HTTP: A PROTOCOL FOR NETWORKED INFORMATION

*Devadharshini Subiksha R

**Ajith kirthic TC

***Samritha V

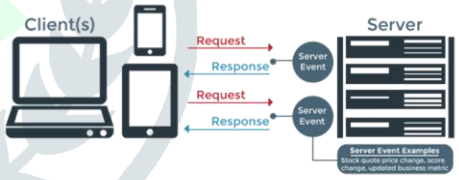
Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for a distributed collaborative hypermedia information system. It is a generic stateless object-oriented protocol, which may be used for many similar tasks such as name servers, and distributed object-oriented systems, by extending the commands, or "methods", used. . A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the development of new advanced representations.

1. INTRODUCTION

HTTP is a protocol which allows the fetching of resources, such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, images, videos, scripts, and more.

Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser, are called requests and the messages sent by the server as an answer are called responses.



HTTP as an application layer protocol, on top of TCP (transport layer) and IP (network layer) and below the presentation layer. Designed in the early 1990s, HTTP is an extensible protocol which has evolved over time. It is an application layer protocol that is sent over TCP, or over a TLS-encrypted TCP connection, though any reliable transport protocol could theoretically be used. Due to its extensibility, it is used to not only fetch hypertext documents, but also images and

videos or to post content to servers, like with HTML form results. HTTP can also be used to fetch parts of documents to update Web pages on demand.

1.1 PURPOSE

The name hypertext transfer protocol refers to HTTP's role in transmitting website data across the internet. Hypertext refers to the standard form of websites in which one page can refer users to another page through clickable hyperlinks, usually simply called links. The purpose of the HTTP protocol is to provide a standard way for web browsers and servers to talk to each other.

Web pages are designed using the hypertext markup language, or HTML, but HTTP is used today to transfer more than simply HTML and the cascading style sheets, or CSS, used to indicate how pages should be displayed. HTTP is also used to transfer other content on websites including images, video and audio files.

1.2 REQUIREMENTS

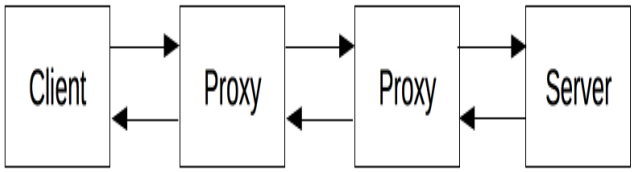
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be "conditionally compliant."

2. Components of HTTP-based systems

HTTP is a client-server protocol: requests are sent by one entity, the user-agent (or a proxy on behalf of it). Most of the time the user-agent is a Web browser, but it can be anything, for example a robot that crawls the Web to populate and maintain a search engine index.

Each individual request is sent to a server, which handles it and provides an answer, called the response. Between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways or caches, for example.



In reality, there are more computers between a browser and the server handling the request: there are routers, modems, and more. Thanks to the layered design of the Web, these are hidden in the network and transport layers. HTTP is on top, at the application layer. Although important to diagnose network problems, the underlying layers are mostly irrelevant to the description of HTTP.

2.1 Client: the user-agent

The user-agent is any tool that acts on the behalf of the user. This role is primarily performed by the Web browser; other possibilities are programs used by engineers and Web developers to debug their applications.

The browser is always the entity initiating the request. It is never the server (though some mechanisms have been added over the years to simulate server-initiated messages).

To present a Web page, the browser sends an original request to fetch the HTML document that represents the page. It then parses this file, making additional requests corresponding to execution scripts, layout information (CSS) to display, and sub-resources contained within the page (usually images and videos). The Web browser then mixes these resources to present to the user a complete document, the Web page. Scripts executed by the browser can fetch more resources in later phases and the browser updates the Web page accordingly.

A Web page is a hypertext document. This means some parts of displayed text are links which can be activated (usually by a click of the mouse) to fetch a new Web page, allowing the user to direct their user-agent and navigate through the Web. The browser translates these directions in HTTP requests, and further interprets the HTTP responses to present the user with a clear response.

2.2 The Web server

On the opposite side of the communication channel, is the server, which serves the document as requested by the client. A server appears as only a single machine virtually: this is because it may actually be a collection of servers, sharing the load (load balancing) or a complex piece of software interrogating other computers (like cache, a DB server, or e-commerce servers), totally or partially generating the document on demand.

A server is not necessarily a single machine, but several server software instances can be hosted on the same machine. With HTTP/1.1 and the

Host header, they may even share the same IP address.

