalità SHOULD di HTTP/1.1
- Nuovo header generale usato per segnalare la chiusura della connessione persistente
`Connection: close`
- L'implementazione della chiusura non è specificata nel protocollo: può essere iniziata dal client o dal server
- Ad esempio, il server può chiudere la connessione unilateralmente allo scadere di un timeout
 - Timeout applicato sul tempo totale di connessione o sul tempo di inattività di una connessione
- In alternativa, il server può fissare il numero massimo di richieste da servire sulla connessione

Esempio (uso di telnet)

- Esempio HTTP/1.0
 - Digitare da riga di comando
telnet www.ce.uniroma2.it 80
 - Premere INVIO e poi digitare sulla nuova linea
GET /index.html HTTP/1.0
 - Premere INVIO per 2 volte
- Esempio HTTP/1.1
 - Digitare da riga di comando
telnet www.ce.uniroma2.it 80
 - Premere INVIO e poi digitare sulla nuova linea
GET /index.html HTTP/1.1
 - Premere INVIO e poi digitare sulla nuova linea
Host: www.ce.uniroma2.it
 - Premere INVIO per 2 volte
 - Quale è il timeout HTTP sul server?

Hosting virtuale

- Ad uno stesso IP possono corrispondere nomi diversi e server diversi
 - Requisito importante per i “provider”
 - IP e porta non sono più sufficienti ad identificare il server
- Nuovo header di richiesta
 - Host:** serve a specificare il nome (e la porta) del server
 - Es.: **Host: www.uniroma2.it**
 - Obbligatorio in HTTP/1.1
 - Permette l’implementazione del virtual hosting senza manipolazioni del routing e multi-addressing IP

Negoziazione del contenuto

- In HTTP/1.1 introdotti nuovi header di richiesta per la negoziazione del contenuto
 - Accept, Accept-Charset, Accept-Encoding, Accept-Language, TE
- Esigenza del client di negoziare con il server la rappresentazione preferita di una risorsa
 - Accept: tipo di media preferito
 - Accept-Charset: insiemi di caratteri preferiti
 - Accept-Encoding: codifica del contenuto preferita
 - Accept-Language: linguaggio preferito
 - TE: codifica del trasferimento preferita
- Modalità di negoziazione del contenuto:
 - **Guidata dal client:** il client sceglie tra le alternative possibili ed indica la propria preferenza al server
 - **Guidata dal server:** il server sceglie la rappresentazione della risorsa in base alle informazioni ricevute dal client

Autenticazione digest in HTTP/1.1

- Il server invia al browser una stringa univocamente generata
 - “nonce”
- Il browser risponde con
 - nome utente
 - un valore crittografato basato su: nome utente, password, URI e nonce (digest MD5 di 128 bit)
- Il browser ricorda l’autorizzazione
- Riferimento
 - RFC 2617
- Vantaggi
 - La password non viene trasmessa in chiaro
 - Meccanismo più sicuro rispetto all’autenticazione Basic
- Ma...
 - Non è sicuro al 100%
 - E’ possibile intercettare la richiesta con URI, nonce e response e riprodurla per accedere alle risorse protette

Autenticazione digest in HTTP/1.1: esempio

- 1: richiesta
GET /private/index.html HTTP/1.1
Host: www.site.com
- 2: risposta
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
realm="Private Area",
nonce="dcd98b7102dd2f0"
- 3: il browser richiede nome e password all'utente
- 4: nuova richiesta
GET /private/index.html HTTP/1.1
Authorization: Digest
username="foo", realm="Private Area", nonce="dcd98b7102dd2f0", uri="/private/index.html", response="6629fae49393a05397450978507c4ef1"
- 5: nuova risposta (2xx oppure 4xx)

HTTPS: cenni

- La soluzione considerata più sicura: HTTPS
- HTTPS trasmette i dati in HTTP semplice su un protocollo di trasporto (SSL) che crittografa tutti i pacchetti
 - Il server ascolta su una porta diversa (per default la porta 443) e si usa uno schema di URI diverso (introdotto da https://)
 - Riferimento: HTTP over SSL (RFC 2818)
- SSL: Secure Socket Layer
 - Crittografia a chiave pubblica (certificato)
 - Trasparente

