Fachhochschule
Bonn-Rhein-Sieg

HTTP

Ausarbeitung im Rahmen der Veranstaltung

Verteilte und parallele Systeme II

an der Fachhochschule Bonn-Rhein-Sieg
Fachbereich Informatik

Wintersemester 04/05

Autoren: Roland Gude und Christoph Wiesen

Inhaltsverzeichnis

1 Einleitung

Das Hypertext Transfer Protocol (HTTP) ist das Anwendungsprotokoll des World-Wide-Web (WWW). Das WWW wurde Ende der 80er Jahre bei dem textitCentre Eueropéenne pour la Recherche Nucléaire (CERN) in Genf als Hypertextsystem für Physiker entwickelt. Es gewann schnell an Popularität als mit dem Browser Mosaic zususätzlich zu dem eigentlichen Hypertext auch Grafiken angezeigt wurden und der Text optisch ansprechend dargestellt werden konnte. Von diesem Zeitpunkt an entwickelte sich das WWW explosionsartig und fand auch den Weg in die Industrie und den privaten Gebrauch. Der Funktionsumfang des WWW wurde bis heute wesentlich erweitert.

HTTP ist die Basis für eine Kommunikation zwischen Web-Client und Web-Server. Es eignet sich mittlerweile für den Austausch jeglicher Dateien und nicht nur, wie der Name suggeriert, für den Austausch von Hypertext. HTTP baut auf einem Anfrage/Antwort - Verfahren auf und basiert auf TCP/IP.

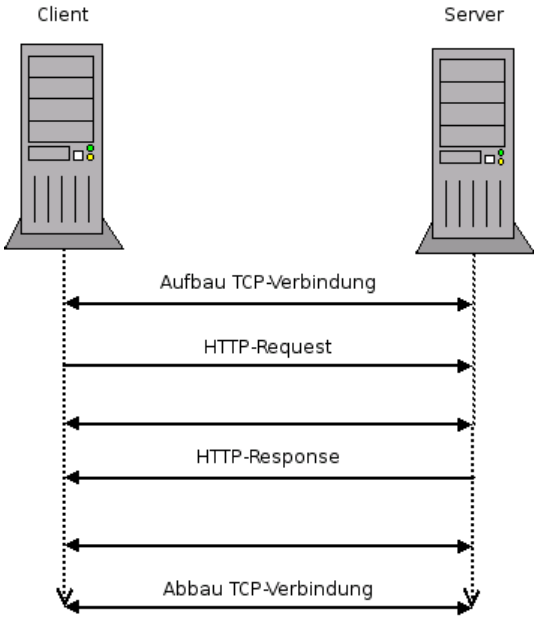

Abbildung 1: Kommunikation über HTTP

2 Geschichte

HTTP war eine der drei Basiskomponenten des World Wide Webs (URI, HTML, HTTP). Das ursprüngliche Design von HTTP war darauf fokussiert, eine leicht zu implementierendes Protokoll zu schaffen, welches simple textbasierte Dokumente von einem Server abholen kann. Ein weiteres wichtiges Designziel für HTTP war Geschwindigkeit. Diese Designziele sind heute immernoch gültig, allerdings fällt einem sparsamen Umgang mit

Ressourcen eine immer größer werdende Bedeutung zu, da Server heute meistens eine viel größere Anzahl an Anfragen bearbeiten müssen als früher.

2.1 HTTP/0.9

Das erste implementierete Protokoll hieß HTTP ohne eine Versionsnummer. Später wurde es nur noch HTTP/0.9 genannt. Diese Implementation wurde nie von einer Organisation abgesegnet. Auf Clientseite gab es nur die *GET*-Methode. Auf die der Server leiglich mit einzelnen Dokumenten antworten konnte.

2.2 HTTP/1.0

Die erste Erweiterung des HTTP Protokolls wurde im Zeitraumzwischen 1992 und 1996 entwickelt und im Internet Informational RFC 1945 [RFCa] veröffentlicht. Allerdings sollte das Dokument lediglich die Erweiterungen beschreiben, die die meisten Programmierer bereits implementiert hatten und war nicht als Standard gedacht.

HTTP/1.0 beinhaltete folgende Erweiterungen:

- MIME-types: Durch die Nutzung der ursprünglich zur Unterscheidung von Dateianhängen bei Emails gedachten Multipurpose Internet Mail Extensions (MIME) wurde HTTP um die Möglichkeit erweitert verschiedene Medientypen (Text, Audio, Grafik) unterscheiden zu können. Ein MIME Type besteht aus der Angabe eines Medientyps und eines Subtyps, wobei beide Angaben durch einen Schrägstrich getrennt werden (z.B. image/gif).

- Kopfzeilen (Header): HTTP/1.0 definierte ein vielseitiges Nachrichtenformat, mit dem es Möglich war, zusätzliche Informationen zwischen Server und Client und zurück auszutauschen. Zum Beispiel Informationen über den MIME-type einer Datei konnten übermittelt werden.

- *POST* Methode: Zusätzlich zur *GET*-Methode, mit der man lediglich Dokumente Anfordern konnte, wurde die *POST*-Methode zum Senden von Daten vom Client an den Server entwickelt. Bei der *POST*-Methode werden die Daten im Nachrichtenrumpf übertragen.

- Benutzerauthentifizierung: HTTP/1.0 führte den Basis Authentifizierungsmechanismus ein, der auch heute noch Bestandteil von HTTP ist. Er wird im Abschnitt 3.6.1 *Basisauthentifizierung* behandelt.

