Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. 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 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 chapter explains HTTP by example and considers also the Uniform Resource Identifiers (URIs, aka URLs) as short strings that identify resources in the Web.

1. The Context of the Hypertext Transfer Protocol

During the early Internet days, information was gathered using FTP (File Transfer Protocol) and Telnet was used for sessions which proved to be a user unfriendly way of interaction or searching information. However, since the majority of people on the net were computer literate, this did not prove to be too great a problem. Things became quite different with the Web: Tim Berners-Lee proposed the first version of HTML (Hypertext Markup Language) in 1989, the first server and browser prototypes came into being between 1990 and 1992. From then on, the Web caused an exponential growth of the Internet: In June 1991 about 500,000 nodes were recorded world-wide. As of June 1998 there were about 35 millions nodes recorded and the monthly growth rate was about 1 million world-wide. Thus, the monthly growth rate far exceeded the total number of nodes only 7 years ago.

The main reasons for the exponential growth rate the Web caused were the easy to use point-and-click graphical user interface and the integration of all relevant services into one tool – the Web browser. As more and more Web servers appeared all over the world, the number of users accessing them also grew.

Mainly three concepts are making up the Web and are responsible for its success: The Hypertext Markup Language (HTML), the Uniform Resource Locator (URL), and the Hypertext Transfer Protocol (HTTP). As HTML is described in a separate chapter, we shortly introduce the concept of URLs before focusing on HTTP.

2. Uniform Resource Locator and Identifier

Titled “Naming and Addressing” the W3C defines (http://www.w3.org/Addressing/) the more general term “Uniform Resource Identifier” as “the generic set of all names/addresses that are short strings that refer to resources”:

The Web is an information space. Human beings have a lot of mental machinery for manipulating, imagining, and finding their way in spaces. URIs are the points in that space. Unlike web data formats, where HTML is an important one, but not the only one, and web protocols, where HTTP has a similar status, there is only one Web naming/addressing technology: URIs.

Uniform Resource Identifiers (URIs, aka URLs) are short strings that identify resources in the web: documents, images, downloadable files, services, electronic mailboxes, and

---

1 Although the term hypertext was initially coined in the 1960's, its popularity first grew when the Web was invented.
other resources. They make resources available under a variety of naming schemes and access methods such as HTTP, FTP, and Internet mail addressable in the same simple way. They reduce the tedium of "log in to this server, then issue this magic command ..." down to a single click.

It is an extensible technology: there are a number of existing addressing schemes, and more may be incorporated over time.

Another important term is the “Uniform Resource Name” URN:

1. An URI that has an institutional commitment to persistence, availability, etc. Note that this sort of URI may also be a URL. See, for example, PURLs.
2. A particular scheme, urn:, specified by RFC 2141 and related documents, intended to serve as persistent, location-independent, resource identifiers.

Consequently, the Uniform Resource Locator URL becomes an informal term (no longer used in technical specifications) associated with popular URI schemes for specifying Internet resources: http, ftp, mailto, etc.

The “generic URI” syntax (RFC 2396) consists of a sequence of four main components: <scheme>://<authority><path>?<query>

- The scheme covers all major Internet protocols, typically http or https (HTTP over Secure Sockets Layer SSL), but also file (Host-specific file names), ftp (File Transfer Protocol), ldap (Lightweight Directory Access Protocol), mailto (Electronic mail address), news (USENET news), nfs (Netfork File System), nntp (Network News Transfer Protocol), pop (Post Office Protocol v3), sip (Session Initiation Protocol), telnet (Reference to interactive sessions). As of Oct 2002, the W3C lists 85 schemes, with only some of them being registered (http://www.w3.org/Addressing/schemes). RFCs 2717 and 2718 provide registration procedures for scheme names and guidelines for new schemes, respectively.
- For schemes that involve the direct use of an IP-based protocol to a specified server on the Internet, the authority looks like <userinfo>@<host>:<port>, where <userinfo> may consist of a user name and, optionally, scheme-specific information about how to gain authorization to access the server. The parts <userinfo>@ and :<port> may be omitted. The host is a domain name of a network host (RFCs 1034, 1123), or its IPv4 address as a set of four decimal digit groups separated by dots. The port is the network port number for the server. Most schemes designate protocols that have a default port number. Another port number may optionally be supplied, in decimal, separated from the host by a colon. If the port is omitted, the default port number (“well known port”) is assumed, e.g. 80 for HTTP or 443 for HTTPS.
- The path component contains data, specific to the authority, identifying the resource within the scope of that scheme and authority. In the case of HTTP it is either a file system path relative to the document root of the respective Web server or the path has a special, pre-configured meaning for the Web server (e.g. a server-side application).

---

2 This practice of using passwords within the URI is not recommended, because the passing of authentication information in clear text (such as URI) has proven to be a security risk in almost every case where it has been used.
• The query component is a string of information to be interpreted by the resource. In the case of HTML and HTTP this may be a set of name-value pairs resulting from a FORM with the GET method, a list of names separated by plus signs resulting from an ISINDEX query or a comma-separated pair of x- and y-pixel-coordinates resulting from an active server-side imagemap (ISMAP).

