7363 - Web-basierte Anwendungen

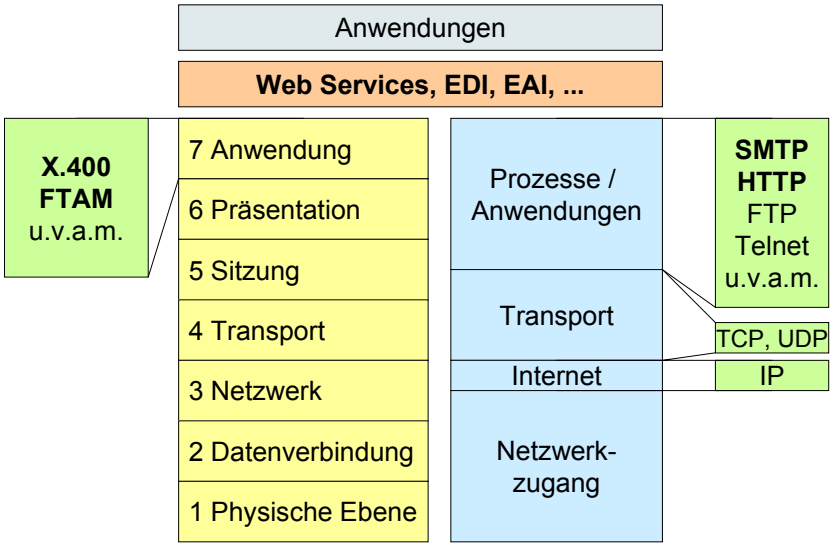
Eine Vertiefungsveranstaltung
mit Schwerpunkt auf XML-Technologien

Grundlagen:
Das OSI-Referenzmodell

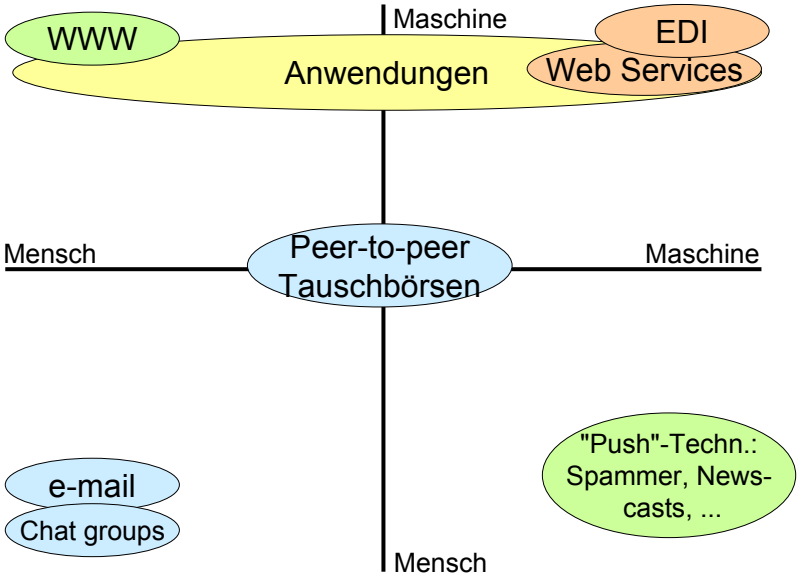
Erinnerung: OSI Referenzmodell

7 Anwendung
... und Schnittstellen zu A. Bsp.: FTP, Telnet, HTTP, SMTP, SNMP, ...
6 Präsentation
insb. Datencodierung (z.B. ASCII vs. Unicode vs. EBCDIC)
5 Sitzung
regelt Datenfluss, etwa: halb- oder full-duplex Verfahren
4 Transport
regelt Ende-zu-Ende Integrität übertragener Daten
fordert ggf. Pakete neu an, arrangiert Paketreihenfolge
3 Netzwerk
regelt das Routing (jenseits des eigenen LAN)
2 Datenverbindung
Fehlererkennung und -beseitigung auf Paketebene
Eingangspuffer
1 Physikalische Ebene
Erzeugung bzw. Verarbeitung von Bitstreams
Trotz des Namens: OHNE Austauschmedien

WS im OSI Referenzmodell und TCP/IP-Modell



Einordnung: Wer "redet" mit wem?



SMTP

Eine kleine Einführung in das

***S**imple **M**ail **T**ransfer **P**rotocol*

SMTP

- Aufgaben:
  - SMTP: Austausch von Textnachrichten (TCP-Port 25)
  - ESMTP: Erweiterung von SMTP
  - MIME: Ergänzung von (E)SMTP um "beliebige" Anhänge
  - S/MIME: MIME-Erweiterung zur Verschlüsselung
- RFCs
  - SMTP: RFC 2821, RFC 2822 (RFC 821, RFC 822)
  - ESMTP: RFC 1869
  - MIME: RFC 2045-2049, RFC 2442 u.v.a.
  - S/MIME: RFC 2311, 2312, 2630-2634, 2785, 2876, 2984, 3058

SMTP: C/S-Design mit einfachem Kommandosatz

- 10 4-Zeichen-Kommandos:
  - HELO
    - Anmeldung des Clients beim Server. Parameter: domain address
    - Server antwortet mit einigen Informationen
  - EHLO
    - Ersetzt HELO bei ESMTP. Client zeigt: Ich unterstütze ESMTP!
  - MAIL
    - Leitet die eigentliche Versendung einer email ein.
    - Absender als optionales Argument (From:)
  - RCPT
    - Benennt den/die Empfänger (TO:)
    - Unter ESMTP erweiterbar, je nach Server
  - DATA
    - Kennzeichnet den Start des eigentlichen Inhalts (nur 7-bit ASCII)

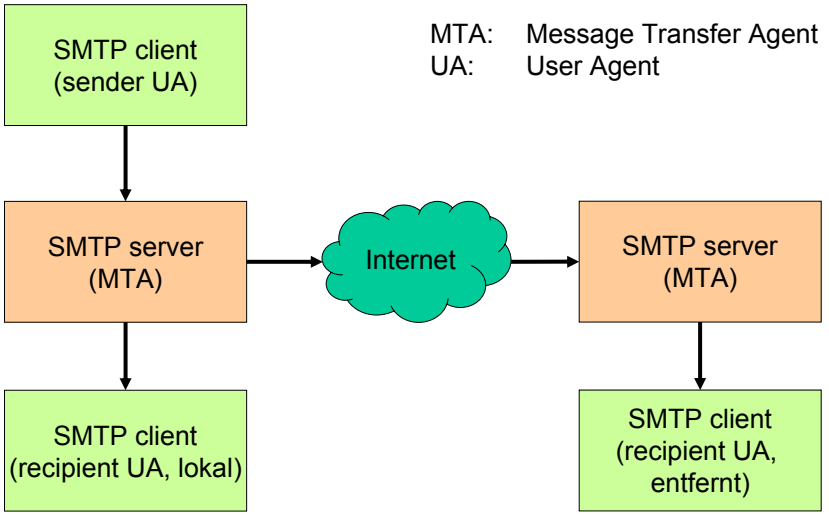
SMTP: C/S-Design mit einfachem Kommandosatz

- 10 4-Zeichen-Kommandos (Forts.):
  - DATA
    - Kennzeichnet den Start des eigentlichen Inhalts (nur 7-bit ASCII)
    - Ende-Kennzeichen: <CRLF>.<CRLF>
      (eine Zeile, die nur aus einem Punkt besteht)
  - RSET
    - Reset, Abbruch der aktuellen Übertragung
    - Der Server sollte bisher erhaltene Daten verwerfen
  - VRFY
    - Verify, Server soll prüfen, ob Empfänger ein User bzw. eine Mailbox und keine Verteilerliste ist.
  - EXPN
    - Expand, Server soll prüfen, ob Empfänger eine Verteilerliste und kein User bzw. Mailbox ist (Gegenstück zu VRFY).

