


# Hypertext Transfer Protocol

Das **Hypertext Transfer Protocol** (**HTTP**, englisch für *Hypertext-Übertragungsprotokoll*) ist ein zustandsloses Protokoll zur Übertragung von Daten auf der Anwendungsschicht über ein Rechnernetz. Es wird hauptsächlich eingesetzt, um Webseiten (Hypertext-Dokumente) aus dem World Wide Web (WWW) in einen Webbrowser zu laden. Es ist jedoch nicht prinzipiell darauf beschränkt und auch als allgemeines Dateiübertragungsprotokoll sehr verbreitet.

HTTP wurde von der Internet Engineering Task Force (IETF) und dem World Wide Web Consortium (W3C) standardisiert. Aktuelle Version ist HTTP/2, welche als RFC 7540 am 15. Mai 2015 veröffentlicht wurde. Die Weiterentwicklung wird von der HTTP-Arbeitsgruppe der IETF (HTTPbis) organisiert. Es gibt zu HTTP ergänzende und darauf aufbauende Standards wie HTTPS für die Verschlüsselung übertragener Inhalte oder das Übertragungsprotokoll WebDAV.

Hypertext Transfer Protocol	
	
Familie:	Internetprotokollfamilie
Einsatzfeld:	Datenübertragung (Hypertext u. a.) auf Anwendungsschicht
aufbauend auf	TCP (Transport)
Einführung:	1991
aktuelle Version:	2 (2015)
Standard:	<p><a href="#">RFC 1945</a> HTTP/1.0 (1996)</p> <p><a href="#">RFC 2616</a> HTTP/1.1 (1999)</p> <p><a href="#">RFC 7540</a> HTTP/2 (2015)</p> <p><a href="#">RFC 7541</a> Header Compression(2, 2015)</p> <p><a href="#">RFC 7230</a> Message Syntax and Routing (1.1, 2014)</p> <p><a href="#">RFC 7231</a> Semantics and Content(1.1, 2014)</p> <p><a href="#">RFC 7232</a> Conditional Requests(1.1, 2014)</p> <p><a href="#">RFC 7233</a> Range Requests(1.1, 2014)</p> <p><a href="#">RFC 7234</a> Caching (1.1, 2014)</p> <p><a href="#">RFC 7235</a> Authentication (1.1, 2014)</p>

## Inhaltsverzeichnis

- 1 **Eigenschaften**
- 2 **Aufbau**
- 3 **Funktionsweise**
- 4 **Geschichte**
  - 4.1 HTTP/1.0
  - 4.2 HTTP/1.1
  - 4.3 HTTP/2
- 5 **HTTP-Anfragemethoden**
- 6 **Argumentübertragung**
  - 6.1 HTTP GET
  - 6.2 HTTP POST
- 7 **HTTP-Statuscodes**
- 8 **HTTP-Authentifizierung**
- 9 **HTTP-Kompression**
- 10 **Applikationen über HTTP**
- 11 **Siehe auch**
- 12 **Weblinks**



Nutzern begegnet das Kürzel häufig z. B. am Anfang einer Web-Adresse in der Adresszeile des Browsers

## Eigenschaften

---

Nach etablierten Schichtenmodellen zur Einordnung von Netzwerkprotokollen nach ihren grundlegenden oder abstrakteren Aufgaben wird HTTP der sogenannten Anwendungsschicht zugeordnet. Diese wird von den Anwendungsprogrammen angesprochen, im Fall von HTTP ist das meist ein Webbrowser. Im ISO/OSI-Schichtenmodell entspricht die Anwendungsschicht der Schicht 7.

HTTP ist ein zustandsloses Protokoll. Informationen aus früheren Anforderungen gehen verloren. Ein zuverlässiges Mitführen von Sitzungsdaten kann erst auf der Anwendungsschicht durch eine Sitzung über einen Sitzungsbezeichner implementiert werden. Über Cookies in den Header-Informationen können aber Anwendungen realisiert werden, die Statusinformationen (Benutzereinträge, Warenkörbe) zuordnen können. Dadurch werden Anwendungen möglich, die Status- beziehungsweise Sitzungseigenschaften erfordern. Auch eine Benutzerauthentifizierung ist möglich. Normalerweise kann die Information, die über HTTP übertragen wird, auf allen Rechnern und Routern gelesen werden, die im Netzwerk durchlaufen werden. Über HTTPS kann die Übertragung aber verschlüsselt erfolgen.