2.3 Proxies

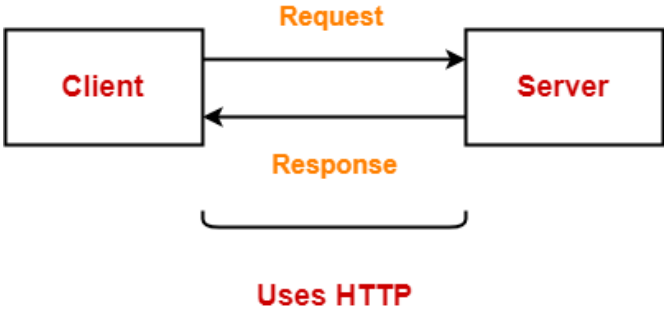
Between the Web browser and the server, numerous computers and machines relay the HTTP messages. Due to the layered structure of the Web stack, most of these operate at the transport, network or physical levels, becoming transparent at the HTTP layer and potentially making a significant impact on performance. Those operating at the application layers are generally called proxies. These can be transparent, forwarding on the requests they receive without altering them in any way, or non-transparent, in which case they will change the request in some way before passing it along to the server. Proxies may perform numerous functions:

- Caching (the cache can be public or private, like the browser cache)
- Filtering (like an antivirus scan or parental controls)
- Load balancing (to allow multiple servers to serve the different requests)
- Authentication (to control access to different resources)
- Logging (allowing the storage of historical information)

2.4 Working

HTTP uses a client-server model where-

- Web browser is the client.
- Client communicates with the web server hosting the website.



Whenever a client requests some information (say clicks on a hyperlink) to the website server.

The browser sends a request message to the HTTP server for the requested objects.

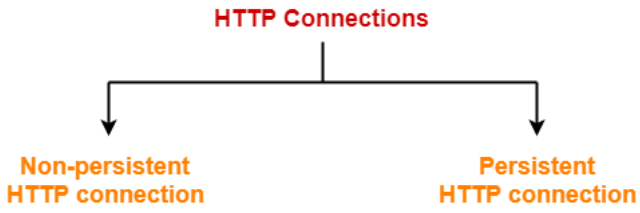
Then-

- HTTP opens a connection between the client and server through TCP.
- HTTP sends a request to the server which collects the requested data.
- HTTP sends the response with the objects back to the client.
- HTTP closes the connection.

3. HTTP CONNECTIONS

HTTP connections can be of two types-

- Non-persistent HTTP connection
- Persistent HTTP connection



3.1 Non-persistent HTTP connection

- Non-persistent HTTP connection is one that is used for serving exactly one request and sending one response.
- HTTP server closes the TCP connection automatically after sending a HTTP response.
- A new separate TCP connection is used for each object.
- HTTP 1.0 supports non-persistent connections by default.

Example-

Suppose a request has been made for a HTML page that contains 10 images (called objects).

Then,

With non-persistent connection, all the 11 objects (1 page + 10 images) will be sent one by one.

For getting each object, a new separate connection will be opened and used.

3.2 Persistent HTTP connection

- Persistent HTTP connection is one that can be used for serving multiple requests.
- HTTP server closes the TCP connection only when it is not used for a certain configurable amount of time.
  - A single TCP connection is used for sending multiple objects one after the other.
  - HTTP 1.1 supports persistent connections by default.

Example-

Suppose a request has been made for a HTML page that contains 10 images (called objects).

Then,

With persistent connection, all the 11 objects (1 page + 10 images) will be sent one after the other using a single TCP connection.

4. BASIC FEATURES OF HTTP

There are three basic features that make HTTP a simple but powerful protocol:

HTTP is connectionless: The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client waits for the response. The server processes the request and sends a response back

after which client disconnect the connection. So client and server knows about each other during current request and response only. Further requests are made on new connection like client and server are new to each other.

HTTP is media independent: It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME-type.

HTTP is stateless: As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

5. HTTP - SECURITY

HTTP is used for communications over the internet, so application developers, information providers, and users should be aware of the security limitations in HTTP/1.1. This discussion does not include definitive solutions to the problems mentioned here but it does make some suggestions for reducing security risks.

5.1 Personal Information Leakage

HTTP clients are often privy to large amount of personal information such as the user's name, location, mail address, passwords, encryption keys, etc. So you should be very careful to prevent unintentional leakage of this information via the HTTP protocol to other sources.

