Fernfachhochschule Schweiz
Studiengang BSc INF 2006

Optimieren der Ladezeiten von Webseiten mit Googles Apache-Modulen

Bachelor-Thesis im Studiengang BSc INF 2006

von

Oliver Gugger

Eingereicht bei:

Prof. Dr. Martin Sutter
Departement Informatik
Departementsleiter

Referent:

Prof. Frank Koch
Fachbereichsleiter
Informationssysteme

Bern, 11. Februar 2013

Zusammenfassung

Diese Arbeit untersucht die zwei Technologien SPDY und PageSpeed, die beide von Google vorgeschlagen und entwickelt wurden. SPDY ist ein neues, experimentelles Netzwerk-Protokoll das versucht, einige der Nachteile des ins Alter gekommenen HTTP-Protokolls zu beseitigen. PageSpeed ist eine Sammlung von Regeln und Anweisungen, die für Web-Entwickler geschaffen wurde, die die Ladezeiten ihrer Webseiten verringern wollen.

Für beide Technologien hat Google Module für den Apache-Webserver entwickelt und veröffentlicht: mod_spdy und mod_pagespeed.

Mit der Hilfe einer Test-Webseite wird der Effekt des Einsatzes dieser Module anhand der Ladezeit der Seite in verschiedenen Szenarien mit unterschiedlichen Bandbreiten gemessen.

Die Resultate dieser Tests zeigen, dass in gewissen Fällen eine Reduktion der Ladezeit um bis zu 80% erreicht werden kann.

Die Arbeit kommt zum Schluss, dass beide Technologien einen genaueren Blick wert sind und dass sie für den produktiven Einsatz reif sind.

Abstract

This paper analyzes the two technologies SPDY and PageSpeed that were both proposed and developed by Google. SPDY is a new, experimental network protocol that tries to overcome some of the disadvantages of the aging HTTP protocol. PageSpeed is a set of rules and instructions created for web developers that helps them to optimize their web sites for page load time.

For both of these technologies Google has created and published modules for the Apache web server: mod_spdy and mod_pagespeed.

With the help of a test web page the effect of the usage of these modules on the page load time are then measured in different scenarios with different bandwidths.

The results of these tests show that in some cases there is a reduction in page load time of up to 80%.

The paper concludes that both technologies are worth having a look at and are ready to be used in a production environment.

Inhaltsverzeichnis

Abkürzungen

Dt.	Deutsch, zu Deutsch
HTTP	HyperText Transfer Protokoll (dt.: Netzwerk-Protokoll für den Transport von Hypertext/HTML)
HTTPS	HTTP-Protokoll über SSL/TLS
HTML	HyperText Markup Language (dt.: Hypertext-Auszeichnungssprache)
CSS	Cascading Style Sheet (dt.: Geschachtelte Gestaltungsvorlagen für Webseiten)
SPDY	SPeeDY (dt.: schnell, experimentelles Netzwerkprotokoll vorgeschlagen von Google)
SSL	Secure Socket Layer (Protokoll für verschlüsselte Verbindungen)
TLS	Transport Layer Security (Nachfolgeprotokoll von SSL)
NPN	Next Protocol Negotiation (dt.: Verhandlung über nächstes Protokoll, Erweiterung für das TLS-Protokoll)
TCP	Transmission Control Protocol (Transportprotokoll im Internet)

1 Einleitung

1.1 Was bedeutet Optimierung von Ladezeiten?

Webseiten haben heutzutage einen sehr hohen Stellenwert. Immer mehr Informationen werden über das Internet angeboten und genutzt. Kommunikation erfolgt über soziale Netzwerke, Wissen wird von Wikipedia abgerufen, Fernsehsendungen werden in Form von Podcasts konsumiert, Geld wird per Online-Banking überwiesen und Zeitungen werden online gelesen.
Um dieser Nachfrage gerecht zu werden, ist die Bandbreite der Internetverbindung, mit der ein Privatkunde versorgt werden kann, stetig gewachsen. Es ist heute keine Seltenheit mehr, dass ein Privatbenutzer zuhause eine Download-Geschwindigkeit von 100 MBit/s zur Verfügung hat.

Doch die Erwartung der Nutzer, dass bei doppelter Bandbreite eine Webseite auch doppelt so schnell (also in der Hälfte der Zeit) geladen ist, kann nur in wenigen Fällen erfüllt werden. Das vor über 20 Jahren [1] entwickelte Protokoll HTTP, welches heute immer noch zentraler Dreh- und Angelpunkt der meisten Webseiten ist, hat einige Einschränkungen, die diese lineare Abhängigkeit zwischen Bandbreite und Ladezeit verhindern.

Studien haben jedoch gezeigt, dass es einen Zusammenhang gibt zwischen der Ladezeit eines Online-Shops und dessen Verkaufszahlen [2 p. 103]. Ein optimierter Internetauftritt kann also durchaus entscheidend sein über Erfolg und Nichterfolg.

Die Optimierung der Ladezeiten von Webseiten im Kontext dieser Arbeit bedeutet also: Welche Mittel und Methoden können eingesetzt werden, um die Einschränkungen von HTTP zu umgehen oder abzuschwächen? Wie kann die Zeit, die ein Browser benötigt um eine Webseite komplett herunterzuladen und anzuzeigen, so klein wie möglich gehalten werden?

1.2 Ausgangslage

Moderne und optisch ansprechende Internetseiten bestehen aus vielen kleinen Dateien (CSS für die Gestaltung, JavaScript für das Verhalten, Bilder, Videos und weitere Inhalte), die einzeln vom Server zum Browser übertragen werden müssen. Beispielsweise sind es bei der neu gestalteten Seite vom Schweizerischen Radio und Fernsehen (www.srf.ch) über 180 Dateien auf der Startseite.

Das HTTP-Protokoll gibt vor, dass für jede Datei einzeln eine Anfrage gestellt und deren Antwort abgewartet wird. Das bedeutet, dass für jede Datei mindestens ein Paket hin und danach eins zurück gesendet werden muss. Diese so genannte Round Trip Time (Rundreise-Zeit) ist aber an physikalische Bedingungen (wie der Lichtgeschwindigkeit) geknüpft und kann auch mit der Erhöhung der Bandbreite nicht verkürzt werden.
Es kann beobachtet werden, dass ab einer gewissen minimalen Bandbreite (beispielsweise 5 MBit/s Download-Geschwindigkeit) die Round Trip Time einen deutlich grösseren Einfluss auf die Ladezeit einer Seite hat als eine noch höhere Bandbreite [3].
Diese Problematik kann teilweise entschärft werden, in dem der Browser mehrere Verbindungen gleichzeitig zum Server öffnet. Dies kostet jedoch Ressourcen auf beiden Seiten, vor allem aber beim Server, der unter Umständen tausende Benutzer gleichzeitig bedienen muss.
Es scheint also, als sei HTTP in seiner heutigen Form an seine Grenzen gelangt.
Aus diesem Grund hat sich Google, als einer der grössten online-Dienstleister, im Jahre 2009 dazu entschlossen, das Internet schneller zu machen [4].
Die Initiative „Let's make the web faster" hat unter Anderem zur Entwicklung von zwei Technologien und den dafür implementierten Apache-Modulen geführt: SPDY und Pagespeed.
Beide Technologien haben das gleiche Ziel: Webseiten schneller zu laden. Wie das Ziel erreicht werden soll ist aber sehr unterschiedlich. Bei SPDY handelt es sich um ein neues, experimentelles Netzwerkprotokoll, während PageSpeed mit dem heute eingesetzten HTTP funktioniert und sich vor allem dem Inhalt von Webseiten annimmt.
Da der Direktvergleich solch unterschiedlicher Technologien nicht einfach ist, wird in dieser Arbeit der praktische Nutzen (also der Geschwindigkeitsvorteil) anhand einer Test-Webseite verglichen.

1.3 Fragestellungen

Diese Arbeit hat das Ziel, die beiden Technologien SPDY und PageSpeed im Detail vorzustellen und die Apache-Module mod_spdy und mod_pagespeed einem Praxistest zu unterziehen. Im Test sollen vor allem folgende Fragen beantwortet werden:

- Welche Reduktion der Ladezeiten kann mit dem Einsatz der Technologien erreicht werden?
- Können beide Apache-Module gleichzeitig betrieben werden?
- Ist der Einsatz von SPDY in einer produktiven Umgebung zu diesem Zeitpunkt schon sinnvoll?

- Ist der Einsatz von PageSpeed empfehlenswert oder kann ein Webseitenbetreiber selbst Verbesserungen vornehmen?

1.4 Relevanz der Arbeit

Die stetig steigende Wichtigkeit des Internets fördert das Bestreben, neue Methoden zu entwickeln, die dessen Effizienz verbessern. Die Geschichte des Internets hat gezeigt, dass sich meistens die Technologien durchsetzen, die einfach in der Handhabung sind und einen grossen Nutzen bringen (Beispiel HTTP). Ob sich eine neu entwickelte Technologie durchsetzen kann, hängt also davon ab, ob sie in den Augen des Anwenders sinnvoll ist. Diese Arbeit versucht, die Nützlichkeit von zwei relativ neuen Technologien objektiv zu ermitteln.

Ein Einsatz von SPDY und PageSpeed ist aber nicht nur für die Benutzer interessant. Neue Herausforderungen wie beispielsweise der massiv steigende mobile Datenverkehr kann auch Betreiber von Netzwerkinfrastrukturen dazu bewegen, sich für neue Technologien einzusetzen, die Verbindungen effizienter nutzen und insgesamt weniger Daten übertragen.

1.5 Abgrenzung zu anderen Themengebieten

Neben den in dieser Arbeit untersuchten Methoden gibt es viele weitere, die die Ladezeit von Webseiten beeinflussen können. Diese werden aber nicht behandelt:

- Änderungen an der Netzwerk-Infrastruktur (Verbessern des Routing von Paketen, Einsatz effizienterer Netzwerkgeräte, etc.)
- Anpassungen am Programmablauf dynamischer Webseiten (Vermeiden von langsamen Datenbankabfragen, Einsatz von effizienteren Programmbibliotheken, etc.)
- Einsatz von Content Delivery Networks (Verteilen von Ressourcen über die ganze Welt, damit sie geographisch „nahe“ beim Benutzer sind)

2 SPDY - das neue Protokoll von Google

Dieses Kapitel behandelt das Netzwerk-Protokoll SPDY (kurz für *SPeeDY*, dt.: *schnell*). Es wird vorausgesetzt, dass der Leser ein grundlegendes Verständnis der Analyse von Netzwerk-Protokollen auf Paket-Ebene besitzt und insbesondere mit den Protokollen HTTP und SSL/TLS vertraut ist.

2.1 Motivation für die Entstehung

Die erste Version des Vorschlages für ein neues Protokoll mit dem Namen SPDY wurde 2009 von zwei Google-Mitarbeitern publiziert [5]. SPDY hat das Ziel, einige Nachteile des in die Jahre gekommenen HTTP-Protokolls zu umgehen.
Dabei wurden vor allem die folgenden Nachteile von HTTP adressiert [6]:

- Pro Verbindung kann immer nur eine Anfrage gleichzeitig an den Server gesendet werden. Erst wenn der Server geantwortet hat, kann eine neue Anfrage gestellt werden.
- Der Browser muss jede Ressource einzeln anfragen. Obwohl der Server möglicherweise wüsste, dass für eine Seite weitere Dateien relevant sind (zum Beispiel CSS- oder JavaScript-Dateien) gibt es keinen Mechanismus, wie er diese gleich mit der Seite ausliefern könnte.
- Anfrage- und Antwort-Header werden immer im Klartext und daher nicht komprimiert gesendet.
- Gewisse Header-Felder sind bei jeder Anfrage gleich (z.B. der User-Agent), müssen aber trotzdem immer mit gesendet werden.
- Die Komprimierung der Nutzdaten wird von HTTP nicht vorgeschrieben, obwohl die übertragene Datenmenge dadurch deutlich reduziert werden könnte.

Mit SPDY haben sich die Entwickler ein ehrgeiziges Ziel gesetzt: Ladezeiten von Webseiten sollen bis auf die Hälfte reduziert werden. Des Weiteren sollte die Installation und Verwendung möglichst einfach gestaltet werden. Weder an der Netzwerkinfrastruktur noch am Inhalt der Webseiten sollten Änderungen notwendig sein [6].
Ob diese Ziele erreicht werden konnten, soll unter Anderem in dieser Arbeit untersucht werden.

2.2 Die Technologie im Überblick

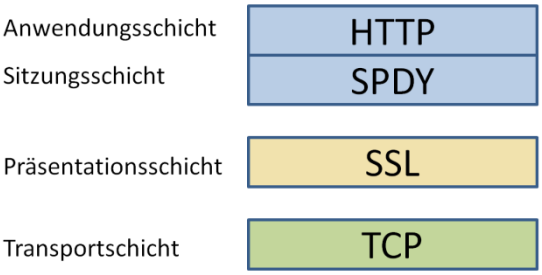


Abbildung 1: SPDY im OSI-Modell

SPDY wurde als zusätzliche Schicht zwischen der Präsentations- (SSL) und der Applikations-Schicht (HTTP) implementiert [6]. Dies bedeutet, dass die darunterliegende Transport-Schicht (TCP) nicht angepasst oder ausgetauscht werden muss. Dies ist eine wichtige Voraussetzung für eine rasche Verbreitung von SPDY im Internet, da TCP allgegenwärtig ist.

Ebenfalls bedeutet dies, dass HTTP nicht abgelöst sondern vielmehr erweitert wird, was die Implementierung in bestehender Client-Software vereinfacht, da die grundlegende Semantik von HTTP bestehen bleibt [6].

2.2.1 Multiplexing der Verbindungen

Die wichtigste Änderung gegenüber HTTP ist das so genannte Multiplexing (die Bündelung) von Verbindungen.

Während pro HTTP-Verbindung jeweils nur eine Anfrage und Antwort gleichzeitig gesendet werden können, wurde für SPDY das Konzept der Streams (dt. *Ströme* oder *Datenströme*) eingeführt. Es können in beide Richtungen mehrere Frames (dt. *Rahmen*) hintereinander gesendet werden, die jeweils einen Stream enthalten, gekennzeichnet mit der Stream-ID, welche aus einer beliebigen aber während einer Session eindeutigen Nummer besteht.

Durch diese Kennzeichnung der Rahmen ist es nun auch nicht mehr notwendig, eine gewisse Reihenfolge bei den Anfragen und Antworten einzuhalten. Da beide Seiten bei jedem Rahmen wissen, um wel-

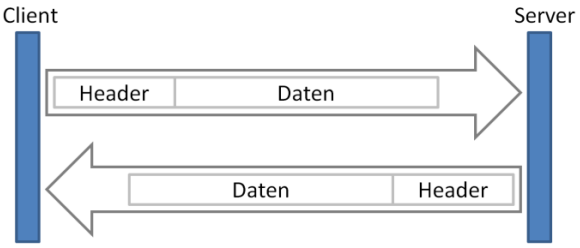


Abbildung 2: Schematische Darstellung einer HTTP-Session

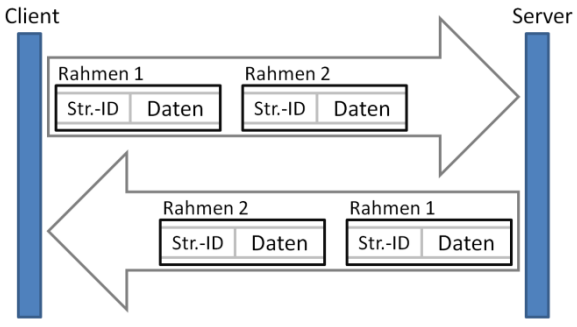


Abbildung 3: Schematische Darstellung einer SPDY-Session

chen Stream es sich handelt, können theoretisch beide Seiten gleichzeitig senden. Dieser Fakt alleine bewirkt bereits eine deutlich bessere Ausnutzung der zur Verfügung stehenden Bandbreite, da nicht eine Seite jeweils auf die Antwort der anderen warten muss.

2.3 Umsetzung als Apache-Modul

Um die Verbreitung des Protokolls voranzutreiben, haben Mitarbeiter von Google für eine Reihe von beliebten Servern und Programmierbibliotheken [6] SPDY-Module implementiert. Darunter ist auch der Apache Webserver, welcher für diese Arbeit verwendet wurde. In diesem Abschnitt wird aufgezeigt, wie der Apache mit dem SPDY-Modul installiert und konfiguriert wird.

