# Il protocollo HTTP

Corso di ***Applicazioni Telematiche***

A.A. 2005-06 – Lezione n.2

Prof. Roberto Canonico

Università degli Studi di Napoli Federico II

Facoltà di Ingegneria

## Il World Wide Web

- Il *World Wide Web* è un sistema distribuito per la presentazione a schermo di documenti multimediali, e per l'utilizzo di link ipertestuali per la navigazione
- Il sistema è distribuito e scalato su tutta Internet, ed è basato su un'architettura Client/Server
- Il ***client*** o ***browser*** è un visualizzatore di documenti ipertestuali e multimediali
  - Può visualizzare testi, immagini e semplici interfacce grafiche, ma non permette di editare documenti
- Il ***server***, nella sua versione base, è un semplice meccanismo di accesso a risorse locali (*file*) in grado di trasmettere documenti individuati da un identificatore univoco (URL)

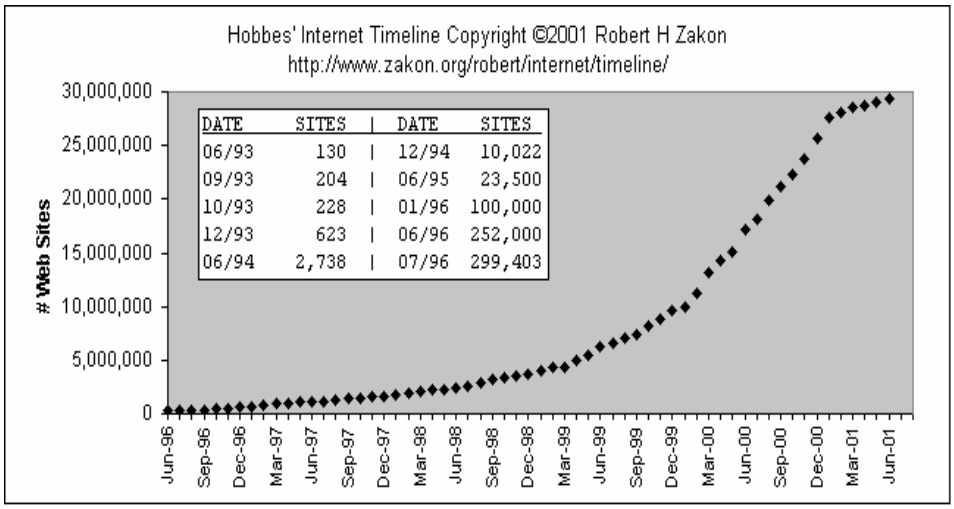
## Storia del WWW (1)

- Nel 1989, un gruppo di ricercatori informatici del CERN (il centro di ricerca in fisica nucleare di Ginevra) ricevette l'incarico da parte della direzione di realizzare un meccanismo per la diffusione rapida di articoli, appunti e opinioni tra i fisici che ruotavano intorno al centro
- Tim Berners-Lee, Robert Cailliau ed altri identificarono Internet, ipertesti e SGML come elementi chiave per questo meccanismo
- Nel 1991, Berners-Lee e Cailliau mostrarono il primo prototipo della loro applicazione, realizzata in client-server su architettura NeXT: World-Wide Web
- Nell'ottobre del 1992 il National Centre for Supercomputing Applications (NCSA) esaminò il prototipo di WWW e decise di realizzarne una versione propria
- Con la realizzazione del server NCSA e del primo browser WWW, chiamato *Mosaic*, l'NCSA decretò l'inizio del successo esplosivo del sistema

