

Il protocollo HTTP



Moreno Marzolla
INFN Sezione di Padova
moreno.marzolla@pd.infn.it
<http://www.dsi.unive.it/~marzolla>

Ringraziamenti

- Parte del materiale presentato è tratto dal Corso di Tecnologie Web, prof. Fabio Vitali, Università di Bologna

Introduzione

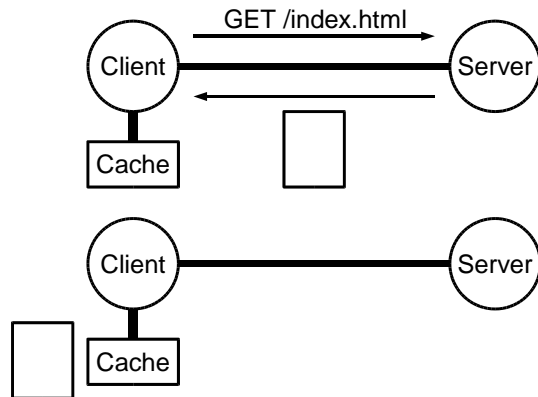
- HTTP é un protocollo client-server generico e stateless utilizzato per il trasporto di risorse del WEB

Caratteristiche di HTTP

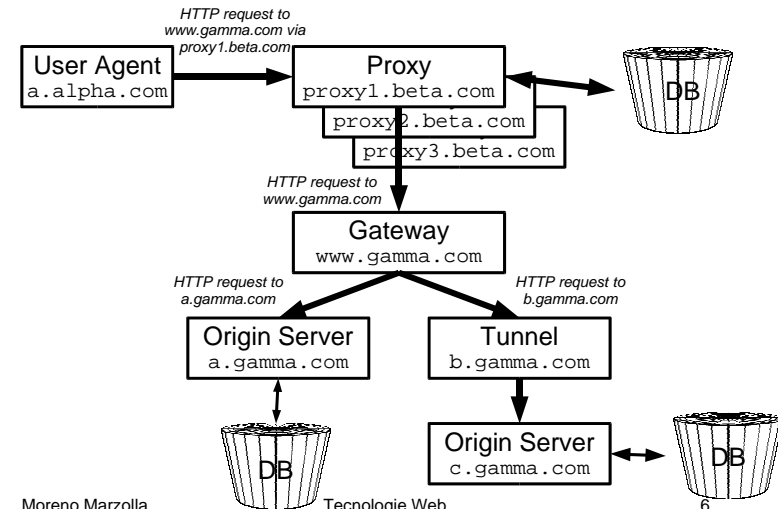
- Client-server
 - In HTTP esistono due ruoli specifici: il client attiva la connessione e richiede dei servizi. Il server accetta la connessione, nel caso identifica il richiedente, e risponde alla richiesta. Alla fine chiude la connessione.
- Protocollo generico
 - HTTP è indipendente dal formato dati con cui vengono trasmesse le risorse. Può funzionare per documenti HTML come per binari, eseguibili, oggetti distribuiti o altre strutture dati più o meno complicate.
- *Statelessness*
 - Il server non è tenuto a mantenere informazioni che persistano tra una connessione e la successiva sulla natura, identità e precedenti richieste di un client. Il client è tenuto a ricreare da zero il contesto necessario al server per rispondere.

Il protocollo HTTP

- Schema generale di una richiesta



Lo scenario tipico



Definizioni

- User agent
 - Il client che inizia una richiesta HTTP (tipicamente un browser, ma può anche essere un bot).
 - Un bot (abbreviazione di robot) è un'applicazione automatica che richiede e scarica pagine HTML e siti web per scopi vari: indicizzazione, catalogazione, verifica di correttezza sintattica, etc. E' uno user agent anche se non vi sono utenti che serve.
- Origin server
 - il server che possiede fisicamente la risorsa richiesta (è l'ultimo della catena)

Definizioni / 2

- Proxy
 - Applicazione intermedia che agisce sia da client che da server. Le richieste sono soddisfatte autonomamente, o passandole ad altri server, con possibile trasformazione, controllo, verifica.
- Gateway
 - Applicazione che agisce da intermediario per qualche altro server. A differenza del proxy, il gateway riceve le richieste come fosse l'origin server: il client può non essere al corrente che si tratta del gateway.
- Tunnel
 - Un programma intermedio che agisce da trasmettitore passivo di una richiesta HTTP. Il tunnel non fa parte della comunicazione HTTP, anche se può essere stato attivato da una connessione HTTP.

