

Reti di Calcolatori I

Prof. Roberto Canonico

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Corso di Laurea in Ingegneria Informatica

A.A. 2019-2020

Il protocollo 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

- Concepito per la trasmissione di ipertesti da Tim-Berners Lee ed implementato per la prima volta al CERN
- 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**

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`

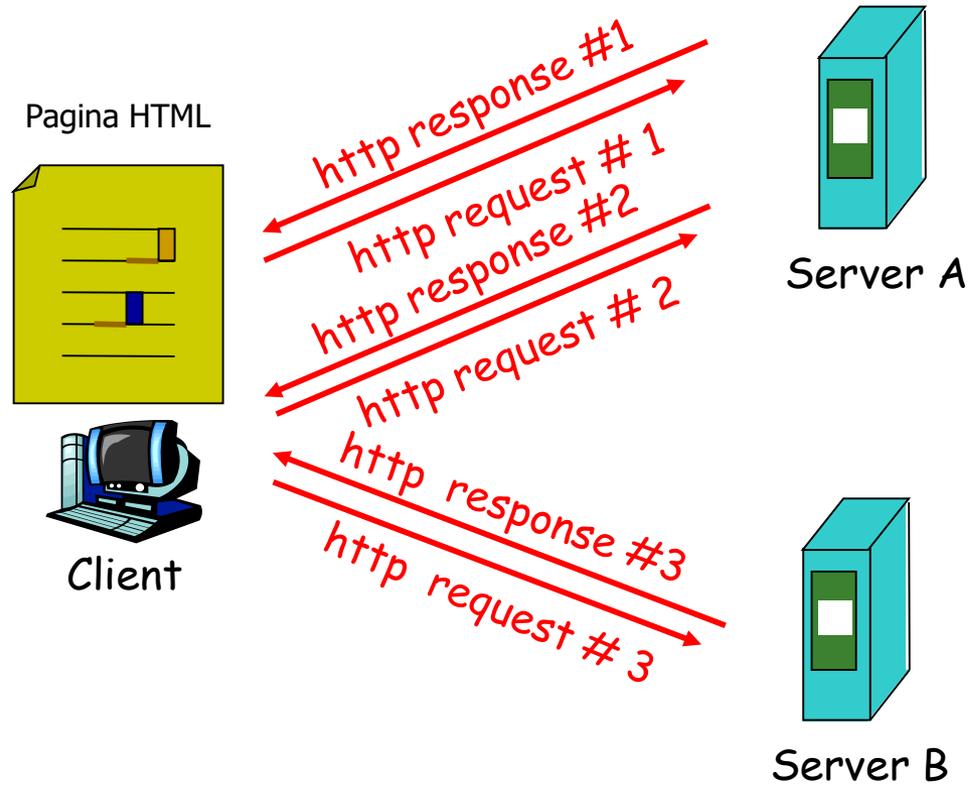
HTTP: versioni

- Il protocollo è definito in documenti RFC
- Diverse versioni:
 - HTTP/0.9
 - HTTP/1.0
 - RFC 1945 (del 1996)
 - HTTP/1.1
 - RFC 2068 (del 1997)
 - RFC 2616 (del 1999)
 - HTTP/2
 - RFC 7540 (del 2015)

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

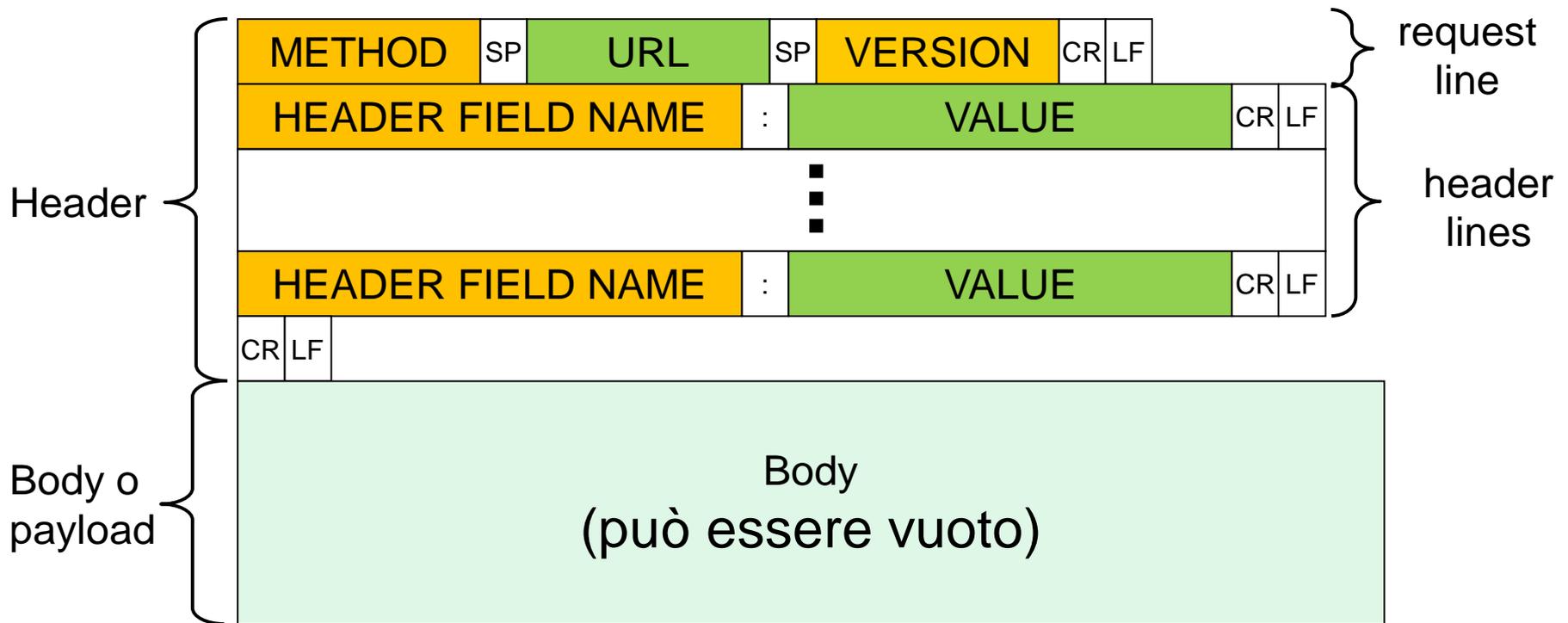
6: il client effettua il parsing del documento (es. HTML) contenuto nel messaggio di risposta HTTP, ne fa il rendering sullo schermo e rileva che all'interno della pagina sono presenti 3 collegamenti ad immagini.

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

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.)

