HTTP:
HyperText Transfer Protocol

E' usato dal 1990 come protocollo di trasferimento per il World Wide Web è definito:

"protocollo di livello applicazione per sistemi di informazione distribuiti, collaborativi ed ipermediali" (RFC 2068)

- usa la porta 80 del TCP
- permette di costruire sistemi di accesso all'informazione indipendenti dal tipo dell'informazione stessa.

HTTP: Generalità di funzionamento

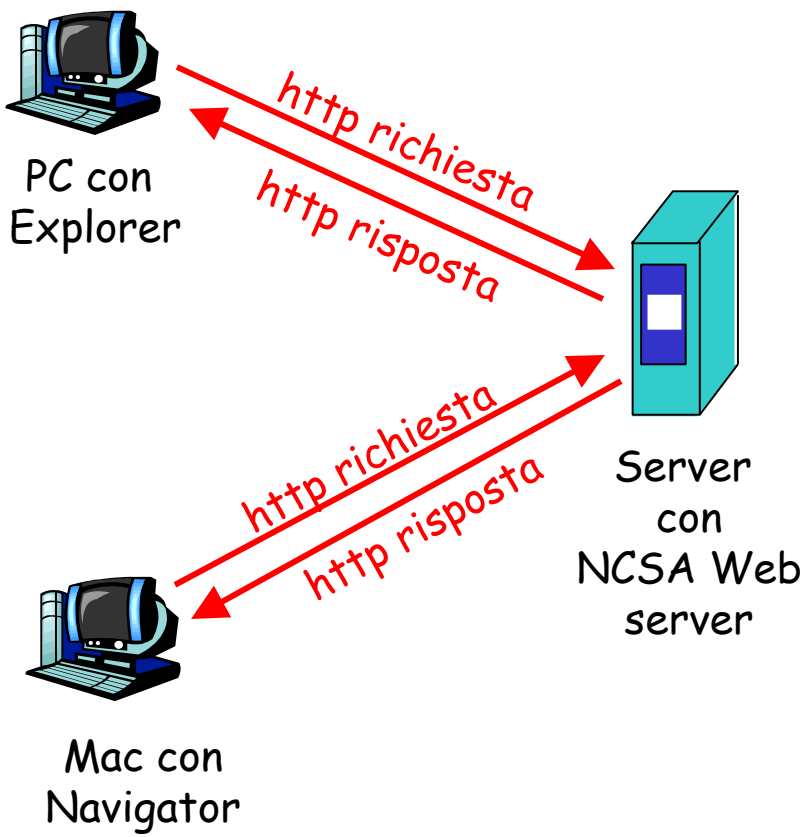
E’ un protocollo di tipo *request/response*

- La connessione viene iniziata dal client, che invia un messaggio di *request.*
- Il server risponde con una *response*
- La connessione viene terminata, oppure (HTTP 1.1) si procede con un’altra coppia di *request/response*.

Il Web: il protocollo http

http: hypertext transfer protocol

- è il protocollo dello strato di applicazione del Web
- modello client/server
  - *client:* browser che richiede, riceve e visualizza gli oggetti Web
  - *server:* Web server che invia oggetti in risposta ad una richiesta
- http1.0: RFC 1945
- http1.1: RFC 2068



Il protocollo http (continua)

http: servizio di trasposto TCP

- il client inizia una connessione TCP (crea una socket) al server, porta 80
- il server accetta la connessione TCP dal client
- i messaggi http (i messaggi del protocollo dello strato applicativo) sono scambiati tra browser (http client) e Web server (http server)
- la connessione TCP viene chiusa

http e "stateless"

- il server non mantiene alcuna informazione sulle richieste passate del client

inciso

I protocolli che mantengono lo "stato" sono complessi!

- La storia passata (stato) deve essere mantenuta
- se il server/client si interrompe, le loro visione degli stati possono essere inconsistenti e devono essere riconciliate

Esempio http

Supponiamo che l'utente digiti l'URL
www.someSchool.edu/someDepartment/home.index

(contiene testo, e riferimenti a 10 immagini jpg)

1a. Il client http inizia la connessione TCP verso il server http (processo) al www.someSchool.edu. Porta 80 è default per il server http .

1b. Il server http dell'host www.someSchool.edu aspetta le richieste di connessione TCP connection alla porta 80. "accetta" la connessione, e lo notifica al client

2. Il client http invia un *messagio di richiesta http* (che contiene lìURL) nel socket di connessione TCP

3. Il server http riceve il msg di richiesta, complila un *messaggio di risposta* che contiene l'oggetto richiesto (someDepartment/home.index), manda il messaggio nella socket

tempo

Esempio http (cont.)