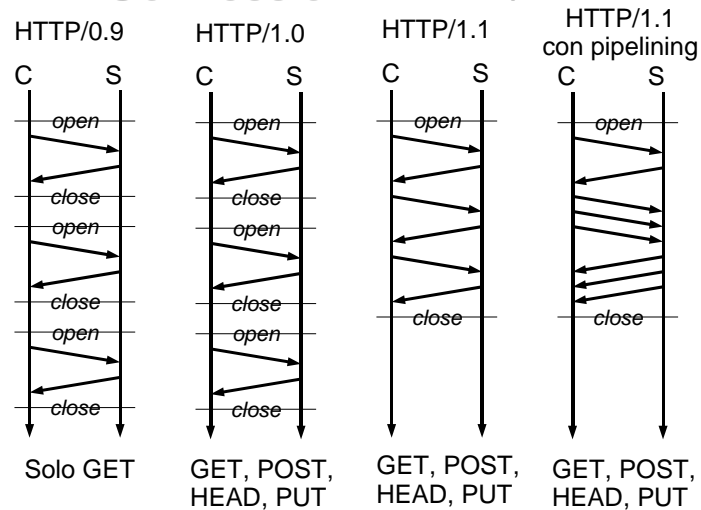
Proxy HTTP

- Un proxy si pone come intermediario tra client e server e decide se e come rispondere al client. I
- Proxy trasparenti (non cambiano la risposta)
 - **Proxy di cache:** Richieste multiple agli stessi URL possono essere salvate in una locazione intermedia per una maggiore efficienza nella gestione delle risposte
 - **Proxy di filtro:** La richiesta è eseguita solo in certi casi (per ragioni di sicurezza o controllo degli abusi), altrimenti la risposta è un generico messaggio di mancata autorizzazione.
- Proxy non trasparenti (cambiano la risposta)
 - 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. (vedi proxyoxy)

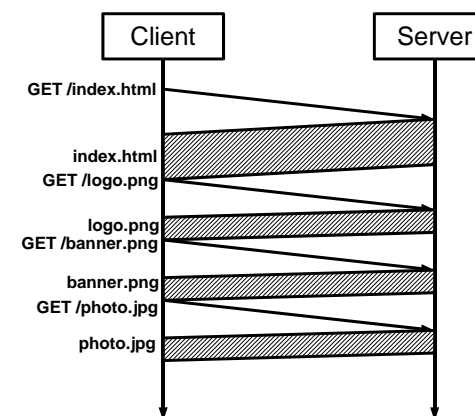
Connessioni HTTP / 1

- La connessione HTTP è composta da una serie di richieste ed una serie corrispondente di risposte.
- La differenza principale tra HTTP 1.0 e 1.1 è la possibilità di specificare coppie multiple di richiesta e risposta nella stessa connessione.
- Le richieste possono essere messe in pipeline, ma le risposte debbono essere date nello stesso ordine delle richieste, poiché non è specificato un metodo esplicito di associazione.

Connessioni HTTP / 2



Schema HTTP/1.0



Connessioni persistenti

- Le connessioni persistenti hanno diversi vantaggi:
 - Richiedono meno connessioni TCP, con vantaggio per le CPU e per la rete
 - Permettono di ridurre l'attesa della visualizzazione
 - Permettono di gestire in maniera migliore gli errori
- Il pipelining è la trasmissione di più richieste senza attendere l'arrivo della risposta alle richieste precedenti
 - Riduce ulteriormente i tempi di latenza, ottimizzando il traffico di rete, soprattutto per richieste che riguardano risorse molto diverse per dimensioni o tempi di elaborazione
 - E' fondamentale che le risposte vengano date nello stesso ordine in cui sono state fatte le richieste (HTTP non fornisce un meccanismo di riordinamento esplicito)

Struttura di una richiesta

```
<METHOD> <URL> <HTTPVERSION>\r\n
<HEADERNAME>: <HEADERVERVAL>\r\n
<HEADERNAME>: <HEADERVERVAL>\r\n
...
\r\n
<DATA, IF POST>
```

```
GET / HTTP/1.1
Host: www.amazon.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.3) Gecko/20041007 Gal
eon/1.3.18 (Debian package 1.3.18-1.1)
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Cookie: skin=; session-id-time=1109232000; session-id=102-0987903-6592124;
obidos_path_continue-shopping=continue-shopping-url=/subst/home/home.html/102-0987903-
6592124&continue-shopping-post-data=&continue-shopping-description=generic.gateway.default;
ubid-main=077-6286623-0470965
Connection: close
```

