



**Fachbereich 2**  
**Informatik und Ingenieurwissenschaften**

# **Abschlussarbeit**

im Studiengang Informatik  
zur Erlangung des akademischen Grades  
Bachelor of Science (B.Sc.)

---

**Untersuchung des Protokolls HTTP/2  
hinsichtlich des Laufzeitverhaltens und  
Datendurchsatzes im Vergleich zu seinen  
Vorgängern**

---

**Autor:** Tobias Schmeiser

**Matrikelnummer:** 988627

**Prüfer:** Prof. Dr. Christian Baun

**Korreferent:** Alexander Mützel

**Eingereicht am:** 07.09.2015

## Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Abschlussarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ort, Datum:

.....

(Unterschrift)

## Danksagung

Zunächst einmal möchte ich mich bei Herrn Prof. Dr. Christian Baun und Herrn Alexander Mützel für die Unterstützung, vor und während dieser Arbeit bedanken.

Für das Korrekturlesen dieser Arbeit möchte ich mich ganz herzlich bei Kim von der Wehl und Christina von der Wehl bedanken. Ein besonderer Dank geht außerdem an Johannes Leist und David Weichert, für den fortwährenden Rat bei dieser Arbeit.

## Zusammenfassung

In dieser Arbeit wird das neue Anwendungsprotokoll HTTP/2 hinsichtlich Laufzeitverhalten und Datendurchsatz untersucht. Dabei werden die maßgeblichen Neuerungen von HTTP/2 vorgestellt, außerdem werden die verschiedenen Versionen (0.9, 1.0, 1.1 und 2.0) des Hypertext Transfer Protocols miteinander verglichen. Das Ziel dieser Arbeit ist es herauszufinden, ob und inwieweit das Abrufen von Webseiten mit HTTP/2 schneller ist als mit HTTP/1.X.

Um dies zu testen, wurde eine Testumgebung, mit einem HTTP/2-fähigen Webserver aufgebaut. In dieser wurden diverse Leistungstests in verschiedenen Testreihen durchgeführt. Dabei wurden die Protokollversionen HTTP/1.1 und HTTP/2, hinsichtlich ihres Laufzeitverhaltens und Datendurchsatzes miteinander verglichen.

Wie die anschließende Auswertung und Analyse ergab, muss im Ergebnis festgehalten werden, dass es in der Tat eine Geschwindigkeitssteigerung beim Abrufen von Webseiten mittels HTTP/2 gegenüber HTTP/1.1 gibt. Wie die Testreihen ergaben, ist dies jedoch nicht immer der Fall. In einzelnen Fällen ist das Abrufen mit HTTP/1.1 schneller.

## Abstract

This thesis examines the new application protocol HTTP/2, especially its run-time behavior and data throughput. The relevant new features of HTTP/2 are presented and the different versions (0.9, 1.0, 1.1, and 2.0) of the Hypertext Transfer Protocol are compared with each other. The aim of this study is to find out if and in how far the retrieval of websites with HTTP/2 is faster than with HTTP/1.X.

To test this, a test environment, with a HTTP/2 web server was set up. Therein various performance tests were run in different test series. The protocol versions HTTP/1.1 and HTTP/2 were compared with respect to their run-time behavior and data throughput. Afterwards the test results were evaluated and analyzed.

The conclusion of this thesis is that there is indeed a performance increase when retrieving web pages using HTTP/2. However, as is shown in different test series, this is not always the case. In particular cases HTTP/1.1 can be faster.

# Inhaltsverzeichnis

<b>Eidesstattliche Erklärung</b>	<b>I</b>
<b>Danksagung</b>	<b>II</b>
<b>Zusammenfassung</b>	<b>III</b>
<b>Abstract</b>	<b>IV</b>
<b>Inhaltsverzeichnis</b>	<b>V</b>
<b>1 Einleitung</b>	<b>1</b>
<b>2 HTTP Versionen im Vergleich</b>	<b>3</b>
2.1 HTTP/0.9 . . . . .	3
2.1.1 Verbindungsaufbau . . . . .	4
2.1.2 Anfrage . . . . .	5
2.1.3 Antwort . . . . .	5
2.1.4 Verbindungsabbau . . . . .	5
2.2 HTTP/1 . . . . .	6
2.2.1 Verbindungsaufbau . . . . .	7
2.2.2 Anfrage . . . . .	7
2.2.3 Antwort . . . . .	8
2.2.4 Verbindungsabbau . . . . .	9
2.3 HTTP/1.1 . . . . .	9
2.3.1 Verbindungsaufbau . . . . .	10
2.3.2 Anfrage . . . . .	10
2.3.3 Verbindungsabbau . . . . .	10
2.4 HTTP/2 . . . . .	12
2.4.1 SPDY und HTTP/2 . . . . .	13
2.4.2 Gegenüberstellung von HTTP/1.1 und HTTP/2 hinsichtlich der Datenübertragung . . . . .	14
2.4.3 Stream Priorisierung . . . . .	21
2.4.4 Server Push . . . . .	23
2.4.5 HTTP Header Komprimierung mittels HPACK . . . . .	24
2.4.6 Wechsel zu HTTP/2 . . . . .	26
2.4.7 HTTP/2 Browser-/Server-Unterstützung . . . . .	27
<b>3 Aufbau der Testumgebung</b>	<b>28</b>
3.1 Anforderungen an die Testumgebung . . . . .	28
3.2 Host-System für Webserver . . . . .	28
3.3 Client-System . . . . .	31
3.4 Programme zum Messen des Datendurchsatzes . . . . .	31
<b>4 Leistungsuntersuchung</b>	<b>33</b>

4.1	Leistungsuntersuchung und Vergleich von HTTP/1.1 und HTTP/2 . . . . .	33
4.2	Testszenarien . . . . .	33
4.3	Vorwort zu den Leistungstests . . . . .	35
4.4	Testreihe 1 . . . . .	37
4.4.1	Test 1 . . . . .	37
4.4.2	Test 2 . . . . .	38
4.4.3	Test 3 . . . . .	39
4.4.4	Test 4 . . . . .	40
4.4.5	Zwischenfazit Testreihe 1 . . . . .	40
4.5	Testreihe 2 . . . . .	41
4.5.1	Test 1 . . . . .	41
4.5.2	Test 2 . . . . .	42
4.5.3	Test 3 . . . . .	42
4.5.4	Test 4 . . . . .	43
4.5.5	Zwischenfazit Testreihe 2 . . . . .	44
4.6	Testreihe 3 . . . . .	45
4.6.1	Test 1 . . . . .	45
4.6.2	Test 2 . . . . .	46
4.6.3	Test 3 . . . . .	46
4.6.4	These . . . . .	47
4.6.5	Test 4 . . . . .	47
4.6.6	Zwischenfazit Testreihe 3 . . . . .	48
4.7	Testreihe 4 . . . . .	49
4.7.1	Test 1 <a href="https://www.google.de">https://www.google.de</a> . . . . .	49
4.7.2	Test 2 <a href="https://http2.akamai.com">https://http2.akamai.com</a> . . . . .	50
4.7.3	Test 3 <a href="https://www.youtube.com">https://www.youtube.com</a> . . . . .	51
4.7.4	Test 4 <a href="https://twitter.com">https://twitter.com</a> . . . . .	52
4.7.5	Zwischenfazit Testreihe 4 . . . . .	53
<b>5</b>	<b>Fazit</b>	<b>54</b>
<b>6</b>	<b>Ausblick</b>	<b>57</b>
<b>7</b>	<b>Literaturverzeichnis</b>	<b>59</b>
	<b>Anhang</b>	<b>VII</b>
<b>A</b>	<b>Abbildungsverzeichnis</b>	<b>VII</b>
<b>B</b>	<b>Tabellenverzeichnis</b>	<b>VIII</b>
<b>C</b>	<b>Listingverzeichnis</b>	<b>VIII</b>
<b>D</b>	<b>Messergebnisse</b>	<b>IX</b>

# 1 Einleitung

Das Internet hat in den vergangenen Jahren zunehmend an Bedeutung gewonnen. Angefangen mit dem ARPANET, welches Ende der 1960er Jahre dazu diente, US-Militäreinrichtungen und US-amerikanische Universitäten, welche für das US-Verteidigungsministerium forschten, miteinander zu verbinden und den Informationsaustausch zu ermöglichen. Zu dieser Zeit hatten nur einige wenige Leute Zugang zum Internet, 1971 hatte das gesamte ARPANET 23 Hosts. [ARP]

Am CERN, der europäischen Organisation für Kernforschung, entwickelten ab 1989, Roy Fielding und Tim-Berners-Lee das Hypertext Transfer Protocol (HTTP). Dieses Protokoll bildete zusammen, mit den Konzepten URL und HTML den Grundstein für das World Wide Web, wie wir es heute kennen. Es wurde 1991 eingeführt. Diese erste Version des Hypertext Transfer Protocols wurde nachträglich als HTTP/0.9 bezeichnet. In dieser Version waren die Funktionen einfach gehalten und nicht sehr vielfältig, jedoch erfüllte es seinen damaligen Zweck, welcher ausschließlich aus dem Laden von HTML-Dokumenten bestand. Im Laufe der Jahre, wurde das Hypertext Transfer Protocol weiterentwickelt und in seinen Funktionen erweitert. Im Jahr 1996 wurde die Version HTTP/1.0 als RFC 1945 offiziell verabschiedet und im Jahre 1999 folgte schließlich die Version HTTP/1.1 spezifiziert im RFC 2616. Seit seiner Einführung wird HTTP/1.1 unverändert hauptsächlich zum Abrufen von Webseiten verwendet.

Das World Wide Web war bei der Einführung von HTTP/1.1 weitgehend statisch und textbasiert. Seitdem hat ein Wandel stattgefunden. Heutzutage sind Webseiten interaktiv, mit dynamischen Inhalten versehen und haben teilweise hunderte Ressourcen, die geladen werden müssen. Das zugrunde liegende Protokoll, welches den Transfer der Ressourcen zwischen Client und Server bewerkstelligt, hat sich jedoch nicht verändert.

In den vergangenen 25 Jahren, seit der Einführung von HTTP in seiner ersten Version (HTTP/0.9), hat sich das Internet rasant weiterentwickelt. Es war nicht absehbar, welche Dimensionen das globale Netz



einmal annehmen wird. Jedoch erkannte man schnell, dass HTTP in der Version HTTP/0.9 nicht ausreichte, aus diesem Grund entwickelte man das Protokoll weiter, bis hin zur Version HTTP/1.1. Diese Version bot viele Verbesserungen in Sachen Leistung gegenüber seinen Vorgängern. Jedoch liegt die Einführung von HTTP in der Version HTTP/1.1, also die Basis des Datentransfers zwischen Client und Server, mittlerweile gut 16 Jahre zurück (Stand: Juli 2015). In diesem Zeitraum ist die Leistung von Computern enorm gestiegen (siehe Mooresches Gesetz<sup>1</sup>).

Im Zuge des Leistungszuwachses moderner Computer spielt die Netzwerkperformance gegenüber der CPU-Leistung und dem Arbeitsspeicher eine immer größere Rolle. Bei der Übertragung von Daten zwischen zwei Systemen (Client-Server) wird die Bandbreite heutiger Übertragungstechnologien nicht voll ausgenutzt. Hier soll die neue Protokoll Version HTTP/2 Abhilfe schaffen. In dieser Arbeit soll das Laufzeitverhalten und der Datendurchsatz von HTTP/2 im Vergleich zu seinen Vorgängern analysiert werden.

In Kapitel 2 dieser Arbeit, werden die verschiedenen Versionen des Hypertext Transfer Protocols vorgestellt und in ihrer Funktionalität miteinander verglichen. Der Aufbau der verwendeten Testumgebung, samt diverser Programme wird in Kapitel 3 beschrieben. Die anschließende Leistungsuntersuchung der beiden Anwendungsprotokolle HTTP/1.1 und HTTP/2 findet in Kapitel 4 statt. In Kapitel 5 wird das Fazit dieser Arbeit präsentiert und in Kapitel 6 ein Ausblick für HTTP/2. Außerdem werden etwaige zusätzliche Leistungsuntersuchen vorgestellt.

---

<sup>1</sup>Das mooresche Gesetz besagt, dass sich die Rechenleistung von Computern regelmäßig verdoppelt, also exponentiell wächst. Als Zeitraum werden 18 Monate angegeben. [MOO]

## 2 HTTP Versionen im Vergleich

Im folgenden Abschnitt werden die Funktionsweisen der verschiedenen Protokollversionen des Hypertext Transfer Protocols genauer beschrieben. Hierbei wird besonders auf den Verbindungsaufbau, den Anfrage-/Antwort-Mechanismus und letztlich den Verbindungsabbau eingegangen.

### 2.1 HTTP/0.9

Das Hypertext Transfer Protocol in der Version HTTP/0.9 wurde im Jahre 1991, eingeführt und diente hauptsächlich dem Abruf von HTML-Dokumenten. Es ist in seiner Funktionsweise ziemlich einfach gehalten und in seinem Umfang stark eingeschränkt. Dies ist auch darauf zurückzuführen, dass das Hypertext Transfer Protocol ursprünglich für eine prototypische Umsetzung der Idee des World Wide Web gedacht war und im Gegensatz zu anderen Protokollen und Schnittstellen keine weitgehende Planung erfahren hatte. Tim Berners-Lee hatte die Verwendung anderer existierender Protokolle in Betracht gezogen, diese aber als unzureichend verworfen. [BL92] Er formulierte lediglich vier kurze Anforderungen an das neue Protokoll:

- Eine Teilmenge der Funktionen des Dateiübertragungsprotokolls FTP (GET)
- Die Möglichkeit von Indexsuchanfragen
- Automatische Formatwahl (automatic format negotiation)
- Weiterleitungen von Clients auf andere Server

Dementsprechend besitzt HTTP/0.9 ausschließlich die GET-Methode (die genauso heißt, wie die entsprechende FTP-Methode). Mittels dieser Methode kann ein Client Ressourcen von einem Server anfordern. Zur Übertragung von Daten kommt das Transmission Control Protocol (TCP) zum Einsatz. [Wil99]

### 2.1.1 Verbindungsaufbau

Der Client stellt eine TCP-Verbindung zum Server her, indem er den Domain-Namen beziehungsweise die IP-Adresse und den entsprechenden Port angibt, dieser ist standardmäßig 80<sup>2</sup>. Anschließend akzeptiert der Server die Verbindung.

Der TCP-Verbindungsaufbau erfolgt via Dreiwege-Handshake (siehe Abbildung 1). Bei dem Client und Server in drei Schritten Kontrollinformationen austauschen.

---

```
1 tobias@debian:~$ telnet google.de 80
2 Trying 216.58.213.3...
3 Connected to google.de.
```

---

Listing 1: Verbindungsaufbau HTTP/0.9

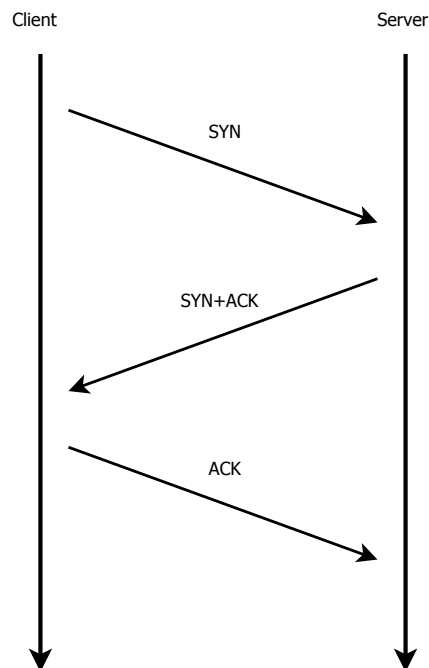


Abbildung 1: Dreiwege-Handshake

---

<sup>2</sup>Telnet dient zum Aufbau der Verbindung

Der Client sendet ein Segment mit der Aufforderung zum Synchronisieren (gesetztes SYN-Bit). Anschließend sendet der Server ein Segment an den Client, in dem er die angeforderte Synchronisation bestätigt (gesetztes ACK-Bit) und gleichermaßen fordert er den Client ebenfalls auf, zu synchronisieren (gesetztes SYN-Bit). Im letzten Schritt des Dreiwege-Handshakes, bestätigt der Client dem Server mit einem Segment in dem das ACK-Bit gesetzt ist.[Bau12]

### 2.1.2 Anfrage

Nach erfolgreicher Verbindung zum Server, sendet der Client eine GET-Anfrage an den Server. Diese ist wie folgt aufgebaut:

```
GET[/[Pfad der gewünschten Ressource]][/[PROTOKOLL-VERSION]
```

---

```
1 GET /index.html HTTP/0.9
```

---

Listing 2: Anfrage HTTP/0.9

### 2.1.3 Antwort

Der Server antwortet mit der vom Client angefragten Ressource, zum Beispiel:

---

```
1 <BOLD>DAS IST EIN HTML-TEXT</BOLD>
```

---

Listing 3: Antwort HTTP/0.9

Man beachte: bei HTTP/0.9 gab es noch keine Antwort-Header.

### 2.1.4 Verbindungsabbau

Nach der kompletten Übertragung, der vom Client angeforderten Ressource beendet der Server die TCP-Verbindung automatisch. Der Verbindungsabbruch signalisiert dem Client, das Ende der angeforderten

Ressource. Sofern der Client eine weitere Anfrage an den Server senden möchte, muss eine erneute Verbindung zwischen beiden Partnern hergestellt werden.

Der Umstand, dass HTTP/0.9 ausschließlich zur Übertragung von Texten (HTML-Code) geeignet ist, und keine Möglichkeit zum Austausch von Statuscodes implementiert ist, führte dazu, dass man das Protokoll weiterentwickelte. Das Ergebnis ist HTTP/1.

