

Corso di Laurea in Ingegneria Informatica



Corso di Reti di Calcolatori I

Roberto Canonico (roberto.canonico@unina.it)

Giorgio Ventre (giorgio.ventre@unina.it)

HTTP

**I lucidi presentati al corso sono uno strumento didattico
che NON sostituisce i testi indicati nel programma del corso**

Nota di copyright per le slide COMICS



Nota di Copyright

Questo insieme di trasparenze è stato ideato e realizzato dai ricercatori del Gruppo di Ricerca COMICS del Dipartimento di Informatica e Sistemistica dell'Università di Napoli Federico II. Esse possono essere impiegate liberamente per fini didattici esclusivamente senza fini di lucro, a meno di un esplicito consenso scritto degli Autori. Nell'uso dovranno essere esplicitamente riportati la fonte e gli Autori. Gli Autori non sono responsabili per eventuali imprecisioni contenute in tali trasparenze né per eventuali problemi, danni o malfunzionamenti derivanti dal loro uso o applicazione.

Autori:

Simon Pietro Romano, Antonio Pescapè, Stefano Avallone,
Marcello Esposito, Roberto Canonico, Giorgio Ventre

Il protocollo HTTP/1.0



- Si basa su TCP
- Il client apre un socket verso il porto TCP 80 del server (se non diversamente specificato)
- Il server accetta la connessione
- Il client manda una richiesta per uno specifico oggetto identificato mediante una URL
- Il server risponde e chiude la connessione
- **Il protocollo HTTP è stateless: né il server né il client mantengono a livello HTTP informazioni relative ai messaggi precedentemente scambiati**

3

URL : Uniform Resource Locator



- Un URL HTTP ha la seguente sintassi (RFC 2396) :
`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

Es.: `http://www.unina.it/immatricolazioni.htm#ingegneria`

HTTP: versioni



- Il protocollo è definito in documenti RFC
- Negli anni si sono avute tre versioni:
 - http 0.9
 - http 1.0 (1996)
 - RFC 1945
 - http 1.1 (1997 e 1999)
 - RFC 2068
 - RFC 2616

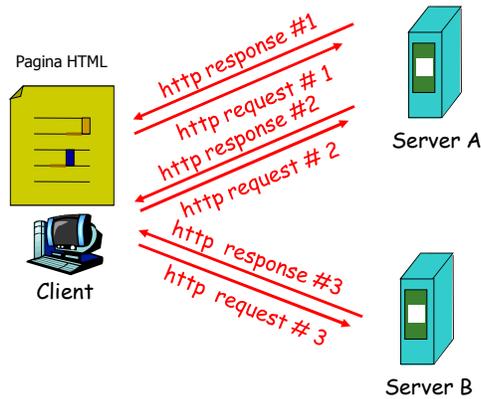
NOTA: un RFC (Request For Comment) è un documento pubblico sottoposto alla comunità Internet al fine di essere valutato. Ciò che è un RFC rappresenta uno standard *de facto* nella comunità Internet. Tutti gli RFC possono essere reperiti al sito dell'Internet Engineering Task Force (<http://www.ietf.org>)

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)



Es: richiesta di una pagina contenente immagini

1: il client apre una connessione TCP sul porto 80 verso l'indirizzo `www.unina.it`

2: il server è in ascolto sul porto 80 ed accetta la connessione

3: il client invia un messaggio di richiesta HTTP della home page

4: il server analizza la richiesta HTTP, prepara la risposta HTTP e la invia al client

5: il server chiude la connessione TCP

6: il client effettua il parsing del documento (es. HTML) contenuto nel messaggio di risposta HTTP, ne fa il rendering sullo schermo e rileva che all'interno della pagina sono presenti 3 collegamenti ad immagini.

7: per ciascuna delle immagini vengono ripetuti i passi da 1 a 5.