- All the confidential information should be stored at the server in encrypted form.
- Revealing the specific software version of the server might allow the server machine to become more vulnerable to attacks against software that is known to contain security holes.
- Proxies that serve as a portal through a network firewall should take special precautions regarding the transfer of header information that identifies the hosts behind the firewall.
- The information sent in the 'From' field might conflict with the user's privacy interests or their site's security policy, and hence, it should not be transmitted without the user being able to disable, enable, and modify the contents of the field.
- Clients should not include a Referrer header field in a (non-secure) HTTP request, if the

referring page was transferred with a secure protocol.

- Authors of services that use the HTTP protocol should not use GET based forms for the submission of sensitive data, because it will cause the data to be encoded in the Request-URI.

5.2 File and Path Names Based Attack

The document should be restricted to the documents returned by HTTP requests to be only those that were intended by the server administrators.

For example, UNIX, Microsoft Windows, and other operating systems use '..' as a path component to indicate a directory level above the current one. On such a system, an HTTP server MUST disallow any such construct in the Request-URI, if it would otherwise allow access to a resource outside those intended to be accessible via the HTTP server.

5.3 DNS Spoofing

Clients using HTTP rely heavily on the Domain Name Service, and are thus generally prone to security attacks based on the deliberate mis-association of IP addresses and DNS names. So clients need to be cautious in assuming the continuing validity of an IP number/DNS name association.

If HTTP clients cache the results of host name lookups in order to achieve a performance improvement, they must observe the TTL information reported by the DNS. If HTTP clients do not observe this rule, they could be spoofed when a previously-accessed server's IP address changes.

5.4 Location Headers and Spoofing

If a single server supports multiple organizations that do not trust one another, then it MUST check the values of Location and Content Location headers in the responses that are generated under the control of said organizations to make sure that they do not attempt to invalidate resources over which they have no authority.

5.5 Authentication Credentials

Existing HTTP clients and user agents typically retain authentication information indefinitely. HTTP/1.1 does not provide a method for a server to direct clients to discard these cached credentials which are a big security risk.

There are a number of works around to the parts of this problem, and so it is recommended to make the use of password protection in screen savers, idle time-outs, and other methods that mitigate the security problems inherent in this problem.

5.6 Proxies and Caching

HTTP proxies are men-in-the-middle, and represent an opportunity for man-

in-the-middle attacks. Proxies have access to security-related information, personal information about individual users and organizations, and proprietary information belonging to users and content providers.

Proxy operators should protect the systems on which proxies run, as they would protect any system that contains or transports sensitive information.

Caching proxies provide additional potential vulnerabilities, since the contents of the cache represent an attractive target for malicious exploitation. Therefore, cache contents should be protected as sensitive information.

6. HTTP – PARAMETERS

6.1 HTTP Version

HTTP uses a **<major>.<minor>** numbering scheme to indicate versions of the protocol. The version of an HTTP message is indicated by an HTTP-Version field in the first line. Here is the general syntax of specifying HTTP version number:

```
HTTP-Version = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

Example:

```
HTTP/1.0 (or) HTTP/1.1
```

6.2 Uniform Resource Identifiers

Uniform Resource Identifiers (URI) are simply formatted, case-insensitive string containing name, location, etc. to identify a resource, for example, a website, a web service, etc. A general syntax of URI used for HTTP is as follows:

```
URI = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

Here if the port is empty or not given, port 80 is assumed for HTTP and an empty abs_path is equivalent to an abs_path of "/". The characters other than those in the reserved and unsafe sets are equivalent to their ""%" HEX HEX" encoding.

Example:

The following two URIs are equivalent:

```
http://abc.com:80/~smith/home.html

http://ABC.com/%7Esmith/home.html
```

6.3 Date/Time Formats

All HTTP date/time stamps MUST be represented in Greenwich Mean Time (GMT), without exception. HTTP applications are allowed to use any of the following three representations of date/time stamps:

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 822, updated by RFC 1123

Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036

Sun Nov  6 08:49:37 1994       ; ANSI C's asctime() format
```

6.4 Character Sets

We use character sets to specify the character sets that the client prefers. Multiple character sets can be listed separated by commas. If a value is not specified, the default is the US-ASCII.

Example :

Following are the valid character sets:

```
US-ASCII (or)  ISO-8859-1 (or) ISO-8859-7
```

6.5 Content Encodings

A content encoding value indicates that an encoding algorithm has been used to encode the content before passing it over the network. Content coding are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity.

All content-coding values are case-insensitive. HTTP/1.1 uses content-coding values in the Accept-Encoding and Content-Encoding header fields which we will see in the subsequent chapters.

Example:

Following are the valid encoding schemes:

```
Accept-encoding: gzip (or)

