Aufbau, Einsatz und Test von SPDY

Ausarbeitung für das Studienfach
„Ultra Large Scale Systems“
an der Hochschule der Medien Stuttgart

von
Pascal Naujoks
pn012@hdm-stuttgart.de | Matrikel-Nr.: 23397

Stuttgart, Juni 2012

Abstract

Das die Ladezeit einen enormen Einfluss auf den Erfolg einer Webseite hat ist längst kein Geheimnis mehr. Verschiedene Studien sprechen hier, je nach Branche und Art der angestrebten Conversion, auf Rückläufe von bis zu 20% bei gerade mal einer halben Sekunde längeren Ladezeit der Webseite [TPOWP]. Aus diesem Grund befassen sich weitere Studien mit der Clientseitigen [CSPOZ] und Serverseitigen [WEOPC] Optimierung von Webseiten. Auf Clientseite gibt es verschiedene Möglichkeiten wie Webseiten Betreiber ihre Seiten optimieren können. Da wären unter anderem die Reduzierung von HTTP-Requests, das Minifiying von JavaScript und CSS-Dateien sowie das Pre- und Lazyloading von Inhalten. Serverseitig gibt es die Möglichkeit die Inhalte einer Webseite für die Übertragung zum Client mittels mod_deflate bzw. mod_gzip zu komprimieren und somit die Ladezeit zu reduzieren.

Alle aufgeführten Maßnahmen der Client- und Serverseitigen Optimierung sowie weitere Techniken können mit Googles neuem, TCP-basierenden Protokoll „SPDY“ *[ˈspiːdi]* abgelöst werden wenn Server und Client das Protokoll unterstützen. Google verspricht in diesem Falle eine Reduktion der Ladezeit von bis zu 64% [SWIPA].

In dieser Ausarbeitung wird der Aufbau und die einzelnen Komponenten von SPDY erläutert sowie auf die Client- und Serverseitige Installation von SPDY eingegangen. Der letzte Teil der Ausarbeitung widmet sich den von Google angegeben Labormesswerten sowie den Ergebnissen eines eigens durchgeführten Feldtests mit SPDY. Dieser Vergleich stellt sich der Frage, ob die von Google angegebenen Messwerte realistisch sind und sich außerhalb des Google Labors im Internet beweisen können.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Ziele von SPDY

Das Hauptziel des Open-Source Projektes SPDY ist es, die Verzögerung beim Ausliefern von Webseiten zu minimieren. Google strebt hierbei eine Reduzierung der Ladezeit beim Client von 50% an. Dies soll durch mehrere gleichzeitige HTTP Anfragen über eine einzige TCP-Verbindung, die Komprimierung von HTTP-Headern und dem entfernen von unnötigen (doppelten) HTTP-Headern erfolgen (siehe Kapitel 2, Aufbau und Bestandteile).

Die Implementierung soll Serverseitig bei der Webserver Anwendung und Clientseitig beim Browser erfolgen (siehe Kapitel 3.2, Installation). Somit sollen bei der Aktivierung von SPDY keine Änderungen am Quellcode von Webseiten notwendig sein.

Als Transport Layer soll SPDY das bestehende TCP nutzen und als Transportprotokoll SSL (siehe Kapitel 2.1, Aufbau von SPDY).

Weiterhin soll SPDY es Webservern ermöglichen unaufgefordert Nachrichten an die verbundenen Clients zu senden (Push). Dies soll die Netzwerklast reduzieren, indem unnötige regelmäßige Pull-Anfragen von den Clients an den Webserver vermieden werden (siehe Kapitel 2, Aufbau und Bestandteile). Dieses verhalten würde das bisher genutzte Preloading bzw. per JavaScript aufgerufene Lazyloading von Inhalten ablösen.

2 Aufbau und Bestandteile von SPDY

2.1 Aufbau von SPDY

SPDY setzt als Session-Layer auf SSL auf und nutzt dabei auf Applikationsebene die regulären HTTP-Befehle GET und POST.