HTTP: formato messaggi di richiesta



CR = Carriage Return = $0D_{16} = 13_{10}$
 LF = Line Feed = $0A_{16} = 10_{10}$

Richieste HTTP: metodi

- Una richiesta HTTP è caratterizzata dal campo Method nella Request Line
 - In HTTP/1.1 il campo Method può assumere i seguenti valori:
 - GET
 - HEAD
 - POST
 - PUT
 - DELETE
 - TRACE
 - OPTIONS
 - Noi descriveremo solo i metodi GET, HEAD, e POST
 - Un metodo HTTP può essere:
 - **Sicuro**: non genera cambiamenti allo stato interno del server
 - I metodi GET e HEAD sono sicuri
 - **Idempotente**: l'effetto sul server di più richieste identiche è lo stesso di quello di una sola richiesta
 - I metodi HTTP sono tutti idempotenti tranne POST
-

Il metodo GET

- Il metodo GET è usato per richiedere una risorsa identificata da un URL
 - Se la risorsa è disponibile, il server la invia nel body del messaggio di risposta
 - GET è un metodo sicuro ed idempotente
 - E' il metodo più frequentemente usato, ed è quello che viene attivato in un browser quando:
 - si fa click su un link ipertestuale di un documento HTML
 - si inserisce un URL nella barra degli indirizzi del 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
-

Un esempio di richiesta 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

Un esempio di richiesta GET

(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	3.368079	143.225.229.163	209.85.129.147	HTTP	GET / HTTP/1.1
6	3.402881	209.85.129.147	143.225.229.163	TCP	http > 4150 [ACK] Seq=1 Ack=574 win=6611 Len=0
7	3.415274	209.85.129.147	143.225.229.163	TCP	[TCP segment of a reassembled PDU]
8	3.415291	209.85.129.147	143.225.229.163	HTTP	HTTP/1.1 200 OK (text/html)
9	3.415314	143.225.229.163	209.85.129.147	TCP	4150 > http [ACK] Seq=574 Ack=2544 win=65535 [TCP CHECKSUM INCORR]
10	6.038681	85.182.14.100	143.225.92.106	TCP	4890 > 27047 [SYN] Seq=0 Len=0 MSS=1452

⊕ Frame 5 (627 bytes on wire (627 bytes captured) on interface 0)

⊕ Ethernet II, Src: wistron_2e:cb:5b (00:0a:e4:2e:cb:5b), Dst: Cisco_6d:04:00 (00:14:f1:6d:04:00)

⊕ Internet Protocol, Src: 143.225.229.163 (143.225.229.163), Dst: 209.85.129.147 (209.85.129.147)

⊕ Transmission Control Protocol, Src Port: 4150 (4150), Dst Port: http (80), Seq: 1, Ack: 1, Len: 573

⊖ Hypertext Transfer Protocol

⊕ GET / HTTP/1.1\r\n

Host: www.google.com\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

Cookie: PREF=ID=42ee83292038ae07:FF=4:LD=en:NR=10:CR=2:TM=1169647218:LM=1187706763:GM=1:S=zCPXwubZ4XF7AyNX; S=awfe=QDzp-Kks\r\n

```

0040  2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 77 77 77 2e  /1.1..Host: www.
0050  67 6f 6f 67 6c 65 2e 63 6f 6d 0d 0a 55 73 65 72  google.com..User
0060  2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c 61 2f  -Agent: Mozilla/
0070  35 2e 30 20 28 57 69 6e 64 6f 77 73 3b 20 55 3b  5.0 (windows; U;
0080  20 57 69 6e 64 6f 77 73 20 4e 54 20 35 2e 31 3b  windows NT 5.1;
0090  20 69 74 2d 49 54 3b 20 72 76 3a 31 2e 37 2e 31  it-IT; rv:1.7.1
00a0  32 29 20 47 65 63 6b 6f 2f 32 30 30 35 30 39 31  2) Gecko/2005091
00b0  39 20 46 69 72 65 66 6f 78 2f 31 2e 30 2e 37 0d  9 Firefox/1.0.7.
00c0  0a 41 63 63 65 70 74 3a 20 74 65 78 74 2f 78 6d  .Accept: text/xml
  
```

[P: 10 D: 10 M: 0 Drops: 0]

Il metodo HEAD

- Il metodo HEAD è simile a GET: si richiede la validità di una risorsa identificata da un URL
 - A differenza di GET, nel caso di HEAD la risorsa non viene inviata nel messaggio di risposta
 - HEAD è un metodo sicuro ed idempotente
 - HEAD serve al client per verificare:
 - Validità di un URL e per apprendere le caratteristiche della risorsa rappresentate nell'header del messaggio di risposta
 - Accessibilità di un URL, cioè per verificare se l'accesso ad una risorsa non sia vincolato ad una autenticazione
 - Verifica di coerenza di una copia della risorsa già disponibile nella cache del browser rispetto all'originale sul server
-

Il metodo POST

- Il metodo POST è usato per trasmettere informazioni dal client al server senza la creazione di una nuova risorsa
 - L'URL presente nel messaggio di richiesta è associato ad un programma eseguito sul server che riceve come input le informazioni inviate dal client nel body del messaggio di richiesta
 - POST è un metodo non sicuro e non idempotente
 - 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
-

Un esempio di richiesta POST

(Untitled) - Ethereal

File Edit View Go Capture Analyze Statistics Help

Filter: http

No.	Time	Source	Destination	Protocol	Info
6	0.107593	192.168.1.2	212.52.84.18	HTTP	POST /email.php HTTP/1.1
8	0.185827	192.168.1.2	212.52.84.18	HTTP	Continuation of non-HTTP traffic (application/x-www-form-urlencoded)
14	5.406089	212.52.84.18	192.168.1.2	HTTP	HTTP/1.1 200 OK (text/html)
16	5.426579	192.168.1.2	212.52.84.18	HTTP	GET /favicon.ico HTTP/1.1
19	5.502876	212.52.84.18	192.168.1.2	HTTP	HTTP/1.1 200 OK (image/x-icon)
22	5.517376	192.168.1.2	212.52.84.18	HTTP	GET /error.html HTTP/1.1
25	5.611096	212.52.84.18	192.168.1.2	HTTP	HTTP/1.1 200 OK (text/html)
33	5.836657	192.168.1.2	195.210.87.1	HTTP	GET /cgi-bin/adv_getbanner.js?sito=Mail&tipo=liberobg&target=
38	5.931878	195.210.87.1	192.168.1.2	HTTP	HTTP/1.1 200 OK (text/javascript)
41	5.966838	192.168.1.2	212.52.84.18	HTTP	GET /xam_rc/template_lowend/back1.htm HTTP/1.1
42	5.968054	192.168.1.2	212.52.84.18	HTTP	GET /xam_rc/template_lowend/header_attJM.htm HTTP/1.1

Frame 6 (881 bytes on wire, 881 bytes captured)

Ethernet II, Src: wistron_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.libero.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.libero.it/\r\n

