Corso di Laurea in Ingegneria Informatica

Corso di Reti di Calcolatori
(a.a. 2010/11)

Roberto Canonico (roberto.canonico@unina.it)
Giorgio Ventre (giorgio.ventre@unina.it)

Protocolli applicativi: HTTP

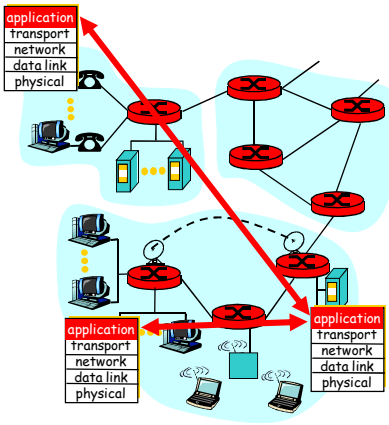
26 ottobre 2010

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

Applicazioni e protocolli dello strato applicativo



3

Strato dell'Applicazione

- Comunicazioni tra processi
- Cosa definisce un protocollo dello strato di applicazione:
  - tipo di messaggi scambiati
  - sintassi dei messaggi
  - semantica dei campi
  - regole di trasmissione
- Applicazioni lato client e lato server (Es.: *servizi Web*, *FTP*)

4

Comunicazione tra processi

- API: Application Programming Interface
  - definisce l'interfaccia tra il livello applicativo e il livello trasporto
  - socket: Internet API
    - due processi comunicano mandando dati in un socket, e leggendo dati dal socket
- Come un processo può "identificare" l'altro processo con cui vuole comunicare?
  - indirizzo IP dell'host che esegue l'altro processo
  - "numero di porta" che permette all'host ricevente di sapere a quale processo locale il messaggio è destinato

5

Livello Trasporto: cenni

- TCP (Transmission Control Protocol):
  - connection oriented
  - trasferimento affidabile
  - congestion control
  - nessuna garanzia su velocità di trasmissione e ritardo
- UDP (User Datagram Protocol):
  - connection less
  - trasferimento non affidabile
  - no congestion control
  - garanzie sul rispetto di una velocità minima (processi real-time)

6

Es.: Requisiti di alcune applicazioni

Application	Data loss	Bandwidth	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kb-1Mb video:10Kb-5Mb	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few Kbps up	yes, 100's msec
Instant messaging	no loss	elastic	yes and no

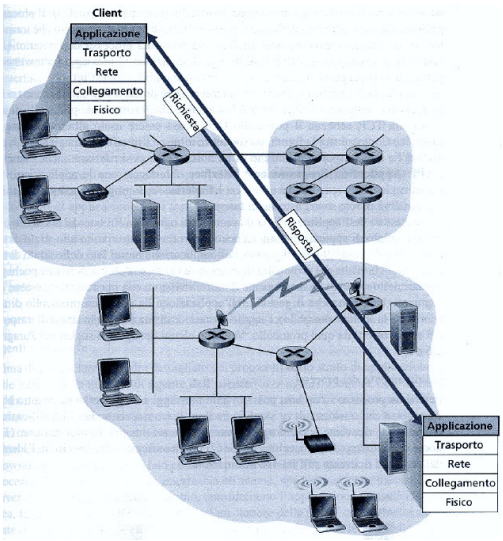
7

Es.: applicazioni e protocolli di trasporto

Application	Application layer protocol	Underlying transport protocol
e-mail	smtp [RFC 821]	TCP
remote terminal access	telnet [RFC 854]	TCP
Web	http [RFC 2068,2616]	TCP
file transfer	ftp [RFC 959]	TCP
streaming multimediale	spesso proprietario (e.g. RealNetworks)	TCP or UDP
remote file server	NFS	TCP or UDP
Internet telephony	proprietary (e.g., Vocaltec)	typically UDP