It is often the case that a group or “tree” of documents has been constructed to serve a common purpose; the vast majority of URI in these documents point to resources within the tree rather than outside of it. Similarly, documents located at a particular site are much more likely to refer to other resources at that site than to resources at remote sites.

Relative addressing of URI allows document trees to be partially independent of their location and access scheme. For instance, it is possible for a single set of hypertext documents to be simultaneously accessible and traversable via each of the file, http, and ftp schemes if the documents refer to each other using relative URI. Furthermore, such document trees can be moved, as a whole, without changing any of the relative references. Experience within the WWW has demonstrated that the ability to perform relative referencing is necessary for the long-term usability of embedded URI.

A URI reference may therefore be absolute or relative, and may have additional information attached in the form of a fragment identifier (following the hash character #). However, “the URI” that results from such a reference includes only the absolute URI after the fragment identifier (if any) is removed and after any relative URI is resolved to its absolute form.

The following examples show typical HTTP URIs:

http://some.where.edu/path/mydocument.html#fragment_id

This URI accesses the server some.where.edu via “well known” port 80 with the protocol HTTP. Considering a standard Web server configuration, this would result in the delivery of the document mydocument.html in the directory path relative to the document root of the Web server. The browser would then open the page and jump to the place in the document, that is marked with the appropriate anchor tag <a name="fragment_id">.

../stuff/other.html

Based on the previous URI, this relative URI would result in the following absolute URI:

http://some.where.edu/stuff/other.html

The following three URIs show possible usages of the query string: The first URI results from pressing the “submit” button of an HTML FORM, which is declared using the GET method. In this case the name/value pairs of the form inputs are described. The second URI results from an ISINDEX query and enumerates the arguments, and the third URI from a server-side active imagemap (ISMAP), which provides the coordinates in pixels from the upper left corner, where within the image the “click” took place.

http://some.where.edu:33333/app/program?a=1&b=2
http://some.where.edu:33333/app/search?arg1+arg2+arg3
http://some.where.edu:33333/app/imagemap?123,45
Usually, the resources in these cases are server-side applications prepared to process the incoming query string in order to generate the following HTML document.

3. Overall Operation of HTTP

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. 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 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. HTTP has been in use by the World-Wide Web global information initiative since 1990.

This definition and parts of further explanations are taken from RFC 2616, which defines the protocol referred to as “HTTP/1.1”, and is an update to RFC 2068. An overview of the RFCs and other documents relevant for HTTP is provided by the World Wide Web Consortium under http://www.w3.org/Protocols/.

The HTTP protocol is 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 a 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 meta-information, and possible entity-body content.

Most HTTP communication is initiated by a user agent and consists of a request to be applied to a resource on some origin server. In the simplest case, this may be accomplished via a single connection between the user agent and the origin server.

A more complicated situation occurs when one or more intermediaries are present in the request/response chain. There are three common forms of intermediary: proxy, gateway, and tunnel. A proxy is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or part of the message, and forwarding the reformatted request toward the server identified by the URI. A gateway is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying server’s protocol. A tunnel acts as a relay point between two connections without changing the messages; tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.

A request or response message that travels the whole chain will pass through several separate connections. This distinction is important because some HTTP communication options may apply only to the connection with the nearest, non-tunnel neighbor, only to the end-points of the chain, or to all connections along the chain. Moreover, each participant in a certain chain may be engaged in multiple, simultaneous communications.

Any party to the communication which is not acting as a tunnel may employ an internal cache for handling requests. The effect of a cache is that the request/response chain is shortened if one of the participants along the chain has a cached response applicable to that request. Not all responses are usefully cacheable, and some requests may contain modifiers which place special requirements on cache behavior.
In fact, there are a wide variety of architectures and configurations of caches and proxies currently being experimented with or deployed across the World Wide Web. These systems include national hierarchies of proxy caches to save transoceanic bandwidth, systems that broadcast or multicast cache entries, organizations that distribute subsets of cached data via CD-ROM, and so on. HTTP systems are used in corporate intranets over high-bandwidth links, and for access via PDAs with low-power radio links and intermittent connectivity. The goal of HTTP/1.1 is to support the wide diversity of configurations already deployed while introducing protocol constructs that meet the needs of those who build web applications that require high reliability and, failing that, at least reliable indications of failure.

HTTP communication usually takes place over TCP/IP connections. The default port is TCP 80 (RFC 1700), but other ports can be used. This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used; the mapping of the HTTP/1.1 request and response structures onto the transport data units of the protocol in question is outside the scope of this specification.

In HTTP/1.0, most implementations used a new connection for each request/response exchange. In HTTP/1.1, a connection may be used for one or more request/response exchanges, although connections may be closed for a variety of reasons. Therefore, HTTP should be treated as connection oriented but stateless protocol.

4. Protocol Parameters

The Hypertext Transfer Protocol uses various parameters, the following section explains the most important ones, details can be found in RFC 2616.