2.3 HTTP/1.1

Die erste Veröffentlichung von HTTP/1.1 erfolgte im November 1997. Der Grund für eine Erweiterung der Version 1.0 war das ineffiziente Handling von Verbindungen. HTTP/1.0 konnte keine Verbindungen zwischen Server und Client aufrecht erhalten. Alle Verbindungen wurden direkt nach dem Senden der Serverantwort geschlossen. Da TCP, das von HTTP genutzte Transportprotokoll, lange zum Aufbauen einer Verbindung brauchte,

kam es zu Performanceproblemen. Dieses Request/Response Modell wurde in HTTP/1.1 überarbeitet. Hierbei wurde das so genannte Persistent HTTP (P-HTTP) in HTTP integriert. P-HTTP wird im Abschnitt 3.7 *Dauehafte Verbindungen* behandelt.

Die wichtigsten Änderungen sind:

- Host-Header Feld: Ein zusätzliches Feld im HTTP-Kopf beinhaltet den Namen des Hosts. Dies ermöglicht die Realisierung von nicht IP-basierten virtuellen Hosts. Ein als virtueller Host konfigurierter Webserver, kann mehrerer Domainnamen bedienen. Bei IP-basierten virtuellen Hosts ist jeder Domainname auf eine eigene IP-Adresse abgebildet und diese werden auf die MAC-Adresse des Netzwerkinterfaces des Webservers abgebildet. Da der IP-Adressraum jedoch begrenzt ist, war es nötig virtuelle Hosts anders Identifizieren zu können. Das Host-Header Feld muss bei einer HTTP/1.1 Übertragung immer vorhanden sein.
- Absolute URIs: In HTTP 1.1 war es zum ersten Mal möglich direkt auf eine absolute URI zuzugreifen. Vorher war es nur möglich auf relative URIs zuzugreifen.
- Neue Request Methoden: In HTTP/1.1 wurden zusätzlich zu *GET* und *POST* noch die Methode *DELETE, OPTIONS, PUT* und *TRACE* eingeführt. Einige dieser Methoden werden in Abschnitt 3.3.1 *Anfragemethoden* erläutert.
- Teilweise Übertragung von Daten: Daten können byteweise angefordert werden wodurch eine unterbrochene Übertragung wieder aufgenommen werden kann.
- Content negotiation: HTTP/1.1 führt einen Mechanismus ein, mit dem es dem möglich ist, für eine zu übertragende Ressource eine bestimmte Repräsentation auszuhandeln. Dieser Mechanismus nennt sich *content negotiation*.
- Block-weise Kodierung: Da bei dauerhaften Verbindungen (P-HTTP) die Datengröße nicht mehr implizit durch das Schließen der Verbindung gekennzeichnet ist, führt HTTP/1.1 für die Daten, deren Größe nicht vorher bestimmt werden kann eine Block-weise Kodierung ein.
- Verbessertes caching: Das Cachingmodell wurde für HTTP/1.1 stark verbessert um Servern und Proxyservern eine bessere Kontrolle über das Cachen von bestimmten Ressourcen zu geben.
- Sicherere Authentifizierung: In HTTP/1.1 gibt es einen verschlüsselten Authentifikationszugang, so dass Passwörter nicht im Klartext übertragen werden.

3 Hypertext Transfer Protocol 1.1 (HTTP/1.1)

3.1 HTTP Nachrichten

HTTP benutzt Nachrichten (Messages) zur Kommunikation zwischen Server und Client. Eine Nachricht kann dabei eine Anfrage (Request) oder eine Antwort (Response) sein. Das Format dieser Nachrichten ist sehr einfach gehalten. Es besteht aus einer

Anfangszeile, einer beliebigen Anzahl an Kopfzeilen (Headers) und einem optionalen Nachrichtenrumf, der die eigentliche Nachricht (Entity) enthält. Nachrichtenkopf und Nachrichtenrumpf werden durch eine Leerzeile voneinander getrennt. Die Anfangszeile ist dabei entweder eine Anfragezeile, oder eine Statuszeile. Request und Response sind Nachrichten im Sinne des RFC 822 [RFCb].

Eine HTTP-Nachricht sieht folgendermaßen aus:

```
Startzeile
*Nachrichtenkopf
CRLF
Nachrichtenrumpf
```

3.2 Kopfzeilen

Wie im Abschnitt 3.1 *HTTP Nachrichten* beschrieben, beinhaltet eine HTTP-Nachricht eine beliebige Anzahl an Kopfzeilen, den so genannten HTTP-Header-fields. Diese Kopfzeilen beinhalten Metainformationen, mit deren Hilfe sich Client und Server besser aufeinander abstimmen können. Außerdem beinhalten die Kopfzeilen auch Informationen die für eine Kommunikation zwingend notwendig sind (z.B. den Namen der angeforderten Ressource).