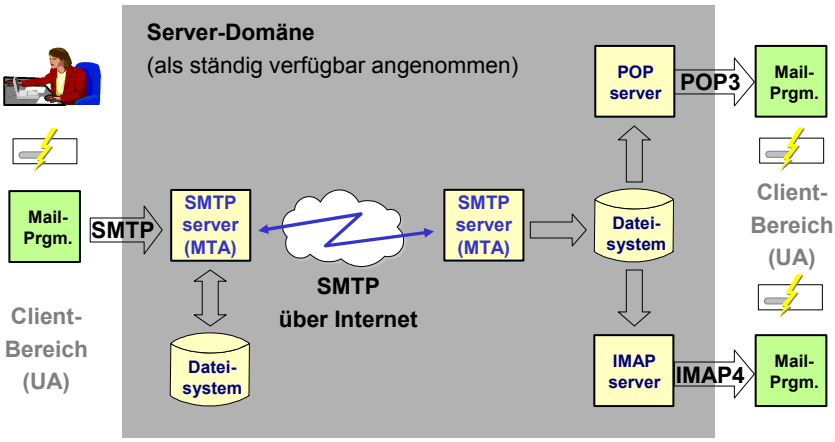
SMTP: C/S-Design mit einfachem Kommandosatz

- 10 4-Zeichen-Kommandos (Forts.):
  - HELP
    - Den Server um weitere Informationen bitten
    - HELP hat keinen Einfluss auf Übertragungen und darf jederzeit erfolgen
  - NOOP
    - Offenbar für Testzwecke. Server soll nur mit "OK" antworten.
  - QUIT
    - Server muss mit "OK" antworten und dann die Verbindung beenden.
- Veraltete Kommandos (vermeiden!)
  - SEND, SOML, SAML, TURN

SMTP: Routing



SMTP mit POP3 / IMAP4



SMTP: Statuscodes eines Servers

- Generell:
  - Der SMTP-Server antwortet mit dreistelligen Statuscodes
  - Vollständige Liste: Siehe RFC 821 (ESMTP)
- Erste Ziffer:
  - 2: Erfolgreich ausgeführt
  - 3: Anfrage verstanden, weitere Informationen erforderlich
  - 4: Fehler (temporär) Erneute Versuche könnten gelingen
  - 5: Fehler (permanent)

SMTP: Statuscodes eines Servers

- Zweite Ziffer:
  - 0: Syntaxfehler
  - 1: Informative Antwort, z.B. auf "HELP"
  - 2: Nimmt Bezug auf Verbindungsstatus
  - 3: nicht spezifiziert
  - 4: nicht spezifiziert
  - 5: Nimm Bezug auf das Mail-System insgesamt bzw. auf den Mailserver selbst.
- Dritte Ziffer:
  - Für weitere Verfeinerungen, vgl. RFC 821

SMTP: Statuscodes eines Servers

- Einige wichtige Statuscodes:
  - 214 Help message
  - 220 Service is ready
  - 221 Closing connection
  - 250 Requested action is okay
  - 251 User not local, forwarding message to <path>
  - 354 Start message input
  - 421 Requested service not available
  - 450 Requested action not taken, mailbox not available (may be busy)
  - 451 Requested action aborted, local processing error
  - 452 Requested action not taken, insufficient system storage

SMTP: Statuscodes eines Servers

- Einschränkungen
  - Reine SMTP-Nachrichten lassen nur ASCII-Zeichen zu
  - Nachrichten werden unverschlüsselt übertragen
  - End-to-End Verbindungen erforderlich (MTA-zu-MTA)
  - Freie Fahrt für Spam - jeder SMTP-Server steht "offen"
- Lösungen, Weiterentwicklungen
  - uuencode / uudecode, später: MIME
  - S/MIME, PGP; SSL zwischen Client und MTA
  - Relay Agents, DNS: "MX records"
  - Authentifizierung, Beschränkung auf Domain-Mitglieder

SMTP: Beispiel-Session, via Telnet

```
werntges@lx1-01(~)$ telnet smtp 25
Trying 195.72.96.97...
Connected to horus.informatik.fh-wiesbaden.de.
Escape character is '^]'.
220 stud.informatik.fh-wiesbaden.de ESMTP Sendmail 8.12.9-
   091003/8.12.9; Mon, 22 Mar 2004 14:57:06 +0100
HELP
214-2.0.0 This is sendmail version 8.12.9-091003
214-2.0.0 Topics:
214-2.0.0       HELO    EHLO    MAIL    RCPT    DATA
214-2.0.0       RSET    NOOP    QUIT    HELP    VRFY
214-2.0.0       EXPN    VERB    ETRN    DSN     AUTH
214-2.0.0       STARTTLS
214-2.0.0 For more info use "HELP <topic>".
214-2.0.0 To report bugs in the implementation send email to
214-2.0.0       sendmail-bugs@sendmail.org.
214-2.0.0 For local information send email to Postmaster at
   your site.
214 2.0.0 End of HELP info
NOOP
250 2.0.0 OK
```


SMTP: Beispiel-Session, via Telnet

```
mail FROM: werntges@informatik.fh-wiesbaden.de
503 5.0.0 Polite people say HELO first
HELO
501 5.0.0 HELO requires domain address
HELO lx1-01
250 stud.informatik.fh-wiesbaden.de Hello lx1-01.cs.fh-wiesbaden.de
   [172.25.81.101], pleased to meet you
MAIL FROM: werntges@informatik.fh-wiesbaden.de
250 2.1.0 werntges@informatik.fh-wiesbaden.de... Sender ok
RCPT TO: werntges@informatik.fh-wiesbaden.de
250 2.1.5 werntges@informatik.fh-wiesbaden.de... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
SUBJECT: Kleine Test-Nachricht

Erste Zeile
Zweite Zeile
.
250 2.0.0 i2MDv6qo031870 Message accepted for delivery
QUIT
221 2.0.0 stud.informatik.fh-wiesbaden.de closing connection
Connection closed by foreign host.
werntges@lx1-01(~)$
```

SMTP: Beschränkungen

- User name
  - 64 Zeichen
- Nomain name / number
  - 255
- MTA Path
  - 255, incl. Interpunktion und Zeichentrenner
- Command / Reply / Text line
  - 512 / 512 / 1000 (in der Praxis oft <= 76!)
- Max. Anzahl "Recipients":
  - 100 (also: Limit für große Verteiler!)
- RFC 2048 lesen!
  - Viele interessante *praktische* Einschränkungen

SMTP: MIME

- Eigenschaften
  - Gestattet auch andere email-Inhalte als reinen Text
  - Identifiziert die Art des Inhalts
  - Auch mehrere unabhängige Teile in einer email möglich (z.B. Anschreiben als Text + Anlagen)
  - Ermöglicht auch Texte mit non-ASCII-Zeichen, z.B. ÄÖÜß
- Methode
  - SMTP-Erweiterung: Konventionen über neue Kopfzeilen
  - Codierung von non-ASCII-Zeichen als Textstrings
  - Pflege von "MIME-Typen" zur Beschreibung der Inhalte
- Relevanz
  - Über SMTP hinaus! Vorbild z.B. für HTTP.

