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

Università degli Studi di Napoli Federico II

Nota di copyright per le slide COMICS

Nota di Copyright

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

Università degli Studi di Napoli
Federico II

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`

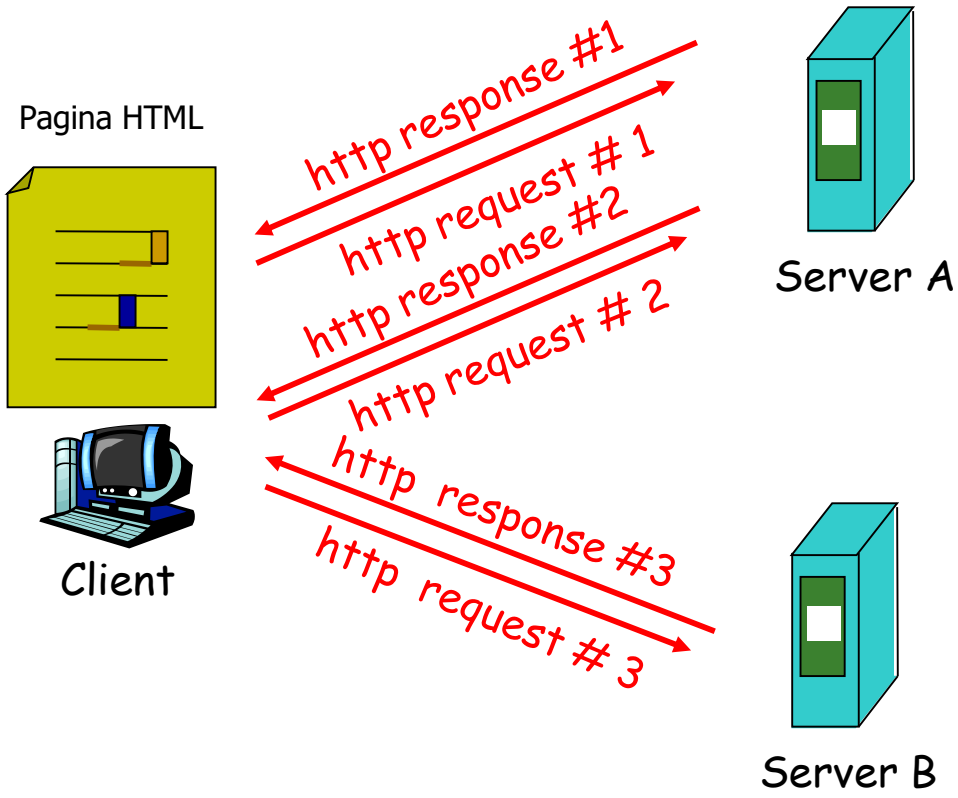
HTTP: versioni

- 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 messggio di risposta HTTP, ne fa il rendering sullo schermo e rileva che all'interno della pagina sono presenti 3 collegamenti ad immagini.

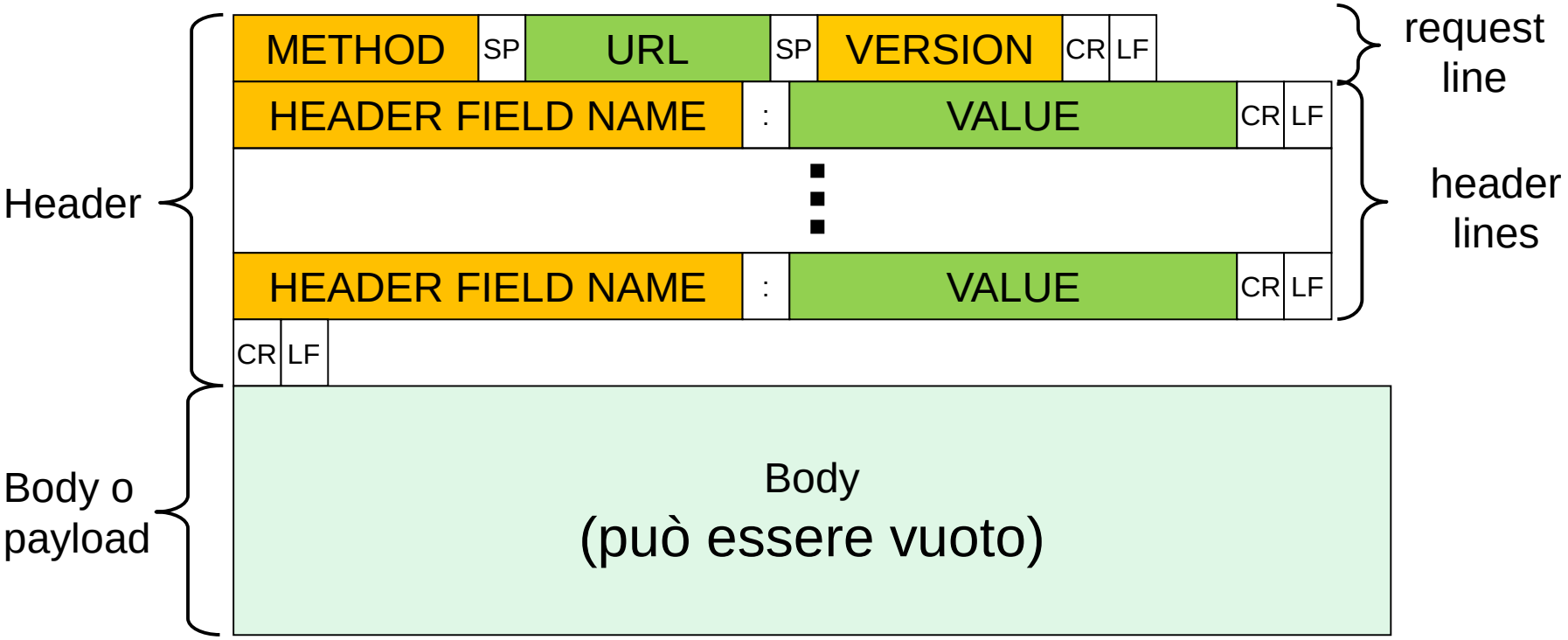
7: per ciascuna delle immagini vengono ripetuti i passi da 1 a 5.

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.)