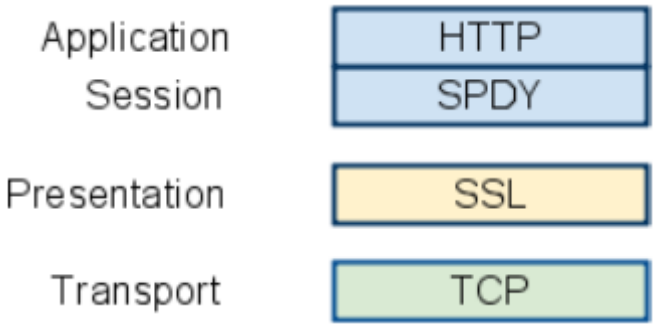

Abbildung 1: SPDY Schichtenmodell [SWIPA]

Obwohl SSL bei einer normalen HTTPS-Verbindung als Verschlüsselungsprotokoll zu sehen ist, wird es bei SPDY zusätzlich als Transportprotokoll genutzt. Dies ist nötig um sicherzustellen, dass die SPDY-Datenpakete nicht durch Proxys aufgeteilt oder anderweitig unbrauchbar gemacht werden.

Im Folgenden die wesentlichen Bestandteile von SPDY.

2.2 Multiplexing Streams

Beim HTTP-Pipelining können, im Gegensatz zu regulären HTTP, mehrere Anfragen gleichzeitig abgeschickt werden bevor die Verbindung wieder geschlossen wird. HTTP-Pipelining ist verfügbar seit HTTP/1.1 und muss sowohl vom Webserver als auch vom Client (Browser) unterstützt werden. Den Zeitvorteil und das Schema von HTTP-Pipelining zeigt die folgende Abbildung 2 aus dem Wikipedia Artikel zu HTTP-Pipelining[1] recht gut:

[1] http://de.wikipedia.org/wiki/HTTP-Pipelining

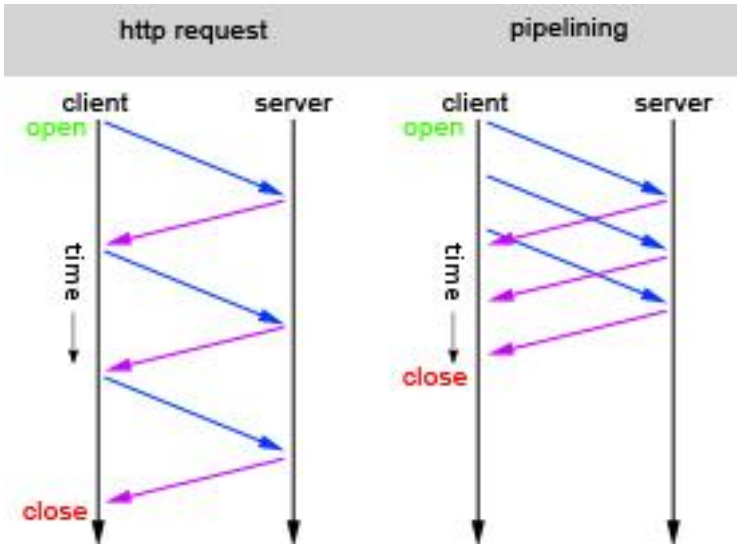

Abbildung 2: Vergleich regulärer HTTP-Request mit HTTP-Pipelining

Aktuell wird HTTP-Pipelining von keinem gängigen Browser standardmäßig unterstützt. Entweder ist es nicht implementiert oder muss erst manuell aktiviert werden [HTTPP]. Der Grund dafür liegt unter anderem im sogenannten „Head-of-line blocking“ (holb) Problem der parallelisierter Kommunikation beim HTTP-Pipelining. Beim HTTP-Pipelining müssen Anfragen in der Reihenfolge zum Client zurückgeschickt werden, in der sie abgeschickt wurden. Ist eine Anfrage nun fehlerhaft oder hat eine lange Bearbeitungsdauer, so müssen alle folgenden Anfragen auf das Eintreffen der vorangegangen Anfrage warten. Die folgende eigene Abbildung 3 zeigt, dass die erste Antwort vom Server sehr lange dauert (roter Pfeil), und dass beim HTTP-Pipelining alle folgenden Anfragen erst nach dem Eintreffen der ersten Anfrage beim Client ankommen[2].

