Hypertext Transfer Protocol

The **Hypertext Transfer Protocol** (**HTTP**) is an application protocol for distributed, collaborative, hypermedia information systems.[1] HTTP is the foundation of data communication for the World Wide Web.

Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

The standards development of HTTP was coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), culminating in the publication of a series of Requests for Comments (RFCs). The first definition of HTTP/1.1, the version of HTTP in common use, occurred in RFC 2068 in 1997, although this was obsoleted by RFC 2616 in 1999.

1 Technical overview

URL beginning with the HTTP scheme and the WWW domain name label.

HTTP functions as a request-response protocol in the client-server computing model. A web browser, for example, may be the *client* and an application running on a computer hosting a web site may be the *server*. The client submits an HTTP *request* message to the server. The server, which provides *resources* such as HTML files and other content, or performs other functions on behalf of the client, returns a *response* message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

A web browser is an example of a *user agent* (UA). Other types of user agent include the indexing software used by search providers (web crawlers), voice browsers, mobile apps, and other software that accesses, consumes, or displays web content.

HTTP is designed to permit intermediate network elements to improve or enable communications between clients and servers. High-traffic websites often benefit from web cache servers that deliver content on behalf of upstream servers to improve response time. Web browsers cache previously accessed web resources and reuse them when possible to reduce network traffic. HTTP proxy servers at private network boundaries can facilitate communication for clients without a globally routable address, by relaying messages with external servers.

HTTP is an application layer protocol designed within the framework of the Internet Protocol Suite. Its definition presumes an underlying and reliable transport layer protocol,[2] and Transmission Control Protocol (TCP) is commonly used. However HTTP can use unreliable protocols such as the User Datagram Protocol (UDP), for example in Simple Service Discovery Protocol (SSDP).

HTTP resources are identified and located on the network by Uniform Resource Locators (URLs), using the URI schemes *http* and *https*. URIs and hyperlinks in Hypertext Markup Language (HTML) documents form inter-linked hypertext documents.

HTTP/1.1 is a revision of the original HTTP (HTTP/1.0). In HTTP/1.0 a separate connection to the same server is made for every resource request. HTTP/1.1 can reuse a connection multiple times to download images, scripts, stylesheets, *etc* after the page has been delivered. HTTP/1.1 communications therefore experience less latency as the establishment of TCP connections presents considerable overhead.

2 History

The term hypertext was coined by Ted Nelson in 1965 in the Xanadu Project, which was in turn inspired by Vannevar Bush's vision (1930s) of the microfilm-based information retrieval and management "memex" system described in his essay As We May Think (1945). Tim Berners-Lee and his team at CERN are credited with inventing the original HTTP along with HTML and the associated technology for a web server and a text-based

Tim Berners-Lee

web browser. Berners-Lee first proposed the “WorldWideWeb” project in 1989 — now known as the World Wide Web. The first version of the protocol had only one method, namely GET, which would request a page from a server.[3] The response from the server was always an HTML page.[4]

The first documented version of HTTP was **HTTP V0.9** (1991). Dave Raggett led the HTTP Working Group (HTTP WG) in 1995 and wanted to expand the protocol with extended operations, extended negotiation, richer meta-information, tied with a security protocol which became more efficient by adding additional methods and header fields.[5][6] RFC 1945 officially introduced and recognized HTTP V1.0 in 1996.

The HTTP WG planned to publish new standards in December 1995[7] and the support for pre-standard HTTP/1.1 based on the then developing RFC 2068 (called HTTP-NG) was rapidly adopted by the major browser developers in early 1996. By March 1996, pre-standard HTTP/1.1 was supported in Arena,[8] Netscape 2.0,[8] Netscape Navigator Gold 2.01,[8] Mosaic 2.7, Lynx 2.5, and in Internet Explorer 2.0. End-user adoption of the new browsers was rapid. In March 1996, one web hosting company reported that over 40% of browsers in use on the Internet were HTTP 1.1 compliant. That same web hosting company reported that by June 1996, 65% of all browsers accessing their servers were HTTP/1.1 compliant.[9] The HTTP/1.1 standard as defined in RFC 2068 was officially released in January 1997. Improvements and updates to the HTTP/1.1 standard were released under RFC 2616 in June 1999.