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

HTTP: formato messaggi di richiesta



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

Richieste HTTP: metodi

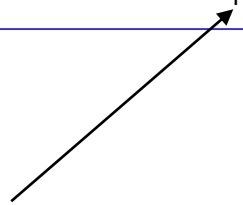
- Una richiesta HTTP è caratterizzata dal campo Method nella Request Line
- In HTTP/1.1 il campo Method può assumere i seguenti valori:
  - GET
  - HEAD
  - POST
  - PUT
  - DELETE
  - TRACE
  - OPTIONS
- Noi descriveremo solo i metodi GET, HEAD, e POST
- Un metodo HTTP può essere:
  - **Sicuro**: non 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

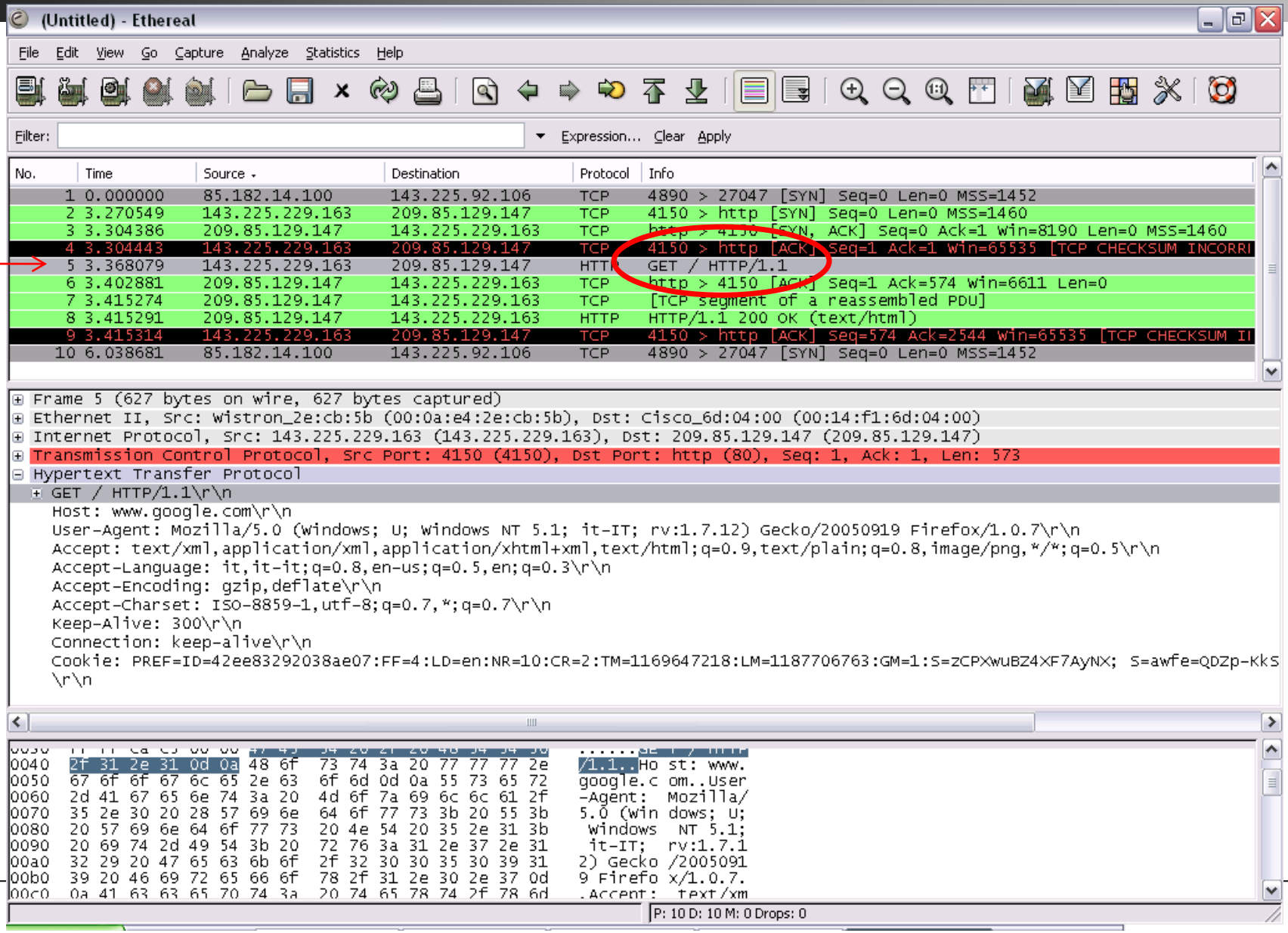
Un esempio di richiesta GET

`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



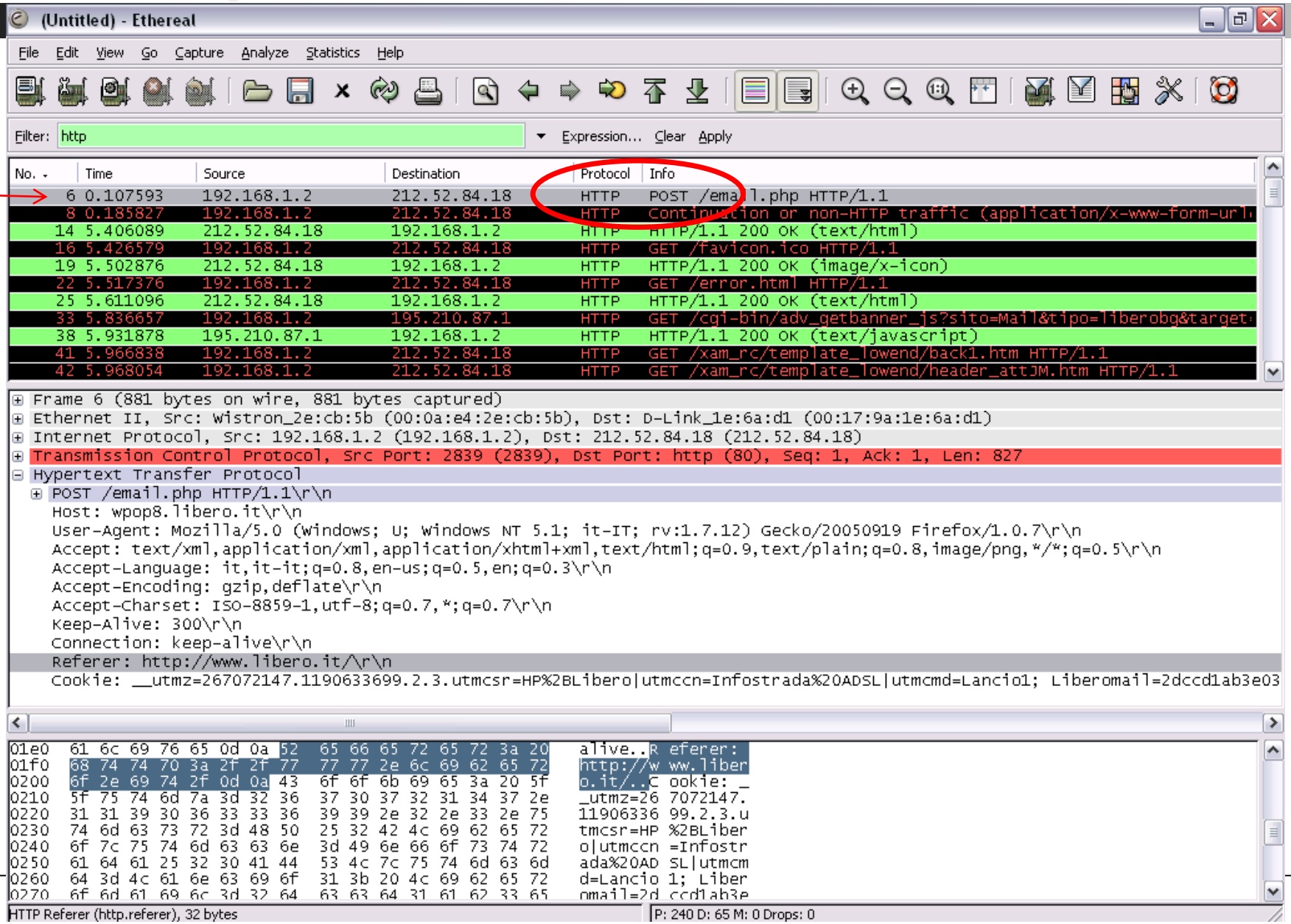
Il metodo HEAD

