

Architecture of Software Systems

HTTP Protocol

Martin Reháč

HTTP Protocol

- Hypertext Transfer Protocol
- Designed to transfer **hypertext** information over the computer networks
- **Hypertext**: Structured text with embedded logical links to other texts or resources.
 - Concept dates back to 1960s, previously included in many software products (help systems in particular, or Gopher)
- Specification maintained by the IETF and W3C
- [RFC 2616](#)
 - Hyperlink included intentionally...

What made it special?

- Several **internal** factors have contributed to HTTP dominance
 - HTTP + HTML + Browser combination
 - Technical simplicity of the text form (Easy debugging&Integration!)
 - Elegance of the URI/URL model
 - Simple technical model
 - Unidirectional connection method
 - Available open source implementations/stacks
 - HTTPS/TLS security model (1994, Netscape)
 - Originally proprietary, opened ([RFC 2818](#))
- **External** factors:
 - Explosive increase in available bandwidth thanks to the massive investments in fiber optics & new encoding techniques
 - Change of security posture on Internet/Enterprise Networks

Below and Around HTTP

Traditional OSI Model

7 - Application	HTTP
6 - Presentation	(JPEG/UTF/ASCII)
5 - Session	(RPC/SQL/NFS)
4 - Transport	TCP
3 - Network	IP
2 - Data Link	Ethernet
1 - Physical	Electrons/Photons

De-Facto HTTP Model

7 - HTTP	Web applications
	Content types
	Cookies/Proxy/Auth
4 - Transport	TCP
3 - Network	IP
2 - Data Link	Ethernet
1 - Physical	Electrons/Photons

Protocol Versions

- WorldWideWeb – 1989 - Tim Berners Lee
- HTTP 0.9 – 1991 – initial version formally described
- HTTP 1.0 – 1996 – added additional methods, headers and meta-fields
- HTTP 1.1 – 1996 (de-facto)/1997 (official). Introduces persistent connections, improves efficiency and latency
- HTTP 1.1. updated in 2014 – 6 RFCs describing the spec
- HTTP 2.0 – 2014 – approved and waiting for publication


































URI/URL

- Departure from the name service/service directory concept
- URL became a standard way how to identify resources, APIs and their methods on internet/intranet
- Reused in other standards as well

<http://example.com:8080/my/resource?p=12>

- URL has five parts:
 1. protocol: http or https,...
 2. hostname or IP address: to locate the host providing the service
 3. (TCP) port number: to identify the process bound to the port
 4. resource identification/filename: to request a specific resource
 5. parameters – Interpreted by scripts/resources or the server to transfer the information (method/service parameters)

(Simple) HTTP Request(s) for wikipedia.org

Name Path	Method	Status Text	Type	Initiator	Size Content	Time Latency	Timeline	100 ms	150 ms	200 ms	250 ms	300 ms	350 ms
 www.wikipedia.org	GET	304 Not Modified	text/html	Other	587 B 41.3 KB	340 ms 339 ms							
 load.php?debug=false&lang=en&modules... bits.wikimedia.org/meta.wikimedia.org	GET	200 OK	text/css	www.wikipedia.org/:12 Parser	(from cache)	0 ms 0 ms							
 Wikipedia_wordmark_1x.png upload.wikimedia.org/wikipedia/meta/6/6d	GET	200 OK	image/png	www.wikipedia.org/:23 Parser	(from cache)	0 ms 0 ms							
 Wiktionary-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/3/3b	GET	200 OK	image/png	www.wikipedia.org/:539 Parser	(from cache)	0 ms 0 ms							
 Wikinews-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/a/aa	GET	200 OK	image/png	www.wikipedia.org/:543 Parser	(from cache)	0 ms 0 ms							
 Wikiquote-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/c/c8	GET	200 OK	image/png	www.wikipedia.org/:547 Parser	(from cache)	0 ms 0 ms							
 Wikibooks-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/7/74	GET	200 OK	image/png	www.wikipedia.org/:551 Parser	(from cache)	0 ms 0 ms							
 Wikidata-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/0/00	GET	200 OK	image/png	www.wikipedia.org/:555 Parser	(from cache)	0 ms 0 ms							
 Wikispecies-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/8/8c	GET	200 OK	image/png	www.wikipedia.org/:559 Parser	(from cache)	0 ms 0 ms							
 Wikisource-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/2/27	GET	200 OK	image/png	www.wikipedia.org/:563 Parser	(from cache)	0 ms 0 ms							
 Wikiversity-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/a/af	GET	200 OK	image/png	www.wikipedia.org/:567 Parser	(from cache)	0 ms 0 ms							
 Wikivoyage-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/7/74	GET	200 OK	image/png	www.wikipedia.org/:571 Parser	(from cache)	0 ms 0 ms							
 Commons-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/9/90	GET	200 OK	image/png	www.wikipedia.org/:575 Parser	(from cache)	0 ms 0 ms							
 MediaWiki-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/1/16	GET	200 OK	image/png	www.wikipedia.org/:579 Parser	(from cache)	0 ms 0 ms							
 Meta-logo_sister_1x.png upload.wikimedia.org/wikipedia/meta/f/f2	GET	200 OK	image/png	www.wikipedia.org/:583 Parser	(from cache)	0 ms 0 ms							
 wikimedia-button.png bits.wikimedia.org/images	GET	200 OK	image/png	www.wikipedia.org/:590 Parser	(from cache)	0 ms 0 ms							
 Wikipedia-logo-v2_1x.png upload.wikimedia.org/wikipedia/meta/0/08	GET	200 OK	image/png	www.wikipedia.org/:1 Parser	(from cache)	0 ms 0 ms							
 Bookshelf-40x201_6.png upload.wikimedia.org/wikipedia/commons...	GET	200 OK	image/png	www.wikipedia.org/:1 Parser	(from cache)	0 ms 0 ms							