## Storia del WWW (2)

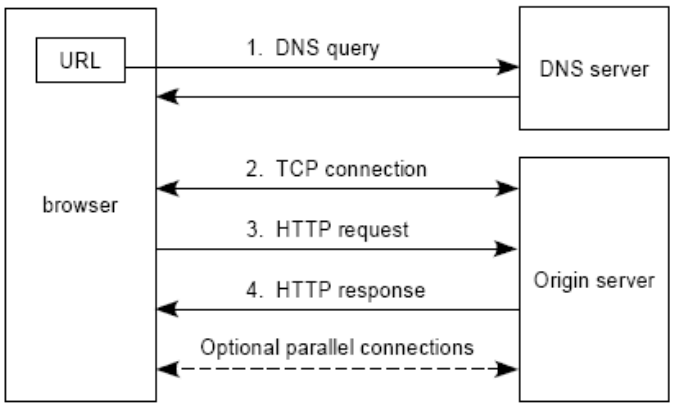
- Marc Andreessen, realizzatore del prototipo di Mosaic su X-Windows, e Jim Clark, ex professore a Berkeley e co-fondatore della Silicon Graphics, fondano nel 1993 la Mosaic Corporation, poi rinominata Netscape Corporation per evitare problemi legali con la NCSA
- La ditta Netscape ha il passaggio più rapido tra la fondazione e la quotazione in borsa della storia, ed una delle quotazioni iniziali di maggior successo
- Nel frattempo, Berners-Lee e Cailliau cercano di mantenere il controllo sull'evoluzione del World Wide Web e fondano il W3C, con fondi della ricerca e dell'università
- Nel marzo 98 Netscape chiede l'aiuto della comunità dei programmatori rilasciando il codice sorgente della versione 5 di Navigator: nasce il progetto mozilla.org
- Microsoft, pur coinvolta in battaglie legali, impone *Internet Explorer* come browser di riferimento in ambito Windows

# Sviluppo del WWW nei primi anni



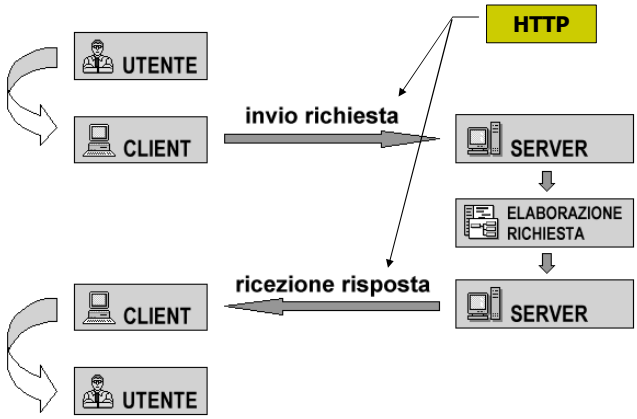

| DATE | SITES | DATE | SITES |
|---|---|---|---|
| 06/93 | 130 | 12/94 | 10,022 |
| 09/93 | 204 | 06/95 | 23,500 |
| 10/93 | 228 | 01/96 | 100,000 |
| 12/93 | 623 | 06/96 | 252,000 |
| 06/94 | 2,738 | 07/96 | 299,403 |

# Cosa succede quando viene richiesta una pagina web



# Il protocollo HTTP

- HTTP, *HyperText Transfer Protocol*, è il protocollo che regola la comunicazione tra un Web browser ed un Web server




# Il protocollo HTTP (2)

- Protocollo di livello applicativo
- Si appoggia su connessioni TCP al livello trasporto
- Usa il modello **client/server**
  - *Client* è il browser che richiede, riceve e "mostra" oggetti Web
  - *Server* è il software "Web server" che invia oggetti in risposta alle richieste
- E' un protocollo generico, **stateless**, che può essere usato anche per scopi diversi dallo scambio di documenti ipertestuali
  - Es. scambio di messaggi in sistemi object-oriented distribuiti
- L'interazione avviene secondo uno **schema richiesta-risposta**
- HTTP è un protocollo ***testuale***, nel senso che l'intestazione dei messaggi è costituita da sequenze di caratteri ASCII
  - Faciltà di implementazione