Struttura di una risposta

```
<HTTPVERSION> <STATUS CODE> <MSG>\r\n
<HEADERNAME>: <HEADERVERVAL>\r\n
<HEADERNAME>: <HEADERVERVAL>\r\n
...
\r\n
<DATA, IF NECESSARY>
```

```
HTTP/1.1 301 Moved Permanently
Date: Thu, 17 Feb 2005 09:09:50 GMT
Server: Stronghold/2.4.2 Apache/1.3.6 C2NetEU/2412 (Unix) amarewrite/0.1
mod_fastcgi/2.2.12
Set-Cookie: skin=; domain=.amazon.com; path=/;
Location: http://www.amazon.com:80/exec/obidos/subst/home/home.html
Transfer-Encoding: chunked
Content-Type: text/plain
Connection: close
```

HTTP/1.0 Metodi

- GET
 - Richiede un documento
- POST
 - invia i dati di un form
- HEAD
 - Simile a GET, ma ritorna solo l'header HTTP e non i dati richiesti. Utile per verificare la presenza di una pagina, o per la gestione della cache lato client
- PUT, DELETE, LINK, UNLINK
 - Non usati, grossi problemi di sicurezza

HTTP/1.0 Headers / 1

- **Authorization**
 - Inviato dal client
 - **Authorization: <credentials>**
 - “Basic Auth” is commonly used
 - <credentials> = Base64 (username:password)
 - Ha senso se all'interno di una connessione SSL
- **Content-Encoding**
 - Inviato dal client o dal server
 - **Content-Encoding: x-gzip**

HTTP/1.0 Headers / 2

- **Content-Length**
 - Inviato dal client o dal server
 - **Content-Length: 56**
 - Indica quante informazioni vengono trasmesse
 - Indispensabili per connessioni HTTP persistenti, o quando si usa POST
- **Content-Type**
 - Inviato dal server
 - **Content-Type: text/html**
 - what MIME type the payload is

HTTP/1.0 Headers / 3

- **Date**
 - **Date: Tue, 15 Nov 1994 08:12:31 GMT**
- **Expires**
 - Inviato dal server
 - **Expires: Thu, 01 Dec 1994 16:00:00 GMT**
 - Indica quando “scade” l'informazioni trasmessa (utile per le cache)
- **If-Modified-Since**
 - Inviato dal client
 - **If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT**
 - Il server restituisce i dati se sono stati modificati dalla data indicata, altrimenti “304 Not Modified”

HTTP/1.0 Headers / 4

- **Last-Modified**
 - Restituito dal server
 - **Last-Modified: Sat, 29 Oct 1994 19:43:31 GMT**
 - Data dell'ultima modifica della risorsa richiesta
- **Location**
 - Restituito dal server
 - **Location: http://www.dsi.unive.it**
 - Usato per le redirezioni
- **Pragma**
 - Inviato dal client o dal server
 - **Pragma: no-cache**

HTTP/1.0 Headers / 5

- **Referer**
 - Inviato dal client (l'errore di spelling ha origini storiche, si direbbe *Referrer*)
 - **Referer:** `http://www.xxx-smut.com`
 - Specifica l'indirizzo di provenienza. **Occhio alla privacy!**
- **Server**
 - Restituito dal server
 - **Server:** `Apache-2.4`
- **User-Agent**
 - Inviato dal client
 - **User-Agent:** `Mozilla/4.07 [en] (X11; I; Linux 2.0.36 i686)`
 - Perché? Ottimizzare il layout in base alle capacità del client

HTTP/1.0 Headers / 6

- **WWW-Authenticate**
 - Inviato dal server
 - **WWW-Authenticate:** `<challenge>`
 - Indica al client di reinviare la richiesta con l'header "Authorization:"

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 (introdotta in HTTP 1.0)
 - Digest access authentication (introdotta in HTTP 1.1)

Autenticazione / 2 Basic Authentication

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

Autenticazione / 3 Digest Authentication

- Introdotto da HTTP 1.1, descritto in RFC 2069 e RFC 2617.
- 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.).

