Internettechnologien I

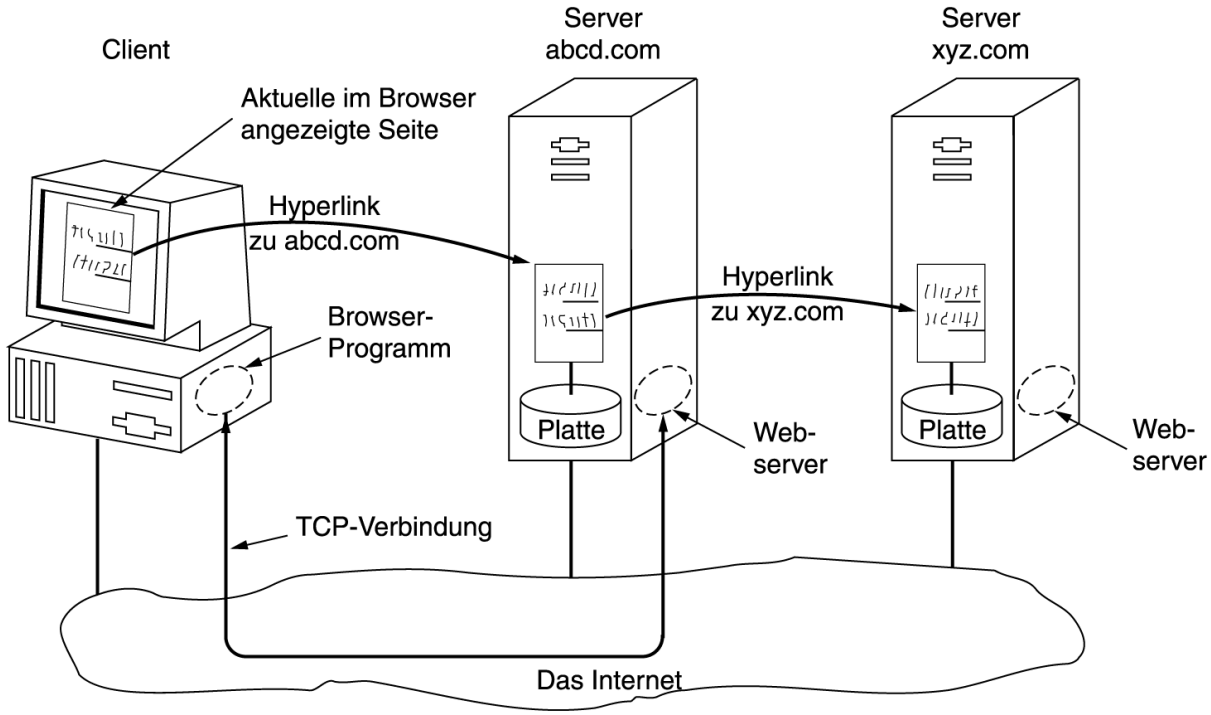
Anwendungsschicht - HTTP

Inhaltsverzeichnis

World Wide Web (WWW)

- Verknüpfung von verteilten Dokum. (Entw. 1989 am CERN)
- Standardübertragungsprotokoll ist HTTP (HyperText Transfer Protocol)
- TCP-Verbindung zu Port 80
- Wichtigster Befehl: `GET Dateiname HTTP/1.1`
- Webseite besteht aus Objekten (HTML, JPEG, Java-Applets, Audio, etc.)
- Webseiten im HTML-Format (HyperText Markup Language)
- Webseiten haben Basis-HTML-Datei mit Referenzen auf andere Objekte

Uniform Resource Locator (URL)



- Wie heißt die Seite?
- Wo befindet sich die Seite? (Host + Pfad)
- Wie kann auf die Seite zugegriffen werden?
- Bsp.:

  `http://www.htw-dresden.de/~jvogt/it1/index.html`

URL

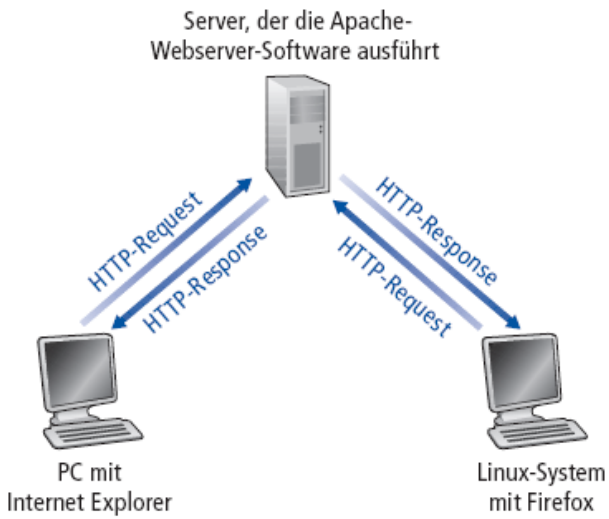
- Aufbau: `<scheme>:<scheme-specific-part>`
- Schema z. B.: http, ftp, mailto, file
- http: host - port - path - query - fragment
- z. B.:

  `http://name:pw@sub.domain.tld:80/path/dat.cgi?var1=a&var2=b#anker`

HTTP-Überblick

HyperText Transfer Protocol

- Anwendungsprotokoll des Web
- Client/Server-Modell
  - Client: Browser, der Objekte anfragt, erhält und anzeigt
  - Server: Webserver verschickt Objekte auf Anfrage
- Verwendet TCP
- Versionen:
  - HTTP 1.0: RFC 1945
  - HTTP 1.1: RFC 2068



HTTP

1. Server wartet auf Port 80
2. Client baut eine TCP-Verbindung zum Server auf
3. Server nimmt die TCP-Verbindung des Clients an
4. HTTP-Nachrichten werden zwischen Browser (HTTP-Client) und Webserver (HTTP-Server) ausgetauscht
5. Die TCP-Verbindung wird geschlossen