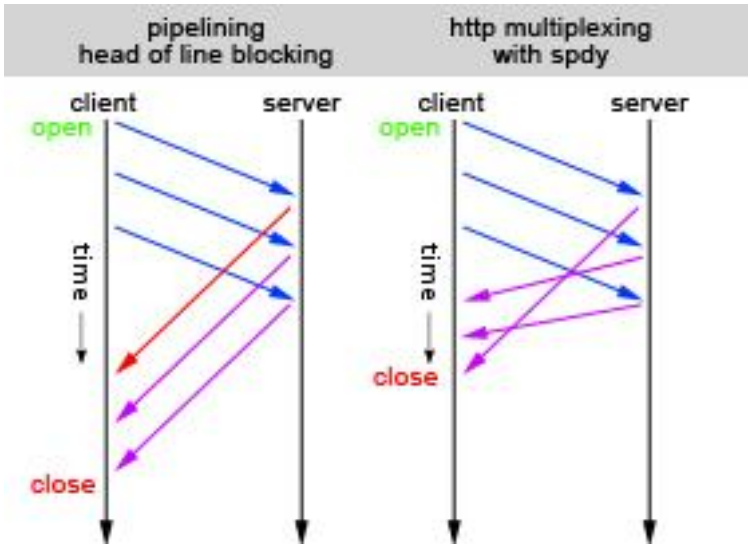

Abbildung 3: Vergleich holb-Problem beim HTTP-Pipelining mit HTTP-Multiplexing

[2] Aus Gründen der Übersichtlichkeit wurde das hierzu nötige Queuing der Anfragen in der Grafik weggelassen

Der Unterschied von HTTP-Multiplexing von SPDY zu regulärem HTTP-Pipelining ist, dass die Anfragen nicht mehr in derselben Reihenfolge ankommen müssen, in der sie abgeschickt wurden. SPDY übernimmt über die Stream-ID in den Datenpaketen (*Data frames)* die Zuordnung zur jeweiligen Anfrage [SPDYP].

Ein weiterer Vorteil vom SPDY HTTP-Multiplexing verfahren ist, dass die verschiedenen Datenpakete der einzelnen Anfragen nicht mehr als Ganzes abgeliefert werden müssen. Somit ist es dem Client möglich die Datenpakete von verschiedenen Anfragen in unterschiedlicher Reihenfolge anzunehmen und den Anfragen korrekt zuzuordnen.

2.3 Request prioritization

Mit SPDY kann jeder Anfrage eine Priorität zugeordnet werden (0 bis 7). Dies reduziert die „gefühlte Ladezeit“ beim Client.

Je nach Implementation von SPDY erfolgt dies beim Client (Browser) oder in der Anwendung. Im Java-basierten Web-Framework „Netty“ erfolgt das setzen der Priorität über den Header „X-SPDY-Priority“[3]. Im Apache Modul mod_spdy gibt es aktuell keine Möglichkeit die Priorität über die Webanwendung zu steuern.

Google empfiehlt den Browser Herstellern und Webentwicklern in ihren „SPDY Best Practices“ [SPDBP] die folgende Priorisierung von Anfragen:

```
html > js, css > *.
```

Dies bedeutet zuerst HTML, dann JavaScript und CSS und dann alle weiteren HTTP-Objekte.