HTTP/1.1

- Diversi problemi associati con HTTP/1.0
 - Consistenza delle cache gestite in modo scadente
 - Difficile implementare multi-homed servers
 - 1 indirizzo IP che corrisponde a diversi siti web
 - Difficile precalcolare il content-length
 - Contenuto dinamico...
 - Se la connessione cade, i dati sono perduti
- HTTP/1.1 risolve in parte questi problemi
 - ...a costo dell'introduzione di un notevole livello di complessità
 - HTTP/1.1 NON è un protocollo semplice da implementare

Che c'è di nuovo in HTTP/1.1

- **Host: www.dsi.unive.it**
 - Header obbligatorio per il client.
 - Risolve il problema dei siti multi-homed
- **Range: bytes=300-304,601-993**
 - Utile se le connessioni si interrompono
- **Age: <seconds, date>**
 - Somme dei tempi di vita nelle cache + tempo di trasporto
- **Etag: fa898a3e3**
 - Tag univoco per identificare il documento
- **Cache-control: <command>**
 - Utile per evitare che la risorsa venga messa nelle cache
- **"chunked" transfer encoding**
 - Documenti "segmentati". Utile nel caso di pagine generate dinamicamente.

Codici di risposta

- **2xx**
 - Successful
- **3xx**
 - Redirected
 - 301 Moved Permanently
 - 307 Temporary Redirect
- **4xx**
 - Client error
 - Bad request, payment required, forbidden, unauthorized, not found...
- **5xx**
 - Server error

Esempi di status code

- 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 CGI mal fatto)
- 501 Not implemented** (metodo non conosciuto dal server)

Cookies

- Informazioni inviate dai server web
 - Tipicamente 4KB
- Vengono mantenute sull'hard disk dell'utente, e possono essere letti solo dal sito che li ha inviati
- Utilizzati per
 - Gestione delle sessioni;
 - Tracking nei siti
 - Ordini online
 - Pubblicità "su misura"
- Possono essere disabilitati
- <http://www.cookiecentral.com/>

Come funzionano in pratica?

- Il server invia gli header seguenti:

```
...
Content-type: text/html
Set-Cookie: foo=bar; path=/; expires Mon, 09-Dec-2002 13:46:00 GMT
```

- Setta un cookie con le seguenti caratteristiche:
 - Il cookie si chiama "foo"
 - Il suo contenuto e' "bar"
 - Il cookie e' valido per l'intero sito (path=/) (alle pagine al di fuori del path *non* viene inviato il cookie)
 - Scade il 9 dicembre 2002
 - Opzionalmente, si puo' settare il dominio di validità del cookie (a quali siti web viene spedito il cookie)

Come funzionano in pratica?

- Il browser, quando accede ad un sito (e ad un percorso) per cui è settato un cookie, invia il seguente header:

```
...
Cookie: foo=bar
...
```

- Questo informa il server della presenza del cookie chiamato "foo" di valore "bar"

Cookies

1. Il browser richiede una pagina web
2. Il server invia la pagina e il comando di memorizzare un cookie nel client
3. Il browser memorizza il cookie
4. Ogni volta che viene richiesta una pagina, il browser controlla se esiste un cookie per quel server e per quella pagina
5. In caso affermativo, il browser trasmette il valore del cookie assieme alla richiesta per la pagina
6. Il server riceve la richiesta con associato il cookie, e può utilizzare tale informazione per "ricordarsi" qualcosa circa l'utente

Esempio: usare i cookies per la pubblicità "su misura"

- Visito l'ipotetico sito *www.animalidomestici.it* e seleziono la pagina "Criceti"
- Ricevo la pagina richiesta, che contiene anche un ipotetico banner:

```

```

- Il sito *www.banner.it* invia l'immagine e un cookie, ad esempio:

```
UserPref=Criceti
```

