

The HTTP protocol (HyperText Transfer Protocol)

Antonio Lioy <lioy@polito.it >

english version created by
Marco D. Aime <m.aime@polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

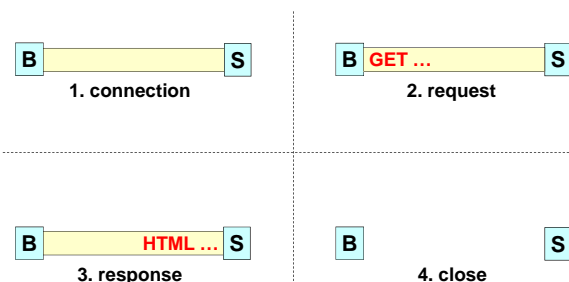
Short history of HTTP

- client-server protocol
- designed for requesting and transmitting HTML pages (and then web resources in general)
- original HTTP (also known as HTTP/0.9)
 - almost only for experimental use
- HTTP/1.0
 - first to achieve large diffusion
 - still largely used
- HTTP/1.1
 - to improve web efficiency
 - to improve cache management

The HTTP 1.0 protocol

- HyperText Transfer Protocol
- RFC-1945 (HTTP/1.0)
- default service: TCP/80
- stateless protocol
- the client can close the connection before receiving the response or parts of it
- channel closed by the server
- 8-bit data (i.e. "8-bit clean")
- default alphabet: ISO-8859-1 (= Latin-1)

Stateless connection (HTTP/1.0)



Links

- URL (Uniform Resource Locator)

schema : // user : password @ host : port / path # anchor

- regular schemes:
 - http, telnet, ftp, gopher, file
- irregular schemes:
 - news:newsgroup
 - mailto:postal-address
- base definition in RFC-1738, plus 1959+2255 (LDAP), 2017 (external-body), 2192 (IMAP), 2224 (NFS)

Link evolution

- URLs are physical addresses (using CNAME it is possible to do something, but limited, at logical level)
- URN (Uniform Resource Name)
 - it's the future evolution, to use logical names, replication, ...
- URI (Uniform Resource Identifier)
 - RFC-3986
 - URI = URL + URN

HTTP/1.0 protocol

- ASCII commands with lines terminated by CR+LF
- data can be binary (since the protocol is “8-bit clean”)
- messages include header + body
- headers are lines beginning with “keyword: ”
- headers are divided from the body by an empty line (i.e. containing only CR+LF)

HTTP/1.0 methods

- **GET uri http-version**
requests the resource associated with the specified URI
- **HEAD uri http-version**
returns only the headers of the response, not data
- **POST uri http-version**
sends data to the server (inside the body) to be elaborated by the specified URI
- **responses** must begin with http-version status-code [text_comment]
- **note that the URI is the resource's**
 - path (when directly connected to the origin server)
 - full URI (when connected to a proxy)

GET method

```

----- ( TCP setup ) -----
request
GET / HTTP/1.0 →
response
HTTP/1.0 200 OK
Date: Wed, 20 May 1998 09:58:21 GMT
Server: Apache/1.0.0
Content-Type: text/html
Content-Length: 1534
Last-modified: Fri, 5 May 1998 12:14:23 GMT
<html>
. . .
</html>
----- ( TCP teardown ) -----

```

HEAD method

```

----- ( TCP setup ) -----
request
HEAD / HTTP/1.0 →
response
HTTP/1.0 200 OK
Date: Wed, 20 May 1998 09:58:21 GMT
Server: Apache/1.0.0
Content-Type: text/html
Content-Length: 1534
Last-modified: Fri, 5 May 1998 12:14:23 GMT
----- ( TCP teardown ) -----

```

POST method

```

----- ( TCP setup ) -----
request
POST /cgi-bin/form.cgi HTTP/1.0
Content-type: text/plain
Content-length: 12
Antonio Lioy
response
HTTP/1.0 200 OK
Date: Mon, 8 Mar 1999 21:30:24 GMT
Server: Apache/1.2.6
Content-type: text/plain
data received from stdin: Antonio Lioy
----- ( TCP teardown ) -----

```

HTTP manual test (Windows)