- 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

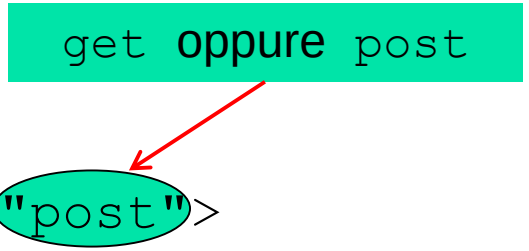
- Il metodo POST è usato per trasmettere informazioni dal client al server senza la creazione di una nuova risorsa
  - L'URL presente nel messaggio di richiesta è associato ad un programma eseguito sul server che riceve come input le informazioni inviate dal client nel body del messaggio di richiesta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



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
- Esempio di form HTML



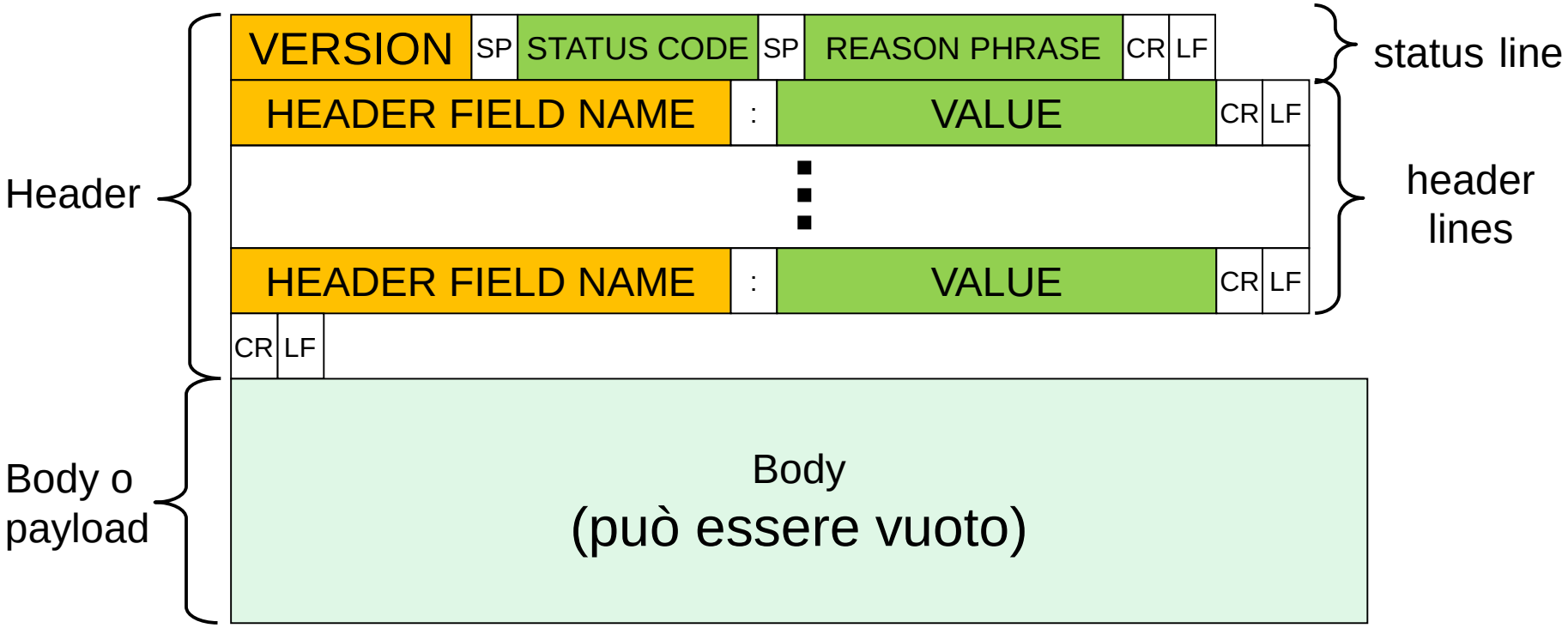

```
<form action="/action_page.php" method="post">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit">
</form>
```

- Verificare la differenza sul sito
  - https://www.w3schools.com/tags/att_form_method.asp