HTTP und Zustände

- Protokolle, die einen Zustand verwalten, sind komplex!
  - Der Zustand muss gespeichert und verwaltet werden
  - Wenn Server oder Client abstürzen, dann muss der Zustand wieder synchronisiert werden
- HTTP ist zustandslos
  - Server merkt sich keine Informationen über frühere Anfragen der Clients

HTTP-Verbindungen

Nichtpersistentes HTTP

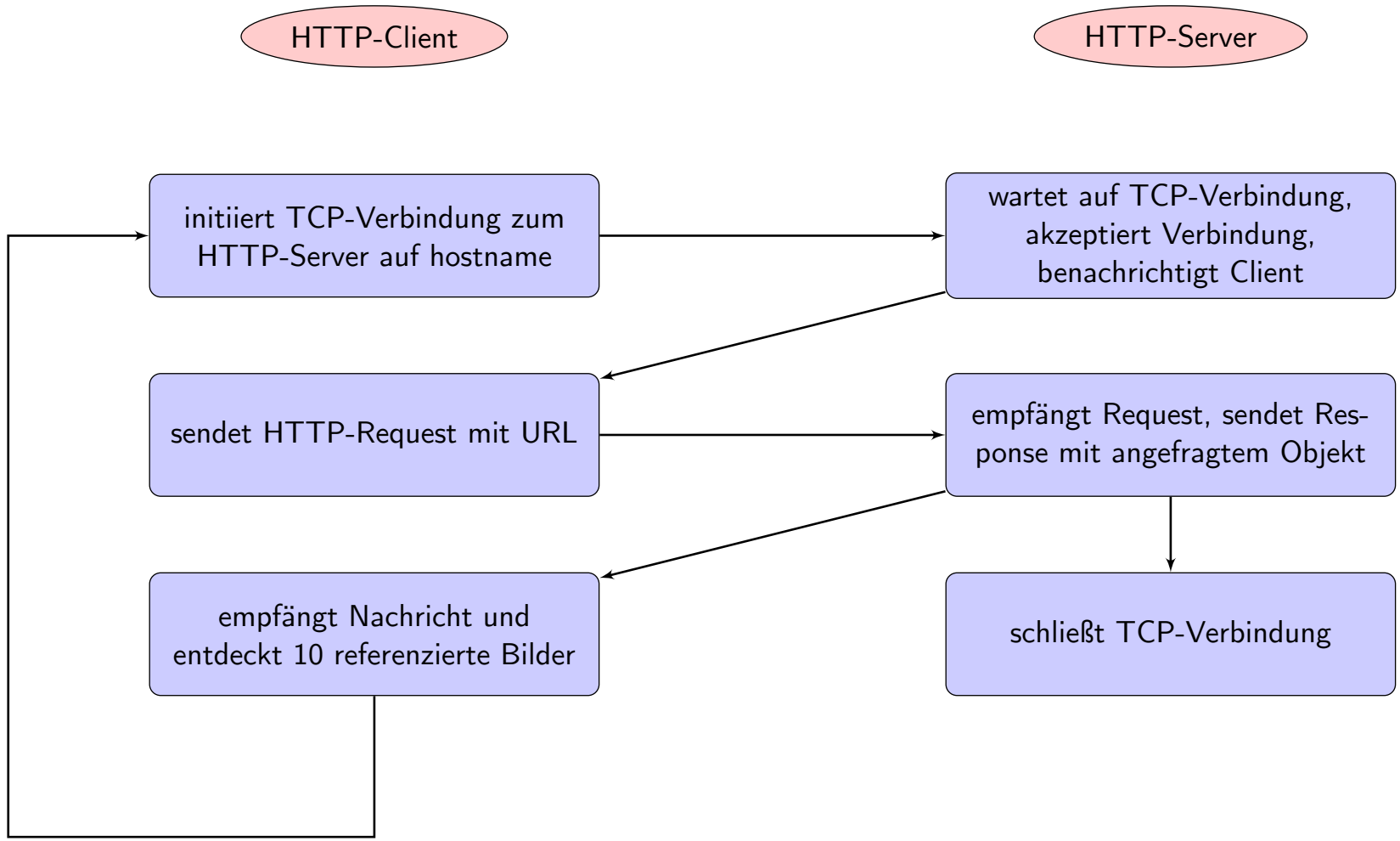
- Maximal ein Objekt wird über eine TCP-Verbindung übertragen
- HTTP/1.0 verwendet nichtpersistentes HTTP

Persistentes HTTP

- Mehrere Objekte können über eine TCP-Verbindung übertragen werden
- HTTP/1.1 verwendet standardmäßig persistentes HTTP

Nichtpersistentes HTTP

Bsp.: Website mit 10 Bildern auf hostname/index



Nichtpersistentes HTTP: Verzögerung

1. Eine RTT für Verbindungsaufbau (Daten können bereits ab dem 3. Segment versendet werden)
2. Eine RTT für den HTTP-Request, bis das erste Byte der HTTP-Response beim Client ist
3. Zeit für das Übertragen der Daten auf der Leitung

- Ergebnis: 2 RTT + Übertragungsverz. pro Objekt (Skizze)
- Aufwand im Betriebssystem für jede TCP-Verbindung
- Browser öffnen häufig mehrere parallele TCP-Verbindungen zur Beschleunigung

Persistentes HTTP

- Server lässt Verbindung nach dem Senden der Antwort offen
- Nachfolgende HTTP-Nachrichten können über dieselbe Verbindung übertragen werden

ohne Pipelining

- Client schickt neuen Request erst, nachdem die Antwort auf den vorangegangenen Request empfangen wurde
- Eine RTT für jedes referenzierte Objekt

mit Pipelining

- Client schickt Requests, sobald er die Referenz zu einem Objekt findet
- Idealerweise wird nur wenig mehr als eine RTT für das Laden aller referenzierten Objekte benötigt
- Standard in HTTP/1.1

Gegenüberstellung der Verfahren

Bei welchen Kanälen bringt Pipelining den größten Nutzen?