Ottimizzazione della banda

- Meccanismi introdotti in HTTP/1.1 per ottimizzare l'uso della banda, evitando sprechi:
 - Trasmissione delle sole parti necessarie della risorsa
 - Meccanismo **range request**
 - Scambio di informazioni di controllo per evitare trasmissioni inutili
 - Meccanismo **Expect/Continue**
 - Trasformazione della risorsa in modo da ridurre la dimensione
 - Compressione

Meccanismo range request

- Utilità di richiedere parti specifiche di una risorsa, anziché l'intera risorsa
 - Ripresa di un download interrotto prima della terminazione
 - Interesse del client per una sola parte della risorsa
 - Ad es., documenti strutturati, pagine di una risorsa PDF, ...
- Header di richiesta Range
 - Permette al client di specificare i byte della parte desiderata
 - Ad es., **Range: 2000-3999**
 - Possibile specificare anche insiemi di range
 - Ad es., **Range: 0-100, 2000-2400, 9600-**
- Header di entità Content-Range
 - Permette al server di specificare la parte di risorsa inviata
 - Ad es., **Content-Range: bytes 200-3999/100000**
- Codice di risposta 206 Partial Content
 - Il server indica al client che la risposta non è completa
- Header di risposta Accept-Range
 - Permette al server di indicare che non accetta richieste di parti; in alternativa, il server può inviare direttamente la risorsa intera

Compressione

- In HTTP solo compressione della risorsa, non degli header
- Header di entità Content-Encoding
 - Già presente in HTTP/1.0: indica una trasformazione che è stata applicata o può essere applicata ad una risorsa
 - Ad es.: **Content-Encoding: gzip**
 - Solo compressione end-to-end
- In HTTP/1.1 anche compressione hop-by-hop
 - Nuovi header di richiesta Accept-Encoding e TE
- Accept-Encoding
 - Permette di restringere l'insieme delle codifiche accettabili nella risposta
 - Ad es.: **Accept-Encoding: compress, gzip**

Meccanismo Expect/Continue

- Meccanismo introdotto per gestire richieste “costose”
- Permette al client di sapere se il server non è in grado di servire la richiesta *prima* di inviare il corpo della richiesta con il metodo POST
 - Ad es., invio di form di grandi dimensioni
- Header di richiesta Expect
 - Esempio di richiesta

```
POST /foo/bar HTTP/1.1
Content-Length: 23248
Expect: 100-Continue
```
 - Se il server può gestire la richiesta risponde con

```
HTTP/1.1 100 Continue
```
 - Il client può continuare inviando il corpo della richiesta
 - Se il server non può gestire la richiesta risponde con

```
HTTP/1.1 417 Expectation Failed
```

Trasmissione del messaggio

- Come può il client essere sicuro di aver ricevuto l'intero messaggio di risposta?
- In HTTP/1.0: unico meccanismo fornito dall'header di entità Content-Length
 - Dimensione facile da conoscere per risorse statiche (chiamata di sistema stat()), ma per risorse dinamiche il server deve aspettare che la risorsa sia generata: latenza eccessiva!
 - Soluzione in HTTP/1.0: chiudere la connessione terminato l'invio della risorsa dinamica
- Ma in HTTP/1.1 connessioni persistenti per default: cosa fare?
- Soluzione: **chunked encoding**
 - Messaggio di risposta inviato in pezzi (*chunk*), specificando la dimensione (in base 16) di ogni chunk ed inviando al termine un chunk di dimensione nulla

Trasmissione del messaggio (2)

- Header di entità Transfer-Encoding

```
Transfer-Encoding: chunked
```
- L'ultimo chunk può essere seguito da un trailer opzionale che contiene gli header di entità
- Esempio di risposta inviata in chunk

```
HTTP/1.1 200 OK
Server: Apache/1.3.2
Date: Thu, 01 Apr 2002 16:10:43 GMT
Transfer-Encoding: chunked
Content-Type: text/html
691
<... data ...>
76
<... data ...>
0
```