HTTP Request Structure

```
GET / HTTP/1.1
Host: microsoft.com

Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.3; Trident/
           7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Proxy-Connection: Keep-Alive
```


Methods

- A Method **must** be specified for any request.
 - Commonly used methods are specified below. Other exist (TRACE), but are not typically used
- GET – retrieve the information (resource) described by URI
- HEAD – retrieve **the header** of the information described by URI, **without the actual content**
- POST – modifies the information on the server
- PUT – uploads/overwrites the resource on the server (API)
- DELETE – removes the resource on the server (API)
- CONNECT – opens a tunnel (proxy/persistent connection)

Method properties

- **Safe** methods: (must) never change the state of the resource accessed. No side-effects in the functional sense.
 - GET, HEAD
- **Idempotent** methods. Repetitive calls should result in identical result and state. Allows refresh/caching.
 - GET, HEAD, PUT, DELETE

HTTP is stateless

- Efficiency achieved through simplicity, and parallelism allowed by statelessness.
- Low level inefficiency (payload encoding, repetitive transfer of information) simplifies the servers and clients enough to achieve high-level efficiencies of scale.

So, how do we handle state when we need it?

Header Fields: Cookies



Header Fields: Cookies

- Provide information to the server about the requesting entity and the context of the request. **And can carry state.**
- The use of cookies transfers the responsibility for state management from server to user (browser).
- Cookies are the most complex header fields – separate RFC 6265.
- Typical cookie size has increased from small (one token) to very large, with actual state stored in the cookie
- Cookie types:
- **Session** cookies: Identify one session and/or carry the information relevant for a single session, e.g. shopping basket.
- **Persistent** cookies: valid across sessions (persistent). Expiration date provided by server.
 - `Set-Cookie: lang=en-US; Expires=Wed, 09 Jun 2021 10:18:14 GMT`

Cookie structure

- Server -> Client

Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly

Set-Cookie: lang=en-US; Path=/; Domain=example.com

- Client - Server

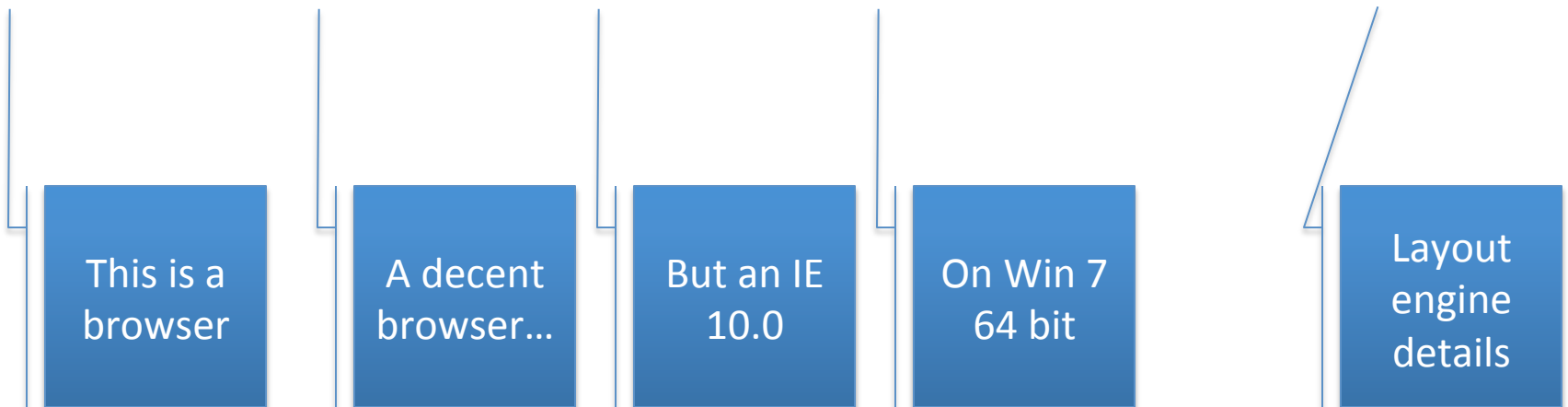
Cookie: SID=31d4d96e407aad42; lang=en-US