- open a command prompt (aka “DOS window”)
- run the following commands:

```

C:\> telnet
Microsoft Telnet> set localecho
Microsoft Telnet> set crlf
Microsoft Telnet> set logging
Microsoft Telnet> set logfile mylog.txt
Microsoft Telnet> open server_web 80
GET / HTTP/1.0 <enter>
<enter>

```
- suggestion: if you don't want to activate logging, activate the window scrolling to be able to see the whole server response (Properties – Layout – Screen buffer size – Height)

HTTP manual test (*nix)

- install the “netcat” program (if not already available)
- open a shell
- run the following command:

```
$ nc server_web 80 > output.txt
GET / HTTP/1.1 <enter>
Host: nome_server_web <enter>
Connection: close <enter>
<enter>
```

HTTP status code

- every response includes a status code or 3 digits **XYZ**
- the first digit (X) provides the major status of the requested action
 - X=1 : informational
 - X=2 : success
 - X=3 : redirection
 - X=4 : client error
 - X=5 : server error
- the second and third digits refine the status indication

Standard status codes for HTTP/1.0

200 OK
201 Created
202 Accepted
204 No Content

301 Moved permanently
302 Moved temporarily
304 Not modified

400 Bad request
401 Unauthorized
403 Forbidden
404 Not found

500 Internal server error
501 Not implemented
502 Bad gateway
503 Service unavailable

Redirection

- if the requested object is not available at the specified URI, the server can indicate the new URI by using a response with 3xx code and the header **Location: new-URI**
- the browser can:
 - connect automatically the client to the new URI, if the requested method was GET or HEAD ...
 - ... but this is prevented if the method was POST
- to support older browsers (= which does not understand Redirect) or lazy ones, it is recommended to send, in the response body, an HTML page providing the new URL in human readable format

Header HTTP/1.0 (I)

- **general header:**
 - Date: *http-date*
 - Pragma: *no-cache*
- **request header:**
 - Authorization: *credentials*
 - From: *user-agent-mailbox*
 - If-Modified-Since: *http-date*
 - 304 if the entity has not changed since the date
 - Referer: *URI*
 - request following a Redirect
 - User-Agent: *product*

Header HTTP/1.0 (II)

- **response header:**
 - Location: *absolute-URI*
 - Server: *product*
 - WWW-Authenticate: *challenge*
- **entity body header:**
 - Allow: *method*
 - Content-Encoding: *x-gzip | x-compress*
 - Content-Length: *length*
 - Content-Type: *MIME-media-type*
 - Expires: *http-date*
 - Last-Modified: *http-date*

HTTP date

- **three possible formats:**
 - RFC-822 = Sun, 06 Nov 1994 08:49:37 GMT
 - RFC-850 = Sunday, 06-Nov-94 08:49:37 GMT
 - asctime = Sun Nov 6 08:49:37 1994
- **accept everyone, generate RFC-822 only**
- **always GMT, never time zone indication**

HTTP/0.9 protocol

- **simpler**
- **not described in any RFC**
- **request: GET *URI***
- **response: *entity-body***

Problems of HTTP/1.0

- **designed for static objects**
 - size known a-priori
 - otherwise data truncation is possible
- **impossible to resume an interrupted connection**
- **designed to request and receive single pages:**
 - every object requires a separate connection
- **one TCP connection for every request:**
 - TCP setup requires time (3-way handshake)
 - TCP slow start requires time
 - when the TCP channel is closed, information about network congestion is lost
- **basic cache handling**

HTTP/1.1 protocol (I)

- **RFC-2616**
- **persistent connections (default) and pipelining**
 - improved speed (multiple transactions on the same channel)
- **improved body transmission**
 - negotiation of format and language for body data
 - fragmentation (chunked encoding) for dynamic pages
 - partial transmission
- **cache management**
 - mechanisms can be specified through the protocol
 - hierarchical proxies

HTTP/1.1 protocol (II)

- **virtual server support**
 - multiple logical servers associated with the same IP address
- **new methods:**
 - PUT, DELETE, TRACE, OPTIONS, CONNECT
- **new authentication mechanism (based on digests) at transport level**

HTTP/1.1: virtual host

- **with HTTP/1.0 an IP address was needed for every web server hosted at a node (multi-homed)**
- **with HTTP/1.1, this is not required anymore:**
 - multiple logical servers associated with the same IP address
 - retrieved through alias in the DNS (CNAME record)
- **the client must specify the desired virtual host by its FQDN (Fully Qualified Domain Name)**
- **new header: "Host: FQDN [: port]"**