 - } Primo chunk di 1681 B
 - } Secondo chunk di 118 B
 - } Terzo chunk di 0 B

Caching in HTTP/1.0

- Caching lato client, lato server o lato intermediario (su un proxy interposto tra client e server)
 - Caching lato server: riduce i tempi di processamento della risposta ed il carico sui server; senza effetti sul carico di rete
 - Caching lato client e lato intermediario: riduce il carico di rete ed in parte il carico sui server
- HTTP/1.0 si basava su tre header:
 - **Expires**: il server specifica la scadenza della risorsa
 - Scadenza specificata con data: richiede sincronizzazione temporale tra server e cache
 - Fenomeno del **cache busting**: il server specifica una scadenza immediata per evitare il caching della risorsa
 - **If-Modified-Since**: il client o il proxy richiede la risorsa solo se modificata dopo la data indicata nella richiesta
 - **Pragma: no-cache**: il client indica ai proxy sul cammino di accettare solo la risposta dal server

Caching in HTTP/1.1

- Introduzione dell'header generale **Cache-Control** per permettere a client e server di specificare direttive per il caching
 - Direttive su **cacheability**, **validazione** e **coerenza** di risposte in cache
 - Due meccanismi per controllare la scadenza di una risorsa:
 - Scadenza **specificata dal server**
 - Scadenza **euristica**
- Introduzione dell'header di entità **ETag** per identificare univocamente la versione di una risorsa
- Introduzione dell'header di risposta **Vary** per elencare un insieme di header di richiesta da usarsi per selezionare la versione appropriata in una cache
- Introduzione dell'header **Via** per conoscere la catena di proxy tra client e server

Header ETag

- ETag: entity tag
- E' un identificare univoco della versione della risorsa
 - Versioni diverse della stessa risorsa hanno diverso ETag
 - Esempio di risposta con entity tag:

```
HTTP/1.1 200 OK
Date: Thu, 01 Apr 2002 16:10:43 GMT
Server: Apache/1.3.2
Last-Modified: Mon, 29 Mar 2002 12:23:26 GMT
ETag: cc678-12d12-66394036
Content-Type: text/html
Content-Length: 12067
...
```
- Può essere generato dal server usando la funzione di crittografia hash MD5
- Fornisce un meccanismo di identificazione delle versioni di una risorsa più fine rispetto all'header Last-Modified
- Consente di disaccoppiare la validazione della risorsa in cache dalla scadenza della risorsa

Header Cache-Control

- Può contenere molteplici direttive che controllano l'uso di tutte le cache situate tra il client che ha originato la richiesta e l'origin server
- Valore dell'header: lista di direttive
 - Ogni direttiva consiste in una parola chiave che identifica la direttiva ed, opzionale, il valore della direttiva
 - Esempio di header Cache-control

```
Cache-Control: max-age = 120, no-transform, proxy-revalidation
```
- Header generale: usato sia nelle richieste sia nelle risposte
 - Comunque la maggior parte delle direttive sono specifiche per richieste o risposte

Header Cache-Control: richiesta

- Le direttive nella richiesta permettono al client di sovrascrivere la gestione di default delle risorse attuata dalle cache
 - **no-cache**: la richiesta non può essere soddisfatta usando una risorsa in cache; si forza la rivalidazione
 - **no-store**: la risposta alla richiesta e la richiesta stessa non possono essere memorizzate in cache (protezione di dati sensibili)
 - **max-age <seconds>**: solo risorse in cache più fresche dell'età massima specificata dovrebbero essere usate per soddisfare la richiesta
 - **min-fresh <seconds>**: solo risorse in cache che sono fresche almeno per l'età specificata dovrebbero essere usate per soddisfare la richiesta
 - **max-stale <seconds>**: risorse in cache che non scadono oltre l'età specificata possono essere usate per soddisfare la richiesta (accettate anche risorse già scadute)
 - **no-transform**: la cache non può fornire una versione modificata della risorsa, ma solo la versione originale
 - **only-if-cached**: in caso di cache miss il proxy non dovrebbe inoltrare la richiesta