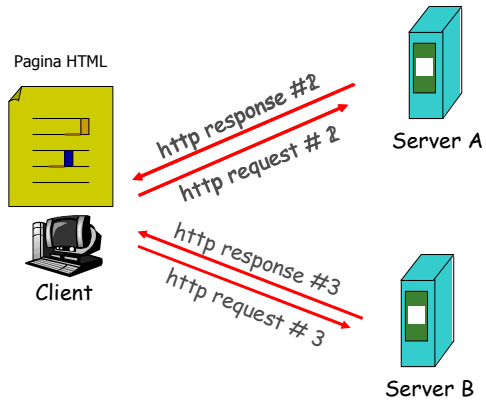
## HTTP: versioni

- http 0.9
- http 1.0 (1996)
  - RFC 1945
- http 1.1 (1997 e 1999)
  - RFC 2068
  - RFC 2616

## HTTP per il trasferimento di pagine web

- Tipicamente, una pagina web è descritta da un file testuale in formato HTML (*Hypertext Markup Language*)
- La pagina è identificata mediante un indirizzo, detto URL
- Un file HTML può contenere riferimenti ad altri oggetti che arricchiscono la pagina con elementi grafici
  - Es. sfondo, immagini, ecc.
- Ciascun oggetto è identificato dal proprio URL
- Questi oggetti possono trovarsi anche su server web diversi
- Una volta ricevuta la pagina HTML, il browser estrae i riferimenti agli altri oggetti che devono essere prelevati e li richiede attraverso una serie di connessioni HTTP

# HTTP per il trasferimento di pagine web (2)



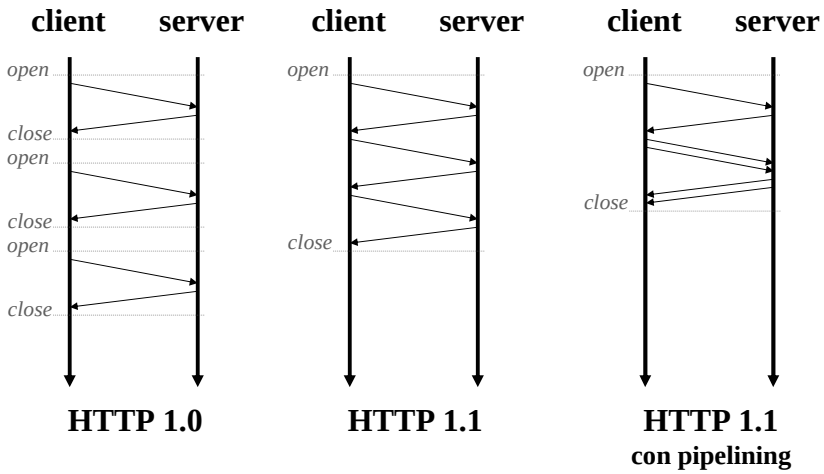

# URL

- Un URL HTTP ha la seguente sintassi:
  **`http://`***`host`*`[:`*`port`*`]/`*`path`*`[#`*`fragment`*`][?`*`query`*`]`
- Host identifica il server
  - Può essere sia un nome simbolico che un indirizzo IP in notazione dotted decimal
- Port è opzionale; di default è 80
- Path identifica la risorsa sul server
  - es: images/sfondo.gif
- #fragment identifica un punto preciso all'interno di un oggetto
  - es: #paragrafo1
- ?query è usato per passare informazioni dal client al server
  - es: dati inseriti nei campi di una form

# La connessione HTTP

- Lo scambio dei messaggi HTTP avviene tramite una connessione TCP (trasporto affidabile di sequenze di byte)
- Di norma, un server web è in attesa di connessioni sul porto TCP 80
- Per connessione HTTP si intende una serie di richieste ed una serie corrispondente di risposte scambiate sulla stessa connessione TCP
- HTTP 1.0 ed 1.1 differiscono principalmente per il modo con cui gestiscono lo scambio di messaggi su una connessione TCP
- La differenza principale è che in HTTP 1.1 è possibile scambiare coppie multiple di richiesta e risposta nella stessa connessione
- Le richieste possono anche essere messe in pipeline (cioè si può trasmettere una nuova richiesta prima che sia arrivata la risposta precedente), ma le risposte debbono essere date nello stesso ordine delle richieste, poiché non è specificato un metodo esplicito di associazione