Cookie: __utmz=267072147.1190633699.2.3.utmcsr=HP%2BLiber o|utmccn=Infostrada%20ADSL|utmcmd=Lancio1; Liberomail=2dccd1ab3e03

```

01e0 61 6c 69 76 65 0d 0a 52 65 66 65 72 65 72 3a 20  alive..R referer:
01f0 68 74 74 70 3a 2f 2f 77 77 77 2e 6c 69 62 65 72  http://w ww.libero
0200 6f 2e 69 74 2f 0d 0a 43 6f 6f 6b 69 65 3a 20 5f  o.it/.c ookie: _
0210 5f 75 74 6d 7a 3d 32 36 37 30 37 32 31 34 37 2e  _utmz=26 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  tmcsr=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%20AD SL|utmcm
0260 64 3d 4c 61 6e 63 69 6f 31 3b 20 4c 69 62 65 72  d=Lancio 1; Liber
0270 6f 6d 61 69 6c 3d 37 64 63 63 64 31 61 67 33 65  nmail=2d ccd1ab3e
  
```

HTTP Referer (http.referer), 32 bytes

P: 240 D: 65 M: 0 Drops: 0

Invio di dati da un client ad un server

- In una applicazione web, l'invio di dati da un client ad un server può avvenire sia usando il metodo GET che il metodo POST

get oppure post

- Esempio di form HTML

```
<form action="/action_page.php" method="post">  
  First name: <input type="text" name="fname"><br>  
  Last name: <input type="text" name="lname"><br>  
  <input type="submit" value="Submit">  
</form>
```

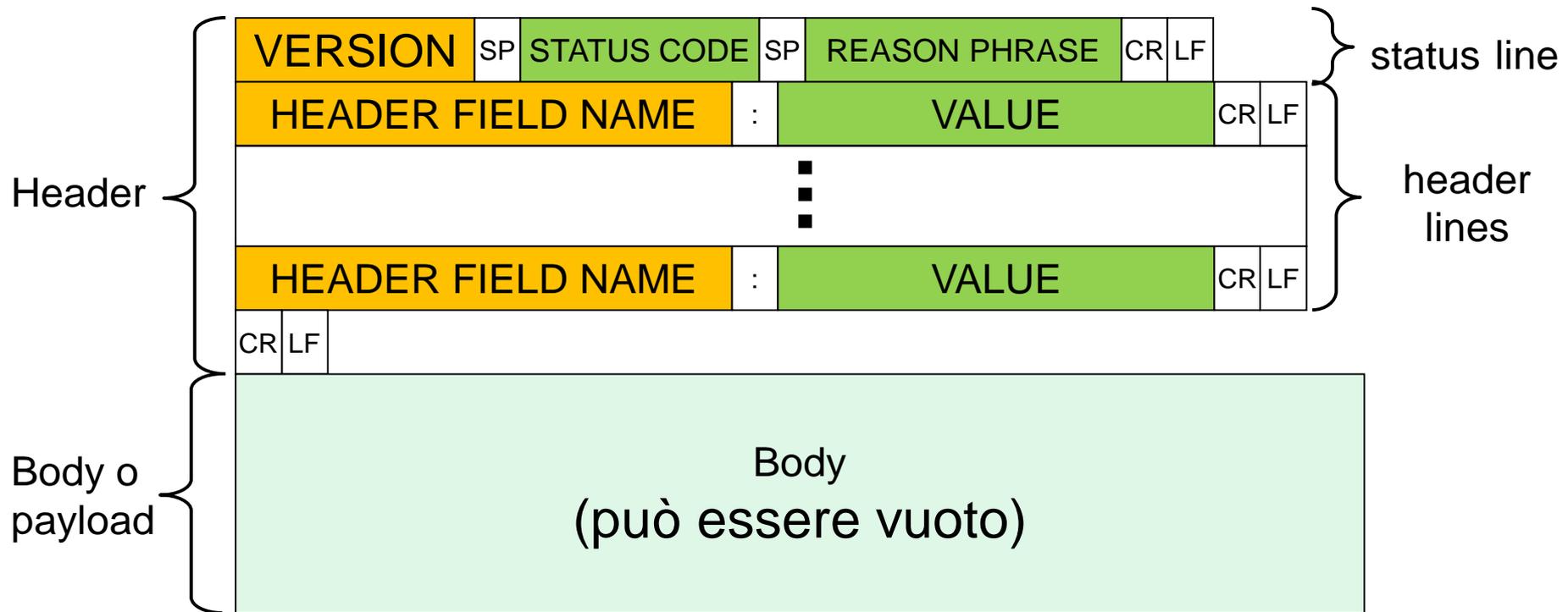
- Verificare la differenza sul sito

- https://www.w3schools.com/tags/att_form_method.asp

Il metodo PUT

- Il metodo PUT serve per trasmettere delle informazioni dal client al server, creando o sostituendo la risorsa specificata dall'URL
 - In caso di risposta positiva ad una richiesta PUT, i dati inviati nel body del messaggio PUT dovrebbero costituire il valore della risorsa che il server restituisce ad una successiva richiesta GET per la stessa URL
 - Per motivi di sicurezza, questo metodo è scarsamente utilizzato
- PUT è un metodo non sicuro ma idempotente

HTTP: formato messaggi di risposta



CR = Carriage Return = $0D_{16} = 13_{10}$
 LF = Line Feed = $0A_{16} = 10_{10}$

Un esempio di messaggio di risposta

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

Un esempio di messaggio di risposta

(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 INCORRI
5	3.368079	143.225.229.163	209.85.129.147	HTTP	GET / HTTP/1.1
6	3.402881	209.85.129.147	143.225.229.163	TCP	http > 4150 [ACK] Seq=1 Ack=574 win=6611 Len=0
7	3.415274	209.85.129.147	143.225.229.163	TCP	[TCP segment of a reassembled PDU]
8	3.415291	209.85.129.147	143.225.229.163	HTT	HTTP/1.1 200 OK (text/html)
9	3.415314	143.225.229.163	209.85.129.147	TCP	4150 > http [ACK] Seq=574 Ack=2544 win=65535 [TCP CHECKSUM II
10	6.038681	85.182.14.100	143.225.92.106	TCP	4890 > 27047 [SYN] Seq=0 Len=0 MSS=1452