Header Cache-Control: risposta

- Le direttive nella risposta permettono al server di controllare la gestione delle risorse attuata dalle cache
 - **no-store** e **no-transform**: come direttive nella richiesta
 - **no-cache**: la risorsa può essere memorizzata in cache ma deve essere rivalidata prima di fornire la risposta
 - **private**: la risposta può essere riusata solo dal client che ha originato la richiesta
 - **public**: la risposta può essere memorizzata in cache e condivisa tra i client
 - **must-revalidate**: la cache deve sempre rivalidare la risorsa se questa è scaduta
 - **proxy-revalidation**: come **must-revalidate** ma si applica solo ai proxy
 - **max-age <seconds>**: la cache dovrebbe rivalidare la risorsa se questa ha raggiunto l'età massima specificata
 - **s-maxage <seconds>**: come **max-age** ma si applica solo ai proxy

Esempio di header Cache-control

- Due risorse fornite dal sito bigmoney.com:
 - welcome.html: risorsa statica
 - Può essere memorizzata in cache: da specificare la scadenza
 - Occorre rivalidare per gestire promozioni per nuovi utenti
 - mysecrets.html: risorsa generata dinamicamente per ogni utente
 - Non deve essere memorizzata in cache
- Risposta di bigmoney.com per welcome.html
`Cache-control: public, max-age=t, proxy-revalidation`
- Risposta di bigmoney.com per mysecrets.html
`Cache-control: no-store`
- Richiesta di Bill
 - Usa un portatile ed una connessione a banda larga
`Cache-control: no-transform, max-age=60`
- Richiesta di Bob
 - Usa un palmare ed una connessione a banda stretta
`Cache-control: max-stale=86400`

Scadenza specificata dal server

- Il server stabilisce una **scadenza** (Time To Live o **TTL**) della risorsa
 - Tramite l'header Expires o con la direttiva **max-age** nell'header Cache-Control
 - La direttiva **max-age** specifica il valore del TTL in secondi (**TTL relativo**)
 - L'header **Expires** specifica il valore del TTL con la data (**TTL assoluto**)
- Se il TTL è scaduto, la risorsa dovrebbe essere rivalidata prima di essere fornita in risposta
- Tuttavia, se la richiesta accetta anche risposte scadute (direttiva **max-stale** in Cache-Control della richiesta), o se l'origin server è irraggiungibile o se il proxy è configurato per sovrascrivere il TTL, la cache può rispondere con la risorsa scaduta
 - Usando nella risposta l'header generale **Warning: 110 Response is stale**

Scadenza specificata dal server (2)

- Se Cache-Control nella risposta del server specifica la direttiva **must-revalidate**, la risorsa scaduta non può *mai* essere rispedita
 - La cache deve rivalidare la risorsa con l'origin server
 - Se il server non risponde, la cache manderà un codice 504 **Gateway time-out** al client
- Se Cache-Control nella richiesta specifica la direttiva **no-cache**, la richiesta deve essere fatta sempre all'origin server

Scadenza euristica

- Ad una risorsa può non essere associato dal server un valore esplicito del TTL
- La cache stabilisce un valore *euristico* di durata della risorsa, dopo la quale assume che sia scaduta
 - Il protocollo HTTP non definisce algoritmi per determinare il TTL euristico
- Queste assunzioni possono essere a volte ottimistiche e risultare in risposte scorrette
 - Se non è valida con sicurezza una risorsa assunta fresca, allora la cache deve fornire nella risposta l'header generale **Warning: 113 Heuristic expiration**

Validazione della risorsa in cache

- Anche dopo la scadenza, nella maggior parte dei casi, una risorsa sarà ancora non modificata, e quindi la risorsa in cache valida
- Occorre rivalidare la risorsa: come fare?
- Modo semplice per rivalidare la risorsa: usare il metodo HEAD
 - Il client o il proxy effettua la richiesta con HEAD e verifica la data di ultima modifica (usando l'header di entità Last-Modified e l'header ETag)
 - In caso di risorsa non valida: sono necessarie due richieste

Validazione della risorsa in cache (2)

- Modo migliore per rivalidare la risorsa: fare una *richiesta condizionale*
 - Il client o il proxy effettua la richiesta condizionale (usando gli header di richiesta If-Modified-Since, If-Match e If-None-Match per specificare il valore dell'ETag)
 - Se la risorsa è stata modificata, viene fornita la nuova risorsa, altrimenti viene fornita la risposta con codice di stato **304 Not Modified** senza corpo della risposta
 - In caso di risorsa non valida: è necessaria una sola richiesta