8

Connessioni persistenti e non persistenti



non persistente

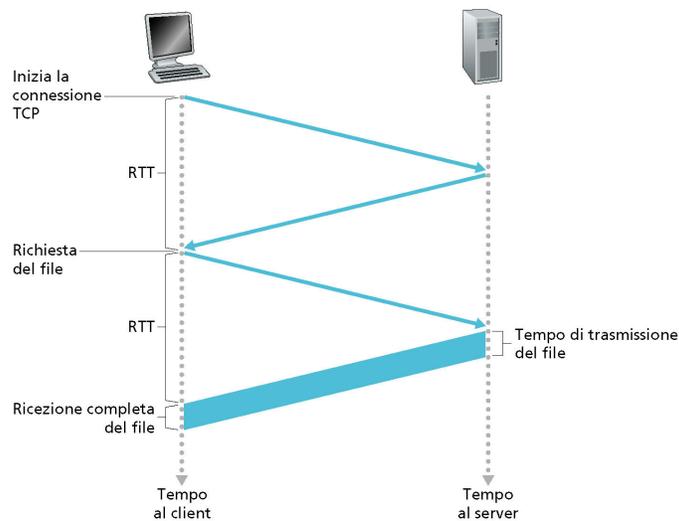
- HTTP/1.0 (RFC 1945)
- Il server analizza una richiesta, la serve e chiude la connessione
- 2 Round Trip Time (RTT) per ciascuna richiesta
- Ogni richiesta subisce lo slow-start TCP

persistente

- HTTP/1.1 (RFC 2068, RFC 2116)
- Sulla stessa connessione il server analizza tutte le richieste e le serve
- Il client riceve la pagina iniziale e invia subito tutte le altre richieste
- Si hanno meno RTT ed un solo slow-start
- Esiste anche una versione con parallelismo (with pipelining)

9

Round Trip Time e connessioni HTTP



10

Il protocollo HTTP



- HTTP è un protocollo testuale
- I messaggi sono costituiti da sequenze di byte
- Ogni byte identifica un carattere secondo la tabella ASCII
- Il payload dei messaggi può essere comunque anche in formato binario (es. un'immagine GIF, un video, ecc.)

* Come si vedrà più avanti HTTP è utilizzato anche per altri scopi.

11

Il messaggio HTTP/1.0 request: un esempio



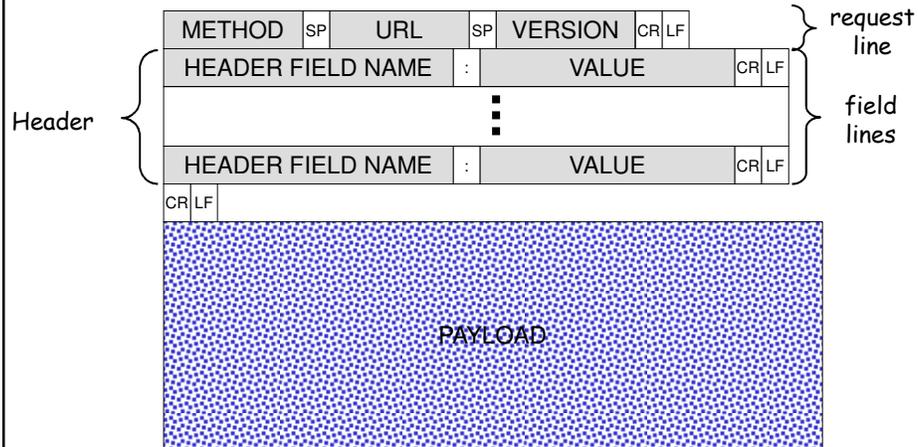
Un esempio di messaggio GET

```
GET /path/pagename.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: it
RIGA VUOTA
```

indica la fine del messaggio

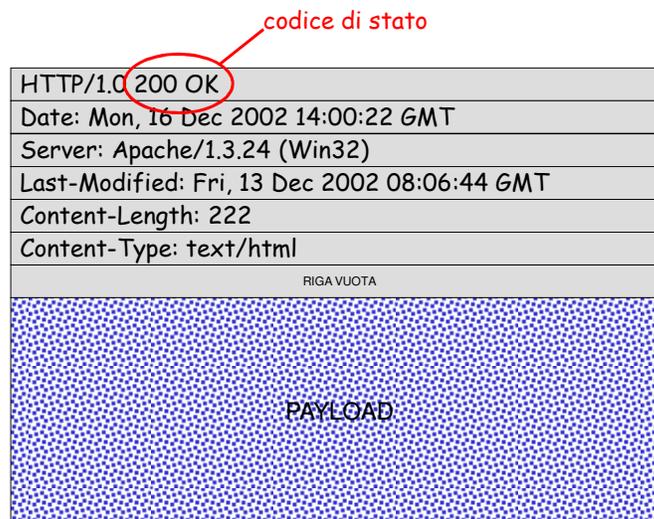
12

Il messaggio HTTP/1.0 request



13

Il messaggio HTTP/1.0 response



14

Esempi di codici di stato



200 OK

- Successo: l'oggetto richiesto si trova più avanti nel messaggio

301 Moved Permanently

- L'oggetto richiesto è stato spostato.
Il nuovo indirizzo è specificato più avanti nel messaggio
(Location:)