- geringe Datenrate, geringe Verzögerung
- hohe Datenrate, geringe Verzögerung
- geringe Datenrate, große Verzögerung
- hohe Datenrate, große Verzögerung

Experimente:

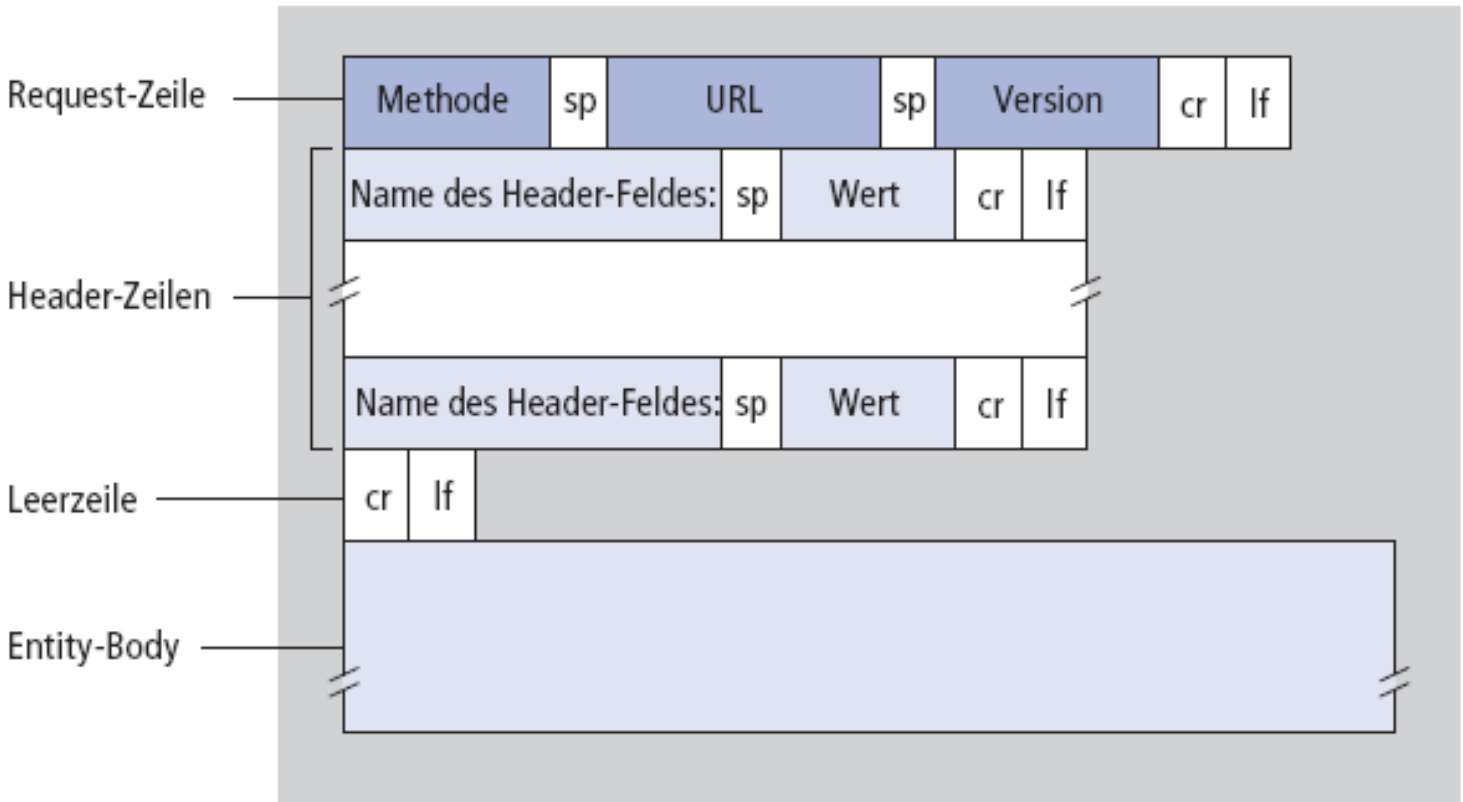
- FF: `about:config` → `network.http.pipelinig`
- Chrome: `chrome://flags` → `HTTP-Pipelining`

HTTP-Request

- Die HTTP-Nachrichten (Request, Response) werden mittels ASCII dargestellt (einfach zu lesen/debuggen)
- Zeilenumbruch bei Übertragung von Text i.d.R. CR+LF (z. B. bei HTTP, SMTP, FTP)
- Bestandteile des HTTP-Requests:
  - Request-Zeile (GET, POST, HEAD, ...)
  - Header-Zeilen
  - Endekennung: Leerzeile (Zusätzliches CR+LF)
- Beispiel:

```
GET /somedir/page.html HTTP/1.1
Host: www.htw-dresden.de
User-agent: Mozilla/4.0
Connection: close
Accept-language:de
```