- **Scope** (domain/path): Determines which sites are allowed to read/modify the cookie
 - `Path=/; Domain=example.com`
 - Send this back for all subdomains of example.com
- **Secure**: Only transmit this over HTTPS (simplified)
- **HttpOnly**: Do not allow the use from javascript/apps

User Agent

- user-agent: determines the user agent used by the client
 - Most often, this is a browser.
 - Should have been formatted as product/version
 - ... but this has shifted due to server-side logic
 - **Mozilla/5.0** became a near-universal browser UA prefix for compatibility reasons

Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Win64; x64; Trident/6.0)



Host Header – server sharing and load balancing

- **Mandatory** since HTTP 1.1, even if request line contains the full host info
- Allows the (non-dedicated) server to distinguish which domain is the request directed towards.
- Allows co-location, easier load balancing and request routing on the application layer.
- Important for HTTPS, where the requests are often to IP addresses/shared load-balancers
- Example:

```
GET /pub/WWW/ HTTP/1.1  
Host: www.example.org
```


Other Headers

- “Accept” Headers may specify the type of the content/language/encoding/compression the client is willing to accept
 - **Accept:** text/html, application/xhtml+xml, */*
 - **Accept-Language:** en-US
 - **Accept-Encoding:** gzip, deflate
- **Proxy-Connection:** Keep-Alive

HTTP Response

```
HTTP/1.1 200 OK
Server: Apache
X-Powered-By: HHVM/3.3.1
Cache-control: s-maxage=3600, must-revalidate, max-age=0
X-Content-Type-Options: nosniff
Vary: Accept-Encoding
X-Varnish: 12043230 11839648, 2606063287, 3164310615
3161137176
Date: Wed, 08 Apr 2015 08:07:03 GMT
X-Cache: cp1055 hit (12521), cp4018 miss (0), cp4009
frontend hit (32068)
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
Last-Modified: Sat, 04 Apr 2015 21:23:59 GMT
Age: 2184
Via: 1.1 varnish, 1.1 varnish, 1.1 varnish, 1.1 [proxyXYZ]:
80 (Cisco-WSA/8.5.0-497)
Connection: keep-alive
Length: unspecified [text/html]
```

HTTP Response

- Headers do contain a large volume of information/fields
- Key fields are:
 - Status: Success/Error Code
 - Content-type: What is being returned -> Informs the client how to interpret/parse the content.
 - Connection – keep alive options for performance optimizations. Reduction of latency

Status Code

- Status code reflects the outcome of the request. Category breakdown:
 - 2XX Success
 - 200:Success,
 - 3XX Redirect:
 - 301: Permanent
 - 4XX Client Error
 - 5XX Server Error