Esempio / 2

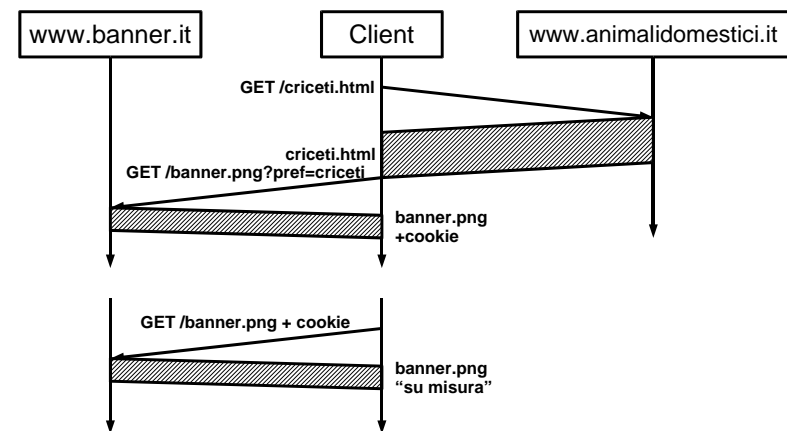
- Da questo momento, ogni volta che dal mio browser ricevo la risorsa generica

```

```

il sito *www.banner.it* mi invierà immagini che hanno a che fare con i criceti

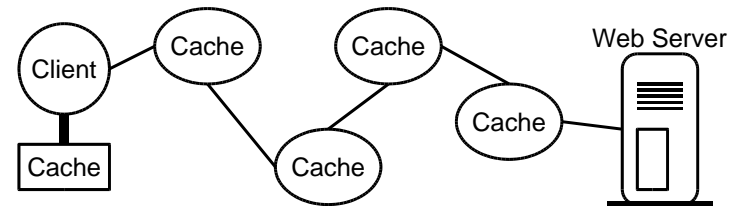
Esempio / 3



Gestione delle cache

- Problema iniziale del web: nessuna gestione della località
- Risultati: server web che forniscono milioni di volte la stessa pagina a comunità localizzate di utenti
 - Es: gli studenti di una università che consultano le pagine web dei docenti, che magari sono ospitate su un server esterno all'università

Soluzione: gerarchie di cache



- Si veda www.squid-cache.org

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 indica una scadenza della risorsa con l'header **Expires** o con la direttiva **max-age** in **Cache-Control**
 - **Expires: Thu, 01 Dec 1994 16:00:00 GMT**
la risorsa scade nella data specificata
 - **Cache-Control: max-age=1000**
Indica che il richiedente è disposto ad accettare una risposta la cui età è non maggiore di quanto specificato (in secondi)
- 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.

Validazione della cache

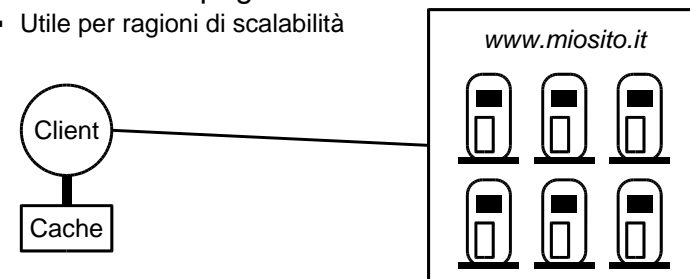
- Anche dopo la scadenza, nella maggior parte dei casi, una risorsa sarà ancora non modificata, e quindi la risorsa in cache valida.
- Un modo semplice per fare validazione è usare HEAD: il client fa la richiesta, e verifica la data di ultima modifica. Ma questo richiede una richiesta in più sempre.
- 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

Sicurezza

- HTTPS (RFC 2818)
 - Introdotto da Netscape, 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://)
- S-HTTP (RFC 2660)
 - Poco diffuso, incapsula richieste e risposte HTTP in un messaggio crittografato secondo o un formato MIME apposito (MIME Object Security Services, MOSS), o un formato terzo (Cryptographic Message Syntax, CMS).
 - E' più efficiente ma più complesso.

Supporto per server multipli

- Abbiamo visto come gestire diversi server web associati ad un singolo indirizzo IP
 - Utilizzando l'header "Host:" di HTTP/1.1
- Che succede se abbiamo diversi server web che forniscono pagine dello stesso sito web?
 - Utile per ragioni di scalabilità

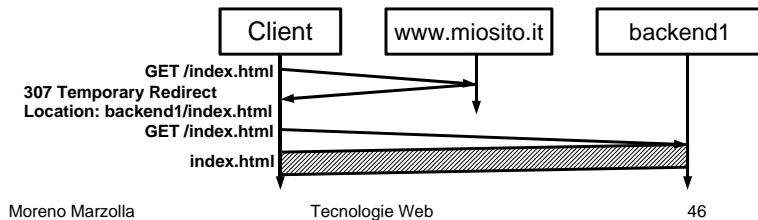


Soluzioni / 1

- DNS round-robin
 - Si assegnano più indirizzi IP ad un singolo nome di dominio
 - Lo stesso nome viene risolto nei diversi indirizzi IP a rotazione
 - Limitazioni
 - I singoli nodi sono esposti direttamente ai client
 - Non tiene conto di fattori come il carico o la potenza di calcolo dei singoli nodi