4. Il server http chiude la connessione TCP

5. Il client http riceve il messaggio di risposta che contiene il file html e lo visualizza. Percorrendo il file trova il riferimento a 10 oggetti jpg

tempo

6. Ripete i passaggi da 1-5 per ognuno dei 10 oggetti jpg

Connessioni Non-persistenti e persistenti

Non-persistente

- HTTP/1.0
- il server esamina una request, risponde, e chiude la connessione TCP
- sono necessari 2 RTTs per recuperare ogni oggetto
- ogni trasferimento di oggetto è sottoposto allo slow start

Però molti browsers 1.0 usano connessioni TCP parallele.

Persistente

- default per HTTP/1.1
- sulla stessa connessione TCP : il server esamina la richiesta, risponde, esamina la nuova richiesta, risponde...
- Il Client manda le requests per tutti gli oggetti in riferimento appena riceve il HTML base.
- Sono necessari meno RTTs e meno slow start.

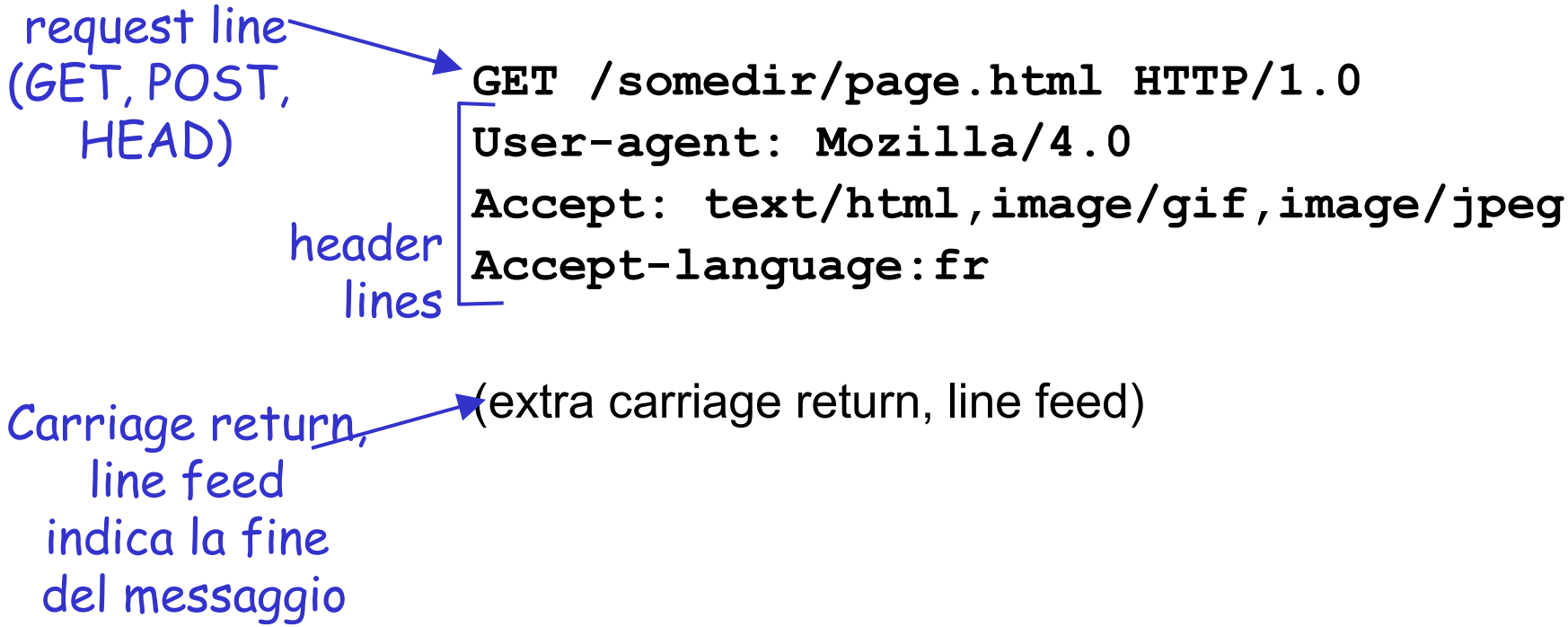
Messaggi HTTP

Un messaggio HTTP può essere di due tipi: *request* o *response*.

