Jetzt bitte Terminal und Chromium öffnen und überprüfen, ob eine Verbindung zum Internet steht.

HTTP und HTTP/2

Bachelor-Seminar "Web Technologies"

Michael Hochleitner
30.11.2017

Was ist HTTP?

- Hypertext Transfer Protocol
- Protokoll für Rechnernetze
- Senden von Datenpaketen
- Client und Server

HTTP spezifiziert die Gestalt der Datenpakete

Motivation zur Entwicklung von HTTP

Verteilen von Hypertext-Dokumenten in einem Rechnernetz → HTTP/0.9

Protokoll flexibler machen → HTTP/1.1

Latenz verringern → HTTP/2

Anfrage

Antwort

- Textprotokoll
- Request-Response

http://info.cern.ch/hypertext/WWW/Protocols/HTTP/AsImplemented.html

Im Terminal ausführen:
telnet info.cern.ch 80
GET /index.html

Im Browser aufrufen:
http://info.cern.ch/index.html

Grenzen von HTTP/0.9

- Schließen der TCP-Verbindung nach jeder Anfrage
- nur ein Dateiformat: HTML
- minimalistische Fehlerbehandlung

http://info.cern.ch/hypertext/WWW/Protocols/HTTP/AsImplemented.html

HTTP/1.1

- Status Codes
- Metadaten
- mehr Methoden

- Wiederverwendung von Verbindungen
- Pipelining

https://tools.ietf.org/html/rfc2616

Header

- Allgemeine Headerfelder
  - Date
- Anfrage Headerfelder
  - Expect : 100
- Antwort Headerfelder
  - Location
- Entity Headerfelder
  - Content-type

https://tools.ietf.org/html/rfc2616#section-4.2

Methoden

- GET
- HEAD

- POST
- PUT

- DELETE

- TRACE

https://tools.ietf.org/html/rfc2616#section-9

Status Codes

- 1xx Informational
  - 100 Continue
- 2xx Successful
  - 200 OK
- 3xx Redirection
  - 302 Found
- 4xx Client Error
  - 404 Not Found
- 5xx Server Error
  - 503 Service Unavailable

https://tools.ietf.org/html/rfc7231#section-6.2

Heutige Anforderungen an HTTP

- viele Objekte auf einer Internetseite
- viele Requests
- Beispiel: yahoo.de

In Chromium:
F12 drücken
yahoo.de anfragen

Heutige Anforderungen an HTTP

- viele Daten, wenig Zeit
- Ressource: TCP-Verbindung
  - begrenzte Anzahl

- effiziente Nutzung der begrenzten Ressource

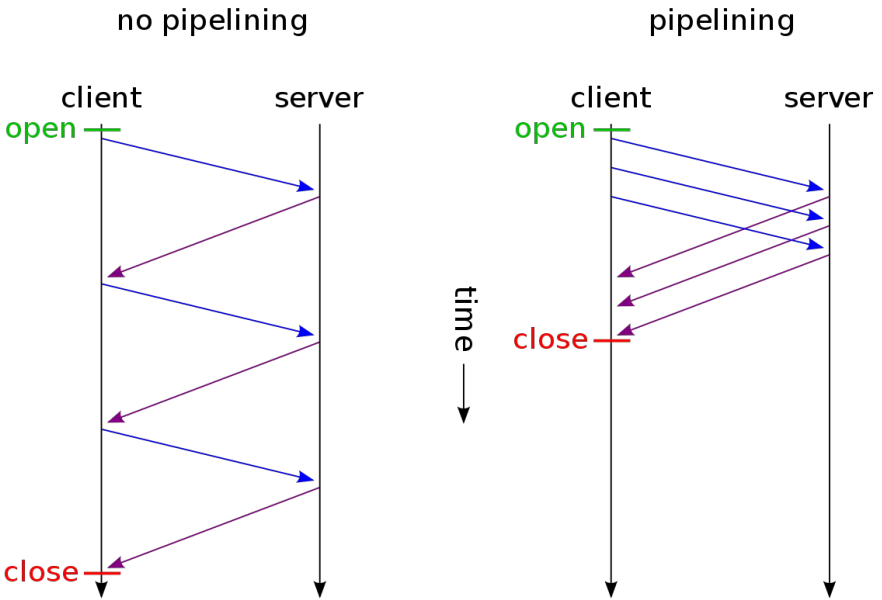
Nutzung von TCP-Verbindungen in HTTP/1.1

- Pipelining auf einer TCP-Verbindung
- Requests werden hintereinander bearbeitet
- Pipelining

https://tools.ietf.org/html/rfc7540#section-1

Pipelining

- FIFO



https://en.wikipedia.org/wiki/HTTP_pipelining#/media/File:HTTP_pipelining2.svg

Nutzung von TCP-Verbindungen in HTTP/1.1

- Stau auf einer Verbindung
- “Head of Line Blocking”

