Il protocollo HTTP (HyperText Transfer Protocol)

Antonio Lioy
< lioy@polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

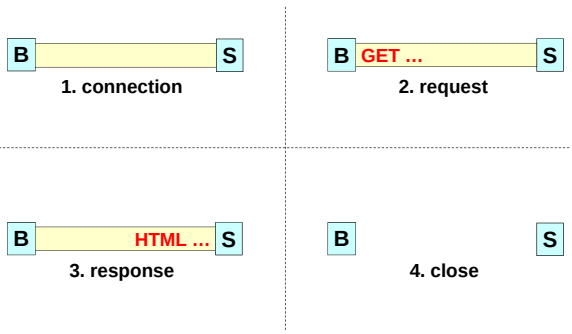
Un po' di storia di HTTP

- **protocollo client-server**
- **ideato per la richiesta e la trasmissione di pagine HTML (e poi di risorse web in generale)**
- **HTTP originale (anche noto come HTTP/0.9)**
  - uso quasi solo sperimentale
- **HTTP/1.0**
  - il primo a larga diffusione
  - ancora molto usato
- **HTTP/1.1**
  - per migliorare l'efficienza del web
  - per migliorare la gestione delle cache

Il protocollo HTTP 1.0

- **HyperText Transfer Protocol**
- **RFC-1945 (HTTP/1.0)**
- **servizio di default: TCP/80**
- **protocollo stateless**
- **lecito per il client chiudere la connessione prima di aver ricevuto la risposta o parte di essa**
- **chiusura del canale da parte del server**
- **dati a 8 bit (ossia "8-bit clean")**
- **alfabeto di default: ISO-8859-1 (= Latin-1)**

Connessione stateless (HTTP/1.0)



I link

- **URL (Uniform Resource Locator)**

  schema* : // *user* : *password* @ *host* : *porta* / *path* # *àncora

- **schemi regolari:**
  - http, telnet, ftp, gopher, file
- **schemi irregolari:**
  - news:newsgroup
  - mailto:indirizzo-postale
- **definizione base in RFC-1738, più 1959+2255 (LDAP), 2017 (external-body), 2192 (IMAP), 2224 (NFS), …**

Evoluzione dei link

- **URL sono indirizzi fisici (usando CNAME si può fare qualcosa di logico, ma poco)**
- **URN (Uniform Resource Name) è l'evoluzione futura, per usare nomi logici, replicazione, ...**
- **URI (Uniform Resource Identifier)**
  - RFC-3986
  - URI = URL + URN

Protocollo HTTP/1.0

- **comandi ASCII con righe terminate da CR+LF**
- **i dati possono essere binari (perché il protocollo è "8-bit clean")**
- **messaggi costituiti da header + body**
- **header sono righe che iniziano con "keyword: "**
- **header separato dal body mediante una riga vuota (ossia contenente solo CR+LF)**

Metodi HTTP/1.0

- **GET *uri http-version***
  richiede la risorsa associata alla URI specificata
- **HEAD *uri http-version***
  restituisce solo gli header della risposta, non i dati
- **POST *uri http-version***
  invia dei dati al server (nel body) affinché vengano elaborati dalla URI indicata
- **risposte devono iniziare con**
  http-version status-code [commento_testuale]
- **notare che la URI è**
  - il path della risorsa richiesta (quando si è connessi direttamente all'origin server)
  - la URI completa (quando si è connessi ad un proxy)

Metodo GET

– – – – – – – – – – – (TCP setup) – – – – – – – – – – –

request

```
GET / HTTP/1.0 ⟶
```

response

```
HTTP/1.0 200 OK
Date: Wed, 20 May 1998 09:58:21 GMT
Server: Apache/1.0.0
Content-Type: text/html
Content-Length: 1534
Last-modified: Fri, 5 May 1998 12:14:23 GMT

<html>
. . .
</html>
```

⟵

– – – – – – – – – – – (TCP teardown) – – – – – – – – – – –

Metodo HEAD

(TCP setup)

request

```
HEAD / HTTP/1.0
```

response

```
HTTP/1.0 200 OK
Date: Wed, 20 May 1998 09:58:21 GMT
Server: Apache/1.0.0
Content-Type: text/html
Content-Length: 1534
Last-modified: Fri, 5 May 1998 12:14:23 GMT
```

(TCP teardown)

Metodo POST

(TCP setup)

request

```
POST /cgi-bin/form.cgi HTTP/1.0
Content-type: text/plain
Content-length: 12

Antonio Lioy
```

response

```
HTTP/1.0 200 OK
Date: Mon, 8 Mar 1999 21:30:24 GMT
Server: Apache/1.2.6
Content-type: text/plain

dati ricevuti da stdin: Antonio Lioy
```

(TCP teardown)

Test manuale di HTTP (Windows)

- aprire una finestra di comando (aka “finestra DOS”)
- dare i seguenti comandi:

```
C:\> telnet
Microsoft Telnet> set localecho
Microsoft Telnet> set crlf
Microsoft Telnet> set logging
Microsoft Telnet> set logfile mylog.txt
Microsoft Telnet> open server_web 80
GET / HTTP/1.0 <enter>
<enter>
```

