HTTP 0.9

Die Urversion des Hypertext Transport Protocols bietet ein einfaches Request-Response Modell aufbauend auf einer TCP Verbindung.

- Client baut eine TCP Verbindung auf, der Default für den Zielport ist 80.
- Er sendet eine Anfragezeile der Form
  GET Path?Search, abgeschlossen durch ein CRLF (ASCII Codes 13, 10)
- Der Server antwortet mit einem Textdokument mit HTML Markup (`http://www.w3c.org/MarkUp/`).
- Das Ende des Dokuments wird durch Schließen der Verbindung angezeigt.

Aktueller Standard

Aktuell ist HTTP in der Version 1.1 (RFC2616). Die Änderungen zur ersten Version machen es zu einem vielseitig verwendbaren Transport.

- Neben Text mit HTML Markup werden andere Datentypen und Formate unterstützt.
- Gegenseitige Authentifizierung der Kommunikationspartner ist möglich.
- Virtual Hosting ermöglicht verschiedene, unabhängige Dienste auf einem Server hinter einer einzigen IP Adresse.
- Für den Nutzer transparente Daten- und Transportkodierungen bieten effiziente Nutzung der Serverresourcen.
- HTTP unterstützt Infrastruktur, die den Einsatz aus privaten Netzen ermöglicht.

SMTP Header (RFC822)

Das Format einer HTTP Nachricht entspricht weitgehend einer E-Mail, d.h. nach der Anfragezeile folgt eine Menge von Name-Wert Paaren (sog. Header), dann - durch eine Leerzeile getrennt - die Nutzdaten. Zeilenumbrüche bestehen immer aus CR-LF (ASCII Codes 13, 10).

- Namen und Werte der Header sind durch Doppelpunkt und Leerzeichen voneinander getrennt.
- Bei den Namen wird nicht zwischen Groß- und Kleinschreibung unterschieden.
- Mehrere Header mit gleichem Namen sind möglich, wenn die Syntax für den Wert eine Folge von durch Komma getrennten Werten vorschreibt.
- Die Reihenfolge von Headern ist nur von Bedeutung, wenn es Header mit gleichem Namen sind.

Header Folding

Lange Zeilen (mehr als 72 Zeichen) sollen zur Verbesserung der Lesbarkeit vermieden werden. Header Zeilen müssen dazu umgebrochen werden. Dazu wird an LWSP, d.h. Leerzeichen (SPACE) oder Tabulator (HT) getrennt.

- Enthält der Wert eines Headers ein LWSP, kann es durch CRLF + LWSP ersetzt werden (Folding).
- Beginnt umgekehrt eine Headerzeile mit einem LWSP, ist es die Fortsetzung der vorausgehenden Zeile. Dann muß vor der weiteren Verarbeitung das vorausgehende CRLF entfernt werden.

Beispiel

```
GET /Test HTTP/1.1
Host: localhost:8080
Accept: text/html, text/plain, text/css,
  text/sgml, */*;q=0.01
Accept-Encoding: gzip
Accept-Language: en
User-Agent: Lynx/2.8.6rel.4 libwww-FM/2.14
  SSL-MM/1.4.1 GNUTLS/1.6.2
```

Statuscodes

HTTP benutzt wie SMTP Statuscodes bestehend aus 3 Ziffern mit folgender Interpretation:

Code	Bedeutung
100-199	Information
200-299	Erfolg
300-399	Wiederhole mit anderen Daten
400-499	Client Fehler
500-599	Server Fehler

Hex/Base16 Kodierung (RFC3548)

RFC822 erlaubt nur den (7 Bit) ASCII Zeichensatz. Um Binärdaten in Headern zu übertragen, müssen diese daher auf Zeichenketten aus dem ASCII Alphabet abgebildet werden.

Bei der Hex Kodierung werden die 16 möglichen Werte einer 4 Bit Folge abgebildet auf die 16 Zeichen 0123456789abcdef.