## 2.2 HTTP/1

Durch die rasante Entwicklung und Ausbreitung des Internet, wurde schnell klar, dass das Hypertext Transfer Protocol in der Version HTTP/0.9 mit seiner Möglichkeit, einfache HTML-Texte zu übermitteln, nicht ausreichte. Somit begann man postwendend, im Jahre 1992 mit der Weiterentwicklung des Protokolls. Im Mai 1996 wurde eine neue Protokollversion freigegeben. Im RFC 1945 wird das Hypertext Transfer Protocol ausführlich beschrieben.[BL96]

Wesentlich verbessert wurde in HTTP/1 gegenüber der Vorgängerversion HTTP/0.9 der Umgang mit MIME-Medientypen<sup>3</sup>. Außerdem werden ab dieser Protokollversion nicht allein die bloßen Daten übermittelt, sondern auch Header. Somit ist der Server nun in der Lage, mehr als nur die Entität als solche an den Client zu senden, sondern auch Informationen über den Medientyp und Statuscodes (Response-Format) an den Client zu übermitteln. Des Weiteren bietet HTTP/1 ein flexibles Nachrichtenformat, bestehend aus Anfangszeile gefolgt von beliebig vielen Headerfeldern. Die Headerfelder dienen der Übermittlung für übertragungsrelevante Informationen. Dies ist notwendig, da HTTP/1.X ein zustandloses Protokoll ist. Das heißt, alle Informationen die für die Übertragung der Ressource relevant sind, werden in der übertragenen Nachricht selbst, in Form von Headern gespeichert.

Zu der in HTTP/0.9 einzigen Methode, GET, sind in HTTP/1 nun die Methoden HEAD, POST, PUT, DELETE, LINK und UNLINK hinzu-

---

<sup>3</sup>Multipurpose Internet Mail Extensions (MIME)

gekommen. Mithilfe der POST-Methode ist es für Clients möglich, Informationen an den Server zu senden.

Der Bereich des Hypertext Transfer Protocols in der Version HTTP/1, welcher keine Veränderung erfahren hat, ist der Netzwerkbereich. Weiterhin basiert eine Verbindung auf einer einzigen Anfrage/ Antwort-Interaktion, bei der der Server die Verbindung nach dem Senden der angeforderten Ressource abbricht.[Wil99]

### 2.2.1 Verbindungsaufbau

Der Verbindungsaufbau hat sich im Wesentlichen nicht geändert hinsichtlich HTTP/0.9 (siehe Verbindungsaufbau HTTP/0.9).

### 2.2.2 Anfrage

Bei der Anfrage einer Ressource von einem Client an den Server gibt es nun die Möglichkeit, zusätzliche Parameter beziehungsweise Argumente als Zusatzinformationen zu übermitteln. Diese Argumente werden in den HTTP-Headerfeldern des HTTP-Headers<sup>4</sup> übergeben.[BL96]

Zum Beispiel können hier die gewünschte Sprache oder ein bestimmter Zeichensatz, vom Client angefordert werden:

---

```
1 GET /index.html HTTP/1
2 Accept-Charset: utf-8
3 Accept-Language: de-DE
```

---

Listing 4: Anfrage HTTP/1

In dem obigen Fall werden zwei optionale HTTP-Headerfelder bei der HTTP-Anfrage verwendet. Der Client fordert die Datei INDEX.HTML vom Server an. Als Zeichensatz wird UTF-8 erwartet und als Sprache deutsch.

---

<sup>4</sup>Ein HTTP-Headerfeld ist eine Zeile im HTTP-Header. Der HTTP-Header ist die Gesamtheit aller HTTP-Headerfelder

### 2.2.3 Antwort

Mit HTTP/1 ist der Server in der Lage, Statuscodes an den Client zu senden, welche Informationen über die angeforderte Ressource enthalten. HTTP-Statuscodes bestehen aus drei Ziffern. Grundsätzlich gibt es fünf verschiedene Kategorien von HTTP-Statuscodes, die im RFC 1945 gelistet sind.

Statuscodes mit einer vorangestellten eins (1xx) signalisieren dem Client, dass die Bearbeitung der Anfrage noch andauert. Statuscodes die mit einer zwei beginnen (2xx) stehen für eine erfolgreiche Anfrage und dass die Antwort verwertet werden kann. Bei Statuscodes, die mit einer drei beginnen (3xx), handelt es sich um eine Umleitung, insofern, dass noch weitere Schritte Seitens des Clients, für eine erfolgreiche Übertragung, notwendig sind.

Beginnt ein Statuscode mit einer vier (4xx) handelt es sich um einen Client-Fehler. Das heißt, das der Grund des Scheiterns der Anfrage in erster Linie auf Seiten des Clients liegt. Beispielhaft ist der Statuscode 400, der besagt, dass die Anfrage fehlerhaft aufgebaut war.

Liegt die Ursache des Scheiterns eher im Verantwortungsbereich des Servers, erscheint ein Statuscode mit vorangestellter fünf (5xx). Beispielhaft für einen solchen Fehler ist der Statuscode 500, welcher auf einen internen Serverfehler hinweist. Dies bedeutet meistens, dass der Server auf dem sich die angefragte Ressource befindet, nicht erreichbar ist.

Tabelle 1 zeigt eine Übersicht häufig vorkommender Statuscodes:

200	OK
400	Bad Request
404	Not Found
500	Internal Server Error

Tabelle 1: häufige Statuscodes

### 2.2.4 Verbindungsabbau

Auch in HTTP/1 beendet der Server die Verbindung zum Client nach Übertragung der vom Client angeforderten Ressource automatisch. Somit ist es nicht möglich, über eine TCP-Verbindung mehrere Transaktionen durchzuführen.

## 2.3 HTTP/1.1

Im Juni 1999 erschien der RFC 2616, welcher die bis dato neueste Version des Hypertext Transfer Protocols, HTTP/1.1, beschreibt.

Ein Problem was seit der Einführung von des Hypertext Transfer Protocols besteht ist die Tatsache, dass nach jeder Übertragung einer Ressource die TCP-Verbindung beendet wird und für das Übertragen einer weiteren Ressource eine neue TCP-Verbindung zwischen Client und Server hergestellt werden muss. Besonders hinderlich ist dabei die Phase des langsamen Starts, welche in der TCP-Überlastungskontrolle<sup>5</sup> implementiert ist.

Mit HTTP/1.1 führte man persistente TCP-Verbindungen ein, welche es dem Client ermöglichen, auch nach dem Erhalt der angeforderten Ressource, weitere Anfragen an den Server über die selbe TCP-Verbindung zu senden.

Des Weiteren sind zu den bestehenden Methoden GET, HEAD, POST, PUT, DELETE, LINK und UNLINK, die Methoden OPTIONS und TRACE hinzugekommen. Bei einer Anfrage von Client an den Server ist nun das Headerfeld Host zu einem Pflichtfeld geworden. Dies ist darin begründet, dass HTTP/1.1 virtuelle Hosts unterstützt. Demnach können einer IP-Adresse mehrere Hosts zugewiesen sein. Um sicherzustellen, dass der gewünschte Ziel-Host erreicht wird, wird dieser im Headerfeld Host explizit angegeben. Sofern bei HTTP/1.1 in der GET-Anfrage kein

---

<sup>5</sup>Die Phase des langsamen Starts in der TCP-Überlastungskontrolle, stellt durch Schrittweise Erhöhung der gesendeten Ressourcen sicher, dass das Netzwerk nicht überlastet wird.[Bau14]



Headerfeld „Host“ vorhanden ist, wird die Anfrage vom Server nicht verarbeitet. Dieser antwortet mit dem Statuscode 400 „Bad Request“.

Eine weitere Neuerung von HTTP/1.1 gegenüber seinen Vorgängerversionen ist Pipelining. Mittels Pipelining ist es möglich, über eine TCP-Verbindung mehrere Anfragen parallel an einen Server zu senden. Jedoch ist das Pipelining in HTTP/1.1 standardmäßig deaktiviert, da es bei der Implementierung zu erheblichen Problemen führt und Head of Line Blocking (siehe 2.4.2) verursacht.

### 2.3.1 Verbindungsaufbau

Der Verbindungsaufbau hat sich im Wesentlichen nicht geändert hinsichtlich HTTP/0.9 (siehe Verbindungsaufbau HTTP/0.9).

### 2.3.2 Anfrage

Das Headerfeld "Host" wird als erforderliches Pflichtfeld eingeführt. Falls dieses nicht vorhanden ist, sendet der Server als Antwort den Statuscode 400 mit der Zusatzinformation „Bad Request“.

---

```
1 GET /index.html HTTP/1.1
2 Accept-Charset: utf-8
3 Accept-Language: de-DE
4 Host: example.org
```

---

Listing 5: Anfrage HTTP/1.1

### 2.3.3 Verbindungsabbau

Bei HTTP/1.1 wird die Verbindung zwischen Client und Server nach erfolgreicher Datenübertragung nicht automatisch getrennt. Die Verbindung bleibt erhalten, es können somit über eine existierende TCP-Verbindung weitere Datenpakete übertragen werden. Der Sinn besteht zum einen darin, dass Zeit eingespart wird, die Client und Server

zum Aufbauen einer TCP-Verbindung benötigen. Zum anderen ist es naheliegend, dass ein Client weitere Ressourcen einer Webseite anfordert, zum Beispiel Bilder oder Videos. Diese können dann direkt beim Server angefragt werden, ohne zuvor eine erneute TCP-Verbindung aufbauen zu müssen, dadurch wird eine Round Trip Time<sup>6</sup> eingespart.

Der Verbindungsabbau geschieht letztendlich, nachdem eine gewisse Zeit ohne jegliche Anfrage an den Server verstrichen ist, diese Zeit kann serverseitig konfiguriert werden.

---

<sup>6</sup>Die Round Trip Time, zu deutsch Paketumlaufzeit, gibt die Zeit an, die ein Datenpaket in einem Rechnernetz benötigt, um von Quelle zu Ziel und wieder zurück zu reisen.[RTT]

## 2.4 HTTP/2

Mit HTTP/2 wird nach über 15 Jahren eine neue Version des Hypertext Transfer Protocols eingeführt. In der Zeitspanne von 1999 bis 2015 war HTTP/1.X das Protokoll, welches zum Abrufen von Webseiten verwendet wurde. Ende 2014 hat eine Webseite im Durchschnitt 1.953 Kilobyte und ist mit 50 Bildern versehen.[t3n]

Fast 50 Prozent der Webseitenbesucher erwarten, dass eine Webseite in zwei Sekunden oder weniger geladen ist und tendieren dazu die Webseite zu verlassen, wenn diese nicht innerhalb von drei Sekunden geladen ist. Gerade bei Verkaufsplattformen im Internet ist die Ladezeit ihrer Webseite, durch viele enthaltene Bilder, eingebettete Programme und Weiterleitungen zu externen Webseiten ein großes Problem. Hierbei ergab sich, dass 79 Prozent der Kunden einer online Verkaufsplattform, welche Probleme mit dem Laden der Webseiten hatten, nicht mehr auf dieser Webseite einkaufen würden.[Spe]

Die Netzwerkinfrastruktur hat sich seit der Einführung von HTTP/1.1 stark verbessert. Mittlerweile ist so gut wie überall eine schnelle Internetanbindung verfügbar. Nur das zugrunde liegende Protokoll hat sich nicht weiterentwickelt.

Um das Problem versinnbildlicht darzustellen:

*„Die Straßen werden immer besser und breiter, Städte sind direkt aneinander angebunden, nur das Heu wird noch immer mit klapprigen Kutschen, mit abgefahrenen Holzrädern, transportiert.“*

Hierbei sollen die immer besser und breiter werdenden Straßen und die direkt aneinander angebunden Städte die Netzwerkinfrastruktur versinnbildlichen. Das Heu stellt die angefragten Informationen dar. Die Kutsche mit den abgefahrenen Rädern ist letztendlich das Protokoll HTTP/1.X, mit dem die angefragten Informationen geliefert werden.

Um diesem Umstand entgegenzuwirken, wurde HTTP/2 von der IETF<sup>7</sup> entwickelt. Als Vorlage diente das Anwendungsprotokoll SPDY, welches von Google entwickelt wurde. HTTP/2 soll hierbei nicht die alten Protokollversionen ersetzen, sondern diese als Alternative ergänzen.

Die Ziele bei der Entwicklung von HTTP/2 waren die effizientere Nutzung der Netzwerk-Ressourcen, besonders von TCP-Verbindungen, die Reduzierung der Latenz, um das Laden von Webseiten zu beschleunigen. Die Kompression der Header um den Overhead zu reduzieren, außerdem Multiplex- und Push-Techniken. An der Semantik von HTTP sollte jedoch nichts verändert werden, sodass alle Methoden, Statuscodes und Headerfelder erhalten bleiben, um somit die Abwärtskompatibilität zu HTTP/1.X zu gewährleisten.[htt]

### 2.4.1 SPDY und HTTP/2

SPDY ist ein von Google entwickeltes experimentelles Netzwerkprotokoll, welches man sich bei der Entwicklung von HTTP/2, zum Vorbild genommen hat. Neue Funktionen von HTTP/2, wie das Multiplexen<sup>8</sup> von Übertragungen, die Priorisierung von Streams, die Komprimierung der Header und die Server-Push Funktion, stammen in ihrer Grundidee von SPDY. Mit SPDY ist eine Verschlüsselung mittels TLS<sup>9</sup> erforderlich. Bei HTTP/2 ist dies nicht mehr der Fall, auch wenn es von Seiten des Webbrowsers erforderlich ist.

---

<sup>7</sup>InternetEngineeringTaskForce

<sup>8</sup>Zusammenmischen verschiedener Nachrichtenrahmen innerhalb einer TCP-Verbindung

<sup>9</sup>Verschlüsselungsprotokoll zur sicheren Übertragung von Daten im Internet

Im folgenden Abschnitt wird auf die wichtigsten Neuerungen von HTTP/2 [gemäß RFC 7540 [RFC15]] hinsichtlich des Laufzeitverhaltens und Datendurchsatzes genauer eingegangen. Des Weiteren werden Vergleiche zu HTTP/1.1 aufgestellt.

#### 2.4.2 Gegenüberstellung von HTTP/1.1 und HTTP/2 hinsichtlich der Datenübertragung

Wenn ein Client mit HTTP/1.X mehrere Anfragen gleichzeitig, beziehungsweise parallel an den Server senden möchte, ist pro Anfrage eine TCP-Verbindung erforderlich. HTTP/1.X verfügt zwar über Pipelining, also die Funktion mehrere Anfragen oder Antworten parallel zu senden, jedoch ergeben sich schwerwiegende Nachteile, wie das „Head of Line Blocking“. Head of Line Blocking tritt dann ein, wenn HTTP-Anfragen mit aktiviertem Pipelining gesendet werden. Die Anfragen werden vom Server nach dem Prinzip First in First Out (FIFO)<sup>10</sup> abgearbeitet.

Ein Client sendet zwei Anfragen parallel an einen Server (siehe Abbildung 2). Die Anfrage „GET /index.html“ wird als erstes gesendet, dementsprechend nach „FiFo“ auch zuerst abgearbeitet. Als zweites sendet der Client die Anfrage „GET /style.css.“ Der Server ist nun dabei, beide Anfragen zu verarbeiten. Die zweite Anfrage, „GET /style.css“ ist bereits nach 20 Millisekunden fertig abgearbeitet, kann jedoch noch nicht gesendet werden, da der Server noch bei der Abarbeitung der ersten Anfrage des Clients ist. Nach 80 Millisekunden ist schließlich auch die erste Anfrage abgearbeitet. Erst jetzt werden beide Antworten an den Client gesendet. Somit muss der Client 60 Millisekunden länger auf die Antwort der Anfrage „GET /style.css“ warten. Das Head of Line Blocking besteht darin, dass die erste Anfrage (Head of Line), die übrige Abarbeitung der restlichen Anfragen (Anfrage 2) blockiert.

---

<sup>10</sup>First in First Out (FIFO) bezeichnet jegliche Verfahren der Speicherung, bei denen diejenigen Elemente, die zuerst gespeichert wurden, auch zuerst wieder aus dem Speicher entnommen werden

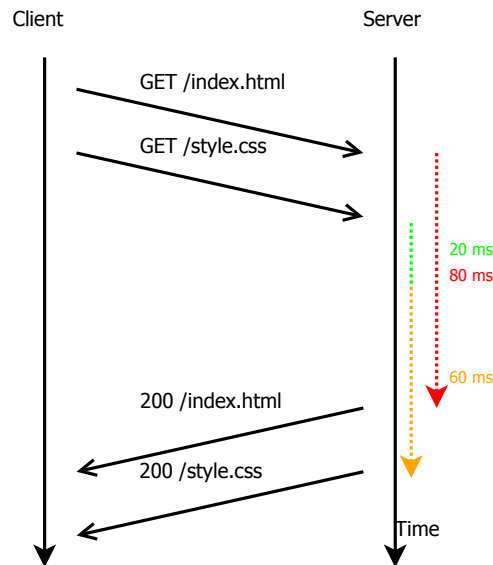


Abbildung 2: Head of Line Blocking

Aus diesem Grund ist das Pipelining für HTTP/1.1, bei Browsern standardmäßig deaktiviert. Jedoch gibt es einen Workaround<sup>11</sup> für dieses Problem, welcher eine performantere Datenübertragung ermöglicht. Clients haben die Möglichkeit, ihre HTTP-Anfragen an den Server auf mehrere TCP Verbindungen zu verteilen. Bis zu sechs TCP-Verbindungen können zwischen Client und Server gleichzeitig bestehen. Es sind maximal sechs Verbindungen zu einem Server möglich um die Serverauslastung nicht zu groß ausfallen zu lassen. Bei HTTP/2 wird nur eine Verbindung zwischen Client und Server benötigt.