Soluzioni / 2

- Front-end redirection
 - Un singolo front-end utilizza gli header per la redirection ("temporary redirect") per dirottare le richieste verso i nodi di backend
 - Limitazioni
 - Il front-end è un punto singolo di fallimento
 - Introdotti dei ritardi per la redirection (effetto ping-pong)



Soluzioni / 3

- IP-level multiplexing through smart router
 - Opera a livello IP per indirizzare i pacchetti a host diversi
 - Cisco, SUN, etc. producono hardware specifico per fare ciò
 - Cisco LocalDirector
 - Problematica la gestione dei fallimenti e dello stato
- Soluzioni basate sui client
 - Es, codice Javascript seleziona il proxy più vicino
 - Non basato su HTTP, non tutti i client supportano tali meccanismi

Altre cose da considerare

- Il web non è solo per le persone
 - spiders, crawlers, worms, etc, possono dare problemi
 - Burst di richieste, cicli infiniti nelle visite
- Netscape, IE, ed Apache sono gli standard de facto
 - La loro semantica può differire dagli standard
 - Occorre adeguarsi

Aspetti positivi del WWW

- La gente lo usa per pubblicare informazioni
 - Esistono gli standard
 - Facile "pubblicare" ipertesti
 - Economico
 - Potenzia le facoltà di pubblicazione individuale
- Accessibile
 - Facile da usare/economico
 - Grafica+testo
- Favorisce la condivisione di informazioni
 - Biologia, fisica...

Aspetti positivi del WWW

- Si possono effettuare ricerche
 - Per parola chiave
 - Grande bacino d'utenza, la struttura dei link o i pattern di accesso possono guidare le metriche di rilevanza
 - E' possibile (ma non facile) effettuare query "imprecise"
- Meccanismo uniforme per individuare le risorse
 - URLs
 - Decentralizzato
- Formato standard HTML/CSS
- Il software per l'accesso al WWW è fornito di default

Aspetti positivi del WWW

- Services / Commerce / Information
 - Shopping, viaggi, compravendita di azioni...
 - Aggiornato 24x7
 - Quando e dove vi pare
 - Si possono acquistare prodotti a prezzi convenienti
- Costruisce comunità distribuite
- Le informazioni possono essere presentate in maniera personalizzata

Aspetti negativi del WWW

- Prestazioni non prevedibili
 - "World Wide Wait"
- Nessun controllo della qualità
 - Tantissime informazioni, alta varianza in qualità e utilità delle informazioni pubblicate
- Mancanza di classificazioni
 - Informazioni pubblicate più velocemente di quanto siano classificate
- La sicurezza è stata aggiunta a posteriori
 - Non è mai una cosa positiva
 - Informazioni personali, anche sensibili, trasmesse elettronicamente

Aspetti negativi del WWW

- (Pressoché) totale assenza di privacy
- La gente abusa dell'HTML
 - Gli standard non vengono rispettati, ma le pagine *devono* potersi visualizzare lo stesso
- "eyeball encoding" delle informazioni
 - Le informazioni vengono codificate per l'uso da parte delle persone, non delle macchine
- Dati malamente/non aggiornati
 - Link interrotti

Aspetti negativi del WWW

- Difficile trovare ciò che si vuole
 - Diversi modi per avere risultati simili
 - Difficile specificare le query "giuste"
 - Difficile sapere quanto l'informazione è aggiornata
 - Difficile valutare la completezza dei risultati delle ricerche
 - "Trappole" per aumentare la popolarità
 - Difficile/impossibile ricercare risorse non testuali
 - Alcune ricerche di immagini, es Google

Aspetti negativi del WWW

- Transazioni lunghe sono la norma
 - Nessuna garanzia di consistenza delle transazioni
 - Nessuna conferma end-to-end
- Fattori sociopolitici
- Replication / Caching / Multicast sono problemi ancora presenti e non completamente risolti
- Traffico altamente variabile