HTTP/1.1: virtual host (example)

```

host.provider.it IN A 10.1.1.1
www.musica.it IN CNAME host.provider.it
www.libri.it IN CNAME host.provider.it
  
```

DNS

```

(connection to host.provider.it, i.e. IP 10.1.1.1)
GET /index.html HTTP/1.1
Host: www.musica.it
  
```

HTTP

```

(connection to host.provider.it, i.e. IP 10.1.1.1)
GET /index.html HTTP/1.1
Host: www.libri.it
  
```

HTTP

HTTP/1.1: persistent connections

- use of a single TCP channel for multiple request-response interactions
- this is the default behaviour in HTTP/1.1
- who behaves differently should declare it with:
Connection: close
- this header is not end-to-end but hop-by-hop
 - important for proxy and gw

Persistent connections: example

```

----- ( TCP setup ) -----
GET / HTTP/1.1
Host: www.polito.it

HTTP/1.1 200 OK
. . .

GET /favicon.gif HTTP/1.1
Host: www.polito.it
Connection: close

HTTP/1.1 200 OK
Connection: close
. . .

----- ( TCP teardown ) -----
  
```

Persistent connections: pros and cons

- **pros:**
 - lower overhead due to channel opening (3-way handshake) and closing (4-way handshake and its timeouts)
 - better handling of network congestions (the TCP window is maintained)
 - the client can perform request *pipelining* (the server must provide responses in the same order)
 - CPU saving (on every node in the chain)
 - “sweet” evolution to newer HTTP versions (the client can try a new version, but then use the older one)
- **cons:**
 - server overload (possible denial-of-service)

HTTP/1.1: pipeline

- possible to send multiple requests without waiting for the corresponding server responses
- optimization of the TCP transmission
- very useful for requesting several elements (of the same resource) at the same time
- **attention:**
 - they are sequential requests, not parallel ones (responses will be received in the same order of the requests)
 - in case of errors, you may need to repeat the entire command

HTTP/1.1: compression

- **savings:**
 - of transmitted bytes
 - of time (as perceived by the final user)
- **costs:**
 - CPU on the client
 - CPU on the server (with “live” compression)
 - but the scarce resource today is bandwidth, not CPU ...
- in HTTP/1.1 implemented as either data encoding or transmission encoding

Data encoding

- encoding applied to a resource before transmitting it (property of the data, not of the transmission)
- since HTTP is 8-bit clean, encoding = compression
- useful to preserve the resource's MIME-type
- headers: Content-Encoding and Accept-Encoding
- possible encodings:
 - gzip = GNU zip (RFC-1952) = LZ-77 + CRC-32
 - compress = Unix program (adaptive LZW)
 - deflate = zlib (RFC-1950) + deflate (RFC-1951)
 - identity = no encoding
 - default when Content-Encoding is missing
 - accept pre-standard names x-gzip & x-compress

Transmission encoding

- specifies the transfer mode for a resource (property of the transmission, not of the data)
- headers: TE and Transfer-Encoding
- similar to the Content-Transfer-Encoding MIME header, but (since the channel here is 8-bit clean) the only real problem is determining the length of the message
- values (multiple are possible):
 - identity (default)
 - gzip, compress, deflate
 - chunked (if present, must be the last one)

"chunked" encoding

- useful when the server does not know a-priori the size of the data to be transferred
- typical case with dynamic servers (e.g. ASP, PHP, JSP)
- the response is fragmented in parts, called "chunks"
- less important in HTTP/1.0 (implicit end of data at channel closing), but protocol less efficient

"chunked" encoding: syntax

- syntax (of the body):
 - *chunk
 - last-chunk
 - *(entity-header CR LF)
 - CR LF
- every chunk:
 - chunk-size [chunk-extensions] CR LF
 - chunk-data CR LF
- size in hex, the last chunk has size=0
- note: "*" means zero or more repetitions of the following object