Durch Erweiterung seiner Anfragemethoden, Header-Informationen und Statuscodes ist HTTP nicht auf Hypertext beschränkt, sondern wird zunehmend zum Austausch beliebiger Daten verwendet, außerdem ist es Grundlage des auf Dateiübertragung spezialisierten Protokolls WebDAV. Zur Kommunikation ist HTTP auf ein zuverlässiges Transportprotokoll angewiesen, wofür in nahezu allen Fällen TCP verwendet wird.

Derzeit werden hauptsächlich zwei Protokollversionen, HTTP/1.0 und HTTP/1.1, verwendet. Neuere Versionen wichtiger Webbrowser wie Chromium, Opera, Firefox und Internet Explorer sind darüber hinaus bereits kompatibel zu der neu eingeführten Version 2 des HTTP (HTTP/2)

## Aufbau

---

Die Kommunikationseinheiten in HTTP zwischen Client und Server werden als *Nachrichten* bezeichnet, von denen es zwei unterschiedliche Arten gibt: die *Anfrage* (englisch *Request*) vom Client an den Server und die *Antwort* (englisch *Response*) als Reaktion darauf vom Server zum Client.

Jede Nachricht besteht dabei aus zwei Teilen, dem Nachrichtenkopf (englisch *Message Header*, kurz: *Header* oder auch HTTP-Header genannt) und dem Nachrichtenrumpf (englisch *Message Body*, kurz: *Body*). Der Nachrichtenkopf enthält Informationen über den Nachrichtenrumpf wie etwa verwendete Kodierungen oder den Inhaltstyp, damit dieser vom Empfänger korrekt interpretiert werden kann (→ Hauptartikel: Liste der HTTP-Headerfelder). Der Nachrichtenrumpf enthält schließlich die Nutzdaten.

## Funktionsweise

---

Wenn auf einer Webseite der Link zur URL `http://www.example.net/infotext.html` aktiviert wird, so wird an den Rechner mit dem Hostnamen `www.example.net` die Anfrage gerichtet, die Ressource `/infotext.html` zurückzusenden.

Der Name `www.example.net` wird dabei zuerst über das DNS-Protokoll in eine IP-Adresse umgesetzt. Zur Übertragung wird über TCP auf den StandardPort 80 des HTTP-Servers eine HTTP-GET-Anforderung gesendet.

Anfrage:

```
GET /infotext.html HTTP/1.1
```

```
Host: www.example.net
```

Enthält der Link Zeichen, die in der Anfrage nicht erlaubt sind, werden diese %-kodiert. Zusätzliche Informationen wie Angaben über den Browser, zur gewünschten Sprache etc. können über den Header (Kopfzeilen) in jeder HTTP-Kommunikation übertragen werden. Mit dem „Host“-Feld lassen sich verschiedene DNS-Namen unter der gleichen IP-Adresse unterscheiden. Unter HTTP/1.0

ist es optional, unter HTTP/1.1 jedoch erforderlich. Sobald der Header mit einer Leerzeile (beziehungsweise zwei aufeinanderfolgenden Zeilenenden) abgeschlossen wird, sendet der Rechner, der einen Web-Server (an Port 80) betreibt, seinerseits eine HTTP-Antwort zurück. Diese besteht aus den Header-Informationen des Servers, einer Leerzeile und dem tatsächlichen Inhalt der Nachricht, also dem Dateiinhalt der *infotext.html*-Datei. Übertragen werden normalerweise Dateien in Seitenbeschreibungssprachen wie (X)HTML und alle ihre Ergänzungen, zum Beispiel Bilder, Stylesheets (CSS), Skripte (JavaScript) usw., die meistens von einem Browser in einer lesbaren Darstellung miteinander verbunden werden. Prinzipiell kann jede Datei in jedem beliebigen Format übertragen werden, wobei die „Datei“ auch dynamisch generiert werden kann und nicht auf dem Server als physische Datei vorhanden zu sein braucht (zum Beispiel bei Anwendung von CGI, SSI, JSP, PHP oder ASP.NET). Jede Zeile im Header wird durch den Zeilenumbruch <CR><LF> abgeschlossen. Die Leerzeile nach dem Header darf nur aus <CR><LF>, ohne eingeschlossene Leerzeichen, bestehen.

```
$ telnet www.perdu.com 80
Trying 208.97.177.124...
Connected to www.perdu.com.
Escape character is '^['.

GET / http/1.1
Host: www.perdu.com

HTTP/1.1 200 OK
Date: Sat, 17 Aug 2013 12:14:56 GMT
Server: Apache
Accept-Ranges: bytes
X-Mod-Pagespeed: 1.1.23.1-2169
Vary: Accept-Encoding
Cache-Control: max-age=0, no-cache
Content-Length: 204
Content-Type: text/html

<html><head><title>Vous Etes Perdu ?</title></head><body><h1>Perdu sur l'Interne
t ?</h1><h2>Pas de panique, on va vous aider</h2><strong><pre>* <----- vous
&ecirc;tes ici</pre></strong></body></html>
Connection closed by foreign host.
$
```