HTTP-Request: Format



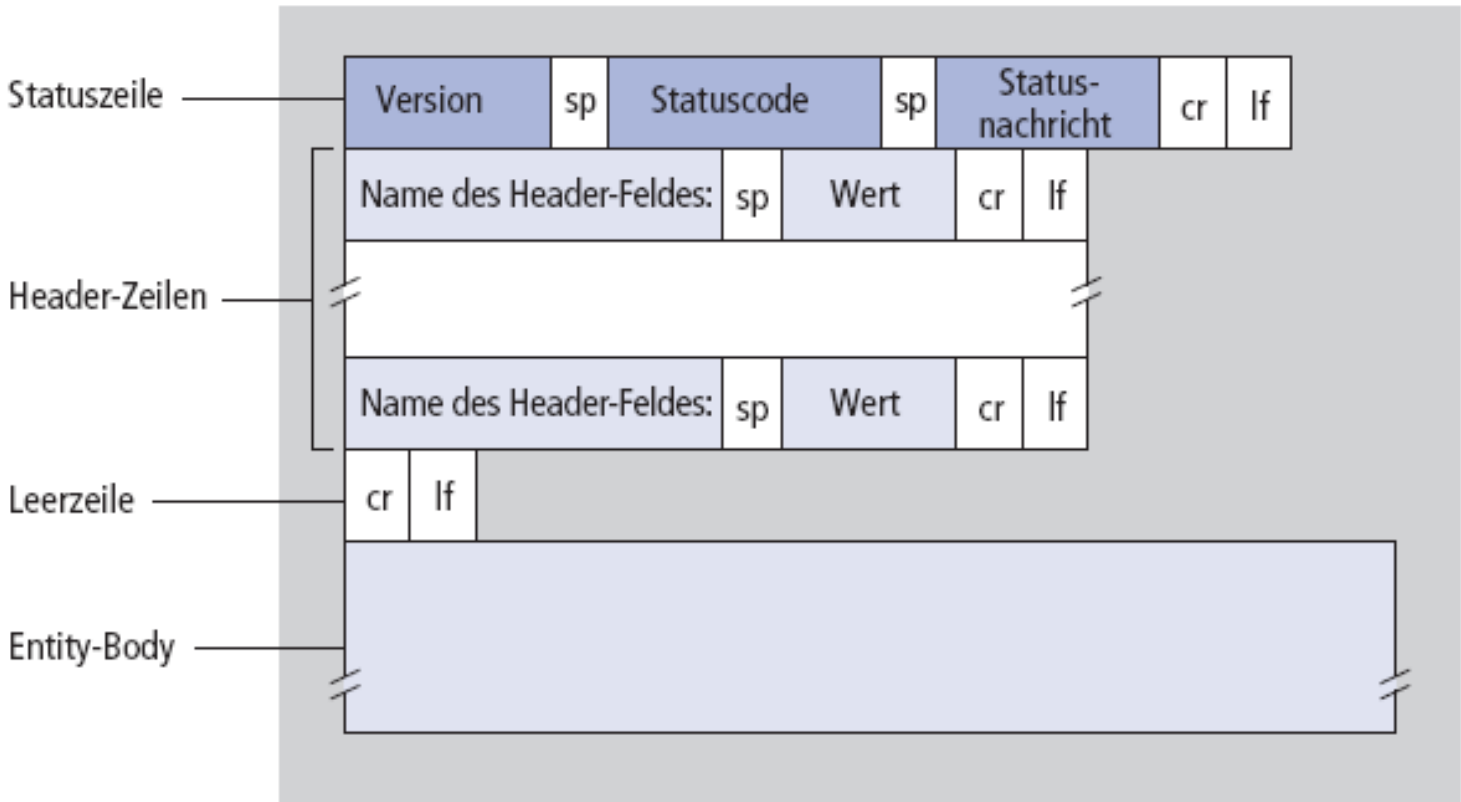
HTTP-Response

- Bestandteile des HTTP-Response:
  - Status-Zeile (Status-Code, Status-Nachricht)
  - Header-Zeilen
  - Endekennung: Leerzeile (Zusätzliches CR+LF)
  - Entity Body: Daten (z. B. angefrage HTML-Seite)
- Beispiel:

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

HTTP-Response: Format



Statuscodes

- Statuszeile gibt Protokollversion, Status Code und Reasons Phrase an
  - 1xx: Informelle Meldungen: Request erhalten, Bearbeitung wird durchgeführt.
  - 2xx: Erfolg: Request wurde erfolgr. erhalten, verstanden, angenommen
  - 3xx: Weiterleiten: es ist eine Weiterleitung notwendig (301: Moved Permanently) (304: Not Modified)
  - 4xx: Clientfehler: Request enthält ungültige Syntax oder ist nicht bearb.
    - (403: Forbidden – Zugriff verweigert)
    - (404: File not Found)
  - 5xx: Serverfehler: Der Server kann einen gültige Request nicht bearb.

Request-Typen

GET	Ressourcenanforderung
POST	Übertragung von Daten an den Server
HEAD	Bittet den Server, nur die Kopfzeilen der Antwort (und nicht das Objekt) zu übertragen
PUT	Lädt die im Datenteil enthaltene Datei an die durch eine URL bezeichnete Position hoch
DELETE	Löscht die durch eine URL angegebene Datei auf dem Server
OPTIONS	Liste mit unterstützten Methoden
PATCH	(RFC 5789) Modifikation von Dateien

- HTTP/1.0 nur GET,POST,HEAD
- GET, PUT, DELETE müssen idempotent sein (keine Veränderung durch mehrfaches Absenden des Requests)
- Einsatz z. B. in REST-Webservices

Hochladen von Daten

Webseiten beinhalten häufig Formulare, in denen Eingaben erfolgen sollen

POST

- Eingaben werden zum Server im Datenteil (entity body) der Post-Anweisung übertragen

GET

- Eingabe wird als Bestandteil der URL übertragen:
- Bsp.: `www.test.de/search?a=http&b=response`

Beispiel: HTTP-Request POST

- Übertragung von Formulardaten

```
POST /somedir/page.html HTTP/1.1
Host: www.htw-dresden.de
User-agent: Mozilla/4.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 17

a=http&b=response
```