In 2007, the **HTTPbis Working Group** was formed, in part, to revise and clarify the HTTP/1.1 specification. In June 2014, the WG released an updated six-part specification obsoleting RFC 2616:

- RFC 7230, *HTTP/1.1: Message Syntax and Routing*
- RFC 7231, *HTTP/1.1: Semantics and Content*
- RFC 7232, *HTTP/1.1: Conditional Requests*
- RFC 7233, *HTTP/1.1: Range Requests*
- RFC 7234, *HTTP/1.1: Caching*
- RFC 7235, *HTTP/1.1: Authentication*

HTTP/2 was published as RFC 7540 in May 2015.

3 HTTP session

An HTTP session is a sequence of network request-response transactions. An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server (typically port 80, occasionally port 8080; see List of TCP and UDP port numbers). An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as “HTTP/1.1 200 OK”, and a message of its own. The body of this message is typically the requested resource, although an error message or other information may also be returned.[1]

4 HTTP Authentication

HTTP provides multiple authentication schemes such as Basic access authentication and Digest access authentication which operate via a challenge-response mechanism whereby the server identifies and issues a challenge before serving the requested content.

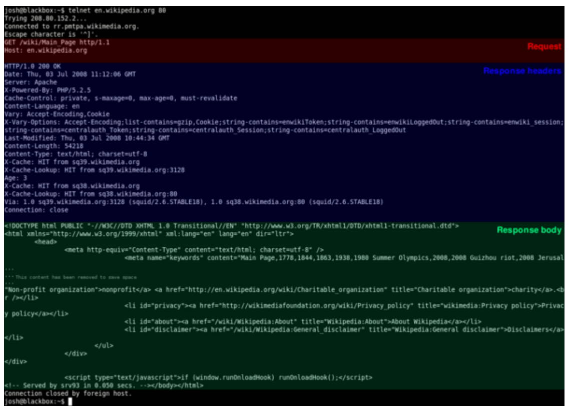
HTTP provides a general framework for access control and authentication, via an extensible set of challenge-response authentication schemes, which can be used by a server to challenge a client request and by a client to provide authentication information.[10]

4.1 Authentication Realms

The HTTP Authentication spec also provides an arbitrary, implementation specific construct for further dividing resources common to a given root URI. The realm value string, if present, is combined with the canonical root URI to form the protection space component of the challenge. This in effect allows the server to define separate authentication scopes under one root URI[10]

5 Request methods

HTTP defines methods (sometimes referred to as *verbs*) to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server. The HTTP/1.0 specification[11] defined the GET, POST and HEAD



```
josh@blackbox:~$ telnet en.wikipedia.org 80
Trying 208.80.152.2...
Connected to rr.pmtpa.wikimedia.org.
Escape character is '^]'.
GET /wiki/Main_Page http/1.1
Host: en.wikipedia.org

HTTP/1.0 200 OK
Date: Thu, 03 Jul 2008 11:12:06 GMT
Server: Apache
X-Powered-By: PHP/5.2.5
Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
Content-Language: en
Vary: Accept-Encoding,Cookie
X-Vary-Options: Accept-Encoding;list-contains=gzip,Cookie;string-contains=enwikiToken;string-contains=enwikiLoggedOut;string-contains=enwiki_session;
string-contains=centralauth_Token;string-contains=centralauth_Session;string-contains=centralauth_LoggedOut
Last-Modified: Thu, 03 Jul 2008 10:44:34 GMT
Content-Length: 54218
Content-Type: text/html; charset=utf-8
X-Cache: HIT from sq39.wikimedia.org
X-Cache-Lookup: HIT from sq39.wikimedia.org:3128
Age: 3
X-Cache: HIT from sq38.wikimedia.org
X-Cache-Lookup: HIT from sq38.wikimedia.org:80
Via: 1.0 sq39.wikimedia.org:3128 (squid/2.6.STABLE18), 1.0 sq38.wikimedia.org:80 (squid/2.6.STABLE18)
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en" dir="ltr">
        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
                                <meta name="keywords" content="Main Page,1778,1844,1863,1938,1980 Summer Olympics,2008,2008 Guizhou riot,2008 Jerusal
...
... This content has been removed to save space
...
"Non-profit organization">nonprofit</a> <a href="http://en.wikipedia.org/wiki/Charitable_organization" title="Charitable organization">charity</a>.<b
r /></li>
                                <li id="privacy"><a href="http://wikimediafoundation.org/wiki/Privacy_policy" title="wikimedia:Privacy policy">Privac
y policy</a></li>
                                <li id="about"><a href="/wiki/Wikipedia:About" title="Wikipedia:About">About Wikipedia</a></li>
                                <li id="disclaimer"><a href="/wiki/Wikipedia:General_disclaimer" title="Wikipedia:General disclaimer">Disclaimers</a>
</li>
                        </ul>
                </div>
</div>

                <script type="text/javascript">if (window.runOnloadHook) runOnloadHook();</script>
<!-- Served by srv93 in 0.050 secs. --></body></html>
Connection closed by foreign host.
josh@blackbox:~$
```