Beispiel einer Transaktion, ausgeführt mit `Telnet`

Antwort:

```
HTTP/1.1 200 OK
Server: Apache/1.3.29 (Unix) PHP/4.3.4
Content-Length: 123456(Größe von infotext.html in Byte)
Content-Language: de(nach RFC 3282 sowie RFC 1766)
Connection: close
Content-Type: text/html
```

(*Inhalt von infotext.html*)

Der Server sendet eine Fehlermeldung sowie einen Fehlercode zurück, wenn die Information aus irgendeinem Grund nicht gesendet werden kann, allerdings werden auch dann Statuscodes verwendet, wenn die Anfrage erfolgreich war, in dem Falle (meistens) 200 OK. Der genaue Ablauf dieses Vorgangs (Anfrage und Antwort) ist in der HTTP-Spezifikation festgelegt.

## Geschichte

Ab 1989 entwickelten Roy Fielding, Tim Berners-Lee und andere am CERN, dem europäischen Kernforschungszentrum in der Schweiz, das Hypertext Transfer Protocol, zusammen mit den Konzepten URL und HTML, womit die Grundlagen des World Wide Web geschaffen wurden. Erste Ergebnisse dieser Bemühungen war 1991 die Version HTTP 0.9.<sup>[1]</sup>

## HTTP/1.0

Die letztlich im Mai 1996 als RFC 1945 (Request for Comments Nr. 1945) publizierte Anforderung ist als HTTP/1.0 bekannt geworden. Bei HTTP/1.0 wird vor jeder Anfrage eine neue TCP-Verbindung aufgebaut und nach Übertragung der Antwort standardmäßig vom Server wieder geschlossen. Sind in ein HTML-Dokument beispielsweise zehn Bilder eingebettet, so werden insgesamt elf TCP-Verbindungen benötigt, um die Seite auf einem grafikfähigen Browser aufzubauen.

## HTTP/1.1

1999 wurde eine zweite Anforderung als RFC 2616 publiziert, die den HTTP/1.1-Standard wiedergibt.<sup>[2]</sup> Bei HTTP/1.1 kann ein Client durch einen zusätzlichen Headereintrag (*Keepalive*) den Wunsch äußern, keinen Verbindungsabbau durchzuführen, um die Verbindung erneut nutzen zu können (persistent connection). Die Unterstützung auf Serverseite ist jedoch optional und kann in Verbindung mit Proxys Probleme bereiten. Mittels HTTP-Pipelining können in der Version 1.1 mehrere Anfragen und Antworten pro TCP-Verbindung gesendet werden. Für das HTML-Dokument mit zehn Bildern wird so nur eine TCP-Verbindung benötigt. Da die Geschwindigkeit von TCP-Verbindungen zu Beginn durch Verwendung des Slow-Start-Algorithmus recht gering ist, wird so die Ladezeit für die gesamte Seite signifikant verkürzt. Zusätzlich können bei HTTP/1.1 abgebrochene Übertragungen fortgesetzt werden.

Eine Möglichkeit zum Einsatz von HTTP/1.1 in Chats ist die Verwendung des MIME-Typs *multipart/replace*, bei dem der Browser nach Senden eines Boundary-Codes und eines neuerlichen *Content-Length*-Headerfeldes sowie eines neuen *Content-Type*-Headerfeldes den Inhalt des Browserfensters neu aufbaut.