Abbildung 3 zeigt das Abrufen einer Webseite mit HTTP/1.1 beziehungsweise mit HTTP/2. In diesem Ausschnitt werden 11 Ressourcen vom Server abgerufen. Auf der linken Seite der Abbildungen kann man erkennen, dass bei dem Abrufen der Webseite mittels HTTP/1.1, die ersten sechs

<sup>11</sup>Ein Hilfsverfahren was das Problem nicht behebt, aber umgeht

Anfragen parallel laufen, dargestellt mit einem Balken in grün beziehungsweise lila. Die folgenden fünf Anfragen haben einen grauen Balken vorangestellt. Dieser graue Balken symbolisiert, dass die Verbindung zu der jeweiligen Ressource zurzeit nicht hergestellt werden kann, die Verbindung also blockiert ist, da die volle Bandbreite an parallel laufenden Verbindungen ausgeschöpft ist. Auf der rechten Seite der Abbildung sieht man die Übertragung mittels HTTP/2. Alle 11 Ressourcen werden parallel über eine einzige TCP-Verbindung geladen oder befinden sich im Zustand wartend.[mul]



Abbildung 3: parallele Verbindungen

HTTP/2 ist ein binäres Protokoll und nicht wie HTTP/1.X textbasiert. Der Vorteil eines binären Protokolls ist, dass sich binäre Daten effizienter analysieren lassen. Außerdem sind sie kompakter und weniger fehleranfällig als textbasierte Protokolle wie HTTP/1.X. An der Semantik von HTTP/2 hat sich nichts geändert, alle Methoden, Statuscodes und Headerfelder sind unberührt geblieben. Allerdings ist die Art der Kodierung während der Übertragung eine andere. HTTP/2-Nachrichten werden vor dem Senden in Frames unterteilt und nicht wie HTTP/1.X-Nachrichten in Klartext übertragen. Dabei ist der größte Unterschied zwischen HTTP/1.X und HTTP/2, dass in HTTP/2 während einer Sendung von Daten, zwischen Client und Server, durch das binär-Format von HTTP/2-Nachrichten, viele Frames, verschiedener Nachrichten, gemischt gesendet werden können. In HTTP/1.X hingegen können während einer Sendung ausschließlich eine Nachricht, Anfrage oder Antwort gesendet werden. Dies ist dadurch bedingt, dass HTTP/1.X-Nachrichten in Klartext, wel-

cher sich nicht aufteilen lässt, übertragen werden (siehe Abbildung 4).

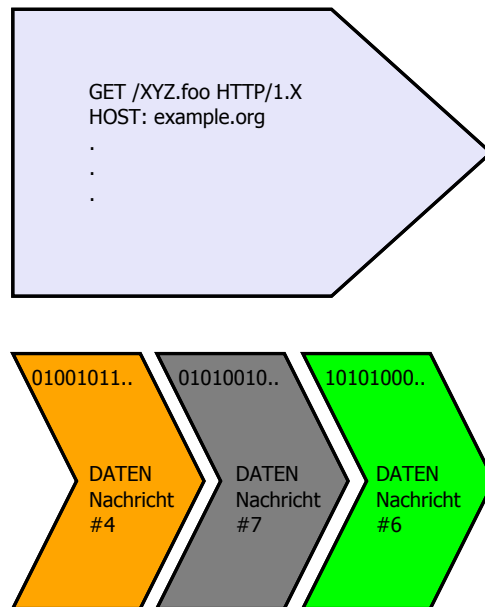


Abbildung 4: Text gegen Frame

Es sind 10 verschiedene Frametypen in RFC 7540 beschrieben, welche für die Übertragung von Daten mittels HTTP/2 zur Verfügung stehen:[fra]

## CONTINUATION

Der CONTINUATION-Frame wird verwendet, wenn die Anzahl an Headerinformationen zu groß ist, um in einen einzelnen HEADER-Frame zu passen.



## DATA

Der DATA-Frame enthält die eigentlichen Daten einer HTTP/2-Anfrage oder -Antwort.

## GOAWAY

Wenn ein Endpunkt, Client oder Server die Verbindung schließen möchte, wird ein GOAWAY-Frame gesendet.

## HEADERS

In diesem Frame werden sämtliche Information aus den optionalen Headerfeldern übertragen.

## PING

Bei dem Senden eines PING-Frames fordert Partner A Partner B dazu auf, mit einem PING-Frame zu antworten.

## PRIORITY

Dieser Frame wird verwendet, um die Priorität eines Streams festzulegen.

## PUSH-PROMISE

Dieser Frame wird vom Server gesendet, wenn er die Push-Funktion verwendet.

## RST-STREAM

Ein Stream wird von einem Partner zwangsweise beendet. Dieser Frame wird nicht für das normale Terminieren eines Frames verwendet.

## SETTINGS

Mit dem SETTINGS- Frame werden die HTTP /2-Verbindungseinstellungen übermittelt.

## WINDOW-UPDATE

Aktualisiert das Flow-Control-Fenster.

Frames haben einen festen Aufbau. Alle Frames beginnen mit einem festen, neun Byte großen Header (siehe Abbildung 5).

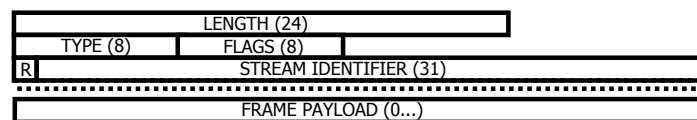


Abbildung 5: Aufbau eines Frames

Im Folgenden werden die einzelnen Felder eines Frames genauer erläutert:

## LENGTH

Das 24-Bit große Feld LENGTH gibt die Länge der zu übertragenen Nutzdaten, exklusive den 9-Bit großen FRAMEHEADER, an.

## TYPE

In dem 8-Bit großen Feld TYPE wird der Typ des Frames definiert.

## FLAGS

Das 8-Bit große Feld FLAGS ist reserviert für Frame-Typ spezifische Flags.

## R

Das 1-Bit große Feld R ist undefiniert. Es muss stets auf (0x0) stehen, sowohl beim Senden als auch beim Empfangen eines Frames.

## STREAM IDENTIFIER

In dem Feld STREAM IDENTIFIER wird angegeben zu welchem Stream ein Frame gehört.

## FRAME PAYLOAD

Die Struktur des Felds FRAME PAYLOAD ist abhängig vom Frame-Typ.

Frames werden über einen Stream, welcher bidirektional ist, von Partner A zu Partner B gesendet. Es ist nicht erforderlich, dass alle Frames innerhalb eines Streams zur selben Nachricht (message) gehören. Durch die Stream-ID werden die Frames nach der Übertragung der richtigen Nachricht (Anfrage oder Antwort) zugeordnet und schließlich wieder zusammengeführt.

Abbildung 6 zeigt das Multiplexing-Verfahren<sup>12</sup>. Der Client sendet eine Anfrage (3) an den Server. Währenddessen antwortet der Server mit zuvor angeforderten Ressourcen. In diesem Fall erhält der Client als Antwort zunächst einen Header-Frame, mit der Stream ID 2, gefolgt von einem Daten-Frame, mit der Stream ID 7 und einem weiteren Header-Frame, mit der Stream ID 4. Schließlich erhält der Client die zu dem Header-Frame, mit der Stream ID 2, gehörenden Nutzdaten in dem Daten-Frame mit der Stream ID 2.

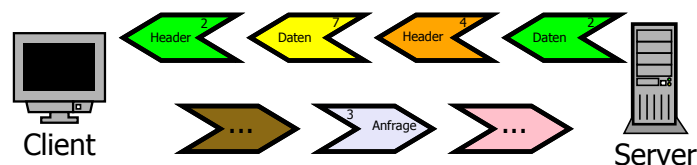


Abbildung 6: Multiplexing

Das Multiplexing-Verfahren ermöglicht das Senden von fertig abgearbeiteten Teilen (Frames) einer Nachricht. Somit ist es nicht mehr erforderlich, wie in HTTP/1.X zu warten bis Anfragen komplett abgearbeitet wurden. Dieser Fakt eliminiert in HTTP/2 das Head of Line Blocking-Problem.[Pru15]

### 2.4.3 Stream Priorisierung

Es ist möglich, bei HTTP/2-Anfragen eine Priorität für einen Stream festzulegen. Standardmäßig arbeitet ein Server die Anfragen eines Clients asynchron, ohne festgelegte Reihenfolge, parallel ab. Durch den Frame

<sup>12</sup>Frames von unterschiedlichen Nachrichten sind in einem Stream zusammen gemischt

PRIORITY kann einem Stream eine Priorität zugewiesen werden, sodass dieser Stream in der Abarbeitung auf Seiten des Servers bevorzugt wird. Dadurch kann dem Client die angefragte Ressource zum frühestmöglichen Zeitpunkt bereitgestellt werden.

Die Priorität eines Streams kann dynamisch während der Laufzeit festgelegt werden. So ist es für einen Webbrowser machbar, Inhalte zu priorisieren. Wenn ein Anwender eine Webseite mit vielen Bildern herunter scrollt, können die wichtigsten Bilder priorisiert werden.

Ebenso ist es möglich, dem Text einer Webseite eine höhere Priorität zu geben als Bildern. Somit kann der Anwender, welcher die Webseite aufruft, die Texte auf der Webseite lesen, auch wenn das vollständige Laden der Webseite noch nicht abgeschlossen ist.[pri]

### 2.4.4 Server Push

Das Server Push-Verfahren erlaubt es einem Server, dem Client mit mehr als nur einer Antwort auf eine Anfrage zu antworten. Der Server ist nun in der Lage, dem Client, mit dem Frame-PUSH-PROMISE, Daten zu übermitteln, ohne zuvor explizit eine Anfrage erhalten zu haben. Wenn ein Client eine HTTP/2-Anfrage an den Server sendet, dieser die Anfrage beantwortet und weiß, dass der Client eine weitere Ressource, die in diesem Kontext steht, benötigt, kann der Server diese an den Client senden. Auf die Performance wirkt sich Server Push positiv aus, da eine oder sogar mehrere Anfragen von Seiten des Clients eingespart werden können.

Abbildung 7 verdeutlicht das Server Push-Verfahren. Der Client hat vom Server eine Ressource, `index.html`, angefordert. Die Antwort wird mit dem Stream, mit der Stream-ID 2 gesendet. Der Server hat bemerkt, dass zu der angeforderten Ressource `index.html` eine weitere Ressource in Zusammenhang steht. Der Server sendet dem Client also die Ressource `style.css` (Stream-ID 9) via Server Push.`[ser]`

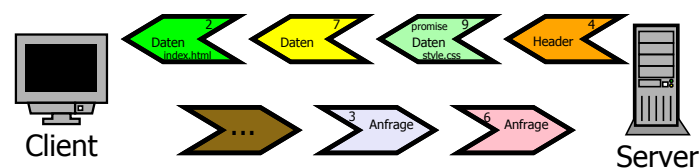


Abbildung 7: Server Push

### 2.4.5 HTTP Header Komprimierung mittels HPACK

Wenn ein Client eine Anfrage für eine Ressource, zum Beispiel einer JPEG-Datei, an einen Server sendet und im Anschluss weitere JPEG-Dateien des selben Hosts anfragt, werden viele Informationen in den Headerfeldern wiederholt gesendet. Diese wiederkehrenden Informationen im Header einer HTTP-Anfrage verursachen einen großen Overhead. Der Overhead, bestehend aus repetitiven Headerinformationen, macht die Datenübertragung ineffizient, durch ihn werden HTTP/1.1 Anfragen noch größer. Dies kann darin enden, dass die an den Server gestellte Anfrage größer als das initiierte TCP-Fenster wird, sodass die Anfrage aufgeteilt werden muss. Dadurch wird die Übertragung sehr langsam, da der Client auf eine Bestätigung (ACK) des Servers warten muss, bevor er den Rest der Anfrage senden kann.

Mit HTTP/2 wird die Header Komprimierung mittels HPACK eingeführt. Client und Server pflegen zugleich eine Kopie einer dynamischen Header Tabelle. Sendet Partner A einen Header an Partner B, kann Partner A diesen auffordern, den empfangenen Header in seiner Header-Tabelle zu speichern. Dieser Header wird anschließend mit einem Index versehen. Sendet Partner A erneut eine Nachricht mit dem selben Header an Partner B, muss Partner A, statt den gesamten Header, nur noch den Index des jeweiligen Headers mitsenden. Beide Partner müssen die Header-Tabelle stets aktuell halten.

HPACK bietet außerdem eine statische Header-Tabelle, in der die am häufigst vorkommenden Header gespeichert sind. HTTP/2 ist kein zustandsloses Protokoll wie seine Vorgänger, da Headerinformationen sowohl auf dem Client als auch auf dem Server gespeichert werden. Bei HTTP/2 sind diese Informationen nicht mehr ausschließlich in der übertragenen Nachricht enthalten und verlieren nicht mit Ende der Verbindung ihre Gültigkeit.[HPA]

Für die Komprimierung der Headerinformationen wird die Huffman-Kodierung verwendet. Dabei werden die am häufigsten vorkommenden Zeichen beziehungsweise Buchstaben mit möglichst wenigen Bits kodiert. Damit ist es möglich, Wörter statt mit der üblichen ASCII-

Kodierung mit acht Bits pro Buchstabe, mit weniger Bits pro Buchstabe zu kodieren. Dabei werden bei der Übertragung der Informationen Bits eingespart, die Informationen sind also komprimiert.[huf]

Mit dem Programm HTTPWatch<sup>13</sup> wurde der Aufruf der Webseite <https://www.google.de>, sowohl mit HTTP/1.1, welches keine Headerkomprimierung verwendet, als auch mit HTTP/2 aufgezeichnet. Der obere Teil von Abbildung 8 zeigt den Aufruf der Webseite via HTTP/1.1. Mit HTTP/1.1 wurden insgesamt 6690 Bytes gesendet und 372188 Bytes empfangen. Mit HTTP/2 hingegen wurden insgesamt 2056 Bytes gesendet und 357464 Bytes empfangen. In der Summe wurden mit HTTP/2, 4634 Bytes weniger gesendet und 14724 Bytes weniger empfangen als mit HTTP/1.1. Dieser Umstand ist auf die Headerkomprimierung zurückzuführen. Besonders gut zu erkennen ist dies an der letzten GET-Methode (sowohl im oberen, als auch im unteren Teil der Abbildung), welche den Statuscode 204 (No Content) sendet. In dieser Antwort des Servers ist kein Inhalt enthalten, dementsprechend sind die gesendeten und empfangenen Bytes ausschließlich Headerinformationen. In der Spalte „Sent“ wird die Größe des Anfrage-Headers angezeigt, in der Spalte „Received“ wird die Größe des Antwort-Headers angezeigt. Hier sind die Header der HTTP/2-Anfrage wesentlich kleiner als die der HTTP/1.1-Anfrage.

---

<sup>13</sup><http://www.httpwatch.com/>

























Sent	Received	Method	Result	HTTP/2	Type	URL
341	53461	GET	200			https://www.google.de/?gws_rd=ssl
581	21151	GET	200			https://www.google.de/images/nav_...
582	14408	GET	200			https://www.google.de/images/srpr/l...
359	10235	GET	200			https://ssl.gstatic.com/gb/images/i1_...
685	126734	GET	200			https://www.google.de/xjs/_js/k=xj...
575	11798	GET	200			https://www.google.de/extern_chro...
359	781	GET	200			https://www.google.com/textinputas...
910	33924	GET	200			https://www.google.de/xjs/_js/k=xj...
428	49136	GET	200			https://www.gstatic.com/og/_js/k=...
476	50288	GET	200			https://apis.google.com/_scs/abc-st...
1394	272	GET	204			https://www.google.de/gen_204?v=...
6690	372188	11 requests				
Sent	Received	Method	Result	HTTP/2	Type	URL
43	53472	GET	200	Yes		https://www.google.de/?gws_rd=ssl
235	20827	GET	200	Yes		https://www.google.de/images/nav_...
62	14052	GET	200	Yes		https://www.google.de/images/srpr/l...
53	9862	GET	200	Yes		https://ssl.gstatic.com/gb/images/i1_...
174	126622	GET	200	Yes		https://www.google.de/xjs/_js/k=xj...
53	422	GET	200	Yes		https://www.google.com/textinputas...
135	27	GET	204	Yes		https://www.google.de/gen_204?at...
344	33515	GET	200	Yes		https://www.google.de/xjs/_js/k=xj...
123	48727	GET	200	Yes		https://www.gstatic.com/og/_js/k=...
164	49888	GET	200	Yes		https://apis.google.com/_scs/abc-st...
670	50	GET	204	Yes		https://www.google.de/gen_204?v=...
2056	357464	11 requests				

Abbildung 8: Header Vergleich

### 2.4.6 Wechsel zu HTTP/2

Die Voraussetzung für eine Übertragung von Ressourcen mit HTTP/2 ist, dass sowohl Client, als auch Server HTTP/2 unterstützen. Der Client beziehungsweise Browser sendet standardmäßig eine HTTP/1.1 Anfrage an den Webserver. Die Anfrage enthält ein Headerfeld „Upgrade“ mit dem Attribut „HTTP/2.0“. Sofern der Server HTTP/2 unterstützt, antwortet er mit „HTTP/1.1 101 Switching Protocols“, er wechselt also zu HTTP/2. Wenn der Server keine HTTP/2 Unterstützung bietet, antwortet er einfach mit HTTP/1.1. Clients merken sich in der Regel, welche Server HTTP/2 unterstützen. Sodass sie bei der nächsten Verbindung zu dem Server direkt Anfragen mittels HTTP/2 stellen.[Pru]

### 2.4.7 HTTP/2 Browser-/Server-Unterstützung

Grundsätzlich ist es so, dass für eine erfolgreiche Datenübertragung von Daten mittels HTTP/2, beide Seiten, also Client und Server, HTTP/2 fähig sein müssen. Gängige Webbrowser wie Google Chrome, Mozilla Firefox, Internet Explorer etcetera, sind in neueren Versionen HTTP/2-fähig, jedoch ausschließlich in Verbindung mit SSL/TLS. Man hat auch die Möglichkeit, falls HTTP/2 nicht verwendet werden soll, die Verwendung in den Browsereinstellungen zu deaktivieren. Die Bandbreite an Webservern, die eine HTTP/2 Unterstützung im Standard gewähren, ist aktuell (Stand: August 2015) noch nicht sehr groß. Webserver wie der Apache HTTP Server, unterstützen HTTP/2, jedoch ist dies nicht standardmäßig der Fall. Beim HTTP-Server Apache ist für HTTP/2 die Version 2.4.12 oder höher erforderlich. Ab dieser Version hat man die Möglichkeit, die Modifikation „mod\_h2“ zu installieren, welche die Unterstützung von HTTP/2 aktiviert. Der von Microsoft stammende IIS<sup>14</sup> unterstützt HTTP/2 ab Windows 10, beziehungsweise Windows Server 2016 im Standard. NGINX<sup>15</sup> bietet einen experimentellen Support für HTTP/2.[Sup]

---

<sup>14</sup>Microsoft Internet Information Services

<sup>15</sup>Webserver-Software

## 3 Aufbau der Testumgebung

Im vorherigen Teil dieser Arbeit wurden die einzelnen Protokollversionen des Hypertext Transfer Protocols beschrieben und deren Funktionsweisen erläutert. In dem folgenden, praktischen Teil wird zunächst der Aufbau der Testumgebung und der verwendeten Programme beschrieben. Es werden auch während der Durchführung aufgetretene Probleme und Unklarheiten erläutert. Des Weiteren werden Testszenarien vorgestellt und anschließend Leistungstests durchgeführt, in denen die Protokollversionen HTTP/1.1 und HTTP/2 miteinander verglichen werden.

