

**Corso di Laurea in Ingegneria Informatica**



**Corso di Reti di Calcolatori I**

**Roberto Canonico ([roberto.canonico@unina.it](mailto:roberto.canonico@unina.it))**

**Giorgio Ventre ([giorgio.ventre@unina.it](mailto:giorgio.ventre@unina.it))**

Protocolli applicativi: 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**



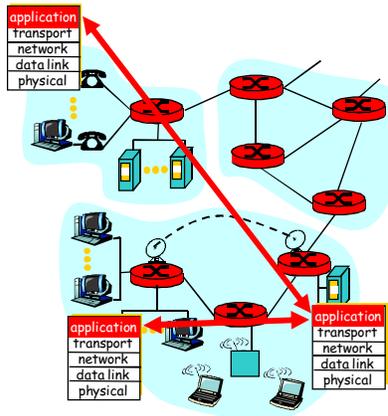
## Nota di Copyright

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

## 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

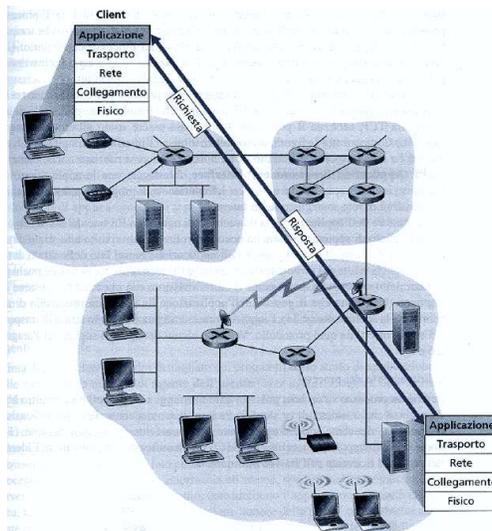
## Es.: applicazioni e protocolli di trasporto



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

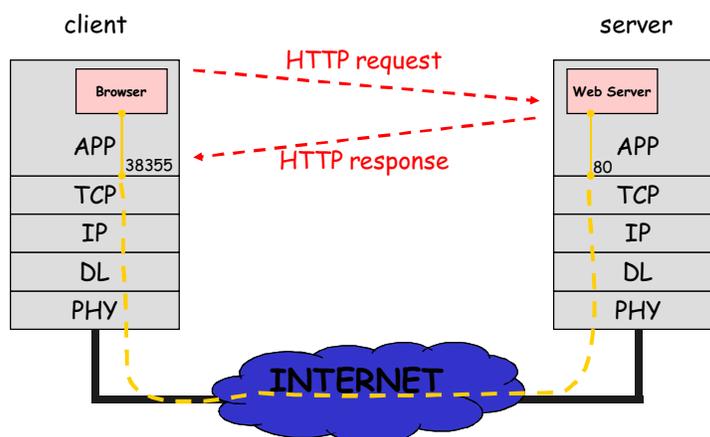
8

# 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>)

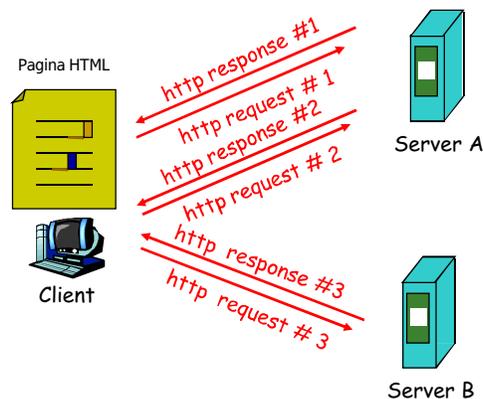
## HTTP per il trasferimento di pagine web