"chunked" encoding: example

```

...
Transfer-Encoding: chunked CRLF
CRLF
6 CR LF }
<html> } -----> chunk n.1
CRLF }
18 CRLF }
<body>CIAO</body></html> } -----> chunk n.2
CRLF }
0 CRLF }
CRLF } -----> chunk n.3 (last)
CRLF }

(resulting body)
<html><body>CIAO</body></html>

```

CSS / PNG / HTTP-1.1 and performance

- CSS simplifies HTML (and thus reduces its size)
- PNG has better compression than GIF
 - e.g. 12% saving over the 40 images of the test
- complex impact of HTTP-1.1: persistent connections, pipelining, compression

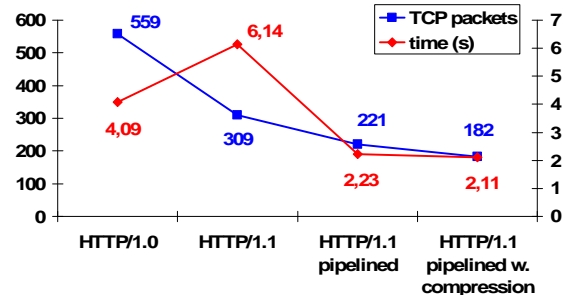
Test HTTP 1.0 vs. 1.1

- test by Nielsen, Gettys *et al* (W3C)
- test page:
 - HTML size = 40 KB
 - contains 43 images in-line (total 130 KB)
- test including 44 GETs (one per each object) in various modes:
 - HTTP/1.0 with 4 simultaneous connections
 - HTTP/1.1 with 1 persistent connection
 - HTTP/1.1 pipeline with 1 persistent connection
 - HTTP/1.1 pipeline + compression with 1 persistent connection

www.w3c.rl.ac.uk/pastevents/RALSymposium98/Talks/br0.html

Results of the test HTTP 1.0 vs. 1.1

WAN first-time retrieval against Apache server



HTTP/1.1: PUT method

- **PUT uri http-version**
 - the body of the request is a new resource to be stored by the server at the specified URI
 - the body of the request is a new version of a resource to replace the one specified by the URI
- **responses: 201 (Created), 200 (OK) or 204 (No content)**
- **example:**

```
PUT /avviso.txt HTTP/1.1
Host: lioy.polito.it
Content-Type: text/plain
Content-Length: 40

On 31/5/2007 there will be no lesson.
```

The "100 Continue" code

- inefficient to transmit a large request body if the server will deny the request before processing it:
 - e.g. unauthorised user or unsupported method
 - typically relative to PUT and POST
- the client asks an explicit confirmation before transmitting the request body

```
PUT /voti.pdf HTTP/1.1
Host: www.abc.com
Expect: 100-continue

... voti.pdf ...

HTTP/1.1 100 Continue
HTTP/1.1 200 OK
```

HTTP/1.1: DELETE method

- **DELETE uri http-version**
 - requests removal of the resource at the specified URI
 - no guarantee of actual removal, even with OK
- **responses:**
 - 200 (OK) if executed + body with details
 - 202 (Accepted) if removal requires manual decision
 - 204 (No content) if executed, but without details
- **example:**

```
DELETE voti.html HTTP/1.1
Host: lioy.polito.it
Content-Length: 0
```

HTTP/1.1: TRACE method

- **TRACE uri http-version**
 - asks to receive back a copy of the original request
 - the request is encapsulated in the response body with type MIME "message/http"
- the "Via" request headers (added by proxies and gateways) allow tracing the connection at application level
- the "Max-Forwards" request header allows limiting the number of traversed proxies (useful in case of loops)

TRACE method: example

TRACE / HTTP/1.1
Host: www.polito.it

S

HTTP/1.1 200 OK
Date: Sun, 20 May 2007 21:29:05 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: message/http

C

29
TRACE / HTTP/1.1
Host: www.polito.it
0

HTTP/1.1: OPTIONS method■ **OPTIONS uri http-version**

- asks for the options supported by the server
- only general options if URI=*, otherwise the general ones + the ones specific to the URI (e.g. languages in which the resource is available)

- the “Max-Forwards: N” request header allows obtaining the options supported by the N-1 traversed proxy rather than by the origin server

OPTIONS method: example

OPTIONS / HTTP/1.1
Host: www.abc.com

S

HTTP/1.1 200 OK
Date: Sun, 20 May 2007 21:51:25 GMT
Server: Apache/1.3.31 (Unix)
Content-Length: 0
Allow: GET, HEAD, OPTIONS, TRACE

