Internet as defined the Federal Networking Council (1995) is the global information system that

1. is logically linked together by a globally unique address space based on the Internet Protocol or its subsequent or follow-ons
2. is able to support communications using the **Transmission Control Protocol/Internet Protocol (TCP/IP)** suite or subsequent extensions/follow-ons
3. provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein.

The internet is at once a world-wide broadcasting capability, a mechanism for information dissemination, and a medium for collaboration and interaction of individuals and their computers without regard for geographic location.

**4 Distinct Aspects of Internet History**

**Technological evolution that began with early research on packet switching and the ARPANET.**

**Operations and management aspect of a global and complex operational infrastructure.**

**Social aspect, which resulted in a broad community of the Internauts working together to create and evolve the technology.**

**Commercialization aspect, resulting in an extremely effective transition of research results into a broadly deployed and available information and infrastructure.**

Origin of the internet

**DARPA**

\*The first recorded description of the social interactions that could be enabled through networking was a **series of memos** written by J.C.R Licklider of MIT in August 1962 – “**Galactic Network**”.

Licklider envisioned a globally interconnected set of computers through which everyone could quickly access data and programs from any site.

\*Leonard Kleinrock at MIT published the first paper on packet switching theory in July 1961.

*Feasibility of communications using packets rather than circuits.*

*Make the computers talk together.*

\*Lawrence Roberts went to DARPA to develop the computer network concept quickly put together his plan for the **ARPANET** after realizing that the circuit-switched telephone system is inadequate and confirmed that there was a need for packet switching.

\*The word packet was adopted from the work at NPL and the proposed line speed to be used in the ARPANET design was upgraded from 2.4 kbps to 50 kbps

\*In August 1968, an RFQ was released by DARPA for the development of one of the key components, the packet switches called **Interface Message Processors (IMP’s).**

**The first nodes of the ARPANET**

\*Klenirock’s **Network Measurement Center at UCLA** (University of California, Los Angeles).

-the first IMP was installed and the first computer was connected.

\*Doug Engelbart’s project on **“Augmentation of Human Intellect” at Stanford Research Institute (SRI)** provided the second node.

-SRI supported the Network Information Center and included functions such as maintaining tables of host name to address mapping as well as a directory of the RFC’s.

*One montlater, the first host-to-host message was sent from Kleinrock’s laboratoryto SRI.*

**\***Two more nodes were added at **UC Santa Barbara** and **University in Utah.**

These last two nodes incorporated application visualization projects and investigation of methods for display of mathematical functions using storage displays to deal with the problem of refresh over the net and methods of 3-D representations over the net.

*By the end of 1969, four host computers into the initial ARPANET.*

\*In December 1970, the initial ARPANET host-to-host protocol was completed called the Network Control Protocol (NCP). Network users begin to develop applications

\*In October 1972, the demonstration of the ARPANET was conducted at the International Computer Communication Conference (ICCC). This was the first public demonstration of the ARPANET to the public. It was during this year that the initial “hot” application, electronic mail, was introduced.

**The World Wide Web**

\*In 1989, Sir Tim Berners-Lee, a British scientist at CERN (European Organization of Nuclear Research), invented the World Wide Web or simply the Web. The web was originally conceived for automatic information-sharing between scientists in universities and institutes around the world.

\*The first website at CERN – and in the world – was dedicated to the Web project itself and was hosted on Berners-Lee’s NeXT computer. The website described the features of the Web; how to access other people’s documents and how to set up your own server.

\*The NeXt machine is the original web server.

\*On 30 April 1993, CERN put the Web software in the public domain. CERN made the next release available with an open license, as a more sure way to maximize its dissemination.

\*The line-mode browser was launched in 1992, and was the first readily accessible browser for the Web

\*The three cornerstone of the web: HTTP, HTML, URI

**Web Architecture**

**\*Web Server** – serves web content. Web resources are html files

\*hosts web documents

**URL – addressing web resources**

\***HTTP –** sends process requests

\*primary protocol being used in web communication.

**HTTP** – Hyper Text Transfer Protocol

\*application layer communication protocol used to access resources (hypertext/hypermedia) on the WWW.

\*Top layer – uses the services of the lower layers.

\*Jointly developed by the W3C (World Wide Web Consortium) and IETF (Internet Engineering Task Force (IETF)

**HTTP/0.9** – original version (original protocol specs) defined in 1991

1. **Connection** – the client makes a TCP-IP connection to the host using the domain name or IP number, and the port number given in the address.

