Verteilte Systeme

HTTP 1.1 – Teil B

by

Dr. Günter Kolousek

Chunked Transfer-Encoding

- Wenn Server die Länge (noch) nicht kennt!
- Antwort wird in Teilen (chunks) gesendet
- `Transfer-Encoding: chunked`
- beliebige binäre Daten

Chunked Transfer-Encoding – 2

- Aufbau eines Chunk-Stroms
  1. `chunk*`
  2. `last-chunk` (Endemarkierung)
  3. `trailer-part` (zusätzliche Header)
  4. `\r\n`
- Aufbau eines `chunk`
  1. Größe (Hexadezimal) `\r\n`
  2. Daten `\r\n`
- Aufbau des `last-chunk`: `0\r\n`
- Aufbau von `trailer-part`: `(Key: Value\r\n)*`
  - explizit verboten: `Transfer-Encoding`, `Content-Length`, `Trailer`

Chunked Transfer-Encoding – 3

```
HTTP/1.1 200 OK
Date: Sun, 21 Aug 2016 23:59:59 GMT
Content-Type: text/plain
Transfer-Encoding: chunked
Trailer: some-footer, another-footer

1a
abcdefghijklmnopqrstuvwxyz
10
1234567890abcdef
0
some-footer: some-value
another-footer: another-value
```

Chunked Transfer-Encoding – 4

```
HTTP/1.1 200 OK
Date: Sun, 21 Aug 2016 23:59:59 GMT
Content-Type: text/plain
Content-Length: 42
some-footer: some-value
another-footer: another-value

abcdefghijklmnopqrstuvwxyz1234567890abcdef
```

Persistent Connections

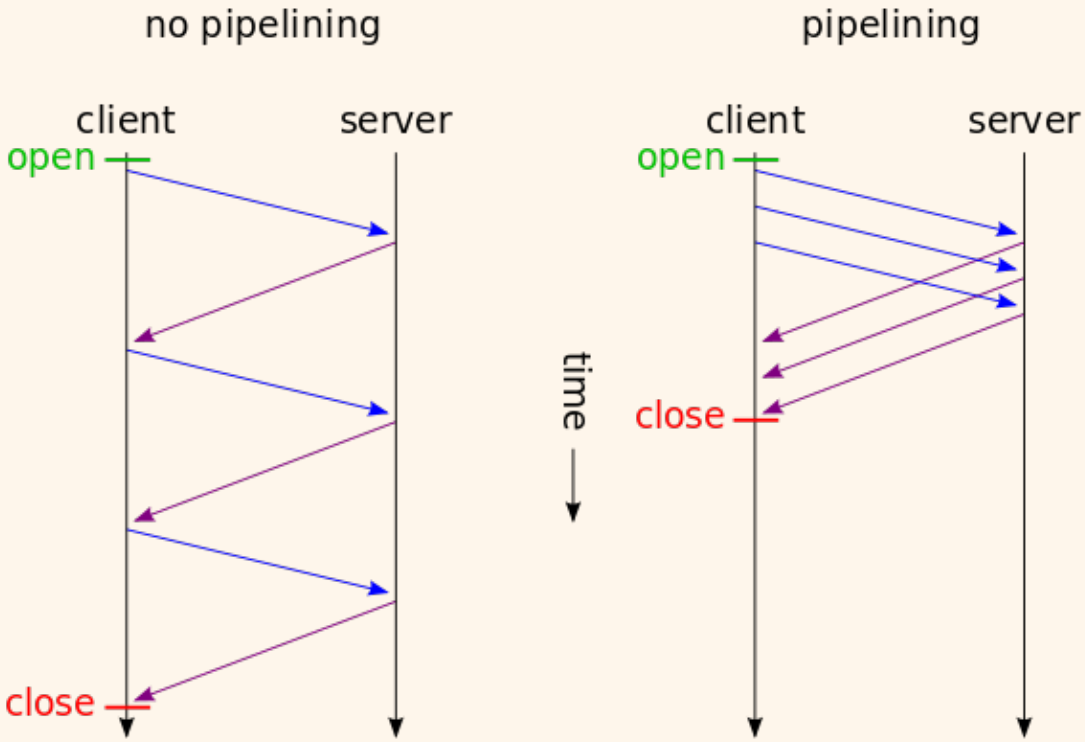
- bis HTTP/1.0 (TCP-)Verbindung $\rightarrow$ geschlossen
  - HTTP/1.0 Client und persistente Verbindung gewünscht $\rightarrow$ `Connection: keep-alive`
- HTTP/1.1 persistente Verbindung $\rightarrow$ default
  - Keine persistente Verbindung gewünscht $\rightarrow$ `Connection: close`
- kann ein Client/Server keine persistente Verbindungen $\rightarrow$ `Connection: close`!
- meist mehrere parallele Verbindungen!
- Länge des Body muss bekannt sein
- wird für Pipelining verwendet