2.3.1 Installation und Konfiguration

Als Server-Umgebung für die Untersuchungen und die späteren Tests wurde ein Linux-Server in Amazons Elastic Compute Cloud (EC2) erstellt, mit folgenden Leistungsmerkmalen [7]:

Merkmal	*Wert*
Instanz-Typ	Linux Micro Instance
Linux-Distribution	Ubuntu 12.04 LTS
Kernel-Version	3.2.0-34-virtual
Prozessor-Architektur	64 Bit, virtualisiert
Arbeitsspeicher	613 MB RAM

Tabelle 1: Leistungsmerkmale der Server-Umgebung

Die Installation des SPDY-Modules gestaltet sich vergleichsweise einfach, da Google vorkompilierte Ubuntu-Pakete zur Verfügung stellt [8]:

```
## Apache installieren
$ sudo apt-get install apache2

## SSL aktivieren
$ sudo a2enmod ssl
```

```
## Vorkompiliertes SPDY-Modul von Google herunterladen
$ cd /tmp
$ wget https://dl-ssl.google.com/dl/linux/direct/mod-spdy-beta_current↵
_amd64.deb

## Modul installieren
$ sudo dpkg -i mod-spdy-beta_current_amd64.deb

## Apache neu starten
$ sudo service apache2 restart
```

Nach diesen Schritten ist das SPDY-Modul aktiviert und es werden standardmässig alle (HTTPS-)Seiten des Servers darüber ausgeliefert, sofern der Browser dies unterstützt. Überprüft werden kann dies mit einer aktuellen Version von Chrome [8]:

1. Chrome öffnen und in der Adresszeile die Adresse `chrome://net-internals/#spdy` eingeben
2. In einem zweiten Tab die zu prüfende Webseite aufrufen, beispielsweise `https://bachelor.aws.guggero.org`
3. Eine allfällig angezeigte Zertifikatswarnung akzeptieren
4. Nach dem Laden der Seite sollte dann im ersten Tab eine SPDY-Session mit der eingegebenen Adresse zu sehen sein:

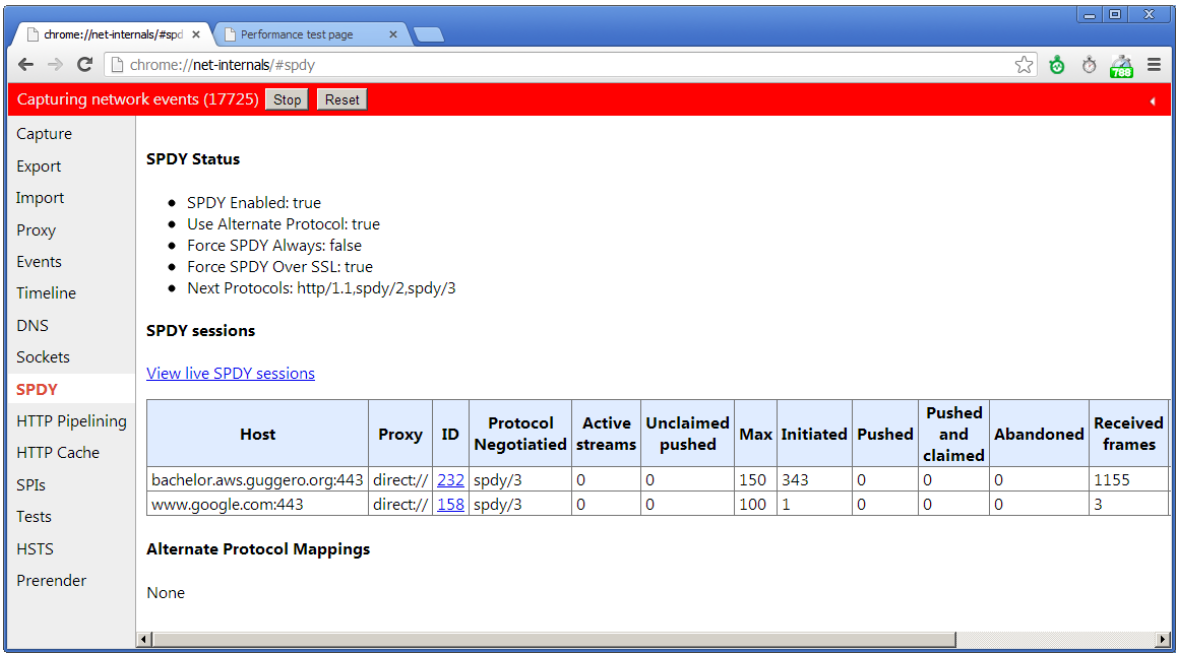


Abbildung 4: Eine SPDY-Session in Chromes Diagnosewerkzeug

2.3.2 Funktionsweise

Damit der Apache-Server auch weiss, ob ein Client bzw. ein Browser SPDY spricht, klinkt sich das SPDY-Modul in den so genannten Next Protocol Negotiation-Mechanismus des Protokolls TLS ein. Dieser Mechanismus erlaubt es, während dem Aufbau einer sicheren Verbindung per TLS (dem so genannten *SSL handshake*), das nächste zu sprechende Protokoll auszuhandeln. Dazu fügt sich mod_spdy zu der Liste der unterstützten Protokolle hinzu und der Client kann dies auswählen, sofern er es auch unterstützt. Ist die sichere Verbindung so zustande gekommen, erwarten ab diesem Zeitpunkt beide Seiten, dass der Inhalt der verschlüsselten Pakete diesem Protokoll entspricht [9].

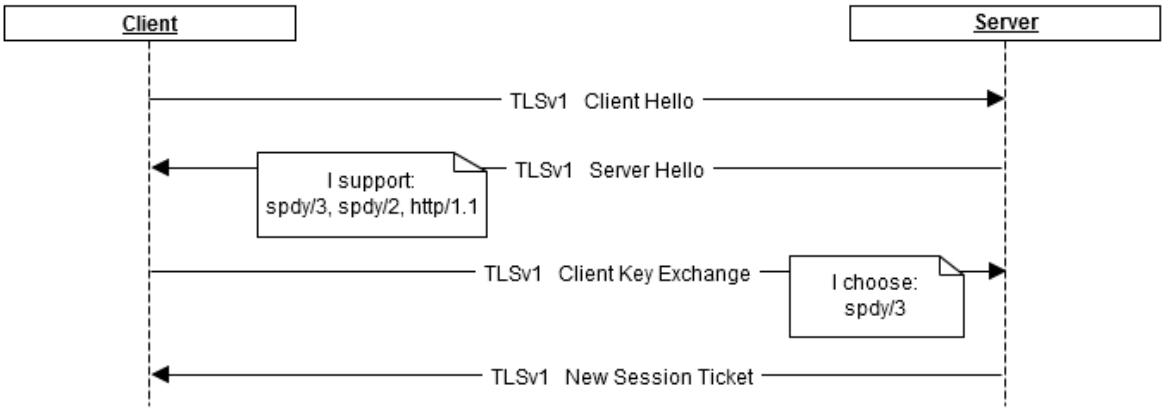


Abbildung 5: Ablauf eines SSL-Handshakes mit NPN und mod_spdy

2.4 Technische Beschreibung

Dieses Kapitel hat nicht das Ziel, die technische Beschreibung der SPDY-Spezifikation [10] im Detail wiederzugeben. Vielmehr wird versucht, anhand eines echten Beispiels einer SPDY-Session die wichtigsten Merkmale des Protokolls aufzuzeigen.

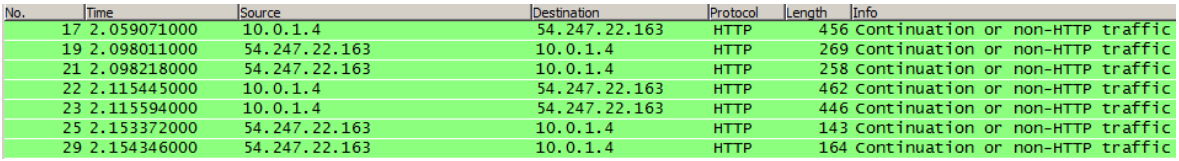
Dafür wurde eine kleine Beispielseite mit zwei verlinkten Ressourcen (eine CSS- und eine JavaScript-Datei) erstellt und diese mit aktiviertem mod_spdy über Firefox aufgerufen. Die Daten wurden mit Wireshark (http://www.wireshark.org/) aufgezeichnet und dann mit Hilfe eines Hex-Editors aufbereitet.

Wie mit SSL/TLS geschützte Verbindungen aufgezeichnet und entschlüsselt werden können, ist in Anhang A beschrieben.

2.4.1 Das Protokoll im Detail

Zur Veranschaulichung einer typischen SPDY-Session und die darin ausgetauschten Datenpakete wurde die Test-Seite https://bachelor.aws.guggero.org/small.html erstellt

und mit Firefox 17.0.1 aufgerufen. Dank der in Anhang A beschriebenen Vorgehensweise konnten die Pakete mit Wireshark aufgezeichnet und entschlüsselt werden.



No.	Time	Source	Destination	Protocol	Length	Info
17	2.059071000	10.0.1.4	54.247.22.163	HTTP	456	Continuation or non-HTTP traffic
19	2.098011000	54.247.22.163	10.0.1.4	HTTP	269	Continuation or non-HTTP traffic
21	2.098218000	54.247.22.163	10.0.1.4	HTTP	258	Continuation or non-HTTP traffic
22	2.115445000	10.0.1.4	54.247.22.163	HTTP	462	Continuation or non-HTTP traffic
23	2.115594000	10.0.1.4	54.247.22.163	HTTP	446	Continuation or non-HTTP traffic
25	2.153372000	54.247.22.163	10.0.1.4	HTTP	143	Continuation or non-HTTP traffic
29	2.154346000	54.247.22.163	10.0.1.4	HTTP	164	Continuation or non-HTTP traffic

Abbildung 6: Beispiel aufgezeichneter SPDY-Pakete im Wireshark

Wie in der oben gezeigten Abbildung ersichtlich ist, erkennt Wireshark die Pakete nicht als SPDY-Daten („non-HTTP traffic“). Der Grund dafür ist, dass SPDY so neu ist, dass es noch keine offizielle Unterstützung dafür gibt [11] (Wireshark 1.8.4, November 2012). Das inoffizielle Plugin „spdyshark“ [12] konnte weder unter Windows noch unter Linux erfolgreich zum Funktionieren gebracht werden. Daher hat der Autor entschieden, die Analyse mit den Binärdaten, die von Wireshark exportiert werden können, selbst durchzuführen.

Für das weitere Verständnis müssen zuerst folgende Begriffe eingeführt werden, die im Zusammenhang mit SPDY verwendet werden [10]:

Begriff	*Bedeutung*
Session	Entspricht einer SSL-Verbindung und steht daher für alle Daten, die zwischen Client und Server für den Aufbau einer kompletten Webseite ausgetauscht werden.
Stream	Ein virtueller Kanal, der für alle ausgetauschten Daten einer einzelnen Server-Ressource steht. Übertragen aus dem HTTP-Protokoll entspricht ein Stream einer einzelnen GET/POST-Anfrage und die Antwort dazu.
Frame	Ein Rahmen ist die kleinste Einheit jedes Protokolls. Bei SPDY wird unterschieden zwischen Control Frames und Data Frames.
Control Frame	Ein Rahmen, der Meta-Informationen zwischen Client und Server überträgt. Ein Kontroll-Rahmen bezieht sich meistens auf einen einzelnen Stream, beispielsweise um einen solchen zu starten oder auf einen bestehenden Stream zu antworten.
Data Frame	Ein Rahmen, der lediglich Nutzdaten enthält. Solch ein Daten-Rahmen bezieht sich immer auf einen einzelnen Stream.
Paket	Alle Daten, die in einem Stück (also in einem TCP-Paket) in eine Richtung übermittelt werden konnten. Kann mehrere Rahmen enthalten.

Tabelle 2: Begriffe im Zusammenhang mit SPDY

2.4.2 Analyse der aufgezeichneten Pakete

Die beschriebene Vorgehensweise dieses Abschnittes ähnelt der von [13 S. 49-69]. Für die Nachvollziehbarkeit der späteren Kapitel ist es aber in den Augen des Autors sinnvoll, hier ein eigenes Beispiel mit der aktuellen Version SPDY 3 und mod_spdy auf der Server-Seite zu zeigen.

Die aufgezeichneten Daten bestehen aus insgesamt 7 TCP-Paketen mit SPDY-Inhalt, wovon drei zum Server und vier zurück zum Client gesendet wurden. Mit Hilfe der hexadezimal dargestellten Rohdaten und der Spezifikation [10] werden diese nun untersucht.

Aufbau eines Kontroll-Rahmens

```
+----------------------------------+
|C| Version(15bits) | Type(16bits) |
+----------------------------------+
| Flags (8)  |  Length (24 bits)   |
+----------------------------------+
|               Data               |
+----------------------------------+
```

Aufbau eines Daten-Rahmens

```
+----------------------------------+
|C|       Stream-ID (31bits)       |
+----------------------------------+
| Flags (8)  |  Length (24 bits)   |
+----------------------------------+
|               Data               |
+----------------------------------+
```

Ob es sich bei einem Rahmen um einen Kontroll- oder Daten-Rahmen handelt, kann also bereits im ersten Bit (oben markiert mit C) festgestellt werden. Ist dieses auf 1 gesetzt, ist es ein Kontroll-Rahmen, sonst ein Daten-Rahmen. Bei ersterem kann dann im 4. Byte festgestellt werden, um welchen Typen von Kontroll-Rahmen es sich handelt.

Paket 1

Das erste Paket ist die erste Anfrage des Browsers an den Server und zeigt gleich die Stärke des Protokolls: Es wurden nämlich zwei Kontroll-Rahmen gleichzeitig übertragen:

Rahmen 1:

```
0000000: 8003 0004  ....

0000004: 0000 000c  ....

0000008: 0000 0001  ....

000000c: 0000 0007  ....
0000010: 1000 0000  ....
```

```
+----------------------------------+
|1|   version    |         4       |
+----------------------------------+
| Flags (8)  |  Length (24 bits)   |
+----------------------------------+
|         Number of entries        |
+----------------------------------+
|          ID/Value Pairs          |
|             ...                  |
```

Es handelt sich um einen Kontroll-Rahmen des Typs 4 (SETTINGS) in dem der Browser dem Server mitteilt, dass er bis zu 256 MByte an Daten in einem Stream empfangen kann (ID = 0x7 SETTINGS_INITIAL_WINDOW_SIZE, Wert = 0x10000000).

Rahmen 2:

```
0000014: 8003 0001  ....
0000018: 0100 0171  ...q
000001c: 0000 0001  ....
0000020: 0000 0000  ....
0000024: 4000 783f  @.x?
0000028: e3c6 a7c2  ....
000002c: 0057 01a8  .W..
0000030: fe00 0000  ....
0000034: 0800 0000  ....
0000038: 073a 6d65  .:me
000003c: 7468 6f64  thod
```

```
+----------------------------------+
|1|    version    |        1       |
+----------------------------------+
|  Flags (8)  |  Length (24 bits)  |
+----------------------------------+
|X|           Stream-ID (31bits)   |
+----------------------------------+
|X| Associated-To-Stream-ID (31bits) |
+----------------------------------+
| Pri|Unused | Slot |              |
+-------------------+              |
| Compressed Name/Value pairs      |
|               ...                |
|               ...                |
|               ...                |
|               ...                |
|               ...                |
```

Der zweite Kontroll-Rahmen, den der Browser sendet, beinhaltet die eigentliche GET-Anfrage an den Browser. Mit dem Typ 1 (SYN_STREAM) teilt er dem Server mit, dass er den Stream mit der ID 1 eröffnet und sendet gleich darauf die komprimierten HTTP-Header-Felder der Anfrage im gleichen Rahmen. Das gesetzte FLAG_FIN (0x1) teilt dem Server mit, dass dies der letzte Rahmen zu dieser Anfrage ist.
Der ganze Rahmen ist 377 Bytes lange und wurde zur Bewahrung der Übersichtlichkeit gekürzt.

Paket 2

Der Server antwortet mit seinem ersten Paket. Dieses enthält wiederum zwei Kontroll-Rahmen.

Rahmen 1:

Da es sich wiederum um ein Kontroll-Rahmen des Typs 4 (SETTINGS) handelt, wurde auf die detaillierte Aufschlüsselung verzichtet. Der Server teilt dem Browser mit, dass er bis zu 150 Streams gleichzeitig bedienen kann
(ID = 0x4 SETTINGS_MAX_CONCURRENT_STREAMS, Wert = 0x96).

Rahmen 2:

```
0000014: 8003 0002  ....
0000018: 0000 00b6  ....
000001c: 0000 0001  ....
0000020: 38ea e3c6  8...
0000024: a7c2 02e5  ....
0000028: 4850 7ab4  HPz.
000002c: 8227 5266  .'Rf
0000030: 6022 0525  `".%
0000034: 4b2b ec49  K+.I
```

```
+-----------------------------------+
|1|    version    |         2       |
+-----------------------------------+
|  Flags (8)  |  Length (24 bits)   |
+-----------------------------------+
|X|           Stream-ID (31bits)    |
+-----------------------------------+
| Compressed Name/Value pairs       |
|               ...                 |
|               ...                 |
|               ...                 |
|               ...                 |
|               ...                 |
```

Dieser Kontroll-Rahmen des Typs 2 (SYN_REPLY) enthält nun die eigentliche Antwort des Servers oder präziser gesagt die Antwort-HTTP-Header-Felder zum Stream mit der ID 1. Der ganze Rahmen ist 190 Bytes lange und wurde gekürzt abgebildet.

Paket 3

Das dritte Paket enthält nun die eigentlichen Nutzdaten, also den angeforderten HTML-Code, in Form eines Daten-Rahmens.

Rahmen 1:

```
0000000: 0000 0001  ....
0000004: 0100 00ab  ....
0000008: 1f8b 0800  ....
000000c: 0000 0000  ....
0000010: 0003 358f  ..5.
0000014: d10d c230  ...0
```

```
+-----------------------------------+
|0|       Stream-ID (31bits)        |
+-----------------------------------+
| Flags (8)  |  Length (24 bits)    |
+-----------------------------------+
|       gzip Compressed Data        |
|               ...                 |
|               ...                 |
|               ...                 |
```

Mit diesem Daten-Rahmen werden insgesamt 171 (0xab) Bytes an komprimierten Nutzdaten für den Stream mit der ID 1 übertragen. Das FLAG_SYN (0x01) ist wiederum gesetzt, der Browser hat zu diesem Stream also keine Daten mehr zu erwarten.

