# Verteilte Systeme

## HTTP 1.1 – Teil A

by

Dr. Günter Kolousek

# Überblick

- Hyper Text Transfer Protocol
- Client/Server Protokoll
- Anwendung
  - ursprünglich zum Abrufen statischer Informationen im WWW
    - $\rightarrow$ relativ einfach
  - allgemeine Vernetzung zwischen Hosts
    - ”HTTP is a generic interface protocol for information systems.” (RFC7230)
    - Netzwerkkomponenten, Hausautomation, Webcams,…
    - $\rightarrow$ RESTful Webservices
- Unabhängig von Transportprotokoll, nur *zuverlässig*
  - de facto: TCP und Defaultport 80
- Version 1.0 vs. Version 1.1 vs. Version 2 vs. Version 3
  - HTTP/1.1: RFC 7230, 7231, 7232, 7233, 7234, 7235

# Charakterisierung

- (relativ) einfaches Client/Server Protokoll (ursprünglich)
  - Request/Response Protokoll
  - Client $\rightarrow$ user agent (z.B. Browser, Spider,...)
- **zustandsloses** Protokoll
  - jeder Request unabhängig
  - verbindungslos (vs. TCP-Verbindung)
- zeichenorientiert
  - muss als ASCII geparst werden
  - aber: ”8bit-clean”
    - kein Problem binäre Daten zu übertragen
- zeilenorientiert
- Verhandlung Datenrepräsentation

# Domain Name im WWW

- DN: Domain Name
  - $\rightarrow$ IP Adresse
  - ASCII Zeichen!
- IDN: Internationalized Domain Name
  - Unicode Zeichen möglich
  - Umsetzung in ASCII mittels Punycode!

# URI vs. URL vs. URN vs. IRI

- URI: Uniform Resource Identifier
  - Überbegriff für URL und URN
- URL: Uniform Resource Locator
  - es kann ”hängende Referenz” entstehen
  - $\rightarrow$ Adresse!
- URN: Uniform Resource Name
  - dauerhafter Name für Ressource
  - z.B. `urn:isbn:3-8266-1378-3`
- IRI: Internationalized Resource Identifier
  - URI in UCS (Universal Character Set $\equiv$ UTF-16)
  - IRI wird auf URI abgebildet
    - non-ASCII $\rightarrow$ UTF-8 $\rightarrow$ urlencoded (Prozentkodierung, siehe später)

# URL

- `scheme:scheme-specific`
- für HTTP:
  `proto://[user:pass@]hostname[:port][/path][?query][#frag]`
  - `proto` … Protokoll: `http` oder `https`
  - `hostname` … IDN oder IP Adresse
  - `path` … beginnt mit oder ohne /
  - `frag` … Fragment: Anker (engl. anchor) in HTML
- de facto ASCII!

# URL-Encoding (Prozentkodierung)

- Betrifft die ”Datenteile” einer URI (`username`, `password`, `path`, `query`, `fragment`)
  - ”a” - ”z”, ”A” - ”Z”, ”0” - ”9” bleiben gleich
  - ”.”, ”-”, ”~”, ”_” bleiben gleich
  - ” ” $\rightarrow$ ”+” (im `query` String)
  - jedes andere Zeichen $\rightarrow$ UTF-8 (Empfehlung!) $\rightarrow$ `%xy`
- entspricht MIME Typ `application/x-www-form-urlencoded`
  - Formulardaten in `POST` - Request

# MIME Type

- soll Typ zu Nachricht herstellen
  - E-Mail
  - HTTP
  - Desktop: Aqua (macOS), XFCE, KDE, Gnome
- `<toplevel>/<subtype>`