Damit wird jedes Byte dargestellt als 2 ASCII Zeichen.
Beispiel: Die Bytes 127, 62 werden dargestellt als `7F 3E`

Beim Dekodieren wird nicht zwischen Groß- und Kleinschreibung unterschieden.

Base64 Kodierung (RFC3548)

Base16 Kodierung ist bei großen Datenmengen ineffizient, da sie die Datenmenge (gemessen in Byte) verdoppelt. Die Base64 Kodierung ist in diesem Fall um 33% besser.

- Je 6 Bit werden als ein Zeichen kodiert.
- Zielalphabet sind die Zeichen A-Z, a-z, 0-9 und “+”, “/” in dieser Reihenfolge (z.B. $27_{10} = 011011_2 = b_{64}$).
- Da immer je 24 Bit kodiert werden müssen, werden Bytefolgen mit 0 Bits aufgefüllt.
- 6 Füllbits werden durch ’=’ dargestellt.

Beispiel: Die Bytes 127, 62 werden dargestellt als `fz4=`, denn: 127,62 entspricht den Bits 011111 110011 1110`00` `000000`. Die je 6 Bit entsprechen den Zahlen 31(f), 51(z), 56(4) und Füller(=).

Aufbau von Nachrichten

HTTP implementiert ein Request-Response Modell.
Eine Nachrich hat den Aufbau:

```
Anfragezeile/Statuszeile
Headerzeilen
Leerzeile
optionaler Datenblock
```

Eine Anfragezeile hat die Form

```
Methode ' ' URI ' ' HTTP-Version
```

Eine Statuszeile hat die Form

```
HTTP-Version Status-Code Reason-Phrase
```

HTTP 1.1 Methoden

Folgende Methoden sind standardisiert:

Name	Zweck
OPTIONS	Lesen von Transportoptionen, z.B. `Allow`
GET	Lesen von Daten vom Server
HEAD	Wie GET, Server sendet keinen Datenblock
POST	Senden von Daten zum Server
PUT	Speichern von Daten unter der URI
DELETE	Löschen von Daten unter der URI
TRACE	Server sendet den Request als Response
CONNECT	Nur für SSL/TLS Proxies

Ein Server muß nicht alle diese Methoden implementieren, Erweiterungen sind möglich.

Uniform Resource Identifier (vgl. RFC2396)

Die HTTP URL hat die Form:

```
"http://" host [ ":" port ] [ path [ "?" query ]]
```

Dabei sind Teile in eckigen Klammern "[]" optional.

- **Host**: Hostname des Dienstes.
- **Port**: Portnummer, 80, falls weggelassen.
- **path**: Pfad zur Resource, jeder Pfad startet mit "/"
- **query**: Serverspezifische Parameter, üblicherweise Name "=" Wert Paare, getrennt durch "&" (sog. Formkeys)

IP Adresse statt Hostname ist in URIs zu vermeiden. Falls eine IPv6 Adresse angegeben werden muß, wird sie in eckige Klammern geschrieben.

Schreibweise für URIs

Da URIs auch über andere Transportwege als Computernetze weitergegeben werden, dürfen sie nur druckbare Zeichen enthalten.

Im Pfad einer URI sind mindestens die folgenden Zeichen US-ASCII Zeichen zu ersetzen:

- Codes 0-31 und 127: Control Character
- Code 32: Space
- < > # % ¨: Begrenzer
- { } | \ ˆ [] ‘: Diese Zeichen können von bestimmten Transportmechanismen verändert werden.

Escape Sequenz in URIs

Einige Zeichen haben an bestimmten Stellen der URI eine spezielle Bedeutung und müssen dann ebenfalls ersetzt werden (z.B. im Query Anteil):x

- ; / ? : @ & = + $,

Spezielle Zeichen und nicht druckbare Zeichen werden durch ihre Hexadezimaldarstellung, eingeleitet durch ein %-Zeichen, angegeben.