+ Frame 8 (1167 bytes on wire, 1167 bytes captured)
 + Ethernet II, Src: Cisco_6d:04:00 (00:14:f1:6d:04:00), Dst: wistron_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: 4150 (4150), Seq: 1431, Ack: 574, Len: 1113
 + [Reassembled TCP segments (2543 bytes): #7(1430), #8(1113)]
 + Hypertext Transfer Protocol
 + HTTP/1.1 200 OK\r\n
 Cache-Control: private\r\n
 Content-Type: text/html; charset=UTF-8\r\n
 Content-Encoding: gzip\r\n
 Server: gws\r\n
 Content-Length: 2364\r\n
 Date: Mon, 24 Sep 2007 10:25:21 GMT\r\n
 \r\n
 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 81 b6 c6 00 00 33 06 04 43 d1 55 81 93 8f e1  ....3. .C.U....
0020  e5 a3 00 50 10 36 a6 a2 e3 00 f7 46 f9 c0 50 18  ...P.6. ...F..P.
0030  19 d3 82 fa 00 00 d4 67 d5 49 d0 32 35 8c 5e 2f  ....g .I.25.A/
0040  e4 85 45 be 74 74 be 84 3f a4 93 b3 03 79 5c ff  ..E.tt. ?....y\
0050  21 58 ee b9 69 50 8a 96 a6 35 99 0c c9 89 87 88  !X..iP.. .5.....
0060  69 05 41 a1 28 13 28 10 20 56 74 ee b3 bf 96 c3  i.A.(.( vt.....
0070  ea 0c d5 36 79 ea a5 65 7a 7c d8 d0 9f 85 0c e5  6) e 7 @
  
```

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

File: "C:\DOCUME~1\dis\IMPOST~1\Temp\ether\XXXXN288YT" 3876 Bytes 00:00:06 | P: 10 D: 10 M: 0 Drops: 0

Status code

- Lo status code è un numero di tre cifre, di cui la prima indica la classe della risposta, e le altre due indicano 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 indica la nuova posizione (Location:)
- **400 Bad request**
 - errore sintattico nella richiesta
- **401 Unauthorized**
 - manca l'autorizzazione
- **403 Forbidden**
 - richiesta non autorizzabile
- **404 Not found**
 - L'oggetto richiesto non è presente sul server, URL non valido
- **500 Internal server error**
 - tipicamente un programma in esecuzione sul server ha generato errore
- **501 Not implemented**
 - metodo non conosciuto dal server
- **505 HTTP Version Not Supported**
 - La versione del protocollo HTTP usata non è supportata dal server

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 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)

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

Una prova con telnet

```
> telnet wpage.unina.it 80
```

```
GET /rcanonical/ HTTP/1.0
```

Request-line

```
HTTP/1.1 200 OK
```

```
Date: Wed, 14 Mar 2018 08:39:32 GMT
```

```
Server: Apache
```

```
Last-Modified: Thu, 22 Oct 2015 16:04:41 GMT
```

```
ETag: "1a754-2fa-a711b840"
```

```
Accept-Ranges: bytes
```

```
Content-Length: 762
```

```
Connection: close
```

```
Content-Type: text/html; charset=ISO-8859-1
```

Response

```
<HTML>
```

```
...
```

```
...
```

```
</HTML>
```

```
Connessione all'host perduta.
```

DOCUMENTO HTML

Una prova con telnet (2)