Ambedue seguono la struttura di un messaggio di e-mail (RFC 822) per trasferire il messaggio, ma non è necessario assumere che il trasferimento sia 7bit-compatible

http message format: request

- Due tipi di messaggi: *request, response*
- http request message:
  - ASCII (leggibile dalle persone)



http request message: formato generale

HTTP request line

```
Request-Line = Method SP
               Request-URI SP
               HTTP-Version CRLF
```

```
Method  = "OPTIONS" | "GET"
        | "HEAD"    | "POST"
        | "PUT"     | "DELETE"
        | "TRACE"   | extension-method
```

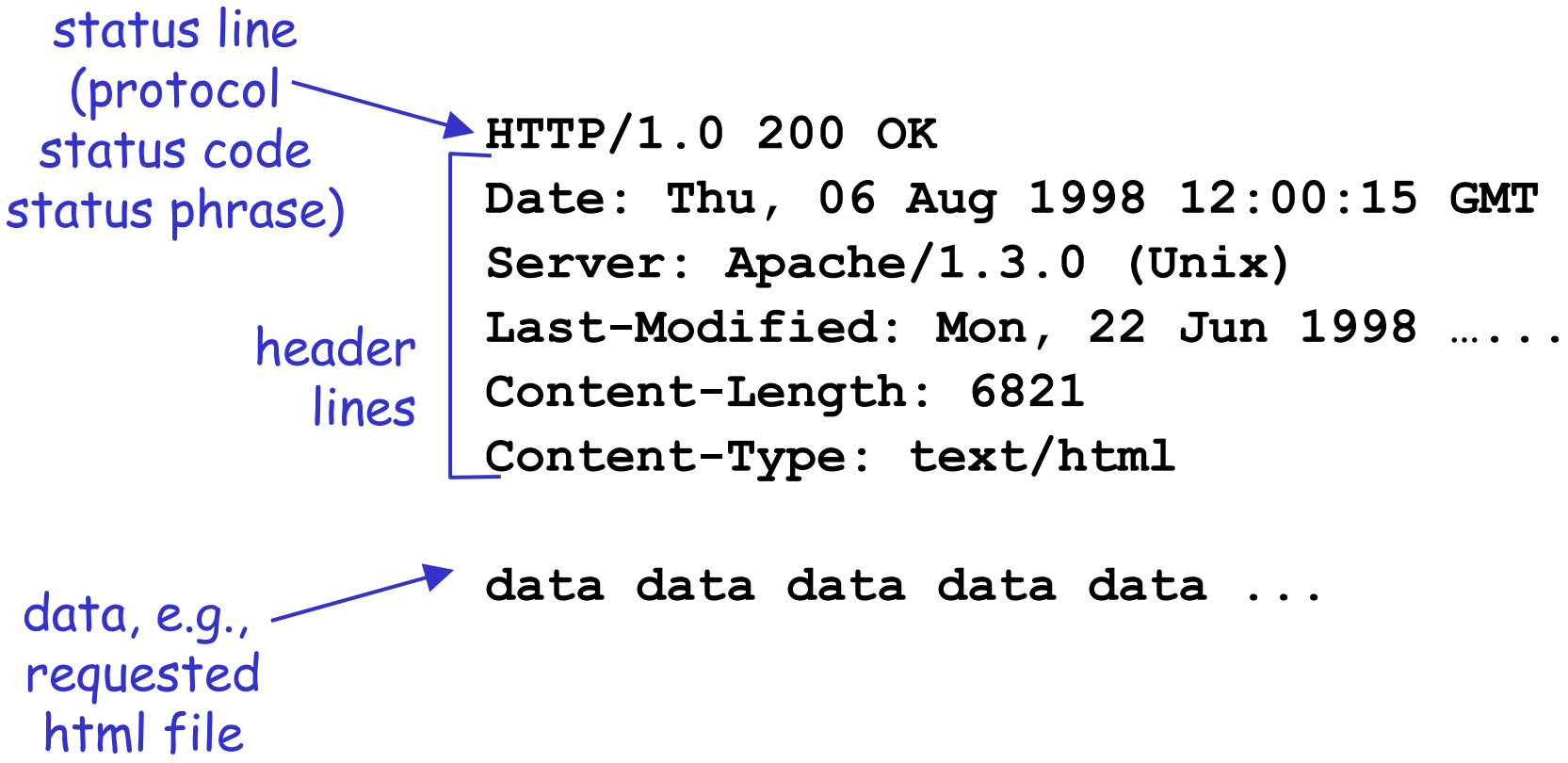
Request headers

```
request-header = Accept                | Accept-Charset
               | Accept-Encoding       | Accept-Language
               | Authorization
               | Proxy-Authorization
               | From                  | Host
               | If-Modified-Since
               | If-Unmodified-Since
               | If-Match              | If-None-Match
               | If-Range
               | Max-Forwards          | Range
               | Referer               | User-Agent
```

Request headers

```
Accept: text/plain; q=0.5, text/html,
        text/x-dvi; q=0.8, text/x-c
Accept-Charset: iso-8859-5,
                unicode-1-1;q=0.8
Accept-Encoding: compress, gzip
```

http message format: respone



http response codici di stato

Nella prima linea del messaggio di risposta server->client.