Beispiel: Host `a.b.c`, Port 80, Path `/a_b/c` und Formkeys M=a+b, N="a=b" wird geschrieben als:
`http://a.b.c/a_b/c?M=a%2Bb&N=%22a%3Db%22`

Minimaler HTTP 1.1 Request

Die URL wird vom Client in einen HTTP 1.1 Request umgesetzt. Dazu wird die Serverinformation, d.h. Hostname und Port in den `Host` Header kopiert. Die Anfragezeile enthält im Normalfall als URI nur Path, Query und HTTP-Version. Ausnahme ist der Proxy Request.
Bespiel: Ein `GET` Request auf
`http://localhost:8080/test` führt zu `GET /test HTTP/1.1`
`Host: localhost:8080`
Der Header mit Namen `Host` ist bei HTTP 1.1 verbindlich.

HTTP 1.1 Header

In Request und Response sind unterschiedliche Header üblich. Prinzipiell können beiden Nachrichten beliebige Header hinzugefügt werden (sog. extension-header).

Für Requests sind die folgenden Headergruppen standardisiert:

- General Header
- Request Header
- Entity Header

Für Responses entsprechend:

- General Header
- Response Header
- Entity Header

General Header

Ein Auszug aus der Liste der standardisierten General Header:

- **Cache-Control**: Legt fest, was beim Caching der Daten zu beachten ist. (z.B. no-cache, max-age)
- **Connection**: Zeigt an, ob die TCP Verbindung für weitere Anfragen verwendet werden kann (z.B. keep, close)
- **Date**: Zeitpunkt, zu dem die Nachricht erzeugt worden ist.
- **Pragma**: Implementationsspezifische Parameter (z.B. no-cache)
- **Trailer**: Bei Chunked-Encoding können die angegebenen Header am Ende der Nachricht im Datenblock auftauchen.
- **Transfer-Encoding**: Legt fest, wie der Datenblock übertragen wird (z.B. chunked).
- **Via**: Wird von Systemen zwischen Quelle und Ziel eingesetzt, um Schleifen zu entdecken.

Transfer-Encodings

RFC2616 erwähnt folgende Kodierungen:

- **identity**: Die Nachrict wird nicht speziell kodiert.
- **gzip**: Transparente Komprimierung mit Lempel-Ziv Algorithmus.
- **compress**: Unix `compress` LZW Format.
- **deflate**: zlib Format (RFC1950 + RFC1951)
- **chunked**: Der Datenblock besteht aus Länge-Wert kodierten Segmenten.
  - Jedes Segment beginnt mit einer Zeile mit Länge in Hexadezimaldarstellung und optionalem Kommentar
  - Es folgt die angegebene Anahl Bytes.
  - Das letzte Segment hat die Länge 0
  - Auf das letzte Segment können HTTP Header folgen.

Request Header

Ein Auszug aus der Liste der Request Header:

- **Accept, Accept-Charset, ...**: Liste der Mediatypen, Kodierungen und Sprachen, die der Client akzeptiert
- **Authorization, Proxy-Authorization**: Übergabe von Daten und Anforderung der Authentifizierung.
- **Host**: Servername (eventuell mit Port)
- **If-Match, If-Modified-Since, ...**: Anforderung von Daten nur unter gegebener Bedingung.
- **Max-Forwards**: Maximale Anzahl von Proxies in der Kette.
- **Referer**: URI, von der aus die aktuelle URI angewählt worden ist.
- **TE**: Liste der möglichen Transfer-Encodings
- **User-Agent**: Identifikation des Clients

Response Header

Ein Auszug der Liste der Response-Header:

- **Age**: Alter eines Dokumentes (im Cache)
- **ETag**: Dokumentversion
- **Location**: URI des Dokumentes (benutzt im `201 Created` und z.B. im `302 Redirect`)
- **Proxy-Authenticate, WWW-Authenticate**: Authentifizierungsdaten des Clients
- **Retry-After**: Im `503 Service Unavailable` und bei `3xx Redirect`, wann die Daten verfügbar sein werden.
- **Server**: Identifikation (Typ, Version) des Servers
- **Vary**: Liste der Request-Header, die die Antwort festlegen, nötig für Proxies, um zu entscheiden, ob die Antwort aus dem Cache benutzt wird.