An HTTP 1.1 request made using telnet. The request message, response header section, and response body are highlighted.

methods and the HTTP/1.1 specification[12] added 5 new methods: OPTIONS, PUT, DELETE, TRACE and CONNECT. By being specified in these documents their semantics are well known and can be depended upon. Any client can use any method and the server can be configured to support any combination of methods. If a method is unknown to an intermediate it will be treated as an unsafe and non-idempotent method. There is no limit to the number of methods that can be defined and this allows for future methods to be specified without breaking existing infrastructure. For example, WebDAV defined 7 new methods and RFC 5789 specified the PATCH method.

GET The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect. (This is also true of some other HTTP methods.)[1] The W3C has published guidance principles on this distinction, saying, "Web application design should be informed by the above principles, but also by the relevant limitations."[13] See safe methods below.

HEAD The HEAD method asks for a response identical that of a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.

POST The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database.[14]

PUT The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.[15]

DELETE The DELETE method deletes the specified resource.

TRACE The TRACE method echoes the received request so that a client can see what (if any) changes or additions have been made by intermediate servers.

OPTIONS The OPTIONS method returns the HTTP methods that the server supports for the specified URL. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.

CONNECT [16] The CONNECT method converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.[17][18] See HTTP CONNECT tunneling.

PATCH The PATCH method applies partial modifications to a resource.[19]

All general-purpose HTTP servers are required to implement at least the GET and HEAD methods,[20] and, whenever possible, also the OPTIONS method.

5.1 Safe methods

Some of the methods (for example, HEAD, GET, OPTIONS and TRACE) are, by convention, defined as *safe*, which means they are intended only for information retrieval and should not change the state of the server. In other words, they should not have side effects, beyond relatively harmless effects such as logging, caching, the serving of banner advertisements or incrementing a web counter. Making arbitrary GET requests without regard to the context of the application's state should therefore be considered safe. However, this is not mandated by the standard, and it is explicitly acknowledged that it cannot be guaranteed.

By contrast, methods such as POST, PUT, DELETE and PATCH are intended for actions that may cause side effects either on the server, or external side effects such as financial transactions or transmission of email. Such methods are therefore not usually used by conforming web robots or web crawlers; some that do not conform tend to make requests without regard to context or consequences.

Despite the prescribed safety of *GET* requests, in practice their handling by the server is not technically limited in any way. Therefore, careless or deliberate programming can cause non-trivial changes on the server. This is discouraged, because it can cause problems for web caching, search engines and other automated agents, which can make unintended changes on the server.

5.2 Idempotent methods and web applications

Methods PUT and DELETE are defined to be idempotent, meaning that multiple identical requests should have the same effect as a single request (note that idempotence refers to the state of the system after the request has completed, so while the action the server takes (e.g. deleting a record) or the response code it returns may be different on subsequent requests, the system state will be the same every time). Methods GET, HEAD, OPTIONS and TRACE, being prescribed as safe, should also be idempotent, as HTTP is a stateless protocol.[1] In contrast, the POST method is not necessarily idempotent, and therefore sending an identical POST request multiple times may further affect state or cause further side effects (such as financial transactions). In some cases this may be desirable, but in other cases this could be due to an accident, such as when a user does not realize that their action will result in sending another request, or they did not receive adequate feedback that their first request was successful. While web browsers may show alert dialog boxes to warn users in some cases where reloading a page may re-submit a POST request, it is generally up to the web application to handle cases where a POST request should not be submitted more than once. Note that whether a method is idempotent is not enforced by the protocol or web server. It is perfectly possible to write a web application in which (for example) a database insert or other non-idempotent action is triggered by a GET or other request. Ignoring this recommendation, however, may result in undesirable consequences, if a user agent assumes that repeating the same request is safe when it isn't.