HTTP-Versionen

- HTTP 0.9 (nicht mehr relevant)
- HTTP 1.0 (RFC 1945)
- HTTP 1.1 (RFC 2616)
  - ist abwärtskompatibel zu HTTP 1.0
  - auch alle http 1.1-Clients (Browser) unterstützen HTTP 1.0
  - Wichtige Verbesserungen:
  - Persistente (dauerhafte) Verbindungen möglich
  - Pipelining möglich (Client sendet nachfolg. Req. an den Server, auch bevor die Antworten des Servers eingetroffen sind)
  - neue Cache-Kontrollfunktionen zur Steuerung des Updates im Browsercache und in Proxies (spart wiederholte Übertragung bereits zwischengespeicherter Inhalte)
- HTTP 2.0 (RFC t.b.d.) Headerkompression, Server-Push, Bündelung von Anfragen

Ausprobieren des HTTP-Clients

Telnet

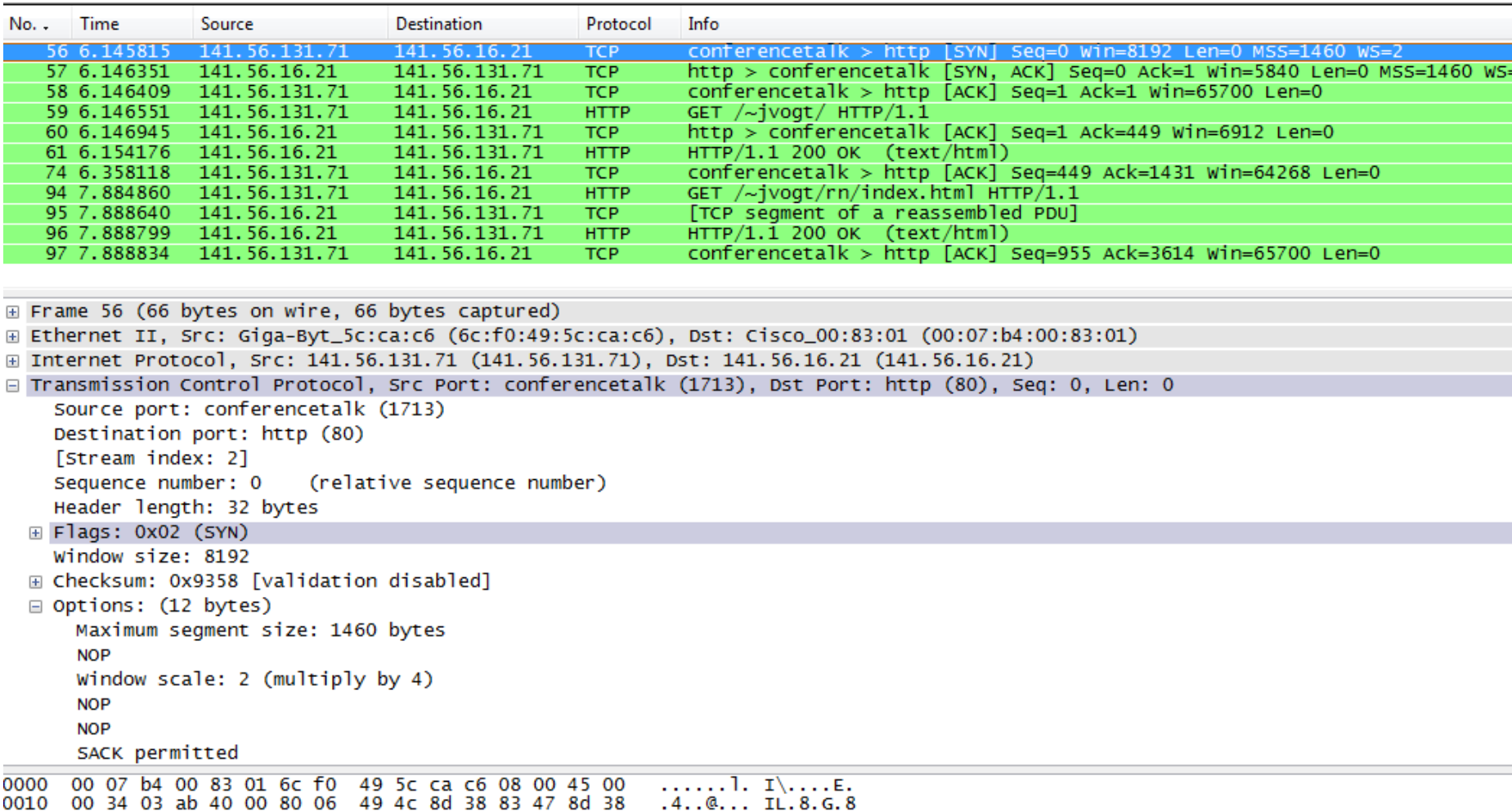
1. Verbindung zu einem Webserver
   `telnet www.htw-dresden.de 80`
2. Eingabe des (minimalen) HTTP-GET-Requests
   (Achtung: Am Ende 2x Return tippen)

```
GET /~jvogt/ HTTP/1.1
Host: www.htw-dresden.de
```

3. Erhalt der Antwort

Curl Vielseitiges Werkzeug mit Unterst. div. Protokolle
`curl http://www.htw-dresden.de/index.html`