SMTP: MIME RFCs

- RFC 2045
  - Beschreibt die verschiedenen *header*, die die Struktur von MIME festlegen.
- RFC 2046
  - Legt die allgemeine Struktur der MIME Medientypen fest und gibt einen ersten Satz solcher Typen vor.
- RFC 2047
  - Erweitert RFC 822 um die Möglichkeit der Verwendung von non-ASCII-Zeichen in header-Feldern
- RFC 2048
  - IANA Registrierungsprozeduren für verschiedene MIME-bezogene Einrichtungen
- RFC 2049
  - *MIME conformance criteria, examples, bibliography*

SMTP: MIME-Header

- **MIME-Version**
  - Normalerweise `1.0`
- **Content-Type** (mit Parametern)
  - Standardmäßig (default) `text/plain; charset=us-ascii`
  - Andere Beispiele: `application/pdf; name="myfile.pdf"`
    `multipart/mixed; boundary="----12345----"`
- **Content-Transfer-Encoding**
  - Typischerweise `base64` bei Binärdaten. Vgl. XML Schema-Datentyp!
- Content-Disposition
  - Bsp.: `inline; filename="somefile.dat"`
- Content-Description
  - Klartext zur Beschreibung des Inhalts, insb. bei Audio- oder Videodaten
- Content-ID
  - Weltweit eindeutiger Identifizierungsstring zum gegebenen Inhalt

SMTP: Die 7 MIME *content types*

- text
  - Das MIME-Objekt enthält unformatierten Text
- multipart
  - ... enthält mehrere Teile mit jeweils eigenen Datentypen
- message
  - ... enthält eine gekapselte Nachricht oder einen Nachrichtenteil (z.B. geeignet zum empfängerseitigen Zusammenbau)
- image
  - ... enthält Grafikdaten, z.B. im JPEG- oder GIF-Format
- video
  - ... enthält Bewegtbilddaten, typischerweise im MPEG-Format
- audio
  - ... enthält Audiodaten
- application
  - "catch-all", z.B. für PDF- oder MS-Office-Daten

SMTP: Erweiterte MIME *content types*

- Nicht standardisierte *content types*
  - sind grundsätzlich möglich
  - Typenbezeichner müssen ggf. mit "X-" beginnen
  - müssen zw. Sender und Empfänger(n) abgestimmt sein
- Inzwischen vorhandener 8. Fall: "model"
- Subtypes
  - Zahlreich! Bedeutung ist festgelegt
  - On-line zeigen: Registrierung und vollständige Liste unter `http://www.iana.org/assignments/media-types/`
- Parameter
  - Die MIME *media types* können auch Parameter erhalten.
  - Hier sind einige zulässige Werte zu finden: `http://www.iana.org/assignments/media-types-parameters`

SMTP: MIME *content transfer encoding*

- Eine Erweiterung zu RFC 821. Mögliche Werte:
  - 7bit, quoted-printable
  - base64
  - 8bit, binary, x-token
- In der Praxis wichtig:
  - 7bit
    - Der abwärtskompatible default-Fall
  - quoted-printable
    - Zeichen dürfen durch ihren Hex-Wert ersetzt werden: =xx
    - Zeilenlänge auf 76 begrenzt; längere durch "=" am Ende bildbar
    - Beispiel: A ist =41, blank ist =30, CRLF ist =0D=0A.
    - Kompromiss zwischen Lesbarkeit und "Transportschutz"
  - base64:
    - Sehr häufig genutzt, für alle "nicht-Text" Fälle. Konvertieren üben!

SMTP: MIME-Beispiele

- RFC 2049, Appendix A
  - http://www.ietf.org/rfc/rfc2049.txt
  - On-line Demo / Diskussion
- FB-Projekt "LV-Listen"
  - Versand der generierten Excel-Dateien per Ruby-Script an hinterlegten Verteiler
  - Demonstriert "multipart/mixed" mit Textbody und Binäranhang (PDF).

SMTP: MIME-fähige User Agents

- Content type: application/octet-stream
  - "catch all"-Fall. UA soll i.d.R. nach Pfad/Dateinamen fragen und Inhalt als Datei speichern

- Andere *application subtypes*
  - Wenn der Start einer passenden Anwendung und die direkte Übergabe der Daten vom UA an die Anwendung gewünscht ist, muss der MIME-Typ mit der jeweiligen lokalen (!) Anwendung verknüpft werden.
  - Moderne Browser unterstützen dies.
    Demo:
    Mozilla, "Helper Applications", Excel-Fall (mimedemo-excel.eml)

HTTP

Hypertext Transfer Protocol

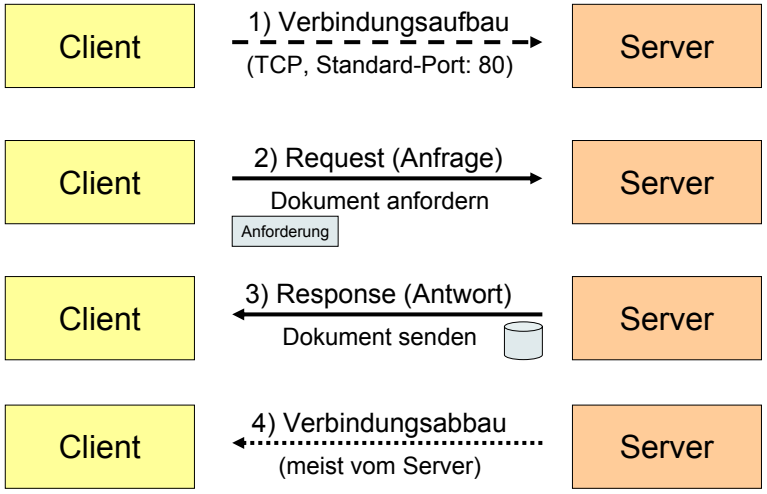
HTTP

- Warum HTTP ausführlich besprechen?
  - Grundlage praktisch aller Web-basierten Anwendungen
  - Grundlage der meisten Web Services
  - Hintergrundwissen zum erfolgreichen Konfigurieren eines Web-Servers (im Kurs: Apache)
  - Voraussetzung für CGI-Programmierung
  - Zeigt Möglichkeiten, Grenzen, Risiken auf
- Daher:
  - Solide Kenntnisse von HTTP sind notwendig für alle, die mehr als nur "kochrezept-artig" entwickeln, sondern verstehen wollen, was passiert!

HTTP 0.9

- oder: 1991 - Der Anfang des WWW
  - Tim Berners-Lee benötigte 1991 ein C/S-Protokoll zum Abholen von HTML-Seiten.
  - Er definierte dazu eines der einfachsten Protokolle: HTTP (0.9)
- HTTP 0.9
  - Client stellt Verbindung zum Server her
    - i.d.R. mit TCP/IP, aber auch andere verbindungsorientierte Dienste wie DECnet oder ISO TP4 möglich.
  - Client fordert Ressource (HTML-Seite) an
    - Einzige Methode von HTTP 0.9: `GET resource <crlf>`
  - Server schreibt (zeilenweise, <= 80 Zeichen/Zeile) HTML zurück
  - Client liest die Daten schnellstmöglich (Server sollte nicht warten)
  - Server beendet Verbindung

HTTP: Ein reines C/S-Protokoll



Request/Response: Der Server wird nur nach Client-Requests aktiv!