200 OK

- request succeeded, requested object later in this message

301 Moved Permanently

- requested object moved, new location specified later in this message (Location:)

400 Bad Request

- request message not understood by server

404 Not Found

- requested document not found on this server

505 HTTP Version Not Supported

Codici di stato

"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" - Moved Temporarily	"303" - See Other
"304" - Not Modified	"305" - Use Proxy
"400" - Bad Request	"401" - Unauthorized
"402" - Payment Required	"403" - Forbidden
"404" - Not Found	"405" - Method Not Allowed
"406" - Not Acceptable	"407" - Proxy Authentication Required
"408" - Request Time-out	"409" - Conflict
"410" - Gone	"411" - Length Required
"412" - Precondition Failed	"413" - Request Entity Too Large
"414" - Request-URI Too Large	"415" - Unsupported Media Type
"500" - Internal Server Error	"501" - Not Implemented
"502" - Bad Gateway	"503" - Service Unavailable
"504" - Gateway Time-out	"505" - HTTP Version not supported

Request method - GET (request)

```
GET http://192.168.11.66 HTTP/1.1
host: 192.168.11.66
Connection: close
```

Metodo che richiede il trasferimento di una URL o operazioni associate all'URL stessa.

Sono possibili **conditional get** (header "If-…") o **partial get** (header "Range")

Request method - HEAD (request)

```
HEAD http://192.168.11.66 HTTP/1.1
host: 192.168.11.66
Connection: close
```

Simile al GET, ma non viene trasferito il message body. Utile per controllare lo stato dei documenti (cache refresh).

ESEMPI -1

- Telnet radar.det.unifi.it 80
- GET o HEAD HTTP/1.0
  - HTTP/1.1 200 OK
  - Date: Thu, 06 Jun 2002 15:04:06 GMT
  - Server: Apache/1.3.22 (Unix) (Red-Hat/Linux) mod_ssl/2.8.5 OpenSSL/0.9.6b DAV/1.0.2 PHP/4.0.6 mod_perl/1.26
  - Last-Modified: Thu, 06 Jun 2002 15:04:06 GMT
  - ETag: W/"464b-231b-3e8d51c8"
  - Accept-Ranges: bytes
  - Content-Length: 8987
  - Connection: close
  - Content-Type: text/html

Request method - HEAD (response)

```
HTTP/1.1 200 OK
Date: Sun, 14 May 2000 20:02:41 GMT
Server: Apache/1.3.9 (Unix)  (Red Hat/Linux)
Last-Modified: Tue, 21 Sep 1999 14:46:36 GMT
ETag: "f2fc-799-37e79a4c"
Accept-Ranges: bytes
Content-Length: 1945
Connection: close
Content-Type: text/html
```

Notare la mancanza del message body

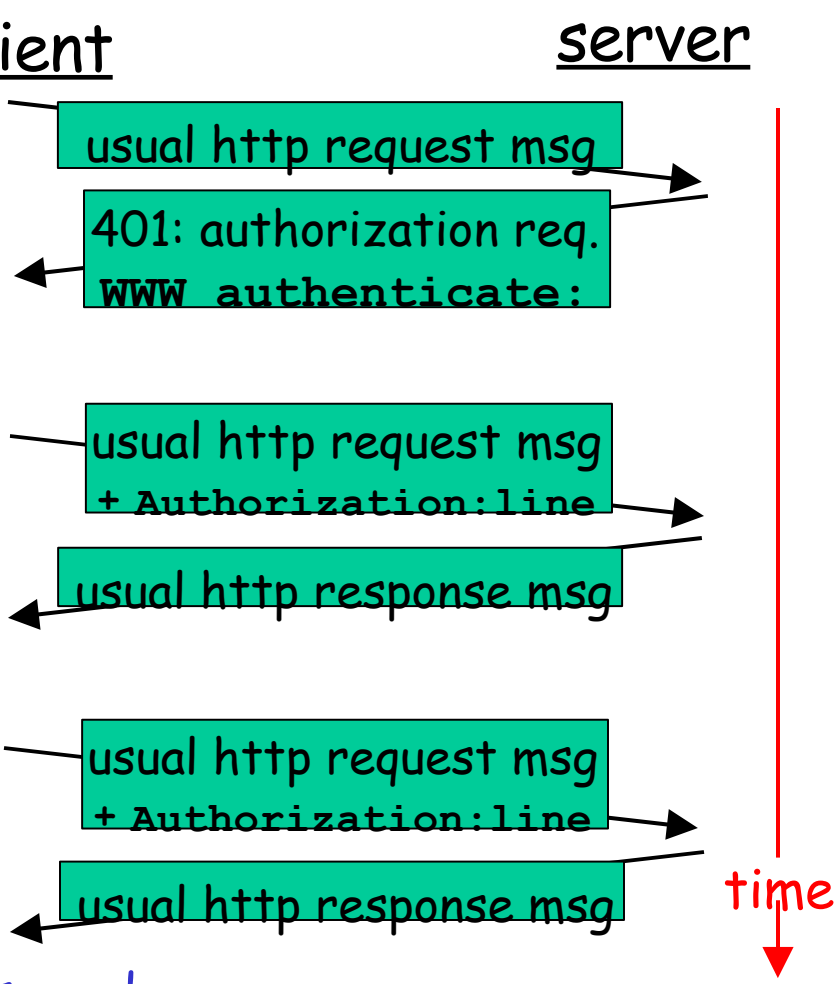
Interazione User-server : l'autenticazione