2.4 HTTP header compression

Bei regulärem HTTP kann lediglich die Anfrage des Clients und die Antwort des Webservers komprimiert werden, nicht jedoch die übertragenen Header-Daten. Im Folgenden ein unkomprimierter Anfrage-Header, der bei jedem Aufruf eines HTTP-Objekts vom Client an den Server gesendet wird:

```
GET /spdy/spdy-whitepaper HTTP/1.1
Host: dev.chromium.org
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:12.0) Gecko/20100101 Firefox/12.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: __utma=166032427.948710308.1334673489.1336379659.1336394482.8; __utmz=166032427...
If-Modified-Since: Mon, 07 May 2012 10:30:30 GMT
If-None-Match: "1336386630169|#public|0|de|||0"
Cache-Control: max-age=0
```

Abbildung 4: Ein unkomprimierter HTTP Anfrage-Header

[3] Siehe http://netty.io/docs/stable/api/org/jboss/netty/handler/codec/spdy/SpdyHttpHeaders.html

SPDY komprimiert auch diese Daten und entfernt zusätzlich noch vor dem Senden unnötige Parameter. Da sich z.B. der User-Agent (Browser des Clients) während einer Sitzung nicht ändert kann dieser ab der zweiten Anfrage weggelassen werden. Damit der Webanwendung die Daten weiterhin zur Verfügung stehen füllt der Serverseitige SPDY-Cache die fehlenden Parameter wieder auf, bevor sie an den Webserver übergeben werden.

2.5 Server push

Mit dem von SPDY zur Verfügung gestellten Server push verfahren soll es möglich sein Daten an den Client zu schicken, ohne dass dieser danach gefragt hat. Ein typisches Scenario könnte das Vorladen eines Stylesheets sein. Währen das Stylesheet regulär erst vom Client angefragt würde wenn das HTML-Dokument vollständig vom Browser geladen und geparst wurde so kann das Stylesheet mit SPDY bereits mit dem angefragten HTML-Dokument mitgeschickt werden.

In der aktuellen Version von mod_spdy ist dies jedoch noch nicht implementiert:

> *„The current version of mod_spdy does not yet support server-pushed resources, but we are working on adding that functionality soon. Once it has been added, we will document here how you can use it to save on request round-trips.“* [4]

2.6 Server hint

Im Gegensatz zu Server push kann mit Server hint einem Client „empfohlen“ werden ein Dokument oder HTTP-Objekt vorzuladen. Dies kann bei im Web abgebildeten Prozeduren nützlich sein, die eine feste Rangfolge der besuchten Seiten vorgeben (wie z.B. ein Bestellprozess in einem Onlineshop).

Google selbst empfiehlt Server hint nicht beim initialen Laden einer Seite einzusetzen [SWIPA]. Gründe hierfür könnten sein, dass dies die Last auf den Server stark erhöht und es zudem in den meisten Fällen nur schwer möglich ist von der Startseite einer Webseite aus auf die vom Besucher besuchte Folgeseite zu schließen.

Server push und Server hint sind sogenannte „Advanced features“ von SPDY und müssen vom Betreiber einer Webseite gesondert konfiguriert werden.

[4] Quelle: http://code.google.com/p/mod-spdy/wiki/OptimizingForSpdy

3 Einsatz von SPDY

3.1 Webserver

SPDY wird aktuell (Stand April 2012) von offizieller Seite [MODSP] lediglich als Modul für den Webserver „*Apache*“ (ab Version 2.2.4) angeboten. Das Modul mod_spdy ist als 32 und 64bit Version für die Linux-Betriebssysteme Debian, Ubuntu, CentOS und Fedora verfügbar. Da SPDY die Verwendung von SSL voraussetzt (siehe Kapitel 2.1, Aufbau von SPDY) wird zusätzlich das Modul mod_ssl benötigt.