400 Bad Request

- Richiesta incomprensibile al server

404 Not Found

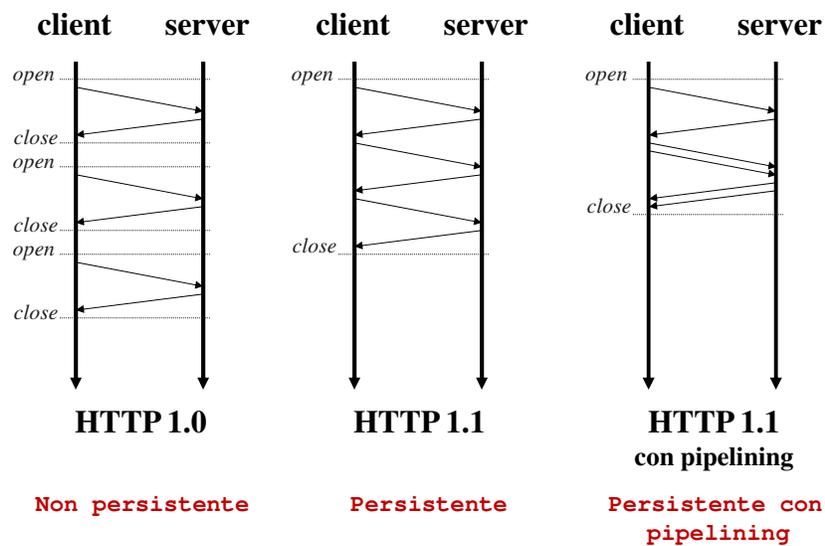
- Il documento non è presente sul server

505 HTTP Version Not Supported

- La versione del protocollo HTTP usata non è supportata dal server

15

La connessione HTTP



16

Lo scambio di messaggi



- Il metodo **GET**
- Il metodo **HEAD**
- Il metodo **POST**
- Il metodo **PUT**

- *La response*

17

Il metodo GET



- Uno dei più importanti metodi 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

18

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 sottomessi alla risorsa specificata; non è stata data risposta
- Non è l'unico modo per inviare dati

21

Il metodo POST: un esempio



The screenshot shows a network capture in Wireshark. The top pane shows a list of packets. Packet 6 is highlighted in red, indicating it is selected. It is an HTTP POST request to /ema11.php. The response, packet 7, is highlighted in green and has a status of 200 OK. The packet details pane shows the structure of the HTTP message, including the request line, headers, and body. The body of the request is visible in the hex and ASCII panes at the bottom.

```
Frame 6 (881 bytes on wire, 881 bytes captured)
Ethernet II, Src: wlstron_2e:cb:5b (00:0a:e4:2e:cb:5b), Dst: D-Link_1e:6a:d1 (00:17:9a:1e:6a:d1)
Internet Protocol, Src: 192.168.1.2 (192.168.1.2), Dst: 212.52.84.18 (212.52.84.18)
Transmission Control Protocol, Src Port: 2839 (2839), Dst Port: http (80), Seq: 1, Ack: 1, Len: 827
Hypertext Transfer Protocol
  POST /ema11.php HTTP/1.1
  Host: wpp08.11bero.it
  User-Agent: Mozilla/5.0 (Windows; u; windows NT 5.1; it-IT; rv:1.7.12) Gecko/20050919 Firefox/1.0.7
  Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
  Accept-Language: it,it;q=0.8,en-us;q=0.5,en;q=0.3
  Accept-Encoding: gzip,deflate
  Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
  Keep-Alive: 300
  Connection: keep-alive
  Referer: http://www.11bero.it
  cookie: ...utmz=267072147.1190633699.2.3.utmcsr=HP%2BLiberol|utmccn=Infostrada%2GADSL|utmcnd=Lancio1; Liberomail=2dccc1ab3e03

11e0 61 6c 69 76 65 0d 0a 32 65 66 65 72 65 72 3a 20  a11ve...r eferer:
01f0 68 74 74 70 3a 2f 2f 77 77 72 6c 69 62 65 72  http://w ww.11ber
0200 69 2e 69 74 2e 0d 66 43 6f 6f 6b 69 65 3a 20 3f  ookie:
0210 3f 74 74 6d 74 3d 32 36 37 30 37 32 31 34 37 2e  utmz=2 7072147,
0220 31 31 39 30 36 33 33 36 39 39 2e 32 2e 33 2e 75  11906336 99.2.3.u
0230 74 6d 63 73 72 3d 48 50 25 32 42 4c 69 62 65 72  tmcscr=HP %2BLiber
0240 6f 7c 75 74 6d 63 63 6e 3d 49 6e 66 6f 73 74 72  o|utmccn =Infostr
0250 61 64 61 25 32 30 41 44 53 4c 7c 75 74 6d 63 6d  ada%2OAD SL|utmcc
0260 64 3d 4c 61 6e 63 69 6f 31 3b 20 4c 69 62 65 72  d=Lancio 1; Liber
0270 ff 6d 61 69 6c 3d 32 64 63 63 6d 31 61 62 33 65  rms11=1 ccd1ab3e
```