Mit HTTP/1.1 ist es neben dem Abholen von Daten auch möglich, Daten zum Server zu übertragen. Mit Hilfe der PUT-Methode können so Webdesigner ihre Seiten direkt über den Webserver per WebDAV publizieren und mit der DELETE-Methode ist es ihnen möglich, Daten vom Server zu löschen. Außerdem bietet HTTP/1.1 eine TRACE-Methode, mit der der Weg zum Webserver verfolgt und überprüft werden kann, ob die Daten korrekt dorthin übertragen werden. Damit ergibt sich die Möglichkeit, den Weg zum Webserver über die verschiedenen Proxys hinweg zu ermitteln, ein traceroute auf Anwendungsebene.

Aufgrund von Unstimmigkeiten und Unklarheiten wurde 2007 eine Arbeitsgruppe gestartet, die die Spezifikation verbessern sollte. Ziel war hier lediglich eine klarere Formulierung, neue Funktionen wurden nicht eingebaut. Dieser Prozess wurde 2014 beendet und führte zu sechs neuen RFCs:

- RFC 7230 – HTTP/1.1: Message Syntax and Routing
- RFC 7231 – HTTP/1.1: Semantics and Content
- RFC 7232 – HTTP/1.1: Conditional Requests
- RFC 7233 – HTTP/1.1: Range Requests
- RFC 7234 – HTTP/1.1: Caching
- RFC 7235 – HTTP/1.1: Authentication

## HTTP/2

Im Mai 2015 wurde von der IETF HTTP/2 als Nachfolger von HTTP/1.1 verabschiedet. Der Standard ist durch RFC 7540 und RFC 7541 spezifiziert.<sup>[3]</sup> Die Entwicklung war maßgeblich von Google (*SPDY*) und Microsoft (*HTTP Speed+Mobility*)<sup>[4]</sup> mit jeweils eigenen Vorschlägen vorangetrieben worden. Ein erster Entwurf, der sich weitgehend an SPDY anlehnte, war im November 2012 publiziert und seither in mehreren Schritten angepasst worden.

Mit HTTP/2 soll die Übertragung beschleunigt und optimiert werden.<sup>[5]</sup> Dabei soll der neue Standard jedoch vollständig abwärtskompatibel zu HTTP/1.1 sein.

Wichtige neue Möglichkeiten sind

- die Möglichkeit des Zusammenfassens mehrerer Anfragen,
- weitergehende Datenkompressionsmöglichkeiten,
- die binär kodierte Übertragung von Inhalten und
- Server-initiierte Datenübertragungen (push-Verfahren).



Sir Tim Berners-Lee, der als Begründer des Webs gilt, erfand auch das HTTP mit.

Eine Beschleunigung ergibt sich hauptsächlich aus der neuen Möglichkeit des Zusammenfassens (Multiplex) mehrerer Anfragen, um sie über eine Verbindung abwickeln zu können. Die Datenkompression kann nun (mittels neuem Spezialalgorithmus namens HPACK<sup>[6]</sup>) auch Kopfdaten mit einschließen. Inhalte können binär kodiert übertragen werden. Um nicht auf serverseitig vorhersehbare Folgeanforderungen vom Client warten zu müssen, können Datenübertragungen teilweise vom Server initiiert werden (push-Verfahren).

Die ursprünglich geplante Option, dass HTTP/2 standardmäßig TLS nutzt, wurde nicht umgesetzt. Allerdings kündigten die Browser-Hersteller Google und Mozilla an, dass ihre Webbrowser HTTP/2 nicht ohne Verschlüsselung unterstützen werden (siehe Application-Layer Protocol Negotiation).

Die meist verbreiteten Browser unterstützen HTTP/2. Darunter Google Chrome (inkl. Chrome unter iOS und Android) ab Version 41, Mozilla Firefox ab Version 36,<sup>[7]</sup> Internet Explorer 11 unter Windows 10, Opera ab Version 28 (bzw. Opera Mobile ab Version 24) und Safari ab Version 8.