Paket 4-7

Die restlichen Pakete enthalten keine neuen Typen von Frames. Um unnötige Redundanz zu vermeiden wurde daher auf die detaillierte Darstellung der restlichen Pakete verzichtet.

Grafische Darstellung der gesamten SPDY-Session

Das folgende Sequenzdiagramm zeigt die komplette SPDY-Session mit allen gesendeten Rahmen:

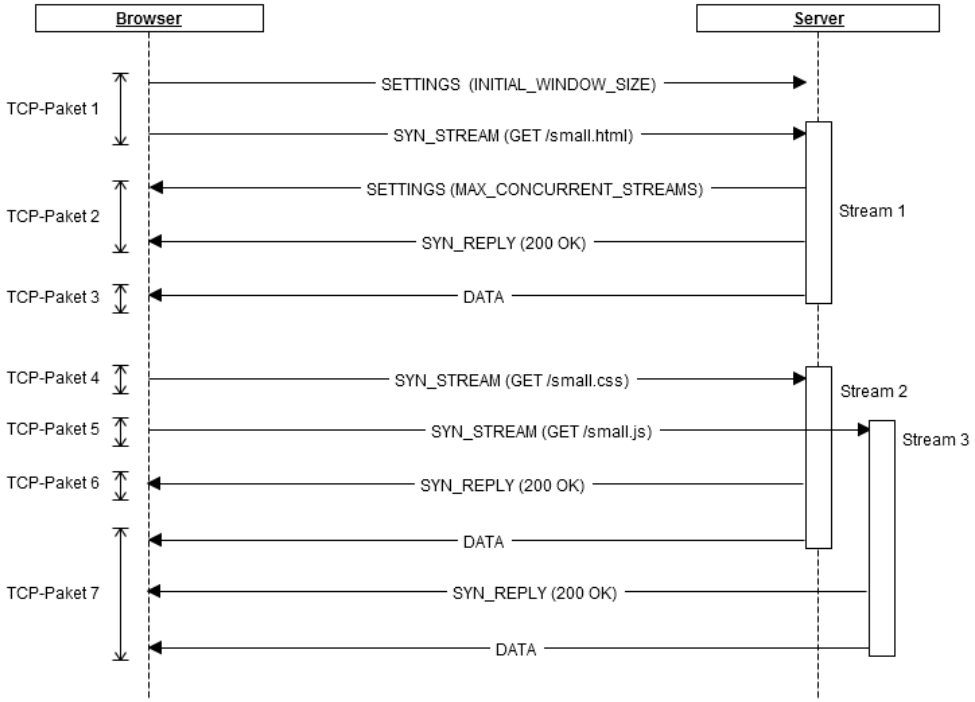


Abbildung 7: Sequenzdiagramm einer SPDY-Session

Bereits diese kleine Beispiel-Seite mit nur drei Dateien zeigt das Multiplexing sehr anschaulich. Nachdem der Browser die HTML-Seite erhalten und übersetzt hat, kann er die beiden zusätzlichen Dateien (Paket 4 und 5) über die gleiche Verbindung direkt nacheinander anfragen. Da die Dateien so klein sind, passten sie sogar beide in das gleiche Antwort-Paket (Paket 7).

Wird dieselbe Seite zum Vergleich ohne SPDY aufgerufen, zeigt sich das sequentielle und damit nachteilige Verhalten von HTTP. Es fällt zwar auf, dass ohne SPDY insgesamt weniger Pakete übertragen werden mussten. Dies liegt vor allem daran, dass Server und Browser über SPDY zuerst gewisse Einstellungen austauschen, die aber dann über die ganze Session gültig sind. Dieser anfängliche Mehraufwand in der Übertragung kann aber durch spätere Effizienz problemlos wieder aufgeholt werden. Denn SPDY ist für die echte Welt gedacht, in der Webseiten nicht aus nur drei Dateien bestehen [13 S. 13].

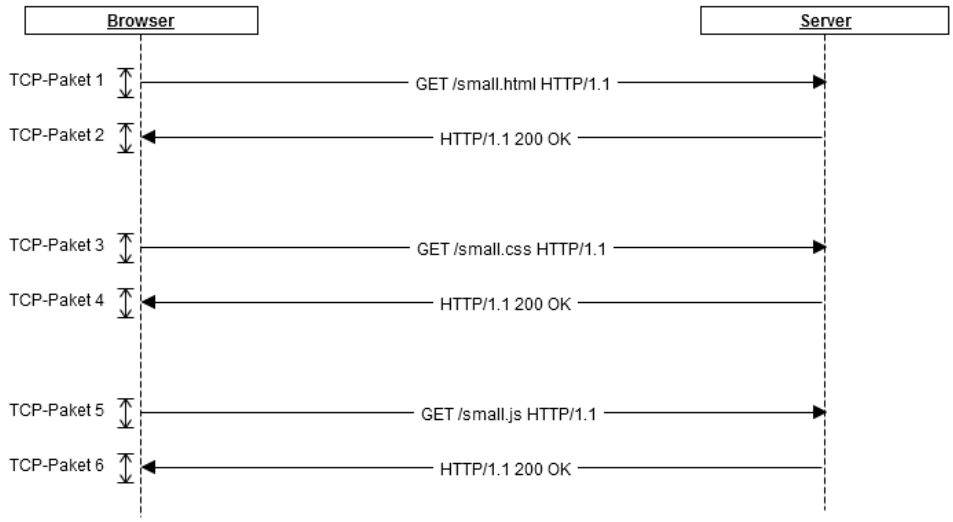


Abbildung 8: Sequenzdiagramm einer HTTP-Session

2.4.3 Weitere Eigenschaften

Neben der im Beispiel gezeigten Bündelung von Anfragen und Antworten und der Komprimierung sämtlicher Header- und Nutzdaten wurden für das Protokoll weitere Eigenschaften spezifiziert, die hier kurz erläutert werden.

Priorisieren von Anfragen

Im Kontroll-Rahmen für die Eröffnung eines Streams (SYN_STREAM) sind drei Bit für die Angabe einer Priorität von 0 (höchste) bis 7 (tiefste) reserviert. Die Gegenseite ist aufgefordert, wenn mehrere zu bearbeitende Streams gleichzeitig vorliegen, diese mit der entsprechenden Priorität zu behandeln [10]. Ein Browser könnte diesen Mechanismus beispielsweise benutzen, um die für das Layout einer Seite benötigten CSS-Dateien mit höherer Priorität zu laden als Bilder im Inhalt der Seite.

Server Push

Server Push meint, dass der Server Daten an einen Client sendet, die dieser nicht explizit angefordert hat. Der Begriff kann verwirrend wirken, wenn man mit Technologien wie Comet, AJAX Long Polling oder Websockets vertraut ist, die alle das Ziel haben, Daten zu einem beliebigen Zeitpunkt vom Server an den Client zu senden (beispielsweise bei einem Ereignis, von dem nur der Server weiss und den Client darüber informieren soll) [13 S. 71].

Server Push in SPDY bedeutet, dass der Server bei der Anfrage eines Clients neben der erwarteten Antwort noch zusätzliche Dateien (Streams) mit senden kann, die zu der Anfrage relevant sind. Somit könnte ein Server bei der Anfrage einer HTML-Seite gleich alle CSS- und JavaScript-Dateien, die für die Anzeige der Seite benötigt werden, mit senden.

Der Browser legt diese in seinen Zwischenspeicher ab und muss sie daher später nicht erneut anfragen, wenn er die Seite aufbaut [13 S. 76].

Server Hint

Ähnlich wie Server Push ist Server Hint (Hinweis) eine Möglichkeit, einen Client zu informieren, dass zu einer angefragten Datei weitere Ressourcen relevant sind. Anders als beim Server Push werden die Ressourcen aber nicht gleich mit gesendet sondern es wird lediglich darauf hingewiesen [6].

2.4.4 Stand der Entwicklung

Die Spezifikation von SPDY befindet sich zum Zeitpunkt der Entstehung dieser Arbeit im Stadium „Entwurf“ für die Version 3 [10].
Obwohl es sich um eine experimentelle Technologie handelt, die noch nicht von der offiziellen Stelle IETF (Internet Engineering Task Force) anerkannt wurde [14], wird SPDY sowohl auf der Browser-Seite wie auf der Server-Seite schon breit angewandt.

Folgende Browser unterstützen mindestens SPDY 2 [14], [15], [16]:

- Google Chrome/Chromium 6+
- Firefox 11+, ab Version 15 auch SPDY 3
- Opera / Opera Mobile 12.10+
- Android Browser 3+
- Chrome für Android 18+
- Firefox für Android 18+

Folgende grösseren Webseiten können bereits per SPDY aufgerufen werden (über HTTPS) [14]:

- Alle Services von Google
- Twitter
- Alle auf www.wordpress.com verfügbaren Blogs

Durch die freie Verfügbarkeit des Apache-Moduls wird sich die Zahl der Seiten, die SPDY unterstützen, laufend erhöhen.
Das Apache-Modul mod_spdy hat die Version 0.9.3.3 (r386, 04.11.2012) erreicht und unterstützt SPDY 3 inklusive Server Push.

3 Ladezeiten von Webseiten optimieren mit mod_pagespeed

Dieses Kapitel untersucht die Funktionsweise des von Google entwickelten Apache-Moduls mod_pagespeed. Es wird davon ausgegangen, dass der Leser weiss, wie eine Webseite aufgebaut ist und die Technologien/Sprachen HTML, CSS und JavaScript kennt.

3.1 PageSpeed als Überbegriff

Als Teil der Initiative „Let's make the web faster" hat Google den Überbegriff PageSpeed eingeführt [4]. Unter dem Begriff versammeln sich mehrere Methoden und Technologien, die alle das gleiche Ziel haben: Die Struktur von Webseiten so zu verbessern, dass sie beim Endbenutzer schneller ankommen und dargestellt werden.
Unter PageSpeed fasst Google folgende Dienste und Produkte zusammen [17]:

- PageSpeed Insights: Ein Online-Service, der eine Webseite analysiert und dann eine Liste mit Vorschlägen generiert, wie die Seite optimiert werden kann.
- PageSpeed Insights Extensions: Browser-Erweiterungen für Mozilla Firefox und Google Chrome, die das gleiche machen wie der Online-Service, aber direkt im Browser. Dies kann von Vorteil sein, wenn eine Seite analysiert werden soll, die nicht öffentlich erreichbar ist.
- PageSpeed Rules: Eine Liste mit Regeln, die von PageSpeed Insights benutzt wird, um Vorschläge für die Optimierung einer Seite aufzustellen. Diese Regeln wurden anhand der „*Performance Best Practices*" (dt.: Erfolgsmethoden für Performanz) erstellt, einer Liste von Anweisungen, wie die Performanz einer Webseite optimiert werden kann [18].
- PageSpeed Service: Ein Service im Teststadium, der es einem Webseitenbetreiber erlaubt, seine Seite über die Google-Server auszuliefern, welche dann alle Optimierungen der PageSpeed Rules darauf anwendet. Dies kann dann sinnvoll sein, wenn der Betreiber der Seite nicht selber Software wie mod_pagespeed auf dem Server installieren kann, wie dies bei Web-Hostern der Fall sein kann.
- mod_pagespeed: Ein Apache-Modul, das beim Ausliefern einer Webseite die Optimierungen der PageSpeed Rules direkt auf diese anwendet.

In dieser Arbeit wird der Begriff PageSpeed benutzt, wenn vom Vorgehen die Rede ist, eine Webseite anhand der mit den PageSpeed Rules aufgestellten Vorschläge zu opti-

mieren. Wird von PageSpeed als Produkt gesprochen, dann ist damit das Apache-Modul mod_pagespeed gemeint.

3.2 Die Idee des Moduls

Die umfangreiche Liste mit Anleitungen, wie eine Webseite optimiert werden kann („*Performance Best Practices*“ [18]), sollte von allen Web-Entwicklern konsultiert werden, die einen schnellen Internetauftritt erstellen wollen.
Doch es gibt Situationen, in denen es nicht sinnvoll oder möglich ist, diese Anpassungen als Entwickler selbst durchzuführen. Dies ist beispielsweise der Fall, wenn ein proprietäres Content Management System (CMS) eingesetzt wird, dessen Lizenz Änderungen am Code nicht zulassen. Oder wenn eine eingesetzte Open Source-Software schlicht zu umfangreich ist, um diese anzupassen.
Für diese oder ähnliche Fälle wurde das Apache-Modul mod_pagespeed entwickelt. Es analysiert eine auszuliefernde Webseite anhand der PageSpeed Rules und führt dann direkt die ermittelten Verbesserungen am Code der Seite aus. Dies geschieht direkt bei der Auslieferung im Arbeitsspeicher des Servers, die Dateien auf der Festplatte werden nicht verändert.
Dafür installiert das Modul über 40 so genannte Filter in den Apache-Server, die alle ihre eigene Aufgabe besitzen und sich einzeln ein- bzw. ausschalten und konfigurieren lassen [19].

3.3 Funktionsweise im Überblick

Jeder der über 40 Filter von mod_pagespeed nimmt sich einem oder mehreren Punkten der Liste „*Performance Best Practices*“ an und setzt eine Optimierung dafür um.
Da es für diese Arbeit zu umfangreich und für das weitere Verständnis nicht notwendig ist, alle Filter einzeln zu beschreiben, werden hier als Überblick die Kategorien aufgeführt und beschrieben.
Die Filter lassen sich in folgende 6 Kategorien zusammenfassen [20]:

- Optimierung des Caching: Im HTTP-Protokoll wurden diverse Header-Felder definiert, die einem Browser dabei helfen sollen zu entscheiden, ob er eine Ressource aus seinem Zwischenspeicher nehmen oder neu abholen soll. Diese Felder werden aber vielfach missachtet oder falsch benutzt, was dazu führt, dass ein Browser Dateien herunterlädt, obwohl dies nicht nötig wäre. Filter in dieser Kategorie sorgen dafür, dass ein Browser so viele Dateien wie möglich aus seinem lokalen Zwischenspeicher benutzen kann.

- Round Trip Times reduzieren: Jede Anfrage an den Server kostet Zeit. Diverse Filter dieser Kategorie sorgen dafür, dass möglichst wenige Anfragen gesendet werden müssen. Dies geschieht vor allem dadurch, dass mehrere Ressourcen des gleichen Typs zu einer zusammengefasst und diese gesammelten Dateien dann in der optimalen Reihenfolge abgefragt werden.
- Grösse der Anfrage verringern: Browser senden bei Anfragen oft Daten mit, die für den Server gar nicht notwendig sind. Dies kann unter Umständen dazu führen, dass eine Anfrage nicht mehr in ein TCP-Paket passt und darum zwei oder mehr Pakete gesendet werden müssen, was die Zeit für eine Anfrage deutlich erhöht. Filter dieser Kategorie versuchen den Browser dazu zu bringen, dass er nur noch das sendet, was auch wirklich nötig ist. Dies wird unter Anderem dadurch erreicht, dass Cookies nur dort gesetzt werden, wo sie auch benötigt werden.
- Grösse der Nutzdaten reduzieren: Je weniger Daten ein Browser herunterladen muss, desto rascher kann er sie darstellen. Diese Filter wenden je nach Dateityp unterschiedliche Methoden an, um deren Inhalt so weit wie möglich zu verkleinern, ohne dass dabei wichtige Informationen verloren gehen. Neben der Komprimierung der Daten bei der Übermittlung wird dies auch durch die Entfernung unwichtiger Inhalte (wie z.B. Kommentare in Scripts oder überflüssige Leerzeichen in Texten) erwirkt.
- Darstellung im Browser beschleunigen: Es gibt einige Tricks, die dem Browser die Arbeit der Darstellung einer Seite erleichtern. Die Filter dieser Kategorie setzen diese Tricks um. Dabei geht es vor allem darum, die einzelnen Ressourcen einer Seite in der richtigen Reihenfolge zu platzieren.
- Diverse Filter: Die letzte Kategorie beinhaltet alle Filter, die sonst in keine Kategorie passen. Darunter sind solche, die eine gewisse Vorarbeit leisten, damit die anderen Filter ihre Arbeit korrekt durchführen können.

3.4 Umsetzung als Apache-Modul

Google hat die erste Version von mod_pagespeed im November 2010 öffentlich zugänglich gemacht. Danach wurde es stetig weiter entwickelt und im Dezember 2012 ist die erste als stabil bezeichnete Version erschienen.
Es gibt vorkompilierte Pakete für diverse Linux-Distributionen, die entweder mit Apache 2.2 oder Apache 2.4 arbeiten [19].

3.4.1 Installation und Konfiguration

Die Installation wird auf demselben virtuellen Server durchgeführt, auf dem bereits mod_spdy eingerichtet ist. Da beide Module unterschiedliche Ansätze verfolgen, sollten sie sich nicht gegenseitig in die Quere kommen. Für die späteren Tests ist es sogar sehr wichtig, dass die Module gut zusammen arbeiten, da sie gleichzeitig eingesetzt werden sollen.
Bei der Installation sind jedenfalls keine Probleme dieser Art aufgetaucht.