# La connessione HTTP (2)

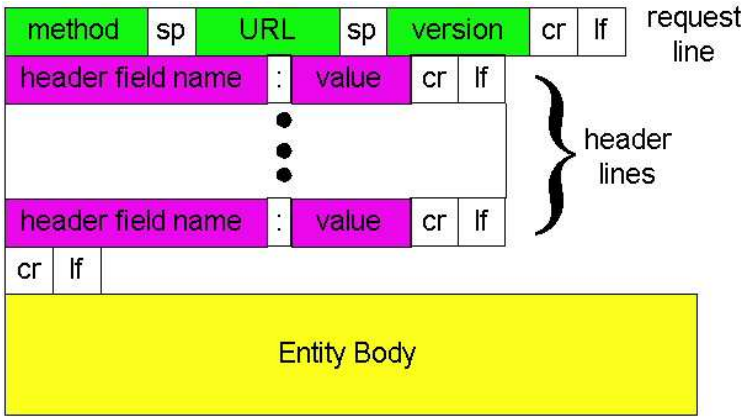


# HTTP: Request

- La richiesta è un messaggio testuale formato da una riga di richiesta e da dati ulteriori facoltativi
- La richiesta semplice è:
  **GET URI CrLf**
- La richiesta completa è:
  **Method URI VersionCRLF**
  **[Header]**
  **CRLF**
  **[Body]**

  dove:
  […] indica un elemento opzionale
  CRLF indica la sequenza di caratteri di codice ASCII
  13 (base 16) = 19 (base 10) → CR = Carriage Return
  10 (base 16) = 16 (base 10) → LF = Line Feed

# HTTP: Request (2)



## HTTP: Request (2)

- La richiesta semplice è stata introdotta nella versione 0.9 (la prima versione di HTTP) e ne è ancora obbligata l'implementazione
- La presenza o meno di **Version** nella linea di richiesta fanno capire al server se si può direttamente creare la risposta o se è necessario attendere altri dati
  - **Version** è "HTTP/1.0" o "HTTP/1.1"
- **Method** può essere:
  - GET (fin da HTTP 0.9)
  - HEAD, PUT, POST, LINK, UNLINK, DELETE (da HTTP 1.0)
  - OPTIONS, TRACE (da HTTP 1.1)
- **URL** è un identificativo di risorsa locale al server
- **Header** sono linee nel formato `header_name: valueCRLF` classificabili come header generali, header di entità, header di richiesta e header di risposta
- **Body** è un messaggio MIME (RFC 2045, RFC 2046, RFC 2047)

## HTTP Request: esempio di GET

```
GET / HTTP/1.1
Accept: image/gif, image/jpeg, image/png, */*
Accept-Language: it
Accept-Encoding: gzip, deflate
Host: www.grid.unina.it
If-Modified-Since: Fri, 14 Mar 2003 10:54:03 GMT
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
```

## Gli header

- Gli header sono righe conformi ad RFC822 che specificano caratteristiche
  - generali della trasmissione
  - dell'entità trasmessa
  - della richiesta effettuata
  - della risposta generata
- Formato generale:
  `header_name: valueCRLF`

## Gli header della richiesta

- Gli header della richiesta sono posti dal client per specificare al server informazioni sulla richiesta e su se stesso
  - **User-Agent**: una stringa che descrive il client che origina la richiesta
    - Di solito indica tipo, versione e sistema operativo del client
  - **Referer**: l'URL della pagina mostrata all'utente nel momento in cui si è originata la richiesta
    - Se l'URL è richiesto con altri metodi che non l'attraversamento di un link, Referer deve essere assente
    - Utile per consentire al server di tracciare la navigazione del client
  - **Host**: il nome di dominio e la porta a cui viene fatta la connessione
    - Permette di realizzare più siti web sulla stessa macchina server
    - Obbligatorio