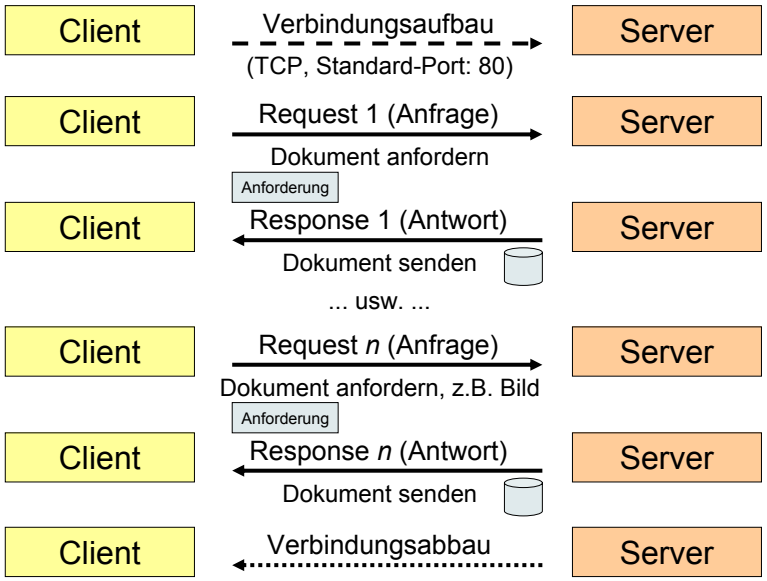
HTTP 0.9

- Fehlerbehandlung
  - Rudimentär!
  - Verbindungsebene:
    - Timeouts, nach ca. 15 sec., auch durch Server
    - Client darf Übertragung vorzeitig beenden durch Beenden der Verbindg.
  - Anwendungsebene:
    - nur HTML-Fehlertexte, keine Fehlercodes
    - Fehlerbedingungen (wie: "Dokument nicht gefunden") sind für UA nicht erkennbar, nur für menschliche Nutzer lesbar
- Quelle:
  - http://www.w3.org/Protocols/HTTP/AsImplemented.html
- Demo 1, Basis "Cygwin / telnet"
  - GET /no_such_file ==> Fehlertext in HTML
  - GET /index.html ==> HTML-Ausgabe

HTTP 1.0

- 1992...1996 - Multimedia und Interaktivität halten Einzug
  - Die einfache Beziehung "1Datei = 1 Anzeigeseite" gilt nicht mehr:
    - Eingebettete Bilder u.a. Multimedia-Quellen erfordern mehrere Server-Zugriffe pro Seite;
    - neben HTML gibt es nun auch andere Inhaltsarten - und die Notwendigkeit, diese zu unterscheiden.
  - Nicht nur Abfragen - auch Publizieren von HTML-Dokumenten!
    - Allgemeiner: Clientseitig erfasste Daten wie HTML-Dateien, aber auch Formulare oder lokale Dateiinhalte sollen auf den Server gelangen.
- Neu:
  - Erweiterter Methodensatz
  - Trennung des Inhalts:
    - Header, Leerzeile und Message (MIME-konform)
  - Uploads ermöglicht durch Message-Teil auch in Requests
  - Fehlercodes in Server-Antworten
  - Optional: Mehrere Requests/Antworten pro Verbindung

HTTP: Mehrere Transaktionen pro Verbindung



HTTP 1.0

- **Requests**

```
Request          = SimpleRequest | FullRequest
SimpleRequest    = GET <uri> CrLf
FullRequest      = Method URI ProtocolVersion CrLf
                   [*<HTRQ Header>]
                   [<CrLf> <data>]
<Method>         = <InitialAlpha>
ProtocolVersion  = HTTP/1.0
uri              = <as defined in URL spec>
<HTRQ Header>    = <Fieldname> : <Value> <CrLf>
<data>           = MIME-conforming-message
```

- Bemerkungen
  - Die Notation ist leider nicht einheitlich: <uri> vs. URI, <CrLf> vs. CrLf
  - URI Strings: "URI encoding"! Oktets sind durch '%xx' zu ersetzen (xx=ASCII-Code in hex), insb. für 0x0-0x1f, space, 0x80 - 0x9f

HTTP 1.0

- Fehlerbehandlung
  - Analog zu HTTP 0.9
  - Zusätzlich: Fehlercode und -Text in *server response header*
  - Beispiel (1. Zeile einer Server-Antwort):

```
HTTP/1.0 404 Not found
```

  - Fehlercodes (Details bitte nachlesen!):
    - 3-stellige Codes, ähnlich wie bei SMTP. Erste Ziffer kategorisiert:
    - 1 = Informative Meldungen, 2 = Erfolgsmeldungen,
      3 = Request weitergeleitet, weitere Angaben notwendig,
      4 = Request unvollständig/fehlerhaft, 5 = Server-Fehler
- Quellen:
  - http://www.w3.org/Protocols/HTTP/HTTP2.html (Vorläufer, 1992)
  - http://www.ietf.org/rfc/rfc1945.txt (HTTP 1.0, 1996)
- Demo 2, Basis "Cygwin / telnet"
  - GET /no_such_file HTTP/1.0 ==> Header + Fehlertext in HTML
  - GET /index.html HTTP/1.0 ==> Header + HTML-Ausgabe

HTTP 1.0

- **Neue Methoden**
  - HEAD
    - Wirkung wie GET, liefert aber nur Header der Antwort
  - PUT
    - Beiliegendes Dokument soll unter gegebenem URL abgelegt werden
    - Dokument muss bereits existieren, ggf. POST nehmen
  - POST
    - Beiliegendes Dokument ist neu und soll mit ggf. gegebener ID unterhalb des mitgeführten URL vom Server angelegt werden. Server antwortet mit neuem URL, unter dem er das Dokument führt.
  - DELETE
    - Loescht serverseitiges Dokument unter gegebenem URL
  - LINK
    - Unglückliche Namensgebung für die Verknüpfung des Headers mit einem bereits vorhandenden Dokument des Servers, analog POST
  - UNLINK
    - Gegenstück zu LINK

HTTP 1.0

- **Methoden, die sich nicht durchsetzten**
  - CHECKIN, CHECKOUT
  - SEARCH
  - SHOWMETHOD
  - SPACEJUMP
  - TEXTSEARCH
- Bemerkungen
  - Offenbar strebte man mit HTTP 1.0 bereits Funktionen eines CMS (Content Management-Systems) an
  - Die Entwicklung verlief anders:
    - CGI-Programme realisierten in flexiblerer Weise diese und weitere Funktionen.
  - "Revival" der Idee der verteilten Autorenschaft als
    - eigener Standard: **WebDAV** (Web Distributed Authoring and Versioning) RFCs 2518 (Feb. 1999), 3253 (März 2003), 3648 (Dez. 2003)
    - Ward Cunningham's **Wiki**-Konzept: http://c2.com/cgi/wiki?WikiWikiWeb

HTTP 1.0

- **Neu ab V 1.0: Header**
- **Die 4 Header-Arten:**
  - Request-Header: Nur von *client requests* verwendet
  - Response-Header: Nur von *server responses* verwendet
  - Entitäts-Header: Info zum übertragenen Dokumentenformat
  - Allgemeine Header: Z.B. Angaben zur Zeit oder zur Verbindung
- Entwicklung der Headeranzahl mit der HTTP-Version:

HTTP-V.	Allgemeine	Request	Response	Entität
0.9	0	0	0	0
1.0	4	9	4	10
1.1	9 (-1+6)	19 (+10)	9 (+5)	10 (-3+3)

HTTP 1.0

- **Allgemeine Header**
  - **Vorbemerkungen:**
    - Fast alle Header sind optional
    - Sie werden hier nur kurz vorgestellt; Einzelheiten ggf. nachlesen!
- Connection: *Optionen*
  - Teilt dem *server* mit, die TCP-Verbindung nach Ende der Transaktion
    - zu beenden ("close", default unter HTTP 1.0), oder
    - offen zu halten ("keep alive", default unter HTTP 1.1)
- Date: *Datumsformat*
  - Es gibt drei zulässige Datumsformate:
    - Nach RFC 1123, String fester Länge; bevorzugtes Format!
      Beispiel: `Fri, 09 Apr 2004 16:23:45 GMT`
    - Nach RFC 1036
      Beispiel: `Friday, 09-Apr-04 16:23:45 GMT`
    - ANSI-C asctime()-Format (ohne Zeitzone - vermeiden!)
      Beispiel: `Fri Apr 9 18:23:45 2004`

HTTP 1.0

- Pragma: *no-cache*
  - Teilt *caching proxies* auf dem Weg zum *server* mit, die Anfrage weiterzuleiten und nicht aus dem *cache* zu bedienen.
  - Beispiel: `Pragma: no-cache`
  - HTTP 1.0-Feature, unter HTTP 1.1 noch verfügbar; wird nicht mehr weiter gepflegt.
  - "no-cache" ist der einzige zulässige Wert dieses Headers

- MIME-Version: *String*
  - Reines HTTP 1.0-Feature, in HTTP 1.1 nicht mehr verwendet.
  - Hier nicht näher besprochen

HTTP 1.0

- **Request-Header**
  - Sie werden nur vom *client* an den *server* gesendet

- From: *E-Mail-Adresse*
  - Wert ist eine Internet Mail-Adresse, z.B. für serverseitige Rückfragen an den Anfragenden. - Surfen Sie eigentlich anonym? Wirklich??
- Accept: *Typ/Subtyp [q=Qualitätswert]*
  - Enthält eine komma-separierte Liste von (MIME) *Content types*
  - Default: `Accept: text/plain, text/html`
  - 'Accept' darf mehrfach vorkommen
  - Typ-Parameter können mit ';' angefügt werden. Bsp. 2:
    `Accept: text/x-dvi; q=.8; mbx=100000; mxt=5.0, text/x-c`
  - Wildcards: Einfache Aussage zur Akzeptanz aller Typen/Untertypen
    - Beispiel: `Accept: audio/*`

HTTP 1.0

- Accept-Charset: *Zeichensatz [q=Qualitätswert]*
  - Gibt die vom *client* bevorzugten Zeichensätze an
  - Formal ähnlich zu 'Accept'
  - Beispiel:
    ```
    Accept-Charset: ISO-8859-15, ISO-8859-1; q=0.8
    ```
- Accept-Encoding: *Codierungstypen*
  - Gibt die vom *client* verstandenen *content transfer encodings* an
  - Formal ähnlich zu 'Accept'
  - Beispiel: `Accept-Encoding: x-compress, x-gzip`
- Accept-Language: *Sprache [q=Qualitätswert]*
  - Die bevorzugte(n) Sprache(n) im Antwortdokument, codiert nach ISO
  - Beispiel: `Accept-Language: de-DE, en-US`

HTTP 1.0

- Authorization: *Schema Angaben*
  - Übermittelt die Autorisierung des *client,* in HTTP 1.0 nur bez. der Autorisierungsart "BASIC"), i.d.R. als Reaktion auf eine Aufforderung durch den *server* (siehe Response-Header "WWW-Authenticate").
  - Enthält "BASIC", dann *username:password* in base64-Codierung
  - Beispiel:
    ```
    Authorization: BASIC d2VibWFzdGVy0npycW1hNHY=
    ```
  - Übung: Decodierung des base64-Strings
  - Sicherheitsdiskussion!
- If-Modified-Since: *Datum*
  - Angeforderte URL-Daten nur übertragen, wenn sie nach dem angegebenen Datum verändert wurden (sonst: Code 304).
  - Beispiel:
    ```
    If-Modified-Since: Fri, 09 Apr 2004 15:55:23 GMT
    ```
  - Unterstützt client-seitiges Caching. Format muss das in "Date" sein!

HTTP 1.0

- Referer: *URL*
  - Teilt dem *server* mit, von welchem URL man auf die in diesem *request* angeforderte Seite geleitet wurde.
  - Bem.: Ein Tippfehler in der Spezifikation, den man nicht mehr korrigierte (die engl. Vokabel schreibt sich "referrer").
  - Soll nicht mitgeschickt werden, wenn der URL manuell eingegeben wurde oder aus einer Linksammlung stammt.
  - Publikumsfrage: Wozu nützlich?
  - Sicherheitsdiskussion: Möglicher Missbrauch?

- User-Agent: *String*
  - Teilt dem Server mit, welches Programm die Anfrage stellt
  - Bietet dem Server die Möglichkeit, UA-abhängig zu antworten
  - Keine verlässliche Information. Bsp: Der IE von MS identifiziert sich als "Mozilla"; auch Proxies könnten den Sinn vereiteln.

HTTP 1.0

- **Response-Header**
  - Sie werden nur vom *server* an den *client* gesendet

- Location: *URL*
  - Wert ist eine Internet Mail-Adresse, z.B. für serverseitige Rückfragen an den Anfragenden. - Surfen Sie eigentlich anonym? Wirklich??

- Retry-After: *Datum*|*Sekunden*
  - Wie der Name schon sagt...
  - Tritt zusammen mit Response-Code 503 "Service Unavailable" auf.
  - Wenn Datumsformat, dann gemäß RFC 1123 (vgl. "Datum")
  - Beispiele:
```
Retry-After: 3600
Retry-After: Fri, 09 Apr 2004 16:23:45 GMT
```

HTTP 1.0

- Server: *String*
  - Identifiziert die Server-Software gegenüber dem Client
  - Beispiel:
    ```
    Server: Apache/1.3.26 (Unix) Debian GNU/Linux
    ```
  - Grundlage von WWW-Statistiken. Diskussion: Sicherheitsrelevant?

- WWW-Authenticate: *Schema Bereich*
  - Fordert den *client* auf, sich für den genannten Bereich mit dem genannten Schema zu authentifizieren
  - Tritt zusammen mit Response-Code 401 ("Unauthorized") auf
  - Vergleiche Request-Header "Authorization"
  - Schema BASIC:
    - Benutzername und Kennwort zu senden
    - Andere Schemata: Digest; NTLM (Windows-Server)
  - Bereich:
    - Ein Server kann viele Bereiche unterhalten
  - Beispiel: `WWW-Authenticate: BASIC realm="admin"`

HTTP 1.0

- **Entitäts-Header**
  - Sie beschreiben Eigenschaften übertragener Dokumente
  - Sie erscheinen daher nur, wenn auf den Header noch Daten folgen.