## HTTP-Anfragemethoden

---

### GET

ist die gebräuchlichste Methode. Mit ihr wird eine Ressource (zum Beispiel eine Datei) unter Angabe eines URI vom Server angefordert. Als Argumente in dem URI können also auch Inhalte zum Server übertragen werden, allerdings soll laut Standard eine GET-Anfrage nur Daten abrufen und sonst keine Auswirkungen haben (wie Datenänderungen auf dem Server oder ausloggen). Die Länge des URIs ist je nach eingesetztem Server begrenzt und sollte aus Gründen der Abwärtskompatibilität nicht länger als 255 Bytes sein. (siehe unten)

### POST

schickt unbegrenzte, je nach physischer Ausstattung des eingesetzten Servers, Mengen an Daten zur weiteren Verarbeitung zum Server, diese werden als Inhalt der Nachricht übertragen und können beispielsweise aus Name-Wert-Paaren bestehen, die aus einem HTML-Formular stammen. Es können so neue Ressourcen auf dem Server entstehen oder bestehende modifiziert werden. POST-Daten werden im Allgemeinen nicht von Caches zwischengespeichert. Zusätzlich können bei dieser Art der Übermittlung auch Daten wie in der GET-Methode an den URI gehängt werden. (siehe unten)

### HEAD

weist den Server an, die gleichen HTTP-Header wie bei GET, nicht jedoch den Nachrichtenrumpf mit dem eigentlichen Dokumentinhalt zu senden. So kann zum Beispiel schnell die Gültigkeit einer Datei im Browser-Cache geprüft werden.

### PUT

dient dazu, eine Ressource (zum Beispiel eine Datei) unter Angabe des Ziel-URIs auf einen Webserver hochzuladen. Besteht unter der angegebenen Ziel-URI bereits eine Ressource, wird diese ersetzt, ansonsten neu erstellt.

### PATCH

Ändert ein bestehendes Dokument ohne dieses wie bei PUT vollständig zu ersetzen. Wurde erst später durch RFC 5789 als Erweiterung des Standard vorgeschlagen (noch nicht verabschiedet).

### DELETE

löscht die angegebene Ressource auf dem Server.

### TRACE

liefert die Anfrage so zurück, wie der Server sie empfangen hat. So kann überprüft werden, ob und wie die Anfrage auf dem Weg zum Server verändert worden ist – sinnvoll für das Debugging von Verbindungen.

### OPTIONS

liefert eine Liste der vom Server unterstützten Methoden und Merkmale.

### CONNECT

wird von Proxyservern implementiert, die in der Lage sind, SSL-Tunnel zur Verfügung zu stellen.

RESTful Web Services verwenden die unterschiedlichen Anfragemethoden zur Realisierung von Webservices. Insbesondere werden dafür die HTTP-Anfragemethoden GET/POST, PUT/PATCH und DELETE verwendet.

WebDAV fügt die Methoden *PROPFIND*, *PROPPATCH*, *MKCOL*, *COPY*, *MOVE*, *LOCK* und *UNLOCK* zu HTTP hinzu.

## Argumentübertragung

Häufig will ein Nutzer Informationen an eine Website senden. Dazu stellt HTTP prinzipiell zwei Möglichkeiten zur Verfügung:

### HTTP-GET

Die Daten sind Teil der URL und bleiben deshalb beim Speichern oder der Weitergabe des Links erhalten. Sie müssen URL-kodiert werden, das heißt reservierte Zeichen müssen mit „%<Hex-Wert>“ und Leerzeichen mit „+“ dargestellt werden.

### HTTP-POST

Übertragung der Daten mit einer speziell dazu vorgesehenen Anfrageart im HTTP-Nachrichtenrumpf, so dass sie in der URL nicht sichtbar sind.

## HTTP GET

Hier werden die Parameter-Wertepaare durch das Zeichen ? in der URL eingeleitet. Oft wird diese Vorgehensweise gewählt, um eine Liste von Parametern zu übertragen, die die Gegenstelle bei der Bearbeitung einer Anfrage berücksichtigen soll. Häufig besteht diese Liste aus Wertepaaren, welche durch das &-Zeichen voneinander getrennt sind. Die jeweiligen Wertepaare sind in der Form *Parametername=Parameterwert* aufgebaut. Seltener wird das Zeichen ; zur Trennung von Einträgen der Liste benutzt.<sup>[8]</sup>

Ein Beispiel: Auf der Startseite von Wikipedia wird in das Eingabefeld der Suche „Katzen“ eingegeben und auf die Schaltfläche „Artikel“ geklickt. Der Browser sendet die folgende oder eine ähnliche Anfrage an den Server:

```
GET /wiki/Spezial:Search?search=Katzen&go=Artikel HTTP/1.1
Host: de.wikipedia.org
...
```

Dem Wikipedia-Server werden zwei Wertepaare übergeben:

Argument	Wert
search	Katzen
go	Artikel

Diese Wertepaare werden in der Form

Parameter1=Wert1&Parameter2=Wert2

mit vorangestelltem ? an die geforderte Seite angehängt.