5.3 Security

Implementing methods such as TRACE, TRACK and DEBUG are considered potentially insecure by some security professionals because attackers can use them to gather information or bypass security controls during attacks. Security software tools such as Tenable Nessus and Microsoft UrlScan Security Tool report on the presence of these methods as being security issues.[21] TRACK and DEBUG are not valid HTTP 1.1 verbs.[22]

6 Status codes

See also: List of HTTP status codes

In HTTP/1.0 and since, the first line of the HTTP response is called the *status line* and includes a numeric *status code* (such as "404") and a textual *reason phrase* (such as “Not Found”). The way the user agent handles the response primarily depends on the code and secondarily on the other response header fields. Custom status codes can be used since, if the user agent encounters a code it does not recognize, it can use the first digit of the code to determine the general class of the response.[23]

The standard *reason phrases* are only recommendations and can be replaced with “local equivalents” at the web developer's discretion. If the status code indicated a problem, the user agent might display the *reason phrase* to the user to provide further information about the nature of the problem. The standard also allows the user agent to attempt to interpret the *reason phrase*, though this might be unwise since the standard explicitly specifies that status codes are machine-readable and *reason phrases* are human-readable. HTTP status code is primarily divided into five groups for better explanation of request and responses between client and server as named: Informational 1XX, Successful 2XX, Redirection 3XX, Client Error 4XX and Server Error 5XX.

7 Persistent connections

Main article: HTTP persistent connection

In HTTP/0.9 and 1.0, the connection is closed after a single request/response pair. In HTTP/1.1 a keep-alive-mechanism was introduced, where a connection could be reused for more than one request. Such *persistent connections* reduce request latency perceptibly, because the client does not need to re-negotiate the TCP 3-Way-Handshake connection after the first request has been sent. Another positive side effect is that in general the connection becomes faster with time due to TCP’s slow-start-mechanism.

Version 1.1 of the protocol also made bandwidth optimization improvements to HTTP/1.0. For example, HTTP/1.1 introduced chunked transfer encoding to allow content on persistent connections to be streamed rather than buffered. HTTP pipelining further reduces lag time, allowing clients to send multiple requests before waiting for each response. Another improvement to the protocol was byte serving, where a server transmits just the portion of a resource explicitly requested by a client.

8 HTTP session state

HTTP is a stateless protocol. A stateless protocol does not require the HTTP server to retain information or status about each user for the duration of multiple requests. However, some web applications implement states or server side sessions using for instance HTTP cookies or Hidden variables within web forms.

9 Encrypted connections

The most popular way of establishing an encrypted HTTP connection is HTTP Secure.[24] Two other methods for establishing an encrypted HTTP connection also exist: Secure Hypertext Transfer Protocol, and using the HTTP/1.1 Upgrade header to specify an upgrade to TLS. Browser support for these two is, however, nearly non-existent.[25][26][27]

10 Request message

The request message consists of the following:

- A request line, for example *GET /images/logo.png HTTP/1.1*, which requests a resource called /images/logo.png from the server.
- Request header fields, such as *Accept-Language: en*.
- An empty line.
- An optional message body.

The request line and other header fields must each end with <CR><LF> (that is, a carriage return character followed by a line feed character). The empty line must consist of only <CR><LF> and no other whitespace.[28] In the HTTP/1.1 protocol, all header fields except Host are optional.

A request line containing only the path name is accepted by servers to maintain compatibility with HTTP clients before the HTTP/1.0 specification in RFC 1945.[29]

11 Response message

The response message consists of the following:

- A Status-Line, which include the status code and reason message. (e.g., *HTTP/1.1 200 OK*, which indicates that the client's request succeeded)
- Response header fields, such as *Content-Type: text/html*.
- An empty line
- An optional message body

The Status-Line and other header fields must all end with <CR><LF> (a carriage return followed by a line feed). The empty line must consist of only <CR><LF> and no other whitespace.[28]

12 Example session

Below is a sample conversation between an HTTP client and an HTTP server running on www.example.com, port 80.