### 3.1 Anforderungen an die Testumgebung

Es wird ein Host-System für den Webserver benötigt, bei welchem diverse Ressourcen via HTTP/1.1 und HTTP/2 angefragt werden. Dieser Webserver muss eine Unterstützung für HTTP/2 bieten. Ein SSL-Zertifikat wird benötigt, da moderne Webbrowser HTTP/2 nur gesichert, also mit SSL/TLS, unterstützen. Der Webserver muss über das Internet erreichbar sein. Ein Szenario, in dem sich Client und Server im selben Netz befinden, kommt nicht in Frage, da versucht werden soll das Abrufen einer Webseite so realistisch wie möglich darzustellen.

Es wird ein Client-System mit einem Webbrowser und diversen Programmen, zum Messen der Geschwindigkeit benötigt. Sowohl der Webserver, als auch das Client-System müssen über ein Ethernet Kabel, an einen Switch beziehungsweise Router angebunden sein, um Verbindungs- und Latenzschwankungen so gering wie möglich zu halten.

### 3.2 Host-System für Webserver

Als Host-System für den Webserver dient ein Debian 8.0 ohne grafische Benutzeroberfläche. Das Linux-System ist eine virtuelle Maschine, virtualisiert unter Windows Hyper-V<sup>16</sup> auf einem Windows Server 2012

---

<sup>16</sup>Hyper-V ist eine Hypervisor-basierte Virtualisierungstechnik von Microsoft für Computer mit x64-fähigem x86-Prozessor

R2. Der Windows Server befindet sich in einem deutschen Rechenzentrum des Webhosting Anbieters Hetzner. Der virtuellen Linux-Maschine stehen permanent 2048 Megabyte Arbeitsspeicher und vier Rechenkerne zur Verfügung.

Der Webserver, der für die Leistungsuntersuchung von HTTP/1.1 und HTTP/2 verwendet werden soll, ist ein Apache HTTP-Server in der Version 2.4.X.

Problem bei Inbetriebnahme des Webservers:

Der mit dem Betriebssystem (Debian 8.0) ausgelieferte Apache HTTP-Server der Version 2.4.10 unterstützt noch nicht standardmäßig HTTP/2. Durch Recherche im Internet und diversen Linux-Foren wurde offenbar, dass es ein Apache-Modul gibt, welches die HTTP/2 Unterstützung bereitstellt. Dieses Modul des Webservers, namens „mod\_h2“ setzt allerdings einen Apache HTTP Server ab Version 2.4.12 voraus.

Somit wurde der Apache HTTP-Server auf die Version 2.4.16 aktualisiert. Im Anschluss wurde die Erweiterung, „mod\_h2“, des Apache HTTP Servers installiert. Während des Installationsvorganges kam es beim Kompilieren zu schwerwiegenden Fehlern, welche selbst nach längerer Recherche nicht behoben werden konnten.

Lösung:

Durch weitere Nachforschungen im Internet bezüglich eines HTTP/2 fähigen Webservers ergab sich, dass auch der weniger verbreitete Caddy Webserver<sup>17</sup> als HTTP-Server für die Vergleichsmessungen anbot. Der Caddy Webserver dient als Lösung für das oben beschriebene Problem. Dieser Webserver unterstützt im Standard HTTP/2. Der Caddy Server wird heruntergeladen, im Anschluss, zum Beispiel mit dem Befehl „gzip“ entpackt. Eine Installation ist nicht notwendig. Die entpackte Datei „cad-

---

<sup>17</sup><http://caddyserver.com>

dy\_linux\_amd64“, wird in einen Ordner gelegt, welcher die Dateien enthält, die als Webseite zur Verfügung gestellt werden sollen. Durch das Ausführen der Datei „caddy\_linux\_amd64“, wird der Inhalt des Ordners, zum Beispiel eine Datei „index.html“, im lokalen Webbrowser, unter <http://localhost:2015>, dargestellt.

Die gesamte Konfiguration des Caddy Webservers wird durch das sogenannte Caddyfile bewerkstelligt.

Der Webserver wurde wie folgt konfiguriert:

---

```
1 0.0.0.0:443
2 tls ../keys/caddy.pem ../keys/caddy.key
3 gzip
```

---

Listing 6: Caddyfile

In der ersten Zeile ist definiert, dass der Caddy Webserver auf jegliche Anfragen hören soll, welche über Port 443<sup>18</sup> kommen.

In der nächsten Zeile ist der Pfad angegeben, in dem das für SSL/TLS notwendige Zertifikat hinterlegt ist. Dieses Zertifikat wurde per OpenSSL<sup>19</sup> erstellt. Eine gesicherte Verbindung per SSL/TLS ist notwendig, da gängige Webbrowser wie Mozilla Firefox, der verwendete Webbrowser Iceweasel, Google Chrome, Internet Explorer etcetera, die Verwendung von HTTP/2 ausschließlich über HTTPS, also einer verschlüsselten Verbindung, zulassen. In der letzten Zeile des Caddyfiles ist definiert, dass GZIP ein freies Kompressionsprogramm zum komprimieren der Ressourcen verwendet werden soll. Der Transfer der Testdaten auf den Webserver wurde mit Filezilla unter Zuhilfenahme des Protokolls SFTP<sup>20</sup> durchgeführt.

---

<sup>18</sup>HTTPS (Hypertext Transfer Protocol in Verbindung mit SSL/TLS

<sup>19</sup>Programm zum Erzeugen und Verwalten von Zertifikaten[Ope]

<sup>20</sup>[https://de.wikipedia.org/wiki/SSH File Transfer Protocol](https://de.wikipedia.org/wiki/SSH_File_Transfer_Protocol)

### 3.3 Client-System

Als zugrunde liegendes Betriebssystem für den Client ist ebenfalls Debian 8.0, jedoch mit grafischer Benutzeroberfläche, in Verwendung. Das Client-System ist keine virtuelle Maschine. Als Webbrowser kommt Iceweasel<sup>21</sup> in der Version 40 zum Einsatz. Das Client-System ist per Ethernetkabel an einen Router angebunden.

Abbildung 9 zeigt den Aufbau der verwendeten Netzwerkstruktur.

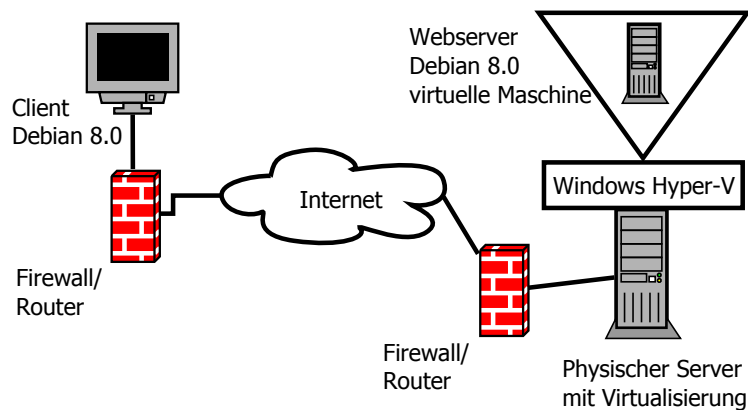


Abbildung 9: Testumgebung

### 3.4 Programme zum Messen des Datendurchsatzes

Folgende Programme zum Messen des Datendurchsatzes wurden in Betracht gezogen:

- Speedtracer (optionales Plugin Google Chrome)
- Firebug (optionales Plugin Iceweasel)
- integrierte Entwicklertools

<sup>21</sup>eine für Debian aus markenrechtlichen Gründen umbenannte Version des Firefox (<https://wiki.debianforum.de/Icweasel>)

- Wireshark

Die Anforderungen an das Tool sind eine gute grafische Übersicht des laufenden Datendurchsatzes sowie die Anzeige der Netto-Ladezeit (benötigte Zeit für das Laden der Ressourcen) und der Brutto-Ladezeit (vergangene Zeit zwischen Senden der Anfrage und dem vollständigen Laden der Webseite), der angefragten Webseite.

Tests der aufgelisteten Programme ergaben, dass das Firefox-/Iceweasel-Plugin Firebug den Anforderungen am besten genügt. Daher wurde im weiteren Verlauf der Arbeit dieses Werkzeug als Referenz für die Leistungsuntersuchungen verwendet.

## 4 Leistungsuntersuchung

Im folgenden Abschnitt dieser Arbeit wird eine Leistungsuntersuchung durchgeführt, in der die Protokolle HTTP/1.1 und HTTP/2 hinsichtlich ihres Laufzeitverhaltens und Datendurchsatzes miteinander verglichen werden.

### 4.1 Leistungsuntersuchung und Vergleich von HTTP/1.1 und HTTP/2

Um die Leistung der beiden Anwendungsprotokolle HTTP/1.1 und HTTP/2 untersuchen zu können, sind diverse Testszenarien von Nöten. Es soll gezielt getestet werden, inwieweit die Verbesserungen von HTTP/2, also das neue Binärformat, Multiplexing und die Headerkomprimierung einen Vorteil gegenüber HTTP/1.1 mit sich bringen. Außerdem stellt sich die Frage, wie das Maximum von sechs parallelen TCP-Verbindungen an einen Server HTTP/1.1 benachteiligt.

Dabei wurde von der Hypothese ausgegangen, dass HTTP/2 bei zahlenmäßiger Erhöhung der angefragten Ressourcen, aufgrund funktionierenden Pipelinings und Multiplexings im Vorteil ist. Aufgrund der Limitierung von HTTP/1.1 auf sechs parallele TCP-Verbindungen, sollte dieses mit wachsender Anzahl paralleler Verbindungen zunehmend im Nachteil sein.

### 4.2 Testszenarien

Als Basis der Testszenarien dient ein HTML-Dokument, welches mit Text versehen und mit diversen Bilddateien vom Typ JPEG verlinkt ist. Die Größen der Bilddateien reichen von 8 Kilobyte bis 35 Megabyte. Die Größe des Textes beläuft sich im Durchschnitt auf 4 bis 20 Kilobyte. Des Weiteren dienen öffentliche Webseiten als Grundlage der Leistungsuntersuchung.



Die Leistungsuntersuchung von HTTP/1.1 und HTTP/2 ist in vier Testreihen, zu je vier Leistungstests unterteilt. In Testreihe 1, werden ausschließlich kleine Bilddateien mit einer Größe zwischen 4,6 Kilobyte und 17,3 Kilobyte vom Server geladen. Hierbei wird die Anzahl der angeforderten Dateien von Test zu Test in etwa verdoppelt.

Testreihe 2 ist vom Aufbau her identisch zu Testreihe eins, jedoch handelt es sich bei den abzurufenden Bilddateien um JPEG-Dateien mittlerer Größe, also zwischen 30 Kilobyte und 650 Kilobyte.

In Testreihe 3 wird das Laufzeitverhalten von HTTP/1.1 und HTTP/2 hinsichtlich der Übertragung großer Bilddateien mit einer Größe zwischen 700 Kilobyte und 35 Megabyte getestet.

In Testreihe 4 wird die Leistung von HTTP/1.1 und HTTP/2 anhand öffentlicher Webseiten getestet, bei denen die angefragten Ressourcen auf mehr als nur einem Server gelagert sind.

Probleme zu Beginn der Leistungstests:

Zu Beginn der Leistungstests ergaben sich häufig auftretende Leistungsschwankungen, unabhängig von der Protokollversion. Dabei variierten die Ladezeiten der auf dem eigenen Webserver gelagerten Webseite um bis zu 20 Sekunden, teilweise endete das Laden der angeforderten Ressourcen in einem Timeout. Dieser Umstand machte es unmöglich, auswertbare Ergebnisse zu erzielen.

Lösung:

Die dem Webserver zur Verfügung stehenden Ressourcen, Arbeitsspeicher und CPU-Leistung waren nicht ausreichend. Außerdem war die virtuelle Maschine des Webservers so konfiguriert, dass sie dynamischen Arbeitsspeicher verwendete. Die Umstellung von 1024 Megabyte auf 2048 Megabyte nicht dynamischen Arbeitsspeicher und die Erhöhung der Re-

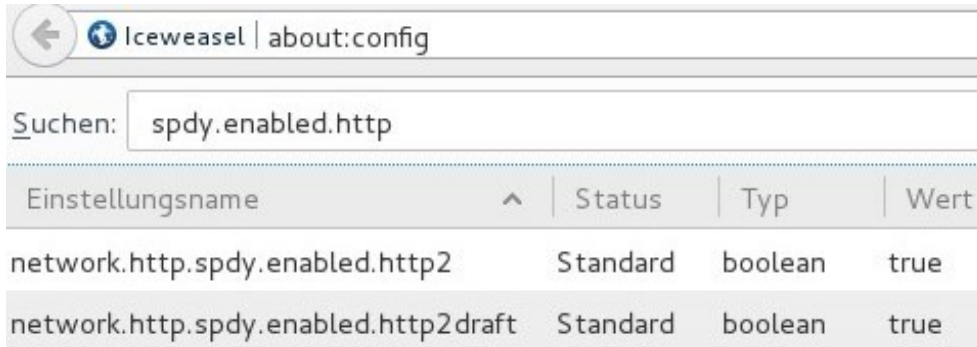
chenkerne von zwei auf vier beseitigte das Problem.

### 4.3 Vorwort zu den Leistungstests

Die folgenden Tests wurden ausschließlich in der im vorherigen Abschnitt beschriebenen Testumgebung durchgeführt. Die einzelnen Gegenüberstellungen von HTTP/1.1 und HTTP/2 haben in direkter zeitlicher Abfolge aufeinander stattgefunden, um den Einfluss tageszeitabhängiger Latenzschwankungen so gering wie möglich zu halten. In den Leistungstests werden zwei Zeiten aufgezeichnet, welche für diese Arbeit als Brutto-Ladezeit und Netto-Ladezeit definiert sein sollen. Die Brutto-Ladezeit, auch Onload genannt, gibt die Zeit an, die vergeht, bis die Webseite fertig geladen ist, also alle nötigen Dateien wie HTML-Dokumente, CSS-Dateien, Bilddateien und Javascript Code lokal verfügbar und geladen sind. Die Netto-Ladezeit gibt die reine Datenübertragungszeit an. Diese kann länger ausfallen als die Brutto-Ladezeit, falls nach dem vollständigen Laden einer Webseite, Ressourcen wie Videos oder Widgets von verknüpften Webseiten nachgeladen werden. Die Brutto-Ladezeit ist die Ladezeit, welche für den Anwender von Interesse ist, da nach deren Ablauf die Webseite insofern vollständig geladen ist, dass die wesentlichen Elemente dargestellt sind. Aus diesem Grund wird der Brutto-Ladezeit in dieser Arbeit eine höhere Gewichtung zugeschrieben als der Netto-Ladezeit.

Die Leistungstests wurden im Webbrowser Icedragon (Firefox) durchgeführt. Mit dem Browserplugin Firebug wurde die Dauer des Ladens der Webseiten gemessen. In jedem Leistungstest wurde eine Webseite 10 mal mittels HTTP/1.1 und 10 mal mittels HTTP/2 aufgerufen. Die Ladezeiten wurden aufgezeichnet und werden im Folgenden analysiert. Die Zeitangaben der Leistungstests sind in Sekunden angegeben. Alle Leistungstests wurden mit aktiviertem Kompressionsprogramm GZIP durchgeführt. HTTP/1.1 wird in den folgenden Säulendiagrammen in blauer Farbe dargestellt, HTTP/2 in grüner Farbe. Die ausführlichen Messergebnisse sind in tabellarischer Form im Anhang zu finden. Jeder der folgenden Leistungstests wurde in einer neuen Instanz des Icedragon Webbrowsers, mit leerem Browsercache durchgeführt.