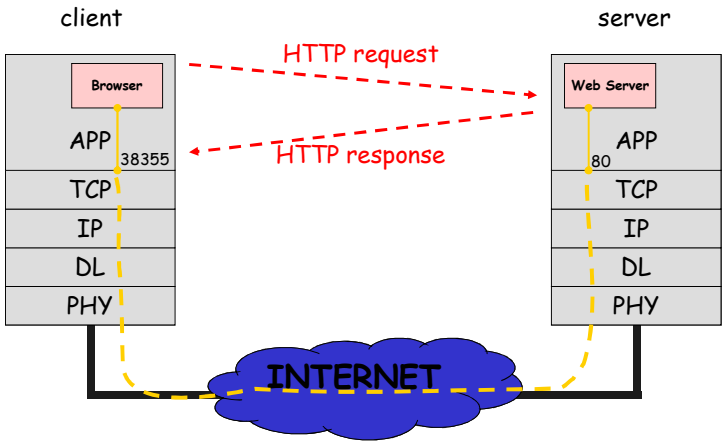
8

Paradigma Client/Server



9

Web: esempio di interazione Client→Server



10

HTTP

11

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**

12

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

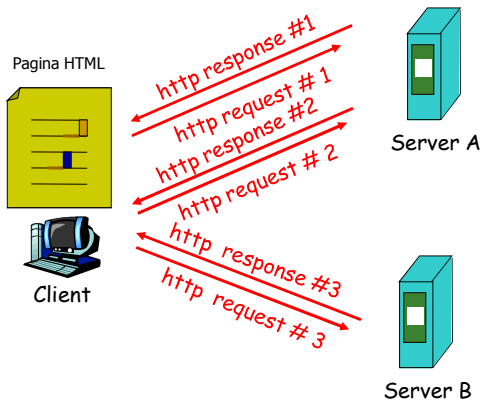
NOTA: un RFC (Request For Comment) è un documento pubblico sottoposto alla comunità Internet al fine di essere valutato. Ciò che è un RFC rappresenta uno standard *de facto* nella comunità Internet. Tutti gli RFC possono essere reperiti al sito dell'Internet Engineering Task Force (http://www.ietf.org)

HTTP per il trasferimento di pagine web

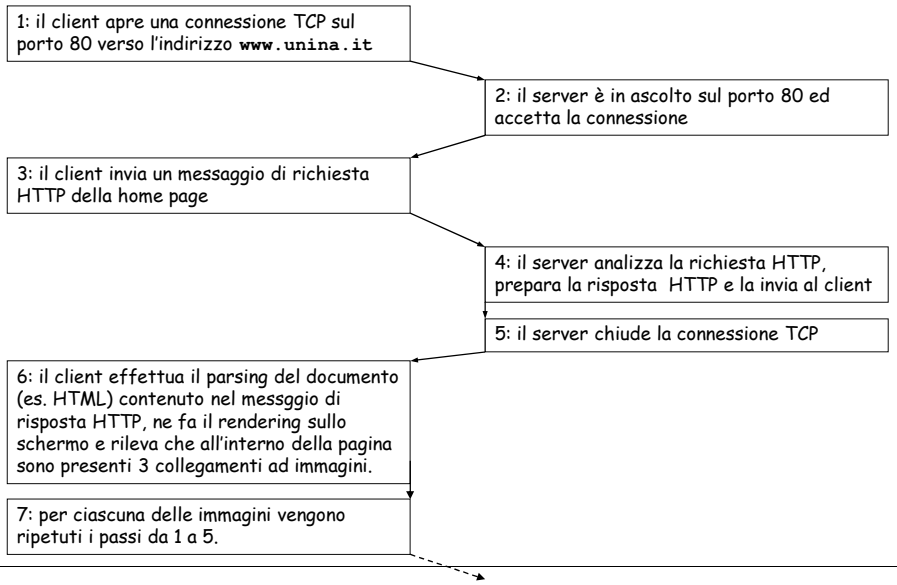
- Tipicamente, una pagina web è descritta da un file testuale in formato HTML (*Hypertext Markup Language*)
- La pagina è identificata mediante un indirizzo, detto URL
- Un file HTML può contenere riferimenti ad altri oggetti che arricchiscono la pagina con elementi grafici
  - Es. sfondo, immagini, 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

15

HTTP per il trasferimento di pagine web (2)