12.1 Client request

GET /index.html HTTP/1.1 Host: www.example.com

A client request (consisting in this case of the request line and only one header field) is followed by a blank line, so that the request ends with a double newline, each in the form of a carriage return followed by a line feed. The "Host" field distinguishes between various DNS names sharing a single IP address, allowing name-based virtual hosting. While optional in HTTP/1.0, it is mandatory in HTTP/1.1.

12.2 Server response

HTTP/1.1 200 OK Date: Mon, 23 May 2005 22:38:34 GMT Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux) Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT ETag: "3f80f-1b6-3e1cb03b" Content-Type: text/html; charset=UTF-8 Content-Length: 138 Accept-Ranges: bytes Connection: close <html> <head> <title>An Example Page</title> </head> <body> Hello World, this is a very simple HTML document. </body> </html>

The ETag (entity tag) header field is used to determine if a cached version of the requested resource is identical to the current version of the resource on the server. *Content-Type* specifies the Internet media type of the data conveyed by the HTTP message, while *Content-Length* indicates its length in bytes. The HTTP/1.1 webserver publishes its ability to respond to requests for certain byte ranges of the document by setting the field *Accept-Ranges: bytes*. This is useful, if the client needs to have only certain portions[30] of a resource sent by the server, which is called byte serving. When *Connection: close* is sent, it means that the web server will close the TCP connection immediately after the transfer of this response.

Most of the header lines are optional. When *Content-Length* is missing the length is determined in other ways. Chunked transfer encoding uses a chunk size of 0 to mark the end of the content. *Identity* encoding without *Content-Length* reads content until the socket is closed.

A *Content-Encoding* like *gzip* can be used to compress the transmitted data.

13 Similar protocols

The Gopher protocol was a content delivery protocol that was displaced by HTTP in the early 1990s. The protocol SPDY is also similar to HTTP, modifying the request-response interaction between client and server.

14 See also

- Basic access authentication
- Constrained Application Protocol – A semantically similar protocol to HTTP but used UDP or UDP-like messages targeted for devices with limited processing capability. Re-uses HTTP and other internet concepts like Internet media type and web linking (RFC 5988)[31]
- Content negotiation
- Curl-loader – HTTP/S loading and testing open-source software
- Digest access authentication
- Fiddler (software)
- HTTP compression
- HTTP/2 – developed by the IETF's Hypertext Transfer Protocol Bis (httpbis) working group.[32]
- HTTP-MPLEX – A backwards compatible enhancement to HTTP to improve page and web object retrieval time in congested networks proposed by Robert Mattson
- List of file transfer protocols
- List of HTTP header fields
- List of HTTP status codes
- Representational state transfer (REST)
- Variant object
- Waka (protocol) – An HTTP replacement proposed by Roy Fielding
- Web cache
- WebSocket
- Wireshark

15 Notes

[1] Fielding, Roy T.; Gettys, James; Mogul, Jeffrey C.; Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee, Tim (June 1999). *Hypertext Transfer Protocol -- HTTP/1.1*. IETF. RFC 2616. https://tools.ietf.org/html/rfc2616.

[2] "Overall Operation". p. 12. sec. 1.4. RFC 2616. https://tools.ietf.org/html/rfc2616#section-1.4.

[3] Berners-Lee, Tim. "HyperText Transfer Protocol". World Wide Web Consortium. Retrieved 31 August 2010.

[4] Tim Berners-Lee. "The Original HTTP as defined in 1991". World Wide Web Consortium. Retrieved 24 July 2010.

[5] Raggett, Dave. "Dave Raggett's Bio". World Wide Web Consortium. Retrieved 11 June 2010.

[6] Raggett, Dave; Berners-Lee, Tim. "Hypertext Transfer Protocol Working Group". World Wide Web Consortium. Retrieved 29 September 2010.

[7] Raggett, Dave. "HTTP WG Plans". World Wide Web Consortium. Retrieved 29 September 2010.

[8] Simon Spero. "Progress on HTTP-NG". World Wide Web Consortium. Retrieved 11 June 2010.

[9] "HTTP/1.1". *Webcom.com Glossary entry*. Retrieved 2009-05-29.