If the port is not specified, 80 is always assumed for HTTP.

The server accepts the connection.

*The request should be one TPDU, but the response may be many.*

1. **Request** - the client sends a document of request consisting of a line of ASCII characters terminated by a CR LF (carriage return, line feed) pair.

\*The request consists of the word “GET”, a space, the document address, omitting the “http: host and port parts when they are the coordinates just used to make the connection. (If a gateway is being used, then a full document address may be given specifying a different naming scheme).

\*The document address will consists of a single word. If any further words are found on the request line, they must either be ignored, or else treated according to the full HTTP spec.

\*The search functionality of the protocol lies in the ability of the addressing syntax to describe a search on a named index.

A search should only be requested by a client when the index document itself has been described as an index using the ISINDEX tag.

1. **Response** – the response to a simple GET request in hypertext mark-up language (HTML). This is a byte stream of ASCII characters.

Lines shall be delimited by an optional carriage return followed by a mandatory line feed character. The client should not assume that the carriage return will be present. Lines may be of any length. Well-behaved servers should restrict line length to 80 characters excluding the CR LF pair.

The format of the message is HTML –that is, trimmed SGML document. Note that this format allows for menus hit lists to be returned as hypertext. It also allows for plain ASCII text to be returned following the PLAINTEXT tag.

The message us terminated by the closing of the connection by the server.

Well-behaved clients will read the entire document as fast as possible. The client shall not wait for user action before reading the whole of the document. The server may impose a timeout of the order of 15 seconds inactivity.

Error response are supplies in human readable text in HTML syntax. There is no way to distinguish an error response from a satisfactory response except for the content of the text.

1. **Disconnection** – the TCP-IP connection is broken by the server when the whole document has been transferred.

The client may abort the transfer by breaking the connection before this, in which case the server shall not record any error condition.

The request is idempotent. The server need not store an information about the request after disconnection.

**HTTP/1.0**

The Hypertext Transfer Protocol is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. Practical Information System require more functionality than simple retrieval, including search, front-end update, and annotation. HTTP allows an open-ended set of methods to be used to indicate the purpose of a request. It builds on the discipline of reference provided by the **Uniform Resource Identifier (URI), as a location (URL) or a name (URN),** for indicating the resource on which a method is to be applied. Messages are passed in a format similar to that used by Internet Mail, and the Multipurpose Internet Mail Extensions (MIME).

HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet Protocols, such as SMTP, NNTP, FRP, Gopher, WAIS, allowing basic hypermedia access to resource available from diverse applications and simplifying the implementation of user agents.

**HTTP/1.1**

The Hypertext Transfer Protocol is an application-level protocol for distributed, collaborative, hypermedia information system. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through the 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.

The Hypertext Transfer Protocol is an application-level protocol distributed, collaborative, hypermedia information systems. The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet. HTTP/1,0, as defined by RFC 1945, improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta-information about the data transferred and modifiers on the request/response semantics. However, HTTP/1.0 does not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, or virtual hosts. In addition, the proliferation of incompletely-implemented applications calling themselves “HTP/1.0” has necessitated a protocol version change in order for two communicating applications to determine each other’s true capabilities.

HTTP/1.1 includes more stringent requirements that HTTP/1.0 in order to ensure reliable implementations of its features.

Practical information systems require more functionality than simple retrieval, including search, front-end update, and annotation. HTTP allows an open-ended set of methods and headers that indicate the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier, as a location URL and as a name URN, for indicating the resource to which a method is to be applied. Messages are passed in a format similar to that used by Internet mail as define by the Multipurpose Internet Mail Extensions MIME.

HTTO is also used as a generic protocol for communication between user agents and proxies/gateways to other internet systems, including those supported by the SMTP, FTP, Gopher, NNTP, and WAIS protocols. In this way, HTTP allows basic hypermedia access to resources available from diverse applications.

The HTTP protocol us a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the message’s protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content.

**HTTP/2**

HTTP/2 is a major revision of the HTTP network protocol used by the World Wide Web. It was developed from the earlier experimental SPDY protocol, originally developed by Google. HTTP/2 was developed by the Hypertext transfer Protocol working group httpbis (where bis means ‘second”) of the IETF.

The standardization effort was supported by Chrome, Opera, Firefox, IE 11, Safar, Amazon Silk, and Edge browsers.