**HTTP Version**

HTTP request and response messages include an HTTP protocol version number. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further HTTP communication, rather than the features obtained via that communication. No change is made to the version number for the addition of message components which do not affect communication behavior or which only add to extensible field values. The HTTP version of an application is the highest HTTP version for which the application is at least conditionally compliant. Proxy and gateway applications need to be careful when forwarding messages in protocol versions different from that of the application, because converting between versions of HTTP may involve modification of header fields required or forbidden by the versions involved. See RFC 2145 for a fuller explanation.

**URI**

As far as HTTP is concerned, Uniform Resource Identifiers are simply formatted strings which identify – via name, location, or any other characteristic – a resource.

**Date/Time**

HTTP applications have historically allowed three different formats for the representation of date/time stamps, but a fixed-length subset of that defined by RFC 1123 (an update to RFC 822) is preferred as an Internet standard. All HTTP date/time stamps must be represented in Greenwich Mean Time (GMT), without exception. For the purposes of HTTP, GMT is exactly equal to UTC (Coordinated Universal Time). The format looks like:

```
Sat, 26 Oct 2002 15:58:04 GMT
```
**Character Sets**
HTTP uses the same definition of the term “character set” as that described for MIME, which is more commonly referred to as a “character encoding.” HTTP character sets are identified by case-insensitive tokens. The complete set of tokens is defined by the IANA Character Set registry (RFC 1700), for example: charset=iso-8859-1.

**Content Codings**
Content coding values indicate an encoding transformation that has been or can be applied to an entity. Content codings are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity of its underlying media type and without loss of information. Frequently, the entity is stored in coded form, transmitted directly, and only decoded by the recipient. HTTP/1.1 uses content-coding values in the Accept-Encoding and Content-Encoding header fields. Although the value describes the content-coding, what is more important is that it indicates what decoding mechanism will be required to remove the encoding. Examples are gzip (RFC 1952), compress (UNIX file compression program), and deflate (RFCs 1950, 1951).

**Transfer Codings**
Transfer-coding values are used to indicate an encoding transformation that has been, can be, or may need to be applied to an entity-body in order to ensure “safe transport” through the network. This differs from a content coding in that the transfer-coding is a property of the message, not of the original entity. An example is Transfer-Encoding: chunked, which modifies the body of a message in order to transfer it as a series of chunks, each with its own size indicator, followed by an optional trailer containing entity-header fields. This allows dynamically produced content to be transferred along with the information necessary for the recipient to verify that it has received the full message.

**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. Media-type values are registered with the Internet Assigned Number Authority (RFC 1700). The media type registration process is outlined in RFC 1590. Use of non-registered media types is discouraged. Typical media types for use with HTTP are: text/html, image/gif, message/http (e.g. response on TRACE request), multipart/byteranges (HTTP 206 (Partial Content) response), and application/x-www-form-urlencoded (POST request resulting from an HTML FORM).

**Product Tokens**
Product tokens are used to allow communicating applications to identify themselves by software name and version. Examples are:

User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2.8.7 OpenSSL/0.9.6b DAV/1.0.3 PHP/4.1.2 mod_perl/1.26

**Quality Values**
HTTP content negotiation uses short “floating point” numbers to indicate the relative importance (“weight”) of various negotiable parameters. A weight is normalized to a real number in the range 0 through 1, where 0 is the minimum and 1 the maximum value. If a parameter has a quality value of 0, then content with this parameter is ‘not acceptable’ for the
client. “Quality values” is a misnomer, since these values merely represent relative
degradation in desired quality.

Language Tags
A language tag identifies a natural language spoken, written, or otherwise conveyed by human
beings for communication of information to other human beings. Computer languages are
explicitly excluded. HTTP uses language tags within the Accept-Language and
Content-Language fields. The syntax and registry of HTTP language tags is the same as
that defined by RFC 1766. Example tags include: en, en-us, de-at, where any two-letter
primary-tag is an ISO-639 language abbreviation and any two-letter initial subtag is an ISO-
3166 country code.

5. HTTP Message

HTTP messages consist of requests from client to server and responses from server to client.
Request and response messages use the generic message format of RFC 822 for transferring
entities (the payload of the message). Both types of message consist of a start-line, zero or
more header fields (also known as “headers”), an empty line (i.e., a line with nothing
preceding the CRLF) indicating the end of the header fields, and possibly a message-body.

HTTP header fields, which include general-header, request-header, response-header, and
entity-header fields, follow the same generic format as that given in Section 3.1 of RFC 822.
Each header field consists of a name followed by a colon (":") and the field value. General
header fields have general applicability for both request and response messages, but do not
apply to the entity being transferred. These header fields apply only to the message being
transmitted. Entity-header fields define metainformation about the entity-body or, if no body
is present, about the resource identified by the request. Some of this metainformation is
optional; some might be required. Request and response header fields are described in the
next section.

The message-body (if any) of an HTTP message is used to carry the entity-body associated
with the request or response. The message-body differs from the entity-body only when a
transfer-coding has been applied, as indicated by the Transfer-Encoding header field.
The rules for when a message-body is allowed in a message differ for requests and responses.
The transfer-length of a message is the length of the message-body as it appears in the
message; that is, after any transfer-codings have been applied.