Esempi di header HTTP

- In tutti gli esempi:
 - Utilizzo di Fiddler per lo sniffing degli header
 - Internet Explorer 7.0 come browser

• Esempio 1: <http://www.repubblica.it/>

- **Es. 1: richiesta HTTP per la home page**

```
GET / HTTP/1.1
Accept: /*
Accept-Language: it
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: www.repubblica.it
Proxy-Connection: Keep-Alive
Pragma: no-cache
Cookie: ebNewBandWidth_www.repubblica.it=669%3A1200041957539;
RMID=a050553d4756ac00; RMFD=011JFCjbO10167U;
RMFW=011JDFNu750163F
```

UA-CPU: header non standard
Proxy-Connection: header esteso

Esempi di header HTTP (2)

- **Es. 1: risposta HTTP per la home page**

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Date: Wed, 16 Jan 2008 18:28:34 GMT
Server: Apache
Accept-Ranges: bytes
Cache-Control: max-age=61
Expires: Wed, 16 Jan 2008 18:29:35 GMT
Age: 0
Content-Type: text/html

339
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
...
0
```

max-age ha priorità rispetto ad Expires per calcolare la scadenza della risorsa
Chunked encoding (dimensione in base 16 dei chunk)

Esempi di header HTTP (3)

- **Es. 1: richiesta HTTP per un'immagine nella cache del browser**

```
GET /sharedfiles/images/la_repubblica_logo.gif HTTP/1.1
Accept: /*
Referer: http://www.repubblica.it/
Accept-Language: it
UA-CPU: x86
Accept-Encoding: gzip, deflate
If-Modified-Since: Thu, 23 Nov 2006 12:25:26 GMT
If-None-Match: "1b08fdb-c85-2982b980"
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Proxy-Connection: Keep-Alive
Host: www.repubblica.it
Pragma: no-cache
Cookie: ebNewBandWidth_www.repubblica.it=669%3A1200041957539;
RMID=a050553d4756ac00; RMFD=011JFCjbO10167U;
RMFW=011JDFNu750163F
```

GET condizionale

Esempi di header HTTP (4)

- **Es. 1: risposta HTTP per un'immagine nella cache del browser**

```
HTTP/1.1 304 Not Modified
Date: Wed, 16 Jan 2008 18:28:41 GMT
Server: Apache
ETag: "1b08fdb-c85-2982b980"
Expires: Wed, 16 Jan 2008 19:05:03 GMT
Cache-Control: max-age=1800
```

Esempi di header HTTP (5)

- Esempio 2: <http://www.google.it/>

- Es. 2: richiesta HTTP per la home page**

GET / HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*

Accept-Language: it

UA-CPU: x86

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)

Host: www.google.it

Proxy-Connection: Keep-Alive

Cookie:

PREF=ID=0d6ea6eed8a5da6:TB=2:TM=1196690767:LM=119809085
4:S=TQ8Dj21qj8TDhL_0

Esempi di header HTTP (6)

- Es. 2: risposta HTTP per la home page**

HTTP/1.1 200 OK

Transfer-Encoding: chunked

Cache-Control: private

Content-Type: text/html; charset=UTF-8

Server: gws

Date: Wed, 16 Jan 2008 18:29:00 GMT

1ABD

<html><head><meta http-equiv="content-type" content="text/html;
charset=UTF-8">

...

0

Risposta privata

gws: Google Web Server

Esempi di header HTTP (7)

- Esempio 3: <http://web.uniroma2.it/>

- Es. 3: richiesta HTTP per la home page**

GET / HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*

Accept-Language: it

UA-CPU: x86

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)

Host: web.uniroma2.it

Proxy-Connection: Keep-Alive

Cookie: navpath=italiano%3AHOM

Esempi di header HTTP (8)

- Es. 3: risposta HTTP per la home page**

HTTP/1.0 200 OK

Date: Wed, 16 Jan 2008 18:31:30 GMT

Server: Apache/2.2.3 (Debian) PHP/4.4.4-9

X-Powered-By: PHP/4.4.4-9

Expires: Thu, 19 Nov 1981 08:52:00 GMT

Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0

Pragma: no-cache

Content-Type: text/html

Set-Cookie: PHPSESSID=a00f3eeeb8e90b718b6fac1b68a2b88b; path=/

Set-Cookie: PHPSESSID=e5a2167942c7e6299c3607ea1d5ae4a1; path=/

Set-Cookie: navpath=italiano%3A; expires=Wed, 16 Jan 2008 19:31:30
GMT

Set-Cookie: navpath=italiano%3AHOM; expires=Wed, 16 Jan 2008
19:31:30 GMT

Via: 1.1 web.uniroma2.it

Connection: close

<!--DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" ...

Via: proxy e versione del protocollo

Chiusura della connessione

Esempi di header HTTP (9)

- **Es. 3: risposta HTTP per l'immagine /home/img/home/logo.gif**

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Date: Wed, 16 Jan 2008 18:31:31 GMT
Server: Apache/2.2.3 (Debian) PHP/4.4.4-9 mod_ssl/2.2.3
      OpenSSL/0.9.8c mod_perl/2.0.2 Perl/v5.8.8
Last-Modified: Mon, 22 May 2006 08:26:47 GMT
ETag: "104151-1a2d-4363bfc0"
Accept-Ranges: bytes
Via: 1.1 web.uniroma2.it
Expires: Wed, 16 Jan 2008 19:30:42 GMT
Age: 49
Connection: close
Content-Type: image/gif
```

