Assignment : 1(1)

- **Research on HTTP protocol using various online resources mainly the latest RFC describing the protocol and prepare a paper in IEEE format with maximum 10 pages and should be covering following aspects.**
  - **Basics of HTTP Protocol**
  - **HTTP Request/Response**
  - **HTTP message headers**
  - **HTTP error codes**

Hyper Text Transfer Protocol Protocol

Rajan M. Sangada (170170107088)
B.E. Student of Computer Engineering
Vishwakarma Government Engineering College
Gujarat Technological University
Ahmedabad, India.

Abstract--The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collabor-ative, hypermediainformation systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use forhypertext, such as name servers and distributed object management systems, through extension of its requestmethods, error codes and headers. A feature of HTTP is the typing and negoti-ation of data representation,allowing systems to be built independently of the data being transferred. HTTP has been in use by the World-Wide Web global information initiative since 1990. This specification definesthe protocol referred to as "HTTP/1.1", and is an update to RFC 2068.

I. Introduction

The Hyper Text Transfer Protocol (HTTP) is the client-server network protocol that has been in use by the World-Wide Web since 1990. Whenever you surf the web, your browser will be sending HTTP request messages for HTML pages, images, scripts and styles sheets. Web servers handle these requests by returning response messages that contain the requested resource.

The term hypertext was coined by Ted Nelson in 1965 in the Xanadu Project, which was in turn inspired by Vannevar Bush's 1930s vision of the microfilm-based information retrieval and management "memex" system described in his 1945 essay "As We May Think". 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 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. 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. There have been several versions of HTTP starting with the original **0.9 version**. RFC 1945 officially introduced and recognized HTTP V1.0 in 1996.

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

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. To prove the theory in action, a simple prototype was built, which implemented a small subset of the proposed functionality:

- Client request is a single ASCII character string.
- Client request is terminated by a carriage return (CRLF).
- Server response is an ASCII character stream.
- 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. The rest, as they say, is history.

From these humble beginnings in 1991, HTTP took on a life of its own and evolved rapidly over the coming years. Let us quickly recap the 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.

Popular web servers, such as Apache and Nginx, still support the HTTP 0.9 protocol—in part, because there is not much to it! If you are curious, open up a Telnet session and try accessing google.com, or your own favorite site, via HTTP 0.9 and inspect the behavior and the limitations of this early protocol.

HTTP/1.0: Rapid Growth and Informational RFC

The period from 1991 to 1995 is one of rapid coevolution of the HTML specification, a new breed of software known as a "web browser," and the emergence and quick growth of the consumer-oriented public Internet infrastructure.

The Perfect Storm: Internet Boom of the Early 1990s

Building on Tim Berner-Lee's initial browser prototype, a team at the National Center of Supercomputing Applications (NCSA) decided to implement their own version. With that, the first popular browser was born: NCSA Mosaic. One of the programmers on the NCSA team, Marc Andreessen, partnered with Jim Clark to found Mosaic Communications in October 1994. The company was later renamed Netscape, and it shipped Netscape Navigator 1.0 in December 1994. By this point, it was already clear that the World Wide Web was bound to be much more than just an academic curiosity.

In fact, that same year the first World Wide Web conference was organized in Geneva, Switzerland, which led to the creation of the World Wide Web Consortium (W3C) to help guide the evolution of HTML. Similarly, a parallel HTTP Working Group (HTTP-WG) was established within the IETF to focus on improving the HTTP protocol. Both of these groups continue to be instrumental to the evolution of the Web.

Finally, to create the perfect storm, CompuServe, AOL, and Prodigy began providing dial-up Internet access to the public within the same 1994–1995 time frame. Riding on this wave of rapid adoption, Netscape made history with a wildly successful IPO on August 9, 1995—the Internet boom had arrived, and everyone wanted a piece of it!

The growing list of desired capabilities of the nascent Web and their use cases on the public Web quickly exposed many of the fundamental limitations of HTTP 0.9: we needed a protocol that could serve more than just hypertext documents, provide richer metadata about the request and the response, enable content negotiation, and more. In turn, the nascent community of web developers responded by producing a large number of experimental HTTP server and client implementations through an ad hoc process: implement, deploy, and see if other people adopt it.

