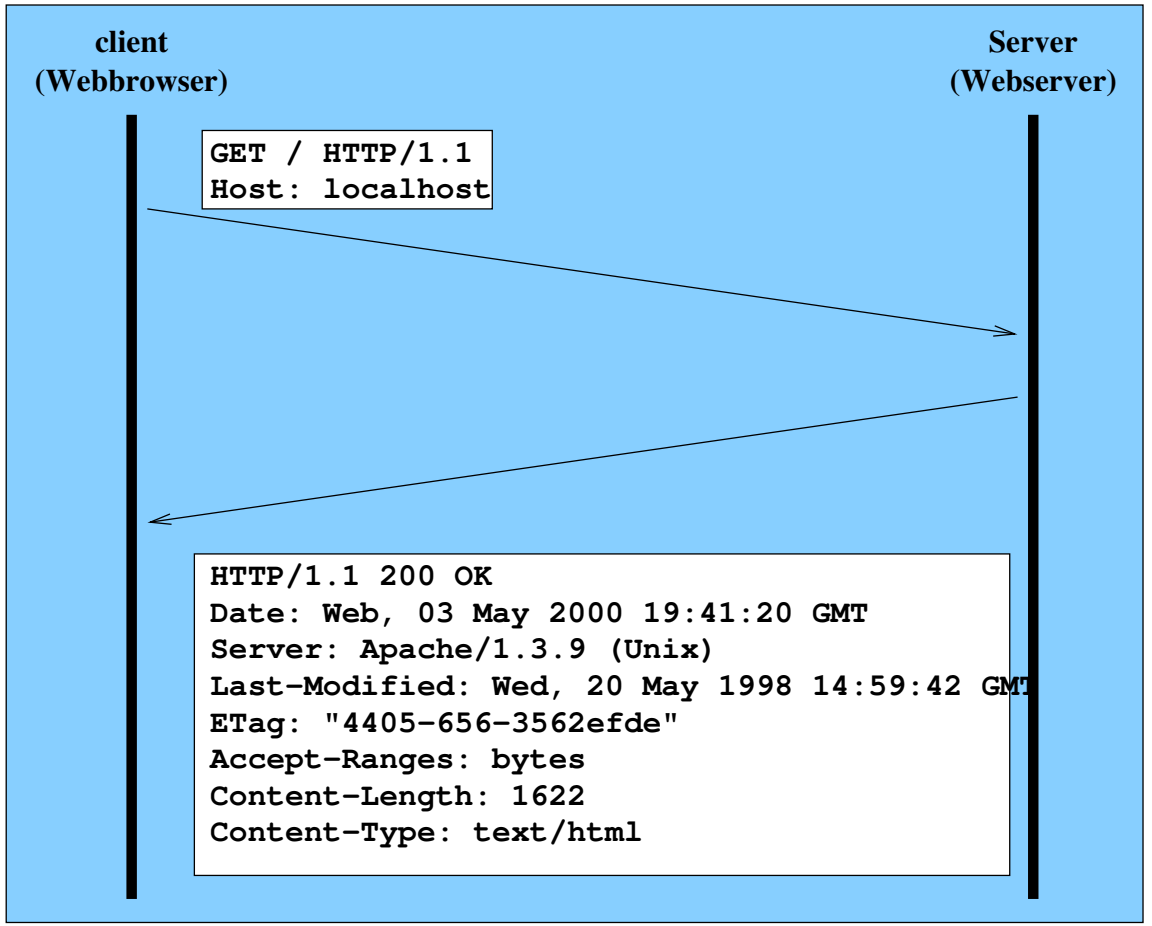
2 Hypertext Transfer Protocol (HTTP)

Aus der Definition von HTTP/1.1 (RFC 2616):

> 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.

Ein einfaches Request/Response Protokoll



HTTP, mit freundlichen Grüßen

Aufbau der Verbindung zum WWW-Server

```
[hanauma] 107 > telnet localhost www
```

TCP/IP Verbindung zum Rechner *localhost* an den Port *www* (80)

Antwort von Telnet

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

Anfrage (Request) an den WWW-Server

```
1|GET / HTTP/1.1
2|Host: localhost
3|
```

Request besteht aus Kopf (Header, Z1-2) und leerem Rumpf (Body), getrennt durch Leerzeile (Z3)