HTTP-Beispiel



No.	Time	Source	Destination	Protocol	Info
56	6.145815	141.56.131.71	141.56.16.21	TCP	conferencetalk > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
57	6.146351	141.56.16.21	141.56.131.71	TCP	http > conferencetalk [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 WS=
58	6.146409	141.56.131.71	141.56.16.21	TCP	conferencetalk > http [ACK] Seq=1 Ack=1 Win=65700 Len=0
59	6.146551	141.56.131.71	141.56.16.21	HTTP	GET /~jvogt/ HTTP/1.1
60	6.146945	141.56.16.21	141.56.131.71	TCP	http > conferencetalk [ACK] Seq=1 Ack=449 Win=6912 Len=0
61	6.154176	141.56.16.21	141.56.131.71	HTTP	HTTP/1.1 200 OK (text/html)
74	6.358118	141.56.131.71	141.56.16.21	TCP	conferencetalk > http [ACK] Seq=449 Ack=1431 Win=64268 Len=0
94	7.884860	141.56.131.71	141.56.16.21	HTTP	GET /~jvogt/rn/index.html HTTP/1.1
95	7.888640	141.56.16.21	141.56.131.71	TCP	[TCP segment of a reassembled PDU]
96	7.888799	141.56.16.21	141.56.131.71	HTTP	HTTP/1.1 200 OK (text/html)
97	7.888834	141.56.131.71	141.56.16.21	TCP	conferencetalk > http [ACK] Seq=955 Ack=3614 Win=65700 Len=0

```
⊞ Frame 56 (66 bytes on wire, 66 bytes captured)
⊞ Ethernet II, Src: Giga-Byt_5c:ca:c6 (6c:f0:49:5c:ca:c6), Dst: Cisco_00:83:01 (00:07:b4:00:83:01)
⊞ Internet Protocol, Src: 141.56.131.71 (141.56.131.71), Dst: 141.56.16.21 (141.56.16.21)
⊟ Transmission Control Protocol, Src Port: conferencetalk (1713), Dst Port: http (80), Seq: 0, Len: 0
    Source port: conferencetalk (1713)
    Destination port: http (80)
    [Stream index: 2]
    Sequence number: 0    (relative sequence number)
    Header length: 32 bytes
  ⊞ Flags: 0x02 (SYN)
    Window size: 8192
  ⊞ Checksum: 0x9358 [validation disabled]
  ⊟ Options: (12 bytes)
      Maximum segment size: 1460 bytes
      NOP
      Window scale: 2 (multiply by 4)
      NOP
      NOP
      SACK permitted
```

```
0000  00 07 b4 00 83 01 6c f0  49 5c ca c6 08 00 45 00   ......l. I\....E.
0010  00 34 03 ab 40 00 80 06  49 4c 8d 38 83 47 8d 38   .4..@... IL.8.G.8
```

HTTP-Beispiel cont.

Betrachtung des TCP-Inhalts (Follow TCP)

```
GET /~jvogt/ HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml, i
Accept-Language: de-DE
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trid
Accept-Encoding: gzip, deflate
Host: www2.htw-dresden.de
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 31 Jan 2011 11:27:33 GMT
Server: Apache/2.2.10 (Linux/SUSE)
Last-Modified: Wed, 15 Dec 2010 11:50:57 GMT
ETag: "b83a25-46c-497718cd96a40"
Accept-Ranges: bytes
Content-Length: 1132
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<HTML>
<HEAD>
```

Inhaltsvereinbarung (Content Negotiation)

- Aushandlung des Inhaltes der übertr. Ressource anhand der Möglichkeiten des Clients (wenn Auswahl vorhanden)
- Durch Komma getrennte Liste an (gewichteten) Eigenschaften
- opt. Param. $q = [0 \dots 1]$ (0-abgeleh., 1-bevorz. (Standardw.))
- Client-Anfragen:

Accept: akzeptierte MIME-Typen -> Server: Content-Type
Accept-Charset: --> Server: charset-Parameter im Content-Type
Accept-Encoding: --> Server: Content-Encoding
Accept-Language: --> Server: Content-Language

Beispiel

```
GET / HTTP/1.0
Accept-Language: de-de, de;q=1.0, en;q=0.5

HTTP/1.0 200 OK
Content-Language: de
```

Ausgewählte Anfrage-HTTP-Header

Host Domainname bezgl. Anfrage (ab HTTP 1.1 Pflicht)

User-Agent: Informationen zum Browser + BS

Connection: Keep-Alive | Close

Cookie: HTTP-Cookie, vom Server bekommen mittels Set-Cookie

POST Operationen

- **Content-Typ:** MIME-Typ (Multipurpose Internet Mail Extensions)
- **Content-Length:** Länge des Datenbereichs
- **Content-Language:** gesendete Sprache
- **Content-Encoding:** z.B: gzip, compress, deflate

Date: Zeitpunkt des Absendens der Daten

If-Modified-Since: Erlaubt Server (304 Not Modified zu senden)

If-None-Match: (ETag) Erlaubt Server (304 Not Modified zu senden)

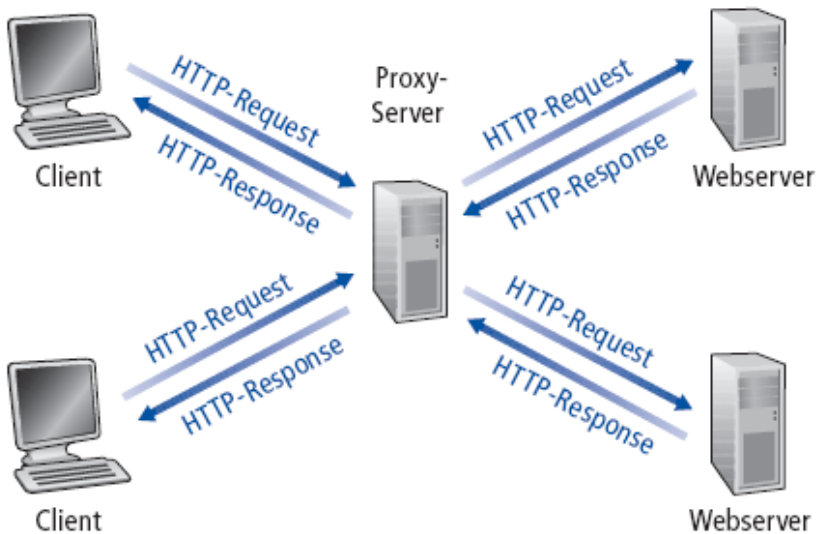
Ausgewählte Antwort-HTTP-Header

Connection:	Keep-Alive | Close
Content-Typ:	MIME-Typ (Multipurpose Internet Mail Extensions)
Content-Length:	Länge des Datenbereichs
Content-Language:	gesendete Sprache
Content-Encoding:	z.B: gzip, compress, deflate
Transfer-Encoding:	z. B. chunked
Date:	Zeitpunkt des Absendens der Daten
Expires:	Zeitpunkt für Ablauf der Gültigkeit ohne Nachfrage
Last-Modified:	Zeitpunkt der Ressourcenerstellung
ETag:	Erkennung der Änderung von Ressourcen (für dyn. Inhalte)
Location:	Umleitung der Anfrage
Server:	Informationen zum Server
Set-Cookie:	HTTP-Cookie