Durch die Open-Source Verfügbarkeit des Projekts sind durch die Community weitere Implementierungen in verschiedene Webserver vorgenommen worden. Diese sind aktuell:

- Jetty (ein Java-basierender Servlet-Container und Webserver)
- Ein Python basierender SPDY Server
- node.js (Serverseitiges Framework zum erstellen von Server und Konsolenanwendungen mit JavaScript5)
- Ruby

Hier fehlen jedoch bis Dato Implementierungen für die, laut Netcraft häufig eingesetzten, Webserver „*nginx*“ sowie Microsofts „*Internet Information Server*“ (IIS).

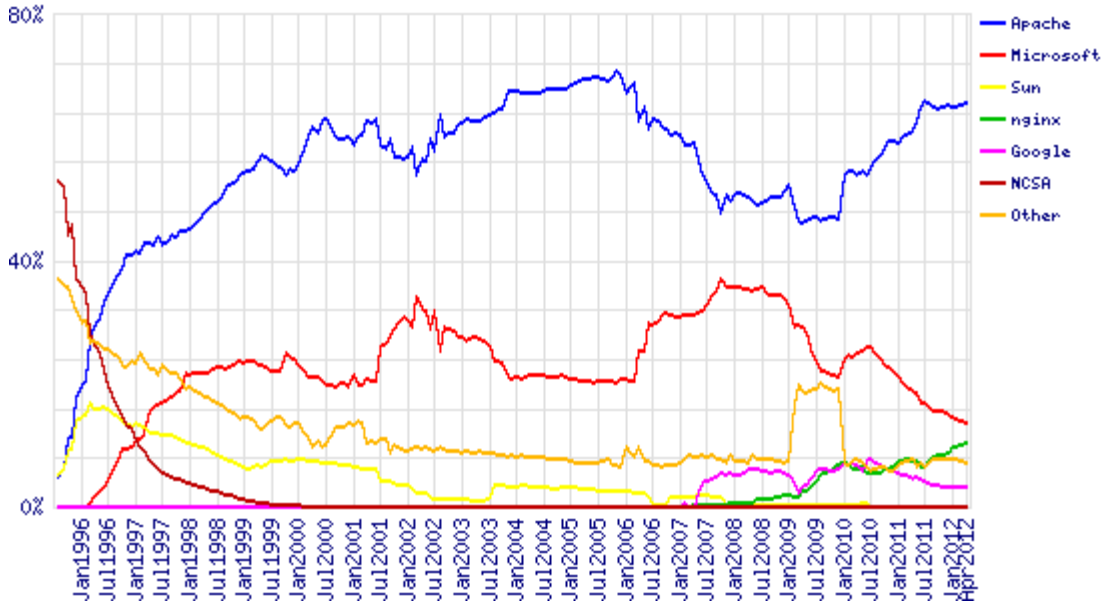

Abbildung 5: Verbreitungsgrad von Webservern [NETCR]

[5] Quelle: http://de.wikipedia.org/wiki/Node.js

Die Implementierung von SPDY in nginx ist ein von der Community häufig gefordertes Feature[6] und auch, laut einem Tweet[7] vom offiziellen nginx Twitter-Account, „für die nächsten Monat geplant“.

Microsoft verfolgt einen anderen Ansatz um ähnliche Ziele wie Google zu erreichen. Laut einer Meldung von heise [HEISE], die sich auf einen Blogpost von Microsoft beziehen [HTTP2], soll in Microsofts „HTTP 2.0“ z.B. die erforderliche SSL-Verschlüsselung von Webseiten nur optional sein und das von SPDY zur Verfügung gestellte Server-Push verfahren (siehe Kapitel 2.5, Server push) nicht implementiert werden. Eine konkrete Implementierung von Microsofts Vorhaben[8] ist bis Dato noch nicht verfügbar.