Antwort (Response) hat das gleiche Format
Kopf (Header) der Antwort

```
1|HTTP/1.1 200 OK
2|Date: Wed, 03 May 2000 19:41:20 GMT
3|Server: Apache/1.3.9 (Unix)
4|Last-Modified: Wed, 20 May 1998 14:59:42 GMT
5|ETag: "4405-656-3562efde"
6|Accept-Ranges: bytes
7|Content-Length: 1622
8|Content-Type: text/html
9|
```

Leerzeile signalisiert das Ende des Headers

Rumpf (Body) der Antwort in diesem Fall ein HTML-Dokument

```
10|<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
11|<HTML>
12| <HEAD>
13|  <TITLE>Test Page for Apache Installation on Web Site</TITLE>
14| </HEAD>
15|<!-- ... -->
16|</HTML>
```


Nach kurzer Wartezeit beendet der Server die Verbindung.

```
Connection closed by foreign host.
```

Alternativ: weitere Requests über die gleiche Verbindung

2.1 Format einer Anfrage

$$\langle Request \rangle \quad ::= \quad \langle Request\text{-}Line \rangle$$
$$\left(\left(\langle general\text{-}header \rangle \mid \langle request\text{-}header \rangle \mid \langle entity\text{-}header \rangle \right) \langle CRLF \rangle \right)^*$$
$$\langle CRLF \rangle$$
$$[\langle message\text{-}body \rangle]$$

Jede Zeile wird durch $\langle CRLF \rangle$, CR (ASCII-Kode 13) gefolgt von LF (ASCII-Kode 10), abgeschlossen.

Vgl. Methodenaufruf

- erste Zeile ($\langle Request\text{-}Line \rangle$):
  Name der Methode und vorgeschriebene Parameter
- Headerzeilen: weitere (optionale) Parameter, durch Schlüsselworte identifiziert
- Rumpf: optionaler Inhalt der Anfrage (Parameter)

Erste Zeile einer Anfrage

⟨*Request-Line*⟩ ::= ⟨*Method*⟩␣⟨*Request-URI*⟩␣⟨*HTTP-Version*⟩⟨*CRLF*⟩

⟨*Method*⟩ ::=	`GET`	*	Anfordern eines Dokuments
	| `HEAD`	*	Anfordern der Header eines Dokuments
	| `POST`		Senden einer Anfrage
	| `PUT`		Ablegen eines Dokuments
	| `DELETE`		Löschen eines Dokuments
	| `TRACE`		Anfordern der empfangenen Anfrage
	| `OPTIONS`		

* erforderliche Methoden

$$\langle \textit{Request-URI} \rangle \quad ::= \quad \langle \textit{abs\_path} \rangle \mid \langle \textit{absoluteURI} \rangle \mid \ldots$$

$$\langle \textit{HTTP-Version} \rangle \quad ::= \quad \texttt{HTTP/1.1}$$

$$\langle \textit{abs\_path} \rangle \quad ::= \quad \texttt{/}\ [\langle \textit{path} \rangle]\ [\texttt{;}\langle \textit{params} \rangle]\ [\texttt{?}\langle \textit{query} \rangle]$$

$$\langle \textit{params} \rangle \quad ::= \quad \ldots$$

$$\langle \textit{query} \rangle \quad ::= \quad \ldots$$

Beispiel (Minimalanfrage)

```
1|GET / HTTP/1.1
2|Host: www.informatik.uni-freiburg.de
3|
```

Format einer Header-Zeile

$$\langle \mathit{header} \rangle \quad ::= \quad \langle \mathit{key\text{-}token} \rangle : \langle \mathit{value} \rangle$$

Beispiel

$$\langle \mathit{Host} \rangle \quad ::= \quad \texttt{Host:}\ \langle \mathit{host} \rangle\ [\ \texttt{:}\ \langle \mathit{port} \rangle\ ]$$

Anfragespezifische Header

⟨*request-header*⟩ ::=			
	⟨*Accept*⟩		media types
|	⟨*Accept-Charset*⟩		
|	⟨*Accept-Encoding*⟩		
|	⟨*Accept-Language*⟩		
|	⟨*Host*⟩	*	Hostname des Servers
|	⟨*Referer*⟩		URL
|	⟨*User-Agent*⟩		
	⋮		

* in jeder direkten Anfrage erforderlich

Beispiele für Anfrageheader

```
Accept:   text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,
          text/plain;q=0.8,image/png,*/*;q=0.5
```

```
Accept-Charset:   ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