# MIME Type – 2

- `text`
  - `plain`, `html`, `css`,...
- `application`
  - `octet-stream`, `pdf`, `xml`, `json`, `javascript`,...
- `audio`
  - `mpeg`, `ogg`,...
- `image`
  - `png`, `jpeg`, `gif`
- `video`
  - `mp4`, `VP8`, `H264`,...
- `font`
  - `otf`, `ttf`, `woff`, `woff2`,...
- `multipart` mehrere Teile `mixed`, `alternative`
- `message`: für E-Mails
- `example`: nur für Beispiele

# Request/Response Protokoll

- Client (User Agent, UA) setzt Request ab
  - zuerst wird TCP Verbindung aufgebaut
  - Standardport 80
- Server antwortet mit Response
  - bis HTTP/1.0: Server schließt TCP Verbindung
    - wenn nicht `Connection: KeepAlive`
  - HTTP/1.1: TCP Verbindung bleibt offen
    - wenn nicht `Connection: Close`
  - Schließen der Verbindung: siehe Folie ”Verbindungsabbau”
- HTTP/1.0 an HTTP/1.1: Responses müssen HTTP/1.0 ”kompatibel” sein!

# Request - Aufbau

1. `Method Request-URI Protokoll\r\n`
   - meist relative URL, z.B. `/index.html` (also absoluter Pfad)
     - wenn Proxy, dann sendet UA eine absolute URL
     - ab HTTP/1.1 muss Server absolute URLs verstehen
   - Daten in URL $\rightarrow$ urlencoded
   - `\r` ... ASCII 13, `\n` ... ASCII 10
2. `(Key: Value\r\n)*`
3. `\r\n`
4. `Daten?`

# Eigenschaften von Methoden

**side effects** Nebeneffekte; ”nicht offensichtliche, unbeabsichtigte oder unerwünschte Nebenwirkungen einer Operation in der Informatik” (Quelle: Wikipedia)

- beim Programmieren: $\rightarrow$ globale oder statische Variable, Verändern eines Arguments, Schreiben auf Stream (z.B. Konsole), Aufrufen einer anderen Funktion (mit Nebeneffekten)
- bei HTTP: hauptsächlich verändern einer (anderen) Ressource

**safe** keine Veränderungen am Server

- no side effects!

**cacheable** kann in einem Cache gespeichert werden

**idempotent** mehrere identische Requests $\equiv$ ein Request

- safe impliziert auch idempotent!

# Methoden

- GET: safe, cacheable
- HEAD: safe, cacheable
- POST: cacheable (gemäß Spezifikation)
- PUT: idempotent
- DELETE: idempotent
- CONNECT
- OPTIONS: safe
- TRACE: safe
- PATCH: cacheable (unter bestimmten Bedingungen)
  - RFC 5789

# GET und HEAD

- GET
  - Zweck: Anforderung, um Darstellung der Ressource herunterzuladen
  - kein Datenbereich in Methode $\rightarrow$ Daten in `query` (z.B. Formularfelder)
    - Regel: nicht für Formulare verwenden!
  - Semantik
    - keine Veränderung am Server
    - mehrmaliges Senden $\rightarrow$ gleiches Ergebnis
- HEAD
  - Zweck: Anforderung, Header der Ressource herunterzuladen
  - Server antwortet wie bei GET jedoch *ohne* Daten

# POST

- Zweck: Übertragung eines Datenblockes an Server
- Daten im Datenblock
- wird verwendet,
  - um neue Ressource anzulegen oder
    - Statuscode 201 und `Location` Header
  - eine Bestehende zu verändern
- Semantik
  - Veränderungen werden vorgenommen
  - mehrmaliges Senden $\rightarrow$ jeweils *neues* Ergebnis

# PUT und DELETE

- PUT
  - Zweck: Anforderung, um eine Ressource zu ”setzen”
  - Semantik
    - neue Ressource anlegen
    - bestehende Ressoure ersetzen (überschreiben)
- DELETE
  - Zweck: Anforderung zum Löschen einer Ressource
  - Semantik
    - Veränderung wird vorgenommen
    - mehrmaliges Senden $\rightarrow$ gleiches Ergebnis