Für die Installation wird ein vorkompiliertes Paket der neusten stabilen Version für Ubuntu 64bit verwendet [19]:

```
## Vorkompiliertes PageSpeed-Modul von Google herunterladen
$ cd /tmp
$ wget https://dl-ssl.google.com/dl/linux/direct/mod-pagespeed-stable_current↵
_amd64.deb

## Modul installieren
$ sudo dpkg -i mod-pagespeed-stable_current_amd64.deb

## Apache neu starten
$ sudo service apache2 restart
```

Danach sollte das Modul funktionieren. Dies kann überprüft werden, in dem man sich die Antwort-HTTP-Header des Servers anschaut. Ist dort die Versionsnummer des Moduls zu finden, ist mod_pagespeed korrekt installiert:

```
x-mod-pagespeed: 1.1.23.2-2258
```

Doch damit alle Filter ihre Arbeit korrekt erledigen können, müssen einige Werte in der Apache-Konfigurations-Datei eingefügt werden:

```
## Apache-Konfiguration anpassen
$ sudo vi /etc/apache2/httpd.conf
```

```
## Folgende fünf Zeilen müssen in der Datei httpd.conf hinzugefügt werden:
<IfModule pagespeed_module>
  ModPagespeedFileCachePath "/var/cache/mod_pagespeed/"
  ModPagespeedLoadFromFileMatch ↵
"^https?://bachelor.aws.guggero.org/(.*).(js|css|jpg|png)" "/var/www/\\1.\\2"
  ModPagespeedEnableFilters ↵
combine_javascript,outline_javascript,move_css_above_scripts,↵
insert_image_dimensions,remove_comments,collapse_whitespace
</IfModule>

## Apache neu starten
$ sudo service apache2 restart
```

Diese zusätzlichen Zeilen weisen das Modul an, wo es seine temporären Dateien ablegen soll und welche Ressourcen es wo auf der Harddisk findet.
Weiter werden einige Filter aktiviert, die standardmässig ausgeschaltet sind. Es handelt sich dabei um Filter, die offiziell noch nicht als stabil betrachtet werden [20], aber in den Augen des Autors einen weiteren Beitrag zur Optimierung leisten können.
Diese sind zwar für den Betrieb des Moduls nicht notwendig, da die späteren Praxistests aber mit diesen Filtern durchgeführt werden, sind sie hier aufgeführt.

3.5 Technische Beschreibung

Für die genauere Untersuchung einiger Eigenschaften von mod_pagespeed wurde wiederum eine kleine Test-Seite erstellt. Die statische HTML-Seite enthält zwei CSS-, zwei JavaScript- und drei Bilddateien. Natürlich entspricht auch dieses Beispiel nicht der realen Situation im Internet, doch reicht es aus um die Funktionsweise einiger der über 40 Filter aufzuzeigen.

Um einen ersten Eindruck des Unterschiedes zwischen ein- und ausgeschaltetem mod_pagespeed zu erhalten, wird die Seite http://bachelor.aws.guggero.org/small2.html in beiden Zuständen aufgerufen und mit dem in Google Chrome eingebauten Entwickler-Werkzeug (kann mit der Taste F12 geöffnet werden) das Netzwerkverhalten betrachtet.
Ist mod_pagespeed ausgeschaltet, werden wie erwartet die 8 Dateien nacheinander in ihrer originalen Dateigrösse übertragen. Es werden insgesamt knapp 150 kByte an Nutzdaten (GZIP-komprimiert) heruntergeladen und der Browser kann diese nach 500 Millisekunden darstellen:

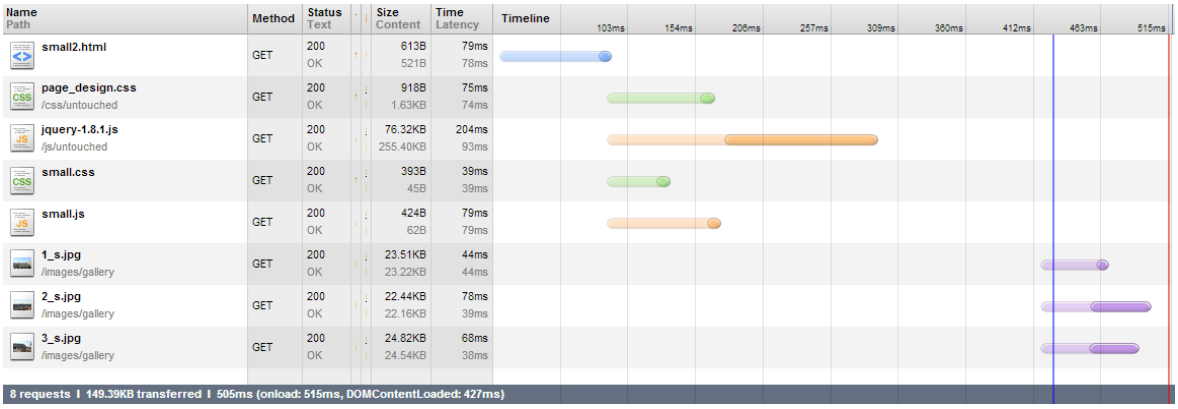


Abbildung 9: Netzwerkverhalten der Test-Seite bei ausgeschaltetem mod_pagespeed

Wird mod_pagespeed auf dem Server eingeschaltet und die Seite im Browser neu geladen, ergibt sich ein komplett anderes Bild:

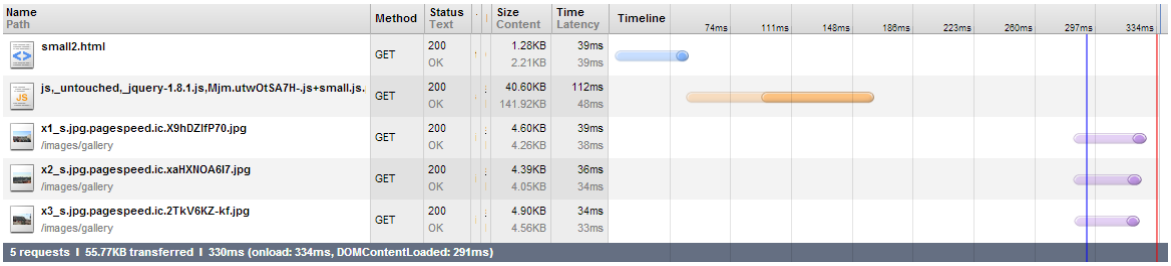


Abbildung 10: Netzwerkverhalten der Test-Seite bei eingeschaltetem mod_pagespeed

Der Browser macht nun noch 5 Anfragen, lädt nur etwas mehr als einen Drittel der ursprünglichen Gesamt-Dateigrösse herunter und zeigt die Seite in 65% der vorherigen Zeit an (330 Millisekunden).

Um nachvollziehen zu können, was mod_pagespeed alles am Inhalt der Seite verändert, werden nun die wichtigsten Filter, die in diesem Beispiel zum Einsatz kommen, aufgelistet und ihre Funktionsweise erläutert. Die Informationen dazu wurden aus [20] und aus der mit dem Modul mit installierten Datei `/etc/apache2/mods-available/pagespeed.conf` zusammengetragen.

3.5.1 Filter extend_cache

Damit der Browser statische Dateien so lange als möglich aus dem lokalen Cache laden kann, muss er sicher sein, dass sich eine Datei in der Zwischenzeit auf dem Server nicht verändert hat. Darum fügt dieser Filter eine Prüfsumme (Hash) des Inhalts einer Ressource in deren Dateiname ein. Ändert sich der Inhalt so ändert sich damit auch der Name einer Datei und der Browser merkt, dass er sie neu anfordern muss, sofern er sie noch nicht hat. Die Prüfsumme ist in der Abbildung 10 gut sichtbar, beispielsweise am Ende der neuen Dateinamen der Bilder.

Über das HTTP-Header-Feld „*Cache-Control*“ wird der Browser zudem aufgefordert, eine solche Ressource standardmässig bis zu einem Jahr zwischen zu speichern.

3.5.2 Filter inline_css

Ist eine CSS-Datei kleiner als die standardmässig eingestellten 2048 Bytes, dann ist der Aufwand für eine eigene Anfrage für diese Datei grösser als der Gewinn, dass sie beim nächsten Aufruf der Seite aus dem Cache geladen werden könnte. Darum werden kleine Style Sheets direkt mit einem `<style>`-Tag in die Seite integriert, statt sie als externe (und daher zwischenspeicherbare) Ressource anzubieten.
Deshalb ist im zweiten Bildschirmfoto keine Anfrage für eine CSS-Datei erkennbar, dafür fällt auf, dass sich die Grösse der HTML-Datei etwa verdoppelt hat.

3.5.3 Filter combine_javascript

Dieses Modul kombiniert mehrere JavaScript-Dateien zu einer grösseren Datei. Dies reduziert die Anzahl an Anfragen deutlich. Im Beispiel ist erkennbar, dass bei eingeschaltetem mod_pagespeed nur noch eine JavaScript-Datei übertragen wird.

3.5.4 Filter rewrite_javascript und rewrite_css

Sowohl CSS als auch JavaScript sind Textdateien. Diese enthalten oftmals Textbausteine, die zwar für den Entwickler sinnvoll aber für den Browser völlig irrelevant sind. Als Beispiel sind hier Kommentare oder Leerzeichen und Umbrüche für die Formatierung zu nennen. Beide Filter haben ein Set von Regeln hinterlegt, nach denen sie die Inhalte der jeweiligen Dateitypen aufs Wesentliche reduzieren können, ohne dabei ihre Funktionsweise zu verändern.
Vergleicht man die Dateigrösse vor der GZIP-Komprimierung (in den Bildschirmfotos in der Spalte „*Size*“ der jeweils zweite Wert) dann stellt man fest, dass nach der Minimierung der kombinierten JavaScript-Datei nur noch etwas mehr als die Hälfte an kByte übrig bleibt.

3.5.5 Filter rewrite_images

Oftmals sind Bilder, vor allem im JPEG-Format, mit einer zu hohen Qualitätsstufe komprimiert. Gerade bei kleinen Vorschau-Bildern ist von blossem Auge kaum ein Unterschied festzustellen, ob sie mit 60% oder 100% Bildqualität komprimiert wurden. Doch die Dateigrösse reduziert sich bei geringerer Qualität beachtlich.

Der Filter macht sich diesen Fakt zunutze und komprimiert Bilder neu, bei denen es sich lohnt. Im Beispiel sind die ursprünglich mit 100% Qualität gespeicherten Bilddateien nach dieser Optimierung noch gut einen Fünftel so gross wie zuvor.

3.5.6 Zusammenfassung

Anhand des gezeigten Beispiels ist gut erkennbar, dass die einzelnen Filter nicht unabhängig voneinander arbeiten sondern im Gegenteil ihr volles Potenzial erst durch die Kombination mehrerer Filter entfalten können. Beispielsweise werden JavaScript-Dateien zuerst minimiert, dann mehrere zusammengefügt und am Schluss für das Caching optimiert.
Die Leistung von mod_pagespeed besteht also nicht nur darin, die Filter zur Verfügung zu stellen, sondern auch ihre Reihenfolge und Kombinationsmöglichkeiten geschickt zu organisieren.

Die fünf untersuchten Filter sollten dem Leser einen Eindruck vermitteln, was hinter der Idee von PageSpeed steckt und wie diese von mod_pagespeed umgesetzt wird. Die komplette Liste aller Filter mit umfangreichen Beschreibungen ist öffentlich zugänglich [20] und sollte allfällige weitere Fragen des Lesers beantworten können.

3.6 Einsatz als Proxy-Server

Da mod_pagespeed gut mit dem Proxy-Modul mod_proxy von Apache zusammen arbeitet, ist es möglich einen Proxy-Server aufzubauen, der beliebige Webseiten für die Nutzer des Proxys optimiert. Würde dieser Server zusätzlich noch SPDY unterstützen, könnte man darüber das ganze Internet für sich beschleunigen. Gerade auf einem Smartphone könnte dies diverse Vorteile mit sich bringen. Eine Arbeit, die sich mit einem Aspekt dieser Idee befasst, ist unter [21] zu finden.

4 Optimierungstechnologien in der Praxis

Um die versprochenen Geschwindigkeitsvorteile der beiden Apache-Module in der Praxis zu überprüfen, wurde eine eigene Test-Webseite erstellt und dann die Ladezeit dieser Seite in verschiedenen Konfigurations-Szenarien getestet. Dieses Kapitel beschreibt die getesteten Szenarien und die dazu eingesetzten Hilfsmittel. Die erhobenen Daten der Tests werden dann in Kapitel 5 gezeigt und interpretiert.

4.1 Die Test-Webseite

Die für diese Arbeit erstellte Webseite soll möglichst viele Aspekte heutiger grösserer Internetauftritte nachempfinden. Der dabei wichtigste Aspekt ist, dass die Seite viele (>150) Dateien enthält, die einzeln geladen werden müssen. Dies sind vor allem Bilddateien aber auch JavaScript-Bibliotheken und CSS-Dateien. Ein weiterer Aspekt ist, dass mehrere JavaScript-Frameworks und dazugehörige Plug-Ins verwendet werden, die aufeinander aufbauen und darum in einer bestimmten Reihenfolge geladen und interpretiert werden müssen. Zusätzlich braucht das Interpretieren von viel JavaScript-Code Zeit im Browser, was die gesamte Ladezeit der Seite verlängert.

Diese Überlegungen haben zu einer Seite mit drei Inhaltsblöcken geführt:

1. Eine Bildergallerie mit 120 Fotos. Da die kleine Vorschau-Version von allen Fotos direkt sichtbar sein soll, muss der Browser zwingend alle laden. Beim Klick auf ein Vorschau-Bild wird das Galerie-Script „FancyBox“ aktiv und erlaubt das komfortable Durchsehen aller Bilder in einer grösseren Version.
2. Eine Tabelle mit 28 Zeilen, wovon jede ein eigenes Icon enthält. Zum Sortieren der Spalten wird das Plug-In „tablesorter“ verwendet.
3. Das häufig eingesetzte JavaScript-Framework „jQuery UI“ dient dazu, optisch ansprechende Benutzeroberflächen zu erstellen. Damit der Browser die Scripts auch wirklich lädt und initialisiert ist eine kleine Demonstration der Bedienelemente im dritten Teil der Seite eingebaut.

Damit beinhaltet die Webseite neben vielen kleinen Dateien, die übertragen werden müssen, auch mehrere JavaScript-Frameworks die der Browser nach dem Laden starten muss. Auch die Gesamtgrösse aller Dateien ist mit 2.7 MByte ausreichend gross, dass in den verschiedenen Tests Unterschiede feststellbar sein sollten.

4.1.1 Detaillierte Angaben zur Webseite

Die Test-Webseite ist unter http://bachelor.aws.guggero.org/index.html erreichbar. Es handelt sich um eine komplett statische Seite, Serverseitig muss also nichts berechnet werden.

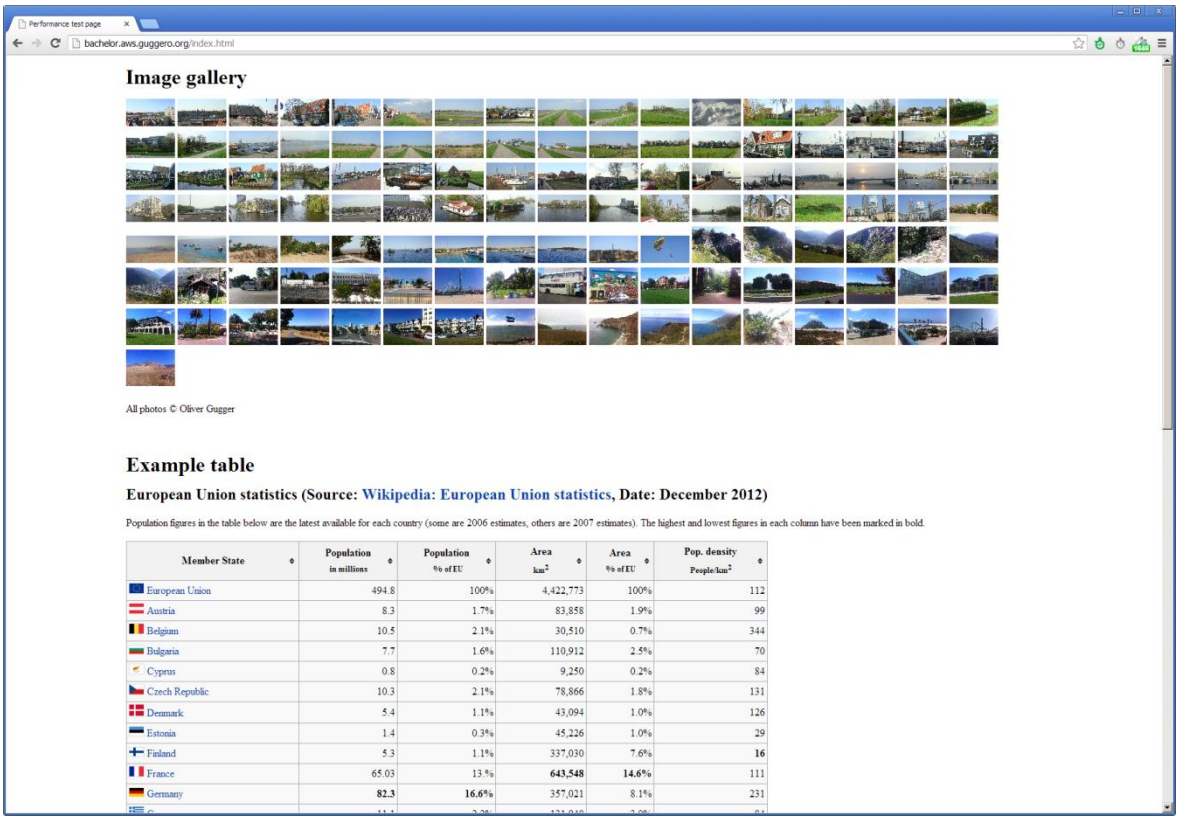


Abbildung 11: Bildschirmfoto der Test-Webseite

Die folgende Tabelle zeigt die Anzahl und Grösse aller Dateien, die für den Aufbau der Seite notwendig sind. Die Gesamtgrösse zeigt die Anzahl effektiv übertragener kBytes. Dieser Wert ist bei textbasierten Inhalten (HTML, CSS, JavaScript) kleiner als die effektive Dateigrösse, weil diese standardmässig GZIP-komprimiert übertragen werden.