From this period of rapid experimentation, a set of best practices and common patterns began to emerge, and in May 1996 the HTTP Working Group (HTTP-WG) published RFC 1945, which documented the "common usage" of the many HTTP/1.0 implementations found in the wild. Note that this was only an informational RFC: HTTP/1.0 as we know it is not a formal specification or an Internet standard!

Having said that, an example HTTP/1.0 request should look very familiar:

```
$> telnet website.org 80
Connected to xxx.xxx.xxx.xxx
GET /rfc/rfc1945.txt HTTP/1.0
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
Accept: */*

HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 137582
Expires: Thu, 01 Dec 1997 16:00:00 GMT
Last-Modified: Wed, 1 May 1996 12:45:26 GMT
Server: Apache 0.84

(plain-text response)
(connection closed)
```

1. Request line with HTTP version number, followed by request headers
2. Response status, followed by response headers

The preceding exchange is not an exhaustive list of HTTP/1.0 capabilities, but it does illustrate some of the key protocol changes:

- Request may consist of multiple newline separated header fields.
- Response object is prefixed with a response status line.
- Response object has its own set of newline separated header fields.
- Response object is not limited to hypertext.
- The connection between server and client is closed after every request.

Both the request and response headers were kept as ASCII encoded, but the response object itself could be of any type: an HTML file, a plain text file, an image, or any other content type. Hence, the "hypertext transfer" part of HTTP became a misnomer not long after its introduction. In reality, HTTP has quickly evolved to become a hypermedia transport, but the original name stuck.

In addition to media type negotiation, the RFC also documented a number of other commonly implemented capabilities: content encoding, character set support, multi-part types, authorization, caching, proxy behaviors, date formats, and more.

Almost every server on the Web today can and will still speak HTTP/1.0. Except that, by now, you should know better! Requiring a new TCP connection per request imposes a significant performance penalty on HTTP/1.0; see Three-Way Handshake, followed by Slow-Start.

HTTP/1.1: Internet Standard

The work on turning HTTP into an official IETF Internet standard proceeded in parallel with the documentation effort around HTTP/1.0 and happened over a period of roughly four years: between 1995 and 1999. In fact, the first official HTTP/1.1 standard is defined in RFC 2068, which was officially released in January 1997, roughly six months after the publication of HTTP/1.0. Then, two and a half years later, in June of 1999, a number of improvements and updates were incorporated into the standard and were released as RFC 2616.

The HTTP/1.1 standard resolved a lot of the protocol ambiguities found in earlier versions and introduced a number of critical performance optimizations: keepalive connections, chunked encoding transfers, byte-range requests, additional caching mechanisms, transfer encodings, and request pipelining.

With these capabilities in place, we can now inspect a typical HTTP/1.1 session as performed by any modern HTTP browser and client:

```
$> telnet website.org 80
Connected to xxx.xxx.xxx.xxx

GET /index.html HTTP/1.1
Host: website.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
10_7_4)... (snip)
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;
q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: __qca=P0-800083390... (snip)

HTTP/1.1 200 OK
Server: nginx/1.0.11
Connection: keep-alive
Content-Type: text/html; charset=utf-8
Via: HTTP/1.1 GWA
Date: Wed, 25 Jul 2012 20:23:35 GMT
Expires: Wed, 25 Jul 2012 20:23:35 GMT
Cache-Control: max-age=0, no-cache
Transfer-Encoding: chunked

100
<!doctype html>
(snip)
100
(snip)
0

GET /favicon.ico HTTP/1.1
Host: www.website.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
10_7_4)... (snip)
Accept: */*
Referer: http://website.org/
Connection: close
```

Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,;q=0.3*
Cookie: __qca=P0-800083390... (snip)

HTTP/1.1 200 OK
Server: nginx/1.0.11
Content-Type: image/x-icon
Content-Length: 3638
Connection: close
Last-Modified: Thu, 19 Jul 2012 17:51:44 GMT
Cache-Control: max-age=315360000
Accept-Ranges: bytes
Via: HTTP/1.1 GWA
Date: Sat, 21 Jul 2012 21:35:22 GMT
Expires: Thu, 31 Dec 2037 23:55:55 GMT
Etag: W/PSA-GAu26oXbDi
(icon data)
(connection closed)

1. Request for HTML file, with encoding, charset, and cookie metadata
2. Chunked response for original HTML request
3. Number of octets in the chunk expressed as an ASCII hexadecimal number (256 bytes)
4. End of chunked stream response
5. Request for an icon file made on same TCP connection
6. Inform server that the connection will not be reused
7. Icon response, followed by connection close

Phew, there is a lot going on in there! The first and most obvious difference is that we have two object requests, one for an HTML page and one for an image, both delivered over a single connection. This is connection keepalive in action, which allows us to reuse the existing TCP connection for multiple requests to the same host and deliver a much faster end-user experience; see Optimizing for TCP.