Web-Caches (Proxyserver)

Ziel: Anfragen des Clients ohne den ursprünglichen Webserver beantworten

- Benutzer konfiguriert Browser: Webzugriff über einen Cache
- Browser sendet alle HTTP-Requests an den Cache
  - Objekt im Cache: Cache gibt Objekt zurück
  - Sonst: Cache fragt das Objekt vom ursprünglichen Server an und gibt es dann an den Client zurück



Web-Caches 2

- Cache arbeitet als Client UND als Server
- Üblicherweise ist der Cache beim ISP installiert (Universität, Firma, ISP für Privathaushalte)
- HTW: www-cache.htw-dresden.de, Port 3128

Vorteile

- Verringert die Antwortzeit
- Verringert den Datenverkehr auf der Zugangsleitung
- Bei Existenz vieler Caches: „arme“ Inhaltsanbieter können ihre Inhalte gut verbreiten (Ähnliches kann durch P2P-Filesharing erreicht werden)

Beispiel

Annahmen

- LAN: 100 Mbps, Zugangsleitung: 15 Mbps
- Durchschnittliche Größe eines Objektes = 100.000 Bit
- Durch. Rate von Anfragen aller Webbrowser der Firma: 150/s
- Verzögerung v. Router d. Firma zum Server und zurück: 2s

Ergebnis

- Datenrate: 15 Mbps
- Auslastung des LAN = 15%
- Auslastung der Zugangsleitung = 100%
- Verzögerung = Internet + Zugangsleitung + LAN = 2s + Minuten + Millisekunden

Lösung 1

Erhöhen der Datenrate der Zugangsleitung auf 100 Mbit/s

Ergebnis

- Auslastung des LAN = 15%
- Auslastung der Zugangsleitung = 15%
- Verzögerung = Internet + Zugangsleitung + LAN = 2 sec + Millisekunden + Millisekunden
- Oft sehr teuer!

Lösung 2

Web-Cache installieren -> Annahme: Trefferrate = 0,4

Ergebnis

- 40% der Requests werden nahezu sofort beantwortet
- 60% der Requests durch normale Webserver beantwortet
- Auslastung der Zugangsleitung auf 60% reduziert, Verzögerungen werden vernachlässigbar klein (z.B. 10 ms)
- Mittlere Verzögerung = Internet + Zugangsleitung + LAN

$$T_m = 0{,}6 \cdot (2{,}01\ \text{s}) + 0{,}4 \cdot (10\ \text{ms}) < 1{,}4\ \text{s}$$

Bedingtes GET

- Ziel: Objekt nicht senden, wenn der Cache eine aktuelle Version besitzt
- Cache: Angeben des Änderungsdatums der gespeicherten Version (kann der HTTP-Response entnommen werden) durch folgende Zeile im HTTP-Request:

  ```
  If-modified-since: <date>
  ```

- Server: HTTP-Response enthält kein Objekt, wenn die Version im Cache aktuell ist:

  ```
  HTTP/1.0 304 Not Modified
  ```

- ansonsten (Objekt verändert):

  ```
  HTTP/1.0 200 OK
  <Daten>
  ```

Chunked-Transfer

- Sinnvoll wenn dem Webserver die genaue Anzahl an zu übertragenden Daten zum Zeitpunkt des Beginns der Übertragung unbekannt ist
- Attribut: Transfer-Encoding: chunked
- Kein Attribut: Content-Length
- Abwechselnde Übertragung:
  `Länge | Daten | Länge | Daten | 0`
- Zeilenumbruch vor und nach der Längenangabe gehört nicht zu den Daten

HTTP-Authentifizierung, RFC 2617

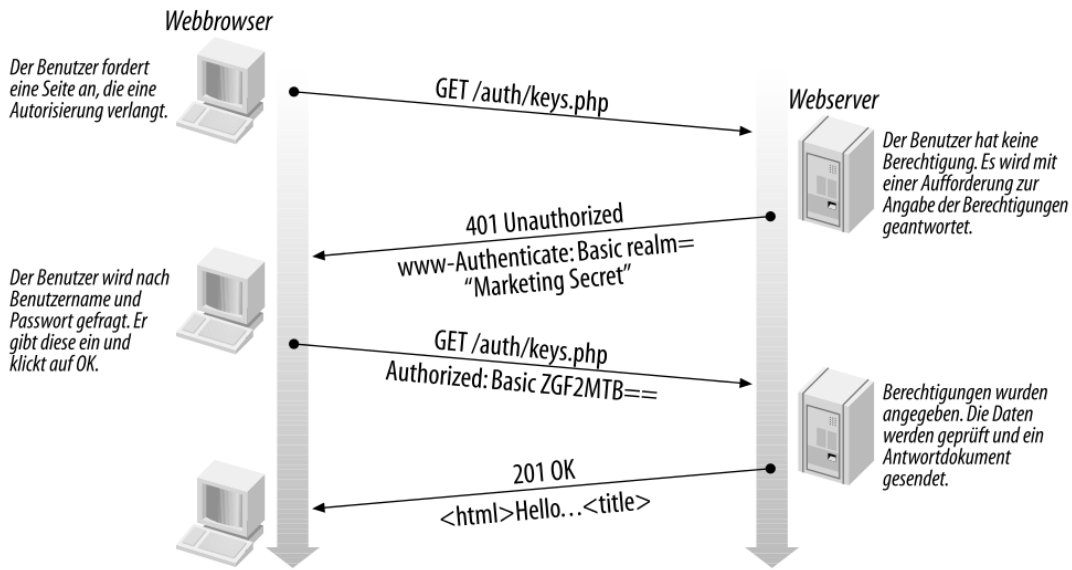
Basic Authentification
- häufige Form der Authentifizierung
- SSL/TLS zum Schutz notwendig