```
Accept-Encoding:   gzip
```

```
Accept-Language:   de,en;q=0.7,en-us;q=0.3
```

```
Host:   www.google.de
```

```
Referer:   http://web-sniffer.net/
```

```
User-Agent:   Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.6) Gecko/20050324
              Debian/1.7.6-1 Web-Sniffer/1.0.20
```

Allgemeine Header

Können in Anfragen und Antworten benutzt werden, meist optional

⟨*general-header*⟩	::=	⟨*Connection*⟩	+	persistent/one-shot
		| ⟨*Date*⟩	+!	
		| ⟨*Transfer-Encoding*⟩	+	

+	häufig in Antworten benutzt
+!	in jeder Antwort erforderlich

Entity Header

Information über den ⟨*message-body*⟩, falls vorhanden:
Content -Encoding, -Language, -Length, -Location, ...

2.2 Format einer Antwort

$$\langle Response \rangle ::= \langle Status\text{-}Line \rangle$$
$$\left(\left(\langle general\text{-}header \rangle \mid \langle response\text{-}header \rangle \mid \langle entity\text{-}header \rangle \right) \langle CRLF \rangle \right)^*$$
$$\langle CRLF \rangle$$
$$[\langle message\text{-}body \rangle]$$

Beispiel

```
1|HTTP/1.1 200 OK
2|Date: Wed, 03 May 2000 19:41:20 GMT
3|Server: Apache/1.3.9 (Unix)
4|Last-Modified: Wed, 20 May 1998 14:59:42 GMT
5|ETag: "4405-656-3562efde"
6|Accept-Ranges: bytes
7|Content-Length: 1622
8|Content-Type: text/html
9|
```

Statuszeile

$$\langle \textit{Status-Line} \rangle \quad ::= \quad \langle \textit{HTTP-Version} \rangle \sqcup \langle \textit{Status-Code} \rangle \sqcup \langle \textit{Reason-Phrase} \rangle \langle \textit{CRLF} \rangle$$

$$\langle \textit{HTTP-Version} \rangle \quad ::= \quad \texttt{HTTP/1.1}$$

$$\langle \textit{Status-Code} \rangle \quad ::= \quad \langle \textit{digit} \rangle \langle \textit{digit} \rangle \langle \textit{digit} \rangle$$

$$\langle \textit{Reason-Phrase} \rangle \quad ::= \quad \text{Text ohne } \langle \textit{CRLF} \rangle$$

Interpretation des Status-Code

`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

Most Famous Status Codes

200 OK Erfolg; Inhalt folgt.

301 Moved Permanently Neue Adresse im `Location` header.

401 Unauthorized Benutzername und Passwort erforderlich.

404 Not Found

500 Internal Server Error

- Siehe RFC 1700 für alle reservierten ⟨*Status-Code*⟩s und ⟨*Reason-Phrase*⟩s
- ⟨*Reason-Phrase*⟩s sind nur Empfehlungen, können von Server und Client ignoriert und/oder geändert werden
- Im Fehlerfall enthält ⟨*message-body*⟩ oft weitere Erklärung

Inhalt der Nachricht ⟨*message-body*⟩

- beliebige Folge von Oktetts
- falls ⟨*Content-Type*⟩ nicht vorhanden
  - Client darf aufgrund der URI raten
  - falls erfolglos `application/octetstream`
- ⟨*Content-Encoding*⟩: `gzip`, `compress`, `deflate`
  beschreiben Kodierungseigenschaften des ursprünglichen Objekts
- ⟨*Transfer-Encoding*⟩ definiert Übertragungskodierung:
  `chunked`, `gzip`, `compress`, `deflate`, `identity`

Transfer-Encoding: `identity` and `chunked`

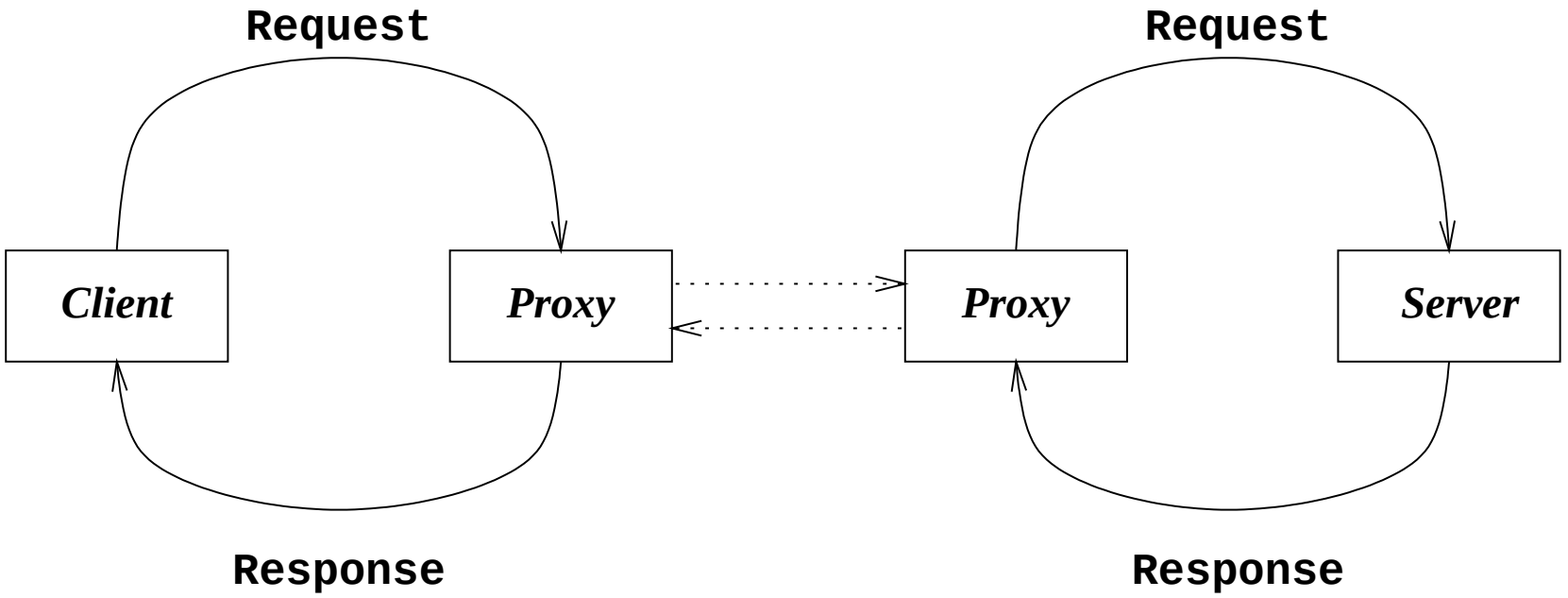
identity ⟨*Content-Length*⟩ Header definiert die Länge des ⟨*message-body*⟩ in octets; ⟨*message-body*⟩ ist Folge von octets dieser Länge
(danach folgt unvermittels die nächste Antwort bzw Verbindungsabbruch)

chunked wird verwendet, wenn die Länge nicht a-priori feststeht.
⟨*message-body*⟩ wird in *chunks* übertragen (verkürzt):