C

HTTP/1.1: CONNECT method

- reserved for use with proxies able to handle secure tunnels (e.g. SSL)

Partial transmission

- useful when resuming interrupted transfers
- server headers:
 - Accept-Range: none | bytes
 - Content-Range: bytes start-stop/total (total=* if the total size is not known)
- client headers:
 - Range: range1, range2, ..., rangeN
- possible ranges:
 - start-stop (interval, including the boundaries)
 - -lastN (the last lastN byte)
 - startByte- (from startByte up to the end)

Entity tag (Etag)

- “opaque” identifier used in place of the modification date to know if an entity has changed
- inserted by servers into the response header with:
Etag: [W/] “hex_string”
- equal tags imply:
 - (strong Etag) identical objects (same bytes)
 - (weak Etag) equivalent objects (same effect)
- values used by clients with the request headers:
 - If-Match (e.g. update an old version with PUT only if not already updated)
 - If-None-Match (e.g. GET of a new version)

HTTP/1.1 request headers (I)

- **Accept: media-range [; q=qualityValue] , ...**
 - accepted data formats
 - e.g. Accept: image/jpeg, image/*;q=0.5
- **Accept-Charset: charset [; q=qualityValue] , ...**
 - accepted character sets
- **Accept-Encoding: cont-enc [; q=qualityValue] , ...**
 - accepted content encodings
- **Accept-Language: language [; q=qualityValue] , ...**
 - accepted languages
 - e.g. Accept-Language: it, en-gb;q=0.8, en;q=0.7

HTTP/1.1 request header (II)

- **Authorization: credentials**
 - credentials as response to an authentication request received from the origin server
- **Expect: expected-behaviour , ...**
 - the server replies with 417 if it does not understand
- **From: rfc822-address**
 - especially important to contact who has activated a robot that is causing problems
 - privacy/spamming problems
- **Host: hostname [: port]**
 - virtual host to be contacted (default port: 80)

HTTP/1.1 request header (III)

- **If-Match: etagValue , ...**
 - apply the method only if the resource corresponds to one of the specified tags
- **If-Modified-Since: http-date**
 - apply the method only if the resource has been modified after the specified date
- **If-None-Match: etagValue , ...**
 - apply the method only if the resource does not correspond to any of the specified tags
- **If-Range: etagValue | http-date**
 - send the whole resource if it has changed, only the specified portion otherwise (used with Range: ...)

HTTP/1.1 request header (IV)

- **If-Unmodified-Since: http-date**
 - apply the method only if the resource has not been modified after the specified date
- **Max-Forwards: max-num-forwards**
 - used with TRACE and OPTIONS to identify the intermediate node that should respond
 - max-num-forwards decremented by every node
- **Proxy-Authorization: credentials**
 - credentials as response to an authentication request received from a proxy

HTTP/1.1 request header (V)

- **Range: interval**
 - for receiving only the specified interval
 - offsetFrom–offsetTo (e.g. bytes=500–999)
 - –lastBytes (e.g. bytes=–500)
- **Referer: absoluteURI | relativeURI**
 - URI of the page that has generated the current request
 - this field is missing for URIs inserted by keyboard
- **TE: [trailers] [transf-enc [; q=qualityValue]] , ...**
 - chunk trailers and acceptable transfer encodings
- **User-Agent: product-name**
 - identifies the client's software implementation

HTTP/1.1 response header (I)

- **Accept-Ranges: bytes | none**
 - specifies if the server accepts partial downloads
- **Age: ageValue**
 - specifies the seconds elapsed since the resource was first inserted into the cache
 - meaningful only for proxies, gateways or servers with cache
- **ETag: entity-tag**
 - opaque unique identifier of the requested resource
- **Location: absolute-URI**
 - signals a redirection to the specified URI

HTTP/1.1 response header (II)

- **Proxy-Authenticate: challenge**
 - challenge sent by a proxy to authenticate the client
- **Retry-After: http-date | seconds**
 - in case of server unavailability
- **Server: product-name**
 - identifies the responder's software implementation
- **Vary: * | request-header , ...**
 - identifies the request fields on which the response depends (and thus the validity of the copy in cache)
- **WWW-Authenticate: challenge**
 - challenge sent by the origin server to authenticate the client