# CONNECT, OPTIONS

- CONNECT
  - Tunnel über Proxy aufbauen (z.B. für TLS)
    1. Client HTTP zu Proxy
    2. Proxy HTTP zu Server
    3. dann nur mehr TCP Tunnel über Proxy
  - z.B. `CONNECT www.orf.at:80 HTTP/1.1` an `proxy.htlwrn.ac.at`
- OPTIONS
  - Fähigkeiten des Servers ermitteln
  - z.B. `OPTIONS * HTTP/1.1`

# TRACE

- Response wie Request!
- Hops vom Client zum Server
  - Jeder Proxy fügt `VIA`-Header hinzu
    - `VIA: protocol-version host`
    - z.B. `VIA 1.1 proxy.htlwrn.ac.at`
    - Hostname … sensitive → Pseudonym, z.B. `VIA 1.1 fred`
    - mehrere `VIA` Felder oder: `VIA 1.1 fred, 1.0 max`
    - nicht nur bei TRACE
- Testen über Firewalls und Proxy-Server
- kann für Angriff genutzt werden
  - ”cross-site tracing” → sperren

# PATCH

- Verändern einer Ressource
  - d.h. teilweises Abändern
- Verwendung: siehe Folien ”REST”

Laut lt. RFC 5789:

- Ein PATCH Response ist cacheable, wenn folgende Header enthalten sind:
  - Aktualitäts-Header (z.B. Expires oder Cache-Control: max-age)
    **und**
  - Content-Location Header, der zu der Request-URI passt
    - zeigt an, dass PATCH Response eine Repräsentation der Ressource ist

# Beispiel für GET

```
GET / HTTP/1.0

HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 3

abc
```

# Header in Request

- `Host: host` ... **muss** in HTTP/1.1 vorhanden sein, gibt Empfänger-Host an;
  - $\rightarrow$ virtuelle Hosts
- `Connection: options`
  - `Keep-Alive`, `Close`, `Upgrade`
- `User-Agent: name` ... Name und Version des UA
- `Accept: type/subtype` ... MIME Type
  - `Accept: text/plain;q=0.5, text/html, text/x-dvi;q=0.8, application/pdf`
- `Accept-Charset` ... gibt Zeichensätze an
  - `Accept-Charset: utf-8, iso-8859-15;q=0.8`
- `Accept-Encoding` ... gibt Kodierung an
  - `Accept-Encoding: compress;q=0.5, gzip;q=1.0, identity;q=0.3`

# Header in einem Request – 2

- `Accept-Language` ... gibt Sprache an
  - `Accept-Language: de, en;q=0.8, *;q=0.5`
- `Referer: uri` (in Englisch: Referrer!)
  - URL, die zu der aktuellen Resource verwiesen
- `Authorization: Credentials`
  - wenn Client 401 gesendet hat
- `Cookie: name=value`
- `Content-Length: n` ... z.B. bei POST
- `Content-Type: type/subtype` (z.B. bei POST)
- `Upgrade: protocols`

# Header in einem Request – 3

- `If-Modified-Since: date-rfc1123-format`
  - Antwort, wenn verändert seit Datum, sonst `304 Not Modified`
- `If-Unmodified-Since: date-rfc1123-format`
  - Antwort bzw. Aktion (non-safe, wie POST), wenn nicht verändert seit Datum, sonst `412 Precondition Failed`
- `If-None-Match: etag`
  - Antwort, wenn ETag anders, sonst `304 Not Modified`
- `If-Match: etag`
  - Antwort bzw. Aktion, wenn ETag gleich, sonst `412 Precondition Failed`
  - z.B. bei POST, PUT, DELETE (Vermeidung von ”lost update”)

# Beispiel für POST-Request

```
POST /post HTTP/1.1
Host: httpbin.org
Content-Length: 22
Content-Type: application/x-www-form-urlencoded

name=maxi%20mustermann
```