Il metodo PUT

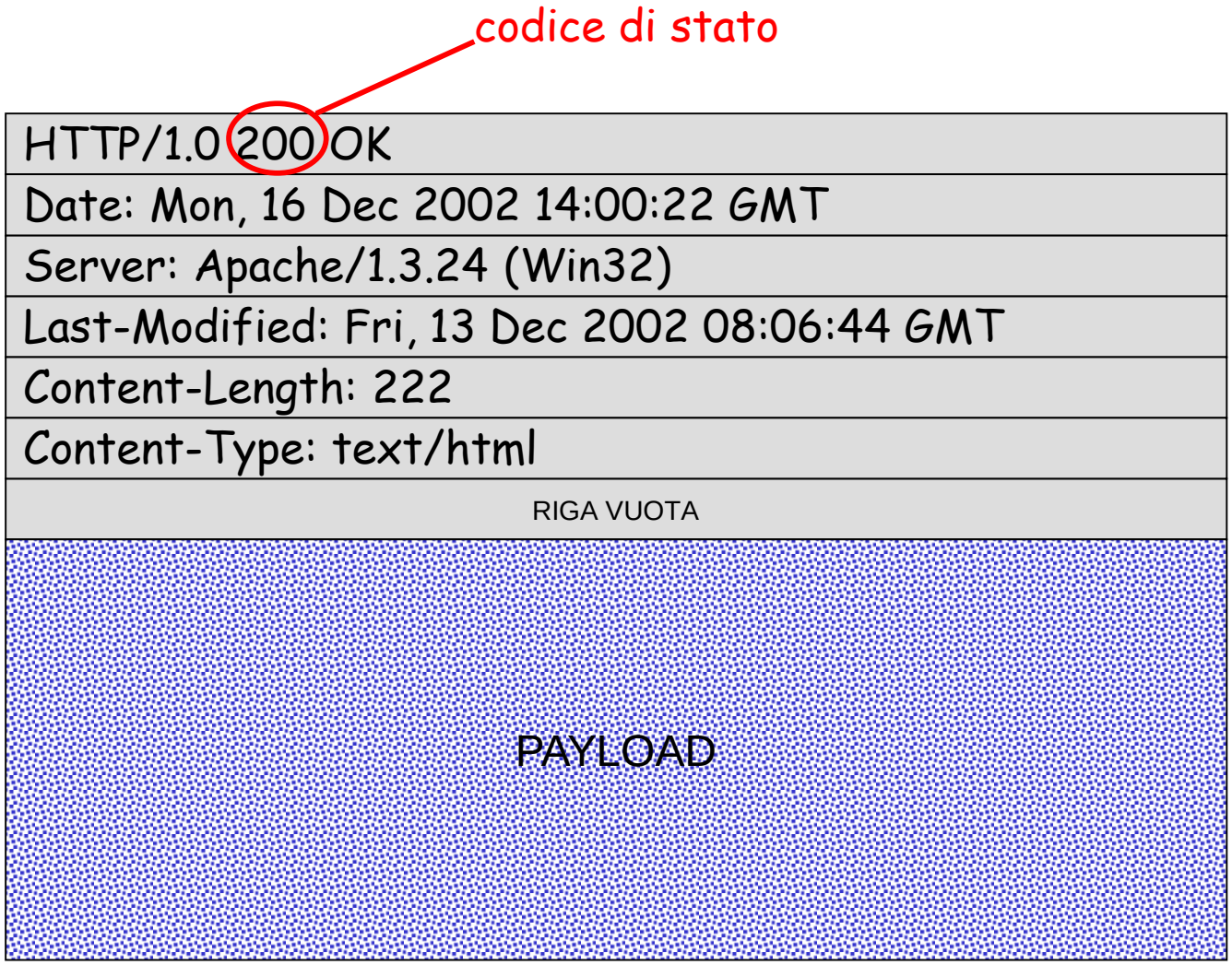
- Il metodo PUT serve per trasmettere delle informazioni dal client al server, creando o sostituendo la risorsa specificata dall'URL
  - In caso di risposta positiva ad una richiesta PUT, i dati inviati nel body del messaggio PUT dovrebbero costituire il valore della risorsa che il server restituisce 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

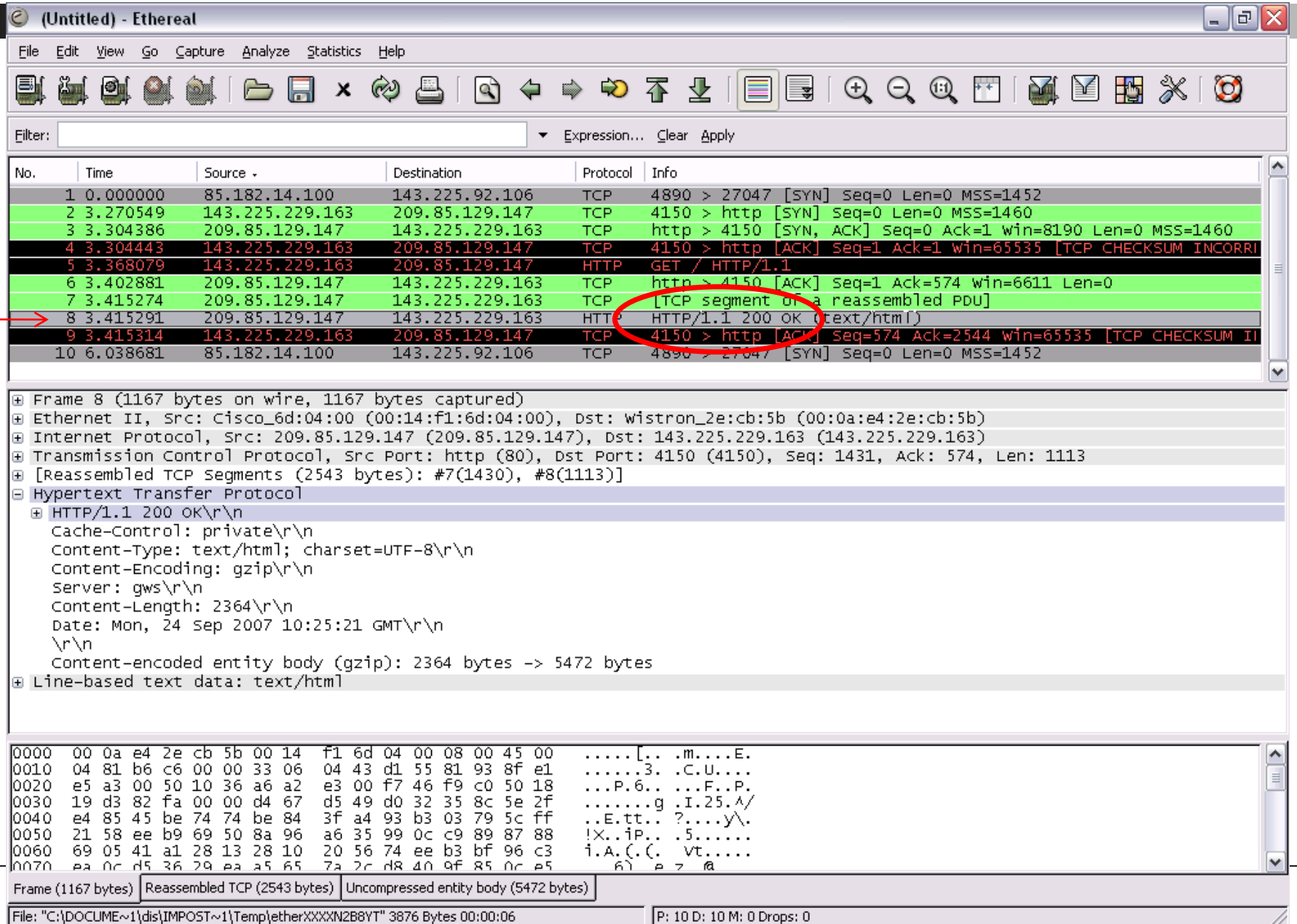


CR = Carriage Return = $0D_{16} = 13_{10}$
LF = Line Feed = $0A_{16} = 10_{10}$

Un esempio di messaggio di risposta



Un esempio di messaggio di risposta



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

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

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 /rcanonic/ 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 /rcanonic/didattica/rc/images/new.gif HTTP/1.0
```

Response