```
> telnet wpage.unina.it 80
```

Request-line

```
GET /rcanonical/didattica/rc/images/new.gif HTTP/1.0
```

```
HTTP/1.1 200 OK
```

```
Date: Wed, 14 Mar 2018 08:40:40 GMT
```

```
Server: Apache
```

```
Last-Modified: Tue, 12 Oct 2010 13:17:52 GMT
```

```
ETag: "fe0068-90-4dd80000"
```

```
Accept-Ranges: bytes
```

```
Content-Length: 144
```

```
Connection: close
```

```
Content-Type: image/gif
```

Response

```
GIF89a ▶í 777!¨♦☺☻, ▶⊙aöÅÇ+⌚
```

```
Ô⊙⌊┘YÔK!S■Y]ÍÉ┘.àcòâPÈ«≡<nÉ¬FQÁ¥P||ÒÁâÁôîédíX⌚ óªµj||t(
```

```
W¼rèàDãdn$8$Ü
```

```
~é_ ; HQê².üÍ↓î;^▣Ó▼
```

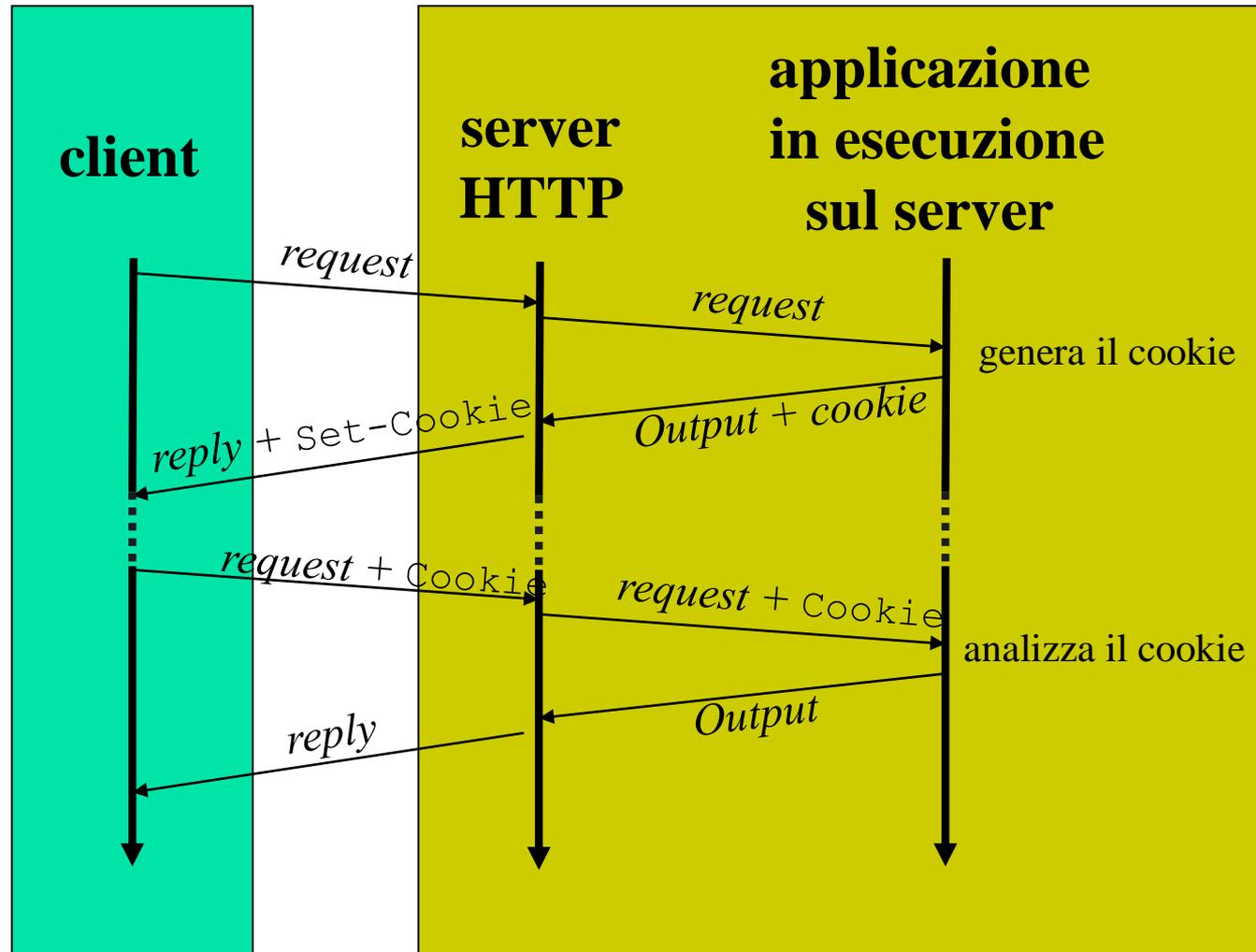
```
Connessione all'host perduta.
```

IMMAGINE GIF

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 cookie sono stati definiti originariamente nel 1997 in RFC2109
 - Attualmente definiti in RFC 2965 "HTTP State Management Mechanism"
 - Negli RFC che definiscono HTTP non si menzionano i cookie

Cookies (2)



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

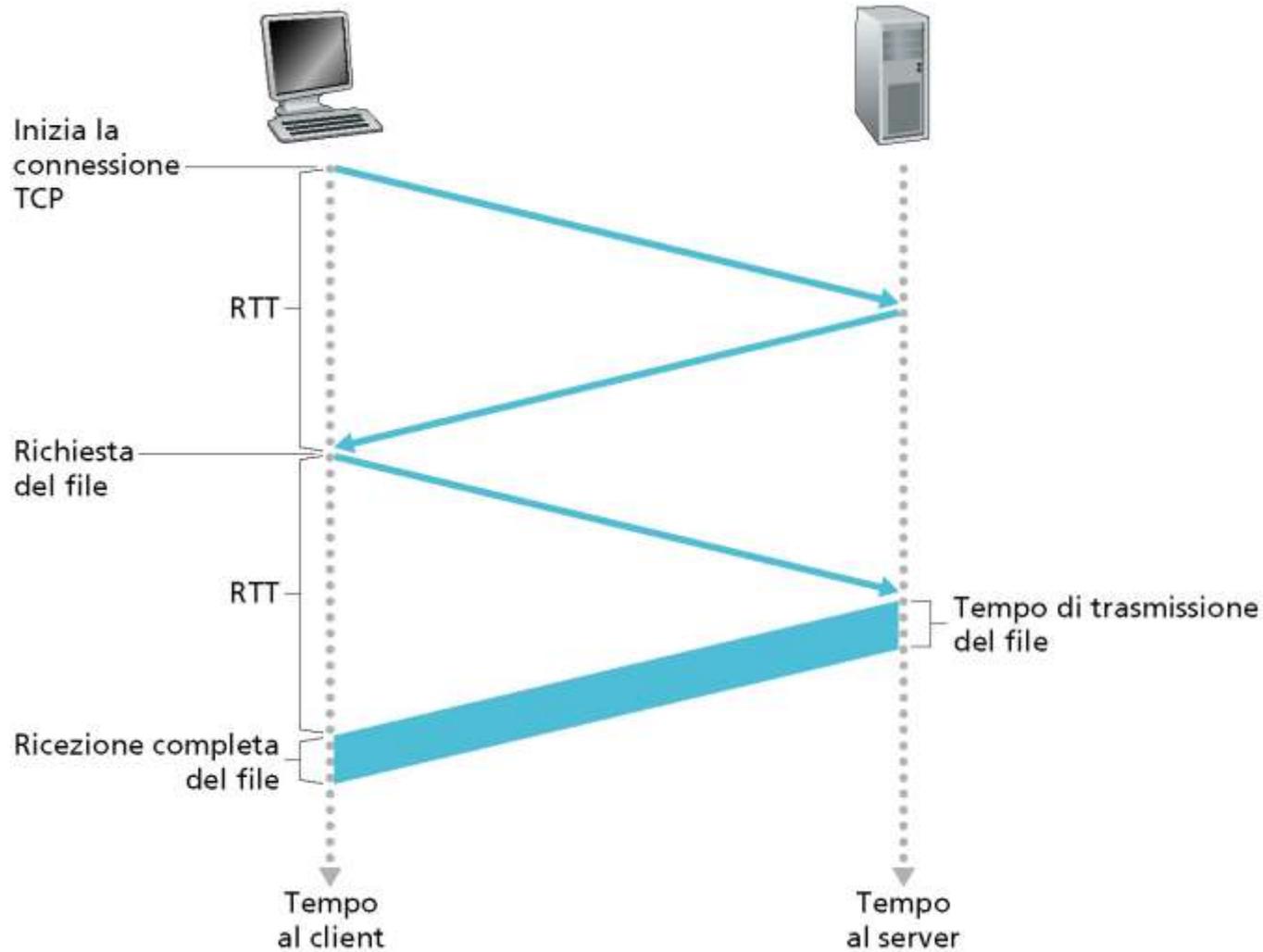
Cookies: attributi di durata

- In aggiunta ad un nome ed un valore, un cookie può avere degli attributi che ne specificano la durata o lo “scope”
 - Questi attributi sono specificati solo nei messaggi di risposta del server e non sono inviati dal browser
- Gli attributi che definiscono la durata di un cookie sono **Expires** e **Max-Age**:
 - L'attributo **Expires** specifica una data oltre la quale il browser deve eliminare il cookie
 - **Set-Cookie:** `sessionToken=abc123; Expires=Sat, 09 Jun 2019 10:18:14 GMT`
 - L'attributo **Max-Age** specifica dopo quanto tempo (in secondi), rispetto al momento della ricezione, il browser dovrà eliminare il cookie
 - **Set-Cookie:** `sessionToken=abc123; Max-Age=300`
 - L'attributo **Max-Age** non è supportato da Internet Explorer
- Un cookie per il quale non è specificata la durata è detto un *session cookie* e dura finché il browser non è chiuso

Cookies: attributi di scope

- L'attributo Domain indica i domini DNS a cui si applica il cookie
 - Se non è specificato, il browser invia il cookie a tutte le successive richieste fatte al server dal quale il cookie è stato ricevuto
 - Ad esempio, se il server docs.foo.com invia il cookie:
 - Set-Cookie: HSID=AYQEVn...DKrdst; Domain=.foo.com;
 - il cookie sarà inviato a tutte le successive richieste fatte ai server del dominio foo.com (es. anche ad images.foo.com)
 - Se Domain non è specificato, il cookie è inviato solo al server docs.foo.com
- L'attributo Path indica a quali oggetti (URL) si deve applicare il cookie
 - Se Path=/ il cookie si applica a tutti gli oggetti

Round Trip Time e connessioni HTTP



Connessioni persistenti e non persistenti

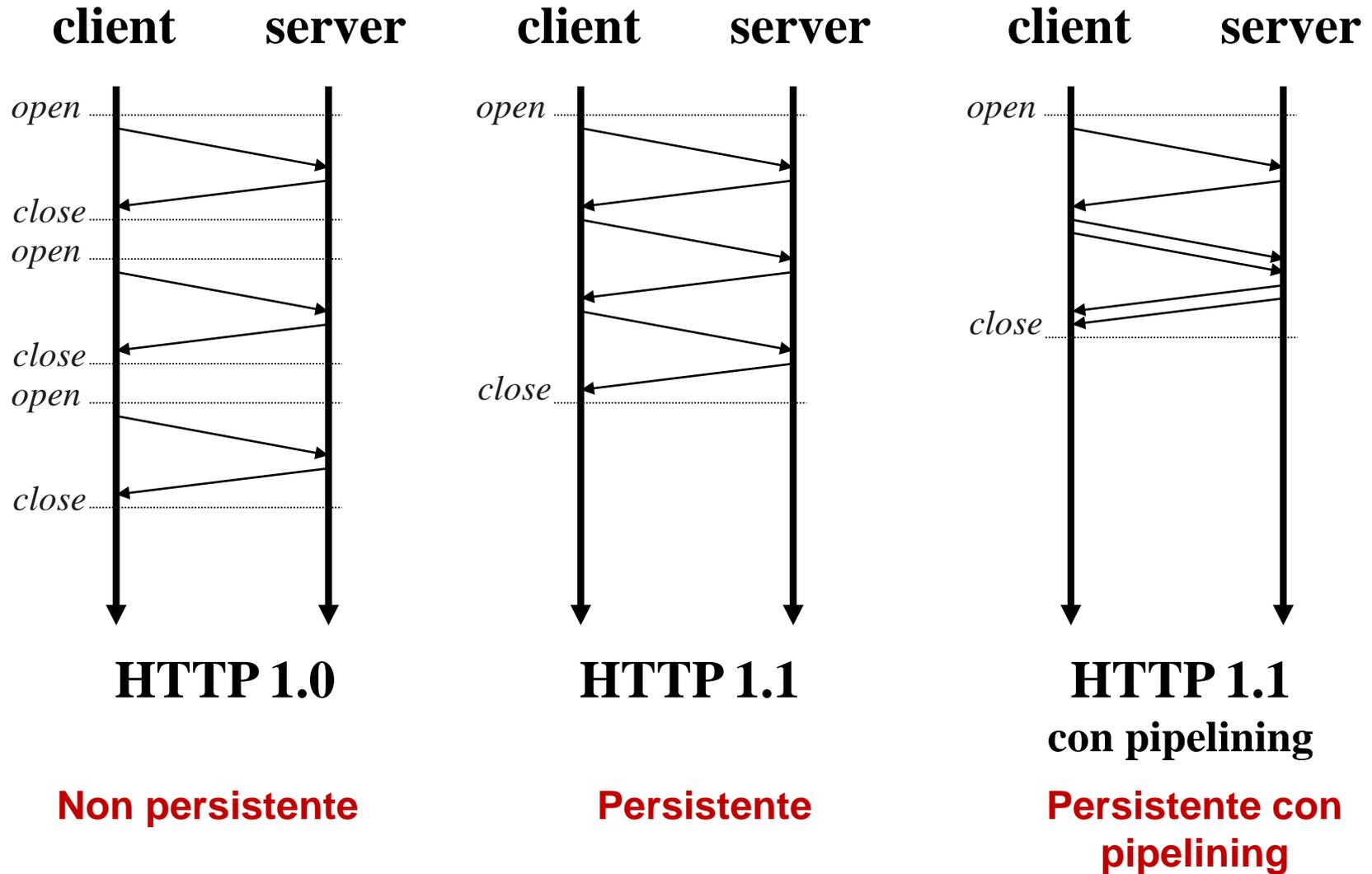