The proposed changes do not require any changes to how existing web applications work, but new applications can take advantage go new features for increased speed.

HTTP/2 leaves most of HTTP 1.1’s high-level syntax, such as methods, status codes, header fields, and URIs, the same. The element that is modified is how the data is framed and transported between the client and the server.

HTTP/2 allows the server to “push” content, that is, to respond with data for more queries than the client requested. This allows the server to supply data it knows a web browser will need to render the web page, without waiting for the browser to examine the first response, and without the overhead of an additional request cycle.

Additional performance improvements in the first draft of HTTP/2 come from multiplexing of requests and responses to avoid the head-of-line blocking problem in HTTP 1 (even when HTTP pipelining is used), header compression, and prioritization of requests.

**FUNDAMENTAL CHARACTERISTICS OF HTTP**

* TCP/IP – is the basic communication language protocol of the internet.
  + TCP/IP is a two-layer program. The higher layer, TCP, manages the assembling of a message or file into smaller packets that are transmitted over the Internet and received by a TCP later that reassembles the packets into the original message. The lower layer, Internet Protocol, handles the address part of each packer so that it gets to the right destination.
  + The TCP/IP and the other applications that uses it are collectively said to be stateless because each client request is considered a new request unrelated to any previous one.
  + HTTP is a protocol that utilizes TCP to transfer its information between computers. The client makes and HTTP request to the Web server using a Web browser, and the web server send the requested information to the client.
* Based on client-server architecture
* Request – response
* Stateless protocol – send and forget protocol.

**FUNCTIONALITIES**

* Cache control

**Locality of reference principle** is the term for the phenomenon in which the same values, or related storage locations, are frequently accessed, depending on the memory access pattern.

* Content/transfer coding
* Content-negotiation (agreement between client and server)
* Client-Server negotiation
* Persistent Connection (HTTP/1.1)
* Request Pipelining (HTTP/1.1)
* Authentication and authorization

**HTTP RESOURCE ADDRESSING**

**URI** (Uniform resource identifier) is used to identify http resources

* URN – Uniform resource name identifies a resource by name in a particular namespace and is used to talk about a resource without implying its location or how to access it.
* URL – Uniform resource location specifies the means of acting upon or obtaining the representation of it.

1. Scheme
2. Authority

User information and host

1. Path
2. Query
3. Fragment identifier

**HTTP Request Message**

**A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use.**

1. Request line

**Method, Request URI, http protocol version**

1. Message Headers

Request Header Fields (general, request and entity headers)

1. Empty line
2. Message body (optional)

**HTTP Response Message**

**After receiving and interpreting a request message, a server responds with an HTTP response message.**

1. Status line

**Http protocol version, status code, reason phrase (description of the status code)**

The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user.

1xx: Informational - Request received, continuing process

- 1xx: Informational - Request received, continuing process

- 2xx: Success - The action was successfully received, understood, and accepted

- 3xx: Redirection - Further action must be taken in order to complete the request

- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled

- 5xx: Server Error - The server failed to fulfill an apparently valid request

1. Message Headers

Response Header Fields (general, response, entity headers)

1. Empty line
2. Message Body (optional)

**HTTP Request methods**

1. **Standard Methods**

**GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE, CONNECT**

**GET**

The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data

**HEAD**

Same as GET, but transfers the status line and header section only.

**POST**

A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

**PUT**

Replaces all current representations of the target resource with the uploaded content.

**DELETE**

Removes all current representations of the target resource given by a URI.

**CONNECT**

Establishes a tunnel to the server identified by a given URI.

**OPTIONS**

Describes the communication options for the target resource.

**TRACE**

Performs a message loop-back test along the path to the target resource.

**STATUS CODES AND REASON-PHRASE**

**Informational 1xx**

**This class of status code indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line.** There are no required headers for this class of status code. Since HTTP/1.0 did not define any 1xx status codes, servers MUST NOT send a 1xx response to an HTTP/1.0 client except under experimental conditions.

A client MUST be prepared to accept one or more 1xx status responses prior to a regular response, even if the client does not expect a 100 (Continue) status message. Unexpected 1xx status responses MAY be ignored by a user agent.

Proxies MUST forward 1xx responses, unless the connection between the proxy and its client has been closed, or unless the proxy itself requested the generation of the 1xx response. (For example, if a proxy adds a "Expect: 100-continue" field when it forwards a request, then it need not forward the corresponding 100 (Continue) response(s).)