Testen, in etwa so:

```
curl -X POST ”https://httpbin.org/post”
  -H ”Content-Type: application/x-www-form-urlencoded”
  -d ”name=maxi mustermann”
```

# Response - Aufbau

1. `Protocol Statuscode Description\r\n`
   - Protokoll: z.B. `HTTP/1.1`
   - Statuscode 1xx - 5xx
2. `(Key: Value\r\n)*`
3. `\r\n`
4. `Daten?`

# Header in einem Response

- `Date: date-rfc1123-format` ... **muss** in HTTP/1.1 vorhanden sein ($\rightarrow$ Caching!)
- `Server: name` ... analog zu `User-Agent`
- `Transfer-Encoding`
  - $\rightarrow$ Daten (payload), um sicheren Transport
  - `chunked`, `gzip`, `deflate` (zlib), `compress` (Unix)
    - z.B. `Transfer-Encoding: gzip, chunked`
- `Content-Length: n`
  - nicht `Transfer-Encoding` = "chunked" **und** `Content-Length`
  - bei persistenten Verbindungen
- `Content-Type: type/subtype`
- `Content-Location: URI`
  - gibt alternative URI für Repräsentation an
  - z.B.: URI ... `/data`, `Accept: application/json` $\rightarrow$ `Content-Location: /data.json`

# Header in einem Response – 2

- `Content-Language: lang`
  - ”Antwort” auf `Accept-Language`
- `Upgrade: protocols`
- `Location: URI`
  - bei Redirection und Response auf POST-Requests
- `Set-Cookie: NAME=VALUE`
- Caching
  - `Last-Modified: date-rfc1123-format`
  - `Expires: date-rfc1123-format`
  - `Cache-control: value` (z.B. `max-age=600`)
    - ersetzt `Expires`
  - `ETag: ”value”`
    - UA verwendet $\rightarrow$ Cache
    - tw. zum Tracking der User verwendet!

# Beispiel für POST-Response

```
HTTP/1.1 200 OK
Connection: keep-alive
Server: gunicorn/19.9.0
Date: Tue, 09 Oct 2018 10:36:21 GMT
Content-Type: application/json
Content-Length: 399
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Via: 1.1 vegur
```

Testen, in etwa so:

```
curl -X POST "https://httpbin.org/post"
  -H "CONTENT-TYPE: application/x-www-form-urlencoded"
  -d "name=maxi%20mustermann" -D headers.txt
```

# Beispiel für POST-Response – 2


```
{
  ”args”: {},
  ”data”: ””,
  ”files”: {},
  ”form”: {
    ”name”: ”maxi mustermann”
  },
  ”headers”: {
    ”Accept”: ”*/*”,
    ”Connection”: ”close”,
    ”Content-Length”: ”22”,
    ”Content-Type”: ”application/x-www-form-urlencoded”,
    ”Host”: ”httpbin.org”,
    ”User-Agent”: ”curl/7.61.1”
  },
  ”json”: null,
  ”origin”: ”195.202.162.6”,
  ”url”: ”https://httpbin.org/post”
}
```

# Herunterladen einer Website

```
$ wget --recursive \
       --page-requisites \
       --convert-links \
       --domains=think-async.com \
       --no-parent \
           think-async.com/Asio/asio-1.12.1/doc/
```

- `--recursive` ... rekursiv absteigen mit (Default-)Tiefe 5
- `--level=n` ... Tiefe angeben
- `--page-requisites` ... lade alle ”notwendigen” Dateien (wie inline Bilder,...)
- `--convert-links` ... nach dem Downloaden: konvertiere Links, sodass diese lokal verwendbar sind
- `--domains=LIST` ... (kommaseparierte) Liste der Domänen, denen gefolgt werden soll
- `--no-parent` ... betrachte nur alles unterhalb des angegebenen Wurzelverzeichnisses