non persistente

- 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 (vedremo in seguito)

persistente

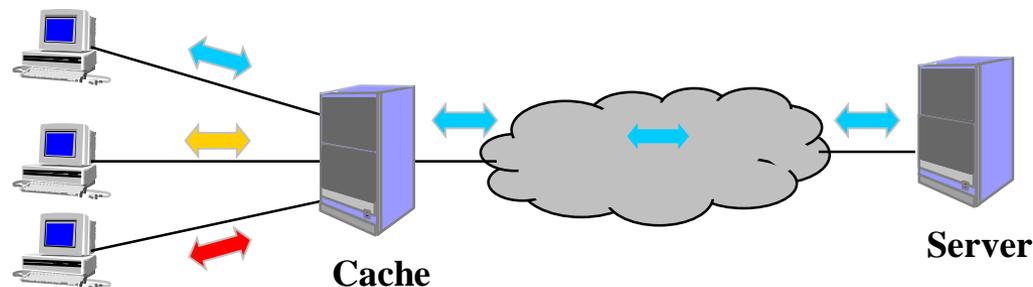
- HTTP/1.1 (RFC 2068, RFC 2116)
- Il server mantiene attiva una connessione TCP fino allo scadere di un timeout
- Il client può usare una connessione TCP precedentemente creata con il server per inviare richieste HTTP consecutive
- Si hanno meno RTT
- Esiste anche una versione con parallelismo (with pipelining)

La connessione HTTP

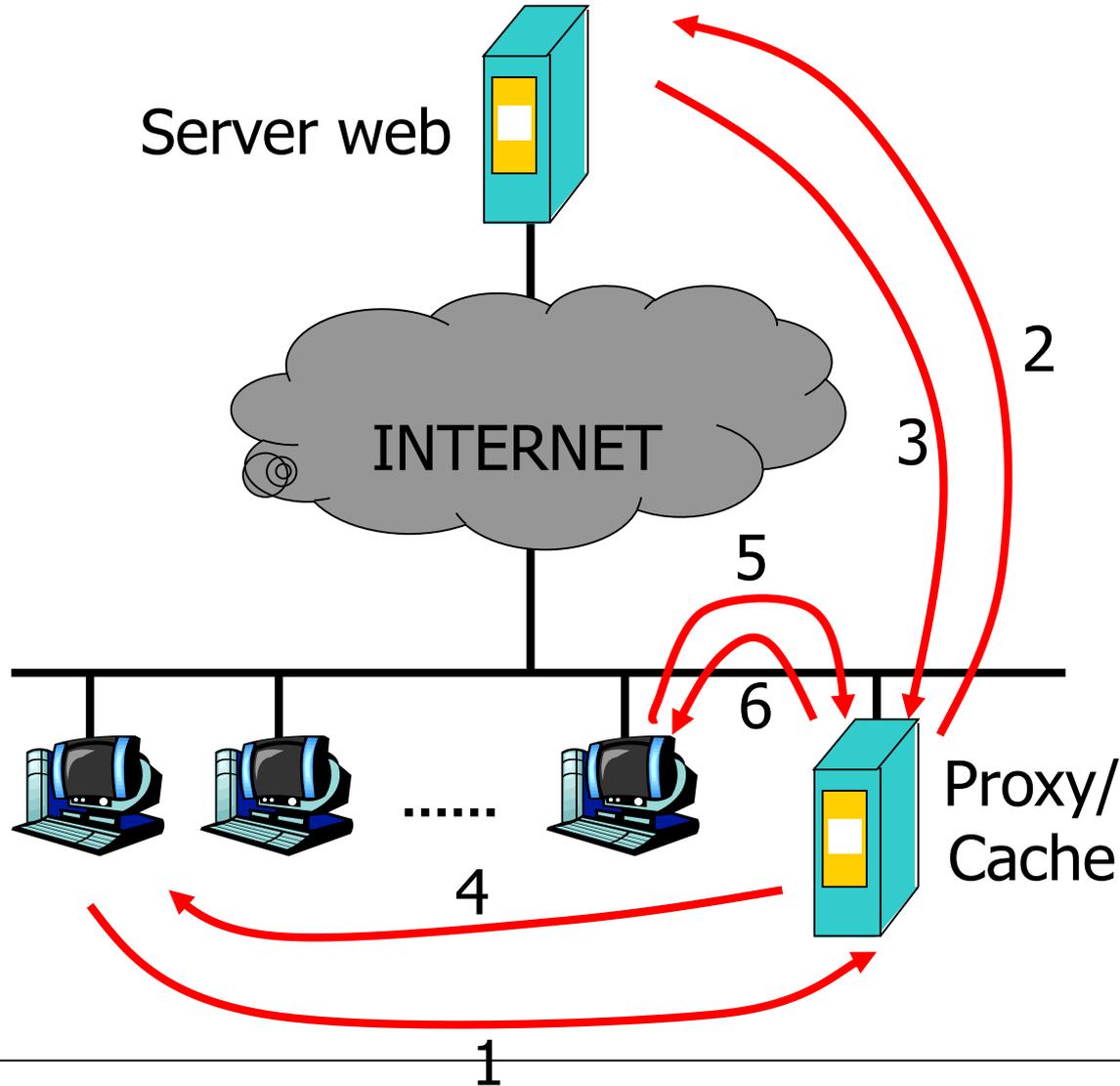


Web caching

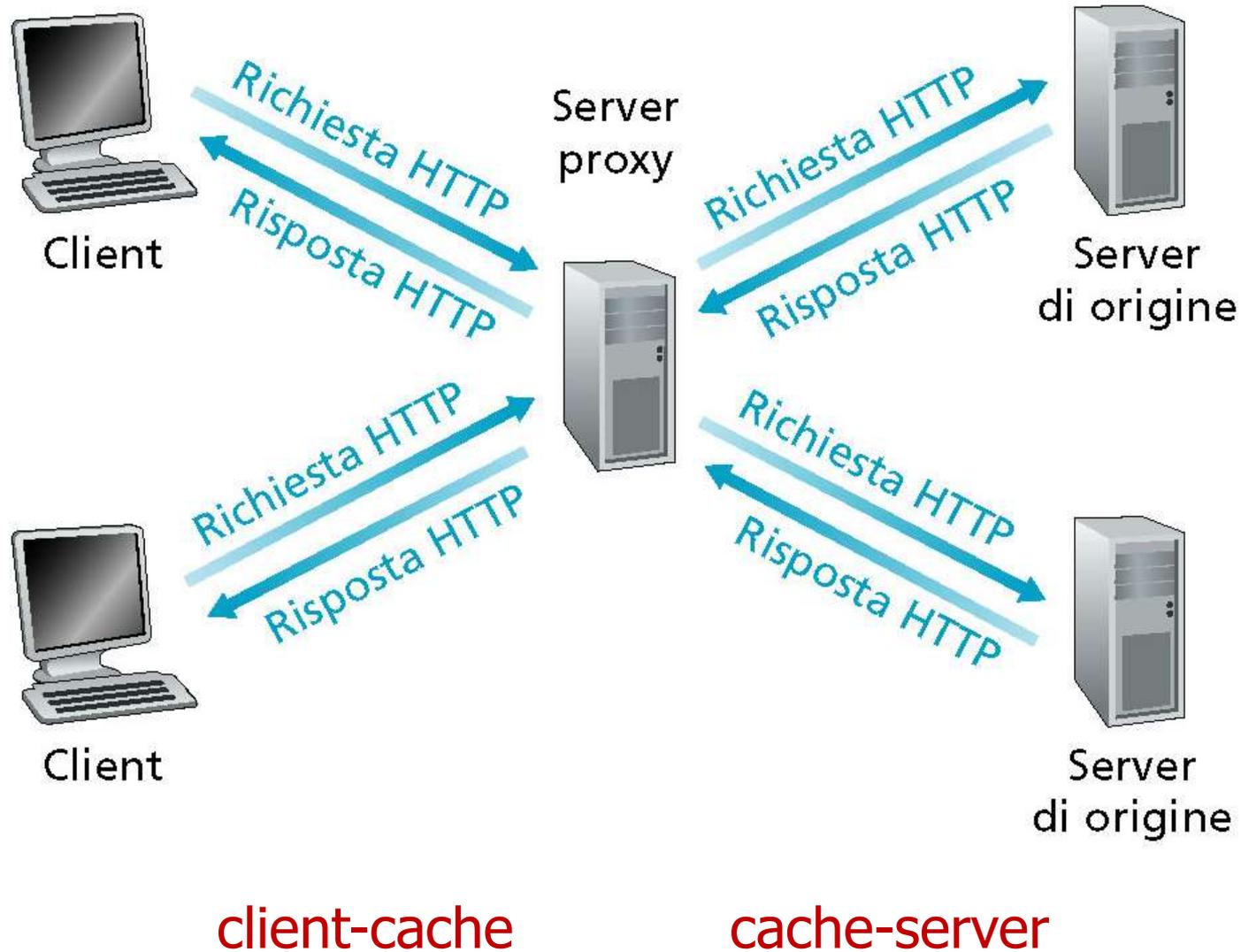
- Si parla genericamente di Web caching quando le richieste di un determinato client non raggiungono il Web Server, ma vengono intercettate da una **cache**
- Tipicamente, un certo numero di client di una stessa rete condivide una stessa cache web, posta nelle loro prossimità (es. nella stessa LAN)
-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



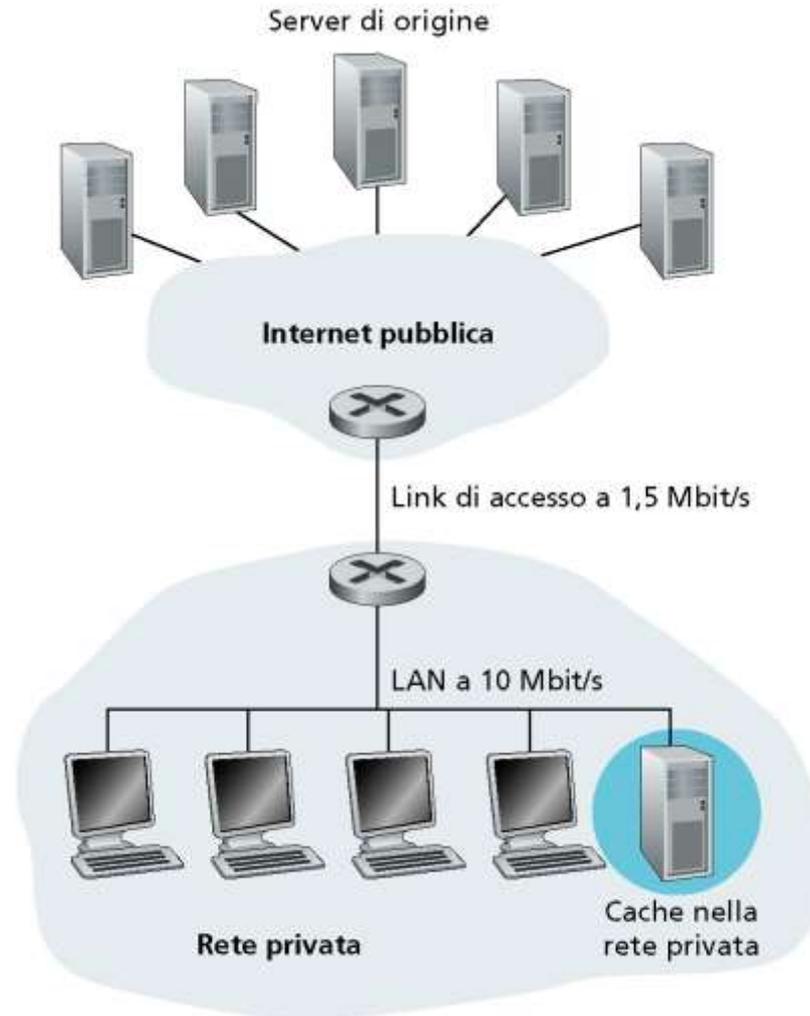
Transazioni web con caching



Server proxy: schema logico

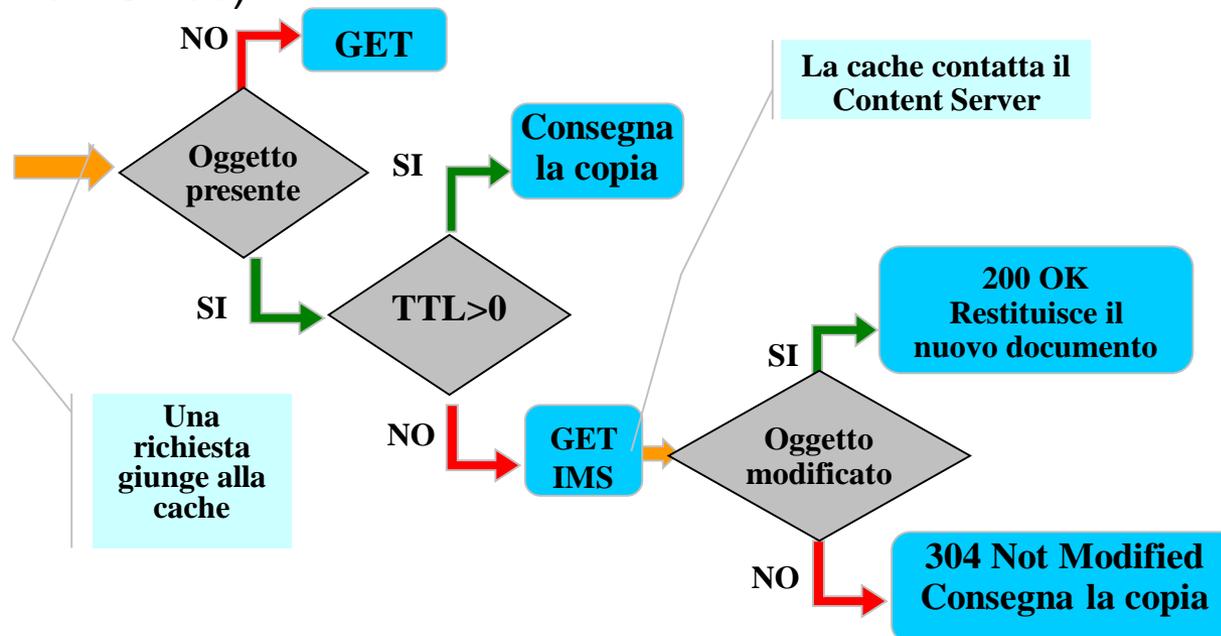


Server proxy in una rete di accesso



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*)



Browser web

- Esistono oggi una molteplicità di browser
 - Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari, Opera, ...
 - In passato NCSA Mosaic, Internet Explorer, Netscape Navigator, ...