Scopo dell'autenticazione: controllare l'accesso ai documenti sul server

- stateless: il client deve presentare un'autorizzazione ad ogni richiesta
- autorizzazione: tipicamente nome, password
  - `authorization:` header line nella request
  - se non c'è autorizzazione il server rifiuta l'accesso e dice

    `WWW authenticate:`

    header line nella response



Il Browser mette in cahce nome & password in modo che l'utente non debba reinserirli

Interazione User-server : cookies

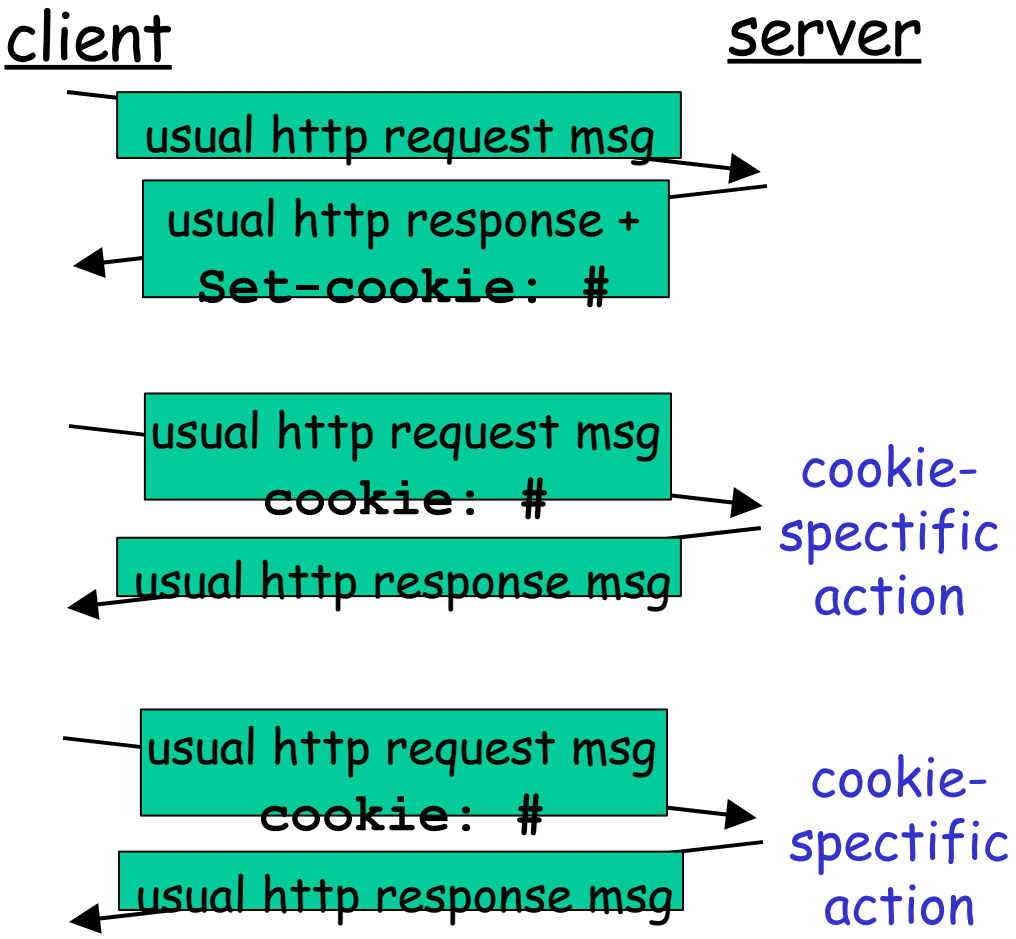
- Il server manda un "cookie" al client nel msg di risposta

  `Set-cookie: 1678453`

- il client presenta il cookie nelle requests successive

  `cookie: 1678453`

- il server associa il cookie alle info immagazzinate sul server
  - autenticazione
  - ricordare le preverenze degli utenti, le scelte precedenti