6. Request and Response

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. Figure 1 shows a simple GET request:

```
GET /first.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-
powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: de-at
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: gutemine.ict.tuwien.ac.at:33333
Connection: Keep-Alive
```

Figure 1: Simple GET Request.
The request-line begins with a method token, followed by the request-URI and the protocol version. The method token indicates the method to be performed on the resource identified by the request-URI. Possible method tokens are GET, HEAD, POST, OPTIONS, TRACE, CONNECT, PUT, and DELETE.

The GET method means retrieve whatever information (in the form of an entity) is identified by the request-URI. If the request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics equivalent to the parameters on a programming language method invocation.

The **Accept** request-header field can be used to specify certain media types which are acceptable for the response. Accept headers can be used to indicate that the request is specifically limited to a small set of desired types, as in the case of a request for an in-line image. The **Accept-Language** request-header field is similar to **Accept**, but restricts the set of natural languages that are preferred as a response to the request. The **Accept-Encoding** request-header field is similar to **Accept**, but restricts the content-codings that are acceptable in the response.

The **User-Agent** request-header field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations.

The **Host** request-header field specifies the Internet host and port number of the resource being requested, as obtained from the original URI given by the user or referring resource (generally an HTTP URL). The Host field value must represent the naming authority of the origin server or gateway given by the original URL. This allows the origin server or gateway to differentiate between internally-ambiguous URLs, such as the root “/” URL of a server for multiple host names on a single IP address (virtual hosting).

The **Connection** general-header field allows the sender to specify options that are desired for that particular connection and must not be communicated by proxies over further connections. Although since HTTP/1.1 persistent connections are the default behavior of any HTTP connection, this header field explicitly demands a persistent connection by using the original HTTP/1.0 form of persistent connections (RFC 2068).

Persistent HTTP connections have a number of advantages:

- By opening and closing fewer TCP connections, CPU time is saved in routers and hosts (clients, servers, proxies, gateways, tunnels, or caches), and memory used for TCP protocol control blocks can be saved in hosts.
- HTTP requests and responses can be pipelined on a connection. Pipelining allows a client to make multiple requests without waiting for each response, allowing a single TCP connection to be used much more efficiently, with much lower elapsed time.
• Network congestion is reduced by reducing the number of packets caused by TCP opens, and by allowing TCP sufficient time to determine the congestion state of the network.
• Latency on subsequent requests is reduced since there is no time spent in TCP’s connection opening handshake.
• HTTP can evolve more gracefully, since errors can be reported without the penalty of closing the TCP connection. Clients using future versions of HTTP might optimistically try a new feature, but if communicating with an older server, retry with old semantics after an error is reported.

The response onto the request of Figure 1 then may look like Figure 2, resulting in a browser view given in Figure 3.

```
HTTP/1.1 200 OK
Date: Sat, 26 Oct 2002 13:51:17 GMT
Server: Apache/1.3.23 (Unix)  (Red-Hat/Linux) mod_ssl/2.8.7 OpenSSL/0.9.6b
DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mod_auth_pam/1.0a
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<html>
  <head>
    <title>My first document</title>
  </head>
  <body>
    <h1>Main Heading of my First Document</h1>
    <p>Some text including a link to the second page of my collection.</p>
  </body>
</html>
```

**Figure 2:** Response.

After receiving and interpreting a request message, a server responds with an HTTP response message. The first line of a response message is the status-line, consisting of the protocol version followed by a numeric status code and its associated textual phrase. The status-code element is a 3-digit integer result code of the attempt to understand and satisfy the request. These codes are fully defined in RFC 2616. The reason-phrase is intended to give a short textual description of the status-code. The status-code is intended for use by automata and the reason-phrase is intended for the human user. The client is not required to examine or display the reason-phrase. The first digit of the status-code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:

- 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.
Figure 3: Resulting browser view.

The response-header fields allow the server to pass additional information about the response which cannot be placed in the status-line. These header fields give information about the server and about further access to the resource identified by the request-URI.

The Date general-header field represents the date and time at which the message was originated, having the same semantics as orig-date in RFC 822. The field value is an HTTP-date, it must be sent in RFC 1123-date format.

The Server response-header field contains information about the software used by the origin server to handle the request. The field can contain multiple product tokens and comments identifying the server and any significant subproducts. The product tokens are listed in order of their significance for identifying the application. Note: 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. Server implementors are therefore encouraged to make this field a configurable option.

The Keep-Alive and Connection headers are here used only for compatibility reasons with HTTP/1.0 Persistent Connections (see RFC 2068).

The Content-Type entity-header field indicates the media type of the entity-body sent to the recipient or, in the case of the HEAD method, the media type that would have been sent had the request been a GET.

The message body finally contains the HTML page displayed in Figure 3.

7. Authentication: Basic and Digest Access

Authentication is described in RFC 2617. The "basic" authentication scheme is based on the model that the client must authenticate itself with a user-ID and a password for each realm. The realm value should be considered an opaque string which can only be compared for equality with other realms on that server. The server will service the request only if it can validate the user-ID and password for the protection space of the request-URI. There are no optional authentication parameters. Imagine the link in Figure 3 is utilized, resulting in the following request (Figure 4).
The Referer request-header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the request-URI was obtained (the "referrer", although the header field is misspelled.) The Referer request-header allows a server to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistyped links to be traced for maintenance. The Referer field must not be sent if the request-URI was obtained from a source that does not have its own URI, such as input from the user keyboard. Therefore, we can see the Referer here for the first time.