Entity Header

- **Allow**: Methoden, die der Server erlaubt
- **Content-Encoding**: Kodierung, die für die Daten verwendet worden ist (z.B. gzip)
- **Content-Language**: ausgewählte Sprache
- **Content-Length**: Anzahl Bytes im Datenblock, falls nicht `Transfer-Encoding: chunked`.
- **Content-Location**: URI des Dokuments
- **Content-MD5**: Hash der Daten, Prüfsumme des (dekodierten) Datenblocks
- **Content-Range**: gelieferter Bereich in der Antwort `206 Partial Content`
- **Content-Type**: Medientyp
- **Expires**: Wann Daten im Cache veraltet sind
- **Last-Modified**: Zeitpunkt der letzten Änderung

Content-Type Header

- Der Content-Type gibt den Medientyp im Datenblock an.
- Format ist stets `Type "/" Subtype*(";" Parameter)`
- Die standardisierten Medientypen finden sich unter `http://www.iana.org/assignments/media-types/`
- Parameter dienen zur weiteren Festlegung der Interpretation, z.B. `text/plain; charset=utf8`.
- Der Typ `multipart/form-data` (vgl. RFC1867) dient der Übertragung von Formkeys mit Zusatzinformationen wie Zeichensatz oder Medientyp bei Dateiübertragung.

Viele Clients (z.B. MS Windows, Mobiltelefone) benutzen zur Feststellung des Medientyps nicht den Content-Type Header, sondern eventuell vorhandene Dateiheader.

HTTP Proxies

Proxies sind integraler Bestandteil einer HTTP Infrastruktur. Haupteinsatzzwecke sind:

- **Effizienzsteigerung**: Zwischenspeichern von statischen Daten reduziert den Netzwerkverkehr
- **Zugriffskontrolle**: Netzwerkbereiche können nur über Proxies erreicht werden, die Authentifizierung vorschreiben.
- **Protokollierung/Abrechnung**: Proxies können Datenvolumen und Zugriffszeiten protokollieren.
- **Routing**: Zugriff aus privaten Netzen kann über die öffentliche Adresse eines Proxies ermöglicht werden.
- **Sicherheit**: Komplexe Webserver werden durch einfache Proxies vom Internet isoliert.

HTTP Proxies

Man unterscheidet folgende Typen von Proxies:

- **Vorwärtsproxies**: Der Client muß konfiguriert werden, um den Proxy zu benutzen.
- **Rückwärtsproxies**: Dem Client gegenüber verhalten sie sich wie Server. Sie blenden Pfade fremder Server in den eigenen Bereich ein.
- **Transparente Proxies**: Die TCP Verbindung von Clients wird abgefangen und auf den Proxy umgeleitet. Dieser baut bei Bedarf eine eigene Verbindung zum Server auf.

Die Anfragezeile bei Vorwärtsproxies enthält nicht nur den Pfad, sondern die komplette URL.

HTTP Authentifizierung

Soll mittels HTTP auf geschützte Bereiche sowohl auf einem Server als auch hinter einem Proxy zugegriffen werden, erlaubt der Standard, daß Authentifizierungsdaten abgefragt werden.

Wird der Zugriff verweigert, sendet ein Server eine Response mit Statuscode 401, ein Proxy eine Response mit Statuscode 407.

In der Response findet sich der `WWW-Authenticate` oder `Proxy-Authenticate` Header, der festlegt, wie der Client auf den geschützten Bereich zugreifen kann.

Der Clent muß dann die Authentifizierungsdaten im `Authorization` bzw. `Proxy-Authorization` Header liefern.

Basic Authentication (RFC2617)

Basic Authentication funktioniert dadurch, daß beim Zugriff auf den geschützten Bereich Benutzername und Kennwort im Klartext übertragen werden.