```
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
```

```
GIF89a ▶í ┐┐┐!¨♦☺☻, ▶☻aöÅÇ+╩
                    Ô☺┴╔YôK!S■Y]ÍÉ╥.àcòåÞË«_═<nÉ¬FQÁ¥P‖ÒÁåÁôîédíX╝ó ªµj‖t(
W¼rëàÐãdn§8$Ü
~é_;          HQê².üÍ↕î¡^▒Ó▼
```

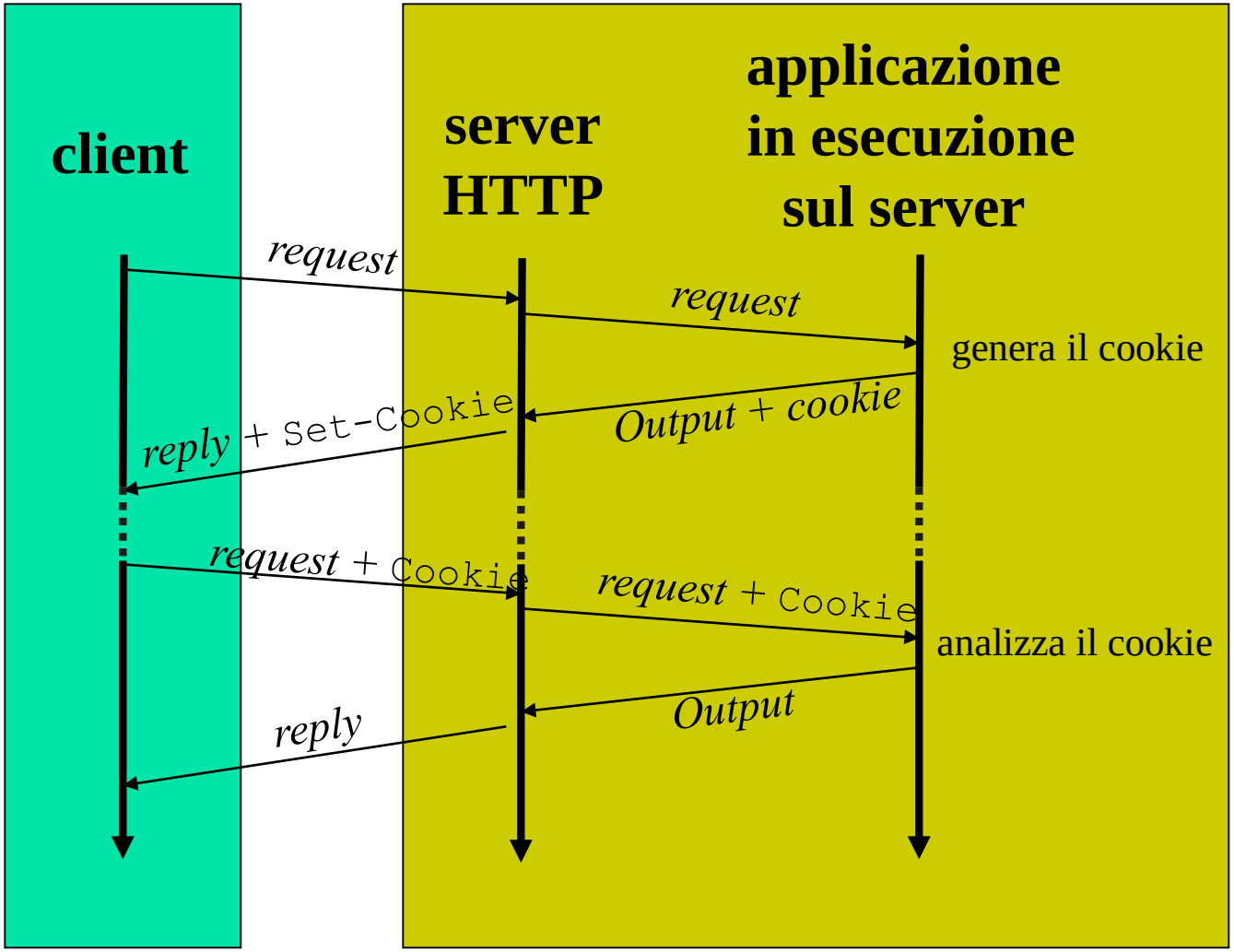
```
Connessione all'host perduta.
```

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)


client
server HTTP
applicazione in esecuzione sul server
request
request
genera il cookie
reply + Set-Cookie
Output + cookie
request + Cookie
request + Cookie
analizza il cookie