Now we assume the resource second.html to be protected in a protection space called "Administration Group". Upon receipt of an unauthorized request for a URI within the protection space, the origin server may respond with a challenge like the following (see Figure 5):

```
WWW-Authenticate: Basic realm="Administration Group"
```

The WWW-Authenticate response-header field must be included in 401 (Unauthorized) response messages. The field value consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the request-URI. Usually the client now has to prompt the user for his username and password (see Figure 6). If the client is not capable of prompting the user or if the request fails for any other reason, the HTML page is displayed, since that entity might include relevant diagnostic information.

To receive authorization, the client sends the userid and password, separated by a single colon (";") character, within a base64 encoded string in the credentials. If the user agent wishes to send the userid "myname" and the password "mypasswd", it would use the following header field:

```
Authorization: Basic bXluYW1lOm15cGFzc3dk
```

The following request would be similar to Figure 4, with the above header line added. If username and password were correct, the server would response appropriately. The Basic Access Authentication scheme is not considered to be a secure method of user authentication (unless used in conjunction with some external secure system such as SSL), as the user name and password are passed over the network as uuencoded cleartext. Therefore, a scheme based on cryptographic hashes, referred to as "Digest Access Authentication" is proposed for better security (RFC 2617).
HTTP/1.1 401 Authorization Required
Date: Sat, 26 Oct 2002 15:42:08 GMT
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2.8.7 OpenSSL/0.9.6b
DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mod_auth_pam/1.0a
WWW-Authenticate: Basic realm="Administration Group"
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1

<html>
<head>
<title>401 Authorization Required</title>
</head>
<body>
<h1>Authorization Required</h1>
<p>This server could not verify that you are authorized to access the document requested. Either you supplied the wrong credentials (e.g., bad password), or your browser doesn't understand how to supply the credentials required.</p>
<hr/>
<address>Apache/1.3.23 Server at gutemine.ict.tuwien.ac.at Port 33333</address>
</body>
</html>

Figure 5: Authorization required.

Figure 6: The client prompts the user for username and password.

8. Caching and Proxies

HTTP is typically used for distributed information systems, where performance can be improved by the use of response caches. The HTTP/1.1 protocol includes a number of elements intended to make caching work as well as possible. Caching would be useless if it did not significantly improve performance. The goal of caching in HTTP/1.1 is to eliminate the need to send requests in many cases, and to eliminate the need to send full responses in many other cases. The former reduces the number of network round-trips required for many operations; the protocol uses an “expiration” mechanism for this purpose. The latter reduces
network bandwidth requirements; the protocol uses a “validation” mechanism for this purpose.

GET http://gutemine.ict.tuwien.ac.at:33333/second.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: de-at
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: gutemine.ict.tuwien.ac.at:33333

Figure 7: GET request to a proxy server.

Figure 7 shows a GET request being sent to a proxy server instead of being sent to the origin server gutemine.ict.tuwien.ac.at, therefore the request-URI in the request-line contains not just path and query, but also scheme and authority. In general, a chain of proxy servers is possible. In our example, the proxy server now forwards the request to the origin server, which is shown in Figure 8.

We recognize a new header field: The Via general-header field must be used by gateways and proxies to indicate the intermediate protocols and recipients between the user agent and the server on requests, and between the origin server and the client on responses. It is analogous to the Received field of RFC 822 and is intended to be used for tracking message forwards, avoiding request loops, and identifying the protocol capabilities of all senders along the request/response chain.

GET /second.html HTTP/1.0
Via: 1.1 ADPROSRV01
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: gutemine.ict.tuwien.ac.at:33333
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: de-at
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

Figure 8: GET request forwarded from proxy server to origin server.

Following the Via header field, the received-protocol indicates the protocol version of the message received by the server or client along each segment of the request/response chain. The received-protocol version (1.1 in our example) is appended to the Via field value when the message is forwarded so that information about the protocol capabilities of upstream applications remains visible to all recipients.

The received-by field (ADPROSRV01 in our example) is normally the host and optional port number of a recipient server or client that subsequently forwarded the message. However, if the real host is considered to be sensitive information, it may be replaced by a pseudonym. If the port is not given, it may be assumed to be the default port of the received-protocol.