3.2 Installation

Die Installation von SPDY erfolgt unterschiedlich je nach gewählter Implementation. Im Rahmen dieser Ausarbeitung wurde mod_spdy für Apache auf einem Webserver installiert. Das Modul steht aktuell unter

```
https://developers.google.com/speed/spdy/mod_spdy/
```

zum Download für die Betriebssystem Debian, Ubuntu, CentOS und Fedora jeweils in der 32-bit und 64-bit Version zur Verfügung.

Das Testsystem ist ein 64-bit System mit Debian. Die Installation erfolgte über die Befehlsfolge:

```
root@pascal-web:/tmp# dpkg -i mod-spdy-*.deb
root@pascal-web:/tmp# apt-get -f install
```

Sowie einen Neustart des Apache Webservers:

```
root@pascal-web:/tmp# /etc/init.d/apache2 restart
```

Da mod_spdy einen Thread-Pool zur Verwaltung des HTTP-Multiplexings nutzt kann es nicht mit nicht-thread-sicheren Modulen wie mod_php arbeiten[9]. Es wird empfohlen mod_fcgi für die Verwendung von PHP einzusetzen. Da auf dem Testsystem das Serververwaltungs-Tool Plesk installiert ist erfolgte dies über die grafische Benutzeroberfläche von Plesk.

6 Siehe Kommentare auf http://googledevelopers.blogspot.de/2012/01/making-web-speedier-and-safer-with-spdy.html

7 Quelle: https://twitter.com/#!/nginxorg/status/150112670966747137

8 Entwurf für die IETF Network Working Group: http://www.ietf.org/id/draft-montenegro-httpbis-speed-mobility-01.txt

9 Siehe http://code.google.com/p/mod-spdy/wiki/KnownIssues

Weiterhin ist mod_spdy aktuell nicht mit der SSLRequireSSL Direktive von mod_ssl kompatibel und muss daher ausgeschaltet werden, bis der Fehler behoben wurde.

Ein einfaches Testskript für PHP verrät, ob die Installation von mod_spdy erfolgreich war:

```php
<?php

$spdy_version = getenv('SPDY_VERSION');

if ($spdy_version) {
    echo "SPDY enabled. Version: " . $spdy_version;
} else {
    echo "SPDY disabled.";
}

?>
```

Abbildung 6: PHP-Skript zur Prüfung einer SPDY Installation

Clientseitig ist SPDY in Googles Browser Chrome seit Version 11 aktiviert und nutzt bereits für 90% der Verbindungen[10] zu Googles Webservern SPDY statt regulärem HTTPS. In Mozilla's Firefox ist SPDY seit Version 11 implementiert, muss jedoch noch wie folgt von Hand aktiviert werden:

1. In die Adressleiste von Firefox „about:config“ eingeben
2. Nach „spdy“ suchen
3. Doppelklick auf „network.http.spdy.enabled“ um den Wert auf „true“ zu setzen
4. Neustart des Browsers

Ab Version 13 ist SPDY in Firefox standardmäßig aktiviert[11]. Um herauszufinden, ob SPDY für die aktuelle Verbindung verwendet wird kann das Firefox-Add-on „SPDY indicator“ Auskunft geben. In Google Chrome kann dies mithilfe des Befehls

- http//chrome//net-internals/#events&q=type:SPDY_SESSION%20is:active

erreicht werden.

Auf den mobilen Versionen der Browser ist SPDY ebenfalls implementiert oder wird in den nächsten Versionen nachgezogen. Im namenlosen Android Browser ist SPDY seit „Honeycomb“ aktiv.

Über die Implementation von SPDY in Microsofts Internet Explorer oder Apples Safari ist bisher nichts bekannt.

[10] Laut einem Tweet vom offiziellen Google Chromium Developer Twitter-Account: https://twitter.com/#!/ChromiumDev/statuses/56086061721464832

[11] Quelle: http://hacks.mozilla.org/2012/03/firefox-aurora-13-is-out-spdy-on-by-default-and-a-list-of-other-improvements/

3.3 Webseiten für SPDY optimieren

Wer seine Webseiten seither im regulären Stiel (siehe [CSPO] und [WEOP]) optimiert hat und jetzt auf SPDY umrüsten möchte muss jetzt ein paar Änderungen rückgängig machen, um das Potential von SPDY voll auszuschöpfen.