ESEMPI - 2

- Telnet mmedia5.det.unifi.it 80
- HEAD /quits/admin/index.php HTTP/1.0
  - HTTP/1.1 401 Authorization Required
  - Date: Thu, 06 Jun 2002 15:10:11 GMT
  - Server: Apache/1.3.22 (Unix) (Red-Hat/Linux) mod_ssl/2.8.5 OpenSSL/0.9.6b DAV/1.0.2 PHP/4.0.6 mod_perl/1.26
  - WWW-Authenticate: Basic realm="Area di Amministrazione"
  - Connection: close
  - Content-Type: text/html; charset=iso-8859-1

ESEMPI - 3

- telnet www.amazon.com 80
- HEAD / HTTP/1.0
  - HTTP/1.1 302 Found
  - Date: Thu, 06 Jun 2002 15:13:36 GMT
  - Server: Stronghold/2.4.2 Apache/1.3.6 C2NetEU/2412 (Unix)
  - Set-Cookie: skin=; domain=.amazon.com; path=/; expires=Wed, 01-Aug-01 12:00:00 GMT
  - Location: http://www.amazon.com:80/exec/obidos/subst/home/redirect.html
  - Connection: close
  - Content-Type: text/html; charset=iso-8859-1

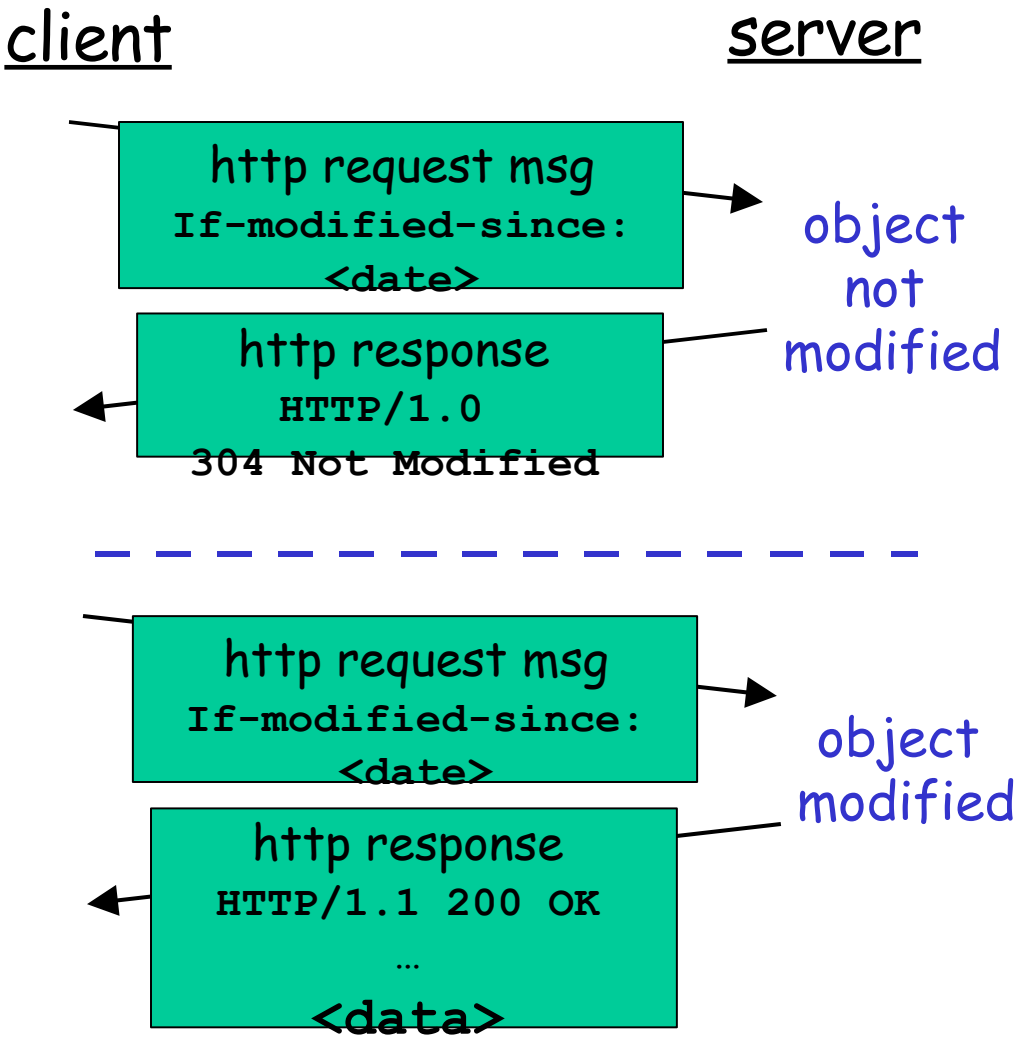
User-server: conditional GET

- Scopo: non mandare gli oggetti se il client ha già immagazzinato la stessa versione dell' l'info (cached)
- client: specifica la data della copia nella cache nel http request

```
If-modified-since:
   <date>
```

- server: la risposta non l'oggetto se la copia cached è aggiornata:

```
HTTP/1.0 304 Not
   Modified
```



Request method - POST

Il metodo POST serve per inviare dal *client* al *server* un'informazione.