Die Syntax dieser Header-Felder ist wie folgt:

```
Schlüssel: Wert {,Wert}
```

Dabei sind die Schlüssel nicht Case-sensitiv, die Werte allerdings schon. Die Reihenfolge der Schlüssel innerhalb der einzelnen Kategorien ist nicht festgelegt. Die Kategorien müssen jedoch in einer festen Reihenfolge übertragen werden:

1. Allgemeiner Kopf
2. Anfragekopf oder Antwortkopf
3. Objektkopf

Der Nachrichtenkopf (HTTP-Header) wird unterteilt in einen *allgemeinen Kopf (General Header)* und einen *Objektkopf (Entity Header)*, sowie einen *Anfragekopf (Request Header)* und einen *Antwortkopf (Response-Header)*. Der *General Header* und der *Entity Header* werden in den Unterpunkten dieses Headers behandelt, *Request-Header* und *Response Header* werden in den Abschnitten 3.3.2 *Anfragekopf* und 3.4.2 *Antwortkopf* behandelt.

3.2.1 Allgemeiner Kopf

Allgemeine Kopfzeilen (General Headers) sind die Felder des Nachrichtenkopfes, die Informationen über die Nachricht, jedoch nicht über das zu übermittelnde Objekt (entity) beinhaltet. Außerdem sind allgemeine Kopfzeilen unabhängig davon, ob es sich bei der

Nachricht um eine Anfrage oder eine Antwort handelt. Die allgemeinen Kopfzeilen werden immer als erstes übertragen.

Allgemeine Header sind zum Beispiel:

- *Connection*: Aufforderung zum Öffnen bzw Schließen der TCP-Verbindung.
- *Date*: Zeitpunkt an dem die Nachricht erzeugt wurde.
- *Transfer-Encoding*: Kodierung der Nachricht (zum Beispiel Block-weise oder gzip-komprimiert).

3.2.2 Objektkopf

Der Objektkopf (Entity Headers) besteht aus den Feldern im Nachrichtenkopf, die Informationen zu dem zu übertragenden Objekt (entity) enthalten. Die Objektkopfzeilen sind, genau wie die allgemeinen Kopfzeilen unabhängig davon ob es sich bei der Nachricht um eine Anfrage oder eine Antwort handelt. Sie werden als letzte Kopfzeilen, aber vor dem Objekt übertragen.

Beispiele für Felder der Objekt-Kopfzeilen sind:

- *Content-Length*: Länge des Objekts in Bytes.
- *Content-Type*: MIME-Type des Objekts.
- *Content-MD5*: MD5 Prüfsumme des Nachrichtenrumpfes.

3.3 Anfrage

Die häufigste Anwendung einer Anfrage eines Clients an den Server (Request), die immer die erste Nachricht in einer HTTP basierten Kommunikation darstellt, ist die Anforderung eines Dokuments. Eine Request Nachricht hat folgendes Aussehen:

```
Methode Request-URI HTTP-Version
{Allgemeine Kopfzeile}
{Anfragekopfzeile}
{Objektkopzeile}

[Nachrichtenrumpf]
```

Die erste Zeile einer Anfragenachricht (Request Zeile) besteht aus dem Methoden Feld, das dem Server mitteilt welche Aktion er mit der im Feld request-URI spezifizierten Ressource durchführen soll. Falls die Anfrage ressourcenunabhängig ist enthält die request-URI lediglich das Zeichen ' * '. Im Feld HTTP-Version wird die von der Nachricht benutzte HTTP Version übertragen. Sowohl die auf die Request Zeile folgenden Kopfzeilen als auch der Nachrichten Körper sind optional. Ein einfaches Beispiel für eine Anfrage ist eine *GET* Anfrage eines Clients:

```
GET /Index.HTML HTTP/1.0
Accept: image/gif, image/jpeg
Accept-Charset: ISO-8859-1
Accept-Language: en
Referer: http://www.inf.fh-brs.de
User-Agent: Mozilla/6.0 [en]
```

In diesem Beispiel wird als Ressource die HTML Datei Index.HTML angefordert. Wie aus der Request Zeile ersichtlich ist verwendet der Client noch die Version 1.0 des HTTP Protokolls. Außer der Request Zeile werden nur Felder des Anfragekopfes mitgeschickt. Diese werden unter dem Punkt 3.3.2 *Anfragekopf* noch im Detail erläutert.

3.3.1 Anfragemethoden

Als Anfragemethoden konnte die erste Zeile einer Anfragenachricht in der ersten Version von HTTP lediglich die Methode *GET* enthalten. Mittlerweile gibt es jedoch viele verschiedene Request Methoden, bei denen zwischen den Eigenschaften sicher und idempotent unterschieden werden.

Sichere Anfragemethoden sind solche, die lediglich dazu dienen Ressourcen anzufordern. Durch sichere Anfragen können somit keine unerwarteten Effekte für den anfordernden Client oder den Server auftreten.

Zu dieser Gruppe gehören:

- *GET*: teilt dem Server mit, die im Feld request-URI angegebene Ressource an den Client zu übertragen. Dabei muss die spezifizierte Ressource nicht bereits vorhanden sein, sondern kann unter Umständen auch erst auf dem Server dynamisch erzeugt werden (z.B. durch ein Script generierter HTML Code).
- *HEAD*: Die Methode *HEAD* ist ähnlich wie die *GET* Methode, außer dass die Server Antwort keinen Nachrichten Körper enthalten muss. Sie wird hauptsächlich zum Testen von URIs gebraucht.

Idempotente Anfragemethoden können beliebig oft durchgeführt werden ohne dass sich im Vergleich zur ersten Durchführung der Anfrage etwas ändert. Sichere Anfragemethoden sind also immer auch idempotent (dass heißt *GET* und *HEAD* sind ebenfalls idempotent).