- öffnen einer neuen Verbindung
- Anzahl von Verbindungen begrenzt

- Stau auf allen Verbindungen möglich

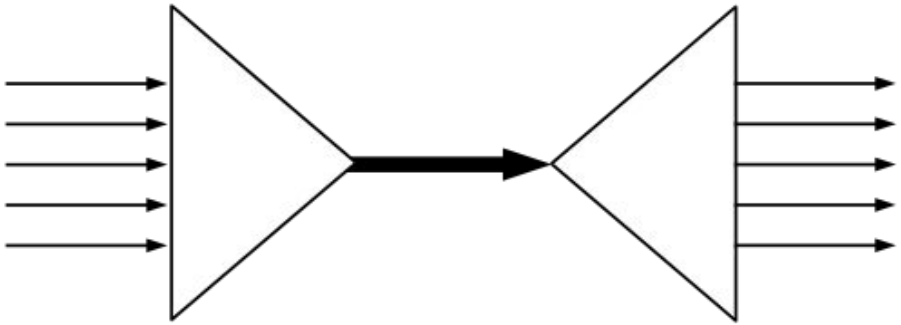
https://tools.ietf.org/html/rfc7540#section-1

HTTP/2

- Multiplexing
  - Netzwerkressourcen effizient nutzen
- Header Compression
- Binäres Protokoll