- suggerimento: se non si vuole attivare il log allora attivare lo scrolling sulla finestra per vedere tutta la risposta del server (Proprietà – Layout – Screen buffer size – Height)

Test manuale di HTTP (*nix)

- installare (se non presente) il programma "netcat"
- aprire una finestra di shell
- dare i seguenti comandi:

```
$ nc server_web 80 > output.txt
GET / HTTP/1.1 <enter>
Host: nome_server_web <enter>
Connection: close <enter>
<enter>
```

Codice di stato HTTP

- tutte le risposte contengono un codice di stato di 3 cifre XYZ
- la prima cifra (X) fornisce il major status dell'azione richiesta
  - X=1 : informational
  - X=2 : success
  - X=3 : redirection
  - X=4 : client error
  - X=5 : server error
- seconda e terza cifra affinano lo stato

Codici di stato standard HTTP/1.0

200 OK	400 Bad request
201 Created	401 Unauthorized
202 Accepted	403 Forbidden
204 No Content	404 Not found
301 Moved permanently	500 Internal server error
302 Moved temporarily	501 Not implemented
304 Not modified	502 Bad gateway
	503 Service unavailable

Redirect

- **se l'oggetto richiesto non è disponibile alla URI indicata, il server può indicare la nuova URI tramite una risposta con codice 3xx e header**
  Location: nuova-URI
- **il browser può:**
  - connettere automaticamente il client alla nuova URI se il metodo richiesto era GET o HEAD …
  - … mentre gli è vietato farlo se il metodo era POST
- **per supportare i browser vecchi (= che non capiscono Redirect) o pigri si consiglia di inviare nella risposta un body costituito da una pagina HTML che fornisca la nuova URL in modo comprensibile da un essere umano**

Header HTTP/1.0 (I)

- **general header:**
  - Date: *http-date*
  - Pragma: no-cache
- **request header:**
  - Authorization: *credentials*
  - From: *user-agent-mailbox*
  - If-Modified-Since: *http-date*
    - 304 se l'entity non è cambiata dalla data indicata
  - Referer: *URI*
    - richiesta susseguente un Redirect
  - User-Agent: *product*

Header HTTP/1.0 (II)

- **response header:**
  - Location: *absolute-URI*
  - Server: *product*
  - WWW-Authenticate: *challenge*
- **entity body header:**
  - Allow: *method*
  - Content-Encoding: x-gzip | x-compress
  - Content-Length: *length*
  - Content-Type: *MIME-media-type*
  - Expires: *http-date*
  - Last-Modified: *http-date*

HTTP date

- **tre possibili formati:**
  - RFC-822 = Sun, 06 Nov 1994 08:49:37 GMT
  - RFC-850 = Sunday, 06-Nov-94 08:49:37 GMT
  - asctime = Sun Nov 6 08:49:37 1994
- **accettare tutti, generare solo RFC-822**
- **sempre GMT, mai l'indicazione del fuso orario**

Il protocollo HTTP/0.9

- **più semplice**
- **non documentato in nessun RFC**
- **request: GET *URI***
- **response: *entity-body***

Problemi di HTTP/1.0

- **progettato per oggetti statici**
  - dimensione nota a priori
  - altrimenti possibile troncamento dei dati
- **impossibile continuare collegamento interrotto**
- **progettato per richiedere e ricevere singole pagine:**
  - ogni oggetto richiede una connessione separata
- **una connessione TCP per ogni richiesta:**
  - TCP setup richiede tempo (3-way handshake)
  - TCP slow start richiede tempo
  - la chiusura del canale fa perdere le informazioni sulla congestione della rete
- **gestione elementare della cache**

Il protocollo HTTP/1.1 (I)

- **RFC-2616**
- **connessioni persistenti (è il default) e pipelining**
  - maggior velocità (più transazioni su stesso canale)
- **miglior trasmissione del body**
  - negoziazione formato dati e lingua del body
  - frammentazione (chunked encoding) per pagine dinamiche
  - trasmissione parziale
- **gestione della cache**
  - modalità specificate dal protocollo
  - proxy gerarchici

Il protocollo HTTP/1.1 (II)

- **supporto per server virtuali**
  - più server logici associati allo stesso indirizzo IP
- **nuovi metodi:**
  - PUT, DELETE, TRACE, OPTIONS, CONNECT
- **nuovo metodo di autenticazione (basato su digest) a livello di trasporto**

HTTP/1.1: virtual host

- **con HTTP/1.0 occorreva un indirizzo IP per ogni server web ospitato su un nodo (multihomed)**
- **con HTTP/1.1 non è più necessario:**
  - più server logici associati allo stesso indirizzo IP
  - ottenibili tramite alias nel DNS (record CNAME)
- **il client deve indicare il virtual host desiderato tramite il suo FQDN (Fully Qualified Domain Name)**
- **nuovo header: "Host: FQDN [: port]"**

HTTP/1.1: virtual host (esempio)

```
host.provider.it   IN  A      10.1.1.1
www.musica.it      IN  CNAME  host.provider.it
www.libri.it       IN  CNAME  host.provider.it
```
DNS

(collegamento a host.provider.it, ossia IP 10.1.1.1)
```
GET /index.html HTTP/1.1
Host: www.musica.it
```
HTTP