Dadurch erfährt der Server, dass der Nutzer den Artikel Katzen betrachten will. Der Server verarbeitet die Anfrage, sendet aber keine Datei, sondern leitet den Browser mit einem Location-Header zur richtigen Seite weiter, etwa mit:

```
HTTP/1.0 302 Found
Date: Fri, 13 Jan 2006 15:12:44 GMT
Location: http://de.wikipedia.org/wiki/Katzen
...
```

Der Browser befolgt diese Anweisung und sendet aufgrund der neuen Informationen eine neue Anfrage, etwa:

```
GET /wiki/Katzen HTTP/1.1
Host: de.wikipedia.org
```

Und der Server antwortet und gibt den Artikel *Katzen* aus, etwa:

```
HTTP/1.1 200 OK
Date: Fri, 13 Jan 2006 15:12:48 GMT
Last-Modified: Tue, 10 Jan 2006 11:18:20 GMT
Content-Language: de
Content-Type: text/html; charset=utf-8

Die Katzen (Felidae) sind eine Familie aus der Ordnung der Raubtiere (Carnivora)
innerhalb der Überfamilie der Katzenartigen (Feloidea).
```

Der Datenteil ist meist länger; hier soll nur das Protokoll betrachtet werden.

## HTTP POST

Da sich die Daten nicht in der URL befinden, können per POST große Datenmengen, zum Beispiel Bilder, übertragen werden.

Im folgenden Beispiel wird wieder der Artikel *Katzen* angefordert, doch diesmal verwendet der Browser aufgrund eines modifizierten HTML-Codes (`method="POST"`) eine POST-Anfrage. Die Variablen stehen dabei nicht in der URL, sondern gesondert im Rumpfteil, etwa:

```
POST /wiki/Spezial:Search HTTP/1.1
Host: de.wikipedia.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 24
```

`search=Katzen&go=Artikel`

Auch das versteht der Server und antwortet wieder mit beispielsweise folgendem Text:

```
HTTP/1.1 302 Found
Date: Fri, 13 Jan 2006 15:32:43 GMT
Location: http://de.wikipedia.org/wiki/Katzen
```

...

## HTTP-Statuscodes

→ *Hauptartikel: [HTTP-Statuscode](#)*

Jede HTTP-Anfrage wird vom Server mit einem HTTP-Statuscode beantwortet. Er gibt zum Beispiel Informationen darüber, ob die Anfrage erfolgreich bearbeitet wurde, oder teilt dem Client, also etwa dem Browser, im Fehlerfall mit, wo (zum Beispiel Umleitung) beziehungsweise wie (zum Beispiel mit Authentifizierung) er die gewünschten Informationen (wenn möglich) erhalten kann.

### 1xx – Informationen

Die Bearbeitung der Anfrage dauert trotz der Rückmeldung noch an. Eine solche Zwischenantwort ist manchmal notwendig, da viele Clients nach einer bestimmten Zeitspanne (*Zeitüberschreitung*) automatisch annehmen, dass ein Fehler bei der Übertragung oder Verarbeitung der Anfrage aufgetreten ist, und mit einer Fehlermeldung abbrechen.

### 2xx – Erfolgreiche Operation

Die Anfrage wurde bearbeitet und die Antwort wird an den Anfrager zurückgesendet.

### 3xx – Umleitung

### Not Found

The requested URL /oldpage.html was not found on this server.

Apache/2.2.3 (CentOS) Server at www.example.com Port 80

Mit Fehler 404 kommen Web-Nutzer am häufigsten in Berührung

Um eine erfolgreiche Bearbeitung der Anfrage sicherzustellen, sind weitere Schritte seitens des Clients erforderlich. Das ist zum Beispiel der Fall, wenn eine Webseite vom Betreiber umgestaltet wurde, so dass sich eine gewünschte Datei nun an einem anderen Platz befindet. Mit der Antwort des Servers erfährt der Client im *Location*-Header, wo sich die Datei jetzt befindet.