**100 Continue**

The client SHOULD continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server MUST send a final response after the request has been completed. See section 8.2.3 for detailed discussion of the use and handling of this status code.

**101 Switching Protocols**

The server understands and is willing to comply with the client's request, via the Upgrade message header field (section 14.42), for a change in the application protocol being used on this connection. The server will switch protocols to those defined by the response's Upgrade header field immediately after the empty line which terminates the 101 response.

The protocol SHOULD be switched only when it is advantageous to do so. For example, switching to a newer version of HTTP is advantageous over older versions, and switching to a real-time, synchronous protocol might be advantageous when delivering resources that use such features.

**Successful 2xx**

**This class of status code indicates that the client's request was successfully received, understood, and accepted.**

**200 OK**

The request has succeeded. The information returned with the response is dependent on the method used in the request, for example:

GET, HEAD, POST, and TRACE.

**201 Created**

The request has been fulfilled and resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the entity of the response, with the most specific URI for the resource given by a Location header field. The response SHOULD include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. The origin server MUST create the resource before returning the 201 status code. If the action cannot be carried out immediately, the server SHOULD respond with 202 (Accepted) response instead.

A 201 response MAY contain an ETag response header field indicating the current value of the entity tag for the requested variant just created, see section 14.19.

**202 Accepted**

The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. There is no facility for re-sending a status code from an asynchronous operation such as this.

The 202 response is intentionally non-committal. Its purpose is to allow a server to accept a request for some other process (perhaps a batch-oriented process that is only run once per day) without requiring that the user agent's connection to the server persist until the process is completed. The entity returned with this response SHOULD include an indication of the request's current status and either a pointer to a status monitor or some estimate of when the user can expect the request to be fulfilled.

**203 Non-Authoritative Information**

The returned metainformation in the entity-header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy. The set presented MAY be a subset or superset of the original version. For example, including local annotation information about the resource might result in a superset of the metainformation known by the origin server. Use of this response code is not required and is only appropriate when the response would otherwise be 200 (OK).

**204 No Content**

The server has fulfilled the request but does not need to return an entity-body, and might want to return updated metainformation. The response MAY include new or updated metainformation in the form of entity-headers, which if present SHOULD be associated with the requested variant.

If the client is a user agent, it SHOULD NOT change its document view from that which caused the request to be sent. This response is primarily intended to allow input for actions to take place without causing a change to the user agent's active document view, although any new or updated metainformation SHOULD be applied to the document currently in the user agent's active view.

The 204 response MUST NOT include a message-body, and thus is always terminated by the first empty line after the header fields.

**205 Reset Content**

The server has fulfilled the request and the user agent SHOULD reset the document view which caused the request to be sent. This response is primarily intended to allow input for actions to take place via user input, followed by a clearing of the form in which the input is given so that the user can easily initiate another input action. The response MUST NOT include an entity.

**206 Partial Content**

The server has fulfilled the partial GET request for the resource. The request MUST have included a Range header field (section 14.35) indicating the desired range, and MAY have included an If-Range header field (section 14.27) to make the request conditional.

If the 206 response is the result of an If-Range request that used a strong cache validator (see section 13.3.3), the response SHOULD NOT include other entity-headers. If the response is the result of an If-Range request that used a weak validator, the response MUST NOT include other entity-headers; this prevents inconsistencies between cached entity-bodies and updated headers. Otherwise, the response MUST include all of the entity-headers that would have been returned with a 200 (OK) response to the same request.

A cache MUST NOT combine a 206 response with other previously cached content if the ETag or Last-Modified headers do not match exactly, see 13.5.4. A cache that does not support the Range and Content-Range headers MUST NOT cache 206 (Partial) responses.

**Redirection 3xx**

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required MAY be carried out by the user agent without interaction with the user if and only if the method used in the second request is GET or HEAD. A client SHOULD detect infinite redirection loops, since such loops generate network traffic for each redirection.

**300 Multiple Choices**

The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent- driven negotiation information (section 12) is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.

Unless it was a HEAD request, the response SHOULD include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content- Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed automatically. However, this specification does not define any standard for such automatic selection.

If the server has a preferred choice of representation, it SHOULD include the specific URI for that representation in the Location field; user agents MAY use the Location field value for automatic redirection. This response is cacheable unless indicated otherwise.