Idempotente Methoden sind zum Beispiel:

- *PUT*: Mit dieser Methode kann der Client ein Dokument auf dem Server speichern.
- *DELETE*: Mit dieser Methode kann der Client eine Ressource auf dem Server löschen. Der Client kann sich allerdings nie sicher sein, dass die Ressource auch wirklich gelöscht wird, da der Server immer nur den Auftrag bestätigt, aber nicht den Erfolg.

- *POST*: Mit der *POST* Methode werden ebenso wie mit der *PUT* Methode Daten an den Server geschickt. Der Unterschied liegt darin, dass bei der *POST* Methode keine neuen Dateien erzeugt werden, sondern Daten an schon existierende Dateien übergeben werden.

3.3.2 Anfragekopf

Im Anfragekopf befinden sich Informationen über die eigentliche Anfrage des Clients an den Server und über die vom Client erwartete Antwort des Servers. Dabei liefern einige Kopfzeilen nur Zusatzinformationen die ignoriert werden können, während andere Kopfzeilen notwendige Informationen enthalten, die für die Bedeutung der Anfrage wichtig sind.

Beispiele für Anfragekopfzeilen sind:

- *Accept*: wird benutzt um dem Server mitzuteilen, welche Medientypen der Client als Antwort akzeptiert. Als Werte werden hier alle akzeptierten MIME-Typen angeführt (z. B. text/HTML).
- *Accept-Charset*: Wie Accept, allerdings bezogen auf die akzeptierten Zeichensätze.
- *Accept-Language*: Ähnlich wie Accept für Sprachen, allerdings werden lediglich Vorlieben mitgeteilt. Dieses Feld kann vom Server ignoriert werden.
- *Referer*: enthält die URI der Ressource, von der die URI der angefragten Ressource erhalten wurde.
- *User-Agent*: Identifiziert das Programm mit dem der Client die Ressource angefordert hat (Kennung des Browsers). Dieses Feld ermöglicht es dem Server, an den Client eine für dessen Browser optimierte Version der Ressource zu schicken.

3.4 Antwort

Als Antworten (Response) bezeichnet man die HTTP Nachrichten, die ein Server an den Client als Reaktion auf eine Anfrage (Request) sendet.

Antworten haben das gleiche Format wie Anfragen:

```
HTTP-Version Status-Code reason-phrase
{Allgemeine Kopfzeile}
{Antwortkopfzeile}
{Objektkopzeile}

[Nachrichtenrumpf]
```

In der ersten Zeile der Antwort wird zunächst die Version der vom Server benutzten HTTP-Version mitgeteilt. Sie wird gefolgt von einem Statuscode. Der Statuscode ist ein dreistelliger ganzzahliger Code, mit dessen Hilfe mitgeteilt wird, ob bei der Verarbeitung

der Anfrage ein Fehler aufgetreten ist. Der letzte Teil der ersten Zeile ist eine kurze textuelle Beschreibung dessen, wofür der Statuscode steht.

Eine Antwort auf eine fehlerfrei bearbeitete Anfrage kann zum Beispiel so aussehen:

```
HTTP/1.1 200 OK
Server: Apache/1.3.29 (Unix) PHP/4.3.4
Content-Length: (Größe von Index.HTML in Byte)
Content-Language: de
Content-Type: text/HTML
Connection: close
```

3.4.1 Statuscode Definitionen

Über den Statuscode teilt der Server dem Client den Erfolgsstatus der HTTP-Anfrage mit. Man Unterscheidet zwischen fünf verschiedenen Kategorien.

1. reine Informationen
2. Anfrage erfolgreich bearbeitet
3. zur erfolgreichen Abarbeitung werden weitere Aktionen benötigt
4. ein Fehler auf der Clientseite
5. ein Fehler auf der Serverseite

CODE	STATUS	BEDEUTUNG
200	OK	Aktion konnte ohne Probleme durchgeführt werden
204	No Content	Der Request wurde erfolgreich ausgeführt, das Dokument enthielt allerdings keine Daten.
302	Found	Die URL der Ressource wurde temporär verändert. Die aktuelle URL wurde dem Client über den Header mitgeteilt.
304	Not Modified	Antwort auf eine If-Modified-Since Anfrage, wenn die Ressource auf die sich die Anfrage bezogen hat nicht geändert wurde.
401	Unauthorized	Die angefragte Ressource erfordert eine Authentifizierung des Clients.
403	Forbidden	Der Zugriff auf die angefragte Ressource wurde verweigert.
404	Not Found	Die angefragte Ressource ist nicht verfügbar.
407	Proxy authentication required	Wie 401 - Unauthorized, wird allerdings von einem Proxy-Server anstatt von einem WWW-Server gesendet.
500	Internal Server Error	Es gab einen Fehler bei der Ausführung eines Server-seitigen Programms (zum Beispiel bei einem CGI-Programm)

Tabelle 1: Statuscodes und deren Bedeutungen

3.4.2 Antwortkopf

Im Antwortkopf (Response Header) befinden sich die Informationen, die sich auf die Antwort des Servers beziehen. Die Kopfzeilen im Antwortkopf geben außerdem Informationen über den Server und über die vom Client angeforderte Ressource.

Mögliche Felder im Antwortkopf sind:

- *Age*: Dieses Feld wird benutzt um mitzuteilen wie alt eine Antwort ist, falls diese aus dem Cache des Servers stammt.
- *Location*: Enthält eine URI die der Client anstelle der Ursprünglich angefragten URI verwenden soll. Wird zum Beispiel bei Statuscode 302 benutzt.
- *Expires*: Verfallsdatum einer Ressource.