#### 4xx – Client-Fehler

Bei der Bearbeitung der Anfrage ist ein Fehler aufgetreten, der im Verantwortungsbereich des Clients liegt. Ein 404 tritt beispielsweise ein, wenn ein Dokument angefragt wurde, das auf dem Server nicht existiert. Ein 403 weist den Client darauf hin, dass es ihm nicht erlaubt ist, das jeweilige Dokument abzurufen. Es kann sich zum Beispiel um ein vertrauliches oder nur per HTTPS zugängliches Dokument handeln.

#### 5xx – Server-Fehler

Es ist ein Fehler aufgetreten, dessen Ursache beim Server liegt. Zum Beispiel bedeutet 501, dass der Server nicht über die erforderlichen Funktionen (das heißt zum Beispiel Programme oder andere Dateien) verfügt, um die Anfrage zu bearbeiten.

Zusätzlich zum Statuscode enthält der Header der Server-Antwort eine Beschreibung des Fehlers in englischsprachigem Klartext. Zum Beispiel ist ein Fehler 404 an folgendem Header zu erkennen:

```
HTTP/1.1 404 Not Found
...
```

## HTTP-Authentifizierung

→ Hauptartikel: HTTP-Authentifizierung

Stellt der Webserver fest, dass für eine angeforderte Datei Benutzername oder Passwort nötig sind, meldet er das dem Browser mit dem Statuscode 401 *Unauthorized* und dem Header *WWW-Authenticate*. Dieser prüft, ob die Angaben vorliegen, oder präsentiert dem Anwender einen Dialog, in dem Name und Passwort einzutragen sind, und überträgt diese an den Server. Stimmen die Daten, wird die entsprechende Seite an den Browser gesendet. Es wird nach RFC 2617 unterschieden in:



HTTP-Authentifizierung

#### Basic Authentication

Die Basic Authentication ist die häufigste Art der HTTP-Authentifizierung. Der Webserver fordert eine Authentifizierung an, der Browser sucht daraufhin nach Benutzername/Passwort für diese Datei und fragt gegebenenfalls den Benutzer. Anschließend sendet er die Authentifizierung mit dem Authorization-Header in der Form *Benutzername:Passwort* Base64-codiert an den Server. Base64 bietet keinen kryptographischen Schutz, daher kann dieses Verfahren nur beim Einsatz von HTTPS als sicher angesehen werden.

#### Digest Access Authentication

Bei der Digest Access Authentication sendet der Server zusätzlich mit dem WWW-Authenticate-Header eine eigens erzeugte zufällige Zeichenfolge (nonce). Der Browser berechnet den Hashcode der gesamten Daten (Benutzername, Passwort, erhaltener Zeichenfolge, HTTP-Methode und angeforderter URI) und sendet sie im Authorization-Header zusammen mit dem Benutzernamen und der zufälligen Zeichenfolge zurück an den Server, der diese mit der selbst berechneten Prüfsumme vergleicht. Ein Abhören der Kommunikation nützt hier einem Angreifer nichts, da sich durch die Verschlüsselung mit dem Hashcode die Daten nicht rekonstruieren lassen und für jede Anforderung anders lauten.

## HTTP-Kompression

Um die übertragene Datenmenge zu verringern, kann ein HTTP-Server seine Antworten komprimieren. Ein Client muss bei einer Anfrage mitteilen, welche Kompressionsverfahren er verarbeiten kann. Dazu dient der Header *Accept-Encoding* (etwa *Accept-Encoding: gzip, deflate*). Der Server kann dann die Antwort mit einem vom Client unterstützten Verfahren komprimieren und gibt im



Header *Content-Encoding* das verwendete Kompressionsverfahren an.

HTTP-Kompression spart vor allem bei textuellen Daten (HTML, CSS, Javascript) erhebliche Datenmengen, da sich diese gut komprimieren lassen. Bei bereits komprimierten Daten (etwa gängige Formate für Bilder, Audio und Video) ist die (erneute) Kompression nutzlos und wird daher üblicherweise nicht benutzt.

In Verbindung mit einer mit TLS verschlüsselten Kommunikation führt die Komprimierung allerdings zum BREACH-Exploit, wodurch die Verschlüsselung gebrochen werden kann.