Durch das Umschalten der folgenden Einstellungen (siehe Abbildung 10), im Webbrowser Iceweasel, auf den Wert „true“ beziehungsweise den Wert „false“, wurde zwischen den Protokollversionen HTTP/1.1 und HTTP/2 gewechselt.



The screenshot shows the Iceweasel browser's configuration page (about:config). A search bar contains the text 'spdy.enabled.http'. Below the search bar is a table with four columns: 'Einstellungsname', 'Status', 'Typ', and 'Wert'. Two rows are visible in the table, both with a value of 'true'.

Einstellungsname	Status	Typ	Wert
network.http.spdy.enabled.http2	Standard	boolean	true
network.http.spdy.enabled.http2draft	Standard	boolean	true

Abbildung 10: Umschalten zwischen Protokollversionen

## 4.4 Testreihe 1

In dieser Testreihe wird das Abrufen einer Webseite mit kleinen JPEG-Dateien, mit HTTP/1.1 und HTTP/2 verglichen. Diese Testreihe besteht aus vier Leistungstests. Die Leistungstests unterscheiden sich in der Anzahl der abzurufenden Dateien. In dieser Testreihe werden ausschließlich Bilddateien im JPEG-Format mit einer Größe zwischen 7 Kilobyte und 18 Kilobyte abgerufen. Die Anzahl der abzurufenden Bilddateien beginnt bei 20 und endet bei 100.

### 4.4.1 Test 1

Als Basis für diesen Leistungstest dient ein HTML-Dokument, welches mit 10 JPEG-Dateien zwischen 7,8 Kilobyte und 17,3 Kilobyte Größe versehen ist. Es werden 11 Anfragen an den Server gestellt (eine Anfrage zum Laden des HTML-Dokumentes und weitere 10 Anfragen zum Laden der in dem HTML-Dokument verlinkten JPEG-Dateien). Insgesamt werden 109,7 Kilobyte Daten vom Server geladen.

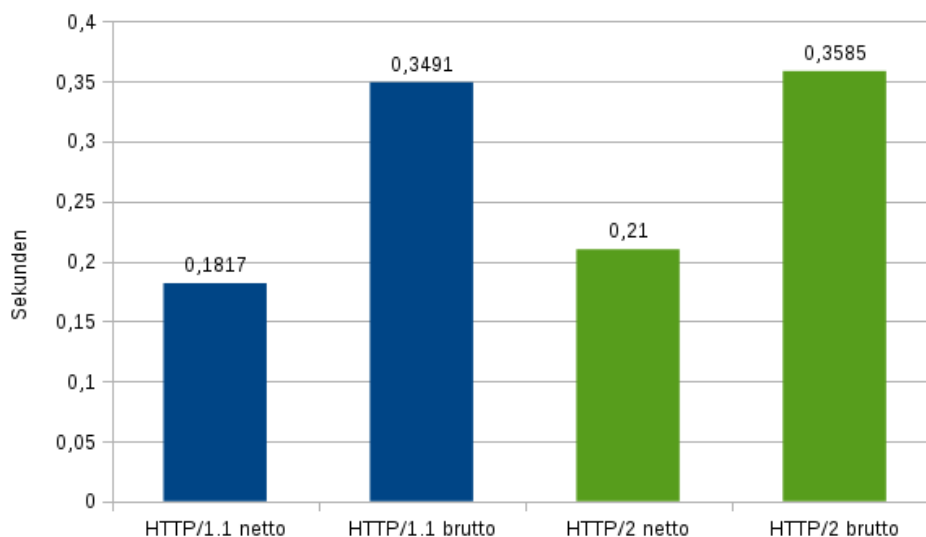


Abbildung 11: Test 1, Testreihe 1

In Abbildung 11 werden die durchschnittlichen Ladezeiten aufgezeigt. Diese verdeutlicht, dass sich HTTP/1.1 in diesem Test durchsetzt. Sowohl

in der Brutto-Ladezeit, als auch in der Netto-Ladezeit ist das Laden der Webseite mit HTTP/1.1 schneller. Die Brutto-Ladezeit von HTTP/1.1 liegt bei 0,35 Sekunden. Der Aufruf der Webseite dauert mit HTTP/2 0,36 Sekunden, also 0,01 Sekunden länger als mit HTTP/1.1. Bei der Netto-Ladezeit der Webseiten fällt der Unterschied noch größer aus. Mit HTTP/1.1 dauert das Laden der Ressourcen 0,18 Sekunden, mit HTTP/2 hingegen 0,21 Sekunden.

In diesem Leistungstest geht das Laden der Webseite mit HTTP/1.1 schneller als mit HTTP/2. Im Laufe der weiteren Tests wird sich zeigen, wie sich das Verhältnis zwischen HTTP/1.1 und HTTP/2 bei zahlen- und größenmäßiger Veränderung der angefragten Ressourcen verändert.

#### 4.4.2 Test 2

In diesem Test ist die Anzahl der abzurufenden Ressourcen hinsichtlich Test 1 verdoppelt worden. Es werden 20 JPEG-Dateien mit einer Größe zwischen 4,6 Kilobyte und 17,3 Kilobyte angefordert. Insgesamt müssen 196,5 Kilobyte an Daten geladen werden. In der Summe werden 21 Anfragen durchgeführt.

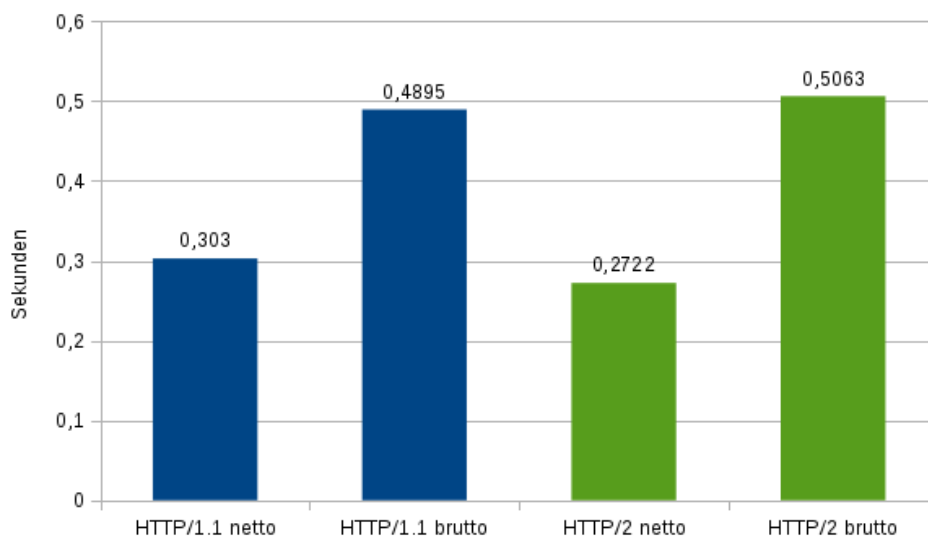


Abbildung 12: Test 2, Testreihe 1

Die Auswertung des Leistungstests (siehe Abbildung 12) zeigt, dass das

Laden mit HTTP/1.1 zwar in der Brutto-Ladezeit um 0,02 Sekunden schneller ist als mit HTTP/2, jedoch beträgt die Netto-Ladezeit mit HTTP/2 0,03 Sekunden weniger. Was die reine Übertragung der Daten angeht, hat sich das Laden mit HTTP/2 als schneller erwiesen. Da jedoch die Brutto-Ladezeit im Vordergrund stehen soll, ist auch hier das Laden der Webseite mit HTTP/1.1 schneller.

#### 4.4.3 Test 3

In diesem Leistungstest werden 50 Bilddateien vom Server geladen, also mehr als das doppelte an Ressourcen, als im vorherigen Test. Die gesamten Nutzdaten betragen 412,8 Kilobyte. Insgesamt werden 51 Elemente angefragt. Auch in diesem Leistungstest haben die JPEG-Dateien eine Größe zwischen 4,6 Kilobyte und 17,3 Kilobyte.

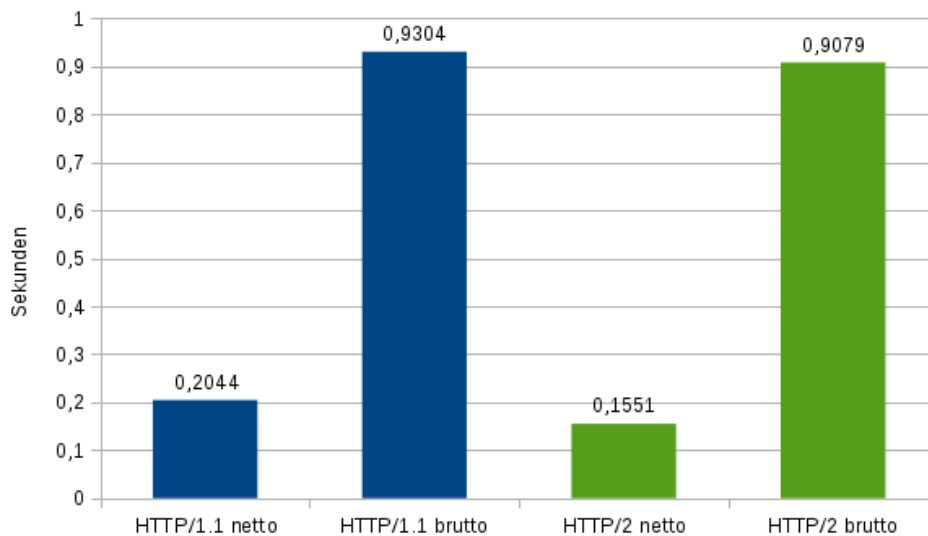


Abbildung 13: Test 3, Testreihe 1

In diesem Leistungstest ist HTTP/2 sowohl in der Netto-Ladezeit als auch in der Brutto-Ladezeit schneller als HTTP/1.1 (siehe Abbildung 13).

#### 4.4.4 Test 4

Es werden 100 JPEG-Dateien mit einer Größe zwischen 4,6 Kilobyte und 17,3 Kilobyte vom Server geladen. Die Nutzdaten betragen in der Summe 800,3 Kilobyte.

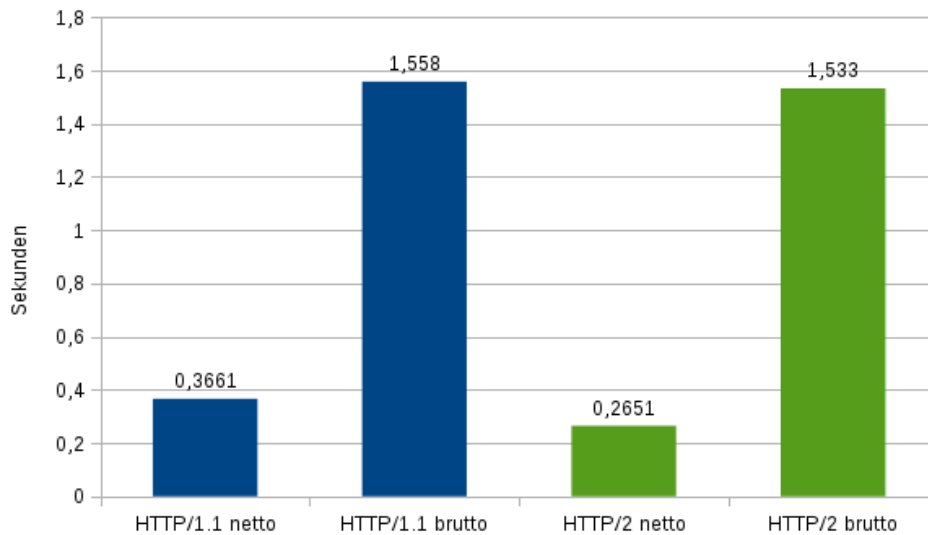


Abbildung 14: Test 4, Testreihe 1

Auch in diesem Test ist HTTP/2 sowohl in der Netto-Ladezeit als auch in der Brutto-Ladezeit schneller als HTTP/1.1 (siehe Abbildung 14).

#### 4.4.5 Zwischenfazit Testreihe 1

Zu Beginn der Testreihe war das Laden der Webseite mit HTTP/1.1 schneller als mit HTTP/2 (siehe Test 1). Im zweiten Test war die Netto-Ladezeit mit HTTP/2 bereits geringer als mit HTTP/1.1. In den letzten beiden Tests der ersten Testreihe war das Laden mit HTTP/2 sowohl in der Brutto-Ladezeit als auch in der Netto-Ladezeit schneller als mit HTTP/1.1.

Je mehr abzurufende Dateien, desto schneller ist das Laden mit HTTP/2.

## 4.5 Testreihe 2

Nachdem in Testreihe 1 lediglich JPEG-Dateien kleiner Größe abgerufen wurden, wird die Größe der JPEG-Dateien in dieser Testreihe erhöht. Es werden ausschließlich JPEG-Dateien mit einer Größe zwischen 7 Kilobyte und 650 Kilobyte abgerufen. Die Anzahl der abzurufenden Bilddateien ist mit der Anzahl aus Testreihe 1 identisch, sie beginnt bei 20 JPEG-Dateien und endet bei 100 JPEG-Dateien.

### 4.5.1 Test 1

In diesem Test werden 10 JPEG-Dateien vom Server abgerufen. Die Bilddateien haben eine Größe zwischen 40 Kilobyte und 150 Kilobyte. Die gesamten Nutzdaten betragen 829,6 Kilobyte. Es finden insgesamt 11 Anfragen an den Server statt.

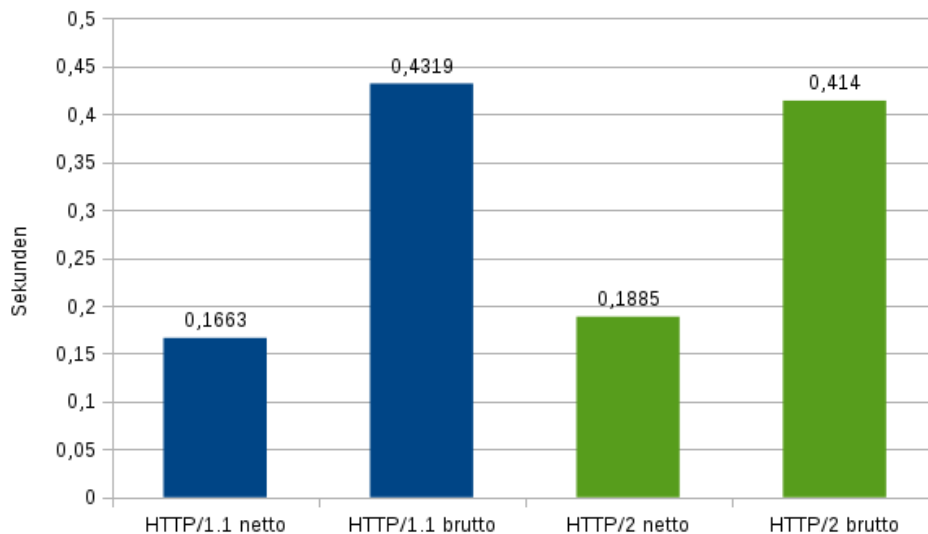


Abbildung 15: Test 1, Testreihe 2

Das Laden der Webseite geht in diesem Test mit HTTP/2 schneller als mit HTTP/1.1 (siehe Abbildung 15), da die Brutto-Ladezeit von HTTP/2 um 0,02 Sekunden geringer ist als die von HTTP/1.1. Jedoch ist die reine Übertragungsdauer der Daten, also die Netto-Ladezeit via HTTP/1.1, um 0,02 Sekunden schneller als mit HTTP/2.



### 4.5.2 Test 2

In diesem Test werden 20 JPEG-Dateien vom Server abgerufen. Die Bilddateien haben eine Größe zwischen 40 Kilobyte und 290 Kilobyte. Die gesamten Nutzdaten betragen 1,9 Megabyte. Es finden insgesamt 21 Anfragen an den Server statt.

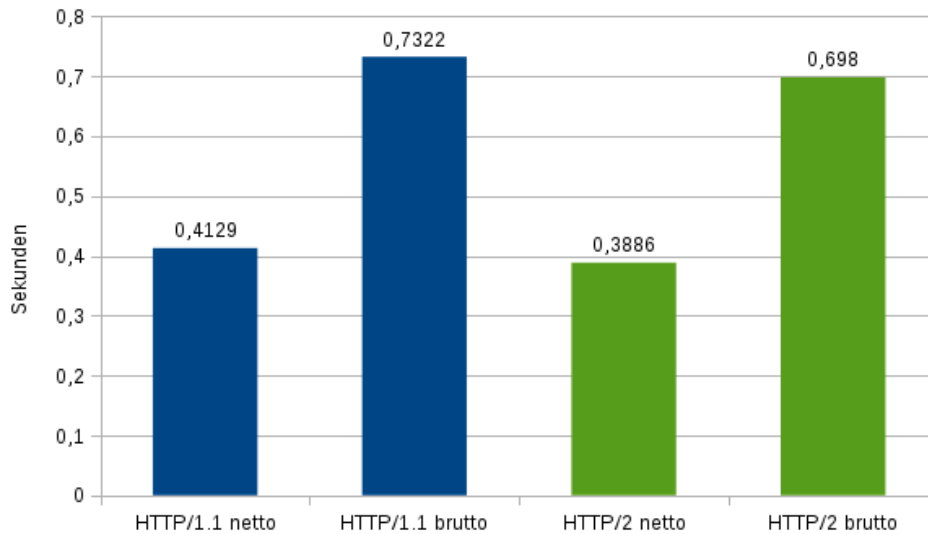


Abbildung 16: Test 2, Testreihe 2

In diesem Test ist das Laden der Webseite, also die Brutto-Ladezeit, mit HTTP/2 um 0,03 Sekunden schneller als mit HTTP/1.1. Auch die reine Übertragung der Daten, also die Netto-Ladezeit, geht mit HTTP/2 um 0,02 Sekunden schneller (siehe Abbildung 16).