Typ	*Anzahl*	*Gesamtgrösse (kByte)*
HTML	1	4.4
CSS	5	8.6
JavaScript	7	187.3
Icons (PNG)	32	6.34
Icons (GIF)	1	0.06
Bilder (JPG)	120	2529.28
Total	166	2735.98

Tabelle 3: Auflistung aller Dateien der Test-Seite und deren Gesamtgrösse

Die Seite wurde absichtlich ohne Berücksichtigung allfälliger Optimierungen erstellt. Das bedeutet, alle JavaScript- und CSS-Dateien wurden in ihrer originalen, nicht minimierten oder komprimierten Form einzeln eingebunden. Die kleinen Vorschau-Versionen der Fotos wurden mit 100% JPEG-Qualität komprimiert.

4.1.2 Manuell optimierte Version der Webseite

Obwohl diese Arbeit explizit das Ziel hat, die beiden Apache-Module zu untersuchen, ohne dass Anpassungen am Inhalt der Seite notwendig sind, ist es gerade zur Beantwortung der Frage, ob der Einsatz von PageSpeed sinnvoll ist, interessant zu sehen, was ein Web-Entwickler selbst zur Optimierung einer Seite beitragen kann.
Zu diesem Zweck wurde eine manuell optimierte Version der Test-Webseite erstellt. Diese ist unter `http://bachelor.aws.guggero.org/optimized.html` erreichbar und ist optisch kaum unterscheidbar von der nicht optimierten Version.

Es wurden folgende Verbesserungen vorgenommen:

- Von allen JavaScript- und CSS-Dateien wurde die minimierte Version eingebunden, die bei den meisten Frameworks mitgeliefert wird
- Alle JavaScript- und CSS-Dateien wurden in der richtigen Reihenfolge in eine jeweils einzelne Datei kopiert
- Die kleinen Vorschau-Versionen der Fotos wurden mit 60% JPEG-Qualität komprimiert
- Die Abmessungen der Vorschau-Bilder in Pixel ist bereits im HTML-Code angegeben

Daraus ergibt sich die folgende neue Gesamtgrösse der Dateien:

Typ	*Anzahl*	*Gesamtgrösse (kByte)*
HTML	1	4.2
CSS	1	6
JavaScript	1	95.8
Icons (PNG)	32	6.34
Icons (GIF)	1	0.06
Bilder (JPG)	120	149.5
Total	156	261.9

Tabelle 4: Auflistung aller Dateien der optimierten Test-Seite und deren Gesamtgrösse

4.2 Die Testmethoden

Es gibt verschiedene Möglichkeiten, die Ladegeschwindigkeit einer Seite zu messen. Beispielsweise könnte ein Kommandozeilen-Programm, wie das aus der Linux-Welt bekannte *wget,* die Seite mehrmals nacheinander herunterladen. Dies würde aber nicht dem entsprechen, was der Benutzer in seinem Browser erlebt. Denn zum Einen verhält sich ein Browser unter Umständen anders als *wget* und zum Anderen wird die Zeit vernachlässigt, die der Browser benötigt, um die geladenen Informationen darzustellen. Ganz zu schweigen davon, dass *wget* noch gar keine Unterstützung für SPDY bietet.
Daher liegt es nahe, den Browser selbst als Messinstrument zu benutzen.

4.2.1 Verwendete Hilfsmittel und Technologien

Zurzeit stehen lediglich zwei Browser zur Verfügung, die SPDY in der Version 3 unterstützen: Mozilla Firefox und Google Chrome. Und von diesen beiden bietet nur Chrome eine Erweiterung an, die eine Webseite mehrmals nacheinander laden und detaillierte Statistiken darüber aufstellen kann.
Daher fiel die Wahl auf Google Chrome in der aktuellen Version 23 mit der Erweiterung „Page Benchmarker“ in der Version 1.3.4.0.

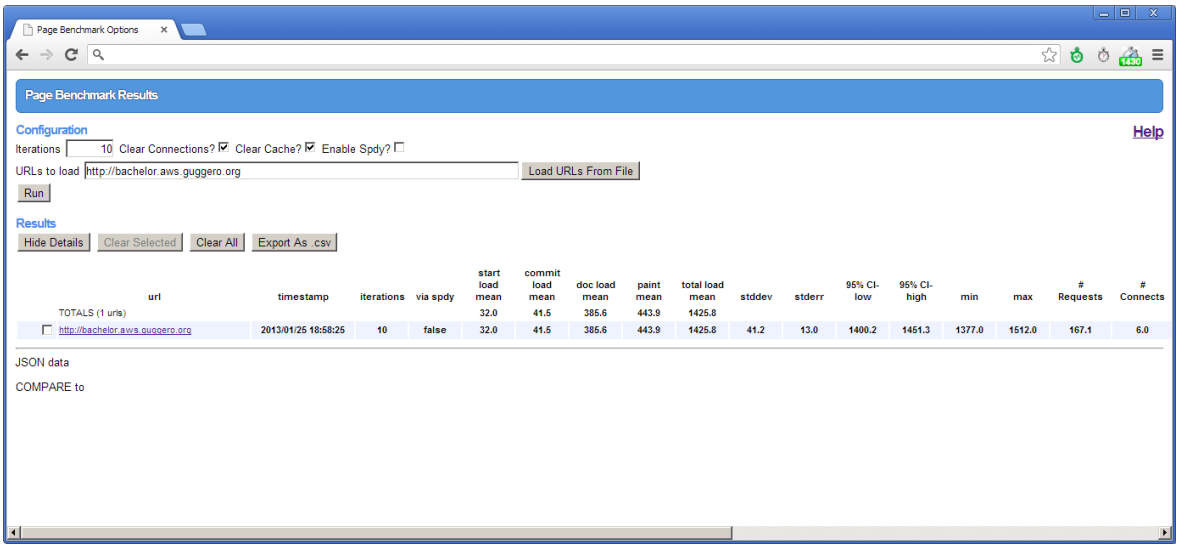


Abbildung 12: Google Chrome mit der Erweiterung "Page Benchmarker"

Diese Benchmark-Erweiterung bietet einige nützliche Optionen. Neben der Anzahl der durchzuführenden Iterationen kann auch angegeben werden, dass aufgebaute Verbindungen nach jedem Durchlauf geschlossen und der Cache geleert wird. Das Schliessen der Verbindungen ist vor allem für Tests mit SSL/TLS wichtig, da der Aufbau dieser Verbindungen zeitaufwändig ist und darum bei jedem Durchlauf erneut berücksichtigt werden soll.

Die Option, ob SPDY verwendet werden soll, wurde für die Tests explizit deaktiviert (siehe Anhang D) da diese in der etwas älteren Version des „Page Benchmarker“ zur Folge hat, dass nur SPDY 2 verwendet wird. Ob bei einem Test SPDY verwendet wird, gibt also allein der Server vor.

Die Daten, welche durch die Erweiterung aufgezeichnet werden, sind umfangreich und decken beinahe alles ab, was für die Tests relevant ist. Lediglich die Anzahl der übertragenen TCP-Pakete in Sende- und Empfangsrichtung mussten manuell mit Wireshark 1.8.4 aufgezeichnet und gezählt werden.

4.2.2 Die 7 Test-Szenarien

Durch die Auseinandersetzung mit den Technologien SPDY und PageSpeed haben sich vier Parameter herauskristallisiert, deren Einfluss auf die Ladegeschwindigkeit in den folgenden Tests ermittelt werden soll:

Parameter	*Beschreibung*
SSL/TLS	Da die Verschlüsselung von Verbindungen eine gewisse Zeit benötigt, ist dies ein Parameter der berücksichtigt werden muss.
SPDY	Der Einfluss von SPDY auf die Ladezeit wird mit diesem Parameter überprüft. Es gibt eine Abhängigkeit zu SSL/TLS, denn SPDY setzt zwingend voraus, dass die Verbindung verschlüsselt ist.
PageSpeed	Ermittelt den Einfluss von PageSpeed auf die Geschwindigkeit der Webseite.
Optimiert	Die manuell optimierte Seite wird als Alternative zu PageSpeed in Betracht gezogen.

Tabelle 5: Parameter der Test-Szenarien

Da nicht alle der möglichen 16 Kombinationen sinnvoll oder möglich sind, wurden die folgenden sieben Szenarien ausgewählt:

Nr.	*Bezeichnung*	*SSL/TLS*	*SPDY*	*PageSpeed*	*Optimiert*
1	HTTP	-	-	-	-
2	HTTPS	✓	-	-	-
3	SPDY	✓	✓	-	-
4	HTTPS + PageSpeed	✓	-	✓	-
5	SPDY + PageSpeed	✓	✓	✓	-
6	HTTP + optimiert	-	-	-	✓
7	SPDY + optimiert	✓	✓	-	✓

Tabelle 6: Die 7 Test-Szenarien

4.2.3 Volle Bandbreite und mobile Bandbreite

Um auch eine Aussage machen zu können, wie sich die verschiedenen Konfigurationen unter eingeschränkter Bandbreite und höheren Latenzzeiten bewähren, wurde entschieden, alle sieben Szenarien auch unter mobilen Bedingungen zu testen. Immerhin gibt es gerade im Android-Umfeld bereits einige Browser für Mobilgeräte, die SPDY unterstützen [16].

Dafür wurden die Bandbreite und die Latenz auf dem Server so simuliert, wie sie in einem mobilen Netz über UMTS/3G messbar sind. Die Ermittlung der dafür nötigen Angaben sowie die Konfiguration des Servers für die Simulation sind in Anhang C beschrieben.

4.2.4 Performance und Netzanbindung der Testgeräte

Der Computer, beziehungsweise die Internetverbindung, auf dem alle Messungen durchgeführt werden, hat folgende Leistungsmerkmale:

Merkmal	*Wert*
Betriebssystem	Microsoft Windows 7
Prozessor	Intel Core i7 (4 Cores, 8 Threads)
Prozessor-Architektur	64 Bit
Arbeitsspeicher	12 GB RAM
Browser	Chrome 23
Netzanbindung	Fiber, 100 MBit/s Download, 50 MBit/s Upload

Tabelle 7: Leistungsmerkmale des Test-Clients

Der Server auf der Gegenseite hat die nachfolgenden Eigenschaften. Dabei ist zu erwähnen, dass die Up- und Downloadraten aus Sicht des Servers gegenüber dem Client zu verstehen sind und nur den gemessenen Maximalwerten entsprechen, da hier die Angaben über die reale Verbindung von Amazon nicht vorliegen.

Merkmal	*Wert*
Typ	Virtueller Ubuntu-Server 12.04 auf Amazon AWS (siehe 2.3.1)
Netzanbindung	Wird von Amazon nicht angegeben. Die folgenden Werte wurden durch Messungen zwischen dem Client und dem Server ermittelt (Siehe Anhang B).
Bandbreite	Gemessen: mind. 45 MBit/s Download, 70 MBit/s Upload
Latenz	Gemessen: 30 Millisekunden Round Trip Time zwischen Client und Server

Tabelle 8: Eigenschaften des Test-Servers

Für die Tests unter mobilen Bedingungen wurden folgende Werte auf dem Server simuliert:

Merkmal	*Wert*
Typ	Virtueller Ubuntu-Server 12.04 auf Amazon AWS (siehe 2.3.1)
Netzanbindung	Wurde für die Tests künstlich gedrosselt, um eine Anbindung über 3G/UMTS zu simulieren (siehe Anhang C).
Bandbreite	Konfiguriert: Download 2.6 MBit/s, Upload 3.5 MBit/s
Latenz	Konfiguriert: 53 Millisekunden Verzögerung, was zu durchschnittlich 83 Millisekunden Round Trip Time führt. Dies entspricht somit in etwa den Messungen aus Anhang C.

Tabelle 9: Simulierte Eigenschaften des Test-Servers für mobile Bedingungen

Durch den relativ grossen Unterschied in der Bandbreite zwischen den beiden Konfigurationen sollte sich deutlich zeigen, welche der beiden untersuchten Technologien bei welchen Bedingungen den grössten Einfluss hat.

4.2.5 Bewertungskriterien

Bei der Messung mit der Browser-Erweiterung „Page Benchmarker“ fällt eine Vielzahl von Messwerten an. Davon sind folgende besonders aussagekräftig und werden deshalb als Kriterien für die Bewertung der verschiedenen Test-Szenarien benutzt:

- Durchschnittliche komplette Ladezeit (total load mean): Die durchschnittliche totale Zeit, die der Browser benötigt, um die Seite komplett zu laden und darzustellen.
- Errechnete Down- und Uploadgeschwindigkeit (Read KBps/Write KBps): Die anhand der übertragenen Bytes und der dafür benötigten Zeit berechnete Übertragungsgeschwindigkeit. Zeigt auf, wie gut die Auslastung der zur Verfügung stehenden Bandbreite ist.
- Anzahl übertragener kBytes an Nutzdaten (Read KB/Write KB): Die Gesamtgrösse der empfangenen und gesendeten Daten.
- Anzahl übertragener TCP-Pakete: Kann Auskunft darüber geben, wie gut die Effizienz eines Protokolls ist, wenn diese Zahl mit den effektiv übertragenen kBytes an Nutzdaten verglichen wird.
- Komplette Ladezeit aller Durchläufe (total loan time samples): Die einzelnen Werte aller Durchläufe können aufzeigen, wie gross die Fluktuation zwischen den einzelnen Messungen ist.

4.3 Testausführung

Nach der Festlegung der Testszenarien und der Rahmenbedingungen, konnten die eigentlichen Tests durchgeführt werden.

Um möglichst zuverlässige Resultate zu erhalten, wurden alle sieben Szenarien für beide Bandbreiten am selben Tag unmittelbar nacheinander getestet. Dabei wurden bei jedem Test genau 100 Durchläufe aufgezeichnet.

Alle Daten die vom „Page Benchmarker“ erhoben wurden, konnten als CSV exportiert und dann mit Microsoft Excel weiter verarbeitet werden.

Für die Durchführung der Tests wurde die gleiche Server-Konfiguration verwendet, wie sie in Kapitel 2.3.1 und 3.4.1 erarbeitet wurde.

Die vier Parameter der sieben Szenarien wurden wie folgt ein- bzw. ausgeschaltet:

Parameter	*Beschreibung*
SSL/TLS	Aufruf der Test-URL mit `https://` statt mit `http://` am Anfang.

SPDY	Einschalten des Apache-Moduls:
	`$ a2enmod spdy && service apache2 restart`
	Ausschalten des Apache-Moduls:
	`$ a2dismod spdy && service apache2 restart`

PageSpeed	Einschalten des Apache-Moduls:
	`$ a2enmod pagespeed && service apache2 restart`
	Ausschalten des Apache-Moduls:
	`$ a2dismod pagespeed && service apache2 restart`

Optimiert	Aufruf der URL `bachelor.aws.guggero.org/optimized.html` statt `bachelor.aws.guggero.org/index.html`

Tabelle 10: Ein-/Ausschalten der Test-Parameter

Zum Zeitpunkt der Auslieferung dieser Arbeit wird auf dem Server sowohl SPDY als auch PageSpeed aktiviert sein, damit ein interessierter Leser beide Technologien in Aktion erleben kann.

5 Analyse und Interpretation der Resultate

Dieses Kapitel beschäftigt sich mit den in den Tests erhobenen Daten. Diese werden zuerst in geeigneter Form präsentiert und danach auf ihre Aussage hin analysiert.

5.1 Aufgezeichnete Daten

Die Anzahl der aufgezeichneten Messwerte ist gross. Obwohl nicht alle davon für die Beantwortung der Fragestellungen dieser Arbeit relevant sind, bleiben dennoch viele aussagekräftige Daten übrig, die in diesem Abschnitt gezeigt werden sollen.
Damit die Übersichtlichkeit nicht allzu sehr leidet, werden die Daten in drei Teilen präsentiert. Die ersten beiden Teile zeigen die von der Bandbreite abhängigen Resultate. Der dritte Teil zeigt die Werte, die über alle Bandbreiten hinweg konstant aber für die Bewertung der untersuchten Technologien ebenso relevant sind.

5.1.1 Volle Bandbreite

Die nachfolgende Tabelle zeigt die Resultate der sieben Szenarien, wie sie ohne Beschränkung der Bandbreite als Durchschnittswerte von jeweils 100 Durchläufen ermittelt wurden:

Szenario	*Ladezeit (ms)*	*Download-geschwindigkeit (kByte/s)*	*Upload-geschwindigkeit (kByte/s)*	*Optimierung Ladezeit gegenüber HTTP*	*Optimierung Ladezeit gegenüber HTTPS*
HTTP	1488.2	15382.6	355.3	-	-17.8 %
HTTPS	1810.4	12756.8	364.0	-21.7 %	-
SPDY	1068.7	21411.6	170.8	28.2 %	41.0 %
HTTPS + PageSpeed	1783.7	3407.7	356.2	-19.9 %	1.5 %
SPDY + PageSpeed	756.9	7631.5	230.1	49.1 %	58.2 %
HTTP + optimiert	1380.2	1821.6	371.8	7.3 %	23.8 %
SPDY + optimiert	665.0	3464.6	285.5	55.3 %	63.3 %

Tabelle 11: Aufgezeichnete Daten bei voller Bandbreite

Diese Tabelle alleine lässt bereits die Aussage zu, dass der Einsatz beider Apache-Module SPDY und PageSpeed zusammen eine Optimierung der Ladezeit von bis zu 49% bzw. 58% bewirkt, je nachdem, ob man HTTP oder HTTPS als Ausgangspunkt nimmt.
Die Berechnung von zwei Prozentwerten mit unterschiedlichen Referenzdaten mag verwirrend wirken. Es soll hier aber dem Fakt gerecht werden, dass viele Webseitenbetreiber aus Sicherheitsgründen gar nicht auf den Einsatz von HTTPS verzichten können und darum den Overhead der Verschlüsselung bereits in Kauf nehmen. In diesem Fall ist ein Vergleich mit HTTP gar nicht sinnvoll, beziehungsweise verfälscht die Aussage für diese Zielgruppe.