## Gli header della richiesta (2)

- **Accept, Accept-Charset, Accept-Encoding, Accept-Language**: Implementazione della negoziazione del formato, per quel che riguarda tipo MIME, codice caratteri, codifica MIME, linguaggio umano
  - Il client specifica cosa è in grado di accettare, e il server fa del suo meglio per accontentarlo
- **If-Modified-Since, If-Match, If-None-Match, If-Range, If-Unmodified-Since**: richieste condizionali che vanno soddisfatte solo se la condizione indicata è vera
- **Authorization, Proxy-Authorization**: una stringa di autorizzazione per l'accesso alla risorsa richiesta (login/password)

## Il metodo GET

- Il più importante (ed unico in v. 0.9) metodo di HTTP è GET
- Usato per richiedere una risorsa ad un server
- Questo è il metodo più frequente, ed è quello che viene attivato facendo click su un link ipertestuale di un documento HTML, o specificando un URL nell'apposito campo di un browser
- GET può essere:
  - **assoluto**
    - la risorsa viene richiesta senza altre specificazioni
  - **condizionale**
    - si richiede la risorsa se è soddisfatto un criterio indicato negli header
      **If-match**, **If-modified-since**, **If-range**, ecc.
  - **parziale**
    - si richiede una sottoparte di una risorsa memorizzata

## Il metodo HEAD

- Simile al metodo GET, ma il server deve rispondere soltanto con gli header relativi, senza il corpo
- Usato per verificare:
  - **la validità di un URI**
    - la risorsa esiste e non è di lunghezza zero
  - **l'accessibilità di un URI**
    - la risorsa è accessibile presso il server, e non sono richieste procedure di autenticazione del documento
  - **la coerenza di cache di un URI**
    - la risorsa non è stata modificata nel frattempo, non ha cambiato lunghezza, valore hash o data di modifica

## Il metodo POST

- Il metodo POST serve per trasmettere delle informazioni dal client al server, ma senza la creazione di una nuova risorsa
- POST viene usato per esempio per sottomettere i dati di una form HTML ad un'applicazione sul server
- I dati vengono trasmessi nel body della richiesta
- 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 sottomossi alla risorsa specificata; non è stata data risposta

## Il metodo PUT

- Il metodo PUT serve per trasmettere delle informazioni dal client al server, creando o sostituendo la risorsa specificata
  - Esempio: upload di un file
- In generale, l'argomento del metodo PUT è la risorsa che ci si aspetta di ottenere facendo un GET in seguito con lo stesso nome

## HTTP: Response

- La risposta HTTP è un messaggio testuale formato da una riga iniziale, da header facoltativi ed eventualmente un body (corpo)

  **Version status-code reason-phrase CRLF**
  **[Header]**
  **CRLF**
  **Body**

  dove:
  […] indica un elemento opzionale
  CRLF indica la sequenza di caratteri di codice ASCII
  13 (base 16) = 19 (base 10) → CR = Carriage Return
  10 (base 16) = 16 (base 10) → LF = Line Feed

# HTTP: Response (2)

Esempio:

**HTTP/1.1 200 OK**
**Date: Thu, 10 Apr 2003 11:46:53 GMT**
**Server: Apache/1.3.26 (Unix) PHP/4.0.3pl1**
**Last-Modified: Wed, 18 Dec 2002 12:55:37 GMT**
**Accept-Ranges: bytes**
**Content-Length: 7394**
**Content-Type: text/html**

**<HTML> … </HTML>**

# Status code

- Lo status code è 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 sono 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