1. Wurde Domain Sharding betrieben oder Content Delivery Networks (CDN) eingesetzt um die durch Webbrowser limitierten zwei bis sechs gleichzeitigen Anfragen pro Verbindung künstlich zu erhöhen so sollte diese wieder von der gleichen Domain geladen werden. Das Multiplexing von Streams durch SPDY löst diesen Trick ab. Das durch CDN nötige Abfrage zusätzlicher Domains bedeutet auch einen zusätzlichen Zeitaufwand beim Anfragen der DNS-Einträge, der sich jetzt vermeiden lässt.
2. Wurde das Preloading verfahren zum vorladen von Inhalten verwendet so sollte jetzt stattdessen das von SPDY bereitgestellte Server Push verwendet werden. Da Webseiten mit SPDY base64 kodiert werden, würden die zusätzlichen Ressourcen die base64-Zeichenkette unnötig vergrößern.
3. Durch das HTTP-Multiplexing (siehe Kapitel 2.2, Multiplexing Streams) kann zukünftig auf Image-Sprites[12] verzichtet werden.

Weitere Tipps für Optimierungsmaßnahmen bei SSL-Zertifikate und Servereinstellungen bzgl. der TCP-Verbindungen findet man auf der Chromium Projektwebseite [CHBP].

[12] Siehe: http://css-tricks.com/css-sprites/

4 Performance von SPDY

In der Einleitung des Whitepapers [SWIPA] von SPDY heißt es, dass SPDY Webseiten zu einer Reduzierung der Ladezeit von bis zu 64% verhelfen kann. Google hat diese Messwerte unter Laborbedingungen erzielt, die im Folgenden näher erläutert werden. Um diese Messwerte zu verifizieren wurde ein Implementation von SPDY installiert, unter verschiedenen Bedingungen getestet und im darauf folgenden Kapitel dokumentiert.

4.1 Messungen von Google

Für die Tests im Google-Labor wurde ein modifizierter Google Chrome Client verwendet[13] und ein eigener Server geschrieben, der für die Tests optimiert war. Der Webserver läuft vollständig im Arbeitsspeicher und kann nach Google eigenen Angaben „HTTP und SPDY anfragen sehr effizient über TCP und SSL beantworten“.

Für die Tests hat Google 25 der „Top 100“ Webseiten mit unterschiedlichen Verbindungsgeschwindigkeiten heruntergeladen und die Downloadzeit mit regulärem HTTP und mit SPDY gemessen und Gegenübergestellt. Im Folgenden die durchschnittliche Ladezeit der Top 25 Webseiten mit SPDY im Vergleich zu regulärem HTTP.

	DSL 2 Mbps downlink, 375 kbps uplink		Cable 4 Mbps downlink, 1 Mbps uplink	
	Ø ms Ladezeit	**Speedup**	**Ø ms Ladezeit**	**Speedup**
HTTP	3111		2348	
SPDY basic multi-domain connection / TCP	2242	27,93%	1325	43,55%
SPDY basic single-domain connection / TCP	1695	45,51%	933	60,23%
SPDY single-domain + server push / TCP	1671	46,29%	950	59,51%
SPDY single-domain + server hint / TCP	1608	48,30%	856	63,53%
SPDY basic single-domain / SSL	1899	38,95%	1099	53,18%
SPDY single-domain + client prefetch / SSL	1781	42,74%	1047	55,40%

Tabelle 1: Google Messungen: Durchschnittle Ladezeit der "Top 25" Webseiten

Der Tabelle ist die angesprochene Reduktion der Ladezeit von 64% (63,53%) zu entnehmen.