Grafisch aufbereitet lässt sich rasch erkennen, welches Test-Szenario die besten Ergebnisse liefert:

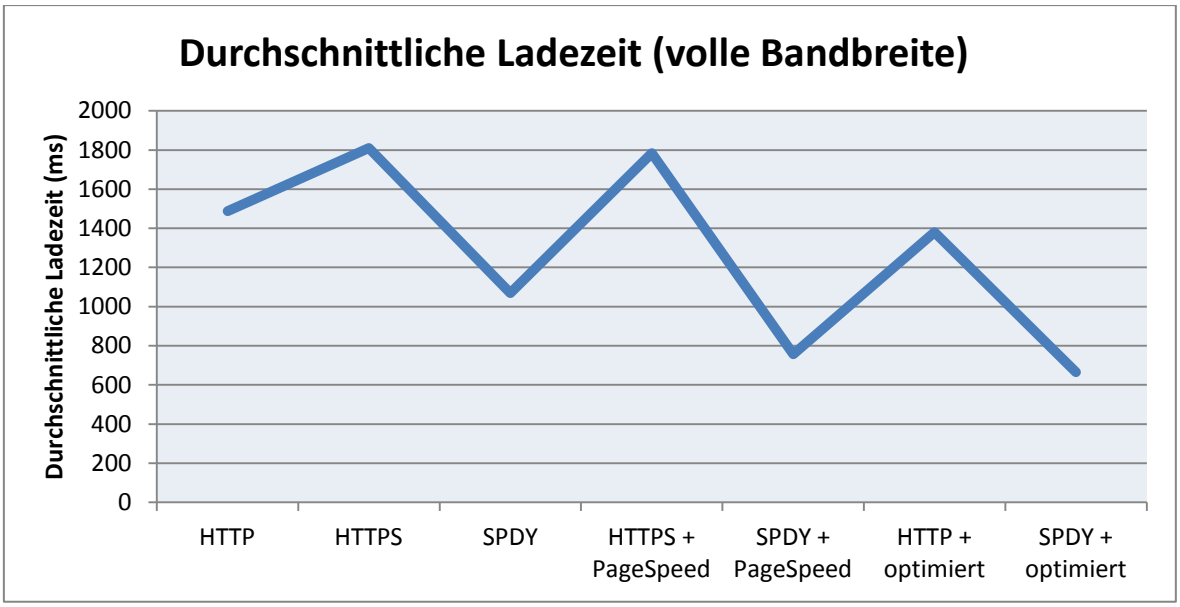


Abbildung 13: Graph der durchschnittlichen Ladezeit bei voller Bandbreite

Bis auf zwei Werte entspricht diese Grafik in etwa dem, was man erwartet, wenn man sich in die involvierten Technologien und deren Versprechen eingelesen hat. Was nicht der Erwartung entspricht ist, dass sowohl PageSpeed und die manuell optimierte Seite alleine (ohne SPDY) bei voller Bandbreite beinahe keine Verbesserung oder sogar eine Verschlechterung der Ladezeit gegenüber HTTP mit sich bringen. Der Grund dafür soll in der späteren Interpretation und Diskussion der Daten ermittelt werden.

Wenn mit Durchschnittswerten gearbeitet wird, ist es immer sinnvoll, die einzelnen Werte auf Anomalien oder extreme Ausreisser hin zu untersuchen, um eine mögliche Verfälschung der Resultate zu verhindern.

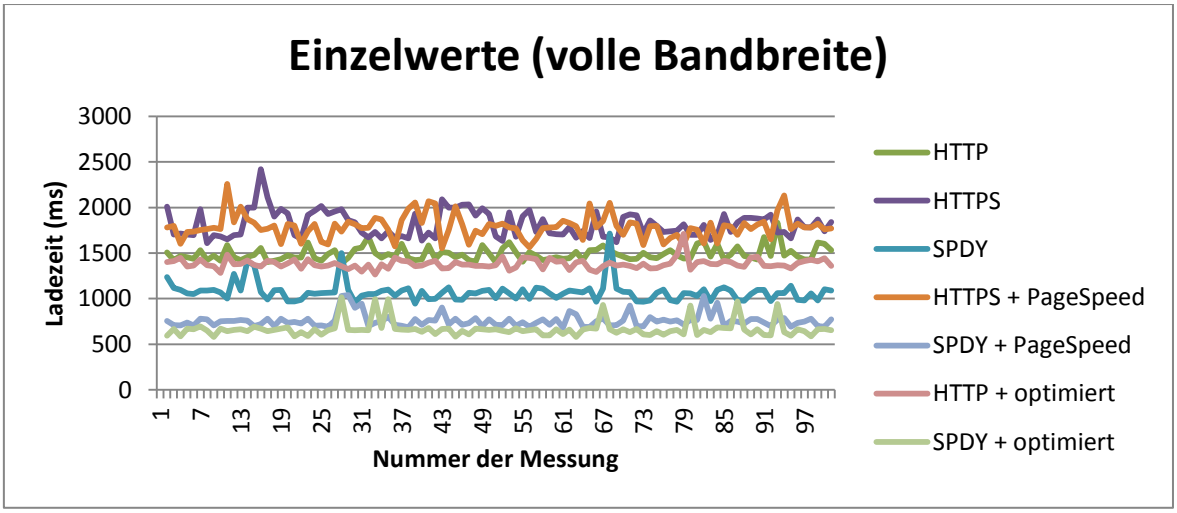


Abbildung 14: Graph der Fluktuation der einzelnen Messwerte bei voller Bandbreite

Die Grafik zeigt zwar einige wenige Ausreisser, diese sollten aber bei einer Stichprobengrösse von 100 Werten nicht zu sehr ins Gewicht fallen.

5.1.2 Mobile Bandbreite

Wird die Bandbreite einer mobilen Umgebung simuliert, ergeben sich folgende Durchschnittswerte von jeweils 100 Durchläufen:

Szenario	*Ladezeit (ms)*	*Download-geschwindigkeit (kByte/s)*	*Upload-geschwindigkeit (kByte/s)*	*Optimierung Ladezeit gegenüber HTTP*	*Optimierung Ladezeit gegenüber HTTPS*
HTTP	7818.7	2923.9	67.5	-	-8.6 %
HTTPS	8553.9	2687.2	76.7	-9.4 %	-
SPDY	7533.3	3021.1	24.1	3.7 %	11.9 %
HTTPS + PageSpeed	4138.7	1463.8	152.2	47.1 %	51.6 %
SPDY + PageSpeed	2610.3	2194.3	66.5	66.6 %	69.5 %
HTTP + optimiert	3259.3	770.5	157.3	58.3 %	61.9 %
SPDY + optimiert	1643.0	1387.8	114.6	79.0 %	80.8 %

Tabelle 12: Aufgezeichnete Daten bei mobiler Bandbreite

Es fällt sofort auf, dass bei geringerer Bandbreite sowohl SPDY wie auch PageSpeed ein deutlich grösseres Optimierungspotenzial aufweisen. Arbeiten beide Technologien zusammen, ist die Seite in nur noch einem Drittel der ursprünglichen Zeit fertig aufgebaut (-66% gegenüber HTTP bzw. -69% gegenüber HTTPS).

Noch deutlicher ist dies in der grafischen Repräsentation zu erkennen:

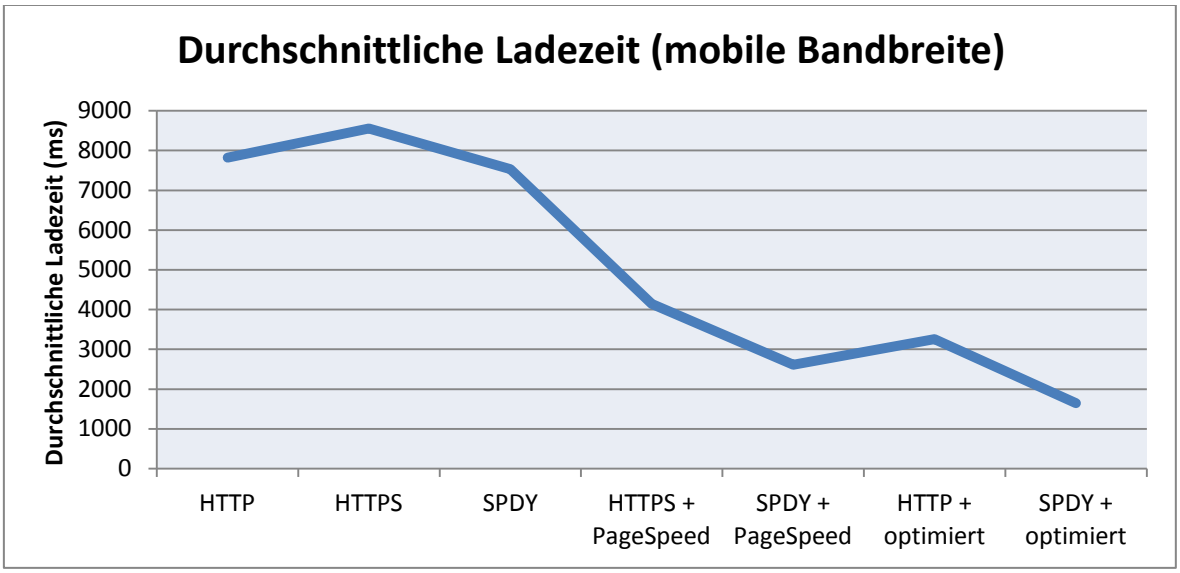


Abbildung 15: Graph der durchschnittlichen Ladezeit bei mobiler Bandbreite

Diese Grafik entspricht relativ gut der Erwartung, die man hat, wenn man sich mit SPDY und PageSpeed auseinandergesetzt hat.

Die Visualisierung der einzelnen Messwerte zeigt, dass die simulierte mobile Umgebung verlässliche Daten liefert, die nur sehr geringe Fluktuationen aufweisen:

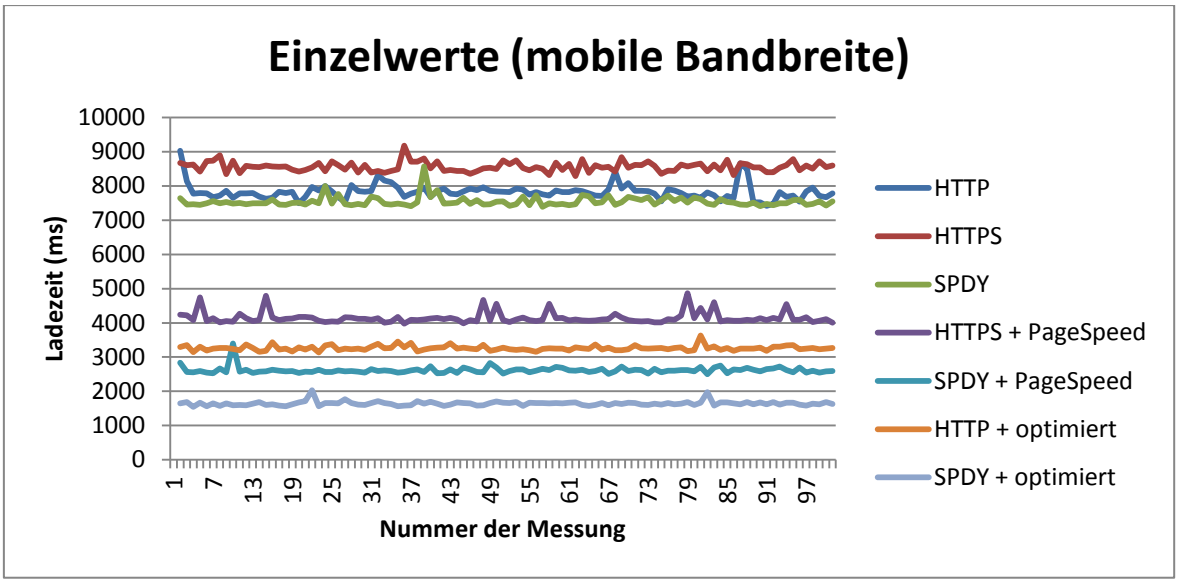


Abbildung 16: Graph der Fluktuation der einzelnen Messwerte bei mobiler Bandbreite

5.1.3 Bandbreitenunabhängige Daten

Die folgende Tabelle enthält Daten, die bei der Interpretation der vorgängig gezeigten Resultate helfen sollen. Diese Werte sind nicht von der Bandbreite, sondern nur vom eingesetzten Protokoll, beziehungsweise dessen Inhalts abhängig:

Szenario	*Download (Pakete)*	*Upload (Pakete)*	*Download (kByte)*	*Upload (kByte)*	*Anzahl Anfragen*
HTTP	2063	1226	2789.3	64.4	167
HTTPS	2218	1313	2805.3	80.1	167
SPDY	1988	1080	2777.9	22.2	167
HTTPS + PageSpeed	775	655	738.8	77.3	131
SPDY + PageSpeed	532	324	700.5	21.1	131
HTTP + optimiert	321	292	306.6	62.6	156
SPDY + optimiert	252	174	278.1	22.9	156

Tabelle 13: Bandbreitenunabhängige Messwerte

Die Tabelle zeigt gut die zentralen Eigenschaften der beiden untersuchten Technologien. SPDY reduziert die Anzahl an kByte, die an den Server gesendet werden müssen, drastisch und optimiert gleichzeitig die Anzahl der Pakete die für den Download der gleichen Nutzdaten wie bei HTTP nötig sind.

PageSpeed hingegen reduziert die Datenmenge, die der Browser herunterladen muss, und die Anzahl der gestellten Anfragen deutlich.

Auffällig ist auch, dass die manuell optimierte Seite über SPDY nur gut die Hälfte der Pakete in beide Richtungen benötigt als dies mit SPDY und PageSpeed der Fall ist. Und

dies obwohl die Anzahl der Anfragen bei der optimierten Seite höher ist. Dieser Fakt soll in der anschliessenden Diskussion noch einmal aufgegriffen werden.

Stellt man die Anzahl der vom Browser empfangenen Pakete zusammen mit den empfangenen kByte an Daten grafisch dar, ergibt sich folgendes Bild.

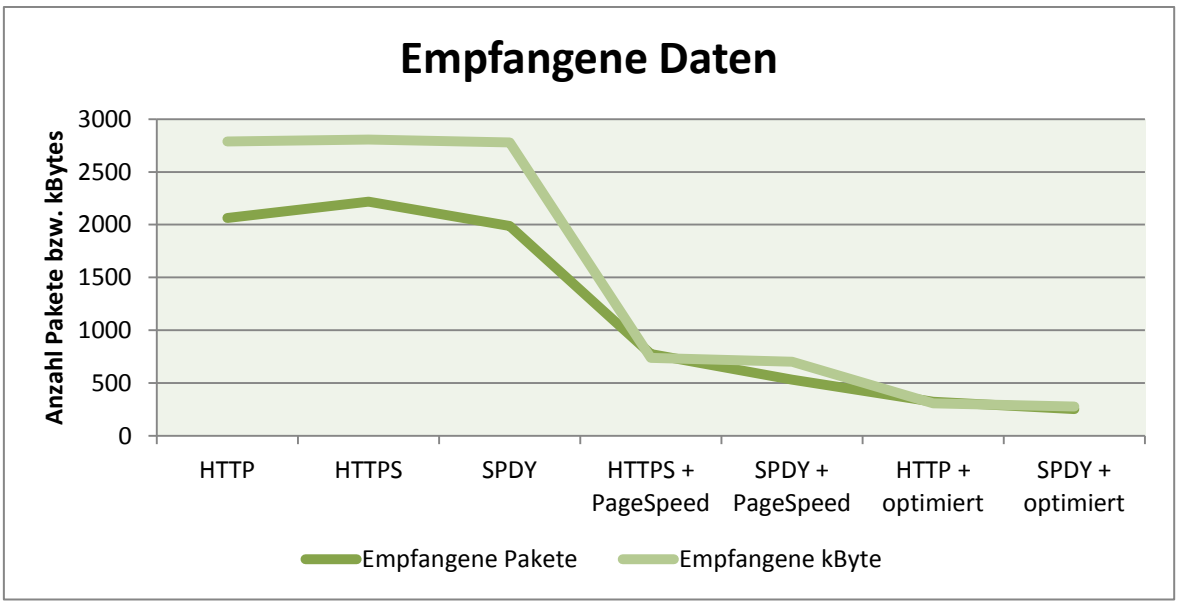


Abbildung 17: Graph der empfangenen Pakete und kByte

Widerholt man den Vorgang mit den gesendeten Daten, entsteht folgende Grafik:

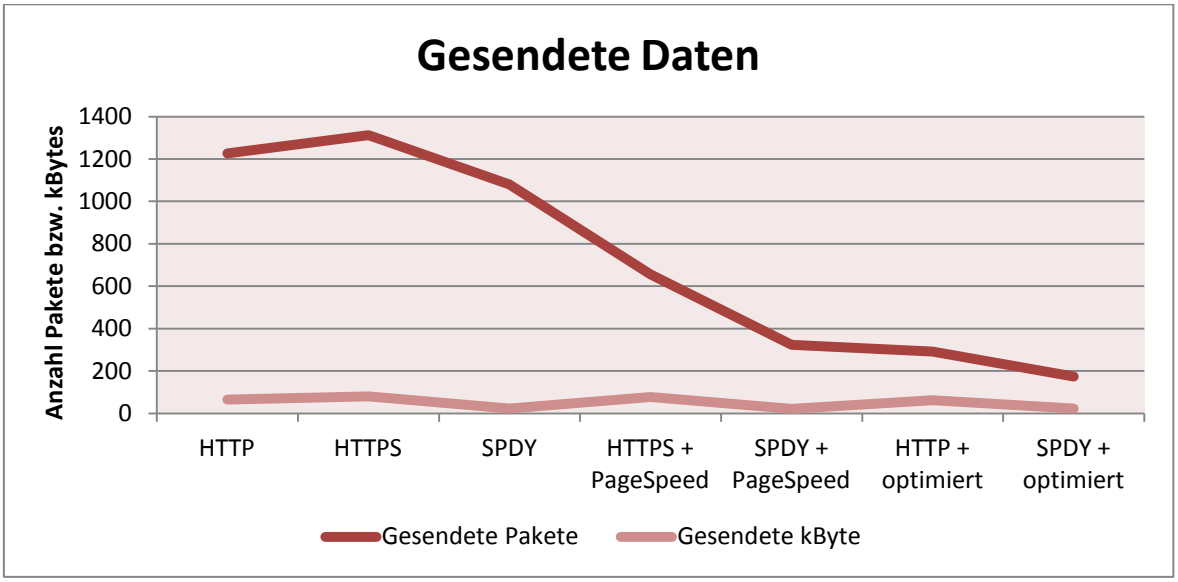


Abbildung 18: Graph der gesendeten Pakete und kByte

Auffallend ist hier vor allem, dass die Anzahl der kByte pro Paket in Senderichtung deutlich kleiner ist, als in Empfangsrichtung. Dies zeigt gut die Asymmetrie des HTTP-Protokolls die auch mit SPDY bestehen bleibt.