(collegamento a host.provider.it, ossia IP 10.1.1.1)
```
GET /index.html HTTP/1.1
Host: www.libri.it
```
HTTP

HTTP/1.1: connessioni persistenti

- uso di un unico canale TCP per trasmettere più interazioni richiesta-risposta
- è il comportamento di default in HTTP/1.1
- chi si comporta diversamente deve dichiararlo con: Connection: close
- non è un header end-to-end ma hop-by-hop
  - cosa importante per proxy e gw

Connessioni persistenti: esempio

- - - - - - - - - - - (TCP setup) - - - - - - - - - - -

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

```
HTTP/1.1 200 OK
. . .
```

```
GET /favicon.gif HTTP/1.1
Host: www.polito.it
Connection: close
```

```
HTTP/1.1 200 OK
Connection: close
. . .
```

- - - - - - - - - - - (TCP teardown) - - - - - - - - - - -

Connessioni persistenti: pro e contro

- **vantaggi:**
  - minor overhead apertura (3-way handshake) e chiusura canale (4-way handshake e suoi timeout)
  - miglior gestione delle congestioni di rete (mantenimento della window TCP)
  - client può fare pipelining delle richieste (server deve fornire risposte nello stesso ordine)
  - risparmio di CPU (su tutti i nodi della catena)
  - evoluzione “dolce” a nuove versioni di HTTP (client può provare nuova versione ma poi usare vecchia)
- **svantaggi:**
  - sovraccarico del server (possibile denial-of-service)

HTTP/1.1: pipeline

- **possibile inviare più richieste senza attendere le risposte del server**
- **ottimizza la trasmissione TCP**
- **molto utile quando si richiedono in un colpo solo più elementi (di una stessa risorsa)**
- **attenzione:**
  - sono richieste sequenziali, non parallele (le risposte saranno ricevute nello stesso ordine delle richieste)
  - in caso di errore, potrebbe essere necessario ripetere tutto il comando

HTTP/1.1: compressione

- **risparmio:**
  - di byte trasmessi
  - di tempo (come percepito dall’utente finale)
- **costi:**
  - CPU sul client
  - CPU sul server (se compressione fatta “al volo”)
  - ma la risorsa oggi scarsa è la banda, non la CPU …
- **in HTTP/1.1 implementata come codifica dei dati oppure come codifica di trasmissione**

Codifiche dati

- **codifica applicata ad una risorsa prima di trasferirla (proprietà del dato, non della trasmissione)**
- **poiché HTTP è 8-bit clean, codifica = compressione**
- **utile per conservare il MIME-type della risorsa**
- **header: Content-Encoding e Accept-Encoding**
- **codifiche possibili:**
  - gzip = GNU zip (RFC-1952) = LZ-77 + CRC-32
  - compress = programma Unix (adaptative LZW)
  - deflate = zlib (RFC-1950) + deflate (RFC-1951)
  - identity = nessuna codifica
    - default se Content-Encoding assente
  - accettabili i nomi pre-standard x-gzip e x-compress

Codifiche di trasmissione

- **specifica la modalità di trasferimento di una risorsa (proprietà della trasmissione, non del dato)**
- **header: TE e Transfer-Encoding**
- **simile all'header MIME Content-Transfer-Encoding ma poiché il canale è pulito a 8 bit l'unica difficoltà è determinare la lunghezza del messaggio**
- **valori (possibili più di uno):**
  - identity (default)
  - gzip, compress, deflate
  - chunked (se presente, deve essere l'ultimo)

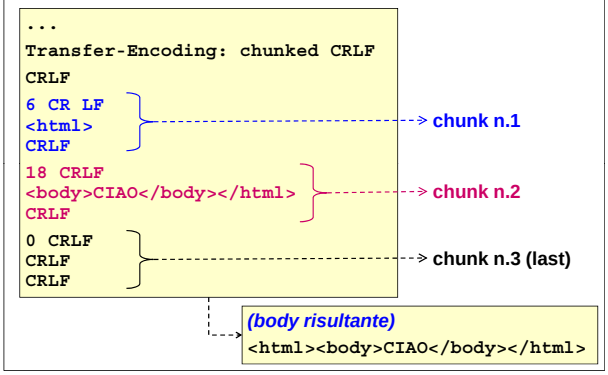
Codifica "chunked"

- **utile quando il server non conosce a priori la dimensione dei dati da trasferire**
- **caso tipico dei server dinamici (es. ASP, PHP, JSP)**
- **si frammenta la risposta in pezzi (detti "chunk")**
- **meno importante in HTTP/1.0 (fine dati implicita alla chiusura del canale) ma protocollo meno efficiente**

Codifica “chunked”: sintassi

- sintassi (del body):
  *chunk
  last-chunk
  *(entity-header CR LF)
  CR LF
- ogni chunk:
  chunk-size [chunk-extensions] CR LF
  chunk-data CR LF
- size in hex, ultimo chunk è quello con size=0
- nota: "*" indica zero o più ripetizioni dell'oggetto seguente

Codifica chunked: esempio



CSS / PNG / HTTP-1.1 e le prestazioni

- CSS semplifica HTML (e quindi ne riduce la dimensione)
- PNG ha compressione migliore di GIF
  - es. risparmio del 12% sulle 40 immagini del test
- impatto di HTTP-1.1 complesso: connessioni persistenti, pipelining, compressione