To terminate the persistent connection, notice that the second client request sends an explicit close token to the server via the Connection header. Similarly, the server can notify the client of the intent to close the current TCP connection once the response is transferred. Technically, either side can terminate the TCP connection without such signal at any point, but clients and servers should provide it whenever possible to enable better connection reuse strategies on both sides.

HTTP/1.1 changed the semantics of the HTTP protocol to use connection keepalive by default. Meaning, unless told otherwise (via Connection: close header), the server should keep the connection open by default.

However, this same functionality was also backported to HTTP/1.0 and enabled via the Connection: Keep-Alive header. Hence, if you are using HTTP/1.1, technically you don't need the Connection: Keep-Alive header, but many clients choose to provide it nonetheless.

Additionally, the HTTP/1.1 protocol added content, encoding, character set, and even language negotiation, transfer encoding, caching directives, client cookies, plus a dozen other capabilities that can be negotiated on each request.

We are not going to dwell on the semantics of every HTTP/1.1 feature. This is a subject for a dedicated book, and many great ones have been written already. Instead, the previous example serves as a good illustration of both the quick progress and evolution of HTTP, as well as the intricate and complicated dance of every client-server exchange. There is a lot going on in there!

For a good reference on all the inner workings of the HTTP protocol, check out O'Reilly's HTTP: The Definitive Guide by David Gourley and Brian Totty.

HTTP/2: Improving Transport Performance

Since its publication, RFC 2616 has served as a foundation for the unprecedented growth of the Internet: billions of devices of all shapes and sizes, from desktop computers to the tiny web devices in our pockets, speak HTTP every day to deliver news, video, and millions of other web applications we have all come to depend on in our lives.

What began as a simple, one-line protocol for retrieving hypertext quickly evolved into a generic hypermedia transport, and now a decade later can be used to power just about any use case you can imagine. Both the ubiquity of servers that can speak the protocol and the wide availability of clients to consume it means that many applications are now designed and deployed exclusively on top of HTTP.

Need a protocol to control your coffee pot? RFC 2324 has you covered with the Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0)—originally an April Fools' Day joke by IETF, and increasingly anything but a joke in our new hyper-connected world.

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol that can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

This article **provides insufficient context for those unfamiliar with the subject**. Please help improve the article by providing more context for the reader. *(November 2019) (Learn how and when to remove this template message)*

HTTP/3	
International standard	Hypertext Transfer Protocol Version 3 (HTTP/3) (draft)
Developed by	IETF
Introduced	Internet-Draft as of December 2019

HTTP/3 is the upcoming third major version of the Hypertext Transfer Protocol used to exchange information on the World Wide Web, succeeding HTTP/2. HTTP/3 is a draft based on a previous RFC draft, then named "Hypertext Transfer Protocol (HTTP) over QUIC".[3] QUIC is a transport layer network protocol developed initially by Google where user space congestion control is used over the User Datagram Protocol (UDP).

On 28 October 2018 in a mailing list discussion, Mark Nottingham, Chair of the IETF HTTP and QUIC Working Groups, made the official request to rename HTTP-over-QUIC as HTTP/3 to "clearly identify it as another binding of HTTP semantics to the wire protocol ... so people understand its separation from QUIC" and to pass its development from the QUIC Working Group to the HTTP Working Group after finalizing and publishing the draft. Nottingham's proposal was accepted by fellow IETF a few days later in November 2018.

Support for HTTP/3 was added to Chrome (Canary build) in September 2019, and while HTTP/3 is not yet on by default in any browser, by 2020 HTTP/3 has non-default support in stable versions of Chrome and Firefox and can be enabled.