### 4.5.3 Test 3

In diesem Test finden 51 Anfragen an den Server statt. Darunter ist ein HTML-Dokument und 50 weitere JPEG-Dateien. Die Bilddateien haben eine Größe zwischen 30 Kilobyte und 290 Kilobyte. Die Summe der Nutzdaten beträgt 5,5 Megabyte.

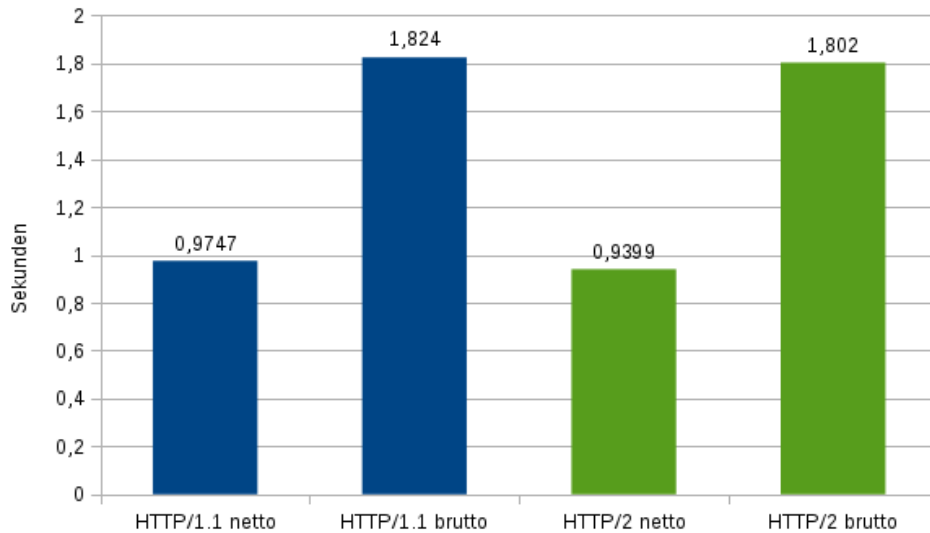


Abbildung 17: Test 3, Testreihe 2

In diesem Leistungstest ist das Laden der Daten und der gesamten Webseite mit HTTP/2, also die Netto-Ladezeit und die Brutto-Ladezeit, um 0,03 beziehungsweise 0,02 Sekunden schneller als mit HTTP/1.1 (siehe Abbildung 17).

#### 4.5.4 Test 4

Dieser Leistungstest ist der letzte Test der zweiten Testreihe. Insgesamt finden 101 Anfragen an der Server statt. Darunter ist ein HTML-Dokument und 100 JPEG-Dateien von 30 Kilobyte bis 650 Kilobyte Größe. Die Nutzdaten betragen in der Summe 11,8 Megabyte.

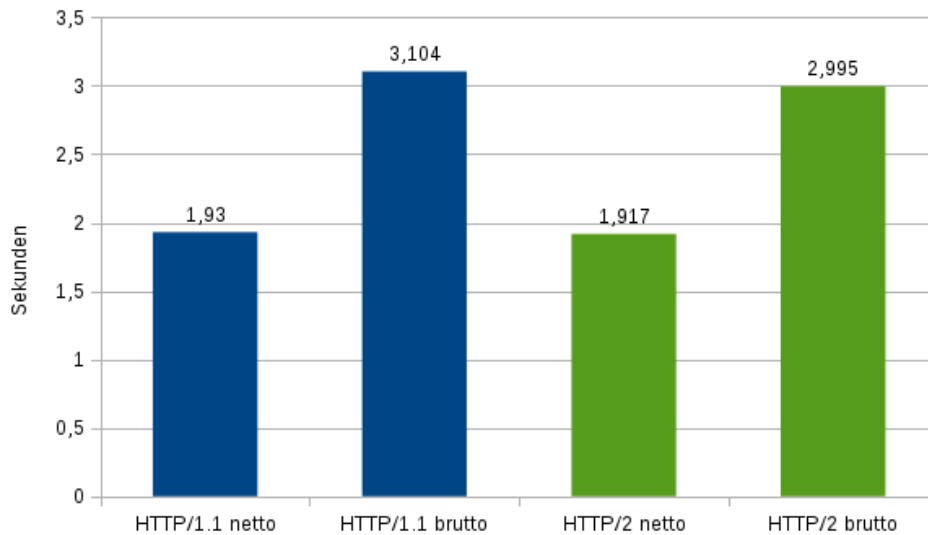


Abbildung 18: Test 4, Testreihe 2

In diesem Test ist das Laden der Webseite mit HTTP/2 in der Brutto-Ladezeit um 0,01 Sekunden schneller und in der Netto-Ladezeit ebenso um 0,01 Sekunden schneller als mit HTTP/1.1 (siehe Abbildung 18).

#### 4.5.5 Zwischenfazit Testreihe 2

In der zweiten Testreihe wird der Trend der ersten Testreihe fortgesetzt. In dem ersten Leistungstest der zweiten Testreihe war das Laden der Webseite mit HTTP/2 schneller, da die Brutto-Ladezeit geringer war, jedoch war die Netto-Ladezeit mit HTTP/1.1 geringer. Bei den folgenden Tests und steigender Anzahl der abzurufenden Ressourcen war das Laden der Webseite mit HTTP/2 in jedem Fall schneller als mit HTTP/1.1.

Bei steigender Anzahl der angefragten Ressourcen wird das Abrufen der Webseite mit HTTP/2 schneller bewältigt als mit HTTP/1.1.

## 4.6 Testreihe 3

Die beiden vorherigen Testreihen haben sich hauptsächlich in der Anzahl der abzurufenden Ressourcen unterschieden, dabei war die Spannweite der Größe der JPEG-Dateien gering. In dieser Testreihe soll besonders die Übertragungszeit großer Dateien, mit einer Größe zwischen 1 Megabyte und 35 Megabyte, gemessen werden.

In einem abschließenden Test werden alle der zuvor verwendeten Dateien auf einmal angefragt und vom Server geladen.

### 4.6.1 Test 1

Bei diesem Leistungstest werden acht JPEG-Dateien mit einer Größe von 600 Kilobyte bis 850 Kilobyte und zwei JPEG-Dateien mit einer Größe von 23,3 Megabyte und 35,5 Megabyte vom Server abgerufen. Die Nutzdaten betragen in der Summe 64,4 Megabyte. Insgesamt werden 11 Anfragen an den Server getätigt.

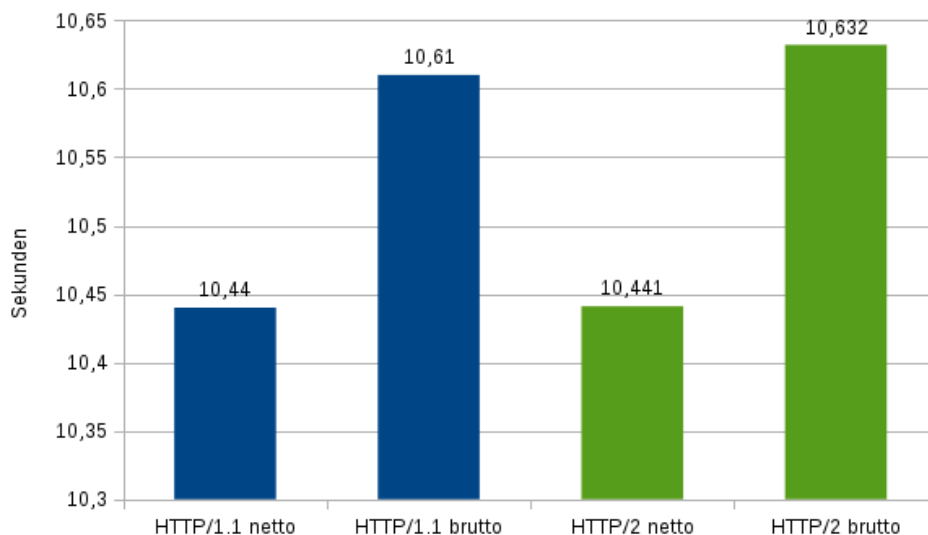


Abbildung 19: Test 1, Testreihe 3

Die Netto-Ladezeiten beider Protokollversionen sind identisch, jedoch hat HTTP/1.1 in der Brutto-Ladezeit einen Vorteil von 0,02 Sekunden (siehe

Abbildung 19).

#### 4.6.2 Test 2

In diesem Leistungstest finden insgesamt 201 Anfragen an den Server statt. Es handelt sich dabei um 200 JPEG-Dateien mit einer Größe zwischen 6 Kilobyte und 650 Kilobyte. Die Nutzdaten betragen in der Summe 12,6 Megabyte.

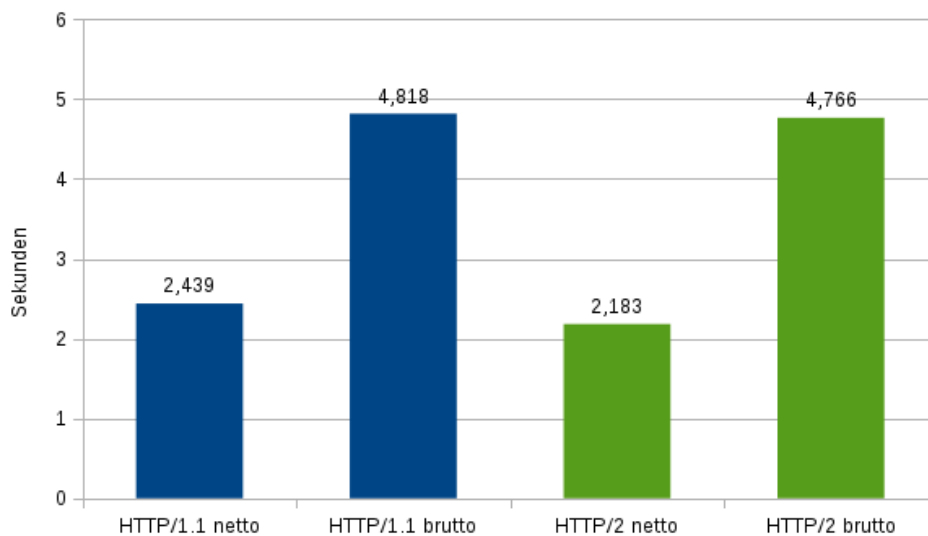


Abbildung 20: Test 2, Testreihe 3

Die Auswertung dieses Leistungstests zeigt, dass das Laden der Webseite mit HTTP/2 in der Netto-Ladezeit um 0,26 Sekunden und in der Brutto-Ladezeit um 0,05 Sekunden schneller ist als mit HTTP/1.1 (siehe Abbildung 20).

#### 4.6.3 Test 3

In diesem Leistungstest werden alle Elemente der vorherigen Tests abgerufen. Dieser Test ist zu Test 2 der dritten Testreihe identisch, bis auf 10 hinzugekommene JPEG-Dateien mit Größen zwischen 800 Kilobyte und 35,5 Megabyte. Es werden insgesamt 210 Anfragen an den Server getätigt. Die Nutzdaten betragen 77 Megabyte. Die JPEG-Dateien haben eine Größe von 6 Kilobyte bis 35,5 Megabyte.

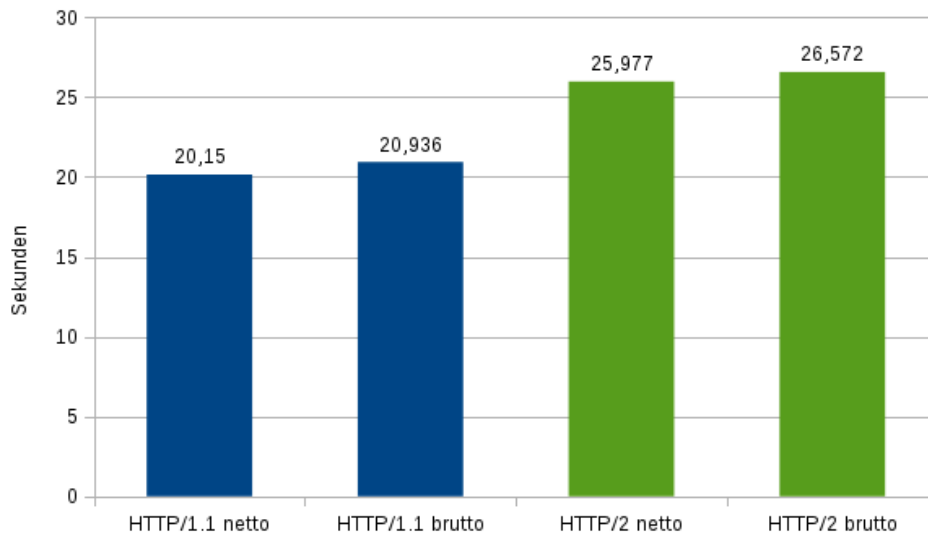


Abbildung 21: Test 3, Testreihe 3

Das Laden der Webseite ist mit HTTP/1.1 schneller als mit HTTP/2. Dabei ist das Laden mit HTTP/1.1 sowohl in der Netto-Ladezeit als auch in der Brutto-Ladezeit um über fünf Sekunden schneller als mit HTTP/2. Im Vergleich zu Test 2 der dritten Testreihe, in dem HTTP/2 einen Geschwindigkeitsvorteil hatte, schneidet HTTP/2 in diesem Leistungstest schlechter ab (siehe Abbildung 21).

#### 4.6.4 These

Vermutlich ist die Übertragung größerer Dateien mit HTTP/2 nicht so effizient wie mit HTTP/1.1.

#### 4.6.5 Test 4

Um die im vorherigen Test erstellte These zu überprüfen, werden in diesem Leistungstest ausschließlich große JPEG-Dateien abgerufen. Es werden zwei JPEG-Dateien mit einer Größe von 23,3 Megabyte und 35,5 Megabyte vom Server abgerufen. Demnach werden drei Anfragen an den Server getätigt. Die Nutzdaten betragen 58,9 Megabyte.

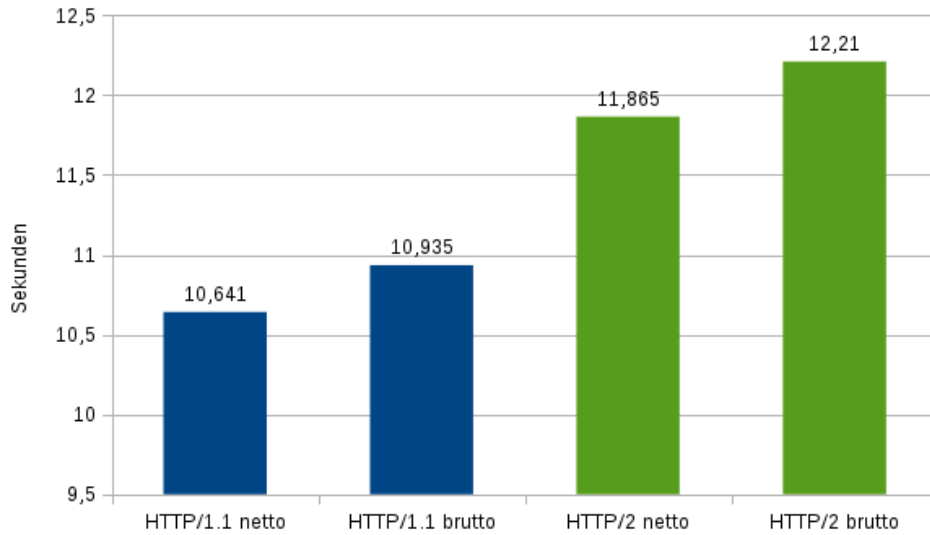


Abbildung 22: Test 4, Testreihe 3

Die Auswertung dieses Leistungstests gibt zu erkennen, dass HTTP/1.1 auch in diesem Testszenario große Dateien schneller abrufen als HTTP/2. Die Netto-Ladezeit ist mit HTTP/1.1 um 1,23 Sekunden geringer als mit HTTP/2. Auch die Brutto-Ladezeit beträgt weniger, nämlich 1,27 Sekunden (siehe Abbildung 22).

#### 4.6.6 Zwischenfazit Testreihe 3

Die dritte Testreihe hat gezeigt, dass die Übertragung großer Dateien mittels HTTP/2 länger dauert als mit HTTP/1.1. Besonders auffällig sind die Ladezeiten von Test 3 der dritten Testreihe. Bei diesem Leistungstest dauert das Laden der Webseite mit HTTP/2 sowohl brutto als auch netto über 5 Sekunden länger als mit HTTP/1.1.

## 4.7 Testreihe 4

In dieser Testreihe werden die Ladezeiten von frei zugänglichen Webseiten aus dem Alltag analysiert. Dafür wurden die folgenden Webseiten, welche eine HTTP/2-Unterstützung bieten, ausgewählt:

- <https://www.google.de>
- <https://http2.akamai.com>
- <https://www.youtube.com>
- <https://twitter.com>

Der Unterschied zu den vorherigen Testreihen ist, dass bei dieser Testreihe die angefragten Ressourcen auf verschiedenen Servern beherbergt sind, sodass TCP-Verbindungen zu unterschiedlichen Servern notwendig sind. Außerdem sind die Ressourcentypen vielfältiger als in den vorherigen Testreihen.

In den folgenden Tests kommt es vor, dass die Netto-Ladezeit höher ist als die Brutto-Ladezeit. Dies ist dadurch bedingt, dass in den Webseiten eingebettete Videos, Widgets etcetera teilweise nachgeladen werden. Weiterhin ist die Brutto-Ladezeit von höherer Bedeutung.