reply
Output

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

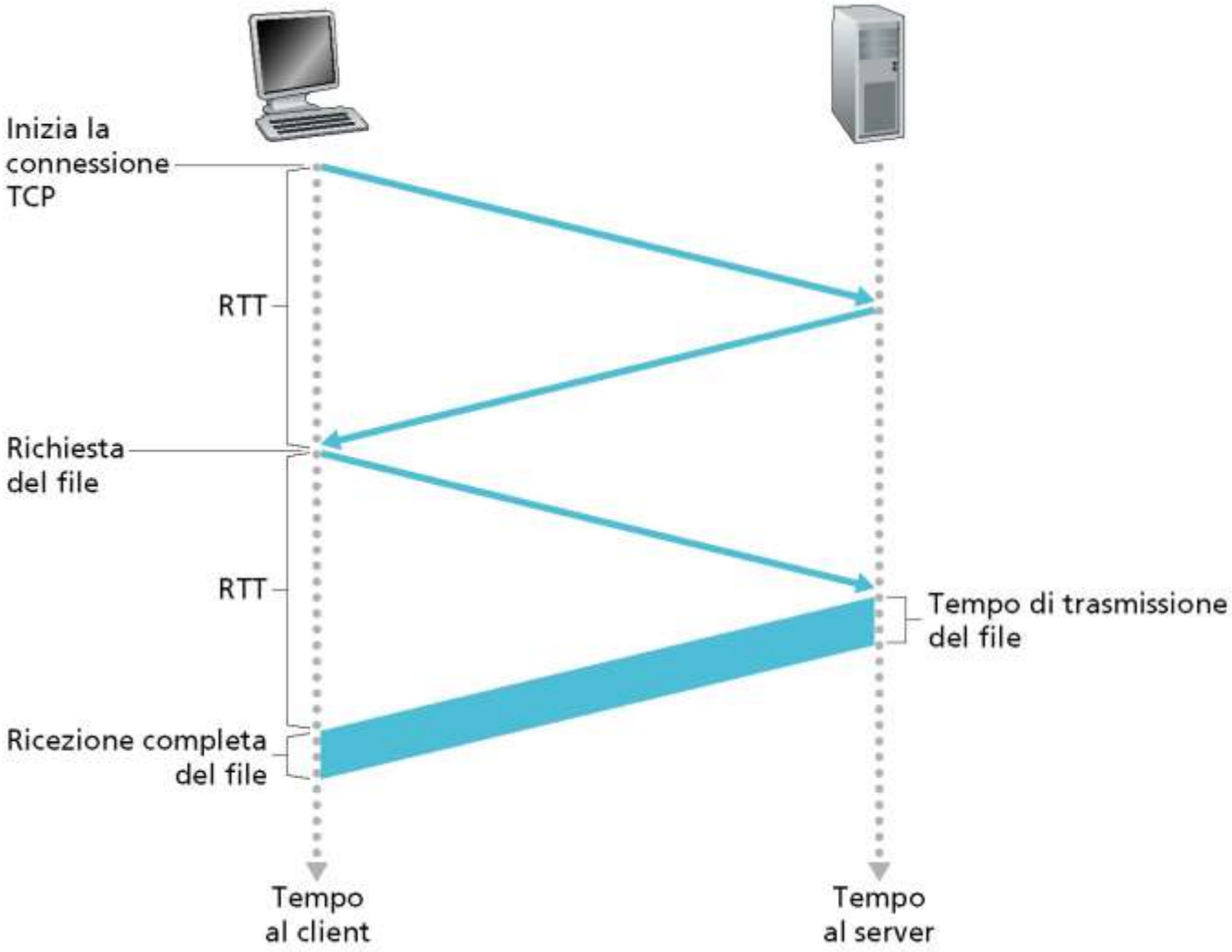
Cookies: attributi di durata

- 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

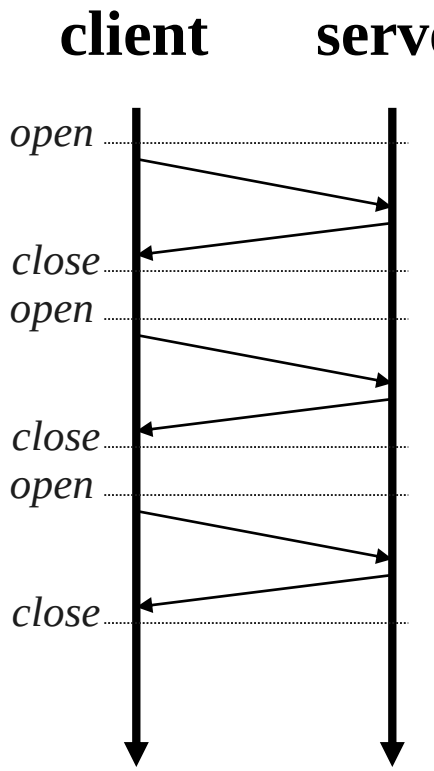
non persistente

- 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

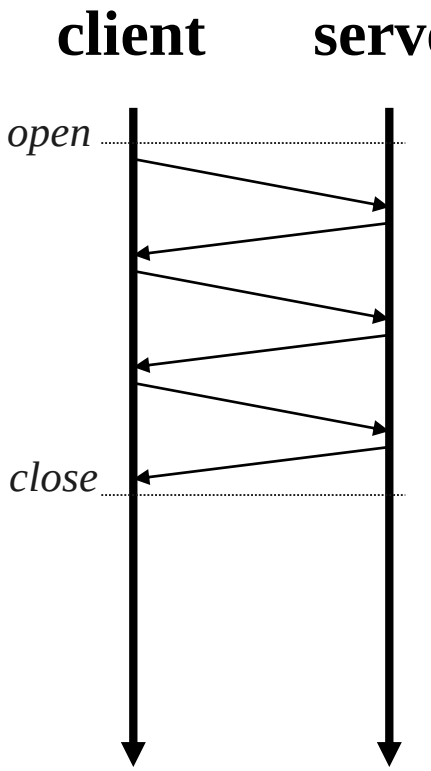
- HTTP/1.1 (RFC 2068, RFC 2116)
- Il server mantiene attiva una connessione TCP fino allo scadere di un timeout