3.5 Content negotiation

Wenn es verschieden Repräsentationen eines Dokuments gibt, die sich zum Beispiel in Sprache, Kompression oder Kodierung unterscheiden, müssen Client und Server sich verständigen, welche der vorhandenen Repräsentationen übertragen werden soll. Diesen Vorgang bezeichnet man als *Content negotiation*.

3.5.1 Server-driven negotiation

Falls die Entscheidung über die zu übermittelnde Version der Ressource auf dem Server getroffen wird, bezeichnet man den Vorgang als Server-driven. Der Server kann sich die zur Entscheidung notwendigen Informationen aus verschiedenen Quellen besorgen. Er kennt alle verfügbaren Repräsentationen und weiß daher auch, in welchen Punkten (Sprache, Kodierung) sie sich unterscheiden. Für diese Informationen braucht der Server also keine weiteren Quellen. Informationen darüber welche Version der Client bevorzugen würde kann der Server aus dem Anfragekopf entnehmen. Natürlich können auch alle anderen ihm zur Verfügung stehenden Quellen nutzen, um eine geeignete Entscheidung zu treffen (z.B. Annahme über den Standort des Clients anhand der IP Adresse und Auswahl einer Repräsentation in der passenden Sprache).

Server-seitige content negotiation ist dann von Vorteil, wenn es für den Client schwierig ist, seine präferierte Repräsentation der Ressource zu beschreiben oder wenn die Informationen, die nötig sind um die Entscheidung zu treffen lediglich dem Server bekannt sind.

3.5.2 Agent-driven negotiation

Als Agent-driven content negotiation bezeichnet man jene content negotiation, bei der der Server dem Client eine Liste aller verfügbaren Repräsentationen der angeforderten Ressource schickt und dieser sich eine Repräsentation aussucht.

Dies funktioniert wie folgt:

1. Der Client schickt eine Anfrage für eine Ressource an den Server.
2. Der Server schickt eine Antwort an den Server in der eine Liste aller verfügbaren Repräsentationen der angeforderten Ressource enthalten sind.
3. Der Client schickt eine Anfrage für eine bestimmt Repräsentation der Ressource.
4. Der Server bearbeitet die Anfrage und sendet eine ihr entsprechende Antwort.

Momentan ist es noch nicht Möglich, das der Client eine automatische Entscheidung darüber trifft, welche Repräsentation zu wählen ist. Dies liegt daran, dass das Format der Liste, die der Client vom Server erhält noch nicht exakt spezifiziert ist. Die einzige Möglichkeit für Agent-driven content negotiation ist es, die vom Server gesendete Liste an den Benutzer weiterzuleiten und ihm die Entscheidung zu überlassen.

Falls der Server dem Client einen Vorschlag machen will welche Repräsentation dieser wählen soll, kann er dies über das Location-Feld im Antwortkopf tun.

3.5.3 Transparent negotiation

Bei der Transparent negotiation gibt es zwischen Client und Server noch einen Proxy. Dieser Proxy fungiert dem Server gegenüber als Client und dem Client gegenüber als Server. Für den Server sieht die Transparent negotiation so aus, wie eine agent-driven

negotiation. Er sendet eine Liste mit allen Repräsentationen einer Ressource an den Proxy. Dieser wählt daraufhin eine Repräsentation aus und leitet die Antwort der neuen Anfrage an den Client weiter. Für diesen sieht dieser Vorgang wie eine Server-driven negotiation aus. Transparent negotiation ist also eine Kombination aus beiden Verfahren.

3.6 Authentifizierung

Normalerweise sind Ressourcen im WWW für jeden zugänglich. Manche Ressourcen sollen jedoch nur einem bestimmten Personenkreis zur Verfügung stehen und vor Zugriff durch andere Personen geschützt werden. Hierzu ist eine Identifizierungs- und Authentifizierungsmechanismus notwendig. In den meisten Fällen baut dieser Mechanismus auf Benutzernamen und Passwörtern auf.

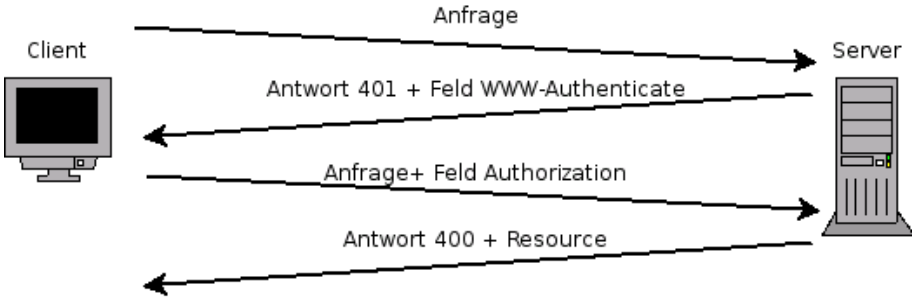

Abbildung 2: HTTP Authentifizierung

Im WWW müssen Identifizierung, Authentifizierung und Autorisierung zustandslos erfolgen, da HTTP auf zustandslosen Anfragen und Antworten basiert. Immer wenn ein Client eine Ressource anfragt, die zugriffsgeschützt ist, antwortet der Server mit dem Statuscode *401 unauthorized*. Der Antwortkopf beinhaltet in diesem Fall das Feld *WWW-Authenticate*, welches die Authentifizierungsmethode beschreibt. Hierauf schickt der Client eine erneute Anfrage, die zusätzlich zu der alten Anfrage das Feld *Authorization* im Anfragekopf mitführt. In diesem Feld werden dann die nötigen Werte zur Authentifizierung übertragen. Der Server prüft anhand dieser Werte ob der Anfragende Zugriffsberechtig (Authorized) ist, und antwortet entsprechend entweder mit der angefragten Ressource oder dem Statuscode *403 Forbidden*.