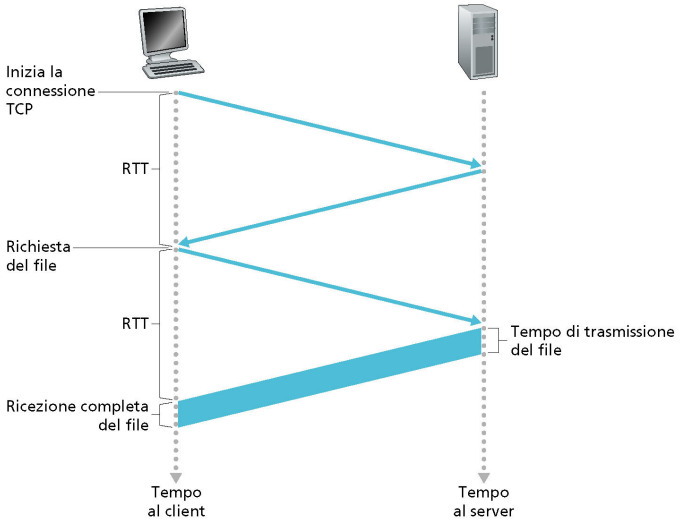
Es: richiesta di una pagina contenente immagini



Connessioni persistenti e non persistenti

non persistente	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	• 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)

Round Trip Time e connessioni HTTP



19

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.