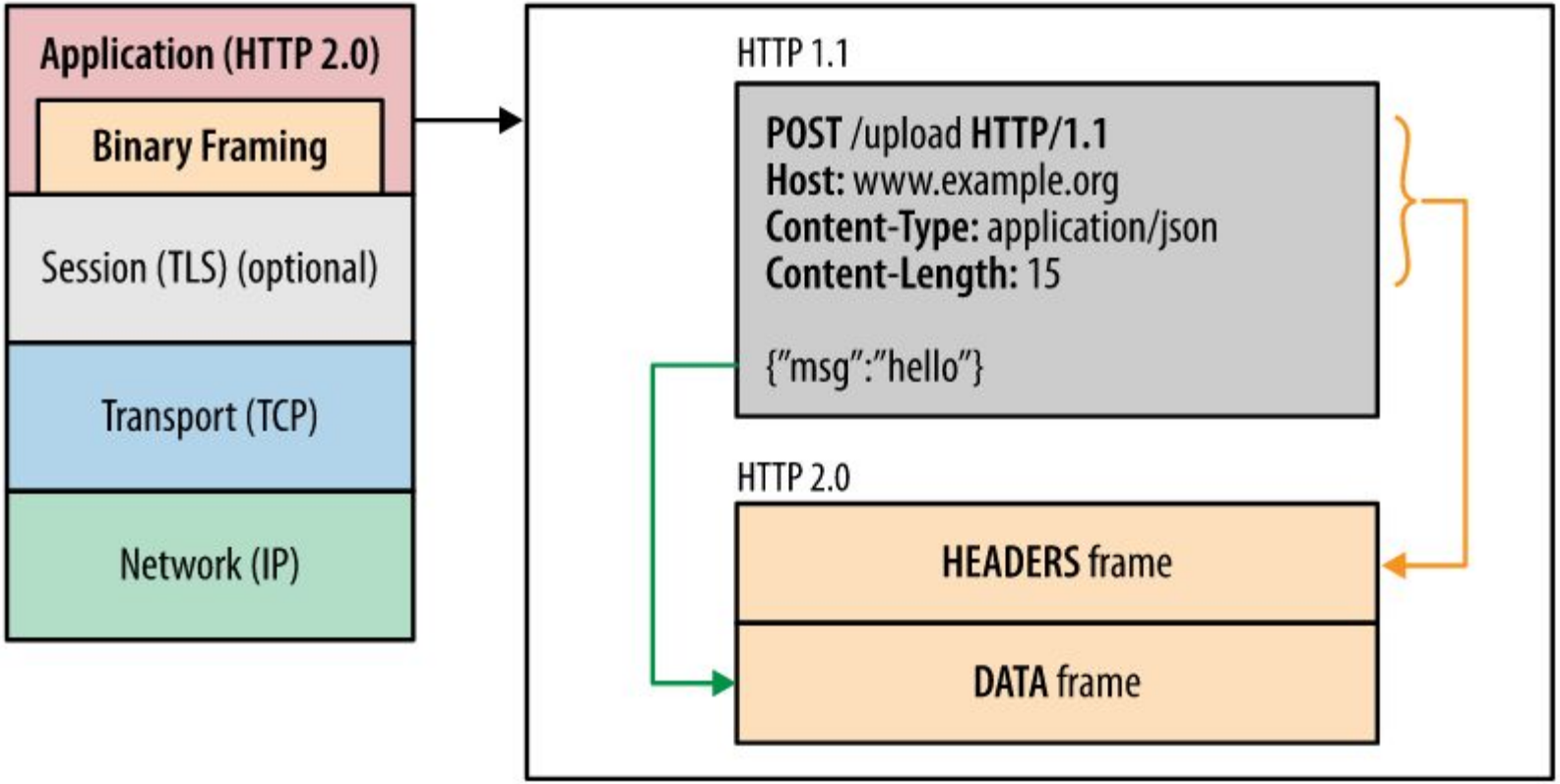
http://httpwg.org/specs/rfc7540.html

Multiplexing

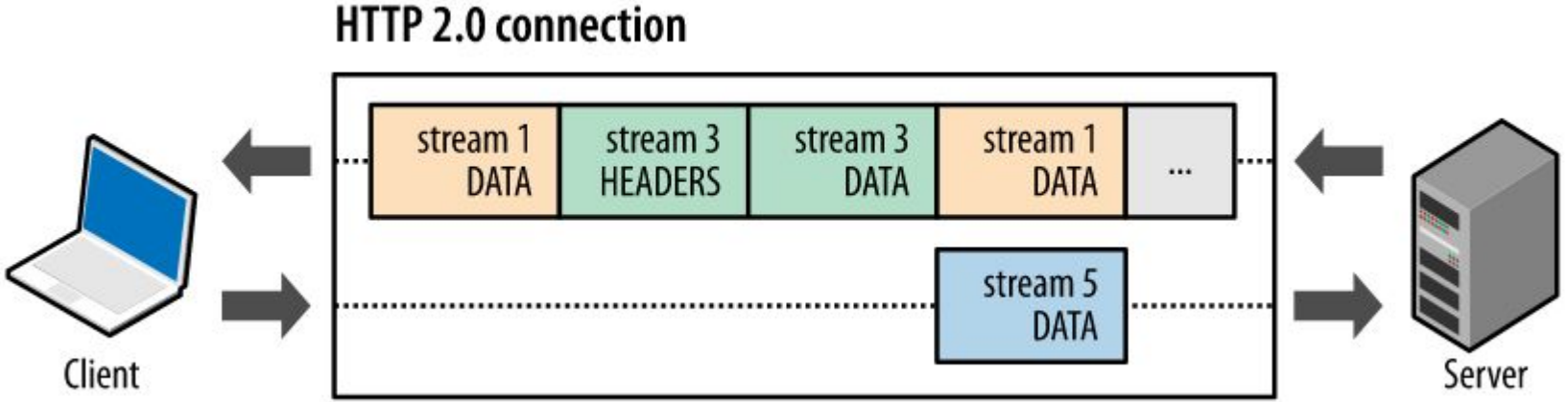


- Zusammenfassung von Verbindungen
- unabhängige Streams für Request-Response-Paare

https://upload.wikimedia.org/wikipedia/commons/thumb/6/6f/Multiplexing_diagram.svg/512px-Multiplexing_diagram.svg.png



https://developers.google.com/web/fundamentals/performance/http2/



https://developers.google.com/web/fundamentals/performance/http2/

Multiplexing

HOL-Blocking

"Multiplexing addresses these problems by allowing multiple request and response messages to be in flight at the same time; [...] " [1]

- Bearbeitungsdauer von Requests unabhängig von vorhergehenden Requests

1: https://bagder.gitbooks.io/http2-explained/en/part6.html

Zusammenfassung

- Netzwerkprotokoll
- verschiedene Versionen
- Verteilung von Dokumenten in einem Rechnernetz
  - HTTP/0.9
- Flexibilität
  - HTTP/1.1
- Latenz bei der Anzeige von Internetseiten verringern
  - HTTP/2

Quellen

RFC 2616, R. Fielding et al.

RFC 7540, M. Belshe et al.

https://bagder.gitbooks.io/http2-explained/content/en/, Daniel Stenberg - Author of cURL

https://developers.google.com/web/fundamentals/performance/http2/

Hier fangen die Detailfolien an, die nicht zum Vortrag gehören.