22

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

23

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 } request line

[Header] } Header
[Header] }

CRLF

Body

dove:

[Header] indica un elemento opzionale

CRLF indica la sequenza di caratteri di codice ASCII

$0D_{16} = 13_{10} \rightarrow$ CR = Carriage Return

$0A_{16} = 10_{10} \rightarrow$ LF = Line Feed

24

HTTP: Response (2)



- Un 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>

25

HTTP Response: un esempio



(Untitled) - Ethereal

File Edit View Go Capture Analyze Statistics Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	85.182.14.100	143.225.92.106	TCP	4890 > 27047 [SYN] Seq=0 Len=0 MSS=1452
2	3.270549	143.225.229.163	209.85.129.147	TCP	4150 > http [SYN] Seq=0 Len=0 MSS=1460
3	3.304386	209.85.129.147	143.225.229.163	TCP	http > 4150 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
4	3.304443	143.225.229.163	209.85.129.147	TCP	4150 > http [ACK] Seq=1 Ack=1 Win=65535 [TCP CHECKSUM INCORR]
5	3.308079	143.225.229.163	209.85.129.147	HTTP	GET / HTTP/1.1
6	3.402851	209.85.129.147	143.225.229.163	TCP	http > 4150 [ACK] Seq=1 Ack=574 Win=6611 Len=0
7	3.413274	209.85.129.147	143.225.229.163	TCP	[TCP segment reassembled PDU]
8	3.415291	209.85.129.147	143.225.229.163	HTTP	HTTP/1.1 200 OK (text/html)
9	3.433314	143.225.229.163	209.85.129.147	TCP	4150 > http [ACK] Seq=574 Ack=2344 Win=65535 [TCP CHECKSUM II]
10	6.038651	85.182.14.100	143.225.92.106	TCP	4890 > 27047 [SYN] Seq=0 Len=0 MSS=1452

Frame 8 (1167 bytes on wire (3167 bytes captured))

- Ethernet II, Src: Cisco_Gd:04:00 (00:14:f1:6d:04:00), Dst: Wlstron_2e:cb:5b (00:0a:e4:2e:cb:5b)
- Internet Protocol, Src: 209.85.129.147 (209.85.129.147), Dst: 143.225.229.163 (143.225.229.163)
- Transmission Control Protocol, Src Port: http (80), Dst Port: http (80), Seq: 1431, Ack: 574, Len: 1113
- [Reassembled TCP Segments (2543 bytes): #7(1430), #8(1113)]
- Hypertext Transfer Protocol
 - HTTP/1.1 200 OK(V)
 - cache-control: private(V)
 - content-type: text/html; charset=UTF-8(V)
 - content-encoding: gzip(V)
 - server: gws(V)
 - content-length: 2364(V)
 - date: Mon, 24 Sep 2007 10:25:21 GMT(V)
 - Content-encoded entity body (gzip): 2364 bytes -> 5472 bytes
 - Line-based text data: text/html