Contents

Implementations
Browser

Browser	**Version implemented**	**Date**
Chrome	Stable build (79)	December 2019
Firefox	Stable build (72.0.1)	January 2020

Libraries
Open source libraries that implement client or server logic for QUIC and HTTP/3 are available.

Name	**Programming language**	**Company**	**Repository**
quiche	Rust	Cloudflare	https://github.com/cloudflare/quiche
neqo	Rust	Mozilla	https://github.com/mozilla/neqo
proxygen	C++	Facebook	https://github.com/facebook/proxygen#quic-and-http3
	C++	Google	https://github.com/chromium/chromium/tree/master/net/quic
lsquic	C	LiteSpeed	https://github.com/litespeedtech/lsquic
Flupke	Java		https://bitbucket.org/pjtr/flupke
h2o	C		https://github.com/h2o/h2o
libcurl	C		https://github.com/curl/curl
aioquic	Python		https://github.com/aiortc/aioquic
quic-go	Go		https://github.com/lucas-clemente/quic-go

Cloudflare's quiche library can be used as a patch to nginx. Support for HTTP/3 is slated for the 1.17 release of nginx.

There are a number of libraries that implement an older draft of the protocol or Google's versions of QUIC (e.g. Q046 used in Chrome 76), such as nghttp3.

HTTP Version and Release Year

Year	HTTP Version
1991	0.9
1996	1.0
199	1.1
2015	2.0
2018	3.0

II. Basic of HTTP

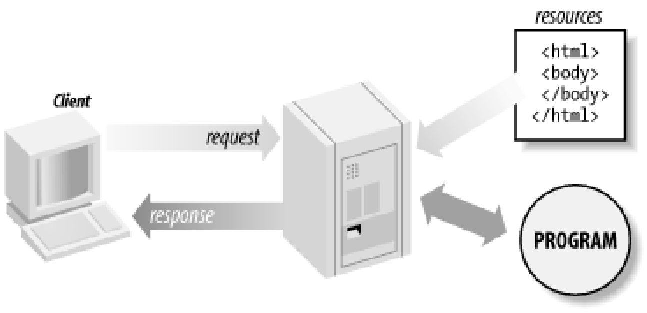
Hypertext Transfer Protocol (HTTP) is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes. HTTP follows a classical client-server model, with a client opening a connection to make a request, then waiting until it receives a response. HTTP is a stateless protocol, meaning that the server does not keep any data (state) between two requests. Though often based on a TCP/IP layer, it can be used on any reliable transport layer, that is, a protocol that doesn't lose messages silently like UDP does. RUDP — the reliable update of UDP — is a suitable alternative.

III. HTTP Request/Response

The data communication starts with a request sent from a client and ends with the response received from a web server.

- A website URL starting with “http://” is entered in a web browser from a computer (client). The browser can be a Chrome, Firefox, Edge, Safari, Opera or anything else.
- Browser sends a request sent to the web server that hosts the website.
- The web server then returns a response as a HTML page or any other document format to the browser.
- Browser displays the response from the server to the user.

The symbolic representation of a HTTP communication process is shown in the below picture:



(Fig. HTTP Request and Response)

The web browser is called as a User Agent and other example of a user agent is the crawlers of search engines like Googlebot.

A. HTTP Request

HTTP requests are messages sent by the client to initiate an action on the server. Their start-line contain three elements:

An HTTP method, a verb (like GET, PUT or POST) or a noun (like HEAD or OPTIONS), that describes the action to be performed. For example, GET indicates that a resource should be fetched or POST means that data is pushed to the server (creating or modifying a resource, or generating a temporary document to send back).

The request target, usually a URL, or the absolute path of the protocol, port, and domain are usually characterized by the request context. The format of this request target varies between different HTTP methods. It can be