HTTP 1.0

- Allow: *Methoden*
  - Schränkt die Methoden für einen bestimmten URL ein
  - Tritt zusammen mit Response-Code 405 "Method not allowed" auf.
  - Beispiel: `Allow: GET, HEAD`
- Content-Encoding: *Codierungsschemata*
  - Gibt das/die verwendete(n) Codierungschema(ta) an, das/die auf den Body angewendet wurde(n)
  - Bei mehreren: In der angewendeten Reihenfolge anzugeben
  - Zulässige Werte:
    `x-compress, x-gzip / compress, gzip, deflate`
- Content-Language: *Sprachen*
  - Gibt die Sprache(n) an, für die der übertragene Body gedacht ist.
  - Beispiel: `Content-Language: fr`

HTTP 1.0

- Content-Length: *n*
  - Die Länge des enthaltenen Dokuments (body), in Oktetts
  - Dieser Header ist nicht immer verfügbar
    - Insbesondere bei dynamisch generierten Dokumenten
  - Beispiel: `Content-Length: 12345`
- Content-Type: *Typ/Subtyp*
  - Beschreibt den Medientyp und -subtyp des Body.
  - Analog zu Header "Accept"
  - Beispiel: `Content-Type: text/html`
- Expires: *Datum*
  - Hilfe an den *client* für dessen *cache management*
  - Weitere Anfragen zum gleichen Dokument vor dem angegebenen Datum können aus dessen Cache beantwortet werden
  - Beispiel: `Expires: Fri, 09 Apr 2004 18:00:00 GMT`

HTTP 1.0

- Last-Modified: *Datum*
  - Datum der letzten Änderung des Dokuments
  - Ein *client* sollte das Dokument neu laden, wenn die Version in seinem *cache* älter ist.
  - Formate: Analog zu "Date"
  - Beispiel: `Last-Modified: Tue, 13 Apr 2004 09:14:35 GMT`

- Link, Title, URL
  - Nur von HTTP 1.0 verwendet, in HTTP 1.1 nicht mehr zulässig
  - Hier nicht weiter vorgestellt. Vermeiden!

HTTP 1.0

- Mehr zum Header "Content-Type"

`Content-Type: text/html`

  - Der übliche Typ bei der Rückgabe von HTML-Dokumenten

`Content-Type: text/plain`

  - Der Basistyp, für reine ASCII-Dokumente wie z.B. die RFC-Dokumente.

`Content-Type: application/x-www-form-urlencoded`

  - Bekannter Typ für Rückgaben von Formularinhalten, insb. im Kontext der CGI-Programmierung.
  - Folgende Zeichen müssen nicht codiert werden:
    - Buchstaben: A - Z, a - z
    - Ziffern: 0 - 9
    - Folgende Zeichen: - _ . ! ~ * ' ( )
  - Alle anderen werden als %xx codiert, mit xx = ASCII-Wert (hex)
  - Leerzeichen werden auch als + codiert, %20 bleibt aber erlaubt.

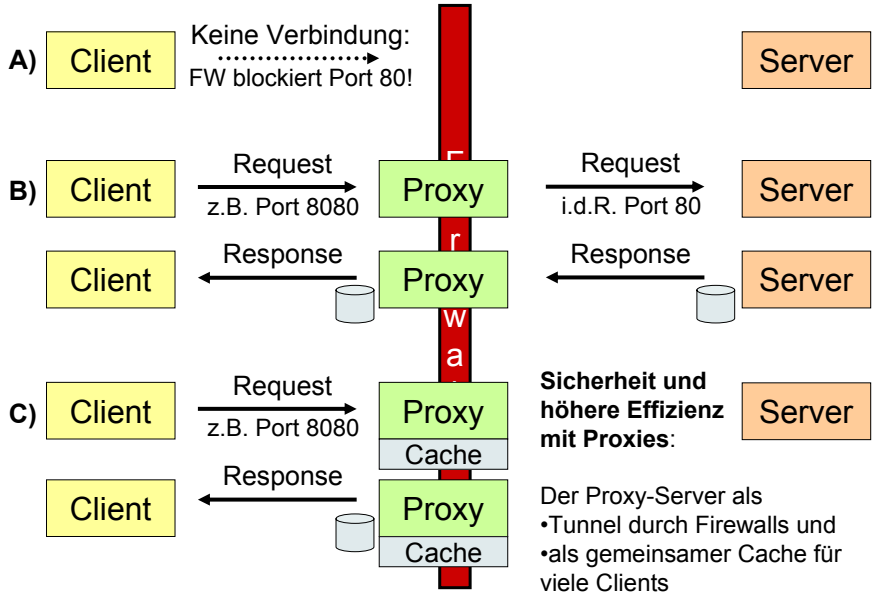
HTTP 1.1

- **1997 HTTP 1.1 - Reifung und Effizienzaspekte**
  - Caching: Mehr Möglichkeiten
  - Optimierungen: Auch Teildokumente abrufbar
  - Verbindung: "keep-alive" nun Standard
  - Proxies: Besseres Zusammenspiel mit mehrstufigen P.
  - Virtuelle Hosts: Konsequente Unterstützung
  - "Chunking": Mehrere Nachrichtenteile
  - Präzisere Spezifikationen ==> bessere Interoperabilität
- Neu:
  - Weitere Methoden
  - Neue Header; Abschaffung einiger HTTP 1.0-Header
  - Mehrere Requests/Antworten pro Verbindung nun Regelfall
  - "Host"-Header nun Pflicht in *client requests*

HTTP 1.1

- Entwickung des Spezifikationsumfangs:
  - 1990/1991: HTTP 0.9 wenige Seiten
  - 1996: HTTP 1.0 RFC 1945 60 Seiten
  - 1997: HTTP 1.1 RFC 2068 162 Seiten
  - 1999: HTTP 1.1 Upd. RFC 2616 176 Seiten
  - 1995/97 - 1999: HTTP-NG (www.w3.org/Protocols/HTTP-NG/)
- HTTP-NG
  - Die Beschränkung von HTTP auf Übertragung von 7-bit Zeichen ist grundsätzlich überflüssig und ineffizient.
  - Umcodierungen (base64) kosten unnötige Ressourcen, insbesondere auf stark frequentierten Servern, Übertragung kostet Bandbreite.
  - Ansatz: HTTP "Next Generation", (1995)1997-1999
    - Vereinfachung durch Modularisierung
    - Grundlage: Effizienter, automatisierter *message transfer*
    - Verzicht auf Kompatibilität mit HTTP 1.x
  - Ergebnis: HTTP-NG konnte sich nicht durchsetzen!