0000 00 0a e4 2e cb 5b 00 14 f1 6d 04 00 08 00 45 00[. .m...E.
0010 04 31 b6 c6 00 00 33 06 04 43 d1 55 81 95 8f e13...C.U...
0020 e5 a3 00 50 10 36 a6 a2 e3 00 f7 46 f9 c0 50 18P.6...F.P.
0030 19 d3 82 fa 00 00 d4 67 d5 49 d0 32 35 8c 5e 2fg...I.25.A/
0040 e4 85 45 be 74 74 be 84 3f a4 93 b3 03 79 5c ffE.t...?..Y\
0050 21 58 ee d9 69 50 8a 9e a6 39 99 0c c9 89 87 88x..P...S.....
0060 69 05 41 a1 28 13 28 10 20 56 74 ee d3 bf 96 c3 1..A..C..C..vt....
0070 ea 0c d5 36 7a ea a5 85 7a 7c d8 40 ef 85 0e a5 ..A...e>A

Frame (1167 bytes) [Reassembled TCP (2543 bytes)] [Uncompressed entity body (5472 bytes)]

File: "C:\Documents\1146180505\1170001\ethereal\20070924\3676.pcap" [9-10-D-10-M-0-Proc-0]

26

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

27

Status code: alcuni 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

28

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)

29

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

* MIME, Multipurpose Internet Mail Extensions - RFC 2045-2056

30

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:** oggetto/formato
 - Ogni coppia oggetto/formato costituisce un 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

31

Una prova con telnet



```
> telnet www.unina.it 80
```

```
GET / HTTP/1.0
```

Request-line

```
HTTP/1.1 200 OK
```

```
Date: Mon, 25 Oct 2010 23:02:38 GMT
```

```
Server: Apache/2.2.14 (Unix) mod_jk/1.2.28 DAV/2
```

```
ETag: W/"2097-1213082454000"
```

```
Last-Modified: Tue, 10 Jun 2008 07:20:54 GMT
```

```
Content-Length: 2097
```

```
Connection: close
```

```
Content-Type: text/html
```

Response

32

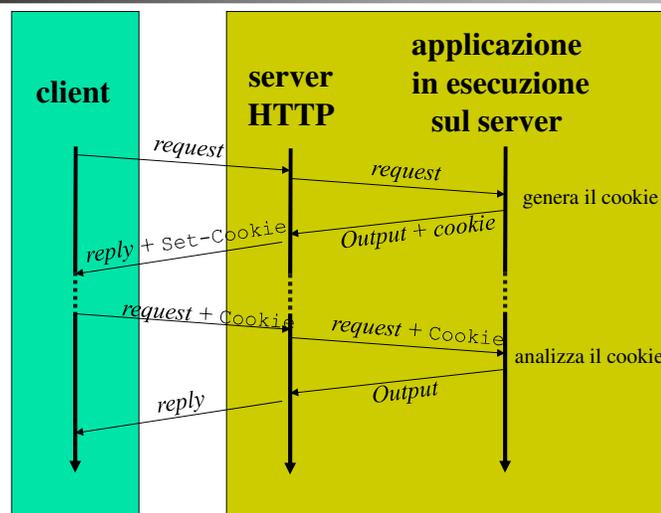
I cookies



- **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
- 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 viene riproposta la propria username all'atto dell'accesso ad un sito per la posta
- I cookies sono definiti in RFC2109(su proposta di Netscape)

33

Cookies (2)



34

Cookies: header specifici



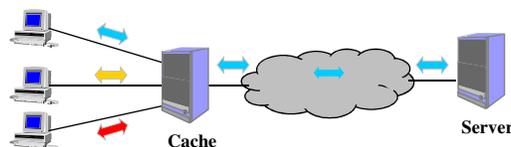
- Il meccanismo dei cookies dunque definisce due nuovi possibili header: uno per la risposta, ed uno per le richieste successive
 - **Set-Cookie**: header della risposta
 - il client può memorizzarlo (se vuole) e rispedito alla prossima richiesta
 - **Cookie**: header della richiesta
 - il client decide se spedito 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