**301 Moved Permanently**

The requested resource has been assigned a new permanent URI and any future references to this resource SHOULD use one of the returned URIs. Clients with link editing capabilities ought to automatically re-link references to the Request-URI to one or more of the new references returned by the server, where possible. This response is cacheable unless indicated otherwise.

The new permanent URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 301 status code is received in response to a request other than GET or HEAD, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

**302 Found**

The requested resource resides temporarily under a different URI. Since the redirection might be altered on occasion, the client SHOULD continue to use the Request-URI for future requests. This response is only cacheable if indicated by a Cache-Control or Expires header field.

The temporary URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 302 status code is received in response to a request other than GET or HEAD, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

**303 See Other**

The response to the request can be found under a different URI and SHOULD be retrieved using a GET method on that resource. This method exists primarily to allow the output of a POST-activated script to redirect the user agent to a selected resource. The new URI is not a substitute reference for the originally requested resource. The 303 response MUST NOT be cached, but the response to the second (redirected) request might be cacheable.

The different URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

**304 Not Modified**

If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server SHOULD respond with this status code. The 304 response MUST NOT contain a message-body, and thus is always terminated by the first empty line after the header fields.

If the conditional GET used a strong cache validator (see section 13.3.3), the response SHOULD NOT include other entity-headers. Otherwise (i.e., the conditional GET used a weak validator), the response MUST NOT include other entity-headers; this prevents inconsistencies between cached entity-bodies and updated headers.

If a 304 response indicates an entity not currently cached, then the cache MUST disregard the response and repeat the request without the conditional. If a cache uses a received 304 response to update a cache entry, the cache MUST update the entry to reflect any new field values given in the response.

**305 Use Proxy**

The requested resource MUST be accessed through the proxy given by the Location field. The Location field gives the URI of the proxy. The recipient is expected to repeat this single request via the proxy. 305 responses MUST only be generated by origin servers.

**306 (Unused)**

The 306 status code was used in a previous version of the specification, is no longer used, and the code is reserved.

**307 Temporary Redirect**

The requested resource resides temporarily under a different URI. Since the redirection MAY be altered on occasion, the client SHOULD continue to use the Request-URI for future requests. This response is only cacheable if indicated by a Cache-Control or Expires header field.

The temporary URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s) , since many pre-HTTP/1.1 user agents do not understand the 307 status. Therefore, the note SHOULD contain the information necessary for a user to repeat the original request on the new URI.

If the 307 status code is received in response to a request other than GET or HEAD, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

**Client Error 4xx**

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents SHOULD display any included entity to the user.

If the client is sending data, a server implementation using TCP SHOULD be careful to ensure that the client acknowledges receipt of the packet(s) containing the response, before the server closes the input connection. If the client continues sending data to the server after the close, the server's TCP stack will send a reset packet to the client, which may erase the client's unacknowledged input buffers before they can be read and interpreted by the HTTP application.

**400 Bad Request**

The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications.

**401 Unauthorized**

The request requires user authentication. The response MUST include a WWW-Authenticate header field (section 14.47) containing a challenge applicable to the requested resource. The client MAY repeat the request with a suitable Authorization header field (section 14.8). If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials. If the 401 response contains the same challenge as the prior response, and the user agent has already attempted authentication at least once, then the user SHOULD be presented the entity that was given in the response, since that entity might include relevant diagnostic information. HTTP access authentication is explained in "HTTP Authentication: Basic and Digest Access Authentication" [43].

**402 Payment Required**

This code is reserved for future use.

**403 Forbidden**

The server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated. If the request method was not HEAD and the server wishes to make public why the request has not been fulfilled, it SHOULD describe the reason for the refusal in the entity. If the server does not wish to make this information available to the client, the status code 404 (Not Found) can be used instead.

**404 Not Found**

The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent. The 410 (Gone) status code SHOULD be used if the server knows, through some internally configurable mechanism, that an old resource is permanently unavailable and has no forwarding address. This status code is commonly used when the server does not wish to reveal exactly why the request has been refused, or when no other response is applicable.

**405 Method Not Allowed**

The method specified in the Request-Line is not allowed for the resource identified by the Request-URI. The response MUST include an Allow header containing a list of valid methods for the requested resource.

**406 Not Acceptable**

The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.