# Statuscode

- 1xx … Information
  - speziell:
    - `100 Continue`
    - `101 Switching Protocols`
- 2xx … Erfolg
  - speziell:
    - `200 OK`
    - `201 Created`

# Statuscode – 2

- 3xx … Weiterleitung
- `Location` Header!
- Permanente Weiterleitung
  - `301 Moved Permanently`
    - GET unveränderte Neuanforderung
    - Wechsel der Methode zu GET von manchen UA (⊥ Spezifikation!)
  - `308 Permanent Redirect` … wie `301`
    - kein Wechsel der Methode erlaubt!

# Statuscode – 3

- Temporäre Weiterleitung
  - `302 Found` ... GET unveränderte Neuanforderung; Problem: Browser implementierten wie `303`, deshalb wurden `303` und `307` hinzugefügt
  - `303 See Other` (HTTP/1.1) ... GET unveränderte Neuanforderung; in Response zu POST (oder PUT) Requests $\rightarrow$ Wechsel zu GET!
  - `304 Not Modified` ... kein `Location` Header
  - `307 Temporary Redirect` ... wie `302`, aber kein Wechsel der Methode erlaubt!
- ”Alternativen”
  - HTML Weiterleitung $\rightarrow$ ”Back”-Button!!!

    ```
    <meta http-equiv=”refresh”
    content=”0;URL='http://www.htlwrn.ac.at/'”/>
    ```
  - JavaScript

    ```
    window.location=”http://www.htlwrn.ac.at/”;
    ```

# Statuscode – 3

- 4xx … Fehler des Clients
  - speziell
    - `400 Bad Request`
    - `401 Unauthorized`
    - `403 Forbidden`
    - `404 Not Found`
- 5xx … Fehler des Servers
  - speziell
    - `500 Internal Server Error`
    - `501 Not Implemented`
    - `503 Service Unavailable`
    - `505 HTTP Version Not Supported`

# Beispiel für GET – 2

```
GET / HTTP/1.1
HOST: www.htlwrn.ac.at

HTTP/1.1 200 OK
Date: Tue, 06 Sep 2016 15:39:42 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.45-0+deb7u4
Set-Cookie: ece...8f8=bta...jp0; path=/
Expires: Mon, 1 Jan 2001 00:00:00 GMT
Last-Modified: Tue, 06 Sep 2016 15:39:49 GMT
Cache-Control: no-store, no-cache, must-revalidate,
  post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
```

# Cookie

- wird von Server an UA gesendet
- UA sendet jedes Mal an Server
- $\rightarrow$ verwalten von Zustand
  - HTTP ist zustandslos!
- Use-Cases
  - Session
    - speziell auch nach Authentifizierung
  - Personalisierung
    - z.B. Einstellungen
  - Tracking
    - $\rightarrow$ Third-Party Cookie

# Cookie – Aufbau

- Key und Value
- Attribute (Direktiven)
  - `Domain`, `Path`
    - an Domäne und Sub-Domänen
    - nicht für andere Domänen setzbar
  - `Expires`
  - `Max-Age` in Sekunden; (wird von IE nicht unterstützt)
  - `Secure`, `HttpOnly` (siehe nächste Folie)

# Cookie – Arten

- Session
  - Prozessende $\rightarrow$ gelöscht
  - kein Ablaufdatum gesetzt (weder `Expires` noch `Max-Age`)
- Persistent
  - Ablaufdatum gesetzt!
    - relativ zur Zeit des Clients (nicht des Servers)
  - auch für Tracking!
- Secure
  - nur über TLS
    - trotzdem keine sensitive Daten in Cookie ablegen!
  - `Secure` Flag gesetzt

# Cookie – Arten – 2

- HTTP-only
  - keine Verwendung in JavaScript
  - Zugriff auf Cookies in JavaScript:
    - setzen (erzeugen), z.B.:

```
document.cookie = "sessionid=1234";
document.cookie = "settings=high risk";
```