[10] Fielding, Roy T.; Reschke, Julian F. (June 2014). *Hypertext Transfer Protocol (HTTP/1.1): Authentication*. IETF. RFC 7235. https://tools.ietf.org/html/rfc7235.

[11] Berners-Lee, Tim; Fielding, Roy T.; Nielsen, Henrik Frystyk. "Method Definitions". *Hypertext Transfer Protocol -- HTTP/1.0*. IETF. pp. 30-32. sec. 8. RFC 1945. https://tools.ietf.org/html/rfc1945#section-8.

[12] "Method Definitions". pp. 51-57. sec. 9. RFC 2616. https://tools.ietf.org/html/rfc2616#section-9.

[13] Jacobs, Ian (2004). "URIs, Addressability, and the use of HTTP GET and POST". *Technical Architecture Group finding*. W3C. Retrieved 26 September 2010.

[14] "POST". p. 54. sec. 9.5. RFC 2616. https://tools.ietf.org/html/rfc2616#section-9.5.

[15] "PUT". p. 55. sec. 9.6. RFC 2616. https://tools.ietf.org/html/rfc2616#section-9.6.

[16] "CONNECT". *Hypertext Transfer Protocol -- HTTP/1.1*. IETF. June 1999. p. 57. sec. 9.9. RFC 2616. https://tools.ietf.org/html/rfc2616#section-9.9. Retrieved 23 February 2014.

[17] Khare, Rohit; Lawrence, Scott (May 2000). *Upgrading to TLS Within HTTP/1.1*. IETF. RFC 2817. https://tools.ietf.org/html/rfc2817.

[18] "Vulnerability Note VU#150227: HTTP proxy default configurations allow arbitrary TCP connections". US-CERT. 2002-05-17. Retrieved 2007-05-10.

[19] Dusseault, Lisa; Snell, James M. (March 2010). *PATCH Method for HTTP*. IETF. RFC 5789. https://tools.ietf.org/html/rfc5789.

[20] "Method". p. 36. sec. 5.1.1. RFC 2616. https://tools.ietf.org/html/rfc2616#section-5.1.1.

[21] "UrlScan Security Tool". *Security TechCenter*. Microsoft. Retrieved 15 Jul 2012.

[22] "TRACE". pp. 56-57. sec. 9.8. RFC 2616. https://tools.ietf.org/html/rfc2616#section-9.8.

[23] "Status-Line". p. 39. sec. 6.1. RFC 2616. https://tools.ietf.org/html/rfc2616#section-6.1.

[24] Canavan, John (2001). *Fundamentals of Networking Security*. Norwood, MA: Artech House. pp. 82–83. ISBN 9781580531764.

[25] Zalewski, Michal. "Browser Security Handbook". Retrieved 30 April 2015.

[26] "Chromium Issue 4527: implement RFC 2817: Upgrading to TLS Within HTTP/1.1". Retrieved 30 April 2015.

[27] "Mozilla Bug 276813 - [RFE] Support RFC 2817 / TLS Upgrade for HTTP 1.1". Retrieved 30 April 2015.

[28] "HTTP Message". p. 31. sec. 4. RFC 2616. https://tools.ietf.org/html/rfc2616#section-4.

[29] "Apache Week. HTTP/1.1". 090502 apacheweek.com

[30] Luotonen, Ari; Franks, John (February 22, 1996). *Byte Range Retrieval Extension to HTTP*. IETF. I-D draft-ietf-http-range-retrieval-00. https://tools.ietf.org/html/draft-ietf-http-range-retrieval-00.

[31] Nottingham, Mark (October 2010). *Web Linking*. IETF. RFC 5988. https://tools.ietf.org/html/rfc5988.

[32] "Hypertext Transfer Protocol Bis (httpbis) – Charter". IETF. 2012.

16 References

- HTTP 0.9 – As Implemented in 1991

17 External links

- "Change History for HTTP". W3.org. Retrieved 2010-08-01. A detailed technical history of HTTP.
- "Design Issues for HTTP". W3.org. Retrieved 2010-08-01. Design Issues by Berners-Lee when he was designing the protocol.
- "Classic HTTP Documents". W3.org. 1998-05-14. Retrieved 2010-08-01. list of other classic documents recounting the early protocol history

18 Text and image sources, contributors, and licenses

18.1 Text