## Applikationen über HTTP

---

HTTP als textbasiertes Protokoll wird nicht nur verwendet zur Übertragung von Webseiten, es kann auch in eigenständigen Applikationen, den Webservices, verwendet werden. Dabei werden die HTTP-eigenen Befehle wie GET und POST weiterverwendet für CRUD-Operationen. Dies hat den Vorteil, dass kein eigenes Protokoll für die Datenübertragung entwickelt werden muss. Beispielfhaft wird dies bei JAX-RS eingesetzt.

## Siehe auch

---

- Request Cycle
- SOAP
- Extensible Markup Language(XML)
- Zeichenkodierung
- Content Negotiation
- HTTP ETag
- HTTP Caching

## Weblinks

---

 **Commons: Hypertext Transfer Protocol** – Sammlung von Bildern, Videos und Audiodateien

 **Wikiversity: Kurs:Internet und Verschlüsselung/World Wide Web/HTTP** – Kursmaterialien, Forschungsprojekte und wissenschaftlicher Austausch

- Arbeitsgruppe der IETF zur Weiterentwicklung von HTTP
- HTTP-Anfrage- und Antwort-Header ansehen(englisch)
- HTTP-Header mehrerer URLs gleichzeitig ausgeben
- HttpTea, ein HTTP-Analysator(Freeware, englisch)
- HTTP/2 Test überprüft HTTP/2 Support einer Webseite
- Linkkatalog zum Thema HTTP bei dmoztools.net (ehemals DMOZ)
- Microsofts ausführlicher Bericht zu http 2.0(englisch)

## Einzelnachweise

---

1. Tim Berners-Lee: *The Original HTTP as defined in 1991* (<http://www.w3.org/Protocols/HTTP/AsImplemented.html>) In: *w3.org*, abgerufen am 13. November 2010.
2. RFC 2616 *Hypertext Transfer Protocol – HTTP/1.1*
3. RFCs für HTTP/2 festgelegt und -geschrieben(<http://heise.de/-2650979>) iX, News-Meldung vom 15. Mai 2015 14:23 Uhr.
4. Christian Kirsch: *Microsoft bringt eigenen Vorschlag für HTTP2*. (<http://heise.de/-1479694>)heise.de, 29. März 2012, abgerufen am 4. April 2012.
5. Christian Kirsch: *Googles SPDY soll das Web beschleunigen*. (<http://heise.de/-858994>)heise.de, 13. November 2009; abgerufen am 4. April 2012
6. Entwurf der Spezifikation von HTTP2 (<https://http2.github.io/http2-spec/compression.html>) dem Header-Kompressionsalgorithmus für HTTP/2). HTTP-Arbeitsgruppe der IETF
7. Firefox 36 implementiert HTTP/2(<http://www.admin-magazin.de/News/Firefox36-implementiert-HTTP-2>)

8. *Appendix B: Performance, Implementation, and Design Notes* In: World Wide Web Consortium [W3C] (Hrsg.): *HTML 4.01 Specification* 24. Dezember 1999, B.2.2: Ampersands in URI attribute values [w3.org \(http://www.w3.org/TR/1999/REC-html401-19991224/appendix/notes.html#h-B.2.2\)](http://www.w3.org/TR/1999/REC-html401-19991224/appendix/notes.html#h-B.2.2)

---

Abgerufen von [https://de.wikipedia.org/w/index.php?title=Hypertext\\_Transfer\\_Protocol&oldid=170324276](https://de.wikipedia.org/w/index.php?title=Hypertext_Transfer_Protocol&oldid=170324276)

---

**Diese Seite wurde zuletzt am 25. Oktober 2017 um 21:23 Uhr bearbeitet.**

Der Text ist unter der Lizenz „Creative Commons Attribution/Share Alike“ verfügbar; Informationen zu den Urhebern und zum Lizenzstatus eingebundener Mediendateien (etwa Bilder oder Videos) können im Regelfall durch Anklicken dieser abgerufen werden. Möglicherweise unterliegen die Inhalte jeweils zusätzlichen Bedingungen. Durch die Nutzung dieser Website erklären Sie sich mit den [Nutzungsbedingungen](#) und der [Datenschutzrichtlinie](#) einverstanden. Wikipedia® ist eine eingetragene Marke der Wikimedia Foundation Inc.