## Status code: esempi

- **100 Continue**
  - se il client non ha ancora mandato il body
- **200 Ok**
  - GET con successo
- **201 Created**
  - PUT con successo
- **301 Moved permanently**
  - URL non valida, il server conosce la nuova posizione
- **400 Bad request**
  - errore sintattico nella richiesta
- **401 Unauthorized**
  - manca l'autorizzazione
- **403 Forbidden**
  - richiesta non autorizzabile
- **404 Not found**
  - URL errato
- **500 Internal server error**
  - tipicamente un programma in esecuzione sul server ha generato errore
- **501 Not implemented**
  - metodo non conosciuto dal server

## Gli header di risposta

- Gli header della risposta sono posti dal server per specificare informazioni sulla risposta e su se stesso al client
  - **Server**: una stringa che descrive il server: tipo, sistema operativo e versione
  - **Accept-ranges**: specifica che tipo di range può accettare (valori previsti: byte e none)
  - **WWW-Authenticate**: l'header di WWW-Authenticate include una challenge (codice di partenza) con cui il meccanismo di autenticazione deve fare match in caso di una risposta 401, (unauthorized). Il client genererà con questo valore un valore di autorizzazione posto nell'header *Authorization* della prossima richiesta.

## Gli header generali

- Gli 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
- **Date**: data ed ora della trasmissione
- **MIME-Version**: la versione MIME usata per la trasmissione (sempre 1.0)
- **Transfer-Encoding**: il tipo di formato di codifica usato per la trasmissione
- **Cache-Control**: il tipo di meccanismo di caching richiesto o suggerito per la risorsa
- **Connection**: il tipo di connessione da usare
  - Connection: Keep-Alive → tenere attiva dopo la risposta
  - Connection: Close → chiudere dopo la risposta
- **Via**: usato da proxy e gateway

## Gli header dell'entità

- Gli header dell'entità danno informazioni sul body del messaggio, o, se non vi è body, sulla risorsa specificata
- **Content-Type**: il tipo MIME dell'entità acclusa
  - Specifica se è un testo, se un'immagine GIF, un'immagine JPG, un suono WAV, un filmato MPG, ecc…
  - Obbligatorio in ogni messaggio che abbia un body
- **Content-Length**: la lunghezza in byte del body
  - Obbligatorio, soprattutto se la connessione è persistente
- **Content-Base, Content-Encoding, Content-Language, Content-Location, Content-MD5, Content-Range**: l'URL di base, la codifica, il linguaggio, l'URL della risorsa specifica, il valore di digest MD5 e il range richiesto della risorsa
- **Expires**: una data dopo la quale la risorsa è considerata non più valida (e quindi va richiesta o cancellata dalla cache)
- **Last-Modified**: la data e l'ora dell'ultima modifica
  - Serve per decidere se la copia posseduta (es. in cache) è ancora valida o no
  - Obbligatorio se possibile

## Autenticazione (1)

- Quando si vuole accedere ad una risorsa su cui esistono restrizioni di accesso, il server richiede l'autenticazione dell'utente.
- Al GET viene fornita la risposta 401 (unauthorized), più un header WWW-Authenticate che specifica i criteri con cui autenticarsi (metodo e parametri da usare).
- HTTP ha due metodi di autenticazione:
  - Basic authentication (introdotto in HTTP 1.0)
  - Digest access authentication (introdotto in HTTP 1.1)

## Autenticazione (2)

- Basic authentication
  - Introdotto da HTTP 1.0
  - L'header della prima risposta WWW-Authenticate contiene il realm dell'autenticazione
  - Il client richiede le informazioni di autorizzazione all'utente
  - Successivamente, il client crea una nuova richiesta GET e fornisce le informazioni di autorizzazione codificate in Base64
  - Il browser continua a mandare lo stesso header per tutte le pagine dello stesso realm
  - **Problema: La password passa in chiaro sulla rete**