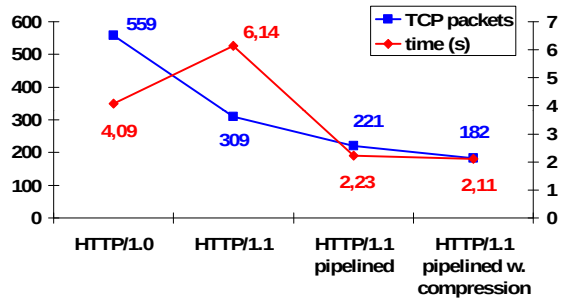
Test HTTP 1.0 vs. 1.1

- **test di Nielsen, Gettys *et al* (W3C)**
- **pagina di test:**
  - dimensione HTML = 40 KB
  - contiene 43 immagini in-line (totale 130 KB)
- **test consistente in 44 GET (una per ciascun oggetto) in vari modi:**
  - HTTP/1.0 con 4 connessioni simultanee
  - HTTP/1.1 con 1 connessione persistente
  - HTTP/1.1 pipeline con 1 connessione persistente
  - HTTP/1.1 pipeline + compressione con 1 connessione persistente

www.w3c.rl.ac.uk/pastevents/RALSymposium98/Talks/br0.html

Risultati test HTTP 1.0 vs. 1.1

WAN first-time retrieval against Apache server



HTTP/1.1: metodo PUT

- **PUT *uri http-version***
  - il request body costituisce una nuova risorsa da memorizzarsi sul server alla URI specificata
  - il request body è una nuova versione da sostituirsi a quella specificata dalla URI
- **risposte: 201 (Created), 200 (OK) o 204 (No content)**
- **esempio:**

```
PUT /avviso.txt HTTP/1.1
Host: lioy.polito.it
Content-Type: text/plain
Content-Length: 40

Il 31/5/2007 non ci sar&agrave; lezione.
```

Il codice 100 Continue

- **è inefficiente trasmettere un grosso request body se il server rifiuterà la richiesta senza neanche guardarlo:**
  - es. utente non autorizzato o metodo non supportato
  - tipicamente relativo a PUT e POST
- **il client chiede conferma esplicita prima di trasmettere il request body**

```
PUT /voti.pdf HTTP/1.1
Host: www.abc.com
Expect: 100-continue
                          HTTP/1.1 100 Continue
... voti.pdf ...
                          HTTP/1.1 200 OK
```

HTTP/1.1: metodo DELETE

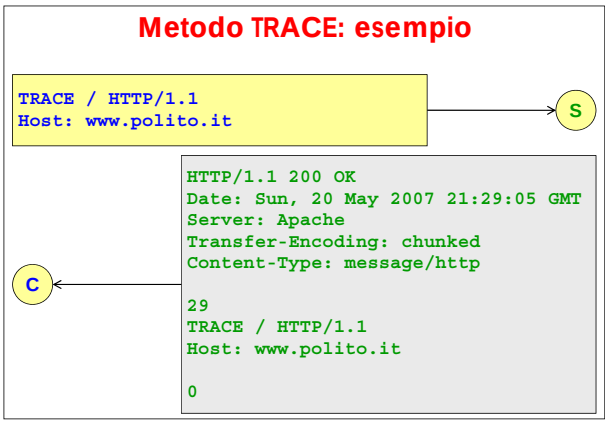
- **DELETE *uri http-version***
  - chiede eliminazione della risorsa alla URI specificata
  - nessuna garanzia che sia cancellata, anche se OK
- **risposte:**
  - 200 (OK) se eseguita + body con dettagli
  - 202 (Accepted) se canc. richiede decisione umana
  - 204 (No content) se eseguita ma senza dettagli
- **esempio:**

```
DELETE voti.html HTTP/1.1
Host: lioy.polito.it
Content-Length: 0
```

HTTP/1.1: metodo TRACE

- **TRACE *uri http-version***
  - richiede di ricevere copia della richiesta
  - richiesta incapsulata in un response body con tipo MIME "message/http"
- **tramite i request header "Via" (aggiunti dai proxy e dai gateway) è possibile tracciare il collegamento a livello applicativo**
- **tramite il request header "Max-Forwards" è possibile limitare il numero di proxy attraversati (utile in caso di loop)**

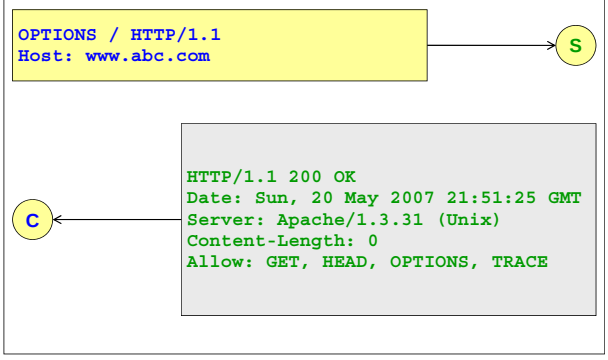
Metodo TRACE: esempio



HTTP/1.1: metodo OPTIONS

- **OPTIONS *uri http-version***
  - richiede di conoscere le opzioni supportate dal server
  - solo opzioni generali se URI=*, altrimenti quelle generali+specifiche della URI (es. lingue in cui la risorsa è disponibile)
- **tramite il request header "Max-Forwards: N" è possibile richiedere le opzioni non all'origin server ma al proxy N-1 attraversato**