HTTP/1.1 general header (I)

- **Cache-Control: req-cache-dir | res-cache-dir , ...**
 - cache control in the request and the response
 - directives sent by client and server towards proxies
- **Connection: close**
 - for non-persistent connections (with server or proxy)
- **Date: http-date**
 - transmission date for the request or the response
- **Pragma: no-cache**
 - for compatibility with HTTP/1.0
 - in HTTP/1.1, Cache-Control is preferred

Request-cache-directive (I)

- **no-cache**
 - I want a response directly from the origin server
- **no-store**
 - do not cache (totally or partially) both the request and the response
- **max-age=seconds**
 - I want a response not older than X s
- **max-stale [=seconds]**
 - I also accept responses that have expired (for no more than X s)
- **min-fresh=seconds**
 - I want a response that will be still valid after X s

Request-cache-directive (II)

- **no-transform**
 - proxy are prohibited to transform the resource (e.g. some proxies change the format of images to save space)
- **only-if-cached**
 - I do not want to contact the origin server
 - useful for overloaded networks

Response-cache-directive (I)

- **public**
 - the response can be cached (shared/private)
- **private [="header-name, ..."]**
 - do not insert in shared cache (all or some headers)
- **no-cache [="header-name, ..."]**
 - do not insert in cache (all or some headers)
- **no-store**
 - do not insert in cache (shared/private) on disc
- **no-transform**
- **must-revalidate**
 - proxy / client must re-validate with the origin server when the cached resource becomes stale

Response-cache-directive (II)

- **proxy-revalidate**
 - proxy / client must re-validate with the origin server when the cached resource becomes stale
- **max-age=seconds**
 - data expiration time
- **s-maxage=seconds**
 - data expiration time in a shared cache
 - has priority on max-age (only for shared caches)
- **note: max-age and s-maxage have priority on a possible "Expires:" header of the resource**

HTTP/1.1 general header (II)

- **Trailer: header-name , ...**
 - the specified headers are moved to the end of the chunked body
 - not applicable to Transfer-Encoding, Content-Length and Trailer
- **Transfer-Encoding: transfer-encoding**

HTTP/1.1 general header (III)

- **Upgrade: protocol, ...**
 - the client proposes to switch to a "better" protocol (chosen by the server among the listed ones)
- **Via: protocol node [comment] , ...**
 - sequence of intermediate nodes (proxy or gateway)
 - node = "host [: port]" or a pseudonym
- **Warning: code agent text [date]**
 - typically used by proxies to signal specific conditions
 - if the text is not ISO-8859-1 then RFC-2047

Warning codes

- **110 Response is stale**
 - response is "old"
- **111 Revalidation failed**
 - failure in contacting the origin server to revalidate
- **112 Disconnected operation**
 - proxy is disconnected by the network
- **113 Heuristic expiration**
 - proxy has chosen an expiration time heuristically
- **199 Miscellaneous warning**
- **214 Transformation applied**
 - proxy has changed the encoding of transmitted data
- **299 Miscellaneous persistent warning**

HTTP/1.1 entity header (I)

- **Allow: method, ...**
 - used by the server with the status 405 (Method Not Allowed) to specify the allowed methods
 - used by the client with PUT to suggest the methods that should be used to access the resource
- **Content-Encoding: encoding**
- **Content-Language: language**
- **Content-Length: num-of-bytes**
 - number (10 basis) of bytes of the body
- **Content-Location: absoluteURI | relativeURI**
 - useful in case a resource provided after a negotiation is also available directly

HTTP/1.1 entity header (I)

- **Content-MD5: base64-md5-digest**
 - to protect the integrity of the transferred data (from random errors, not from deliberate attacks!)
- **Content-Range: interval**
- **Content-Type: mime-type**
- **Expires: http-date**
- **Last-Modified: http-date**

State management

- **web applications may need to keep information on the users while browsing:**
 - across different pages
 - across different times (minutes, hours, days)
- **various motivations:**
 - user preferences (e.g. language, font size)
 - business data (e.g. shopping kart)
- **problem with HTTP:**
 - HTTP/1.0 makes single transactions and is stateless
 - HTTP/1.1 allows multiple transactions over the same connection, but is stateless among successive connections

Possible state management mechanisms

- **at application level:**
 - via special URLs, dynamically generated
 - e.g. `www.x.com/basket/12ab34/`
 - via "hidden" fields (`type=hidden`) of a form
 - passed in hidden way (POST method)
 - passed in visible way (GET method)
 - e.g. `www.x.com/basket?id=12ab34`
- **at transport protocol level:**
 - cookies in HTTP
 - v1 in HTTP/1.0
 - v2 (+ v1) in HTTP/1.1

Why managing the state in HTTP?