5.2 Interpretation und Diskussion

Die erhobenen Daten werden nun auf ihre Aussage hin interpretiert. Zusätzlich findet eine Diskussion über die Nützlichkeit der untersuchten Technologien statt und es wird ermittelt, ob die zu Beginn der Arbeit aufgestellten Fragestellungen beantwortet werden können.

5.2.1 SPDY

Wie Abbildung 13 und Abbildung 15 (Durchschnittliche Ladezeit bei voller und mobiler Bandbreite) zeigen, ist SPDY sehr effizient darin viele kleine Dateien rasch hintereinander vom Server zum Browser zu übertragen (vergleiche SPDY+PageSpeed und SPDY+optimiert gegenüber nur SPDY). Sind die Dateien grösser, verringert sich dieser Effekt. Dies liegt vor allem daran, dass SPDY mehrere sehr kleine Dateien in einem einzigen TCP-Paket übermitteln kann, was bei klassischem HTTP nicht möglich ist. Sind die Dateien grösser als ein einzelnes Paket, ist dies nicht mehr möglich und es ist ein ähnliches Übertragungsverhalten wie bei HTTP ersichtlich, wo dann die zur Verfügung stehende Bandbreite eine grössere Rolle spielt.
Was aber in jedem Fall durch SPDY deutlich optimiert wird, ist die Anzahl der vom Browser gesendeten Pakete und kByte. Durch die Komprimierung der HTTP-Header und der Bündelung mehrerer Anfragen fällt die hochgeladene Datenmenge gegenüber HTTPS auf etwa einen Viertel.
Auch wenn SPDY alleine gegenüber HTTP im besten Fall (volle Bandbreite) „nur“ eine Verbesserung von 28% erreicht, ist nicht ausser Acht zu lassen, dass als zusätzlicher Bonus die Daten verschlüsselt übertragen werden. SPDY gleicht also sowohl bei grosser als auch limitierter Bandbreite mindestens den Overhead von SSL/TLS aus.

Dass drei der meistgenutzten Browser für die Desktopumgebung SPDY bereits unterstützen [16] und der Fakt, dass Google alle seine Dienstleistungen mit SPDY anbietet [15], sprechen deutlich für den Einsatz des Protokolls, auch wenn es offiziell immer noch den Status „experimentell“ hat.
Dadurch, dass die Daten mit TLS verschlüsselt werden, kann die „*Next Protocol Negotation*“ (NPN) eingesetzt werden. Dies verhindert, dass ein alter Browser, der noch nichts von SPDY weiss, einen Dienst plötzlich nicht mehr benutzen kann. Dank TLS sollte SPDY auch über die meisten Firmen-Proxy-Server hinweg funktionieren, da diese den Inhalt von HTTPS-Paketen im Normalfall nicht anschauen können.

Aus all diesen Gründen kann die Frage, ob SPDY trotz experimentellem Status bereits eingesetzt werden sollte, aus der Sicht des Autors mit Ja beantwortet werden.

5.2.2 PageSpeed

Bei den Ergebnissen der Tests mit voller Bandbreite fällt auf, dass HTTPS+PageSpeed beinahe keine Verbesserung (1.5%) gegenüber nur HTTPS bringt. Und dies obwohl die Anzahl der Anfragen und die zu herunterladende Gesamtdatenmenge deutlich reduziert sind. Die Erklärung dafür liegt in der Bandbreite. Diese ist genügend gross, dass die Reduktion der heruntergeladenen Datenmenge auf fast einen Viertel keinen grossen Unterschied macht. Deutlich mehr Einfluss hat bei diesen Rahmenbedingungen die Round Trip Time. Der Browser muss immer noch 131 Anfragen stellen und erhält für jede davon ein oder mehrere TCP-Pakete. Auch die Tatsache, dass mit PageSpeed zwar 36 Anfragen weniger gestellt werden müssen, aber die vom Browser gesendete Gesamtdatenmenge nicht wesentlich geringer ist als mit HTTPS, unterstützt diese Erklärung.
Betrachtet man die Ladezeiten bei mobiler Bandbreite wird klar, dass die Download-Geschwindigkeit klein genug ist, dass die Reduktion der herunterzuladenden Datenmenge einen grösseren Einfluss hat als die Round Trip Time. Bei geringeren Bandbreiten kann PageSpeed sein Potential also deutlich besser entfalten.

Die Fragestellung, ob mod_spdy und mod_pagespeed gleichzeitig eingesetzt werden können, kann anhand der Ergebnisse mit einem klaren Ja beantwortet werden. Arbeiten beide Module zusammen, reduziert sich die Ladezeit sowohl bei voller als auch bei mobiler Bandbreite deutlich. Aufgrund ihrer unterschiedlichen Ansätze gleichen sie gegenseitig ihre Schwächen aus. SPDY ist nicht sehr effizient bei grossen Dateien, darum verkleinert PageSpeed diese. PageSpeed wiederum produziert zwar kleinere Dateien, ist jedoch bei einer grossen Anzahl derer an die Limitationen von HTTP und der Round Trip Time gebunden. Hier greift SPDY mit der Bündelung von Anfragen unter die Arme und optimiert so die Übertragung.

Wichtig für die Interpretation der Ergebnisse von PageSpeed ist es zu erwähnen, dass eine der Stärken des Moduls, nämlich die Optimierung des Cachings, bei den durchgeführten Tests vernachlässigt wurde. Nach jedem Seitenaufruf wurde der Zwischenspeicher geleert. Es ist also schwer zu sagen, welche Geschwindigkeitsvorteile bei weiteren Seitenaufrufen ohne dem Löschen des Caches zu beobachten wären.

Die letzte Fragestellung, ob PageSpeed eingesetzt werden soll, oder ob es sinnvoller ist eigene Optimierungen vorzunehmen, kann nicht eindeutig für die eine oder die andere Teilfrage mit Ja beantwortet werden. Denn der Einsatz von PageSpeed lohnt sich durch-

aus, wenn man keine Möglichkeit zur Anpassung seines Internetauftritts hat oder wenn man mit wenig Aufwand eine Verbesserung erzielen möchte.
Da aber PageSpeed die Bedeutung der einzelnen Inhalte nicht kennt, wählt es oftmals einen defensiven Ansatz bei der Optimierung. Ein Entwickler, der die Webseite genau kennt, kann oftmals mehr Geschwindigkeit erzielen, wenn er die Tipps der PageSpeed Rules selbst umsetzt.
Daher ist sowohl der Einsatz von PageSpeed als auch die manuelle Optimierung sinnvoll. Es kommt dabei vor allem auf die Situation und die Rahmenbedingungen an.

5.2.3 Manuelle Optimierungen

Aus den gemessenen Daten geht klar hervor, dass die optimierte Seite im Zusammenspiel mit SPDY der „Testsieger“ dieser Arbeit ist.
Ein Grund dafür ist, dass die Vorschau-Bilder durch eine noch stärkere Komprimierung nun so klein sind, dass die meisten davon in ein einzelnes TCP-Paket passen. Dies ist aber nur möglich, wenn der Entwickler weiss, dass es sich lediglich um eine Vorschau handelt und darum die Qualität nicht sehr wichtig ist. PageSpeed hat hier einen defensiveren Ansatz gewählt, was zur Folge hat, dass pro Bild mehrere TCP-Pakete nötig sind.
Was weiter auffällt ist, dass die vom Browser gesendete Gesamtdatenmenge bei der optimierten Seite fast gleich gross ist wie bei PageSpeed, obwohl 25 Anfragen mehr gemacht werden müssen. Dies kann unter Anderem damit erklärt werden, dass PageSpeed für seine Optimierungsmechanismen einen deutlich längeren Dateinamen generiert, der bei jeder Anfrage übermittelt werden muss. Somit passen auch mit dem Einsatz von SPDY weniger Anfragen in ein einzelnes TCP-Paket.

Als Schlussfolgerung kann festgestellt werden, dass sich der Aufwand, eine Webseite anzupassen, durchaus lohnt. Ein Entwickler hat dadurch auch die bessere Kontrolle darüber, wie seine Seite aufgebaut ist und muss sich nicht auf die Algorithmen von PageSpeed verlassen.
Aber natürlich ist es auch nicht ausgeschlossen, dass man beide Wege wählt. Man optimiert das, was mit vertretbarem Aufwand möglich ist und über lässt den „Feinschliff“ dann dem PageSpeed-Modul. Somit kann ein Mittelweg zwischen voller Kontrolle und maximaler Performanz gefunden werden.

5.2.4 Zusammenfassung

Zusammenfassend kann behauptet werden, dass die beiden untersuchten Technologien, neben einigen unerwarteten aber erklärbaren Effekten, das halten was sie versprechen. Dass die beiden Module so gut zusammen arbeiten und sich gegenseitig ergänzen ist aber zumindest für den Autor ein unerwartet positives Ergebnis. Dennoch ist es die Meinung des Autors, dass sich ein Web-Entwickler selbst so gut wie möglich um die Geschwindigkeit seiner Webseite kümmern und dies nicht ausschliesslich den Modulen von Google überlassen sollte.

6 Zusammenfassung und Ausblick

Mit dem Blick zurück auf die gesteckten Ziele der Arbeit kann festgestellt werden, dass alle Fragestellungen beantwortet werden konnten:

- Welche Reduktion der Ladezeiten kann mit dem Einsatz der Technologien erreicht werden?

  Bei gleichzeitigem Einsatz von mod_spdy und mod_pagespeed kann eine Reduktion der Ladezeit bei voller Bandbreite von 49% und bei mobiler Bandbreite von 67% gegenüber HTTP beobachtet werden. Werden manuelle Optimierungen vorgenommen, sinkt die Ladezeit über SPDY bei mobiler Bandbreite sogar auf 79% gegenüber HTTP.
- Können beide Apache-Module gleichzeitig betrieben werden?

  Die Module arbeiten problemlos zusammen und ergänzen sich sogar gegenseitig bei ihrer Aufgabe.
- Ist der Einsatz von SPDY in einer produktiven Umgebung zu diesem Zeitpunkt schon sinnvoll?

  Bei den durchgeführten Tests konnten keine Probleme beim Einsatz von SPDY festgestellt werden. Die Version 3 von SPDY läuft stabil und wird bereits von einigen grösseren Webseiten angeboten. Für Browser, die SPDY nicht unterstützen, gibt es einen Mechanismus, damit diese über ein etabliertes Protokoll wie HTTP mit dem Server kommunizieren können.
- Ist der Einsatz von PageSpeed empfehlenswert oder kann ein Webseitenbetreiber selbst Verbesserungen vornehmen?

  Der Einsatz von PageSpeed kann dort empfohlen werden, wo keine Möglichkeit besteht, manuell Optimierungen vorzunehmen. Wer eine Webseite selbst entwickelt erzielt aber mit eigenen Verbesserungen das bessere Ergebnis bei der Ladezeit. Es ist aber auch eine Kombination aus dem Einsatz von PageSpeed und eigenen Optimierungen denkbar.

Von den beiden untersuchten Apache-Modulen wurde sowohl die Funktionsweise als auch der praktische Nutzen anhand mehrerer Beispiele und Tests aufgezeigt. Die Installation und Konfiguration der Module gestaltet sich vergleichsweise einfach, daher sollte ein Webseiten-Administrator mit grundlegenden Linux-Kenntnissen keine Probleme damit haben.

Abschliessend kann behauptet werden, dass die Ziele der Arbeit vollumfänglich erreicht wurden.

6.1 Ausblick

Ein Thema, das in dieser Arbeit nur teilweise untersucht wurde, in der Zukunft aber immer wichtiger werden wird, ist der Einsatz der Optimierungstechnologien auf internetfähigen Smartphones (und nicht nur die Simulation einer eingeschränkten Bandbreite). Die Frage, ob man mit den beiden Modulen SPDY und PageSpeed einen Proxy-Server bauen könnte, der beliebige Webseiten für Smartphones optimiert, wäre durchaus eine Frage für eine zukünftige Arbeit. Ein Teil dieser Arbeit wurde in [21] bereits durchgeführt, doch die Kombination mit SPDY wäre durchaus interessant zu sehen. Vor allem, da es nun auch SPDY-fähige Browser für Smartphones gibt [16].

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] **Wikipedia.** HTTP. *Wikipedia.* [Online] January 06, 2013. [Cited: January 12, 2013.] http://en.wikipedia.org/wiki/HTTP.

[2] **Stefanov, Stoyan.** *Web Performance Daybook, Vol. 2.* s.l. : O'Reilly, 2012.

[3] **Belshe, Mike.** More Bandwith Doesn't Matter (Much). *Mike's Lookout.* [Online] May 24, 2010. [Cited: August 19, 2012.] http://www.belshe.com/2010/05/24/more-bandwidth-doesnt-matter-much/.

[4] **Hoelzle, Urs and Coughran, Bill.** Let's make the web faster. *Google Official Blog.* [Online] June 24, 2009. [Cited: January 12, 2013.] http://googleblog.blogspot.ch/2009/06/lets-make-web-faster.html.

[5] **Peon, Roberto and Belshe, Mike.** SPDY Protocol - Draft 1. *The Chromium Projects.* [Online] November 11, 2009. [Cited: January 01, 2013.] http://dev.chromium.org/spdy/spdy-protocol/spdy-protocol-draft1.

[6] SPDY: An experimental protocol for a faster web. *The Chromium Project.* [Online] 2012. [Cited: January 01, 2013.] http://dev.chromium.org/spdy/spdy-whitepaper.

[7] **Amazon.** Kostenloses Nutzungskontingent für AWS. *Amazon web services.* [Online] 2012. [Zitat vom: 26. Dezember 2012.] http://aws.amazon.com/de/free/.

[8] **Google.** mod_spdy. *Google Developers.* [Online] May 01, 2012. [Cited: December 26, 2012.] https://developers.google.com/speed/spdy/mod_spdy/.

[9] —. mod_spdy. *mod-spdy.* [Online] 2012. [Cited: January 02, 2013.] http://code.google.com/p/mod-spdy/.

[10] **Belshe, Mike and Peon, Roberto.** SPDY Protocol - Draft 3. *The Chromium Projects.* [Online] August 2012. [Cited: November 4, 2012.] http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3.

[11] **Wireshark Foundation.** Display Filter Reference. *Wireshark.* [Online] June 2012. [Zitat vom: 02. January 2013.] http://www.wireshark.org/docs/dfref/.

[12] spdyshark. *SPDY Wireshark Plugin.* [Online] 06. June 2012. [Zitat vom: 02. January 2013.] http://code.google.com/p/spdyshark/wiki/BriefInstallInstructions.

[13] **Strom, Chris.** *The SPDY Book.* 2011.

[14] **Wikipedia.** SPDY. *Wikipedia.* [Online] December 19, 2012. [Cited: January 04, 2013.] http://en.wikipedia.org/wiki/SPDY.

[15] **Belshe, Mike.** SPDY Status. [Online] [Cited: January 04, 2013.] http://www.ietf.org/proceedings/80/slides/httpbis-7.pdf.

[16] Can I use SPDY networking protocol? *Can I use.* [Online] January 2013. [Cited: February 01, 2013.] http://caniuse.com/spdy.

[17] **Google.** PageSpeed Tools. *Google Developers.* [Online] June 09, 2012. [Cited: February 01, 2013.] https://developers.google.com/speed/pagespeed/.