Metodo OPTIONS: esempio



HTTP/1.1: metodo CONNECT

- **riservato all'uso con proxy capaci di creare un tunnel sicuro (es. SSL)**

Trasmissione parziale

- **utile quando si vuole riprendere un trasferimento interrotto**
- **header del server:**
  - Accept-Ranges: none | bytes
  - Content-Range: bytes start–stop/total (total=* se dimensione totale ignota)
- **header del client:**
  - Range: range1, range2, …, rangeN
- **possibili range:**
  - start–stop (intervallo, estremi inclusi)
  - –lastN (gli ultimi lastN byte)
  - startByte– (da startByte sino alla fine)

Entity tag (Etag)

- **identificativo "opaco" da usarsi in alternativa alla data di modifica per sapere se una entity è cambiata**
- **inserito dal server nel response header tramite:**
  Etag: [W/] "hex_string"
- **tag uguali implicano:**
  - (strong Etag) oggetti identici (stessi byte)
  - (weak Etag) oggetti equivalenti (stesso effetto)
- **valore usato dal client in request header con:**
  - If-Match (es. aggiornare con PUT una versione precedente solo se non è già stata modificata)
  - If-None-Match (es. fare GET di nuova versione)

HTTP/1.1 request header (I)

- **Accept: media-range [; q=qualityValue] , …**
  - formati dati accettabili
  - es. Accept: image/jpeg, image/*;q=0.5
- **Accept-Charset: charset [; q=qualityValue] , …**
  - set di caratteri accettabili
- **Accept-Encoding: cont-enc [; q=qualityValue] , …**
  - codifiche del contenuto accettabili
- **Accept-Language: language [; q=qualityValue] , …**
  - lingue accettabili
  - es. Accept-Language: it, en-gb;q=0.8, en;q=0.7

HTTP/1.1 request header (II)

- **Authorization: credenziali**
  - credenziali in risposta ad una richiesta di autenticazione da parte dell'origin server
- **Expect: comportamento-atteso , …**
  - il server risponde con 417 se non capisce
- **From: rfc822-address**
  - importante soprattutto per contattare chi ha attivato un robot che crea problemi
  - problemi di privacy/spamming
- **Host: hostname [: port]**
  - virtual host da contattare (porta di default: 80)

HTTP/1.1 request header (III)

- **If-Match: etagValue , …**
  - applicare il metodo solo se la risorsa corrisponde ad uno dei tag indicati
- **If-Modified-Since: http-date**
  - applicare il metodo solo se la risorsa è stata modificata dopo la data indicata
- **If-None-Match: etagValue , …**
  - applicare il metodo solo se la risorsa non corrisponde ad alcuno dei tag indicati
- **If-Range: etagValue | http-date**
  - inviare tutta la risorsa se è cambiata, altrimenti solo la porzione indicata (da usarsi con Range: …)

HTTP/1.1 request header (IV)

- **If-Unmodified-Since: http-date**
  - applicare il metodo solo se la risorsa non è stata modificata dopo la data indicata
- **Max-Forwards: max-num-forwards**
  - da usarsi con TRACE e OPTIONS per identificare il nodo intermedio che deve rispondere
  - max-num-forwards decrementato da ogni nodo
- **Proxy-Authorization: credenziali**
  - credenziali in risposta ad una richiesta di autenticazione da parte di un proxy

HTTP/1.1 request header (V)

- **Range: intervallo**
  - chiede di ricevere solo l'intervallo di dati indicato
    - offsetFrom–offsetTo (es. bytes=500–999)
    - –lastBytes (es. bytes=–500)
- **Referer: absoluteURI | relativeURI**
  - URI della pagina che ha generato l'attuale richiesta
  - campo assente se URI inserita da tastiera
- **TE: [trailers] [transf-enc [; q=qualityValue]] , …**
  - chunk trailers e codifiche di trasferimento accettabili
- **User-Agent: nome-prodotto**
  - identifica il software che implementa il client

HTTP/1.1 response header (I)

- **Accept-Ranges: bytes | none**
  - indica se il server permette il download parziale
- **Age: *ageValue***
  - indica i secondi trascorsi da quando la risorsa è stata inserita in cache
  - ha senso solo in proxy, gateway o server con cache
- **ETag: *entity-tag***
  - un identificativo opaco unico della risorsa richiesta
- **Location: *absolute-URI***
  - effettua un redirect alla URI indicata

HTTP/1.1 response header (II)

- **Proxy-Authenticate: sfida**
  - sfida proposta da un proxy per autenticare il client
- **Retry-After: http-date | secondi**
  - in caso di indisponibilità del server
- **Server: nome-prodotto**
  - identifica il software che implementa il rispondente
- **Vary: * | request-header , ...**
  - indica i campi della richiesta da cui dipende la risposta (e quindi la validità di una copia in cache)
- **WWW-Authenticate: *sfida***
  - sfida proposta dall'origin server per autenticare il client

HTTP/1.1 general header (I)

- **Cache-Control: req-cache-dir | res-cache-dir , …**
  - controllo della cache nella richiesta e nella risposta
  - direttive del client e del server verso i proxy
- **Connection: close**
  - per connessione non persistenti (con server o proxy)
- **Date: http-date**
  - data di invio della richiesta o della risposta
- **Pragma: no-cache**
  - per compatibilità con HTTP/1.0
  - in HTTP/1.1 si preferisce Cache-Control

Gestione della cache

- **richieste del client**
  - voglio o meno risposte presenti in cache
  - …
- **risposte del server**
  - non mettere in cache
  - ...