20

Il messaggio HTTP/1.0 request: un esempio

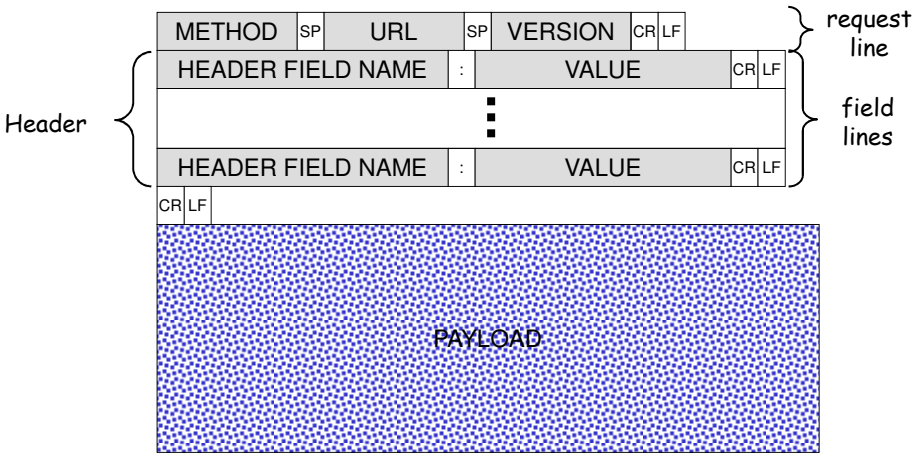
Un esempio di messaggio GET

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

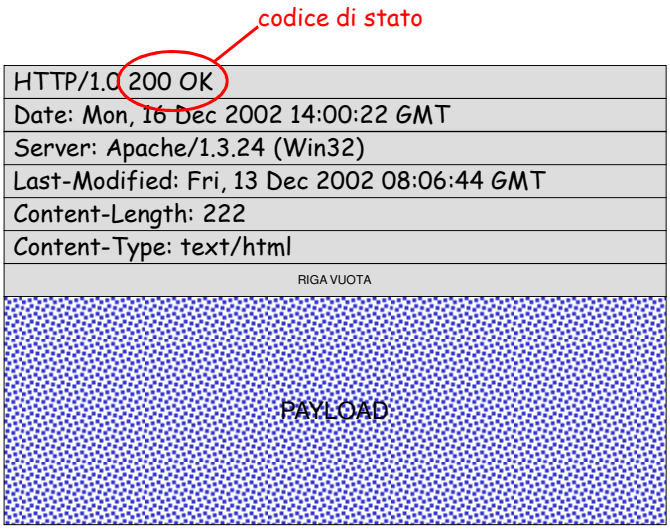
RIGA VUOTA

indica la fine del messaggio

Il messaggio HTTP/1.0 request



Il messaggio HTTP/1.0 response



Esempi di codici di stato

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

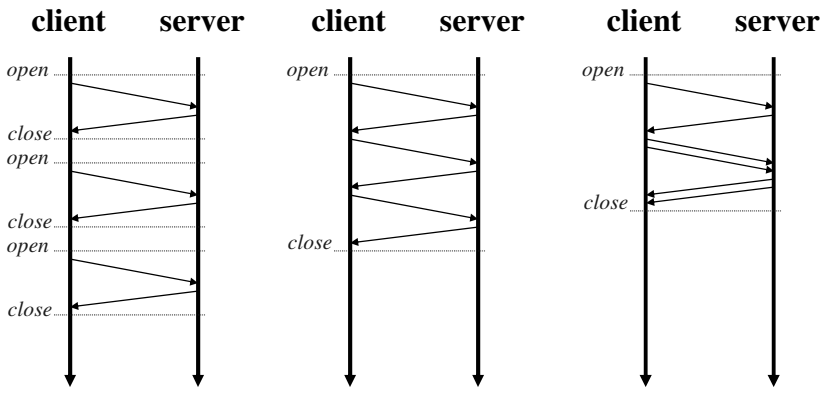
301 Moved Permanently
- L'oggetto richiesto è stato spostato. Il nuovo indirizzo è specificato più avanti nel messaggio (`Location:`)