- Un browser è costituito da:
 - un *browser engine*
 - un'interfaccia utente (UI)
- Un *browser engine* è, a sua volta, composto da
 - Un *layout engine* che decodifica e visualizza il documento HTML e gli oggetti multimediali in essa contenuti, tenendo in conto le indicazioni contenute in file CSS che determinano l'aspetto grafico (stile)
 - Un *JavaScript engine* che esegue il codice JavaScript incapsulato nel documento HTML o contenuto in altri file esterni (file .js)

Browser engine

- Nel corso degli anni sono stati sviluppati diversi browser engine
 - WebKit usato da Apple Safari e Chrome (fino alla versione 27)
 - Blink usato da Google Chrome (da v.28), deriva da WebKit
 - Gecko usato da Firefox
 - Trident usato da Internet Explorer, anche noto come MSHTML
 - EdgeHTML usato da Microsoft Edge, derivato da Trident
- Tranne EdgeHTML/Trident, gli altri engine sono open source
- Alcuni browser engine sono utilizzati da diversi browser
 - Ad esempio, il browser engine Blink è anche usato da:
 - Chromium (versione open-source di Chrome),
 - Oculus,
 - Brave,
 - Vivaldi,
 - Amazon Silk,
 - Opera ,

HTTP: non solo browsing...

- HTTP non è utilizzato solo per il Web
- Ad esempio:
 - Web Services e SOA (Service Oriented Architecture)
 - Video streaming
 - DASH: Dynamic Adaptive Streaming over HTTP
 - Peer-to-peer