### 4.7.1 Test 1 <https://www.google.de>

In diesem Leistungstest wird die Ladezeit der Webseite <https://www.google.de>, mit HTTP/1.1 und HTTP/2 untersucht. Es finden 11 Anfragen statt, die angefragten Ressourcen liegen auf 4 verschiedenen Servern. Insgesamt werden 368,1 Kilobyte geladen.



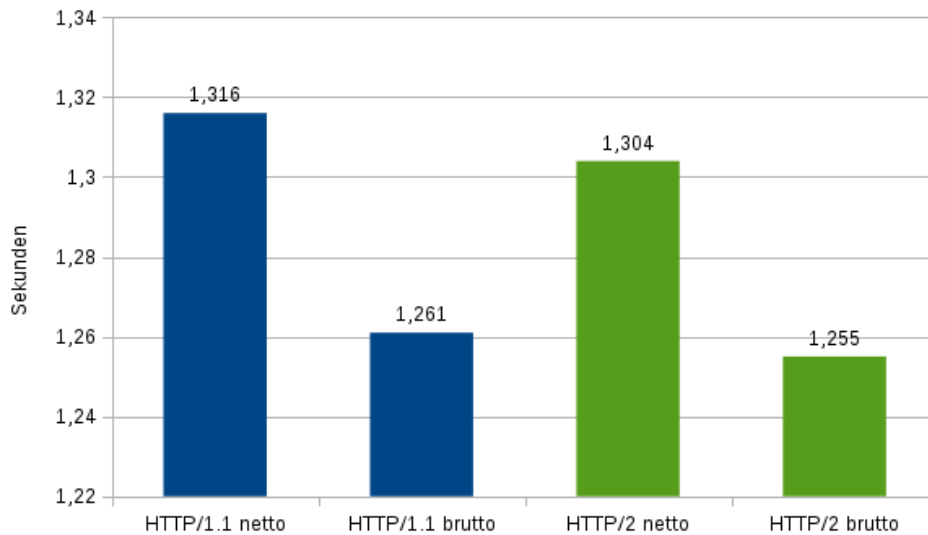


Abbildung 23: Test 1, Testreihe 4

Das Ergebnis dieses Leistungstests zeigt, dass das Laden der Webseite <https://www.google.de>, mit HTTP/2 in der Netto-Ladezeit um 0,055 Sekunden und in der Brutto-Ladezeit um 0,049 Sekunden schneller ist als das Laden mit HTTP/1.1 (siehe Abbildung 23).

#### 4.7.2 Test 2 <https://http2.akamai.com>

In diesem Leistungstest wird die Ladezeit der Webseite <https://http2.akamai.com> mit HTTP/1.1 und HTTP/2 untersucht. Es finden 18 Anfragen statt, in denen 637,6 Kilobyte Nutzdaten geladen werden. Die Ressourcen liegen auf sechs verschiedenen Servern.

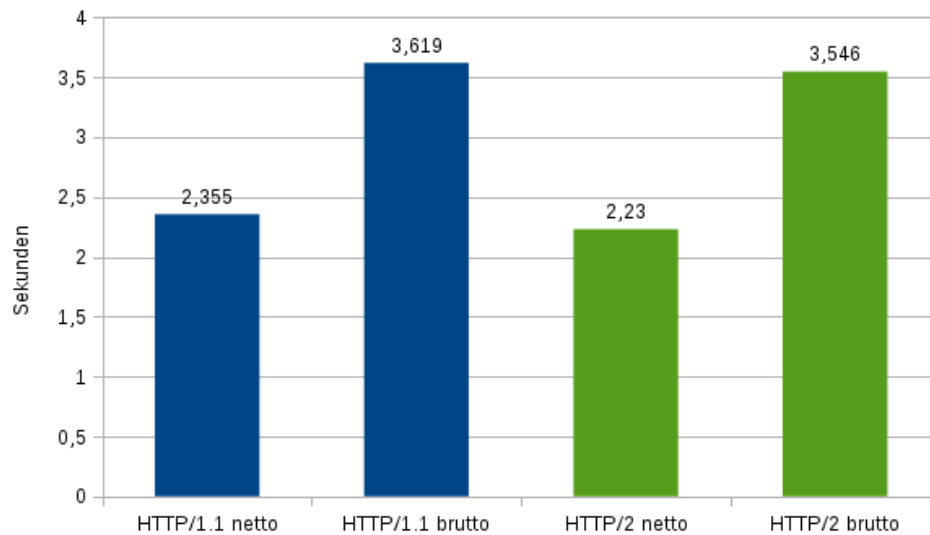


Abbildung 24: Test 2, Testreihe 4

Als Ergebnis dieses Leistungstests ist zu sehen, dass das Laden der Seite mit HTTP/2 schneller ist als mit HTTP/1.1. Dabei beträgt die Netto-Ladezeit mit HTTP/2 0,125 Sekunden und die Brutto-Ladezeit 0,073 Sekunden weniger als mit HTTP/1.1 (siehe Abbildung 24).

#### 4.7.3 Test 3 <https://www.youtube.com>

In diesem Leistungstest wird die Ladezeit der Webseite <https://www.youtube.com> mit HTTP/1.1 und HTTP/2 untersucht. Es finden zwischen 68 und 71 Anfragen an fünf verschiedene Server statt. Insgesamt werden zwischen 1,2 Megabyte und 1,5 Megabyte Nutzdaten geladen.

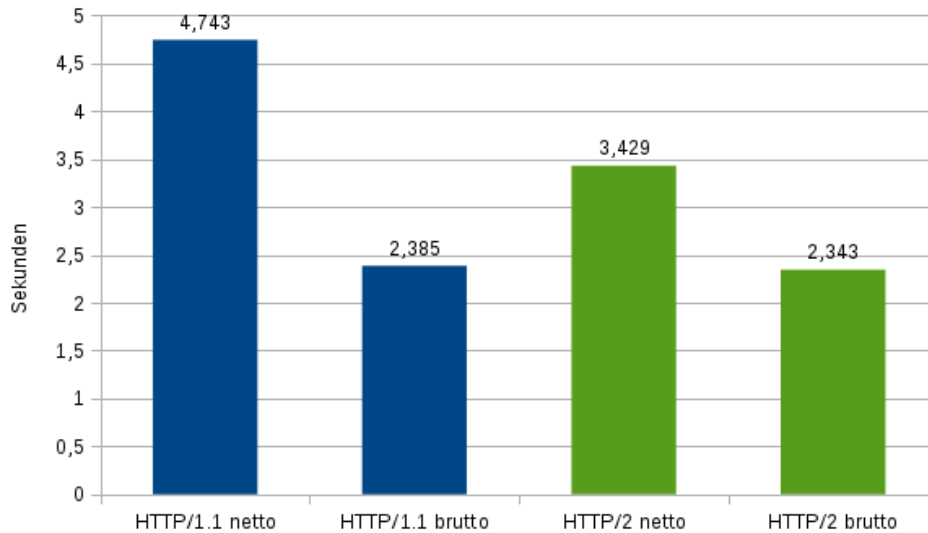


Abbildung 25: Test 3, Testreihe 4

Auch in diesem Leistungstest ist das Laden der Webseite mit HTTP/2 schneller als mit HTTP/1.1. Auffällig bei diesem Test ist, dass das Nachladen diverser Elemente, in diesem Fall die Netto-Ladezeit, zwischen beiden Protokollversionen um 1,313 Sekunden differiert. Der Unterschied bei der Brutto-Ladezeit fällt geringer aus, hier sind es 0,042 Sekunden (siehe Abbildung 25).

#### 4.7.4 Test 4 <https://twitter.com>

In diesem Leistungstest wird die Ladezeit der Webseite <https://twitter.com> mit HTTP/1.1 und HTTP/2 untersucht. Es finden 15 bis 18 Anfragen statt, die Nutzdaten betragen 1,2 Megabyte bis 1,5 Megabyte. Die angefragten Ressourcen sind auf 4 verschiedenen Servern beherbergt.

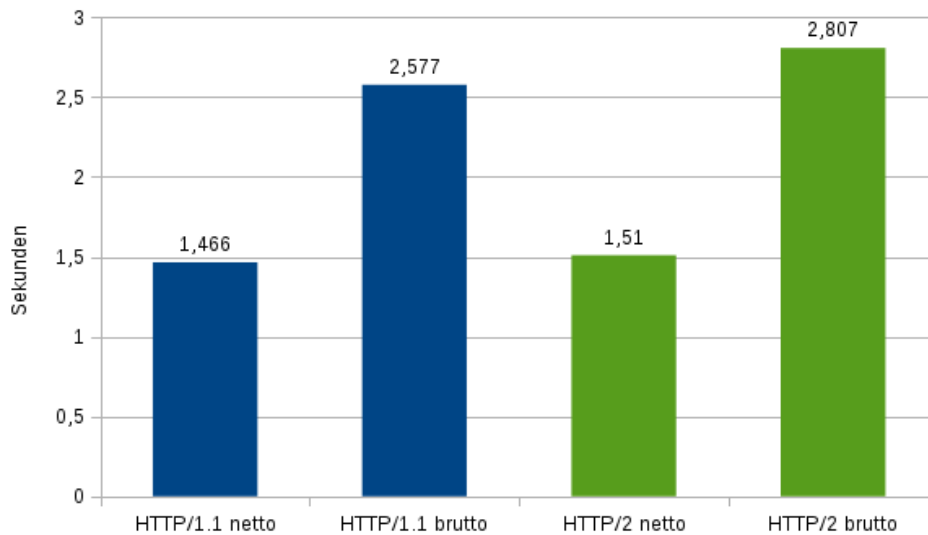


Abbildung 26: Test 4, Testreihe 4

Bei diesem Leistungstest ist das Laden der Webseite mit HTTP/1.1 schneller als mit HTTP/2. Die Netto-Ladezeit ist mit HTTP/1.1 um 0,044 Sekunden und die Brutto-Ladezeit um 0,23 Sekunden schneller als mit HTTP/2 (siehe Abbildung 26).

#### 4.7.5 Zwischenfazit Testreihe 4

In dieser Testreihe hat sich gezeigt, dass HTTP/2 in der realen Anwendung durchaus einen Geschwindigkeitsvorteil im Laden einer Webseite bietet. Das Laden der Webseiten war in drei der vier Tests mit HTTP/2 schneller als mit HTTP/1.1.

Besonders die Netto-Ladezeit von Test 3 dieser Testreihe zeigt, dass das Laden vieler Ressourcen unterschiedlicher Hosts mit HTTP/2 schneller ist.

## 5 Fazit

Im folgenden Abschnitt der Arbeit wird das Fazit der Leistungsuntersuchung bekannt gegeben. Dabei wird auf die einzelnen Testreihen hinsichtlich beider Protokollversionen, genauer eingegangen.

Um nicht steuerbaren Faktoren, wie der Netzwerkauslastung und der Latenz entgegenzuwirken, die die Messergebnisse verfälschen könnten, wurden folgende Maßnahmen ergriffen: Alle voneinander abhängigen Leistungstests wurden in direkter zeitlicher Abfolge aufeinander durchgeführt. Die Leistungstests wurden allesamt in den Morgenstunden zwischen 08:00 Uhr und 10:00 Uhr, Mitteleuropäischer Zeit durchgeführt.

Besonders in den ersten beiden Testreihen, bei denen die angefragten Ressourcen ausschließlich auf einem Server lagen, hat sich gezeigt, dass HTTP/2 das Abrufen vieler Elemente geringer ( $< 20$  Kilobyte) bis mittlerer Größe ( $< 1$  Megabyte) schneller bewältigt als HTTP/1.1. Dies ist auf das Multiplexing von HTTP/2 zurückzuführen. Hier hat HTTP/1.1 durch die Limitierung von sechs gleichzeitigen TCP-Verbindungen zu einem Server das Nachsehen. In Testreihe 3, in der die Größe der abzurufenden Elemente stark erhöht wurde (bis 35 Megabyte), zeigt HTTP/2 Schwäche. Hier war das Laden der Seite mit HTTP/1.1 um bis zu 5 Sekunden schneller als mit HTTP/2. Die Größe der Nutzdaten, welche in Testreihe 3 bis 77 Megabyte betragen, scheinen ein Problem für HTTP/2 darzustellen.

Während den Messungen der ersten drei Testreihen machte HTTP/1.1 insgesamt einen solideren Eindruck. Es waren weniger Ausfälle zu beklagen, außerdem war die Streuung der Ladezeiten bei HTTP/1.1 geringer als bei HTTP/2. Die Übertragung von HTTP/2 hingegen musste des Öfteren abgebrochen werden, da das Laden in einem Timeout endete. Diese Fehlmessungen sind nicht in die Auswertung mit eingeflossen.

In Testreihe 4, welche den Gebrauch beider Protokollversionen am realistischsten widerspiegelt, da Ressourcen verschiedenster Dateitypen von

diversen Servern zu laden waren, war das Laden der Webseiten in drei von vier Fällen mit HTTP/2 schneller als mit HTTP/1.1. Besonders in Test 3 der vierten Testreihe konnte ein erheblicher Geschwindigkeitsvorteil in der Netto-Ladezeit mit HTTP/2 festgestellt werden.

Insgesamt muss man berücksichtigen, dass Browser und HTTP-Servercode für das HTTP/1.1 Protokoll seit über 15 Jahren entwickelt und optimiert werden. Dagegen sind die HTTP/2 Implementierungen noch relativ neu, das heißt mit der weiteren Entwicklung und Optimierung des neuen Protokolls sind noch weitere Leistungssteigerungen durch Code-Optimierung zu erwarten.

Die signifikanten Leistungssteigerungen bei öffentlichen Webangeboten mit HTTP/2 gegenüber HTTP/1.1 sind möglicherweise nicht allein protokollbedingt, sondern auch auf die eingesetzte Hardware zurückzuführen, da modernere Serversoftware tendenziell auch auf modernerer Hardware läuft.

Bei der Verwendung anderer Webserver (zum Beispiel Apache mit `mod_h2`) könnten sich auch bei den Testergebnissen, die auf eigener Serverhardware stattgefunden haben, Abweichungen ergeben, die implementierungsbedingt (anderer Programmcode) sind. Hier ist mehr Sicherheit hinsichtlich der gemessenen Performance-Unterschiede zwischen den Protokollen zu erwarten, je mehr unterschiedliche Implementierungen (HTTP-Server) in den Testreihen berücksichtigt werden, da dadurch gemittelt verstärkt weniger die konkrete Implementierung als das Protokoll ausschlaggebend für die gemessene Performance ist.

Caches im Internet beziehungsweise beim Internetserviceprovider können ebenfalls Einfluss auf die gemessenen Werte haben. Um diese Einflüsse auszuschalten, könnte man weitere Testreihen gegen den Webserver fahren, die nicht über das Internet laufen, zum Beispiel indem man den Webbrowser auf einer weiteren virtuellen Maschine im selben Subnetz durchführt.

Ungeachtet der Messtoleranzen und möglicher Fehler lässt sich dennoch als Fazit folgendes festhalten: Auch die jetzt noch jungen HTTP/2 Imple-

mentierungen sind generell robust und bringen kaum Geschwindigkeitsnachteile gegenüber HTTP/1.1 mit sich. In Zukunft ist zu erwarten, dass speziell die Fälle, in denen HTTP/2 Implementierungen langsamer sind als HTTP/1.1 seitens der Softwarehersteller Maßnahmen ergriffen werden, diese zu eliminieren. Weiterhin ist die Internetinfrastruktur durch den langjährigen Einsatz von HTTP/1.1 geprägt. Auch hier sind Verbesserungen beispielsweise bei Proxies und Caches zu erwarten, die HTTP/2 tendenziell stärkere Vorteile und Leistungssteigerungen verschaffen könnten.

Die Hypothese, dass sich für mehr parallele Anfragen, speziell in der Größenordnung mehr als sechs parallele Anfragen, deutliche Vorteile für HTTP/2 gegenüber der Vorgängerversion ergeben, konnte erhärtet werden. Hier lieferten speziell die Testreihen 1 und 2 deutliche Hinweise.

## 6 Ausblick

Im letzten Abschnitt dieser Arbeit gibt es einen Ausblick für HTTP/2, außerdem werden Möglichkeiten für eine aussagekräftigere Leistungsuntersuchung vorgestellt.

Obwohl mittlerweile alle gängigen Browser HTTP/2 unterstützen, ist die HTTP/2 Implementierung bei Webservern noch nicht so weit verbreitet. Auch wenn bereits viele Seiten mit hohen Nutzerzahlen, zum Beispiel Google auf HTTP/2 setzen, setzen viele Anbieter noch überwiegend auf HTTP/1.1. Im Laufe der Zeit wird sich dies wahrscheinlich ändern, wenn namhafte Webserver erst einmal HTTP/2 im Standard implementiert haben.

Wie schon in einem vorherigen Teil dieser Arbeit erwähnt, ist HTTP/2 nicht dafür gedacht alte Protokollversionen zu ersetzen, sondern diese eher zu ergänzen. Es wird mit Sicherheit spezifische Anwendungsfälle für gewisse Protokollversionen geben. Bei Webseiten mit vielen Elementen wird es mit HTTP/2 einen Geschwindigkeitsvorteil gegenüber HTTP/1.1 geben. Jedoch ist, wie sich in der Leistungsuntersuchung dieser Arbeit herausgestellt hat, die neue Protokollversion des Hypertext Transfer Protocols nicht immer im Vorteil, gerade bei dem Abrufen großer Dateien ist das Laden mit HTTP/2 nicht so schnell wie mit HTTP/1.1.

Trotz allem wird HTTP/2 einen positiven Effekt auf das Internet haben, besonders weil die Serverauslastung durch die minimierte Anzahl an TCP-Verbindungen stark reduziert wird.