35

Web caching

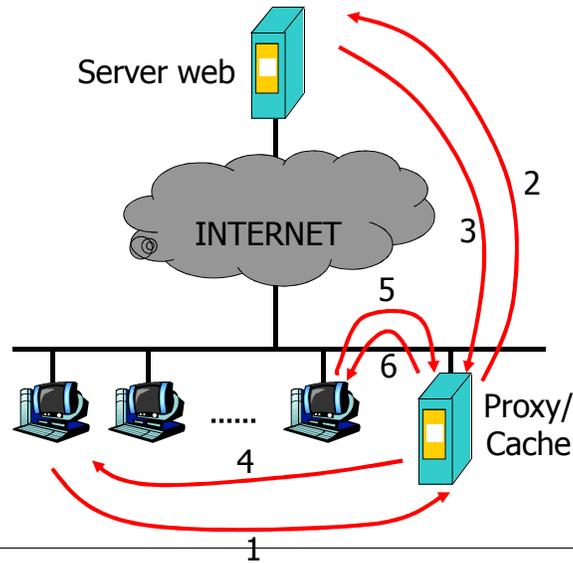


- Si parla genericamente di Web caching quando le richieste di un determinato client non raggiungono il Web Server, ma vengono intercettate da una **cache**
- Tipicamente, un certo numero di client di una stessa rete condivide una stessa cache web, posta nelle loro prossimità (es. nella stessa LAN)
- 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**



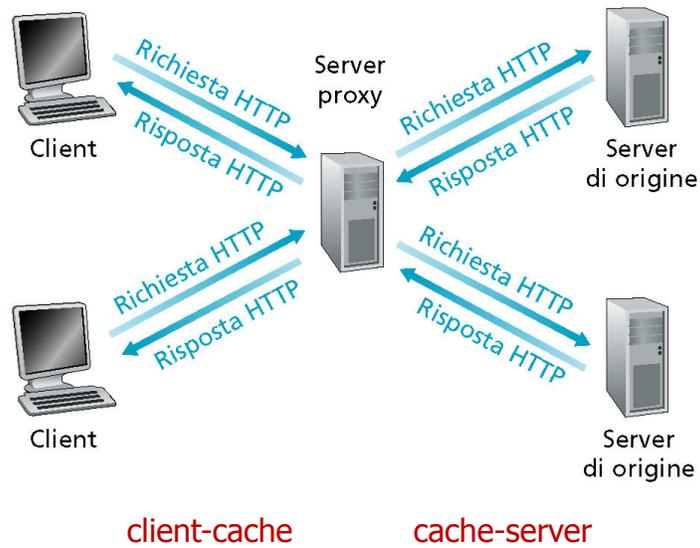
36

Transazioni web con caching



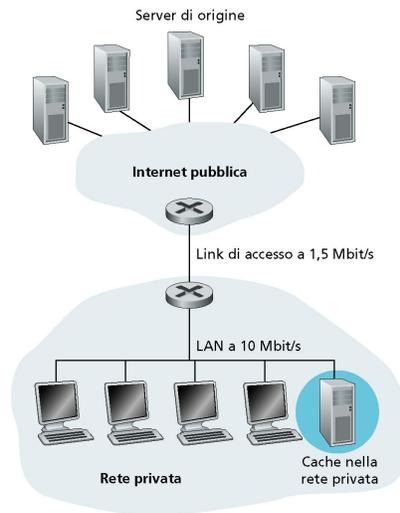
37

Server proxy: schema logico



38

Server proxy in una rete di accesso

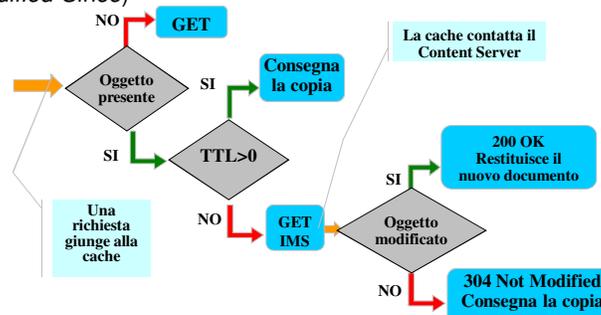


39

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 coerenza:
 - 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*)



40

Non solo browsing...



- HTTP non è utilizzato solo per il Web
- Ad esempio:
 - XML e VoiceXML trasportati su HTTP
 - Web Services e SOA (Service Oriented Architecture)
 - Video streaming
 - Peer-to-peer