```
Chunked-Body   = *chunk
                 last-chunk
                 CRLF

chunk          = chunk-size CRLF
                 chunk-data CRLF
chunk-size     = 1*HEX
last-chunk     = 1*("0") CRLF
chunk-data     = chunk-size(OCTET)
```

2.3 Proxies



Client Initiiert Requests

Sendet an Server oder Proxy

Erhält Antworten

Proxy Mittelsmann (mit Cache)

erhält Requests

falls gültiger Eintrag vorhanden: Antwort aus Cache (an Proxy oder Client)

anderenfalls: Weiterleitung des Requests (an Proxy oder Server)

Erhält Antworten: evtl. Update des Cache, Weiterleitung der Antwort (an Proxy oder Client)

Server Erhält Requests

Liefert "Original"-Antworten (*resources*)

Anfrage an Proxyserver

$$\langle \textit{Request-URI} \rangle \quad ::= \quad \langle \textit{abs\_path} \rangle \mid \langle \textit{absoluteURI} \rangle \mid \ldots$$

$$\langle \textit{absoluteURI} \rangle \quad ::= \quad \langle \textit{scheme} \rangle \texttt{://} \langle \textit{authority} \rangle \langle \textit{abs\_path} \rangle$$

Minimalanfrage an Proxyserver

```
1|GET http://www.informatik.uni-freiburg.de/ HTTP/1.1
2|Host: www.informatik.uni-freiburg.de
3|
```

Caching

Client: `GET http://foo.com/icons/bar.jpg`

Cache:

- if resource not cached: get fresh from server