Multiple Via field values represents each proxy or gateway that has forwarded the message. Each recipient must append its information such that the end result is ordered according to the sequence of forwarding applications. For example, a request message could be sent from an HTTP/1.1 user agent to an internal proxy code-named ADPROSRV01, which uses HTTP/1.0 to forward the request to a public proxy at some.where.com, which completes the request.
by forwarding it to the origin server. The request received by the origin server would then have the following \texttt{Via} header field:

\texttt{Via: 1.1 ADPROSRV01, 1.0 some.where.com}

As there is only one proxy in our example, the origin server now responds as shown in Figure 9, stores the response in its cache, and forwards the response to the user agent as shown in Figure 10.

\begin{verbatim}
HTTP/1.1 200 OK
Date: Sat, 26 Oct 2002 15:57:32 GMT
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2.8.7 OpenSSL/0.9.6b
DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mod_auth_pam/1.0a
Connection: close
Content-Type: text/html

<HTML>
 <HEAD>
  <TITLE>My second document</TITLE>
 </HEAD>
 <BODY>
  <H1>Main Heading of my Second Document</H1>
  <P>More text.</P>
 </BODY>
</HTML>
\end{verbatim}

\textbf{Figure 9:} Response of the origin server received by the proxy.

\begin{verbatim}
HTTP/1.1 200 OK
Via: 1.1 ADPROSRV01
Connection: close
Content-Type: text/html
Date: Sat, 26 Oct 2002 15:58:04 GMT
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2.8.7 OpenSSL/0.9.6b
DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mod_auth_pam/1.0a
Keep-Alive: timeout=15, max=100

<HTML>
 <HEAD>
  <TITLE>My second document</TITLE>
 </HEAD>
 <BODY>
  <H1>Main Heading of my Second Document</H1>
  <P>More text.</P>
 </BODY>
</HTML>
\end{verbatim}

\textbf{Figure 10:} Response of the proxy received by the user agent.

If the same request of Figure 7 is now sent to the proxy again, the proxy server will check the validity of its cache regarding the request and if it is valid will answer immediately without first contacting the origin server as shown in Figure 11.
HTTP/1.1 200 OK
Via: 1.1 ADPROSRV01
Connection: close
Content-Length: 157
Age: 588
Content-Type: text/html
Date: Sat, 26 Oct 2002 16:07:52 GMT
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2.8.7 OpenSSL/0.9.6b
DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mod_auth_pam/1.0a
Keep-Alive: timeout=15, max=100

<HTML>
  <HEAD>
    <TITLE>My second document</TITLE>
  </HEAD>
  <BODY>
    <H1>Main Heading of my Second Document</H1>
    <P>More text.</P>
  </BODY>
</HTML>

Figure 11: Response of the proxy, when the requested information is retrieved from a valid cache.

The Age response-header field conveys the sender's estimate of the amount of time since the response (or its revalidation) was generated at the origin server. A cached response is fresh if its age does not exceed its freshness lifetime. Age values are non-negative decimal integers, representing time in seconds. Age values are calculated as specified in RFC 2616 and used for cache expiration control.

9. Further HTTP Request Methods by Example

This section provides further examples for HTTP request methods.

OPTIONS
The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval. In Figure 12 the server allows the methods GET, HEAD, TRACE, and OPTIONS.

OPTIONS / HTTP/1.1
Host: gutemine.ict.tuwien.ac.at:33333

HTTP/1.1 200 OK
Date: Sat, 26 Oct 2002 17:18:34 GMT
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2.8.7 OpenSSL/0.9.6b
DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mod_auth_pam/1.0a
Content-Length: 0
Allow: GET, HEAD, OPTIONS, TRACE

Figure 12: The OPTIONS method request and an appropriate response.

HEAD
The HEAD method is identical to GET except that the server must not return a message-body in the response (see Figure 13). The metainformation contained in the HTTP headers in response to a HEAD request should be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by
the request without transferring the entity-body itself. This method is often used for testing
hypertext links for validity, accessibility, and recent modification.

```
HEAD /goeschka/test/first.html HTTP/1.1
Host: gutemine.ict.tuwien.ac.at:33333

HTTP/1.1 200 OK
Date: Sat, 26 Oct 2002 17:21:32 GMT
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2.8.7 OpenSSL/0.9.6b
DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mod_auth_pam/1.0a
Content-Type: text/html
```

**Figure 13:** The HEAD method request and an appropriate response.

**POST**
The POST method is used to request that the origin server accepts the entity enclosed in the
request as a new subordinate of the resource identified by the request-URI in the request-line.
POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form, to a data-handling process;
- Extending a database through an append operation.

The actual function performed by the POST method is determined by the server and is usually
dependent on the request-URI. The posted entity is subordinate to that URI in the same way
that a file is subordinate to a directory containing it, a news article is subordinate to a
newsgroup to which it is posted, or a record is subordinate to a database.

Consider a Web form, like in Figure 14: Important is the FORM Tag, describing the request-URI and the request method for submitting the form. Pressing the submit button then causes the user agent to initiate a POST request as shown in Figure 15.
Figure 14: Web form: HTML and user agent representation.

```
POST /cgi-bin/form HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: de-at
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: gutemine.ict.tuwien.ac.at:33333
Content-Length: 58
Connection: Keep-Alive

srch=dogfish&srch_type=Exact+Match&srvr=Canada&srvr=Sweden
```

Figure 15: POST request resulting from the Web form in Figure 14.