Age: stima dell'età della risorsa quando essa viene fornita da una cache

Presenza dell'header Age: risposta proveniente da una cache

Esempi di header HTTP (10)

- **Es. 3: risposta HTTP per <http://www.uniroma2.it/>**

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Date: Wed, 16 Jan 2008 21:05:52 GMT
Server: Apache/1.2.4
Last-Modified: Mon, 04 Apr 2005 11:04:25 GMT
ETag: "115b-d2-42511f39"
Accept-Ranges: bytes
Content-Type: text/html
```

```
D2
<HTML>
<HEAD>
<META HTTP-EQUIV="pragma" CONTENT="no-cache">
<TITLE>Universit&agrave; degli Studi di Roma "Tor Vergata"</TITLE>
<META HTTP-EQUIV="refresh" CONTENT="0;URL=http://web.uniroma2.it">
</HEAD>
</HTML>
```

Redirezione da www.uniroma2.it a web.uniroma2.it

0

Esempi di header HTTP (11)

- Esempio 4: <http://www.tim.it/>

- **Es. 4: risposta HTTP per <http://www.tim.it/>**

```
HTTP/1.1 302 Moved Temporarily
Server: Sun-ONE-Web-Server/6.1
Date: Wed, 16 Jan 2008 20:57:47 GMT
Content-length: 0
Content-type: text/html
Location: http://www.tim.it/consumer/homepage.do
```

Redirezione HTTP

- **Es. 4: nuova richiesta HTTP dopo la redirezione**

```
GET /consumer/homepage.do HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-
      shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint,
      application/msword, */*
Accept-Language: it
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: www.tim.it
Proxy-Connection: Keep-Alive
Cookie: __utma=237224453.517890037.1200148164.1200139164.1300149164.1;
      __utmz=237224463.1201149164.1.1.utmccn=(direct)|utmcsr=(direct)|utmcmd=(
      none); vgnvisitor=2wp0kw109AN0000ChUzgSMR4Q2
```

Esempi di header HTTP (12)

- **Es. 4: risposta HTTP**

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Server: Sun-ONE-Web-Server/6.1
Date: Wed, 16 Jan 2008 20:57:48 GMT
Content-type: text/html; charset=UTF-8
Cache-Control: max-age=300
X-Powered-By: Servlet/2.4 JSP/2.0
```

```
54A
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN
...
```

Esempi di header HTTP (13)

- Esempio 5: <http://www.akamai.com/>

- **Es. 5: risposta HTTP per la home page**

HTTP/1.1 200 OK

Transfer-Encoding: chunked

Server: Apache/1.3.37 (Unix)

Content-Type: text/html; charset=utf-8

Set-Cookie: industry=3;path=/;domain=www.akamai.com;

Setting del cookie

ETag: "11b6fd-3b9a-478cd519"

Vary: Accept-Encoding

Vary: header per la
negoiazione del
contenuto guidata dal
server

Expires: Wed, 16 Jan 2008 18:30:32 GMT

Cache-Control: max-age=0, no-cache, no-store

Pragma: no-cache

Date: Wed, 16 Jan 2008 18:30:32 GMT

Connection: close

...