Il suo uso è comune nell'invio di:

- Annotazioni ad URL esistenti
- FORMs HTML
- Posting a message boards, newsgroups, mailing list, etc.
- Aggiunte a database

Request method - POST

Il metodo POST, pur essendo una *request*, contiene un *message body*.

E' questo che consente di inviare dati al server

Non effettua necessariamente una lettura di informazioni dal server ma, in risposta ad una POST, il server può anche inviare una *response* con un *message body*
(FORMs concatenate).

Entity headers

```
entity-header  = Allow
               | Content-Base
               | Content-Encoding
               | Content-Language
               | Content-Length
               | Content-Location
               | Content-MD5
               | Content-Range
               | Content-Type
               | ETag
               | Expires
               | Last-Modified
               | extension-header
```

Entity headers

```
Content-Base
  URI assoluta da usare per risolvere le URL
  relative contenute nell'entity body
Content-Encoding
  codifica dell'entity body (es: gzip)
Content-Language
  lingua dell'entity body (es: en, it)
Content-Type
  tipo dell'entity body (es: text/html)
Expires                (utile per caching)
  val. tempor. dell'entity body
Last-Modified          (utile per caching)
  data dell'ultima modifica sul server
```

HTTP: Generalità di funzionamento

E’ un protocollo di tipo *request/response*

- La connessione viene iniziata dal client, che invia un messaggio di *request*.
- Il server risponde con una *response*
- La connessione viene terminata, oppure (HTTP 1.1) si procede con un’altra coppia di *request/response*.

request/response chain

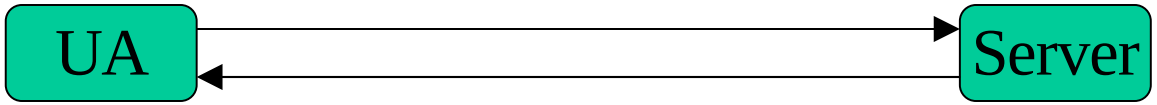
- Il percorso logico che seguono *request* e *response* è detto *chain (catena)*
- Una *chain* può essere modificata da 3 entità di rete con funzionalità diverse:
  - PROXY
  - GATEWAY
  - TUNNEL

HTTP - Proxy, Gateway e Tunnel

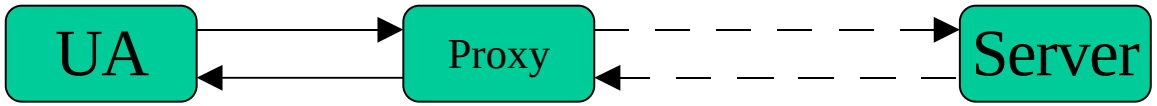
- **PROXY**: entità che *può interrompere* una *chain* e che permette il *caching* delle informazioni. E' *trasparente* all'utente.
- **TUNNEL**: entità che permette il superamento di *firewalls*, è *completamente trasparente*.
- **GATEWAY**: è un punto di accesso alla rete che funziona da ingresso in un'altra rete. Gli host che controllano il traffico entro la rete aziendale o all'ISP locale sono nodi detti gateways. Un una rete aziendale spesso un gateway agisce anche da proxy server e da firewall server

HTTP: Proxy, Gateway e Tunnel

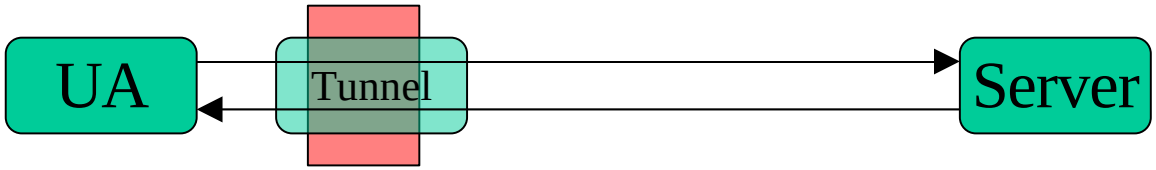
Situazione normale



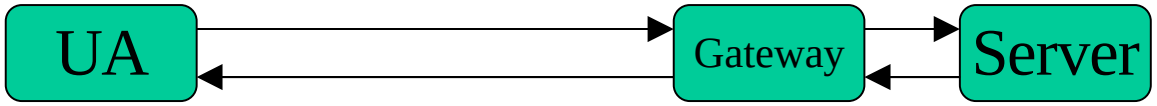
Proxy: l'UA non si accorge se la chain è stata interrotta.