HTTP/1.1: viele sehr ähnliche Requests

viele Objekte pro Webseite

viele fast gleiche Anfragen

gute Möglichkeit für Kompression

https://bagder.gitbooks.io/http2-explained/en/part6.html

POST

The HTTP POST method is used to send user-generated data to the web server. For example, a POST method is used when a user comments on a forum or if they upload a profile picture. A POST method should also be used **if you do not know the specific URL** of where your newly created resource should reside. In other words, if a new forum thread is created and the thread path is not specified then you could use some like:

```
POST /forums HTTP/2.0
Host: https://yourwebsite.com/
```

Using this method, the URL path would be returned from the origin server and you would receive a response similar to:

```
HTTP/2.0 201 Created
Location: /forums/<new_thread>
```

In short, the **POST method should be used to create a subordinate** (or child) of the resource identified by the Request-URI. In the example above, the Request-URI would be `/forums` and the subordinate or child would be `<new_thread>` as defined by the origin.

https://www.keycdn.com/support/put-vs-post/

POST

Let's go back to the HTTP/1.1 RFC for the definition of POST.

> *The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI ... The posted entity is subordinate to that URI in the same way that a file is subordinate to a directory containing it, a news article is subordinate to a newsgroup to which it is posted, or a record is subordinate to a database.*

Practically speaking, POST is used to append a resource to an existing collection. In the following example you do not know the actual URL of the resource – the server decides the location where it is stored under the `user` collection.

```
POST /user HTTP/1.1
Host: http://sookocheff.com
```

https://sookocheff.com/post/api/when-to-use-http-put-and-http-post/

Flexibilität

- Header
  - verschiedene Dateitypen
- verschiedene Funktionen
  - Semantik von Nachrichten
- Definition neuer Header möglich
- Definition neuer Methoden möglich

```
GET /en-US/docs/Glossary/Simple_header HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/en-US/docs/Glossary/Simple_header
```

- Versionsinformation
- Header
- Übertragung verschiedener Dateitypen

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

200 OK
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 20 Jul 2016 10:55:30 GMT
Etag: "547fa7e369ef56031dd3bff2ace9fc0832eb251a"
Keep-Alive: timeout=5, max=1000
Last-Modified: Tue, 19 Jul 2016 00:59:33 GMT
Server: Apache
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding

- Status Code
- Header

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

Verhältnis von Request/Response-Paaren auf TCP-Verbindungen

Multiplexing

- Streams für Request/Response-Paare
- Streams unabhängig voneinander
- keine Blockierung
- Priorisierung von Streams

"A stream is an independent, bi-directional sequence of frames exchanged between the client and server within an http2 connection."

http://httpwg.org/specs/rfc7540.html

https://bagder.gitbooks.io/http2-explained/en/part6.html

HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used; the mapping of the HTTP/1.1 request and response structures onto the transport data units of the protocol in question is outside the scope of this specification.

https://tools.ietf.org/html/rfc2616#section-1.4

In HTTP/1.0, most implementations used a new connection for each request/response exchange. In HTTP/1.1, a connection may be used for one or more request/response exchanges, although connections may be closed for a variety of reasons (see section 8.1).

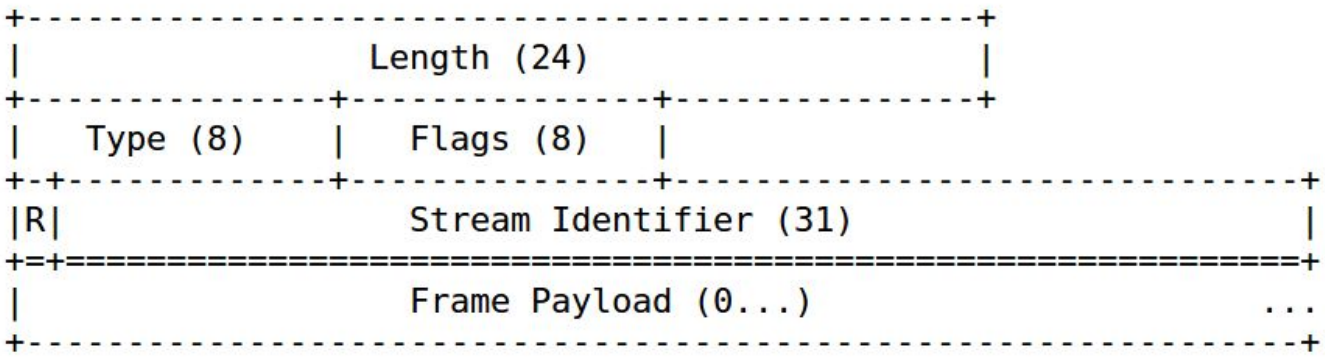
https://tools.ietf.org/html/rfc2616#section-1.4

Frames

- HTTP/1.1 Struktur
  - Protokollversion
  - Header
  - Body

https://tools.ietf.org/html/rfc2616

Frames



https://tools.ietf.org/html/rfc7540#section-4

- Länge des Payloads
- Typ des Frames

Server Push

- Server initiierte Datenübertragung