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

La connessione HTTP



25

Lo scambio di messaggi

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

- La *response*

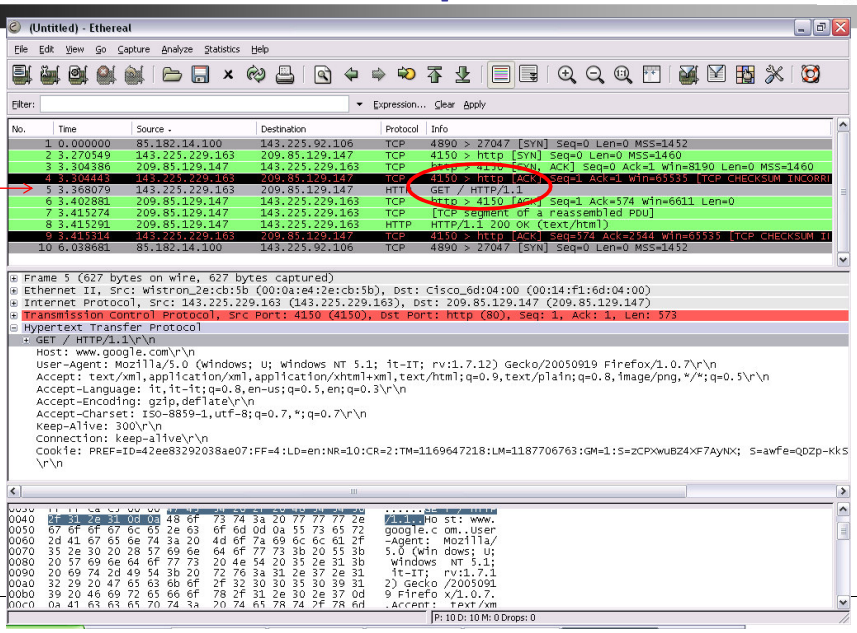
26

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

27

Il metodo GET: un esempio



28

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

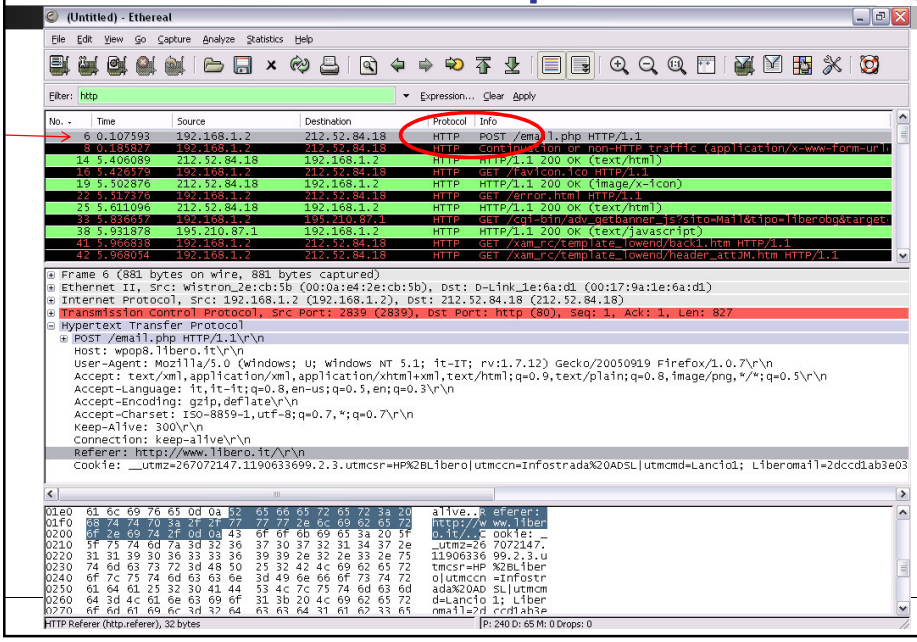
29

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

30

Il metodo POST: un esempio



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 } 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

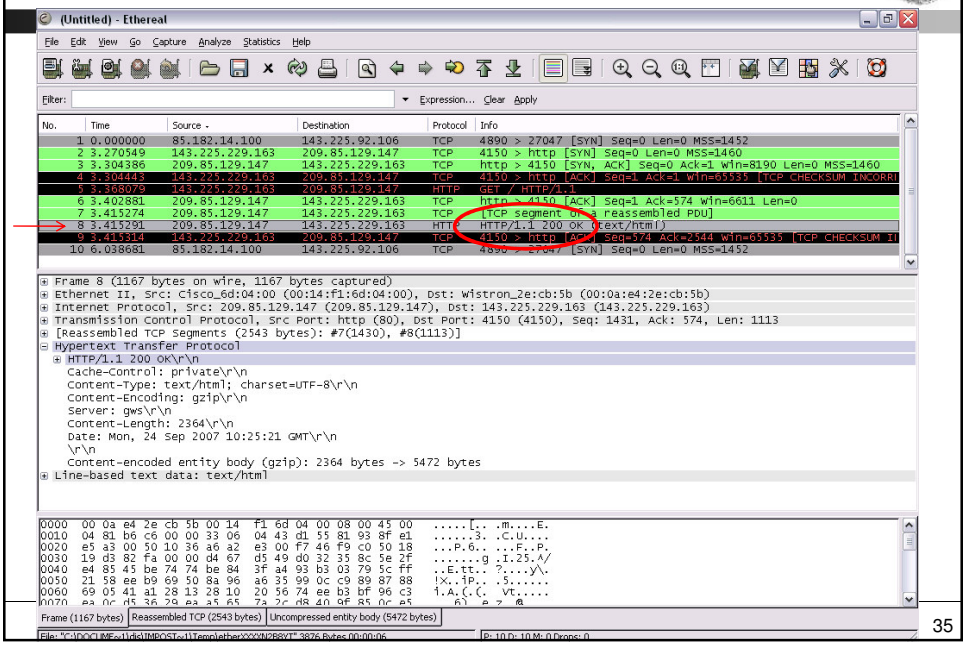
HTTP: Response (2)

