Main improvements of **http2** compared to **http**

Samet Yildiz

Was ist HTTP?

- **H**yper**T**ext **T**ransfer **P**rotocol
- Application Layer Protocol (OSI Schicht 7)
- Entwicklung angestoßen durch **Tim Berners-Lee** am **CERN** 1989

Erstpublikationen der Standards im RFC

HTTP/1.0 → RFC 1945: **1996**

HTTP/1.1 → RFC 2068: **1997**

HTTP/2 → RFC 7540: **2015**

SPDY: Vorgänger von HTTP/2

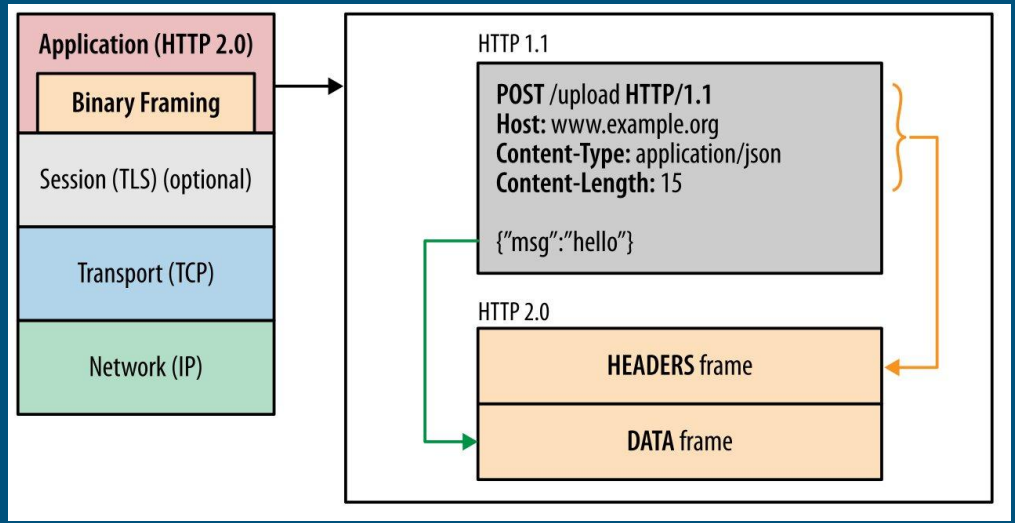
- Vorgänger SPDY(“speedy”) entwickelt durch Google → 2009
  - Chrome, Firefox und Opera unterstützt SPDY → 2012
  - eingestellt und abgelöst durch HTTP/2 → 2015

Main improvements

1. Binary Frame Layer
2. Header Compression HPACK
3. Multiplexing
4. Stream Prioritization
5. Server Push

1. Binary Frame Layer

- HTTP/1.X ist ein **text-basiertes** Protocol
- HTTP/2 ist ein **binär-basiertes** Protocol
  - Daten in Frames teilen
    - **Headers** Frames
    - **Data** Frames



https://developers.google.com/web/fundamentals/performance/http2/images/multiplexing01.svg

1. Binary Frame Layer

Vorteile:

- binäre Daten sind einfacher zu parsen als Text
- weniger fehleranfällig

Probleme beim Text parsen:

- Zeilenumbruch (Windows CRLF “\r\n” vs Linux LF “\n”)
- Leerzeichen
- Leere Zeilen

2. Header Compression

HTTP/1.X

- Text-basierte Header

HTTP/2

- im HPACK komprimierte Header
- Huffman kodiert
- Statische Tabelle
  - für häufig verwendete Header Felder
- Dynamische Tabelle
  - noch nicht gesehen Felder speichern für späteren Lookup

2. Header Compression

“vordefinierte und unveränderliche” statische Tabelle aus dem RFC 7541

- erstellt aus den beliebtesten Webseiten im Netz
- statt ganzen header namen zu verschicken nurnoch indizes verschicken und Tabellen lookup