- Il client può usare una connessione TCP precedentemente creata con il server per inviare richieste HTTP consecutive
- Si hanno meno RTT
- Esiste anche una versione con parallelismo (with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

La connessione HTTP



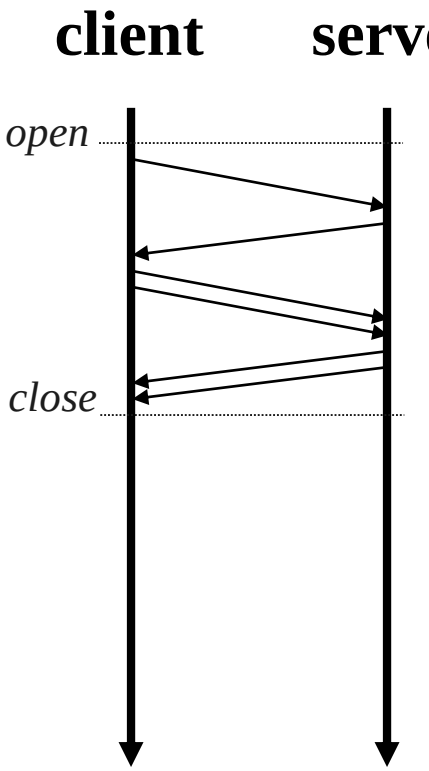
HTTP 1.0

Non persistente



HTTP 1.1

Persistente

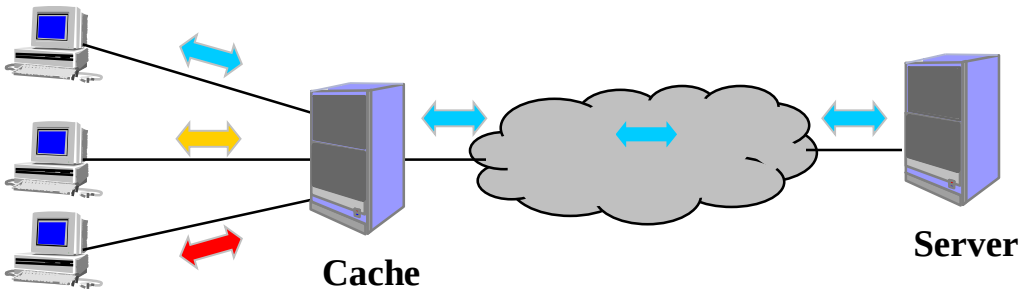


HTTP 1.1
con pipelining

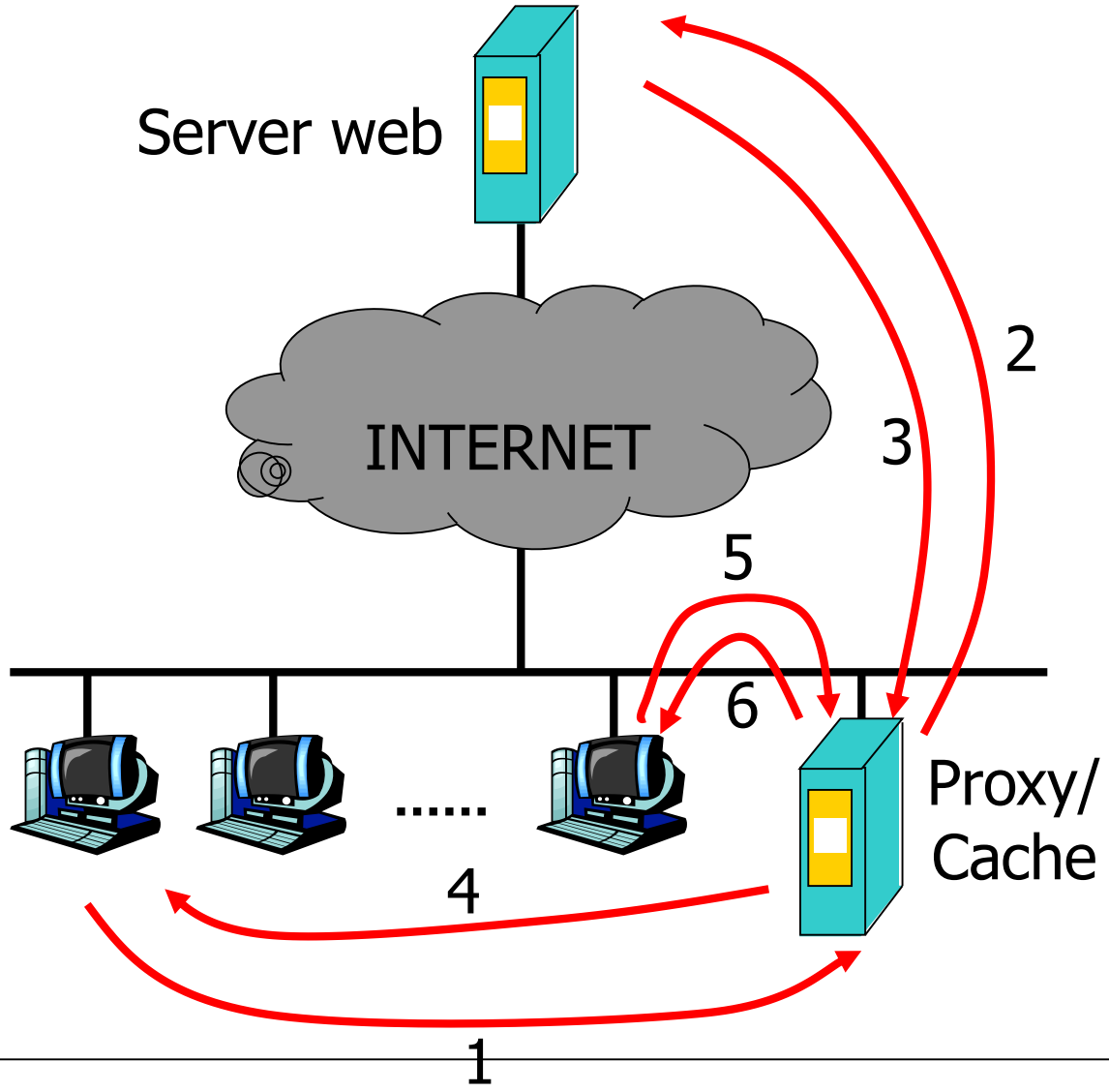
Persistente con pipelining

Web caching

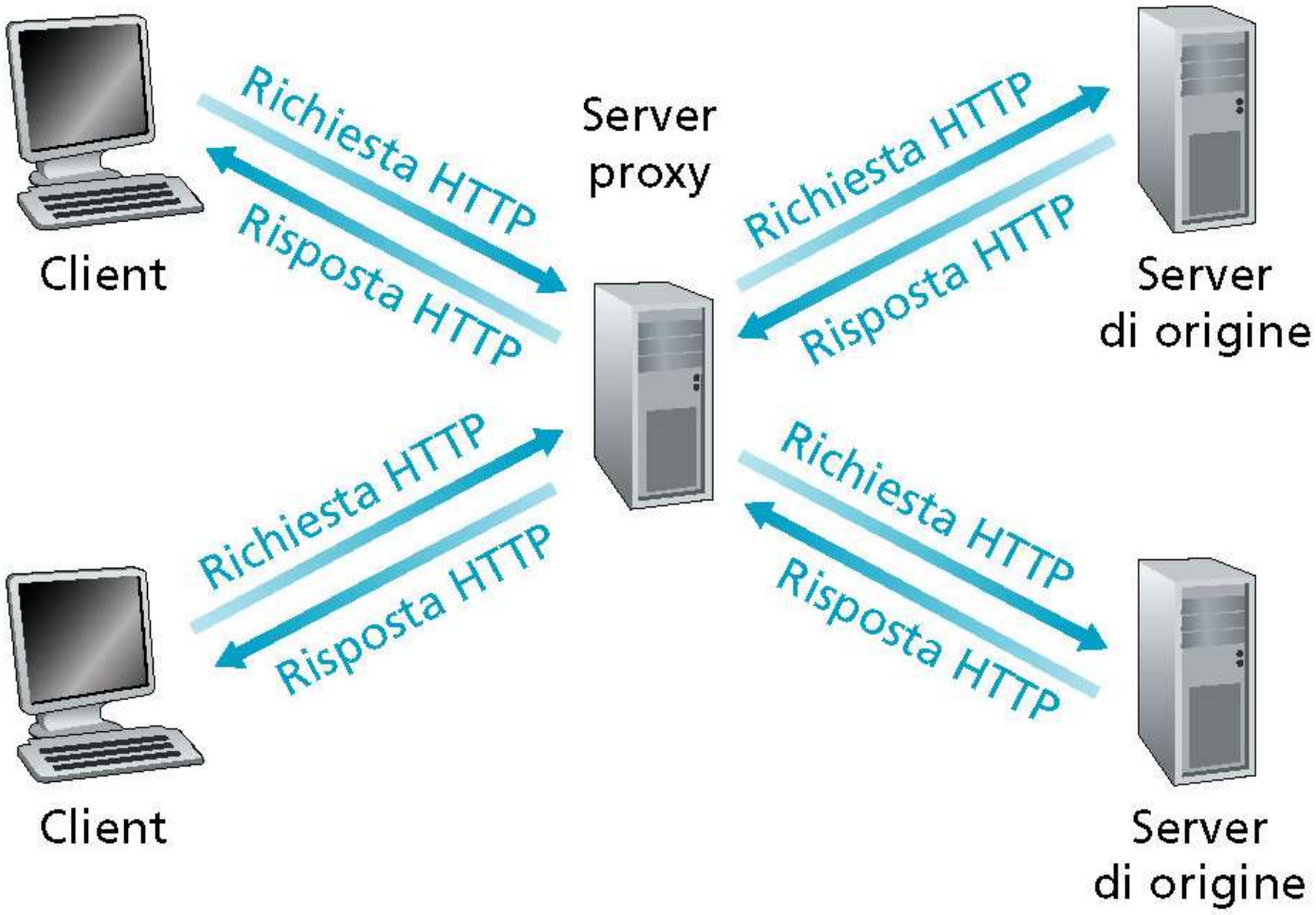