Länge des Body

- hier geht es nur um *echte* Daten
  - kein HEAD, kein 1xx,…
- `Transfer-Encoding` vorhanden
  - `chunked` am Ende → Länge durch `chunked`
  - `chunked` nicht am Ende
    - Response → bis zum Schließen der Verbindung
    - Request → 400 (Bad Request)
- `Content-Length` vorhanden → ✓
- weder `Transfer-Encoding` = `chunked` noch `Content-Length`
  - Request → keine Daten, d.h. Länge = 0 (z.B. GET)
  - Response → bis zum Schließen der Verbindung

Pipelining

- Senden mehrerer Requests *ohne* auf Response zu warten
  - $\rightarrow$ Images, CSS, JavaScript!



Quelle Wikipedia

Pipelining – 2

- *möglich* wenn persistente Verbindungen
  - Weitere Verbesserung nach ”persistenter Verbindung”
- Server *kann* parallel verarbeiten, wenn Methoden ”safe” sind
- Antworten immer in Reihenfolge der Anfragen!!
- kein Pipelining nicht-idempotenter Requests (SOLL)!
  - Wenn nicht-idempotenter Request, dann warten mit Pipelining bis Antwort *dieses* Requests eingetroffen
- Client soll unbeantwortete Requests wieder stellen, wenn Verbindung abgebrochen
- Problem des HOL Blocking (Head-Of-Line Blocking) besteht trotzdem noch
  - HOL: nachfolgende Pakete sind blockiert wegen erstem Paket
  - $\rightarrow$ mehrere parallele Verbindungen
  - $\rightarrow$ HTTP/2
- seit 2018: pipelining nicht mehr default-mäßig aktiviert in modernen Browsern (wg. HOL und buggy Browser)!

Verbindungsabbau

- Verbindungen können immer geschlossen werden
  - beabsichtigt
  - unbeabsichtigt (aka Abbruch)
- Verbindungsabbau ausgehend vom
  - Client
  - Server
- Wiederholung nach Wiederaufbau der Verbindung
  - bei idempotenten Methoden: ok
  - *nicht* bei nicht-idempotenten Methoden
    - außer Client ist sicher, dass ok

Verbindungsabbau – Client

1. Client sendet `Connection: close`
   - darf danach nicht mehr senden!
2. Server sendet Response (soll `Connection: close` enthalten)
   - darf keine weiteren Requests verarbeiten
3. Server schließt ausgehenden Stream
4. Client empfängt Response von Server und schließt Verbindung
5. Server schließt auf jeden Fall wenn ACK vom Response

Verbindungsabbau – Server

1. Server sendet `Connection: close`
   - darf danach nicht mehr senden!
2. Server schließt ausgehenden Stream
3. Client empfängt `close`
   - darf danach nicht mehr senden!
4. Client schließt Verbindung
   - soll nicht annehmen, dass Requests in der Pipeline vom Server verarbeitet werden
5. Server schließt auf jeden Fall wenn ACK vom Response