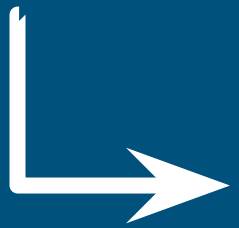
Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206
11	:status	304
12	:status	400
13	:status	404
14	:status	500
...		
54	server	
55	set-cookie	
56	strict-transport-security	
57	transfer-encoding	
58	user-agent	
59	vary	
60	via	
61	www-authenticate	

https://datatracker.ietf.org/doc/html/rfc7541#appendix-A

3. Multiplexing

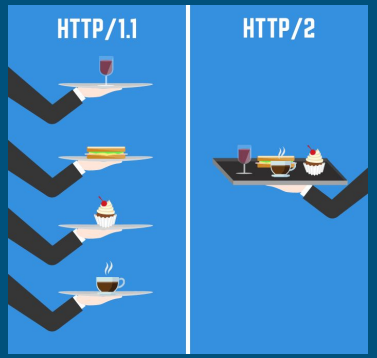
HTTP/1.X

- pro TCP verbindung darf nur eine Request gleichzeitig ausgeführt werden → mehrere TCP Verbindungen für parallelen Datenaustausch
  - overhead durch TCP Handshakes
- muss auf die beendigung einer Request warten bevor ein neuer starten kann



Problem: **Head of Line Blocking**

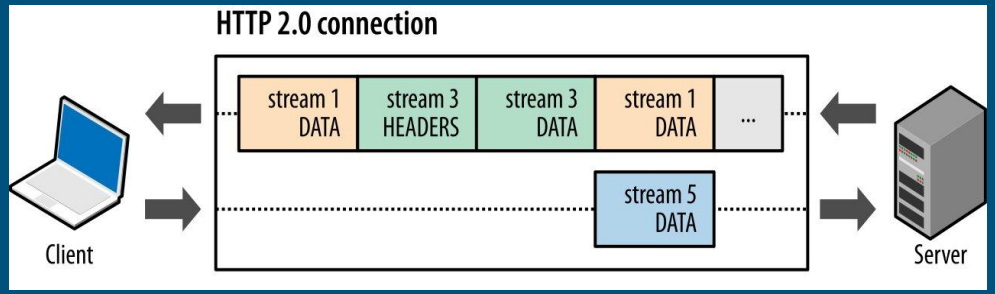
3. Multiplexing



https://i2.wp.com/css-tricks.com/wp-content/uploads/2017/02/image02.gif?ssl=1

HTTP/2

- nur **eine** TCP Verbindung zum Datenaustausch
  - verringert Server Last im gesamten Netz
- Requests und Responses können gleichzeitig verschickt und empfangen werden (=multiplexing)
- jeder Request stellt einen Stream dar mit einem dazugehörigen Identifier



https://developers.google.com/web/fundamentals/performance/http2/images/multiplexing01.svg

4. Stream Prioritization

HTTP/1

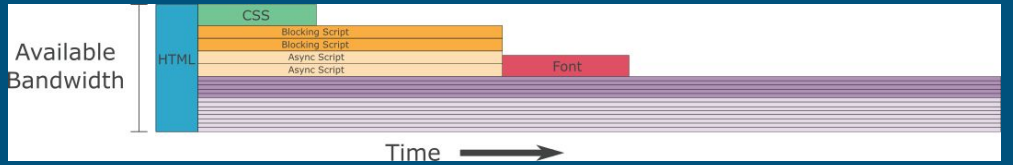
- auf jede Request muss gewartet werden sodass Client selber priorisiert was er als erstes anfragt verteilt über mehrere TCP Verbindungen
- Server empfängt und bearbeitet requests in chronologischer Reihenfolge pro TCP Connection

4. Stream Prioritization

HTTP/2

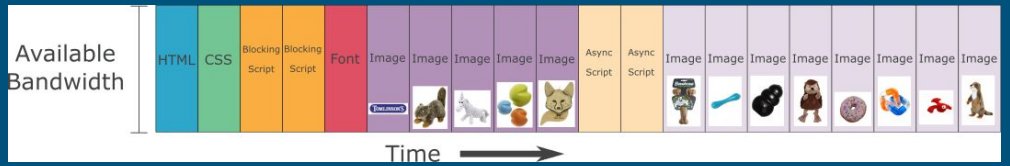
- Requests kommen alle zeitnahe an → **Multiplexing**
- Client kann priorisieren was er als erstes empfangen will
- verschiedene Browser haben verschiedene Priorisierung Strategien