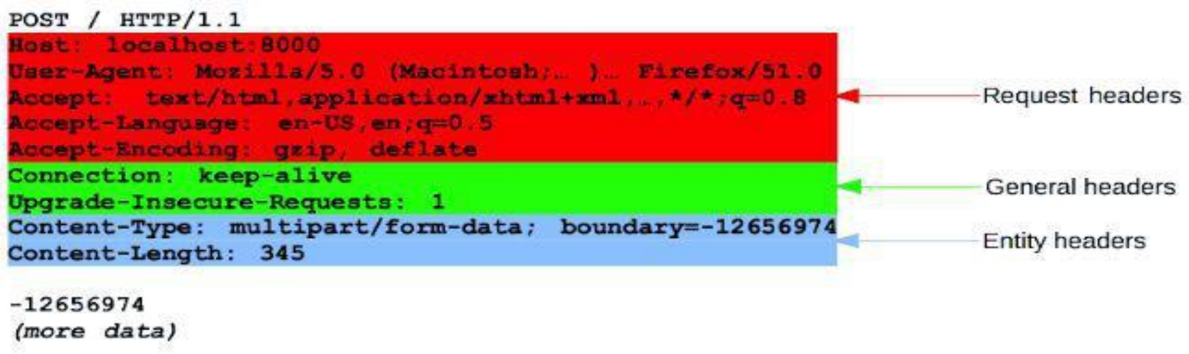
- An absolute path, ultimately followed by a '?' and query string. This is the most common form, known as the origin form, and is used with GET, POST, HEAD, and OPTIONS methods.Headers (Example – Content-Type: html)
- A complete URL, known as the absolute form, is mostly used with GET when connected to a proxy.
- The authority component of a URL, consisting of the domain name and optionally the port (prefixed by a ':'), is called the authority form. It is only used with CONNECT when setting up an HTTP tunnel.
- The asterisk form, a simple asterisk ('*') is used with OPTIONS, representing the server as a whole.
- The HTTP version, which defines the structure of the remaining message, acting as an indicator of the expected version to use for the response.

Header

HTTP headers from a request follow the same basic structure of an HTTP header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the header. The whole header, including the value, consist of one single line, which can be quite long.

There are numerous request headers available. They can be divided into several groups:

- General headers, like Via, apply to the message as a whole.
- Request headers like User-Agent, Accept-Type, modify the request by specifying it further (like Accept-Language), by giving context (like Referer), or by conditionally restricting it (like If-None).
- Entity headers, like Content-Length which apply to the body of the request. Obviously, there is no


```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;... )... Firefox/51.0
Accept: text/html,application/xhtml+xml,...,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

such header transmitted if there is no body in the request.

B. HTTP Response

The start line of an HTTP response, called the status line, contains the following information:

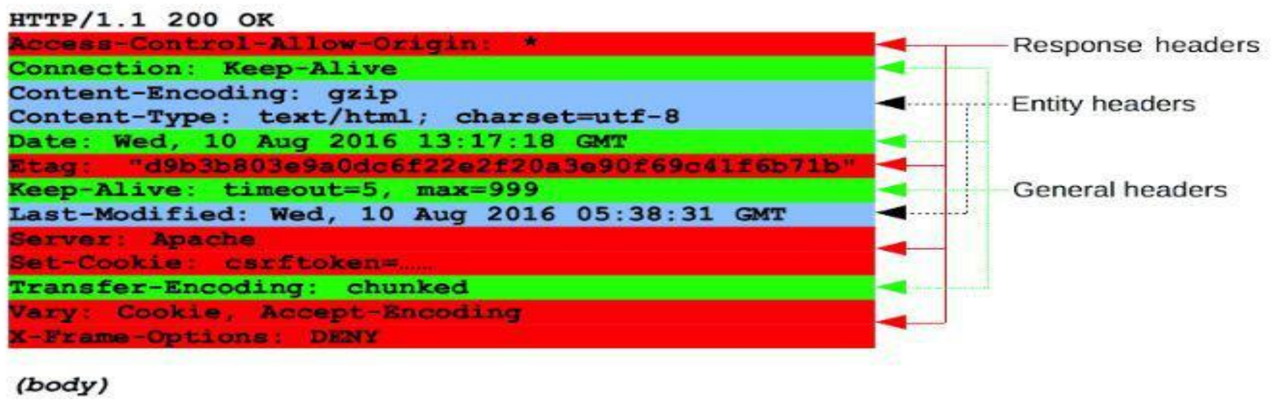
- The protocol version, usually HTTP/1.1.
- A status code indicating success or failure of the request. Common status codes are 200, 404, or 302
- A status text. A brief, purely informational, textual description of the status code to help a human understand the HTTP message. A typical status line looks like: HTTP/1.1 404 Not Found.

Header

HTTP headers for responses follow the same structure as any other header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the type of the header. The whole header, including its value, presents as a single line.

There are numerous response headers available. These can be divided into several groups:

- General headers, like Via, apply to the whole message.
- Response headers, like Vary and Accept-Ranges, give additional information about the server which doesn't fit in the status line.
- Entity headers, like Content-Length, apply to the body of the response. Typically, no such headers are transmitted when there is no body in the response.