Request-cache-directive (I)

- **no-cache**
  - voglio una risposta dall'origin server
- **no-store**
  - non memorizzare (in tutto o in parte) né la richiesta né la risposta
- **max-age=secondi**
  - voglio una risposta non più vecchia di X s
- **max-stale [=secondi]**
  - accetto anche risposte scadute (da non più di X s)
- **min-fresh=secondi**
  - voglio una risposta che sia ancora valida tra X s

Request-cache-directive (II)

- **no-transform**
  - ai proxy è vietato effettuare trasformazioni della risorsa (es. alcuni cambiano il formato delle immagini per risparmiare spazio)
- **only-if-cached**
  - non voglio contattare origin server
  - utile con reti sovraccariche

Response-cache-directive (I)

- **public**
  - risposta può essere messa in cache (shared/private)
- **private [="header-name, …"]**
  - non mettere in shared cache (tutto o alcuni header)
- **no-cache [="header-name, …"]**
  - non mettere in cache (tutto o alcuni header)
- **no-store**
  - non mettere in cache (shared/private) su disco
- **no-transform**
- **must-revalidate**
  - il proxy / client deve rivalidare con origin server quando la risorsa in cache diventa stale

Response-cache-directive (II)

- **proxy-revalidate**
  - il proxy / client deve rivalidare con origin server quando la risorsa in cache diventa stale
- **max-age=secondi**
  - scadenza dei dati
- **s-maxage=secondi**
  - scadenza dei dati in una shared cache
  - ha priorità su max-age (solo per shared cache)
- **nota: max-age e s-maxage hanno priorità su un eventuale header Expires: della risorsa**

HTTP/1.1 general header (II)

- **Trailer: header-name , …**
  - gli header indicati sono in coda al body chunked
  - non possono essere Transfer-Encoding, Content-Length e Trailer
- **Transfer-Encoding: codifica-di-trasferimento**

HTTP/1.1 general header (III)

- **Upgrade: protocollo, …**
  - il client propone di passare ad un protocollo "migliore" (a giudizio del server tra quelli elencati)
- **Via: protocol nodo [commento] , …**
  - sequenza di nodi intermedi (proxy o gateway)
  - nodo = "host [: porta]" oppure uno pseudonimo
- **Warning: codice_di_stato agente testo [data]**
  - tipicamente usato dai proxy per avvisare di condizioni particolari
  - se testo non ISO-8859-1 allora RFC-2047

Codici di stato per i warning

- **110 Response is stale**
  - la risorsa è “vecchia”
- **111 Revalidation failed**
  - impossibile contattare origin server per rivalidare
- **112 Disconnected operation**
  - proxy scollegato dalla rete
- **113 Heuristic expiration**
  - proxy ha scelto euristicamente una scadenza
- **199 Miscellaneous warning**
- **214 Transformation applied**
  - proxy ha cambiato la codifica dei dati trasmessi
- **299 Miscellaneous persistent warning**

HTTP/1.1 entity header (I)

- **Allow: metodo, …**
  - fornito dal server con stato 405 (Method Not Allowed) per individuare i metodi permessi
  - fornito dal client con PUT per suggerire i metodi con cui la risorsa dovrebbe essere accessibile
- **Content-Encoding: codifica**
- **Content-Language: language**
- **Content-Length: num-bytes**
  - numero (in base 10) di byte del body
- **Content-Location: absoluteURI | relativeURI**
  - utile nel caso una risorsa fornita a seguito di una negoziazione sia anche accessibile direttamente

HTTP/1.1 entity header (I)

- **Content-MD5: base64-md5-digest**
  - per proteggere l’integrità dei dati trasferiti (da errori casuali, non da attacchi!)
- **Content-Range: intervallo**
- **Content-Type: mime-type**
- **Expires: http-date**
- **Last-Modified: http-date**

Codici di stato standard HTTP/1.1

100 Continue
101 Switching protocols

200 OK
201 Created
202 Accepted
203 Non-authoritative information
204 No content
205 Reset content
206 Partial content

300 Multiple choices
301 Moved permanently
302 Found (Moved temporarily)
303 See other
304 Not modified

400 Bad request
401 Unauthorized
403 Forbidden
404 Not found

500 Internal server error
501 Not implemented
502 Bad gateway
503 Service unavailable

Gestione dello stato

- **le applicazioni web devono mantenere informazioni sugli utenti quando navigano:**
  - tra pagine diverse
  - in tempi successivi (minuti, ore, giorni)
- **vari motivi:**
  - preferenze utente (es. lingua, dimensione font)
  - dati di business (es. carrello dei prodotti)
- **problema con HTTP:**
  - HTTP/1.0 effettua singole transazioni ed è stateless
  - HTTP/1.1 permette transazioni multiple sulla stessa connessione ma è stateless tra connessioni successive