Findet ein Zugriff ohne gültige Authentifizierung statt, folgt vom Server eine Antwort mit Status 401 und `WWW-Authenticate` Header mit Parametern `basic` und dem Namen des Bereiches.

Der Client wiederholt den Request und sendet den Header `Authorization: Credentials`, wobei Credentials Benutzername ":" Kennwort in Base64 Kodierung ist.

Beispiel Basic Authentication, erster Versuch

```
GET //scripts/Literaturliste.pdf HTTP/1.0
Host: www.comnets.rwth-aachen.de


HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic
realm=ßcript-downloads"
Content-Type: text/html; charset=iso-8859-1

HTML Fehlertext
```

Beispiel Basic Authentication, erfolgreich

```
GET //scripts/Literaturliste.pdf HTTP/1.0
Host: www.comnets.rwth-aachen.de
Authorization: Basic dXNlcjpwYXNzd29yZA==


HTTP/1.1 200 OK
Content-Length: 34544
Content-Type: application/pdf

PDF Dokument
```

Digest Access Authentication (RFC2617)

Der entscheidende Nachteil der Basic-Authentication ist, daß jeder, der die Datenübertragung abhört, Benutzernamen und Kennwort erfährt. Dies wird bei der Digest-Authentication vermieden.

Der `WWW-Authenticate` oder `Proxy-Authenticate` Header hat die Form: `Digest Challenge`, wobei Challenge eine Folge von Name "=" Wert Paaren ist, die durch "," getrennt werden.

Die Antwort hat die Form `Digest Response`, wobei Response dasselbe Format wie Challenge hat.

Challenge Parameter

- **realm**: Name des Sicherheitsbereiches
- **domain**: Liste von URI Präfixen, für die die Credentials gelten
- **nonce**: Eindeutige Challenge
- **opaque**: Wert, der vom Client zum Server zurückgeschickt wird
- **stale**: Zeigt an, daß der vorherige Nonce abgelaufen ist.
- **algorithm**: Zu verwendender Hash Algorithmus, z.B. MD5, SHA1
- **qop**: angebotene Sicherheitsstufen, z.B. Authentizität (auth), Integrität (auth-int)
- **auth-param**: z.Zt nicht benutzt

Response Parameter

- **username**: Name, unter dem der Client sich anmeldet
- **realm, nonce, algorithm, opaque**: die Werte der Challenge.
- **uri**: URI, die zur Challenge geführt hat (die Anfragezeile könnte von einem Proxy verändert worden sein)
- **response**: Hash, der aus Nonce, Benutzername, Kennwort und möglicherweise weiteren Bestandteilen der Nachricht berechnet wurde.
- **cnonce**: Zufälliger Text des Clients, der in den Hash einbezogen wird und Chosen Plaintext Angriffe verhindert.
- **qop**: gewählte Sicherheitsstufe
- **nonce-count**: Zähler, wie oft der nonce in Requests verwendet worden ist
- **auth-param**: z.Zt. nicht benutzt

Digest Auth Beispiel

```
GET /apache2-default/ HTTP/1.1
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US)
Host: localhost
Accept: text/html;q=0.9,text/plain;q=0.8,*/*;q=0.5
Accept-Language: en
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: close
If-Modified-Since: Sat, 20 Nov 2004 20:16:24 GMT
Cache-Control: max-age=0
```