Safari



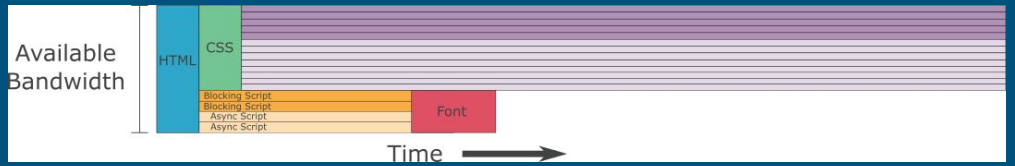
https://blog.cloudflare.com/content/images/2019/05/safari-1.png

Chrome



https://blog.cloudflare.com/content/images/2019/05/chrome-1.png

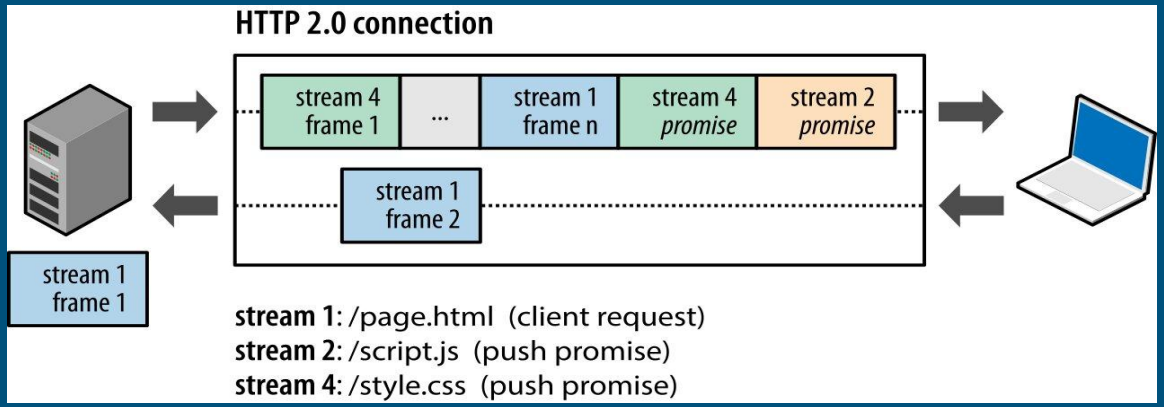
Firefox



https://blog.cloudflare.com/content/images/2019/05/firefox-1.png

5. Server Push

- Server kann Ressourcen senden bevor der Client sie anfragt durch Push_Promise frames gefolgt von den Daten
  - Client erspart sich die Anfrage der Ressourcen
- Client kann die Daten cachen oder ablehnen.



https://developers.google.com/web/fundamentals/performance/http2/images/push01.svg

Fazit

- HTTP/2 ist schneller und effizienter als HTTP/1.X
- HTTP/2 verringert den Overhead der im ganzen Netzwerk entsteht durch HTTP/1.X
- HTTP/2 bietet Flexibilität für Client/Server Entwickler wie sie den Datenverkehr gestalten wollen

Quellen

- https://developers.google.com/web/fundamentals/performance/http2
- https://blog.cloudflare.com/better-http-2-prioritization-for-a-faster-web/
- https://http2.github.io/faq/
- https://www.digitalocean.com/community/tutorials/http-1-1-vs-http-2-what-s-the-difference
- https://en.wikipedia.org/wiki/HTTP/2
- https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- https://datatracker.ietf.org/doc/html/rfc7541 - HPACK
- https://datatracker.ietf.org/doc/html/rfc7540 - HTTP/2

Demo Seite HTTP/2 vs HTTP/1.1 https://http1.golang.org/gophertiles?latency=0