- 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>
```

HTTP Response: un esempio



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: 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

37

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)

38

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

39

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

40

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

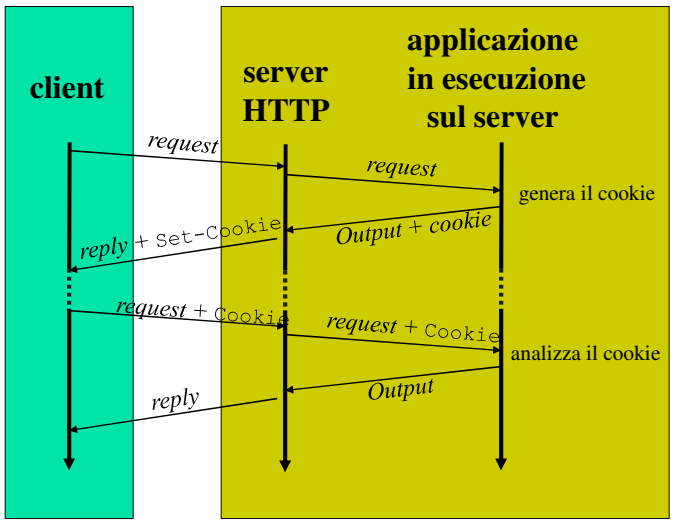
41

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)

42

Cookies (2)



43

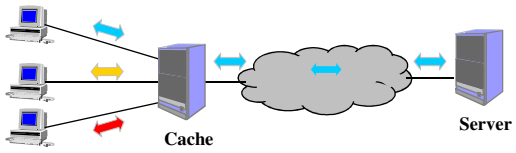
Cookies: header specifici

- 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 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

44

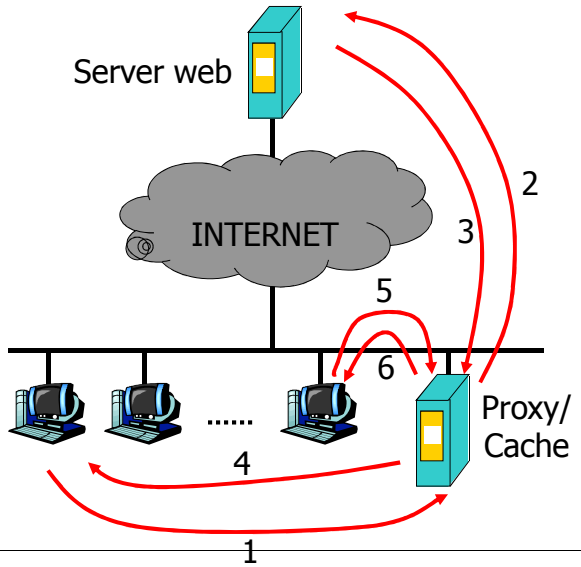
Web caching

- 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



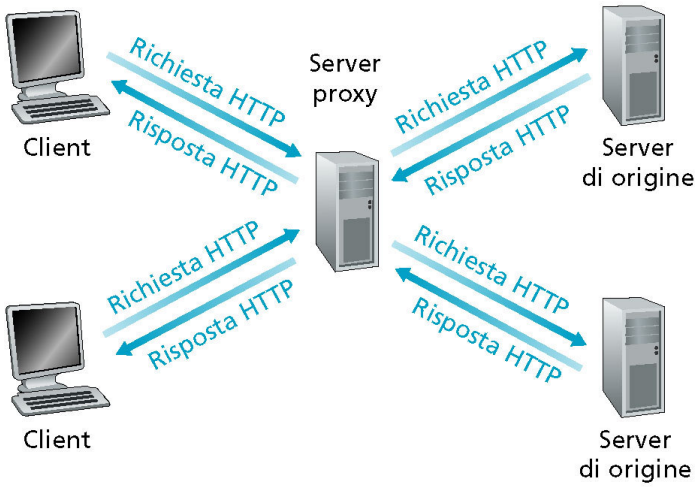
45

Transazioni web con caching



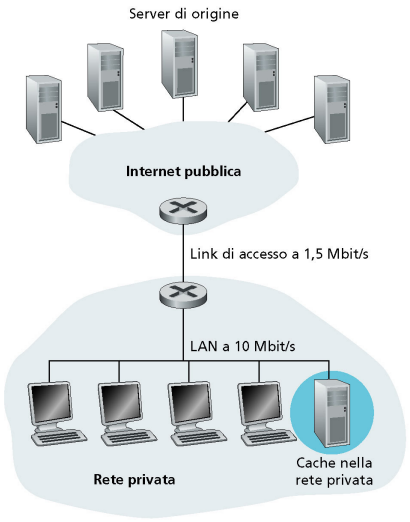
46

Server proxy: schema logico



47

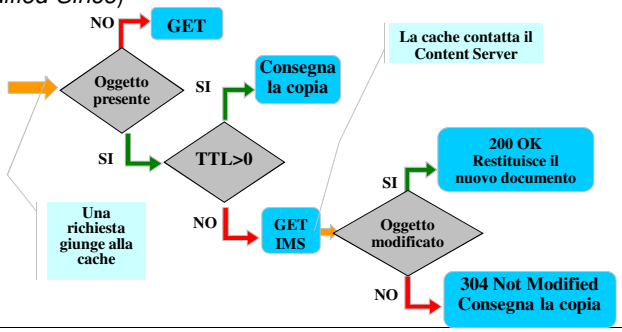
Server proxy in una rete di accesso



48

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*)



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