- Si parla genericamente di Web caching quando le richieste di un determinato client non raggiungono il Web Server, ma vengono intercettate da una cache
- Tipicamente, un certo numero di client di una stessa rete condivide una stessa cache web, posta nelle loro prossimità (es. nella stessa LAN)
- Se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

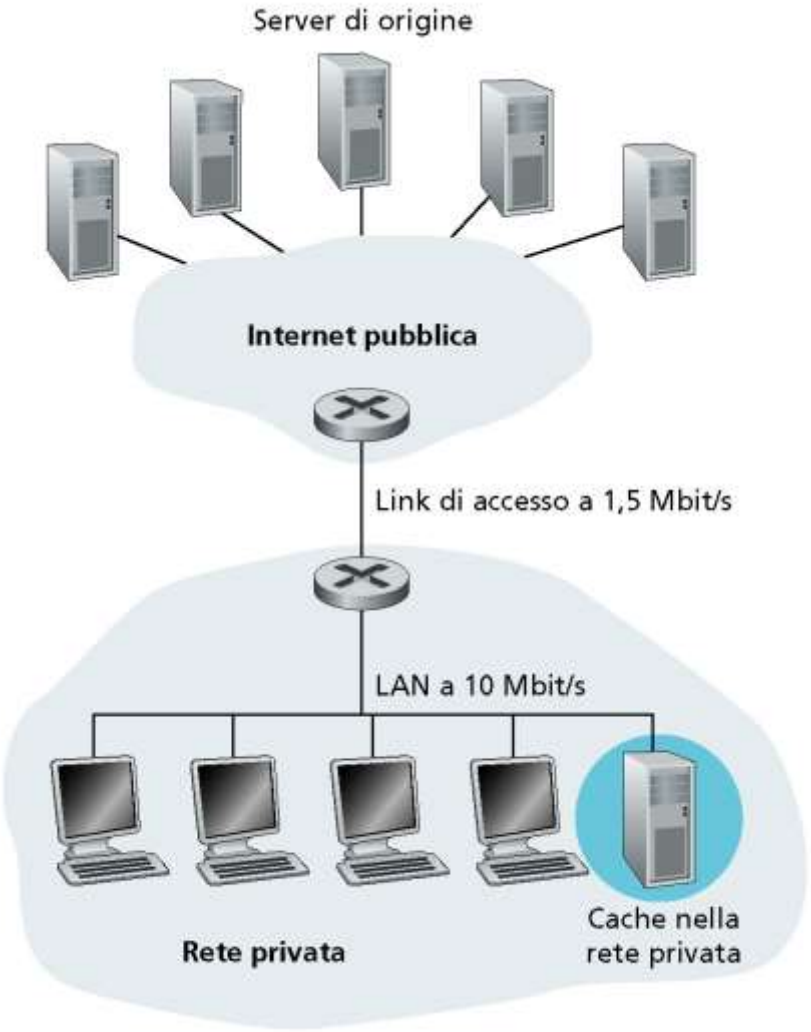


Server proxy: schema logico



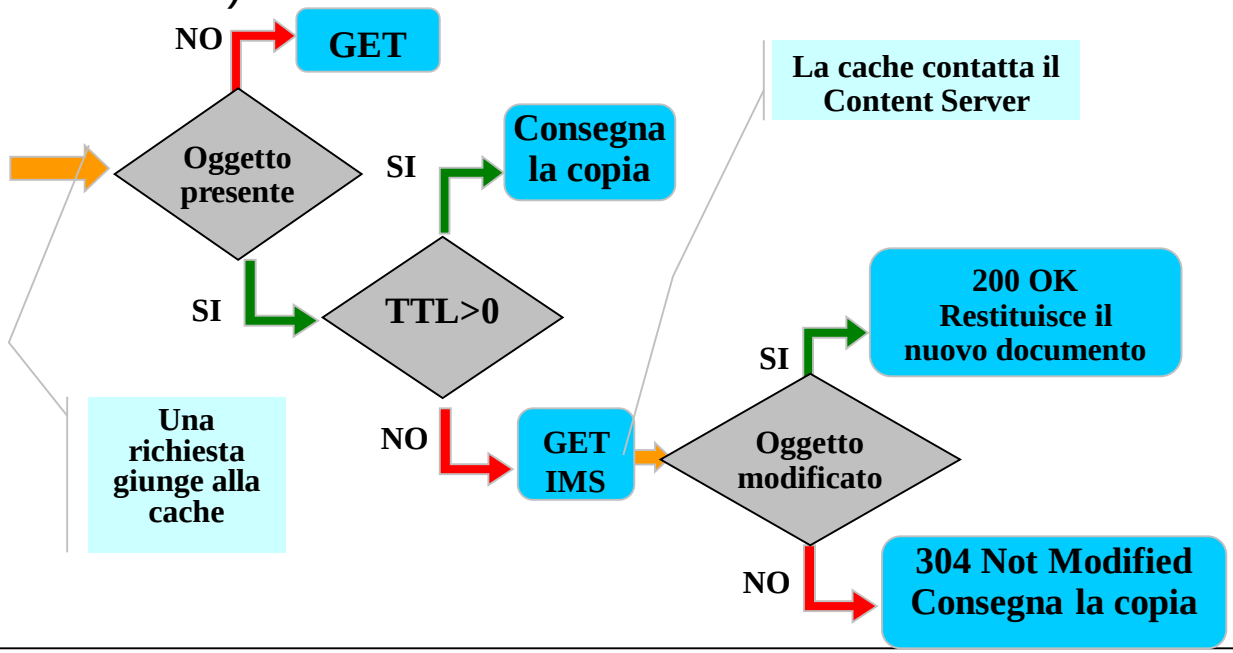
client-cache cache-server

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