Digest Access Authentication (DAA)
- Challenge-Responce-Verfahren, sicher vor Abhörversuchen

- Alternative: Sessionbasierte Authentifizierung

Basic Authentifizierung

- Aufforderung: (Bereich, Domäne)
  `WWW-Authenticate: Basic realm="RealmName"`
- Antwort (Base64 von name:passwort):
  `Authorization: Basic g4FsdNs8H==`
- Klartextübertragung: SSL/TLS notwendig!
- Berechtigung wird vom Browser gesammelt und bei weiteren Anfragen an den geschützten Bereich automatisch mitgesendet



Digest Access Authentication (DAA)

- Challenge-Responce-Verfahren, sicher vor Abhörversuchen
- vom Apache unterstützt, nicht von allen Browsern unterstützt
- Ablauf:

1. Server sendet zufällige Zeichenfolge (Nonce)
2. Browser berechnet Hash (z. B. MD5) aus Benutzername+Passwort+Nonce+HTTP-Methode+URI
3. Browser sendet an Server: Benutzername, Nonce, Hash
4. Server prüft Hash

WebDAV

Web-based Distributed Authoring and Versioning

- Erweiterung des HTTP-Protokolls zur Unterstützung von Onlinespeicher (Internet-Dateisystem)
- HTTP Extensions for Distributed Authoring, RFC 2518
- HTTP Ext. for Web Distr. Authoring and Version. RFC 4918
- Hohe Verfügbarkeit da Nutzung des HTTP-Prot. mit Port 80
  - Einf. Nutzung von Firewalls im Gegensatz zu SSH, FTP, etc.
- Möglichkeit der Übertragung von mehreren Dateien
- Unterstützung bei den meisten Servern / Clients vorhanden
- Neue Methoden:
  - MKCOL erstellt ein Verzeichnis (Collection)
  - COPY Kop. eine Ressource unter Ang. der URI für Dateinam.
  - MOVE Verschiebt eine Ressource
  - LOCK/UNLOCK Sperrung/Entsperrung einer Ressource
  - PROPPATCH Änderung von Eigenschaften einer Ressource
  - PROPFIND Infor. zu einer Ressource/Verzeichnisstrukt.

Restful Webservices

REST – Representational State Transfer

- Nutzung von HTTP für Webservices
- REST ist kein Produkt oder Standard
- Prinzipien eines Restful-Webservice
  - Adressierbarkeit mittels URI
  - Unterschiedl. Darstellungsformen möglich (XML, JSON, etc.)
  - Zustandslosigkeit
  - Alle notwendigen Informationen sind in einer Anfrage enthalten
  - Definierte Operationen (GET, POST, PUT, DELETE, ...)
  - Nutzung von Verknüpfungen

Beispiel REST-Webservice

- Webshop
- Alle Objekte der Anwendung sind über URLs erreichbar

```
GET   /warenkorb/73
GET   /artikel/60
POST  /warenkorb/73 (Ergänzen des Warenkorbs)
          artikelnummer=61
PUT   /artikel      (Anlegen einer neuen Ressouce)
    <artikel>
       ...
    </artikel>

HTTP/1.1 201 Created    (Antwort mit URL zum neuen Artikel)
Location: http://shop.de/artikel/05

DELETE  /artikel/05   (Ressource löschen)
```

Zustand halten: HTTP-Cookie, RFC2109

Erinnerung: HTTP ist zustandslos

Bestandteile:

1. Cookie-Kopfzeile in der HTTP-Response-Nachricht
2. Cookie-Kopfzeile in der HTTP-Request-Nachricht
3. Cookie-Datei, die auf dem Rechner des Anwenders angelegt und vom Browser verwaltet wird
4. Backend-Datenbank auf dem Webserver

- Können auch lokal per Skript erzeugt werden
- Cookies können weitere Attr. beinhalten (Expire,Path, etc.)
- Arten: Sitzungscookies (Session-ID), Permanenter Cookie, Third-party cookie
- Max. Größe typisch 4 KB
- Alternativen: Übertragung in URI (index?sid=1234) oder verstecktes Formularfeld

HTTPS-Erweiterung: Public-Key-Pinning

- Versendung eines Pins im HTTP-Header
- Festlegung auf bestimmte Schlüssel
- Zeitdauer

Constrained Application Protocol (CoAP)

- Protokoll für einfache Geräte (RFC 7252)
- Reduziertes binärcodiertes HTTP
- Unterstützt UDP
- Kleiner Header
- Unterstützung einer Suchfunktion für Ressourcen
- Abonnement von Ressourcen
- Mapping zwischen CoAP und HTTP möglich mittels Proxy
- Implementierungen: Copper (FF Plugin), Californium (Java), Erbium (C)

```
coap://[2001:0233::3e4b:2713]/sensors/temp1
```

Fragen

- Wofür wird HTTP prinzipiell verwendet?
- Was bedeutet Pipelining und Persistenz?
- Wie funktioniert "Content Negotiation"?
- Hat HTTP Zustände?
- Wie können Zustände erzeugt werden?
- Wie funktioniert HTTP-Authentifizierung?

Zusammenfassung

- HTTP ist ein zentrales (Anwendungs)Protokoll zum Datenaustausch und basiert auf TCP
- Request / Response -Nachrichten
- Inhaltsvereinbarung
- Viele Anwendungsprotokolle nutzen wiederum HTTP als Grundlage (z.B. WebDAV)