Accept-encoding: compress (or)

Accept-encoding: deflate
```

6.6 Media Types

HTTP uses Internet Media Types in the Content-Type and Accept header fields in order to provide open and extensible data typing and type negotiation. All the Media-type values are registered with the Internet Assigned Number Authority (IANA). The general syntax to specify media type is as follows:

```
media-type    = type "/" subtype *( ";" parameter )
```

The type, subtype, and parameter attribute names are case--insensitive.

Example :

```
Accept: image/gif
```

6.7 Language Tags

HTTP uses language tags within the Accept-Language and Content-Language fields. A language tag is composed of one or more parts: a

primary language tag and a possibly empty series of subtags:

language-tag = primary-tag *("-" subtag)

White spaces are not allowed within the tag and all tags are case-insensitive.

Example:

Example tags include:

en, en-US, en-cockney, i-cherokee, x-pig-latin

where any two-letter primary-tag is an ISO-639 language abbreviation and any two-letter initial subtag is an ISO-3166 country code.

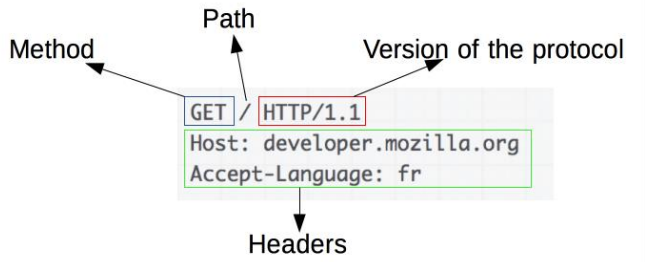
7. HTTP MESSAGES

HTTP messages, as defined in HTTP/1.1 and earlier, are human-readable. In HTTP/2, these messages are embedded into a binary structure, a frame, allowing optimizations like compression of headers and multiplexing. Even if only part of the original HTTP message is sent in this version of HTTP, the semantics of each message is unchanged and the client reconstitutes (virtually) the original HTTP/1.1 request. It is therefore useful to comprehend HTTP/2 messages in the HTTP/1.1 format.

There are two types of HTTP messages, requests and responses, each with its own format.

7.1 Requests

An example HTTP request:



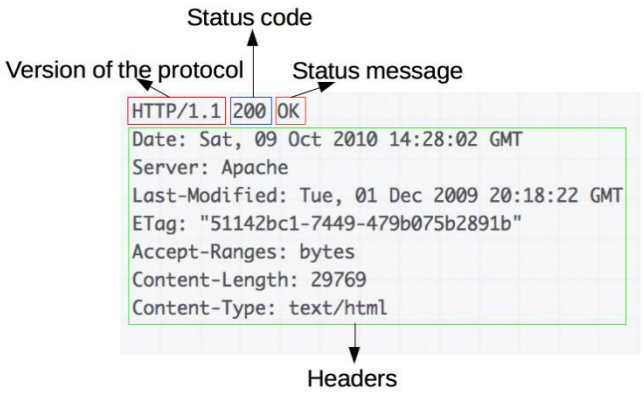
Requests consist of the following elements:

- An HTTP method, usually a verb like GET, POST or a noun like OPTIONS or HEAD that defines the operation the client wants to perform. Typically, a client wants to fetch a resource (using GET) or post the value of an HTML form (using POST), though more operations may be needed in other cases.
- The path of the resource to fetch; the URL of the resource stripped from elements that are obvious from the context, for example without the protocol (http://), the domain (here, developer.mozilla.org), or the TCP port (here, 80).
- The version of the HTTP protocol.
- Optional headers that convey additional information for the servers.

- Or a body, for some methods like POST, similar to those in responses, which contain the resource sent.

7.2 Responses

An example response:



Responses consist of the following elements:

- The version of the HTTP protocol they follow.
- A status code, indicating if the request was successful, or not, and why.
- A status message, a non-authoritative short description of the status code.
- HTTP headers like those for requests.
- Optionally, a body containing the fetched resource.

8. ADVANTAGES OF HTTP

Hypertext Transfer Protocol, better known to millions of Web surfers as HTTP, was invented in 1990 by Tim Berners-Lee at the CERN Laboratories in Geneva, Switzerland. Today, it is the foundation of the World Wide Web and the Hypertext Markup Language or HTML. Three versions of HTTP were developed: 0.9, 1.0 and 1.1. Both 1.0 and 1.1 are in common usage today.

8.1 Identification

HTML was intended to be quick and lightweight. Speed of delivery is enabled by creating a notification of file type in the header of the data being transferred, known as MIME type. This enables the receiving application to quickly open the incoming file without having to ask the sender what application should be used to read or view the contents of the file.

8.2 Specialization

A Web page contains mixed elements such as text and images. Each element requires a different amount of resources to store and download. HTTP enables multiple connections to download separate elements concurrently, thus speeding up transmission. Each element is assigned its own particular file type and therefore can be handled faster and more efficiently by the receiving computer.

8.3 Addressing

The addressing scheme used by HTTP was also a revolutionary

advancement. When computers had to be addressed using an IP address consisting of a series of numbers, the public found it difficult to engage with the Internet. Mapping IP addresses to easily recognizable names made the World Wide Web commercially viable.

8.4 Flexibility