```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 10 Aug 2016 13:17:18 GMT
Etag: "d9b3b803e9a0dc6f22e2f20a3e90f69c41f6b71b"
Keep-Alive: timeout=5, max=999
Last-Modified: Wed, 10 Aug 2016 05:38:31 GMT
Server: Apache
Set-Cookie: csrftoken=......
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
X-Frame-Options: DENY

(body)
```

IV. HTTP Error Code

The Error-Code element in a server response, is a 3-digit integer where the first digit of the Error-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

S.N.	Code and Description
1	**1xx: Informational** It means the request has been received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all the registered status codes. Given below is a list of all the status codes.

Error Code Description

1xx: Informational

Message	Description
100 Continue	Only a part of the request has been received by the server, but as long as it has not been rejected, the client should continue with the request.
101 Switching Protocols	The server switches protocol.

2xx: Success

Message	Description
200 OK	The request is OK.
201 Created	The request is complete, and a new resource is created .
202 Accepted	The request is accepted for processing, but the processing is not complete.
203 Non-authoritative Information	The information in the entity header is from a local or third-party copy, not from the original server.
204 No Content	A status code and a header are given in the response, but there is no entity-body in the reply.
205 Reset Content	The browser should clear the form used for this transaction for additional input.

3xx: Redirection

Message	Description
300 Multiple Choices	A link list. The user can select a link and go to that location. Maximum five addresses .
301 Moved Permanently	The requested page has moved to a new url .
302 Found	The requested page has moved temporarily to a new url .
303 See Other	The requested page can be found under a different url .
304 Not Modified	This is the response code to an *If-Modified-Since* or *If-None-Match* header, where the URL has not been modified since the specified date.
305 Use Proxy	The requested URL must be accessed through the proxy mentioned in the *Location* header.
306 *Unused*	This code was used in a previous version. It is no longer used, but the code is reserved.

4xx: Client Error

Message	Description
400 Bad Request	The server did not understand the request.
401 Unauthorized	The requested page needs a username and a password.
402 Payment Required	*You can not use this code yet.*
403 Forbidden	Access is forbidden to the requested page.
404 Not Found	The server can not find the requested page.
405 Method Not Allowed	The method specified in the request is not allowed.
406 Not Acceptable	The server can only generate a response that is not accepted by the client.
407 Proxy Authentication Required	You must authenticate with a proxy server before this request can be served.
408 Request Timeout	The request took longer than the server was prepared to wait.
409 Conflict	The request could not be completed because of a conflict.
410 Gone	The requested page is no longer available .
411 Length Required	The "Content-Length" is not defined. The server will not accept the request without it .
412 Precondition Failed	The pre condition given in the request evaluated to false by the server.
413 Request Entity Too Large	The server will not accept the request, because the request entity is too large.
414 Request-url Too Long	The server will not accept the request, because the url is too long. Occurs when you convert a "post" request to a "get" request with a long query information .
415 Unsupported Media Type	The server will not accept the request, because the mediatype is not supported .

5xx: Server Error

Message	Description
500 Internal Server Error	The request was not completed. The server met an unexpected condition.
501 Not Implemented	The request was not completed. The server did not support the functionality required.
502 Bad Gateway	The request was not completed. The server received an invalid response from the upstream server.
503 Service Unavailable	The request was not completed. The server is temporarily overloading or down.

504 Gateway Timeout	The gateway has timed out.
505 HTTP Version Not Supported	The server does not support the "http protocol" version.

V. Conclusion

In this paper, we have summarized and outlined some basics of HTTP protocols, HTTP Requests/Response, HTTP message header, HTTP error code.

VI. References

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages
[2] https://www.oreilly.com/library/view/javaserver-pages-3rd/0596005636/ch02s01.html
[3] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[4] https://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf
[5] https://www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-01
[6] https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
[7] https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html
[8] https://www.javatpoint.com/computer-network-http
[9] https://www.tutorialspoint.com/http/http_overview.htm