Unless it was a HEAD request, the response SHOULD include an entity containing a list of available entity characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed automatically. However, this specification does not define any standard for such automatic selection.

If the response could be unacceptable, a user agent SHOULD temporarily stop receipt of more data and query the user for a decision on further actions.

**407 Proxy Authentication Required**

This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy. The proxy MUST return a Proxy-Authenticate header field (section 14.33) containing a challenge applicable to the proxy for the requested resource. The client MAY repeat the request with a suitable Proxy-Authorization header field (section 14.34). HTTP access authentication is explained in "HTTP Authentication: Basic and Digest Access Authentication" [43].

**408 Request Timeout**

The client did not produce a request within the time that the server was prepared to wait. The client MAY repeat the request without modifications at any later time.

**409 Conflict**

The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body SHOULD include enough information for the user to recognize the source of the conflict. Ideally, the response entity would include enough information for the user or user agent to fix the problem; however, that might not be possible and is not required.

Conflicts are most likely to occur in response to a PUT request. For example, if versioning were being used and the entity being PUT included changes to a resource which conflict with those made by an earlier (third-party) request, the server might use the 409 response to indicate that it can't complete the request. In this case, the response entity would likely contain a list of the differences between the two versions in a format defined by the response Content-Type.

**410 Gone**

The requested resource is no longer available at the server and no forwarding address is known. This condition is expected to be considered permanent. Clients with link editing capabilities SHOULD delete references to the Request-URI after user approval. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) SHOULD be used instead. This response is cacheable unless indicated otherwise.

The 410 response is primarily intended to assist the task of web maintenance by notifying the recipient that the resource is intentionally unavailable and that the server owners desire that remote links to that resource be removed. Such an event is common for limited-time, promotional services and for resources belonging to individuals no longer working at the server's site. It is not necessary to mark all permanently unavailable resources as "gone" or to keep the mark for any length of time -- that is left to the discretion of the server owner.

**411 Length Required**

The server refuses to accept the request without a defined Content- Length. The client MAY repeat the request if it adds a valid Content-Length header field containing the length of the message-body in the request message.

**412 Precondition Failed**

The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server. This response code allows the client to place preconditions on the current resource metainformation (header field data) and thus prevent the requested method from being applied to a resource other than the one intended.

**413 Request Entity Too Large**

The server is refusing to process a request because the request entity is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request.

If the condition is temporary, the server SHOULD include a Retry- After header field to indicate that it is temporary and after what time the client MAY try again.

**414 Request-URI Too Long**

The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret. This rare condition is only likely to occur when a client has improperly converted a POST request to a GET request with long query information, when the client has descended into a URI "black hole" of redirection (e.g., a redirected URI prefix that points to a suffix of itself), or when the server is under attack by a client attempting to exploit security holes present in some servers using fixed-length buffers for reading or manipulating the Request-URI.

**415 Unsupported Media Type**

The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

**416 Requested Range Not Satisfiable**

A server SHOULD return a response with this status code if a request included a Range request-header field (section 14.35), and none of the range-specifier values in this field overlap the current extent of the selected resource, and the request did not include an If-Range request-header field. (For byte-ranges, this means that the first- byte-pos of all of the byte-range-spec values were greater than the current length of the selected resource.)

When this status code is returned for a byte-range request, the response SHOULD include a Content-Range entity-header field specifying the current length of the selected resource (see section 14.16). This response MUST NOT use the multipart/byteranges content- type.

**417 Expectation Failed**

The expectation given in an Expect request-header field (see section 14.20) could not be met by this server, or, if the server is a proxy, the server has unambiguous evidence that the request could not be met by the next-hop server.

**Server Error 5xx**

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included entity to the user. These response codes are applicable to any request method.

**500 Internal Server Error**

The server encountered an unexpected condition which prevented it from fulfilling the request.

**501 Not Implemented**

The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.

**502 Bad Gateway**

The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.

**503 Service Unavailable**

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay MAY be indicated in a Retry-After header. If no Retry-After is given, the client SHOULD handle the response as it would for a 500 response.

**504 Gateway Timeout**

The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI (e.g. HTTP, FTP, LDAP) or some other auxiliary server (e.g. DNS) it needed to access in attempting to complete the request.

The server does not support, or refuses to support, the HTTP protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, as described in section 3.1, other than with this error message. The response SHOULD contain an entity describing why that version is not supported and what other protocols are supported by that server.