Possibili metodi per gestire lo stato

- **a livello di applicazione:**
  - tramite URL speciali generate dinamicamente
    - es. www.x.com/basket/12ab34/
  - tramite campi “nascosti” (type=hidden) di un form
    - passati in modo invisibile (metodo POST)
    - passati in modo visibile (metodo GET)
      - es. www.x.com/basket?id=12ab34
- **a livello di protocollo di trasporto:**
  - cookie in HTTP
    - v1 in HTTP/1.0
    - v2 (+ v1) in HTTP/1.1

Perché gestire lo stato in HTTP?

- **gestire lo stato senza modificare le URL**
  - gli utenti possono scambiarsi le URL tra loro (senza per questo scambiarsi lo stato)
  - non si "sporcano" le cache intermedie con
    - dati uguali ma corrispondenti ad URL diverse
    - pagine diverse ma corrispondenti ad URL uguali
- **minimizzare l'impatto sulla configurazione del server e dell'applicazione**
- **associare l'utente allo stato anche se è anonimo (es. non passa da portale di autenticazione)**
- **salvare lo stato in memoria permanente**
  - permane tra reboot, start/stop di UA, cambio di IP

I 4 comandamenti dei cookie

- **è lecito mantenere traccia dello stato di una sessione HTTP se …**
- **1) l'utente ne è conscio e consenziente**
- **2) l'utente può cancellare lo stato quando vuole**
- **3) le informazioni di stato non sono fornite a terzi senza il consenso esplicito dell'utente**
- **4) le informazioni di stato non contengono dati sensibili e non posso essere usate per ottenere dati sensibili**

Cookie

- **scopo: memorizzare informazioni da parte del server in locale sul client in cui "gira" il browser**
- **in questo modo si alleggerisce il server**
- **... ma si introducono elementi di:**
  - aleatorietà (=UA che non supporta/accetta cookie)
  - rischio (=cookie letti/manipolati sul client o in rete)

Definizione dei cookie

- **inizialmente definito da Netscape:**
  - "Persistent client state – HTTP cookies"
- **diventato poi standard IETF:**
  - "HTTP state management mechanism":
  - v1 = RFC-2109
  - v2 = RFC-2965
- **le specifiche Netscape e v1 sono obsolete ma ancora usate**
- **RFC-2964 "Use of HTTP state management" contiene avvertenze su problemi / pericoli dei cookie**

Caratteristiche dei cookie

- **dati di piccole dimensioni registrati sul client (persistent information) e inviati al server**
- **cookie creati dal server sul client tramite header:**
  - Set-Cookie: ... (v1)
  - Set-Cookie2: ... (v2)
- **cookie trasmessi dal client al server tramite header:**
  - Cookie: ...
- **i cookie sono nel formato *nome=valore***
  - salvati localmente dal browser (in un file)
- **dati comunicati al server ai successivi accessi**
- **usati per creare un collegamento logico tra richieste HTTP diverse verso lo stesso server**

Supporto dei cookie nei browser

- **Netscape Navigator**
  - dalla versione 2.0
  - registrati nel file cookies.txt in (default in Netscape\Users\username)
- **Microsoft IE**
  - dalla versione 3.0
  - registrati in \Windows\Cookies o nel profilo utente \Documents and Settings\username\Cookies

Limiti dei cookie

- **limitazioni riguardo il minimo numero di cookie che devono essere trattati**
- **totale minimo per user agent (su disco):**
  - max 300 cookie
  - max 4 kB per cookie
  - totale = 1.2 MB max
- **per server/dominio (su disco dell'UA, nella RAM del server ma solo per gli utenti simultanei):**
  - max 20 cookie / client
  - max 4 kB per cookie
  - totale = 80 kB max / client

Trasmissione dei cookie (v1): S > C

- **cookie inserito nell'header della risposta HTTP**
- **sintassi del cookie:**

```
Set-Cookie:
  cookieName=cookieValue
  [ ; EXPIRES=dateValue ]
  [ ; DOMAIN=domainName ]
  [ ; PATH=pathName ]
  [ ; SECURE ]
```

Cookie: valore, scadenza, secure

- **il nome del cookie ed il relativo valore sono stringhe qualsiasi:**
  - prive di virgola, punto-e-virgola e spazio
  - tali caratteri devono essere espressi in esadecimale (%2C, %3B, %20)
- **EXPIRES: data di scadenza dopo la quale il cookie può essere eliminato dal client**
  - formato rfc-822 Wdy, DD Mon YY HH:MM:SS GMT
  - se non viene specificata una scadenza, il cookie scade alla chiusura del browser (memorizzato non su disco ma in RAM: cookie volatile / temporaneo)
- **SECURE: cookie trasmesso solo su canale HTTPS**