Noteable is the Content-Type application/x-www-form-urlencoded used for submitting Web form data to server side applications. Also important the Content-Length, because unlike most other requests, the POST request usually does have a content. If the FORM action GET would be used instead of POST, the resulting GET request is shown in Figure 16. Now the query string is not delivered within the message content but rather appended to the request-URI. Therefore, no Content-Type or Content-Length are necessary.

```
GET /cgi-bin/?srch=dogfish&srch_type=Exact+Match&srvr=Canada&srvr=Sweden HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: de-at
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: gutemine.ict.tuwien.ac.at:33333
Connection: Keep-Alive
```

Figure 16: GET request resulting from the Web form in Figure 14, if GET is used as FORM action instead of POST.
TRACE
The TRACE method is used to invoke a remote, application-layer loop-back of the request message. The final recipient of the request should reflect the message received back to the client as the entity-body of a 200 (OK) response. A TRACE request must not include an entity. TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information. The value of the Via header field is of particular interest, since it acts as a trace of the request chain. Use of the Max-Forwards header field allows the client to limit the length of the request chain, which is useful for testing a chain of proxies forwarding messages in an infinite loop. If the request is valid, the response should contain the entire request message in the entity-body, with a Content-Type of message/http. Responses to this method must not be cached. Figure 17 shows an example of a TRACE request to a proxy (PAULI) and the respective response.

```
TRACE http://www.ict.tuwien.ac.at/goeschka/test/first.html HTTP/1.1
Host: www.ict.tuwien.ac.at

HTTP/1.0 200 OK
Via: 1.0 PAULI
Content-Type: message/http
Date: Sat, 26 Oct 2002 20:56:12 GMT
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2.8.7 OpenSSL/0.9.6b
DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mod_auth_pam/1.0a

TRACE /goeschka/test/first.html HTTP/1.0
Connection: Keep-Alive
Host: www.ict.tuwien.ac.at
Via: 1.0 PAULI
```

Figure 17: The TRACE method.

10. Further HTTP Response Status Codes by Example

This section provides further examples for HTTP response status codes. For the complete list of all status codes, please refer to RFC 2616.

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. 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.

Successful 2xx

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

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.

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. 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. For example, the simple request from Figure 4 may result in a response shown in Figure 18.

HTTP/1.1 302 Found
Date: Sat, 26 Oct 2002 22:22:08 GMT
Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_ssl/2.8.7 OpenSSL/0.9.6b
DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mod_auth_pam/1.0a
Location: http://some.other.host.ac.at:22222/second.html
Content-Type: text/html; charset=iso-8859-1

<HTML>
<HEAD>
<TITLE>302 Found</TITLE>
</HEAD>
<BODY>
<H1>Found</H1>
<P>The document has moved
<A HREF="http://some.where.else.ac.at:22222/second.html">here</A>.
</P>
</BODY>
</HTML>

Figure 18: Found response.

The Location response-header field is used to redirect the recipient to a location other than the request-URI for completion of the request or identification of a new resource. For 3xx responses, the location should indicate the server’s preferred URI for automatic redirection to the resource. The field value consists of a single absolute URI. If the user-agent understands the response, it will immediately send the following request to the new destination as shown in Figure 19. Otherwise, the user-agent will display the HTML-content of Figure 18 to the user as shown in Figure 20.

GET /second.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: de-at
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: some.where.else.ac.at:22222

Figure 19: Redirected request.
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. Figure 21 shows a conditional GET request to a proxy server.

```
GET http://gutemine.ict.tuwien.ac.at:33333/second.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: de-at
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: gutemine.ict.tuwien.ac.at:33333
If-Modified-Since: Sat, 26 Oct 2002 22:13:35 GMT

HTTP/1.1 304 Not Modified
Content-Length: 0
Connection: close
Via: 1.1 ADPROSRV01
```

Figure 21: Conditional GET request and Not Modified response.

The If-Modified-Since request-header field is used with a method to make it conditional: if the requested variant has not been modified since the time specified in this field, an entity will not be returned from the server; instead, a 304 (not modified) response will be returned without any message-body. A GET method with an If-Modified-Since header and no Range header requests that the identified entity be transferred only if it has been modified since the date given by the If-Modified-Since header. Assuming the document has not been modified since the given time, Figure 21 shows the 304 response from the proxy to the client.

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.
404 Not Found
Unfortunately, one of the most common response status codes: 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. Figure 22 shows the HTTP response and Figure 23 the respective display for the user.

407 Proxy Authentication Required
This code is similar to 401 (Unauthorized) explained in section 7, but indicates that the client must first authenticate itself with the proxy. The proxy must return a Proxy-Authenticate header field containing a challenge that indicates the authentication scheme and parameters applicable to the proxy for this request-URI. The client may repeat the request with a suitable Proxy-Authorization header field (section 14.34). HTTP access authentication is explained in RFC 2617.
HTTP/1.1 407 Proxy authentication required
Proxy-Authenticate: NTLM
Proxy-Connection: close
Content-Length: 503
Content-Type: text/html

<html>
<head>
<title>Error 407</title>
<meta name="robots" content="noindex">
&lt;META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
</head>
<body>
<h2>HTTP Error 407</h2>
<p><strong>407 Proxy Authentication Required</strong></p>
<p>You must authenticate with a proxy server before this request can be serviced. Please log on to your proxy server, and then try again.</p>
<p>Please contact the Web server's administrator if this problem persists.</p>
</body>
</html>

Figure 24: 407 – Proxy authentication required.

