## HTTP 0.9: The One-Line Protocol

The original HTTP proposal by Tim Berners-Lee was designed with *simplicity in mind* as to help with the adoption of his other idea:

The World Wide Web, the strategy appears to have worked: aspiring protocol designers.

In 1991, Berners-Lee outlined the motivation for the new protocol and listed several high-level design goals: file transfer functionality, ability to request an index search of a hypertext archive, format negotiation, and an ability to refer the client to another server.

* Server response is a hypertext markup language (HTML).
* Connection is terminated after the document transfer is complete.

However, even that sounds a lot more complicated than it really is. What these rules enable is an extremely simple,

Telnet-friendly protocol, which some web servers support to this very day:

*$> telnet google.com 80*

*Connected to 74.125.xxx.xxx*

*GET /about/*

*(hypertext response)*

*(connection closed)*

The request consists of a single line:

* GET method and the path of the requested document.
* The response is a single hypertext document - no headers or any other metadata, just the HTML. It really couldn’t get any simpler.

Further, since the previous interaction is a subset of the intended protocol, it unofficially acquired the HTTP 0.9 label. From the beginning in 1991, HTTP took on a life of its own and evolved rapidly over the coming years.

**Features of HTTP 0.9:**

* Client-server, request-response protocol.
* ASCII protocol, running over a TCP/IP link.
* Designed to transfer hypertext documents (HTML).
* The connection between server and client is closed after every request

## HTTP/1.0: Rapid Growth and Informational RFC

From 1991 to 1995 is one of rapid coevolution of the HTML specification,

* Software known as a "web browser,"
* Emergence and quick growth of the consumer-oriented public Internet infrastructure.
* The status code line was sent at the beginning of the response, and it allows the browsers to understand the success or the failure of the request.
* With the help of the ***new HTTP headers***, the ability to transmit documents other than plain HTML files was added (content-Type Header).
* November 1996, an informational document about common practices was published ***RFC 1945***.

**Basic Rule**

The following rules are used to describe basic parsing constructs.

The US-ASCII coded character set is defined by:

* *OCTET = <any 8-bit sequence of data>*
* *CHAR = <any US-ASCII character (octets 0 - 127)>*
* *UPALPHA = <any US-ASCII uppercase letter "A"-"Z">*
* *LOALPHA = <any US-ASCII lowercase letter "a"-"z">*
* *ALPHA = UPALPHA | LOALPHA*
* *DIGIT = <any US-ASCII digit "0"-"9">*
* *CTL = <any US-ASCII control character (octets 0 - 31) and DEL (127)>*
* *CR = <US-ASCII CR, carriage return (13)>*
* *LF = <US-ASCII LF, linefeed (10)>*
* *SP = <US-ASCII SP, space (32)>*
* *HT = <US-ASCII HT, horizontal-tab (9)>*
* *<"> = <US-ASCII double-quote mark (34)>*

**Protocol Parameters**

**HTTP Version**

HTTP uses <major> and <minor> numbering scheme to indicate versions of the protocol. This is intended to allow the sender to indicate what format and capacity of a message for understanding further HTTP communication.

* The <minor> number will increment when changes is made into the protocol add features which it does not change the general message.
* The <major> number is incremented when the format of the message within the protocol was changed.

**Status Code and Reason Phrase**

* **1xx**: Information – Not Used, but reserved for future use.
* **2xx**: Success – The Action was successfully received, understood, and accepted.
  + "200" ; OK
  + "201" ; Created
  + "202" ; Accepted
  + "204" ; No Content
* **3xx**: Redirection – Further action must be taken in order complete the request.
  + "301" ; Moved Permanently
  + "302" ; Moved Temporarily
  + "304" ; Not Modified
* **4xx**: Client Error – The Request contains bad syntax or cannot be fulfilled.
  + "400" ; Bad Request
  + "401" ; Unauthorized
  + "403" ; Forbidden
  + "404" ; Not Found
* **5xx**: Server Error – The Server failed to fulfill an apparently valid request.
  + "500" ; Internal Server Error
  + "501" ; Not Implemented
  + "502" ; Bad Gateway
  + "503" ; Service Unavailable

**Method Definitions**

* **GET:** 
  + Retrieve whatever information that is identified by the Request-URI.
* **HEAD:** 
  + Is identical to the GET, but the server must not return any Entity-Body (The Entity Body (if any) sent with an HTTP request or response is in a format and encoding defined by the Entity-Header fields.) in the response, this method is often used for testing hypertext links for validity.
* **POST:**
  + The POST method is used to request that the destination server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.
  + The Post method is determined by the server and it is usually dependent on the Request-URI.

## HTTP/1.1: The Standardized Protocol / Internet Standard

HTTP/1.1 was published in the early 1997 that is only a few months after HTTP/1.0, HTTP/1.1 clarified the ambiguities of the last version and introduced numerous improvements.

* Pipelining
  + Without waiting for each of the response you can send multiple request on the same Socket, allowing a single TCP connection to be used efficiently with much lower elapsed time.
* Chunked responses are supported
  + Instead of processing the whole page, it will generate all of the HTML and the HTML will be divided into chunks and then send it to the client one after the other.
* Additional Cache control mechanism
* Transfer Encoding
  + Transfer encoding and message length interact in a way that it is required fixing exactly when chunked encoding is used this is to allow transfer encoding that it may not be self delimiting, and it is important to straighten out exactly how message length are computed.
* The Ability to host different domains at the same IP, thanks to the Host header.
* January 1997 - HTTP/1.1 was published as ***RFC 2068***.
* A number of improvements and updates were incorporated into the standard and were released as ***RFC 2616***.

**Method Definitions**

* **Safe Methods** – these are HTTP method that does not modify the resources, Safe methods are also methods that can be cached, it is not possible to ensure the server will not generate side effects as a result when using the GET request**.**
  + **HEAD**
  + **GET**
  + **OPTION**
    - This method represents a request for information about the communication options available on the request or response identified by the Request-URI.
    - Responses are not cacheable.
* **Idempotent Methods –** these are HTTP methods that can be called many times without different outcomes, however, it is possible that a sequence of several requests is non-idempotent, even if all of the methods executed in that sequence are idempotent.
  + **OPTION**
  + **GET**
  + **HEAD**
  + **PUT**
    - This method requests that the enclosed entity be stored under the supplied Request-URI.
  + **DELETE**
    - This method requests that the origin server delete the resource identified by the Request-URI.

## HTTP/2: Improving Transport Performance

HTTP/2 modifies how data is framed and transported between client and server. HTTP/2 has more efficient use of network resources and it reduces the latency by using the field compression. The Primary Focus of HTTP/2 is on improving the transport performance and enabling the higher throughput and lower latency.

* Response Multiplexing
* Minimize protocol overhead by compression of HTTP header fields
* Request prioritization
* Server Push

## References

**https://tools.ietf.org/html/rfc1945**

**https://hpbn.co/brief-history-of-http/**

**https://tools.ietf.org/html/rfc2616**

**https://tools.ietf.org/html/rfc7540**