Hinsichtlich der Bewertung der Ergebnisse von Testreihe 3, bei der sich erwies, dass HTTP/2 bei großen Dateien tendenziell geringere oder gar keine Geschwindigkeitsvorteile gegenüber HTTP/1.1 hat, ergaben sich weitere Fragen. Insbesondere drängte sich auf, ob die gemessenen Werte nicht vielleicht ursächlich in den Protokollschichten unterhalb der HTTP-Schicht begründet sind. Die zugehörige Überlegung ist, dass wenn eine einzelne Anfrage innerhalb einer TCP-Verbindung abgearbeitet wer-



den kann, dann spielen die Optimierungen der Protokollschichten unter HTTP (insbesondere Geschwindigkeitszuwachs nach slow-start) eine größere Rolle, die dann den Header- und Frame-Overhead von HTTP/2 stärker nachteilig hervortreten lassen. Um dies auszuschließen zu können, könnte man eine weitere Messreihe durchführen, bei der zunächst ermittelt wird, wie lange die Antwortzeit für eine besonders große Ressource unter HTTP/1.1 ist. Anschließend wird die HTTP-Server-Konfiguration so gewählt, dass dieser im Schnitt  $x$ -mal für diese Ressource in einen Timeout läuft, was bei HTTP/1.1 und den derzeitigen Implementierungen für Server und Browser bedingt, dass eine entsprechende Anzahl TCP-Verbindungen für die Ressource aufgebaut werden müssen. Somit würden die Optimierungen der darunterliegenden Protokollschichten weitestgehend wegfallen. Das gleiche würde man mit HTTP/2, mit demselben Timeout durchführen und im Anschluss die gemessenen Werte vergleichen und analysieren. Dann sollte sich zeigen inwieweit Optimierungen der Protokollschichten unterhalb der HTTP-Schicht Einfluss auf die Messungen dieser Arbeit hatten.

## 7 Literaturverzeichnis

- [ARP] ARPANET. <https://en.wikipedia.org/wiki/ARPANET>. – [Online; aufgerufen am 31.08.2015]
- [Bau12] BAUN, Christian: *Computernetze kompakt*. Springer Vieweg, 2012
- [Bau14] BAUN, Christian: Foliensatz 11: Transportschicht / Transport Layer. (2014). <http://christianbaun.de/2>. – [Online; aufgerufen am 01.09.2015]
- [BL92] BERNERS-LEE, T.: Why a new protocol? (1992). <http://www.w3.org/History/19921103-hypertext/hypertext/WWW/Protocols/WhyHTTP.html>. – [Online; aufgerufen am 31.08.2015]
- [BL96] BERNERS-LEE, T.: Request for Comments: 1945. (1996). <http://tools.ietf.org/html/rfc1945>. – [Online; aufgerufen am 05.08.2015]
- [fra] An in depth overview of HTTP/2-Wire Format. <http://undertow.io/blog/2015/04/27/An-in-depth-overview-of-HTTP2.html>. – [Online; aufgerufen am 05.08.2015]
- [HPA] An in depth overview of HTTP/2-HPACK. <http://undertow.io/blog/2015/04/27/An-in-depth-overview-of-HTTP2.html>. – [Online; aufgerufen am 25.08.2015]
- [htt] HTTP/2 Frequently Asked Questions. <http://http2.github.io/faq/>. – [Online; aufgerufen am 07.08.2015]
- [huf] Huffman-Code. <http://www.iti.fh-flensburg.de/lang/algorithmen/code/huffman/huffman.htm>. – [Online; aufgerufen am 28.08.2015]
- [MOO] Mooresches Gesetz. [https://de.wikipedia.org/wiki/Mooresches\\_Gesetz](https://de.wikipedia.org/wiki/Mooresches_Gesetz). – [Online; aufgerufen am 01.09.2015]
- [mul] Using Multiple TCP Connections. [http://chimera.labs.oreilly.com/books/1230000000545/ch11.html#HTTP11\\_MULTIPLE\\_CONNECTIONS](http://chimera.labs.oreilly.com/books/1230000000545/ch11.html#HTTP11_MULTIPLE_CONNECTIONS). – [Online; aufgerufen am 08.08.2015]

- [Ope] OpenSSL. [https://wiki.debian.org/Self-Signed\\_Certificate](https://wiki.debian.org/Self-Signed_Certificate). – [Online; aufgerufen am 20.08.2015]
- [pri] The http2 protocol-Priorities and dependencies. <http://http2-explained.readthedocs.org/en/latest/src/http2protocol.html#Priorities-and-dependencies>. – [Online; aufgerufen am 24.08.2015]
- [Pru] PRUSTY, Narayan: HTTP/2 Compatibility with old Browsers and Servers. <http://qnimate.com/http2-compatibility-with-old-browsers-and-servers>. – [Online; aufgerufen am 15.08.2015]
- [Pru15] PRUSTY, Narayan: What is Multiplexing in HTTP/2? (2015). <http://qnimate.com/what-is-multiplexing-in-http2/>. – [Online; aufgerufen am 20.08.2015]
- [RFC15] Hypertext Transfer Protocol Version 2 (HTTP/2). (2015). <https://tools.ietf.org/html/rfc7540>. – [Online; aufgerufen am 05.08.2015]
- [RTT] Paketumlaufzeit. <https://de.wikipedia.org/wiki/Paketumlaufzeit>. – [Online; aufgerufen am 01.09.2015]
- [ser] Chapter 12. HTTP/2-Server Push. [http://chimera.labs.oreilly.com/books/1230000000545/ch12.html#\\_flow\\_control](http://chimera.labs.oreilly.com/books/1230000000545/ch12.html#_flow_control). – [Online; aufgerufen am 07.08.2015]
- [Spe] Speed Is A Killer – Why Decreasing Page Load Time Can Drastically Increase Conversions. <https://blog.kissmetrics.com/speed-is-a-killer/>. – [Online; aufgerufen am 25.08.2015]
- [Sup] HTTP/2-Browser Support. [https://en.wikipedia.org/wiki/HTTP/2#Browser\\_support](https://en.wikipedia.org/wiki/HTTP/2#Browser_support). – [Online; aufgerufen am 11.08.2015]
- [t3n] Das Web wird fetter: Durchschnittliche Webseite legte 2014 um 15 Prozent zu. <http://t3n.de/news/web-css-html-2014-586617/>. – [Online; aufgerufen am 29.08.2015]
- [Wil99] WILDE, Erik: *World Wide Web*. Springer, 1999

## Anhang

### A Abbildungsverzeichnis

Abb. 1	Dreiwege-Handshake . . . . .	4
Abb. 2	Head of Line Blocking . . . . .	15
Abb. 3	parallele Verbindungen . . . . .	16
Abb. 4	Text gegen Frame . . . . .	17
Abb. 5	Aufbau eines Frames . . . . .	19
Abb. 6	Multiplexing . . . . .	21
Abb. 7	Server Push . . . . .	23
Abb. 8	Header Vergleich . . . . .	26
Abb. 9	Testumgebung . . . . .	31
Abb. 10	Umschalten zwischen Protokollversionen . . . . .	36
Abb. 11	Test 1, Testreihe 1 . . . . .	37
Abb. 12	Test 2, Testreihe 1 . . . . .	38
Abb. 13	Test 3, Testreihe 1 . . . . .	39
Abb. 14	Test 4, Testreihe 1 . . . . .	40
Abb. 15	Test 1, Testreihe 2 . . . . .	41
Abb. 16	Test 2, Testreihe 2 . . . . .	42
Abb. 17	Test 3, Testreihe 2 . . . . .	43
Abb. 18	Test 4, Testreihe 2 . . . . .	44
Abb. 19	Test 1, Testreihe 3 . . . . .	45
Abb. 20	Test 2, Testreihe 3 . . . . .	46
Abb. 21	Test 3, Testreihe 3 . . . . .	47
Abb. 22	Test 4, Testreihe 3 . . . . .	48
Abb. 23	Test 1, Testreihe 4 . . . . .	50
Abb. 24	Test 2, Testreihe 4 . . . . .	51
Abb. 25	Test 3, Testreihe 4 . . . . .	52
Abb. 26	Test 4, Testreihe 4 . . . . .	53
Abb. 27	Auswertung 1, Testreihe 1 . . . . .	IX
Abb. 28	Auswertung 2, Testreihe 1 . . . . .	IX
Abb. 29	Auswertung 3, Testreihe 1 . . . . .	X
Abb. 30	Auswertung 4, Testreihe 1 . . . . .	X
Abb. 31	Auswertung 1, Testreihe 2 . . . . .	XI
Abb. 32	Auswertung 2, Testreihe 2 . . . . .	XI
Abb. 33	Auswertung 3, Testreihe 2 . . . . .	XII
Abb. 34	Auswertung 4, Testreihe 2 . . . . .	XII

Abb. 35	Auswertung 1, Testreihe 3	XIII
Abb. 36	Auswertung 2, Testreihe 3	XIII
Abb. 37	Auswertung 3, Testreihe 3	XIV
Abb. 38	Auswertung 4, Testreihe 3	XIV
Abb. 39	Auswertung 1, Testreihe 4	XV
Abb. 40	Auswertung 2, Testreihe 4	XV
Abb. 41	Auswertung 3, Testreihe 4	XVI
Abb. 42	Auswertung 4, Testreihe 4	XVI

## B Tabellenverzeichnis

Tab. 1	häufige Statuscodes	8
--------	---------------------	---

## C Listingverzeichnis

Lst. 1	Verbindungsaufbau HTTP/0.9	4
Lst. 2	Anfrage HTTP/0.9	5
Lst. 3	Antwort HTTP/0.9	5
Lst. 4	Anfrage HTTP/1	7
Lst. 5	Anfrage HTTP/1.1	10
Lst. 6	Caddyfile	30

## D Messergebnisse

### Testreihe 1

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
0,095	0,325	0,184	0,364
0,1	0,324	0,216	0,331
0,89	0,358	0,17	0,359
0,109	0,357	0,21	0,371
0,127	0,342	0,242	0,382
0,116	0,325	0,225	0,363
0,088	0,341	0,21	0,356
0,078	0,341	0,189	0,343
0,108	0,439	0,226	0,344
0,106	0,339	0,228	0,372
0,1817	0,3491	0,21	0,3585

Abbildung 27: Auswertung 1, Testreihe 1

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
0,418	0,505	0,273	0,561
0,497	0,53	0,266	0,488
0,304	0,5	0,228	0,474
0,272	0,473	0,277	0,492
0,304	0,494	0,248	0,511
0,239	0,453	0,246	0,465
0,234	0,46	0,301	0,545
0,285	0,48	0,288	0,504
0,237	0,498	0,299	0,518
0,24	0,502	0,296	0,505
0,303	0,4895	0,2722	0,5063

Abbildung 28: Auswertung 2, Testreihe 1

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
0,179	1	0,156	1,03
0,183	0,856	0,151	0,849
0,217	0,881	0,151	0,857
0,209	0,886	0,156	0,889
0,192	0,847	0,151	0,896
0,265	0,998	0,142	0,854
0,217	0,916	0,166	1,01
0,202	0,885	0,161	0,879
0,188	1,17	0,16	0,96
0,192	0,865	0,157	0,855
0,2044	0,9304	0,1551	0,9079

Abbildung 29: Auswertung 3, Testreihe 1

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
0,354	1,53	0,284	1,58
0,357	1,48	0,249	1,45
0,327	1,43	0,266	1,5
0,344	1,73	0,265	1,51
0,383	1,59	0,264	1,73
0,396	1,8	0,247	1,41
0,367	1,47	0,269	1,5
0,378	1,56	0,266	1,54
0,359	1,53	0,272	1,58
0,396	1,46	0,269	1,53
0,3661	1,558	0,2651	1,533

Abbildung 30: Auswertung 4, Testreihe 1

## Testreihe 2

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
0,159	0,414	0,2	0,502
0,163	0,432	0,174	0,418
0,159	0,386	0,185	0,397
0,184	0,38	0,237	0,404
0,159	0,348	0,176	0,423
0,158	0,39	0,183	0,428
0,175	0,426	0,183	0,389
0,167	0,689	0,179	0,401
0,163	0,432	0,189	0,379
0,176	0,422	0,179	0,399
0,1663	0,4319	0,1885	0,414

Abbildung 31: Auswertung 1, Testreihe 2

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
0,383	0,902	0,622	0,825
0,615	0,826	0,352	1,02
0,387	0,903	0,353	0,655
0,443	0,86	0,351	0,638
0,346	0,622	0,353	0,629
0,333	0,618	0,353	0,666
0,333	0,556	0,383	0,591
0,331	0,606	0,378	0,669
0,632	0,79	0,37	0,656
0,326	0,639	0,371	0,631
0,4129	0,7322	0,3886	0,698

Abbildung 32: Auswertung 2, Testreihe 2



HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
0,973	1,83	0,968	1,93
0,924	1,87	0,854	1,36
0,901	1,75	0,926	1,93
1,24	1,95	0,933	2,04
0,968	1,62	0,921	1,6
0,903	1,82	0,928	1,79
0,907	1,91	0,957	1,86
0,963	1,84	0,92	1,73
0,996	1,93	0,932	1,71
0,972	1,72	1,06	2,07
0,9747	1,824	0,9399	1,802

Abbildung 33: Auswertung 3, Testreihe 2

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
1,94	3,1	1,92	2,84
1,92	3,03	1,91	3,06
1,92	3,11	1,91	2,8
1,93	3,13	1,92	2,72
1,93	3,06	1,92	2,82
1,94	3,13	1,92	3,23
1,94	3,11	1,93	2,79
1,93	3,15	1,89	2,68
1,91	3,05	1,93	3,39
1,94	3,17	1,92	3,62
1,93	3,104	1,917	2,995

Abbildung 34: Auswertung 4, Testreihe 2

## Testreihe 3

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
10,43	10,49	10,34	10,51
10,42	10,6	10,29	10,5
10,25	10,48	10,71	10,87
10,45	10,51	10,3	10,46
10,32	10,51	10,27	10,44
10,53	10,7	11,34	11,52
10,7	10,93	10,29	10,48
10,43	10,62	10,28	10,48
10,6	10,8	10,3	10,52
10,27	10,46	10,29	10,54
10,44	10,61	10,441	10,632

Abbildung 35: Auswertung 1, Testreihe 3

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
2,8	5,1	2,13	5,07
2,44	5,58	2,16	4,01
2,55	4,64	2,16	5,3
2,35	5,32	2,17	5,22
2,34	4,73	2,3	5,32
2,37	2,23	2,19	4,23
2,36	4,83	2,14	5,16
2,41	5,39	2,16	4,06
2,49	5,2	2,22	3,97
2,28	5,16	2,2	5,32
2,439	4,818	2,183	4,766

Abbildung 36: Auswertung 2, Testreihe 3

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
21,8	23,52	27,8	28,2
19,47	20,12	28,77	29,65
19,93	20,58	23,04	23,52
20,8	21,5	25,84	26,55
19,47	20,64	25,58	26,47
19,21	19,68	25,97	26
19,29	19,94	24,14	24,92
20,08	20,77	26,85	27,5
21,3	21,77	27,67	28,33
20,15	20,84	24,11	24,58
20,15	20,936	25,977	26,572

Abbildung 37: Auswertung 3, Testreihe 3

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
10,56	10,96	10,03	10,24
11,75	11,97	11,47	11,52
9,69	10,12	13,02	13,75
11,53	11,66	10,28	10,72
9,72	9,88	12,22	12,28
9,52	10,02	13,1	13,3
12,86	13,19	12,61	12,66
11,66	11,84	10,84	11,47
9,67	9,82	11,98	12,86
9,45	9,89	13,1	13,3
10,641	10,935	11,865	12,21

Abbildung 38: Auswertung 4, Testreihe 3

## Testreihe 4

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
1,3	1,25	1,26	1,21
1,29	1,24	1,26	1,22
1,27	1,22	1,34	1,3
1,43	1,37	1,28	1,23
1,25	1,2	1,21	1,17
1,33	1,26	1,27	1,22
1,27	1,22	1,39	1,33
1,41	1,35	1,33	1,27
1,24	1,19	1,34	1,29
1,37	1,31	1,36	1,31
1,316	1,261	1,304	1,255

Abbildung 39: Auswertung 1, Testreihe 4

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
2,73	3,43	2,51	3,98
2,21	3,15	2,25	3,27
2,22	3,23	1,65	3,33
1,59	3,91	2,39	3,52
2,51	3,69	2,29	3,54
2,63	3,64	2,33	3,24
2,65	3,75	2,55	3,9
2,36	3,96	2,37	3,39
2,52	3,68	2,63	3,71
2,13	3,75	1,33	3,58
2,355	3,619	2,23	3,546

Abbildung 40: Auswertung 2, Testreihe 4

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
5,67	2,65	3,32	2,49
3,95	2,38	2,87	2,31
5,16	2,07	3,27	2,32
4,55	2,1	4,17	2,27
4,56	2,59	2,8	2,11
4,35	2,26	3,48	3,03
4,93	2,91	3,01	2,29
5,18	2,03	4,09	2,47
3,91	2,12	4,08	2,28
5,17	2,28	3,2	2,2
4,743	2,385	3,429	2,343

Abbildung 41: Auswertung 3, Testreihe 4

HTTP/1.1		HTTP/2	
HTTP/1.1 net	HTTP/1.1 brut	HTTP/2 net	HTTP/2 brut
1,37	3,76	1,74	2,97
1,3	2,38	1,42	2,69
2,28	2,34	1,38	2,83
1,38	2,41	1,37	2,46
1,41	2,6	1,53	2,67
1,47	2,52	1,47	3,66
1,35	2,43	1,48	2,65
1,35	2,41	1,51	2,63
1,42	2,53	1,48	2,62
1,33	2,39	1,72	2,89
1,466	2,577	1,51	2,807

Abbildung 42: Auswertung 4, Testreihe 4