HTTP Error 407

407 Proxy Authentication Required

You must authenticate with a proxy server before this request can be serviced. Please log on to your proxy server, and then try again.

Please contact the Web server's administrator if this problem persists.

Figure 25: 407 information displayed to the user.

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.

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. This status code is also used for cache control. Consider a request to a proxy server using the Cache-control header field (see Figure 26).

The Cache-Control general-header field is used to specify directives that must be obeyed by all caching mechanisms along the request/response chain. The directives specify behavior intended to prevent caches from adversely interfering with the request or response. These
directives typically override the default caching algorithms. Cache directives are unidirectional in that the presence of a directive in a request does not imply that the same directive is to be given in the response.

In our example the cache-request-directive only-if-cached is used: In some cases, such as times of extremely poor network connectivity, a client may want a cache to return only those responses that it currently has stored, and not to reload or revalidate with the origin server. To do this, the client may include the only-if-cached directive in a request. If it receives this directive, a cache should either respond using a cached entry that is consistent with the other constraints of the request, or respond with a 504 (Gateway Timeout) status (see Figure 27) which is then displayed to the user (see Figure 28).

```
GET http://www.ict.tuwien.ac.at/goeschka/test/second.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: de-at
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: www.ict.tuwien.ac.at
Cache-control: only-if-cached

Figure 26: Request with Cache-control header.
```

```
HTTP/1.1 504 Proxy Error ( This operation returned because the timeout period expired. )
Via: 1.1 ADPROSRV01
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage 2.0">
<title>Proxy Server Report: Default</title>
</head>
<body bgcolor="#BEBEBE">
<h2>Proxy Reports:</h2>

............
</body>
</html>

Figure 27: Proxy Error response (HTML content shortened).
```

The cache-response-directive no-cache has the following meaning in this example: If the no-cache directive does not specify a field-name, then a cache must not use the response to satisfy a subsequent request without successful revalidation with the origin server. This allows an origin server to prevent caching even by caches that have been configured to return stale responses to client requests.

ThePragma general-header field is used to include implementation-specific directives that might apply to any recipient along the request/response chain. All pragma directives specify optional behavior from the viewpoint of the protocol; however, some systems may require that behavior be consistent with the directives.
When the no-cache directive is present in a request message, this pragma directive has the same semantics as the no-cache cache-directive and is defined for backward compatibility with HTTP/1.0. HTTP/1.1 caches should treat Pragma: no-cache as if the client had sent Cache-Control: no-cache. No new Pragma directives will be defined in HTTP. As the meaning of Pragma: no-cache as a response header field is not actually specified, it does not provide a reliable replacement for Cache-Control: no-cache in a response.

Figure 28: 504 information (proxy error) displayed to the user.

References

There is no other way than to study the RFCs (Request For Comments) if you want to get the information source, for instance from ftp://ftp.isi.edu/in-notes/.

For the HTTP the following RFCs are relevant:

2616 Hypertext Transfer Protocol – HTTP/1.1 (obsoletes 2068) [1999/06]
2617 HTTP Authentication: Basic and Digest Access Authentication (obsoletes 2069) [1999/06]
2145 Use and Interpretation of HTTP Version Numbers [1997/05]
2109 HTTP State Management Mechanism [1997/02]
2068 Hypertext Transfer Protocol – HTTP/1.1 [1997/01] – obsoleted by 2616
2067 An Extension to HTTP : Digest Access Authentication [1997/01] – obsoleted by 2617
1945 Hypertext Transfer Protocol – HTTP/1.0 [1996/05]
Regarding URI/URL consider the following RFCs:

2717 Registration Procedures for URL Scheme Names [1999/11]
2718 Guidelines for new URL Schemes [1999/11]
2396 Uniform Resource Identifiers (URI): Generic Syntax (obsoletes 1808, 1738) [1998/08]
2141 URN Syntax (updates 1737, 1630) [1997/05]
1808 Relative Uniform Resource Locators [1995/06] – obsoleted by 2396
1738 Uniform Resource Locators (URL) [1994/12] – obsoleted by 2396
1630 Universal Resource Identifiers in WWW [1994/06] – updated by 2141

The following RFCs are related to HTTP and URN:
1952 GZIP file format specification version 4.3 [1996/05]
1951 DEFLATE Compressed Data Format Specification version 1.3 [1996/05]
1950 ZLIB Compressed Data Format Specification version 3.3 [1996/05]
1766 Tags for the Identification of Languages [1995/03]
1700 Assigned Numbers [1994/10]
1590 Media Type Registration Procedure [1994/03]
1123 Requirements for Internet Hosts -- Application and Support [1989/10]
822 Standard for the format of ARPA Internet Text Messages [1982/08]

Of course the World Wide Web Consortium provides various resources on their Web page http://www.w3.org/, especially http://www.w3.org/Protocols/ and http://www.w3.org/Addressing/, and is an excellent starting point for information search on Web issues in general.