    - lesen, z.B. `console.log(document.cookie)` → `sessionid=1234; settings=high risk`
  - `HttpOnly` Flag gesetzt
- SameSite
  - verhindert, dass Cookie im Zuge von Cross-site Requests an andere Site gesendet wird
  - `SameSite` Flag
    - `strict`
    - `lax`
  - seit 11/2017: Firefox, Chrome, Opera
  - → siehe Folien `websec.pdf`

# Cookie – Ablauf

- Server

```
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: sessionid=1234; Path=/private;
  Secure; Max-Age: 3600
Set-Cookie: settings=high risk; Expires=
  Wed, 09 Jun 2021 10:18:14 GMT; HTTPOnly;
Set-Cookie: trackid=4711
```

- Client

```
GET /private HTTP/1.0
Cookie: sessionid=1234; settings=high risk;
  trackid=4711
```

```
GET / HTTP/1.0
Cookie: settings=high risk; trackid=4711
```

# Cookie und Session

- Übertragen des gesamten Zustandes
  - z.B. alle Produkte im Warenkorb
  - Größe der Daten des Zustandes!
  - Sicherheit!
- Übertragen der Session-ID
  - Daten am Server in Speicher oder DB
  - üblicherweise lange Folge von zufälligen Zeichen
  - Größe der Daten des Zustandes irrelevant
  - meist nach erfolgter Authentifizierung
- Supercookie
  - ”gehört” zu einer Top-Level Domain (z.B. `.com` oder `.ac.at`)
    - Public Suffix List von Mozilla (Nutzung von Firefox, Chrome,…)
    - …werden ignoriert
  - Achtung: nicht exakt definiert! (siehe Tracking)

# Cookie – Third-Party

- gehört zu einer anderen Domain als die in ”URL-Leiste”
- zum Tracken!
  1. Benutzer surft zu news.foo.com
     - Website bindet Werbung von ad.xyz.com ein
     - diese setzt Cookie für Domäne ad.xyz.com
  2. Benutzer surft zu news.bar.com
     - Website bindet Werbung von ad.xyz.com ein
     - erstes Cookie wird an ad.xyz.com gesendet
     - Website setzt wiederum ein Cookie für Domäne ad.xyz.com
  3. Cookies werden immer an ad.xyz.com gesendet, wenn Requests an diese gesendet werden
- $\rightarrow$ in Browser deaktivieren

# Cookie – Alternativen

- HTTP Basic Authentication (theoretisch)
- Hidden Form Fields
- URL mit query
  - wird von Java Serlets und PHP verwendet, wenn keine Cookies akzeptiert werden
- ausschließliche Verwendung von POST-Requests
- HTML5 Web Storage
  - Datenspeicherung im Browser (mind. 5MB)
    - getrennt nach Origin (Domäne, Protokoll, Port)
  - werden nicht ”automatisch” an Server gesendet
    - $\rightarrow$ JavaScript API

# Tracking

- Third-Party Cookie
- Zombie-Cookie
  - erzeugt sich nach Löschen wieder
  - gespeichert an mehreren Orten (z.B. Web Storage und LSO)
- Supercookie-2 (kein HTTP-Cookie)
  - Web Storage
  - Flash-Cookie $\equiv$ Local Shared Object (LSO)
- Supercookie-3
  - Unique Identifier Header (UIDH) Header eingefügt durch ISP

# Tracking – 2

- `ETag`
- Web beacon (Web bug, Zählpixel,…)
  - 1x1 transparente Pixelgraphik, IFRAME, style, script, object,…
  - WWW und E-Mail
  - $\rightarrow$ nur Text `;-)`
  - $\rightarrow$ kein JavaScript…
- Browser- und Geräte-Fingerprint
  - Betriebssystem, `User-Agent`, Bildschirmauflösung,…