[18] —. Web Performance Best Practices. *Google Developers.* [Online] March 28, 2012. [Cited: August 19, 2012.] https://developers.google.com/speed/docs/best-practices/rules_intro.

[19] —. mod_pagespeed. *Google Developers.* [Online] June 09, 2012. [Cited: August 19, 2012.] https://developers.google.com/speed/pagespeed/mod.

[20] —. mod_pagespeed Filters. *Google Developers.* [Online] December 17, 2012. [Cited: February 01, 2013.] https://developers.google.com/speed/docs/mod_pagespeed/filters.

[21] **Denis, Frank.** mod_pagespeed as a proxy for your phone. *Frank DENIS random thoughts.* [Online] November 09, 2012. [Cited: February 01, 2013.] https://00f.net/2012/06/02/mod-pagespeed-as-a-proxy-for-your-phone/.

[22] **Waters, Alec.** Decrypting SSL traffic with Wireshark, and ways to prevent it. *Wirewatcher.* [Online] July 20, 2010. [Cited: December 26, 2012.] http://wirewatcher.wordpress.com/2010/07/20/decrypting-ssl-traffic-with-wireshark-and-ways-to-prevent-it/.

Anhang

Anhang A: SSL-Daten mit Wireshark entschlüsseln

Wie im Kapitel 2.3.2 verdeutlicht wird, basiert SPDY im Zusammenhang mit Apache immer auf SSL, beziehungsweise auf dessen Nachfolger-Protokoll TLS, weil nur dieses einen Mechanismus hat, wie sich Server und Browser über unterstützte Protokolle austauschen können. Diese Tatsache hat zur Folge, dass die ausgetauschten Daten immer verschlüsselt sind und sich nicht einfach so mit einem Netzwerk-Analyse-Programm wie Wireshark (http://www.wireshark.org/) mitschneiden lassen.
Zu Testzwecken lässt sich aber im Apache ein selbst erstelltes und signiertes SSL-Zertifikat installieren, mit dessen Privatschlüssel Wireshark dann in der Lage ist, die übertragenen Daten zu entschlüsseln. Es wird im Folgenden davon ausgegangen, dass grundlegende Kenntnisse von SSL und dem Umgang mit Wireshark vorhanden sind.

Server konfigurieren

Auf dem Server müssen folgende Schritte durchgeführt werden [13 S. 108-110]:

```
## OpenSSL installieren
$ sudo apt-get install openssl

## Verzeichnis für Zertifikate vorbereiten
$ sudo mkdir /etc/apache2/ssl
$ cd /etc/apache2/ssl

## Privaten Schlüssel generieren
$ sudo openssl genrsa -out bachelor.aws.guggero.org.key 1024

## Signierungs-Anfrage erstellen
$ sudo openssl req -new -key bachelor.aws.guggero.org.key ↵
-out bachelor.aws.guggero.org.csr
```

Bei dem letzten Befehl muss eine Reihe von Angaben eingetippt werden. Diese sind für diesen Zweck irrelevant, bis auf den „Common Name“, der genau der DNS-Adresse des Servers entsprechen muss:

```
…
Country Name (2 letter code) [AU]:CH
State or Province Name (full name) [Some-State]:Bern
Locality Name (eg, city) []:Bern
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Personal
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:bachelor.aws.guggero.org
Email Address []:oligugger@gmx.ch
…
```

Im letzten Schritt wird das Zertifikat mit dem eigenen Schlüssel signiert und der Apache so konfiguriert, dass er es für SSL-Anfragen verwendet:

```
## Zertifikat mit Privatschlüssel signieren
$ sudo openssl x509 -req -days 365 -in bachelor.aws.guggero.org.csr ↵
-signkey bachelor.aws.guggero.org.key -out bachelor.aws.guggero.org.crt

## Apache-Konfiguration anpassen
$ sudo vi /etc/apache2/sites-available/default-ssl

## Folgende drei Zeilen müssen in der Datei default-ssl angepasst bzw.
## hinzugefügt werden:
        SSLCertificateFile      /etc/apache2/ssl/bachelor.aws.guggero.org.crt
        SSLCertificateKeyFile   /etc/apache2/ssl/bachelor.aws.guggero.org.key
        SSLCipherSuite          RC4-SHA

## SSL-Konfiguration aktivieren und Server neu starten
$ sudo a2ensite default-ssl
$ sudo service apache2 restart
```

Die Zeile “SSLCipherSuite” dient dazu, dass Client und Server immer die Verschlüsselungs- und Hashing-Algorithmen verwenden, die Wireshark in auch entschlüsseln kann. In [13 S. 119] wird zwar die Cipher-Suite „DES-CBC3-SHA“ empfohlen, diese hat aber mit Firefox 17 und Wireshark 1.7.1 nicht zufriedenstellend funktioniert, weil jeweils nur die Antworten des Servers entschlüsselt werden konnten, die Anfragen jedoch nicht. Daher wurde die in [22] empfohlene Cipher-Suite „RC4-SHA“ verwendet.

Wireshark auf dem Client konfigurieren

Zuerst muss der im vorherigen Schritt erstellte Privatschlüssel (`bachelor.aws.guggero.org.key`) auf den Client-Computer kopiert werden.
Danach kann Wireshark so konfiguriert werden, dass er für eine bestimmte IP-Adresse und ein bestimmtes Protokoll diesen Schlüssel benutzt um die SSL-Daten zu entschlüsseln. Die Anleitung aus [13 S. 145-149] wurde so angepasst, damit sie mit der Version 1.7.1 von Wireshark übereinstimmt.

1. Unter *Edit -> Preferences* auf der linken Seite das Protokoll „SSL“ selektieren und auf den Button „Edit…“ bei „RSA keys list“ klicken.
2. Im neu geöffneten Fenster ein neues Profil per „New“ hinzufügen und im Dialog die entsprechenden Daten eingeben.

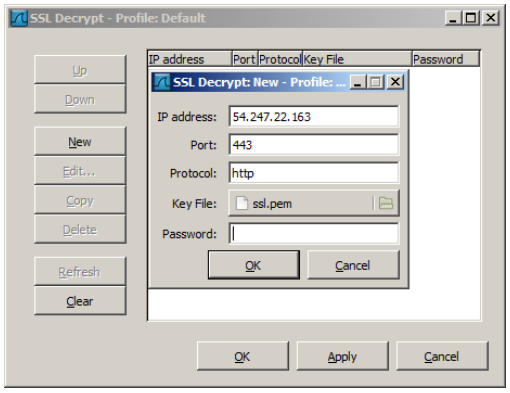


Abbildung 19: Erfassen eines neuen SSL-Schlüssels im Wireshark

3. Beide Dialogfenster mit OK bestätigen.
4. Zurück bei den Einstellungen zu SSL muss bei „SSL debug file“ eine Datei angegeben werden, wo Wireshark seine Debug-Meldungen hineinschreiben kann. Wird keine Datei angegeben, funktioniert die Entschlüsselung nicht.
5. Mit OK bestätigen und Wireshark neu starten

Sind diese Schritte ausgeführt, können SSL-Pakete inspiziert werden:

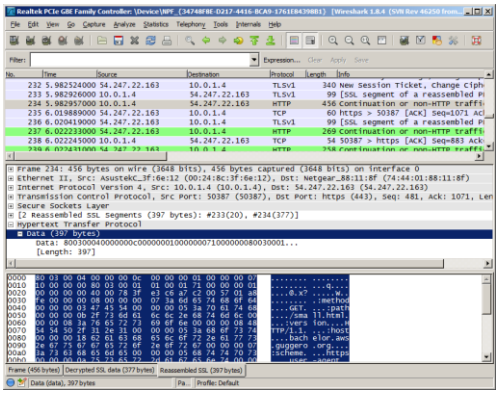


Abbildung 20: Entschlüsselte SSL-Pakete im Wireshark

Anhang B: Ermittlung der Bandbreite und Latenz zwischen Server und Client

Damit die ermittelten Messdaten in einen Kontext gebracht werden können, ist es sinnvoll, die maximal zur Verfügung stehende Bandbreite und die durchschnittliche Latenz zwischen Server und Client zu ermitteln.
Die Art der Arbeit legt es nahe, die Bandbreite über HTTP zu messen, mit mehreren parallelen Downloads. Dies ist zwar nur eine Annäherung an die effektiv mögliche Übertragungsrate von TCP, die Genauigkeit sollte aber für diese Arbeit genügen.

Sowohl auf dem Client als auch auf dem Server wird eine 100 MB grosse Datei mit dem folgenden Befehl angelegt:

```
## Random-Datei mit 100MB erstellen
$ dd if=/dev/urandom of=bandwidth_test.100mb.bin bs=1024 count=102400
```

Danach wird diese Datei jeweils von der Gegenseite mehrfach parallel heruntergeladen.
Dafür bietet sich das im Apache beiliegende Test-Programm *ab* (Apache Benchmark) an, welches für diesen Zweck entwickelt wurde.
Die Test-Datei wird von beiden Seiten jeweils 16 Mal heruntergeladen, wobei immer 8 parallele Downloads laufen sollen:

Beispiel Download von Server auf Client (Ausgabe gekürzt auf relevante Informationen):

```
## Apache Benchmark aufrufen (ausführen auf Client!)
$ ab -n 16 -c 8 http://bachelor.aws.guggero.org/bandwidth_test.100mb.bin
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
…
Benchmarking bachelor.aws.guggero.org (be patient).....done
…
Concurrency Level:      8
…
HTML transferred:       1677721600 bytes
Requests per second:    0.09 [#/sec] (mean)
Time per request:       91249.652 [ms] (mean)
Time per request:       11406.207 [ms] (mean, across all concurrent requests)
Transfer rate:          8977.59 [Kbytes/sec] received
…
```

Der Server muss also mindestens eine **Upload-Rate von 8.98 MByte/s (~71 MBit/s)** haben.

Beispiel Upload von Client auf Server (Ausgabe gekürzt auf relevante Informationen):

```
## Apache Benchmark aufrufen (ausführen auf Server!)
$ ab -n 16 -c 8 http://CENSORED/bandwidth_test.100mb.bin
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
…
Benchmarking CENSORED (be patient).....done
…
Concurrency Level:      8
…
HTML transferred:       1677721600 bytes
Requests per second:    0.06 [#/sec] (mean)
Time per request:       145053.893 [ms] (mean)
Time per request:       18131.737 [ms] (mean, across all concurrent requests)
Transfer rate:          5647.57 [Kbytes/sec] received
…
```

Der Server muss also mindestens eine **Download-Rate von 5.65 MByte/s (~45 MBit/s)** haben.

Für die Ermittlung der Latenz (beziehungsweise der Round Trip Time) bietet sich das Kommando *ping* an. Wird dies auf dem Client ausgeführt ergeben sich folgende Werte:

```
## 10 ICMP-Pakete an Server senden
$ ping -n 10 bachelor.aws.guggero.org
Pinging ec2-54-247-22-163.eu-west-1.compute.amazonaws.com [54.247.22.163] with
32 bytes of data:
Reply from 54.247.22.163: bytes=32 time=30ms TTL=48
…
Reply from 54.247.22.163: bytes=32 time=30ms TTL=48
Ping statistics for 54.247.22.163:
    Packets: Sent = 10, Received = 10, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 30ms, Maximum = 30ms, Average = 30ms
```

Die Round Trip Time zwischen Client und Server beträgt also **30 Millisekunden**.

Anhang C: Ermittlung und Simulation mobiler Bandbreite und Latenz

Damit eine Umgebung simuliert werden kann, die den Bedingungen in einem mobilen Datennetz entspricht, müssen diese zuerst ermittelt werden.
Dazu wurden auf dem Orange UMTS-Netz mit einem iPhone 5 und der App „Speed Test“ zehn Messungen durchgeführt:

	Ping (ms)	*Download (MBit/s)*	*Upload (MBit/s)*
	69	4.54	3.69
	79	4.14	2.47
	77	2.59	4.04
	76	3.82	3.4
	77	3.37	2.48
	77	3.46	2.45
	79	1.44	2.44
	141	4.13	1.6
	79	3.89	1.83
	78	4	1.77
Average	**83.2**	**3.538**	**2.617**
Avg. Variation	11.56	0.6584	0.6558

Tabelle 14: Ergebnisse Speed Test mit Orange UMTS

Für die Simulation eines mobilen Netzwerkes werden also folgende Werte verwendet:

- Zeitverzögerung 83 ms abzüglich bestehender RTT (30 ms) = **53 ms**
- Variation der Latenz 11 ms
- Download-Rate **3.5 MBit/s**
- Upload-Rate **2.6 MBit/s**

Da das Drosseln der Internet-Verbindung auf einem mobilen Endgerät oder unter Windows eher umständlich ist, wurde dies direkt auf dem Server durchgeführt. Denn unter Linux ist dies mit dem Programm *tc* (Traffic Control) relativ einfach zu erreichen. Das folgende Script erleichtert die Ein- und Ausschaltung der Simulation. Natürlich sind die auf dem Endgerät gewünschten Up- und Download-Raten umzukehren, wenn man diese auf dem Server konfiguriert.

Benutzung:

```
## Starten der Bandbreiten-Simulation
$ ./traffic-control.sh start
## Stoppen der Bandbreiten-Simulation
$ ./traffic-control.sh stop
```

Gekürztes und angepasstes Shell-Script

```
#!/bin/bash
# original version available at
# http://www.topwebhosts.org/tools/traffic-control.php
# created by Scott Seong
# modified by Oliver Gugger, added package delyay

# Name of the traffic control command.
TC=/sbin/tc
# The network interface we're planning on limiting bandwidth.
IF=eth0
# Download limit (in mega bits)
DNLD=2.6mbit          # DOWNLOAD Limit
# Upload limit (in mega bits)
UPLD=3.5mbit          # UPLOAD Limit
# Packet delay (delay, variation)
DELAY="53ms 11ms"
# IP address of the machine we are controlling
IP=$(/sbin/ifconfig $IF | grep 'inet addr:' | cut -d: -f2 | awk '{ print $1}')
# Filter options for limiting the intended interface.
U32="$TC filter add dev $IF protocol ip parent 1:0 prio 1 u32"
start() {
  $TC qdisc add dev $IF root handle 1: htb default 30
  $TC class add dev $IF parent 1: classid 1:1 htb rate $DNLD
  $TC class add dev $IF parent 1: classid 1:2 htb rate $UPLD
  $TC qdisc add dev $IF parent 1:2 handle 2:0 netem delay $DELAY
  $U32 match ip dst $IP/32 flowid 1:1
  $U32 match ip src $IP/32 flowid 1:2
}
stop() {
  # Stop the bandwidth shaping.
  $TC qdisc del dev $IF root
}
case "$1" in
  start)
    echo -n "Starting bandwidth shaping: "
    start
    echo "done"
    ;;
```

```
  stop)
    echo -n "Stopping bandwidth shaping: "
    stop
    echo "done"
    ;;
esac
exit 0
```

Anhang D: Benchmarking mit Chrome und Page Benchmarker

Für die Durchführung der Ladezeiten-Tests wurde Google Chrome mit der frei erhältlichen Erweiterung „Page Benchmarker“ (erhältlich über den in Chrome eingebauten „Extensions Store“) verwendet. Damit die Erweiterung alle benötigten Werte ermitteln kann, muss Chrome aber mit zwei Kommandozeilen-Optionen gestartet werden (siehe http://code.google.com/p/chromium/issues/detail?id=97101), die den Chrome anweisen, diese Daten auch aufzuzeichnen und der Erweiterung zur Verfügung zu stellen:

```
## Chrome mit Benchmarking-Optionen starten
C:\ "C:\PATH_TO_CHROME_INSTALLATION\chrome.exe" --enable-benchmarking ↵
--enable-stats-table
```

Leider wurde die Erweiterung seit längerer Zeit nicht weiter entwickelt und ist daher immer noch nur für die Version 2 von SPDY optimiert. Der Haken „Enable Spdy?“ bewirkt darum, dass zwar SPDY benutzt wird, jedoch nur in Version 2, obwohl der Server die Version 3 anbietet. Dieser Umstand kann aber mit einem kleinen Eingriff in den Code der Erweiterung behoben werden. Wird nämlich SPDY im Programm-Code weder aktiviert noch deaktiviert, werden automatisch die Standard-Einstellungen verwendet. Das bedeutet in diesem Fall, dass Chrome auch SPDY 3 benutzt, wenn der Server dies anbietet.
Folgende vier Zeilen müssen in der Datei *background.js* auskommentiert werden:

C:\Users\<username>\AppData\Local\Google\Chrome\User Data\Default\Extensions\<extension ID>\1.3.4.0_0\background.js:

```
    /*if (window.enableSpdy) {
      chrome.benchmarking.enableSpdy(true);
    } else {
      chrome.benchmarking.enableSpdy(false);
    }*/
```

Damit wird zwar der Schalter „Enable Spdy?“ komplett ignoriert, da aber das Ein- und Ausschalten von SPDY für die Tests sowieso auf dem Server vorgenommen werden, stellt dies kein Problem dar.

Selbständigkeitserklärung

„Ich erkläre hiermit, dass ich diese Thesis selbständig verfasst und keine andern als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche kenntlich gemacht. Ich versichere zudem, dass ich bisher noch keine wissenschaftliche Arbeit mit gleichem oder ähnlichem Inhalt an der Fernfachhochschule Schweiz oder an einer anderen Hochschule eingereicht habe. Mir ist bekannt, dass andernfalls die Fernfachhochschule Schweiz zum Entzug des aufgrund dieser Thesis verliehenen Titels berechtigt ist."

Ort, Datum, Unterschrift