  ```
  GET /icons/bar.jpg
  Host: foo.com
  ```

- check if resource cached with ETag e at time t is *fresh*:
  1. server specified expiration time $t' > now$ **or**
  2. query server for freshness (conditional GET):

     ```
     GET /icons/bar.jpg
     If-Modified-Since: t
     ```

     or
  3. query server for matching ETag (conditional GET)

     ```
     GET /icons/bar.jpg
     If-None-Match: e
     ```

  sometimes combination of 2 and 3
- return fresh resource to client

Cache-Control:

"*to cache or not to cache*"

- `Age:` in seconds
- `Expires:` date and time
- `Last-Modified:` date and time
- `Cache-Control:` public, pricate, no-cache, no-store

How can you tell a cached response?

- `Via:` header field inserted by each cache

Content Negotiation

Client: asks for certain media types, languages, etc

all with quality rating q=r with $0 < r <= 1$

- `Accept:` media types
- `Accept-Charset:`
- `Accept-Encoding:`
- `Accept-Language:`

Server: offers different instances

- `TCN:` (transparent content negotiation)
  offers choice of different answers
- `Vary:` indicates further keys to cache

2.4 Authentifikation

- Zugriff oder Sperre nur für bestimmte *IP-Adressen*
- Programmatisch durch *Web-Formulare*
- HTTP-basierte Authentifikation:

  HTTP Basic ein challenge-response Protokoll

  HTTP Digest (schwach) verschlüsselte Version von **HTTP Basic**

HTTP Basic Authentication

Beim ersten Zugriff auf geschützte Seite:

```
GET /teaching/swt/2004/videos/1_Introduction_1.avi HTTP/1.1
```

Anwort:

```
HTTP/1.1 401 Authorization Required
Date:                Sun, 17 Apr 2005 20:14:49 GMT
Server:              Apache/2.0.49 (Unix) SVN/1.1.0 mod_ssl/2.0.49 OpenSSL/0.9.7d PH
WWW-Authenticate:    Basic realm="Videos swt 04"
Content-Length:      401
...
```

Nach Eingabe von Benutzername/Passwort wird die Anfrage mit zusätzlichem Header wiederholt:

```
Authorization: Basic R3Vlc3Q6MDR2aWRTV1Q=
```

. . . die Base64 Kodierung von *Benutzername*:*Passwort*, d.h. **Klartext!!!**

HTTP Digest verschlüsselt diesen String . . .

Inside WWW-Authenticate [RFC 2617]

Server: `WWW-Authenticate:` ⟨*challenge*⟩ (, ⟨*challenge*⟩)*

⟨*challenge*⟩	::=	⟨*auth-scheme*⟩ ⟨*auth-param*⟩(,⟨*auth-param*⟩)*
⟨*auth-scheme*⟩	::=	`Basic` | ...
⟨*auth-param*⟩	::=	`realm=`"⟨*string*⟩" | ...

Basic Authentication: `Authorization: Basic` ⟨*base64-user-pass*⟩

To receive authorization, the client sends the userid and password, separated by a single colon (":") character, within a base64 [7] encoded string in the credentials. ⇒ EXTREMELY UNSAFE

Digest Authentication: `Authorization: Digest` $\langle digest\text{-}challenge \rangle$

- challenge-response type:
  S: sends $\langle nonce \rangle$ and $\langle realm \rangle$
  C: sends H($\langle nonce \rangle + \langle username \rangle + \langle password \rangle$)
  H is a cryptographic hash function (e.g., MD5)
- avoids sending password in clear
- still cryptographically weak, open to a number of attacks

Proxy Authentication

similar, but with different header fields

- request: `GET http://...`
- response: `407 Proxy Authentication Required`
- response: `Proxy-Authenticate: ...`
- request: `Proxy-Authorization: ...`
- only for one "hop"

2.5 SSL und TLS

- HTTP versendet alle Daten im Klartext
- Für Geschäftsanwendungen gibt es weitere Sicherheitsanforderungen
  1. Vertraulichkeit
  2. Integrität
  3. Glaubwürdigkeit
  4. Unleugbarkeit
- SSL (Secure Session Layer) garantiert 1, 2 und teilweise 3
- Jetzt TLS (Transport Layer Security)
- HTTPS = HTTP + TLS

Eine HTTPS Verbindung

- Verbindungsaufbau
  - Verhandlung der Verschlüsselungstechnik
  - geheimer SSL-Sitzungsschlüssel
  - Server-Zertifikat wird überprüft
  - public-key Handshake
- Sicherer Kanal zwischen Client/Server
  - symmetrische Verschlüsselung

2.6 HTTP: Zusammenfassung

- Einfaches Protokoll für Client/Server Kommunikation
  nicht für streaming, publish/subscribe, P2P
- Zustandsloses Protokoll (robust, verlässlich)
  keine Vorkehrung für interaktive Anwendungen
  (Sitzungskonzept), z.B. webbasiertes Einkaufen
- Basierend auf unsicherem Protokoll TCP/IP
  Angriffe: Man-in-the-middle, Replay, Spoofing