HTTP Proxy-Server



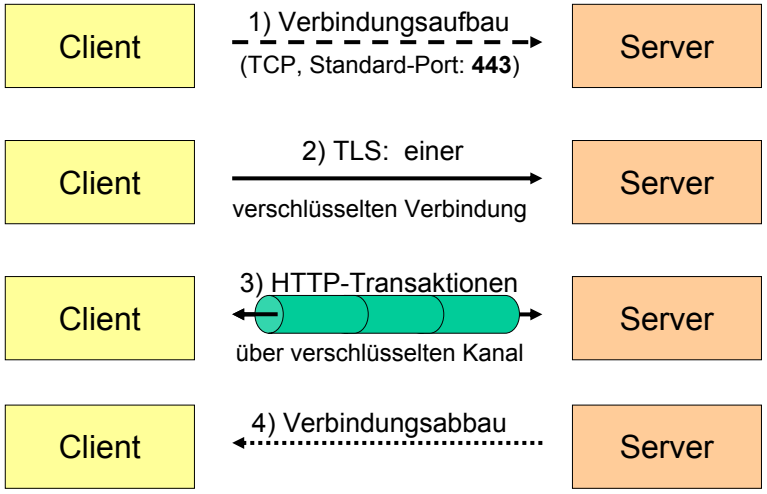
Sicherheit und höhere Effizienz mit Proxies:

Der Proxy-Server als
•Tunnel durch Firewalls und
•als gemeinsamer Cache für viele Clients

HTTP Proxy-Server

- **Bemerkungen**
  - Für einen Server sind Clients hinter Proxy-Servern nicht mehr unterscheidbar:
    - Ein Proxy kann Request-Header verändern (und so z.B. die Art des Clients verschleiern)
    - Selbst die IP ist kein Mittel mehr zur auch nur kurzfristigen Wiedererkennung eines Clients (-> Session management)
  - Proxies erschweren Debugging
  - Proxies müssen von auf HTTP aufbauenden Protokollen wie XML-RPC und SOAP (und deren Werkzeuge) beachtet werden.
  - Achten Sie unter Unix/Linux auf die Umgebungsvariable "**http_proxy**".
    Sie sollte bei uns den Wert "http://www-cache:8080" annehmen.
  - Gelegentlich ist analog "**HTTP_PROXY**" zu setzen, je nach eingesetzter Software.

HTTP mit TLS: Verschlüsselte Transaktionen



Benutzung von Port 443 statt 80 (bzw. "https" statt "http" im URL)

HTTP mit TLS (Transport Layer Security)

- **Bemerkungen**
  - Transparenz
    - Die TLS-Schicht ist OSI-Schicht 4/5, also unterhalb von HTTP
    - Ist die Verbindung erst einmal auf Verschlüsselung umgestellt, werden über ihr normale HTTP-Transaktionen ausgetauscht.
  - Verbindung
    - Vorsicht vor Timeouts - erneuter TLS-Aufbau erforderlich, wenn die TCP/IP-Verbindung unterbrochen wird.
  - Sicherheit
    - Senden Sie Kennwörter nie ohne SSL-Verbindung über das Internet. Bsp.: Nur "Sicheres Einloggen (SSL)" bei eBay nutzen!
  - Secure-HTTP (S-HTTP, RFC
    - Hierbei handelt es sich um eine Protokollerweiterung zu HTTP, die Verschlüsselung innerhalb HTTP anstrebte.
    - Hat sich nicht durchgesetzt. Nicht mit "https" / SSL verwechseln!

HTTP mit TLS (Transport Layer Security)

- **TLS**
  - Anfänge gesetzt durch Netscape's Secure Socket Layer
    - SSL, meist SSL-2, zuletzt 1996 als Version SSL 3.0 diskutiert
    - Quelle: http://wp.netscape.com/eng/ssl3/draft302.txt
  - 1999: TLS, in RFC 2246
  - 2000: Upgrading to TLS Within HTTP/1.1, RFC 2817
  - 2000: HTTP over TLS, RFC 2818
  - 2003: TLS-Update in RFC 3546
- TLS via Proxy-Server
  - Eigenes Thema, hier nicht weiter vertieft.
    - HTTP/1.1 hier mit besserer Unterstützung als HTTP/1.0
- Werkzeug für eigene Vorhaben:
  - www.openssl.org

HTTP 1.1

- **Neue Methoden**
  - OPTIONS
    - Weitere Optionen für den URL anfordern; "*" für den gesamten Server
    - Antwort ist Liste der zulässigen Methoden und Optionen für die Ressourcen (ALLOW für einzelne Dokumente, PUBLIC für den Server)
  - CONNECT
    - Einleitung einer HTTPS-Verbindung über einen Proxy.
    - Siehe Seite zu "Proxies"
  - TRACE
    - Ermöglicht zu erkennen, wie eine Nachricht verändert wird, wenn sie durch eine Kette von Proxy-Servern läuft.
    - Zusammenspiel mit neuen Headern "Max-Forwards" und "Via"
    - Analogie zum Netzwerk-Utility "traceroute"

HTTP 1.1

- **Neue allgemeine Header**
  - **Vorbemerkungen:**
    - Die neuen HTTP 1.1-Header werden hier nur noch benannt
    - Einzelheiten ggf. nachlesen!

- Cache-Control: *Direktiven*
- Trailer: *Trailer-Header (bei "chunked messages")*
- Transfer-Encoding: *Codierungstyp (nur "chunked")*
- Upgrade: *Protokoll/Version*
- Via: *Protokoll Host*
- Warning*: Code Host String*

HTTP 1.1

- **Neue Request-Header**

- Expect: *Erwartung (an den Server)*
- Host: *Hostname[:Port]*
  - Pflicht-Header! Unterstützt "Multihome-Server"
- If-Match: *Entitäts-Tag*
- If-None-Match: *Entitäts-Tag*
- If-Range: (*Entitäts-Tag|Datum)*
- If-Unmodified-Since: *Datum*
- Max-Forwards: *n*
- Proxy-Authorization*: Angaben*
- Range*: Bytes=n-m*
- *TE: Transfercodierungen*

HTTP 1.1

- **Neue Response-Header**

- Accept-Ranges: *bytes|none*
- Age: *Sekunden*
- ETag: *Entitäts-Tag*
- Proxy-Authenticate: *Schema Bereich*
- Vary: *Header*

- **Neue Entitäts-Header**

- Content-Location: *URL* (falls mehrere Entitäten pro Dokument)
- Content-MD5: *Digest* (zur Identitätsprüfung beim Empfang)
- Content-Range: *bytes n-n/m* (Angaben zum Einfügen)

HTTP 1.1

- **Vor- und Nachteile persistenter Verbindungen**
  - Die Vorgabe "Connection: keep-alive" von HTTP 1.1 hat Vorteile, kann aber auch Probleme bereiten:

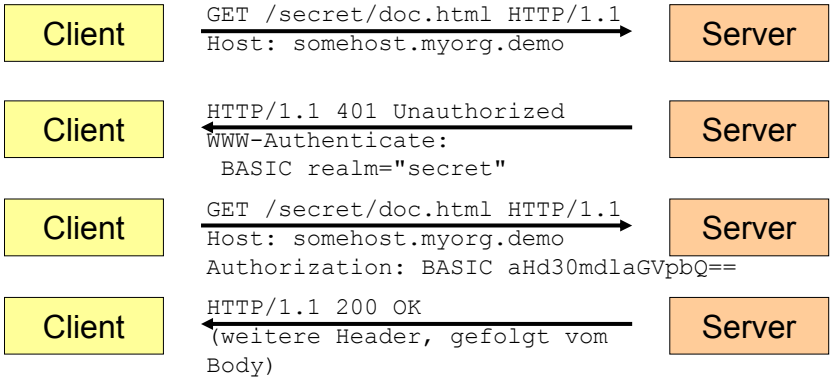
- Vorteile:
  - Effizientes Laden aller Ressourcen einer neuen Seite im Browser
    - HTML-Dokument + alle enthaltenen Bilder können innerhalb einer einzigen TCP/IP-Verbindung geladen werden
  - Andere nur verbindungsbezogene Zustände wie die Authentifizierung oder die TLS-Verschlüsselung sind seltener erforderlich.