Tunnel: l'UA non si accorge della presenza di un Firewall



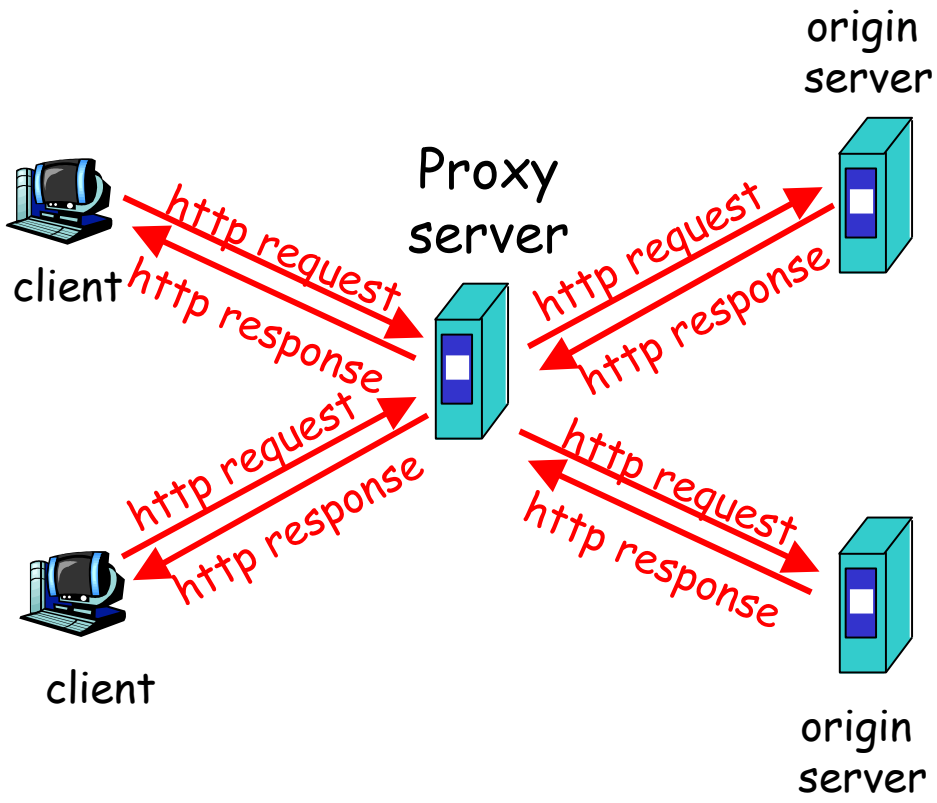
Gateway: l'UA pensa che il server sia il Gateway



Web Caches (proxy server)

Scopo: soddisfare le richieste senza implicare il server originario

- L'utente imposta il browser : l'indirizzo Web viene acceduto via web cache
- il client manda tutte le http requests alla web cache
  - se gli oggetti sono nella web cache, questa invia l'oggetto nella http response
  - altrimenti richiede l'oggetto al server orginario ed invia la risposta al client (e ne immagazzina una copia)



HTTP vs SMPT

- Supponiamo HTTP connessione persistente
- entrambi usano TCP
- HTTP è di tipo pull
- SMTP è di tipo push
- SMTP richiede che tutto il messaggio (incluso il body) sia in ASCII a 7 bit
- HTTP incapsula ogni oggetto in un msg risposta
- SMTP inserisce tutti gli oggetti in un solo messaggio

Riferimenti (Packet analyzer)

Dal Politecnico di Torino:

http://netgroup-serv.polito.it/analyzer/install/default.htm

Nota: Pacchetto in uso gratuito

Efficienza della rete

- HTTP è responsabile della maggioranza del traffico della rete
- Latenza e Larghezza di banda devono essere ancora considerate
- HTTP 1.1 è stato progettato per limitare il sovraccarico del TCP dovuto al HTTP 1.0, ma non il sovraccarico del dovuto al protocollo stesso
- il 90% dell richieste HTTP sono ridondanti
- il modo di comunicazione tra client e server deve essere ancora migliorato

La soluzione?

- Il W3C sta lavorando al HTTP – Next Generation (HTTP-NG)
- proposto inizialmente nel 1997
- propone un nuovo modo di progettare l' HTTP per gestire l'uso futuro del web
- e per fare in modo che
  - il Web continui a funzionare come se avesse l' HTTP
  - che si migliorino le prestazioni
  - che il TCP rimanga alla base dell'architettura di connessione pur essendo compatibile con ogni altro flusso affidabile di dati.
  - …
  - http://www.w3.org/Protocols/HTTP-NG/Activity.html