Esempi di header HTTP (14)

- Esempio 6: <http://www.cnn.com/>

- **Es. 6: richiesta HTTP per una risorsa (quale?)**

GET

/html.ng/site=cnn&cnn_position=126x31_spon1&cnn_rollup=homepage
&page.allowcompete=yes¶ms.styles=fs&tile=8071638050021&do
mId=403093 HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*

Referer: <http://www.cnn.com/>

Referer: rivela il sito di
provenienza

Accept-Language: it

UA-CPU: x86

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)

Proxy-Connection: Keep-Alive

Host: ads.cnn.com

Cookie: SelectedEdition=www

Esempi di header HTTP (15)

- **Es. 6: risposta HTTP per una risorsa (quale?)**

HTTP/1.1 200 OK

Date: Wed, 16 Jan 2008 18:32:42 GMT

Server: Apache

Set-Cookie: NGUserID=aa5122a-1392-1200508362-1; expires=Wednesday, 30-
Dec-2037 16:00:00 GMT; path=/

AdServer: ad8ad2:9678:1

P3P: CP="NOI DEVa TAla OUR BUS UNI"

P3P header (per privacy)

Cache-Control: max-age=0, no-cache

CP: Compact Policy token

Expires: Wed, 16 Jan 2008 18:32:42 GMT

Pragma: no-cache

Content-Type: text/html

Content-Length: 440

<a target="_blank"

href="/event.ng/Type=click&FlightID=4621&AdID=4789&TargetID=1515&Segme
nts=730,2743,3030,3285,8463,8796,9496,9557,9779,9781,9853,10381,12639&
Targets=16035,1515&Values=31,43,51,60,72,83,91,100,110,1266,1557,1588,2
677,4413,4418,4442,47181,47736,49516,49553,52508,52738,52778,52897,546
82&RawValues=&Redirect=http://www.cnn.com">

Esempi di header HTTP (16)

- Esempio 7: <http://www.kataweb.it/>

- **Es. 7: richiesta HTTP per un'immagine**

GET /adimages/T/379686.jpg HTTP/1.1

Accept: */*

Referer: <http://www.kataweb.it/>

Accept-Language: it

UA-CPU: x86

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)

Proxy-Connection: Keep-Alive

Host: annunci.kataweb.it

Cookie: RMID=5237e9f447626190; RMFD=011JFFBeO10167U

- **Es. 7: risposta HTTP per un'immagine**

HTTP/1.1 200 OK

Date: Wed, 16 Jan 2008 21:01:15 GMT

Server: Apache

X-Powered-By: PHP/5.2.1

Expires: Thu, 17 Jan 2008 21:01:15 GMT

Cache-Control: Public

Pragma: Public

Last-Modified: Tue, 06 Nov 2007 16:28:06 GMT

Content-Length: 2209

Content-Type: image/jpeg

Risorsa cacheable (public)
ma con scadenza pari a 0!

Esempi di header HTTP (17)

- **Es. 7: richiesta HTTP per accedere all'account di email**

POST /registrazione/katamail/infologin.jsp HTTP/1.1

Metodo POST

Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*

Referer: http://www.kataweb.it/

Accept-Language: it

Content-Type: application/x-www-form-urlencoded

UA-CPU: x86

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)

Proxy-Connection: Keep-Alive

Content-Length: 71

Content-Length del body

Host: login.kataweb.it

Pragma: no-cache

Cookie: RMID=5437e9f447546190; RMFD=021JFFBeO31167

frames=no&lang=it_IT&domain=katamail.com&login=xxxxxxx&passwd=y
yyyyyy

Login e password trasmesse in chiaro nel body della richiesta!