- Nachteile:
  - Ressourcen-Blockaden auf Serverseite möglich
  - DoS-Angriffe erleichtert (DoS = Denial of Service)
  - Prozess-Overhead auf Serverseite (ein Prozess pro aktiver Verbindung erforderlich!)
  - Kompromiss: Multithreading?

HTTP 1.1

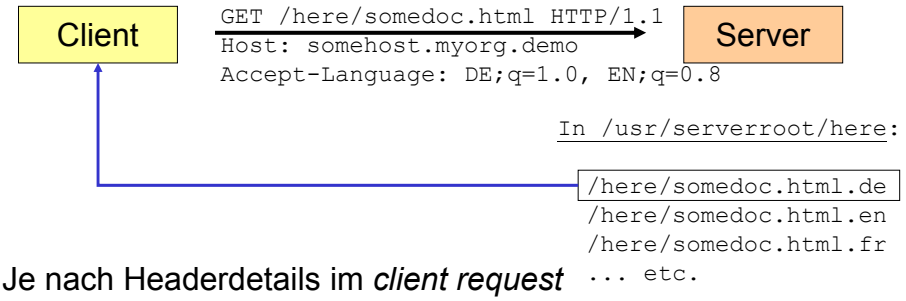
- **Effizienter Download mit HTTP 1.1**
  - Wie schon beschrieben: Mehrere Transaktionen pro Verbindung
  - Kompression der zu übertragenden Inhalte (optional) nun verbreiteter
  - Caching: Bessere Unterstützung für wirksame Caching-Strategien
  - Transfer auch von Teildokumenten ("Range") möglich!

HTTP: Autorisierung - Der Ablauf



Der Client provoziert mittels GET oder HEAD zunächst einen Serverfehler "401". Der Server fordert zur Autorisierung auf, unter Nennung des betroffenen Serverbereichs.
Der Client liefert z.B. das gewünschte Paar aus Benutzer und Kennwort (SSL nutzen!), worauf der Server die Daten bereitstellt. Komfortable Clients verwalten A.-Daten und vermeiden Benutzerabfragen - aber: Vorsicht!

HTTP: Trennung von URL und Dokument



Je nach Headerdetails im *client request* kann der HTTP-Server einen einzigen URL mit verschiedenen Dokumenten identifizieren.

Im Beispiel: Sprach-Präferenz nicht im URL erkennbar, sondern z.B. aus Browser-Einstellungen übernommen.

Herstellung des Zusammenhangs: In Serverkonfigurationen!

HTTP: Multihome-Server

- Idee:
  - Trennung von URL und eigentlichem Server!
- Vorbereitungen:
  - 1. DNS-Einträge, Beispiele
    - www.meine-domain.de ==> 100.200.30.40
    - www.deine-domain.de ==> 100.200.30.40 (gleich!)
  - 2. Multihome-Server unter IP 100.200.30.40
    - Dateisystem für Serverseiten, Beispiel
      /opt/www/homepages/kunde001/index.html,
      /opt/www/homepages/kunde005/index.html, etc.
    - Konfigurationsdaten des WWW-Servers dazu
      www.meine-domain.de ==> kunde005,
      www.deine-domain.de ==> kunde001, etc.

HTTP: Multihome-Server

- Die Folge
  - Abruf von http://www.meine-domain.de/index.html:
  - 1. DNS-Lookup ergibt: TCP/IP-Verbindung mit 100.200.30.40
  - 2. Client request:
    GET /index.html HTTP/1.1
    Host www.meine-domain.de
  - 3. Ergebnis:
    Dokument unter .../kunde005/index.html
- Wozu nützlich?
  - Effizientes Hosting vieler Homepages möglich
  - Server passt sich Clients besser an (sofern konfiguriert).

HTTP: Zustandsverwaltung (*session management*)

- Das Problem
  - HTTP ist ein "zustandsloses" Protokoll: Jede Transaktion ist in sich abgeschlossen; Server und Client "erinnern" sich auf Protokollebene nicht an frühere Transaktionen.
  - Anforderungen wie die Warenkorbfunktion eines Online-Shops erfordern aber Persistenz auf *session*-Ebene, d.h. über die Dauer einer ganzen Reihe von Transaktionen hinweg.
- Techniken für die Zustandsverwaltung
  - URL-Erweiterungen wie Query-Strings und *fragment identifier*
  - Versteckte Felder
  - Cookies

HTTP: Zustandsverwaltung (*session management*)

- URL-Erweiterungen
  - Grundsätzlich gibt es keine statischen Dokumente mehr:
    - Jede Anfrage wird von einem Programm auf Vorhandensein der identifizierenden URL-Erweiterung geprüft.
    - Fehlt diese, wird eine neue *session* unterstellt und eingeleitet.
    - Ist eine vorhanden, dient sie als Verbindung zwischen Transaktionen
- Vorteile
  - Mit jedem Client einsetzbar
- Nachteile
  - Serverlast
  - Programmieraufwand
  - URL kann leicht gefälscht werden
  - Unterbrechung einer *session* durch Verlassen des Servers

HTTP: Zustandsverwaltung (*session management*)

- Versteckte Felder
  - Erkennungssequenzen für eine *session* werden in (für den Anwender nicht sichbaren) versteckten Formularfeldern gespeichert.
- Vorteile
  - Mit jedem Client einsetzbar
  - Relativ performant
  - Leicht umsetzbar
- Nachteile
  - Funktioniert nur über eine Reihe aufeinanderfolgender Formulare
  - Unterbrechung einer *session* durch Verlassen des Servers

HTTP: Zustandsverwaltung (*session management*)

- Cookies
  - Eine ursprünglich von Netscape eingeführte Erweiterung zu HTTP:
  - Inzwischen weitgehend akzeptiert. Ein Standardisierungsansatz (RFC 2109) konnte sich nicht durchsetzen.
  - Das Prinzip:
    - Der Server sendet kleine Informationshäppchen (name/value-Paare) an den Client mit dem HTTP-Header "Set-Cookie".
    - Der Client speichert diese lokal (Festplatte) oder hält sie zumindest während seiner Laufzeit im Hauptspeicher vor.
    - Beim weiteren Abrufen eines "passenden" URL sendet der Client das Cookie zurück an den Server mit dem HTTP-Header "Cookie"
- Vorteile
  - Relativ performant und leicht implementierbar
  - Funktioniert auch bei unterbrochenen *sessions*
- Nachteile
  - Client muss Cookies unterstützen und auch zulassen (!)

HTTP: Zustandsverwaltung (*session management*)

Mehr zu Cookies

- Sicherheitsfragen
  - Es entwickelten sich "Verbundsysteme" von Servern, die einheitliche Cookies setzten und auswerteten, insb. in den USA. Dadurch ließen sich Bewegungsprofile im WWW ermitteln.
  - Obwohl Cookies ein Verfallsdatum tragen, verbreitete sich die Unsitte, dieses unnötig lange zu setzen (etwa: 30 Jahre statt 1 Tag)
- Die Netscape Cookie-Parameter
  - name, value; domain, expires, path, secure
  - Quelle: http://wp.netscape.com/newsref/std/cookie_spec.html
- Beispiel:
  - ```
    Set-Cookie: matr_nr=12345; domain=.fh-wiesbaden.de; path=/cgi;
                expires=Fri, 16-Apr-2004 07:15:00 GMT; secure
    ```
- Grenzen:
  - <= 4kB pro Cookie, <= 20 pro Domain, <= 300 pro Client
    ```