Genau wie ein WWW-Server, kann es für den Client auch nötig sein, sich bei einem Proxy-Server anzumelden, bevor dieser die Anfragen des Clients weiterleitet. Dies geschieht in einem sehr ähnlichen Verfahren. Hier Antwortet der Proxy auf die erste Anfrage mit dem Statuscode *407 Proxy authentication required* und der Client übermittelt in der zweiten Anfrage die Authentifizierungsdaten nicht im Feld *Authorization* sondern im Feld *Proxy-Authenticate*.

3.6.1 Basisauthentifizierung

Seit HTTP/1.0 ist ein Basismechanismus zur Identifizierung/Authentifizierung spezifiziert. Bei der Basisauthentifizierung enthält das Feld *WWW-Authenticate* die Kennung des Basisauthentifizierungsschemas, sowie einen Bezeichner für den geschützten Bereich (Realm). Das *Authorization*-Feld wird bei der erneuten Anfrage des Clients mit Benutzername und Passwort im Klartext belegt.

Aufgrund der Unverschlüsselten Übertragung von Benutzername und Passwort vom Client zum Server ist die Basisauthentifizierung sehr unsicher. Benutzername und Passwort können durch Filterung der TCP/IP Pakete von einem Angreifer in Erfahrung gebracht werden.

3.6.2 Digest access Authentifizierung

Aufgrund der mangelhaften Sicherheit der Basisauthentifizierung wurde mit HTTP/1.1 der Mechanismus der *digest access authentication* eingeführt. Die Grundidee hierbei ist es, nur einen MD5-Hashwert des Passwortes und nicht das Passwort selber zu übertragen. Allerdings könnten Hacker nun auch diesen Hashwert abfangen und von nun an benutzen um Zugang zum Server zu erhalten. Also hätte man hiermit noch nichts gegenüber der Basisauthentifizierung gewonnen. Aus diesem Grund wird nicht nur ein Hash über das Passwort gebildet, sondern über das Passwort und einige Zusatzinformationen aus der Anfrage, so dass es beinahe unmöglich ist, den gleichen Hashwert noch einmal nutzen zu können. Diese Zusatzinformationen sind:

- *Benutzername*
- *Passwort*
- *Realm*: Kennzeichnung des geschützten Bereichs.
- *Nonce*: Der Nonce-Wert ist ein Wer, der dem Client vom Server in dessen erster Antwort mitgeteilt wurde. Es soll Replay-Attacken verhindern. Es kann zum Beispiel aus der IP-Adresse des Clients oder einem Zeitstempel bestehen. Auch ein Kombination beider Felder ist denkbar.
- *HTTP-Methode*: Die bei der Anfrage genutzte HTTP-Methode. Entspricht der Methode in der Statuszeile.
- *Angefragte URI*

Beim digest access Verfahren kann die Serverantwort auch eine MD5-Prüfsumme über die Entity enthalten, so dass der Client prüfen kann, ob die Entity fehlerfrei übertragen wurde.

Bei Nutzung von digest access Authentifizierung ist es im Gegensatz zu Basisauthentifizierung nicht möglich das Passwort eines Benutzers abzufangen. Auch die Wiederverwendung eines abgefangenen Hashwertes ist kaum möglich. Allerdings wird auch bei diesem Verfahren der Nachrichtenrumpf nicht verschlüsselt übertragen. Angreifer können also schützenswerte Ressourcen immernoch abfangen und auslesen.

3.7 Dauehafte Verbindungen

Bei HTTP/1.0 wird für jede Kommunikation zwischen Client und Server (als pro Anfrage plus dazugehörige Antwort) eine TCP/IP-Verbindung geöffnet. Hat ein Client mehrere Anfragen für einen Server, dann müssen bei HTTP/1.0 auch mehrere TCP/IP-Verbindungen geöffnet werden. Das Öffnen und Schließen von TCP/IP-Verbindungen verbraucht in solchen Fällen zu viele Systemressourcen. Insbesondere der Server kann hierdurch stark belastet werden. Außerdem wird ein unnötig hoher Netzwerkverkehr erzeugt. Aus diesem Grund wurde in HTTP/1.1 das Konzept der persistenten Verbindungen eingeführt.

Bei persistenten Verbindungen wird die TCP/IP-Verbindung nicht nach der Serverantwort geschlossen, so dass mehrere Kommunikationsvorgänge über eine TCP/IP-Verbindung abgewickelt werden können. Möchte ein Server oder Client die Verbindung nicht weiter aufrecht erhalten, dann muss er im Allgemeinen Kopf der Anfrage beziehungsweise Antwort dem Feld *connection* den Wert *close* zuweisen. Kann eine HTTP/1.1 Anwendung keine persistenten Verbindungen aufrecht erhalten, so muss sie dieses Feld immer mit *close* belegen, da HTTP/1.1 immer persistente Verbindungen nutzt.