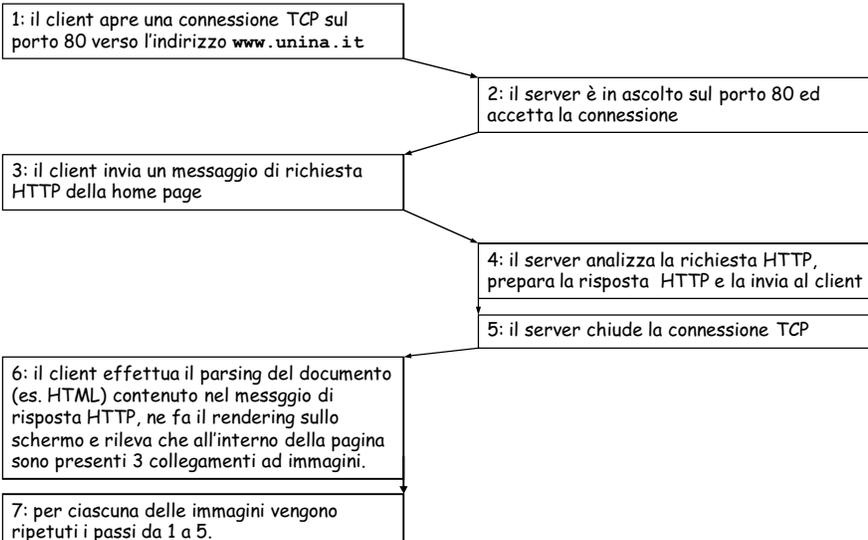
- Tipicamente, una pagina web è descritta da un file testuale in formato HTML (*Hypertext Markup Language*)
- La pagina è identificata mediante un indirizzo, detto URL
- Un file HTML può contenere riferimenti ad altri oggetti che arricchiscono la pagina con elementi grafici
  - Es. sfondo, immagini,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



17

## Connessioni persistenti e non persistenti



### non persistente

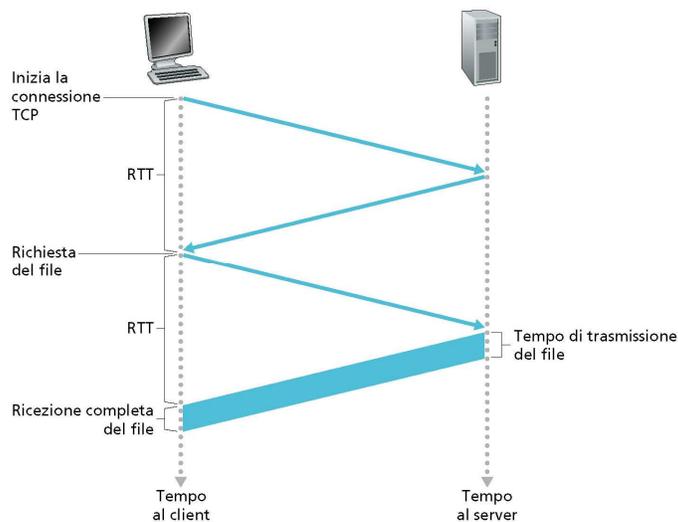
- HTTP/1.0 (RFC 1945)
- Il server analizza una richiesta, la serve e chiude la connessione
- 2 Round Trip Time (RTT) per ciascuna richiesta
- Ogni richiesta subisce lo slow-start TCP

### 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)

18

## 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

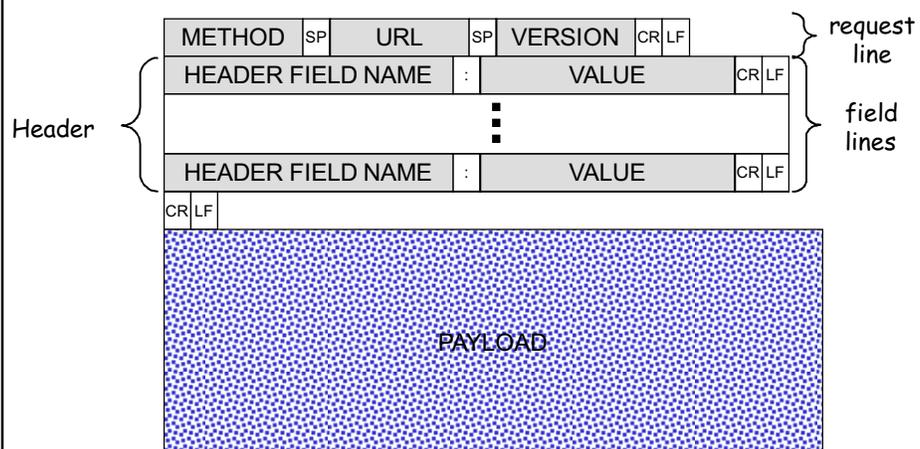
```
GET /path/pagename.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: it
```

RIGA VUOTA

indica la fine del messaggio

21

## Il messaggio HTTP/1.0 request



22

## Il messaggio HTTP/1.0 response



codice di stato

HTTP/1.0 200 OK
Date: Mon, 16 Dec 2002 14:00:22 GMT
Server: Apache/1.3.24 (Win32)
Last-Modified: Fri, 13 Dec 2002 08:06:44 GMT
Content-Length: 222
Content-Type: text/html
RIGA VUOTA
PAYLOAD