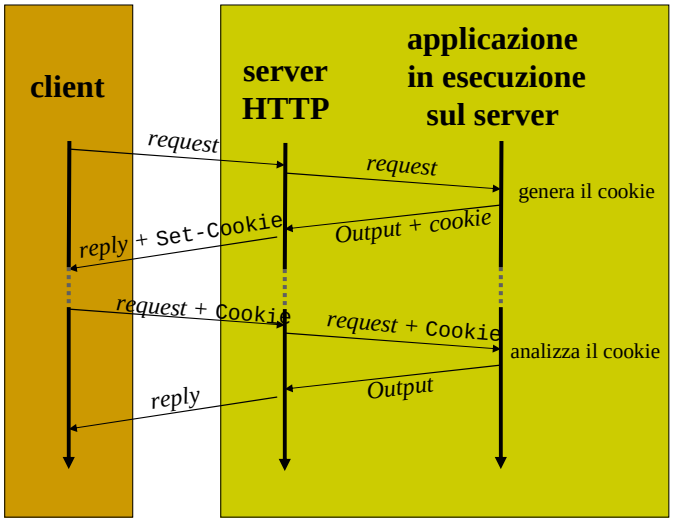
## Autenticazione (3)

- Digest access authentication
  - Introdotto da HTTP 1.1, descritto in RFC 2069
  - Non manda la password in chiaro, ma una fingerprint della password, ovvero la password crittografata con il metodo MD5 (RFC 1321)
  - Per evitare l'abuso della password, anche se crittografata, insieme alla fingerprint vengono codificate anche informazioni come lo username, il realm, l'URI richiesto, una time stamp, ecc.)
  - Ovviamente non risolve il problema di fare arrivare al server la password la prima volta!

## I cookies

- **HTTP è stateless**: il server non è tenuto a mantenere informazioni su connessioni precedenti
- Un cookie è una breve informazione scambiata tra il server ed il client che serve a mantenere un'informazione di stato tra una richiesta e le successive
- Tramite un cookie il client mantiene lo stato di precedenti connessioni, e lo manda al server di pertinenza ogni volta che richiede un documento
- Esempio: tramite un cookie si viene rediretti sulla pagina in Italiano tutte le volte che ci si ricollega allo stesso server
- I cookies furono inizialmente proposti da Netscape e successivamente definiti in RFC 2109 ed in RFC 2965

# Cookies (2)




# Cookies: header specifici

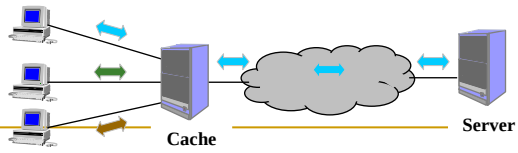
- I cookies dunque usano due header: uno per la risposta, ed uno per le richieste successive:
  - **Set-Cookie**: header della risposta
    - il client può memorizzarlo (se vuole) e rispedirlo alla prossima richiesta
  - **Cookie**: header della richiesta
    - il client decide se spedirlo sulla base del nome del documento, dell'indirizzo IP del server, e dell'età del cookie
- Un browser può essere configurato per accettare o rifiutare i cookies
- Alcuni siti web richiedono necessariamente la capacità del browser di accettare i cookies

# Tipi di proxy HTTP

- In generale un proxy si pone come intermediario tra client e server e decide se e come rispondere al client
- Proxy cache
  - Richieste multiple agli stessi URL possono essere salvate in una locazione intermedia per una maggiore efficienza nella gestione delle risposte
- Proxy di filtro
  - Esigenze di sicurezza o di controllo degli abusi di una rete possono richiedere l'effettiva esecuzione della richiesta solo in certi casi, e altrimenti la risposta con un generico messaggio di mancata autorizzazione.
- Intermediari
  - Un proxy trasparente esegue tutte le richieste e fornisce tutte le risposte, ma in certi casi può convertire o modificare la risposta.
  - Ad esempio fornire link a vocabolari, togliere i banner, convertire i formati ignoti, ecc.
  - Ad esempio, WBI di IBM (**http://www.almaden.ibm.com/cs/wbi/**)