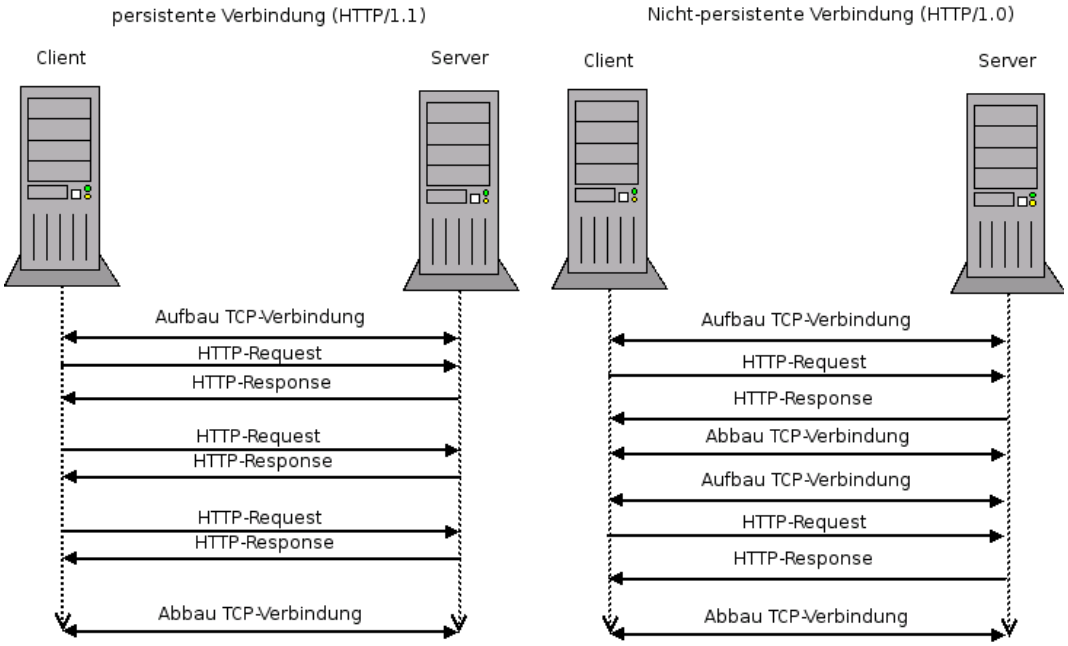

Abbildung 3: persistente Verbindungen

Ein Problem bei persistenten Verbindungen ist es, dass ein Nachrichtenempfänger nicht weiß, wann ein Nachrichtenrumpf aufhört und eine neue Nachricht anfängt. Dies war bei HTTP/1.0 implizit durch das Schließen der Verbindung gekennzeichnet. Aus diesem Grund gibt es in HTTP/1.1 die Objektkopfzeile *content-length*, in dem die Größe der Entity in bytes steht. Mit Hilfe dieser Zeile kann das Ende eines Nachrichtenrumpfs bestimmt werden.

3.8 Block-weise Kodierung

Wie im vorherigen Abschnitt erwähnt, muss einem Kommunikationspartner bei persistenten Verbindungen mitgeteilt werden wie groß eine Entity ist, die in einer Nachricht übermittlet wird. Die größe einer Entity ist aber nicht immer bekannt. Zum Beispiel wenn diese erst generiert werden muss (durch CGI-Programme oder Scripte). Für solche Fälle gibt es in HTTP/1.1 die *Block-weise Kodierung*.

Hierbei wird die Entity in Blöcke bekannter Größe aufgeteilt. Diese Blöcke werden dann nacheinander übertragen. Ein Block besteht dabei aus eine Größenangabe gefolgt von den Daten die in diesem Block sind. Als letzter Block wird ein Block mit der Größenangabe 0 und ohne Daten verschickt. Daraufhin folgt ein Trailer, der die Objektkopfzeilen enthält, für die die Länge der Nachricht bekannt sein muss.

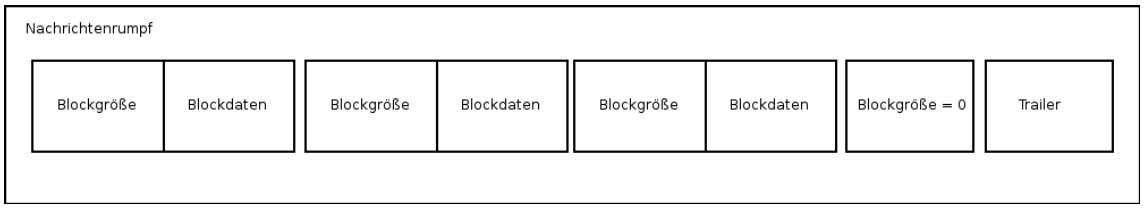

Abbildung 4: Block-weise Kodierung

4 Sicherheit

Im Abschnitt 3.6.2 *Digest access Authentifizierung* hatten wir bereits erwähnt, dass bei den Authentifizierungsmethoden die es bei HTTP/1.1 gibt lediglich Benutzernamen und Passwörter vor unbefugtem Zugriff geschützt sind, die zu Übertragenden Ressourcen jedoch nicht (Der Nachrichtenrumpf wird nicht verschlüsselt). In vielen Bereichen ist es allerdings notwendig, die Vertraulichkeit der Ressourcen auch bei der Übertragung zu gewährleisten. Hierzu werden im folgenden zwei wichtige Ansätze kurz vorgestellt.