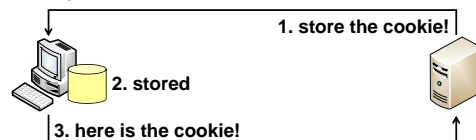
- **to manage the state without modifying the URLs**
 - users can exchange URLs together (without exchanging also their own state)
 - avoid "busting" intermediate caches with
 - same data corresponding to different URLs
 - different pages corresponding to the same URL
- **minimise the impact on server and application configuration**
- **associate the status to users even if anonymous (e.g. not passing across the authentication portal)**
- **save the state in non-volatile memory**
 - preserved across reboots, UA start/stop, IP change

The 4 recommendations for cookies

- maintaining the state of an HTTP session is appropriate only if ...
- 1) the user is conscious and consents to it
- 2) the user can delete the state at any time
- 3) state information is not disclosed to third parties without the user's explicit consent
- 4) state information does not contain sensitive data and cannot be used to obtain sensitive data

Cookies

- **goal: let the server store information locally to the client running the browser**
- **in this way the server is unloaded**
- **... but it introduces elements of:**
 - unpredictability (=UA not supporting/accepting cookies)
 - risk (=cookies read/modified on client or network)



Cookies' specification

- **first defined by Netscape:**
 - "Persistent client state – HTTP cookies"
- **then become IETF standard:**
 - "HTTP state management mechanism":
 - v1 = RFC-2109
 - v2 = RFC-2965
- **Netscape and v1 specifications are obsolete but still used**
- **RFC-2964 "Use of HTTP state management" contains recommendations on cookies' problems / dangers**

Cookies' features

- **small size data stored at the client (persistent information) and sent to the server**
- **cookies set by server on client with the headers:**
 - Set-Cookie: ... (v1)
 - Set-Cookie2: ... (v2)
- **cookies sent by client to server with the header:**
 - Cookie: ...
- **every cookie has the format `name=value`**
 - saved locally by the browser (in a file)
- **data sent to the server at next accesses**
- **used to create a link among different HTTP requests to the same server**

Cookie support at browsers

- **Netscape Navigator**
 - from version 2.0
 - stored in the cookies.txt file (by default in Netscape\Users\username)
- **Microsoft IE**
 - from version 3.0
 - stored in \Windows\Cookies or in the user profile \Documents and Settings\username\Cookies

Implementation limits for cookies

- **limits related to the minimum number of cookies that should be supported by UA implementations**
- **total per client (e.g. on client's disk):**
 - max 300 cookies
 - max 4 kB per cookie
 - total = 1.2 MB max
- **per unique server/domain (e.g. in server's RAM):**
 - max 20 cookie / client
 - max 4 kB per cookie
 - total = 80 KB max / client

Cookie transmission (v1): S > C

- **cookie inserted inside the header of the HTTP response**
- **syntax:**

```
Set-Cookie:
  cookieName=cookieValue
  [ ; EXPIRES=dateValue ]
  [ ; DOMAIN=domainName ]
  [ ; PATH=pathName ]
  [ ; SECURE ]
```

Cookie: value, expiration, secure

- **the cookie name and the value can be any string:**
 - without commas, semicolons, and white spaces
 - these special characters must be encoded in hex (%2C, %3B, %20)
- **EXPIRES: expiration date after which the cookie can be deleted by the client**
 - rfc-822 format: Wdy, DD Mon YY HH:MM:SS GMT
 - if no expiration is set, the cookie expires at browser closure (stored in RAM instead of disk: volatile / temporary cookie)
- **SECURE: cookie transmitted only on HTTPS channels**