Digest Auth Beispiel

```
HTTP/1.1 401 Authorization Required
Date: Sun, 09 Dec 2007 15:07:31 GMT
Server: Apache/2.2.6 (Debian)
WWW-Authenticate: Digest realm="my-realm",
  nonce="6by31ttABAA=1b869666ab4d0c05c785475d376797a
  algorithm=MD5, qop="auth"
Content-Length: 475
Connection: close
Content-Type: text/html; charset=iso-8859-1

475 Bytes Fehlertext
```

```
GET /apache2-default/ HTTP/1.1
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US)
Host: localhost
Accept: text/html;q=0.9,text/plain;q=0.8,*/*;q=0.5
Accept-Language: en
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: close
If-Modified-Since: Sat, 20 Nov 2004 20:16:24 GMT
Cache-Control: max-age=0
Authorization: Digest username="username",
  realm="my-realm",
  nonce="6by31ttABAA=1b869666ab4d0c05c785475d376797ac126
  uri="/apache2-default/", algorithm=MD5,
  response="6f8e48f7d00453d12f63405b53984537",
  qop=auth, nc=00000001, cnonce="de371d3080a00b58"
```

Cookies

HTTP bietet im Protokoll keine Möglichkeit, Zustandsinformationen zwischen verschiedenen Requests zu transportieren. Zwar gibt es eine Vielzahl von Ansätzen, dieses Problem zu lösen, die jedoch alle anwendungsabhängig sind oder nur eingeschränkte Möglichkeiten bieten (z.B. HTML Hidden Felder oder Session ID in der URL).

Netscape hat daher HTTP um sogenannte Cookies erweitert:

- Die originale Spezifikation findet sich unter `http://wp.netscape.com/newsref/std/cookie_spec.h`
- RFC2109 erweitert den Vorschlag von Netscape so, daß bisherige Server und Clients interoperieren können.
- RFC2965 führt einen neuen Header ein, um die Zustandsinformation zu speichern.

Austausch von Cookies

Cokies werden in der Response vom Server im “Set-Cookie” oder “Set-Cookie2” (RFC2965) Header gesetzt. Sie bestehen aus Segmenten, die durch Semikolon voneinander getrennt sind. Erstes Segment ist ein Name “=” Wert Paar, das den Namen des Cookies festlegt.

In weiteren Requests sendet der Client das Cookie im “Cookie” Header an den Server zurück, sofern im Cookie enthaltene Bedingungen erfüllt sind.

Der Server kann über das Cookie verschiedene Requests demselben Client zuordnen.

Felder im Cookie

- **Comment**: Für Menschen lesbarer Kommentar zum Cookie
- **Domain**: Domain (oder ein Suffix beginnend mit .), für den das Cookie gilt.
- **Max-Age**: Nach dieser Zeit in Sekunden soll der Client das Cookie nicht mehr benutzen.
- **Path**: Pfad (oder ein Präfix), für den das Cookie gilt. Nur für URIs mit einem Pfad unterhalb des Attributes wird das Cookie benutzt.
- **Secure**: Versende das Cookie nur bei ausreichend hoher Sicherheit (z.B. bei verschlüsselter Verbindung).
- **Version**: Version des Cookies (verbindlich = 1 für RFC2109, nicht benutzt für Netscape)

RFC2965 benutzt einige weitere Felder. Version ist ebenfalls 1.

Verarbeitung auf Clientseite

- Ist **Domain** nicht angegeben, wird der FQDN der URI benutzt.
- Ist **Max-Age** nicht angegeben, wird das Cookie bei Beenden des Clients verworfen.
- Der Standardwert für **Path** ist der Path der aktuellen URI bis zum letzten “/”.
- Ein fehlendes **Secure** wird als nicht gesetzt angenommen.

Cookies werden vom Client verworfen, wenn

- **Path** kein Präfix des aktuellen Path ist.
- der Host nicht zur **Domain** gehört.
- **Domain** keinen eingeschlossenen Punkt enthält.
- Falls das Präfix, das zusammen mit **Domain** den FQDN des Hosts ergibt, einen Punkt enthält.

Beispiel

```
GET http://www.google.de/ HTTP/1.0
Host: www.google.de


HTTP/1.0 200 OK
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie:
PREF=ID=112381d8e7dc6b1e:TM=1197301885:...;
  path=/; domain=.google.de
Via: 1.0 localhost.localdomain:3128
(squid/2.6.STABLE17)
Proxy-Connection: close
Connection: close

...HTML Text...
```