Informational Status Codes	Client Request Incomplete	Server Errors
<p>100 – Continue [The server is ready to receive the rest of the request.]</p> <p>101 – Switching Protocols [Client specifies that the server should use a certain protocol and the server will give this response when it is ready to switch.]</p>	<p>400 – Bad Request [The server detected a syntax error in the client's request.]</p> <p>401 – Unauthorized [The request requires user authentication. The server sends the WWW-Authenticate header to indicate the authentication type and realm for the requested resource.]</p> <p>402 – Payment Required [reserved for future.]</p> <p>403 – Forbidden [Access to the requested resource is forbidden. The request should not be repeated by the client.]</p> <p>404 – Not Found [The requested document does not exist on the server.]</p> <p>405 – Method Not Allowed [The request method used by the client is unacceptable. The server sends the Allow header stating what methods are acceptable to access the requested resource.]</p> <p>406 – Not Acceptable [The requested resource is not available in a format that the client can accept, based on the accept headers received by the server. If the request was not a HEAD request, the server can send Content-Language, Content-Encoding and Content-Type headers to indicate which formats are available.]</p> <p>407 – Proxy Authentication Required [Unauthorized access request to a proxy server. The client must first authenticate itself with the proxy. The server sends the Proxy-Authenticate header indicating the authentication scheme and realm for the requested resource.]</p> <p>408 – Request Time-Out [The client has failed to complete its request within the request timeout period used by the server. However, the client can re-request.]</p> <p>409 – Conflict [The client request conflicts with another request. The server can add information about the type of conflict along with the status code.]</p> <p>410 – Gone [The requested resource is permanently gone from the server.]</p> <p>411 – Length Required [The client must supply a Content-Length header in its request.]</p> <p>412 – Precondition Failed [When a client sends a request with one or more If... headers, the server uses this code to indicate that one or more of the conditions specified in these headers is FALSE.]</p> <p>413 – Request Entity Too Large [The server refuses to process the request because its message body is too large. The server can close connection to stop the client from continuing the request.]</p> <p>414 – Request-URI Too Long [The server refuses to process the request, because the specified URI is too long.]</p> <p>415 – Unsupported Media Type [The server refuses to process the request, because it does not support the message body's format.]</p> <p>417 – Expectation Failed [The server failed to meet the requirements of the Expect request-header.]</p>	<p>500 – Internal Server Error [A server configuration setting or an external program has caused an error.]</p> <p>501 – Not Implemented [The server does not support the functionality required to fulfill the request.]</p> <p>502 – Bad Gateway [The server encountered an invalid response from an upstream server or proxy.]</p> <p>503 – Service Unavailable [The service is temporarily unavailable. The server can send a Retry-After header to indicate when the service may become available again.]</p> <p>504 – Gateway Time-Out [The gateway or proxy has timed out.]</p> <p>505 – HTTP Version Not Supported [The version of HTTP used by the client is not supported.]</p>
<p>Client Request Successful</p> <p>200 – OK [Success! This is what you want.]</p> <p>201 – Created [Successfully created the URI specified by the client.]</p> <p>202 – Accepted [Accepted for processing but the server has not finished processing it.]</p> <p>203 – Non-Authoritative Information [Information in the response header did not originate from this server. Copied from another server.]</p> <p>204 – No Content [Request is complete without any information being sent back in the response.]</p> <p>205 – Reset Content [Client should reset the current document. I.e. A form with existing values.]</p> <p>206 – Partial Content [Server has fulfilled the partial GET request for the resource. In response to a Range request from the client. Or if someone hits stop.]</p>	<p>Request Redirected</p> <p>300 – Multiple Choices [Requested resource corresponds to a set of documents. Server sends information about each one and a URL to request them from so that the client can choose.]</p> <p>301 – Moved Permanently [Requested resource does not exist on the server. A Location header is sent to the client to redirect it to the new URL. Client continues to use the new URL in future requests.]</p> <p>302 – Moved Temporarily [Requested resource has temporarily moved. A Location header is sent to the client to redirect it to the new URL. Client continues to use the old URL in future requests.]</p> <p>303 – See Other [The requested resource can be found in a different location indicated by the Location header, and the client should use the GET method to retrieve it.]</p> <p>304 – Not Modified [Used to respond to the If-Modified-Since request header. Indicates that the requested document has not been modified since the specified date, and the client should use a cached copy.]</p> <p>305 – Use Proxy [The client should use a proxy, specified by the Location header, to retrieve the URL.]</p> <p>307 – Temporary Redirect [The requested resource has been temporarily redirected to a different location. A Location header is sent to redirect the client to the new URL. The client continues to use the old URL in future requests.]</p>	<p>Unused status codes</p> <p>306- Switch Proxy</p> <p>416- Requested range not satisfiable</p> <p>506- Redirection failed</p>

HTTP protocol version 1.1 Server Response Codes

<http://www.w3.org/Protocols/rfc2616/rfc2616.html>

Chart created September 5, 2000 by Suso Banderas(suso@suso.org). Most of the summary information was gathered from Appendix A of "Apache Server Administrator's Handbook" by Mohammed J. Kabir.

Header Fields (some)

- Server: Name and version of the server
- Content-type: MIME type of the content, according to IANA classification.
- Typically follows `type/subtype;parameters` pattern, e.g.:
 - `text/html; charset=utf-8`
 - `image/jpeg`
 - `application/pdf`
 - `application/rss+xml`
 - `application/soap+xml`
 - `application/javascript` or `text/javascript`

HTTP Proxy

- How to use and handle proxies and load balancers

Conditional request headers – for REST

If-Match Only perform the action if the client supplied entity matches the same entity on the server. This is mainly for methods like PUT to only update a resource if it has not been modified since the user last updated it. If-Match:

"737060cd8c284d8af7ad3082f209582d" Permanent

If-Modified-Since Allows a 304 Not Modified to be returned if content is unchanged If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT

Permanent

If-None-Match Allows a 304 Not Modified to be returned if content is unchanged, see HTTP ETag If-None-Match:

"737060cd8c284d8af7ad3082f209582d" Permanent

If-Range If the entity is unchanged, send me the part(s) that I am missing; otherwise, send me the entire new entity If-Range:

"737060cd8c284d8af7ad3082f209582d" Permanent