Cookie: domain, path

- **DOMAIN specifies the domain authorised to manage the cookie (e.g. politico.it)**
 - only requests to this domain trigger the transmission of the cookie
 - if the domain is not specified, the browser uses the name of the server that sent the cookie
- **PATH specifies the path for which the cookie is intended (e.g. /didattica/)**
 - only requests to URLs in this path trigger the transmission of the cookie
 - if the path is not specified, the browser uses the one of the URL that caused the reception of the cookie

Cookie transmission (V1): C > S

- **before accessing a given URL, the browser checks if it has some cookies associated with the domain and path**
- **if it does, it includes all the name/value pairs of these cookies inside the HTTP request header**
- **if the HTTP request specifies a URL of a dynamic page (e.g. CGI or ASP), the application will retrieve the cookie string from the environment variable HTTP_COOKIE**

```
Cookie: name1=value1; name2=value2; ...
```

Example (step 1)

- cookie used to manage the list of the books selected by the user of a virtual shop
- the client accesses the server and receives:

```
Set-Cookie:
  customer=john_smith;
  expires=Sat, 28-Aug-99 00:00:00 GMT;
  path=/cgi/bin/;
  domain=books.virtualShopping.com;
  secure
```

Example (step 2)

- when accessing a URL within the path /cgi/bin, the client sends to the server a header with:
 - Cookie: customer=john_smith
- the client receives as a response:
 - Set-Cookie: part_number=book1; path=/cgi/bin ; ...
- the client requests a URL within the path /cgi/bin and sends:
 - Cookie: customer=john_smith; part_number=book1

Example (step 3)

- the client receives:
 - Set-Cookie: shipping=fedex; path=/cgi/bin/deliver; ...
- the client requests a URL within the path /cgi/bin/deliver and sends:
 - Cookie: customer=john_smith; part_number=book1; shipping=fedex

Problems with cookies

- the cookie mechanism allows building profiles of the users
- user tracking = term used to indicate the possibility to trace the sites visited by a given user and thus its habits and interests
- example: if a user download an advertisement banner from a site, it receives a cookie in addition to an image
- for every site inside the circuit it is possible to set and retrieve the value of the cookies relative to the user browsing preferences
- cookies can be disabled on the browser

Problems with cookies

- performing authentication based on cookies (e.g. commerce sites associating user to the kart, one-click order):
 - eavesdropping of cookies during transmission or on the client
- attacks that allows intercepting cookies:
 - packet sniffing
 - web spoofing
 - attacks against the client (virus, worm, javascript, ...)
 - sub-domain attack (which part of the domain is responsible of the service using the cookie?)

Cookie transmission (v2): S > C

- new syntax (v2) specified in RFC-2965
- one or more cookies (comma separated list)

```
Set-Cookie2: cookieName=cookieValue
[ ; Comment=commentText ]
[ ; CommentURL="URL" ]
[ ; Discard ]
[ ; Domain=domainName ]
[ ; Max-Age=dateValue ]
[ ; Path=pathName ]
[ ; Port="comma-separated-port-list" ]
[ ; Secure ]
[ ; Version=versionNumber ]
```

New attributes of cookies v2

- **Comment and/or CommentURL**
 - inform the user about the purpose of the cookie
- **Discard**
 - requires the UA to delete the cookie when it terminates (=volatile cookie, stored only in RAM)
- **Max-Age**
 - number of seconds after which the cookie must be deleted
- **Port**
 - TCP ports on which the cookie can be transmitted
- **Version**
 - version number (now equal to 1)

Semantic change

- in v1 the Secure attribute requires transmission only on HTTPS
- in v2 the Secure attribute is an advice by the server:
 - “to be treated securely”
 - when sending the cookie back to the server, the client should use a channel with no less than the same level of security as was used when the cookie was received

Cookie transmission (v2): C > S

- new syntax (v2) specified in RFC-2965
- one or more cookies (comma separated list)

```
Cookie: $Version=cookieVersion ;  
      cookieName=cookieValue  
      [ ; $Domain=domainName ]  
      [ ; $Port="portNumber" ]  
      [ ; $Path=pathName ]
```

Request header “Cookie2”

- when a client transmits at least a cookie with a version number greater than the one understood by the client, it must signal to the server the understood version with:

```
Cookie2: highest-cookie-version-understood
```