23

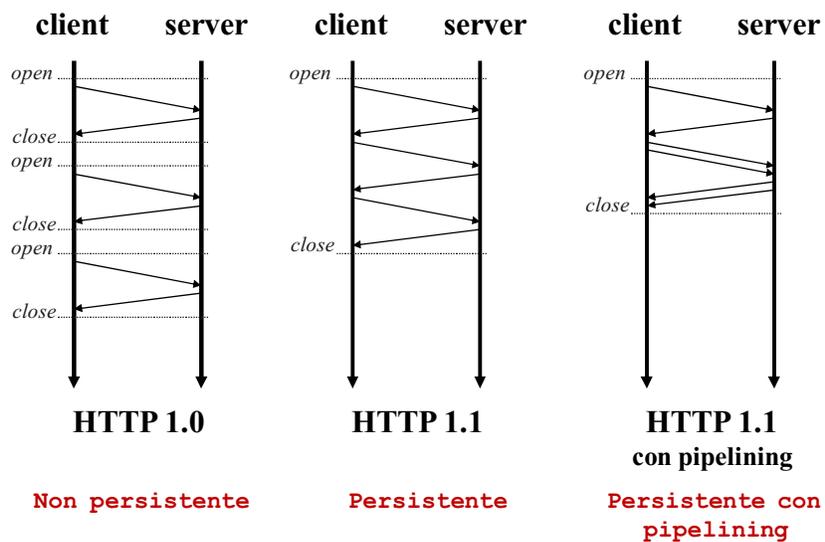
## Esempi di codici di stato



- 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

24

## 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



The screenshot displays a network capture in Wireshark. The packet list pane shows a GET request from 143.225.229.163 to 209.85.129.147 on port 4150. The packet details pane shows the request structure: GET / HTTP/1.1, Host: www.google.com, 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-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, Cookie: PREF=ID=42e83292038ae07:FF=4:L0=en:NR=10:CR=2:TM=1169647218:LM=1187706763:GM=1:S=zCPXwUB24XF7AyNk; S=awfeQ0Zp-KLS. The packet bytes pane shows the raw data of the request.

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

The screenshot displays a Wireshark capture of an HTTP POST request. The packet list shows several entries, with the selected packet (No. 6) being a POST request to /email.php. The packet details pane shows the following structure:

- 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 /email.php HTTP/1.1\r\n
  - Host: wpop8.liberof.it\r\n
  - User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it-IT; rv:1.7.12) Gecko/20050919 Firefox/1.0.7\r\n
  - Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5\r\n
  - Accept-Language: it, it-it;q=0.8,en-us;q=0.5,en;q=0.3\r\n
  - Accept-Encoding: gzip,deflate\r\n
  - Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7\r\n
  - Keep-Alive: 300\r\n
  - Connection: keep-alive\r\n
  - Referer: http://www.liberof.it/\r\n
  - Cookie: \_\_utmz=267072147.1190633699.2.3.utmcsrc=HP%2BLiberofutmccn=Infostrada%20ADS%20SLutmcmd=Lancfo1; Liberomail=2dccc1ab3e03

The raw data pane shows the hexadecimal and ASCII representation of the request body, starting with 'allve...eReferer:'.

## 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

33

## 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>
```

34

# HTTP Response: un esempio



35

# 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

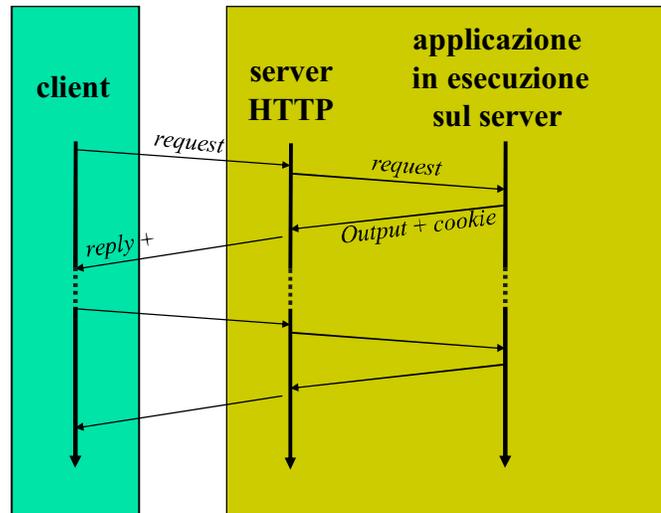
## I cookies



- **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