With file type notification preceding data transmission, the receiving application has the option of quickly downloading extensions or plug-ins if additional capabilities are needed to display the data. These add-ons include Flash players and PDF document readers.

8.5 Security

HTTP 1.0 downloads each file over an independent connection and then closes the connection. This reduces the risk of interception during transmission, as the connection does not persist beyond the transfer of a single element of a Web page. Hypertext Transfer Protocol Secure (HTTPS) encrypts the HTTP exchange to add further security.

8.6 Ease of Programming

HTTP is coded in plain text and therefore is easier to follow and implement than protocols that make use of codes that require lookups. Data is formatted in lines of text and not as strings of variables or fields.

8.7 Search Capabilities

Although HTTP is a simple messaging protocol, it includes the ability to search a database with a single request. This allows the protocol to be used to carry out SQL searches and return results conveniently formatted in an HTML document.

8.8 Persistent Connections

One minor drawback of HTTP is the need to create multiple connections in order to transmit a typical Web page, which causes an administrative overhead. HTTP 1.1 has the ability to maintain an open connection for several requests. In addition, the concept of "pipelining" was added, enabling many requests to be sent to the receiving computer before the first request is served. These two measures speed up the response time for delivering a Web page.

9. DISADVANTAGES OF HTTP

- Information sent via HTTP is not encrypted and can pose a threat to your privacy.
- Packet headers are larger than other protocols as they are needed for security and quality assurance of the information being transferred.
- Information sent through a browser can expose your computer

to virtual threats. Such information sent through your browser headers can be the name of your computer, your IP address, what OS you are running and any other requested information that the website you have requested wants to know.

- Is slower than other native protocols but is more secure, reliable and can transfer larger chunks of data.

10. HTTP METHODS

A request line has three parts, separated by spaces: a method name, the local path of the requested resource, and the version of HTTP being used. There are three HTTP Methods, namely GET, HEAD, and POST.

10.1 The GET Method

GET is the most common HTTP method; it says "give me this resource". Other Method names are always in uppercase. The GET method can also be used to submit forms. The form data are URL-encoded and appended to the request URL.

10.2 The HEAD Method

A HEAD request is just like a GET request, except it asks the server to return the response headers only, and not the actual resource (i.e., no message body). This is useful to check characteristics of a resource without actually downloading it, thus saving bandwidth. Use HEAD when you don't actually need a file's contents. The response to a HEAD request must never contain a message body, but just the status line and headers.

10.3 The POST Method

POST request is used to send data to be processed to the server in some way, such as using a CGI script. A POST request is different from a GET request in the following ways:

- There's a block of data sent with the request, in the message body. There are usually extra headers to describe this message body, like Content-Type: and Content-Length:.
- The request URL is not a resource to retrieve; it's usually a program to handle the data you're sending.

The HTTP response is normally program output, not a static file. The most common use of POST, by far, is to submit HTML form data to CGI scripts. In this case, the Content-Type: header is usually application/x-www-form-url encoded, and the Content-Length: header gives the length of the URL-encoded form data. You can use a POST request to send whatever data you want, not just form submissions. Just make sure the sender and the

receiving program agree on the format.

11. CONCLUSION

HTTP is an extensible protocol that is easy to use. The client-server structure, combined with the ability to simply add headers, allows HTTP to advance along with the extended capabilities of the Web.

12. RREFERENCE LINKS

1. https://www.w3.org/Protocols/rfc2616/rfc2616.html
2. https://www.answers.com/Q/What_is_the_disadvantages_of_HTTP
3. https://www.webnms.com/cagent/help/technology_used/c_http_overview.html
4. https://www.webnots.com/what-is-http/
5. https://www.techwalla.com/articles/what-is-hypertext-transfer-protocol
6. https://www.w3schools.com/whatis/whatis_http.asp