9. Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. 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), most notably RFC 2616 (June 1999), which defined HTTP/1.1, the version of HTTP most commonly used today. In June 2014, RFC 2616 was retired and HTTP/1.1 was redefined by RFCs 7230, 7231, 7232, 7233, 7234, and 7235. HTTP/2 is currently in draft form.

9.1 Technical overview

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,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 Identifiers (URIs)—or, more specifically, Uniform Resource Locators (URLs)—using the http or https URI schemes. URIs and hyperlinks in Hypertext Markup Language (HTML) documents form webs of 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.

9.2 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 (1930's) 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 are credited with inventing the original HTTP along with HTML and the associated technology for a web server and a text-based 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. The response from the server was always an HTML page.

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. RFC 1945 officially introduced and recognized HTTP V1.0 in 1996.

The HTTP WG planned to publish new standards in December 1995[8] 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, Netscape 2.0, Netscape Navigator Gold 2.01, 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. 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 spec. 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

9.3 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.

9.4 Request methods

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

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 defined the GET, POST and HEAD methods and the HTTP/1.1 specification 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

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.) 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." See safe methods below.

HEAD

Asks for the response identical to the one that would correspond to 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

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.

PUT

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.

DELETE

Deletes the specified resource.

TRACE

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

OPTIONS

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

Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy. See HTTP CONNECT Tunneling.

PATCH

Applies partial modifications to a resource.

HTTP servers are required to implement at least the GET and HEAD methods and, whenever possible, also the OPTIONS method.

9.5 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's explicitly acknowledged that it's impossible to guarantee such a thing.

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.

9.6 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. 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.

9.7 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. TRACK and DEBUG are not valid HTTP 1.1 verbs.

9.8 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.

Also, 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.

9.9 Persistent connections

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.

9.10 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.

9.11 Encrypted connections

The most popular way of establishing an encrypted HTTP connection is HTTP Secure.

Two other methods for establishing an encrypted HTTP connection also exist, called Secure Hypertext Transfer Protocol and the HTTP/1.1 Upgrade header. Browser support for these latter two is, however, nearly non-existent;[citation needed] so HTTP Secure is the dominant method of establishing an encrypted HTTP connection.

9.12 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. 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.

9.13 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.

9.14 Example session

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

9.15 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.

9.15 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: 131

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[26] 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.

9.16 Similar protocols

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