4.1 HTTP über SSL

Der am weitesten verbreitete Ansatz zur Gewährleistung der Vertraulichkeit von per HTTP übertragenen Ressourcen ist das von Netscape entwickelte HTTP über SSL (HTTPS). Wie in diesem Dokument mehrfach erwähnt, arbeitet HTTP über eine TCP/IP-Verbindung. Bei HTTPS werden die Daten nicht direkt von HTTP an TCP/IP weitergeleitet, sondern erst über einen Zwischenschritt verschlüsselt. Zur Verschlüsselung wird der so gennannte Secure Socket Layer (SSL) verwendet. SSL leitet nach der Verschlüsselung die Daten an TCP/IP weiter. Auf der Gegenseite werden die Daten mit TCP/IP empfangen und per SSL entschlüsselt, bevor sie an das HTTP weitergeleitet werden. Es wird also einfach eine weiterer Netzwerklayer nach dem ISO/OSI Referenzmodell eingefügt.

SSL ist in der Lage weit mehr als nur Vertraulichkeit von Daten zu gewährleisten. Es bietet einen Identifizierungs- und Authentifizierungsmechanismus sowie Integrität und Verbindlichkeit von Daten. Außerdem ermöglicht es Sitzungen (Sessions). Das bedeutet, für die Kommunikationspartner ist gewährleistet, dass ihr Gegenüber über mehrere Anfragen und Antworten hinweg das gleiche ist.

Ausser SSL kann HTTPS auch noch über *Transport Layer Security (TLS)* verschlüsseln. Hierbei handelt es sich um eine freie im Funktionsumfang ähnliche Implementierung von SSL.

4.2 Secure HTTP

Neben HTTPS gibt es noch Secure HTTP (SHTTP). Im Gegensatz zu HTTPS nutzt SHTTP keine Zwischenschicht zur Ver- und Entschlüsselung, sondern ist eine Protokollerweiterung von HTTP. Es kann zwei Wege nutzen um Ressourcen zu verschlüsseln und somit Vertraulichkeit zu gewährleisten. Zum einen kann es die Nachricht nach dem *MIME Object Security Services (MOSS)* zum anderen nach der *Cryptographic Message Syntax (CMS)* verschlüsseln. Es ist so konzipiert, dass problemlos weitere Verschlüsselungsmethoden zum Protokoll hinzugefügt werden können.

5 Cookies

Da HTTP ein zustandsloses Protokoll ist, sind Informationen immer nur für ein Anfrage/Antwort Paar gültig. Möchte man jedoch Informationen über einen längeren Zeitraum hinweg speichern (zum Beispiel um Sitzungen zu realisieren) kann man diese in Form von Cookies auf dem Client speichern. Der Server kann diese Cookies nach belieben auslesen. Ein Cookie ist ein Sammlung von Schlüssel/Wert Paaren, die in einer Textdatei auf dem Client gespeichert werden. Sie haben in der Regel ein Verfallsdatum. Wird dieses überschritten, wird das Cookie gelöscht.

Mit Hilfe von Cookies kann eine Aktion ist es möglich, dass eine Anfrage eines Clients etwas bewirkt, was alle folgenden Anfragen beeinflusst. Ein klassisches Beispiel hierfür ist der Warenkorb eines Webshops. Hier werden die von einem Kunden zum Kauf ausgewählten Waren in Cookies gespeichert, bis der Kunde die Bestellung abschickt. Ausserdem kann eine Kennung zu Identifizierung des Kunden in einem Cookie gespeichert werden.

6 Zukunft von HTTP

HTTP/1.1 erfüllt die meisten Anforderungen, die heute an ein Kommunikationsprotokoll im WWW gestellt werden. Schwachstellen sind insbesondere das Authoring und Versioning von Ressourcen im Web. Diese Schwachstellen versucht die HTTP-Erweiterung *WWW Distributed Authoring and Versioning (WebDAV)* zu beheben.

WebDAV ermöglicht über eine Reihe neuer Methoden ein Distributed Authoring und eine Versionskontrolle für Ressourcen im Web.

Diese Methoden sind:

- *PROPFIND*: Fragt Eigenschaften einer Ressource, oder die Verzeichnisstruktur eines entfernten Systems ab.
- *PROPPATCH*: Ändert Eigenschaften einer Ressource.
- *MKCOL*: Erstellt Verzeichnisse.
- *COPY*: Kopiert eine Ressource.
- *MOVE*: Verschiebt eine Ressource.
- *LOCK*: Setzt eine Sperre auf eine Ressource
- *UNLOCK*: Entfernt eine Sperre von einer Ressource.

Tabellenverzeichnis

Abbildungsverzeichnis

Literatur

[Klu96] Rainer Klute. *Das World Wide Web*. Addison-Wesley, Bonn, 1996.

[RFCa] RFC 1945 - HTTP/1.0. URL `www.ietf.org/rfc/rfc1945.txt`.

[RFCb] RFC 822 - Internet Nachrichtenformat. URL `www.ietf.org/rfc/rfc822.txt`.

[SEL] HTML Webportal. URL `www.selfhtml.org/`.

[Wil99] Erik Wilde. *Wilde's WWW*. Springer-Verlag, Berlin/Heidelberg, 1999.

[WIP] Online Enzyklopädie. URL `www.wikipedia.org/`.

[Wöh04] Heiko Wöhr. *Web-Technologien*. dpunkt-Verlag, Heidelberg, 2004.