# Web caching

- Si parla genericamente di Web caching quando le richieste di un determinato client non raggiungono il Web Server ma vengono intercettate da una proxy cache
- Tipicamente, un certo numero di client di una stessa rete condividono una stessa cache web, posta nelle loro prossimità (es. nella stessa LAN)
- Se l'oggetto richiesto non è presente nella cache, questa lo richiede *in vece* del client conservandone una copia per eventuali richieste successive
- Richieste successive alla prima sono servite più rapidamente
- Due tipi di interazione HTTP: client-cache e cache-server



## Caching (1)

- Può essere client-side, server-side o intermedia (su un proxy)
- La cache server-side riduce i tempi di computazione di una risposta, ma non ha effetti sul carico di rete
- Le altre riducono il carico di rete
- HTTP 1.0 si basava su tre header:
  - **Expires**: il server specifica la data di scadenza di una risorsa
  - **If-Modified-Since**: il client richiede la risorsa solo se modificata dopo il giorno X. Richiede una gestione del tempo comune tra client e server
  - **Pragma: no-cache**: Fornita dal server, istruisce il client di non fare cache della risorsa in ogni caso
- HTTP 1.1 introduce due tipi di cache control:
  - Server-specified expiration
  - Heuristic expiration

## Caching (2)

- Server-specified expiration
  - Il server stabilisce una data di scadenza della risorsa, con l'header Expires o con la direttiva max-age nell'header Cache-Control
  - Se la data di scadenza è già passata, la richiesta deve essere rivalidata. Se la richiesta accetta anche risposte scadute, o se l'origin server non può essere raggiunto, la cache può rispondere con la risorsa scaduta ma con il codice 110 (Response is stale)
  - Se Cache-Control specifica la direttiva must-revalidate, la risposta scaduta non può mai essere rispedita. In questo caso la cache deve riprendere la risorsa dall'origin server. Se questo non risponde, la cache manderà un codice 504 (Gateway time-out)
  - Se Cache-Control specifica la direttiva no-cache, la richiesta deve essere fatta sempre all'origin server

## Caching (3)

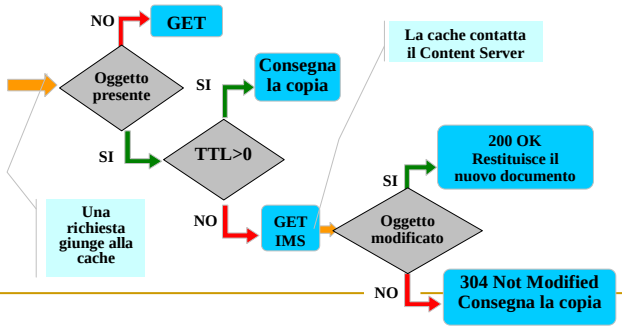
- Heuristic expiration
  - Poiché molte pagine non conterranno valori espliciti di scadenza, la cache stabilisce valori euristici di durata delle risorse, dopo le quali assume che sia scaduta
  - Queste assunzioni possono a volte essere ottimistiche, e risultare in risposte scorrette

## Caching (4)

- 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
  - Un modo semplice per fare validazione è usare HEAD: il client fa la richiesta, e verifica la data di ultima modifica. Ma questo richiede una richiesta in più sempre
  - Un modo più corretto è fare una richiesta condizionale: se la risorsa è stata modificata, viene fornita la nuova risorsa normalmente, altrimenti viene fornita la risposta 304 (not modified) senza body della risposta. Questo riduce il numero di richieste

# Gestione della coerenza

- Problema: cosa succede se l'oggetto presente nel server è aggiornato ?
- La copia in cache deve essere aggiornata per mantenersi uguale all'originale
- HTTP fornisce due meccanismi per la gestione della coerenze:
  - TTL (Time To Live) : il server quando fornisce un oggetto dice anche quando quell'oggetto "scade" (header *Expires*)
    - Quando TTL diventa < 0, non è detto in realtà che l'oggetto sia stato realmente modificato
  - Il client può fare un ulteriore controllo mediante una GET condizionale (*If-Modified-Since*)




# Domande?