Cookie: dominio, path

- **DOMAIN indica il dominio autorizzato a gestire il cookie (ad es. polito.it)**
  - soltanto le richieste effettuate a tale dominio provocano la trasmissione del cookie
  - se il dominio non è specificato, il browser usa il nome del server che ha mandato il cookie
- **PATH indica il percorso in cui si è autorizzati a usare il cookie (ad es. /didattica/)**
  - soltanto le richieste a URL in tale percorso potranno provocare la trasmissione del cookie
  - se il path non è specificato, il browser usa quello della URL che ha scatenato l'invio del cookie

Trasmissione di un cookie (V1): C > S

- **quando il browser sta per effettuare accesso ad una URL, controlla se esistono cookie associati al dominio e al percorso**
- **in caso affermativo include nell'header HTTP della richiesta tutte le coppie nome/valore dei relativi cookie**
- **se l'URL specificata nella richiesta HTTP è quella di una pagina dinamica (es. CGI o ASP), l'applicazione troverà la stringa dei cookie nella variabile d'ambiente HTTP_COOKIE**

```
Cookie: nome1=valore1; nome2=valore2; ...
```

Esempio (passo 1)

- **cookie usato per gestire la lista dei libri selezionati da un utente che accede ad un negozio virtuale**
- **il client accede al server e riceve:**

```
Set-Cookie:
  customer=john_smith;
  expires=Sat, 28-Aug-99 00:00:00 GMT;
  path=/cgi/bin/;
  domain=books.virtualShopping.com;
  secure
```

Esempio (passo 2)

- **quando il client accede ad una URL nel percorso /cgi/bin invia al server un header che contiene:**
  - Cookie: customer=john_smith
- **il client riceve come risposta:**
  - Set-Cookie: part_number=book1; path=/cgi/bin ; ...
- **il client richiede una URL nel percorso /cgi/bin e manda:**
  - Cookie: customer=john_smith; part_number=book1

Esempio (passo 3)

- **il client riceve:**
  - Set-Cookie: shipping=fedex; path=/cgi/bin/deliver; ...
- **il client accede ad una URL nel percorso /cgi/bin/deliver e manda:**
  - Cookie: customer=john_smith; part_number=book1; shipping=fedex

Problemi dei cookie

- **il meccanismo basato su cookie permette di creare profili degli utenti**
- **user tracking = termine usato per indicare la possibilità di tracciare i siti visitati da un utente e quindi le sue abitudini e i suoi interessi**
- **esempio: se un utente scarica un banner pubblicitario da un sito, oltre a ricevere un'immagine riceve anche un cookie**
- **per tutti i siti appartenenti al circuito è possibile impostare e recuperare il valore dei cookie riguardanti le preferenze di navigazione di un utente**
- **i cookie possono essere disabilitati sul browser**

Problemi dei cookie

- **autenticazione tramite cookie (es. siti commerciali che associano l'utente al carrello virtuale, one-click order) è fortemente sconsigliata:**
  - lettura dei cookie durante la trasmissione e sul client
- **attacchi che permettono di intercettare i cookie:**
  - packet sniffing
  - web spoofing
  - attacchi al client (virus, worm, javascript, ...)
  - sub-domain attack (quale parte del dominio è responsabile del servizio web che usa il cookie?)

Trasmissione dei cookie (v2): S > C

- **nuova sintassi (v2) specificata in RFC-2965**
- **uno o più cookie (lista separata da virgole)**

```
Set-Cookie2: cookieName=cookieValue
   [ ; Comment=commentText ]
   [ ; CommentURL="URL" ]
   [ ; Discard ]
   [ ; Domain=domainName ]
   [ ; Max-Age=dateValue ]
   [ ; Path=pathName ]
   [ ; Port="comma-separated-port-list" ]
   [ ; Secure ]
   [ ; Version=versionNumber ]
```

Nuovi attributi dei cookie v2

- **Comment e/o CommentURL**
  - informano l'utente circa lo scopo del cookie
- **Discard**
  - chiede all'UA di cancellare il cookie quando termina (=cookie volatile, da conservarsi solo in RAM)
- **Max-Age**
  - numero di secondi dopo cui cancellare il cookie
- **Port**
  - porte TCP a cui restituire il cookie
- **Version**
  - numero di versione (attualmente pari a 1)

Cambiamento di significato

- **in v1 l'attributo Secure richiedeva trasmissione solo su HTTPS**
- **in v2 l'attributo Secure è un suggerimento del server:**
  - "trattare in modo sicuro"
  - il client dovrebbe trasmettere il cookie al server tramite un canale con sicurezza non inferiore a quella con cui è stato ricevuto

Trasmissione dei cookie (v2): C > S

- **nuova sintassi (v2) specificata in RFC-2965**
- **uno o più cookie (lista separata da virgole)**

```
Cookie: $Version=cookieVersion ;
  cookieName=cookieValue
  [ ; $Domain=domainName ]
  [ ; $Port="portNumber" ]
  [ ; $Path=pathName ]
```

Request header "Cookie2"

- **quando un client trasmette almeno un cookie avente numero di versione maggiore di quella implementata dal client deve avvisare il server dichiarando la propria versione tramite:**

```
Cookie2: highest-cookie-version-understood
```

HttpOnly

- nuovo attributo dei cookie, supportato dai browser più recenti ma non ancora standard
- rende i cookie non accessibili da JS client-side
- superabile con tecniche più sofisticate