[13] Sourcecode siehe http://src.chromium.org/viewvc/chrome/trunk/src/net/spdy/

In einem weiteren Test mit den gleichen Webseiten vergleicht Google die Ladezeit von SPDY gegenüber HTTP bei unterschiedlichen Packet loss (Verlust von Datenpaketen durch eine schlechte Verbindung) Werten.

	Ø ms Ladezeit		**Speedup**
Packet loss rate	**HTTP**	**SPDY basic (TCP)**	
0,0%	1152	1016	11,81%
0,5%	1638	1105	32,54%
1,0%	2060	1200	41,75%
1,5%	2372	1394	41,23%
2,0%	2904	1537	47,70%
2.5%	3028	1707	43,63%

Tabelle 2: Google Messungen: Durchschnittliche Ladezeit nach Packet loss

Der Tabelle ist zu entnehmen, dass SPDY mit schlechten Verbindungen deutlich besser zurecht kommt als reguläres HTTP.

Der letzte Test vergleicht die Ladezeit von SPDY gegenüber HTTP bei steigenden Paketumlaufzeiten (RTT).

	Ø ms Ladezeit		**Speedup**
RTT in ms	**HTTP**	**SPDY basic (TCP)**	
20	1240	1087	12,34%
40	1571	1279	18,59%
60	1909	1526	20,06%
80	2268	1727	23,85%
120	2927	2240	23,47%
160	3650	2772	24,05%
200	4498	3293	26,79%

Tabelle 3: Google Messungen: Durchschnittliche Ladezeit nach RTT

Auch hier ist beim Einsatz von SPDY ein Geschwindigkeitszuwachs mit zunehmender Distanz zu beobachten.

4.2 Eigene Messungen

Um die Messungen von Google zu verifizieren wurden eigene Messungen durchgeführt. Da das aufstellen einer Laborumgebung mit einem Dummynet[14] den Rahmen dieser Ausarbeitung sprengen würde, wurde auf Online-Tools zurückgegriffen und mit Hilfe des Website Performance Test-Tools „WEBPAGETEST“[15] durchgeführt. Weitere, detaillierte Tests sollten daher in einer zusätzlichen Arbeit durchgeführt werden.

Die verwendete Software ist, im Gegensatz zu den Tests von Google, Standardsoftware. Der Server ist ein gängiger Webserver mit installiertem Apache 2.2 auf folgender Hardware Konfiguration:

- Intel Cori i7 950 CPU
- 24 GB Arbeitsspeicher
- 64 Bit Debian Squeeze Betriebssystem
- 2 Festplatten im Raid-1 Verbund
- 100 Mbit Uplink

Die URL der getesteten Webseite lautet https://www.meinabschluss.de. Die Webseite besteht aus mehreren JavaScript-, CSS- und Bilddateien sowie einem Video.

Die Konfiguration[16] der einzelnen Testreihen lautete wie folgt:

Testreihe	Standort des Clients	Verbindungsart und Verbindungsgeschwindigkeit	Paketumlaufzeit (RTT)
1	Buenes Aires, Argentina	Modem 0,049 Mbps Download, 0,030 Kbps Upload	120ms
2	New York, NY USA	DSL 1,5 Mbps Download, 384 Kbps Upload	50ms
3	Frankfurt, DE	Kabel 5 Mbps Download, 1 Mbps Upload	28ms
4	Montreal, Canada	Glasfaser 20 Mbps Download, 5 Mbps Upload	4ms
5	Montreal, Canada	Kabel 5 Mbps Download, 1 Mbps Upload	28ms
6	Frankfurt, DE	Glasfaser 20 Mbps Download, 5 Mbps Upload	4ms

[14] Ein Testnetzwerk. Infos siehe: http://info.iet.unipi.it/~luigi/dummynet/

[15] Siehe http://www.webpagetest.org/

[16] Detaillierte Testkonfiguration siehe Appendix A

In jeder Testreihe wurde die Ladezeit der Webseite mit regulärem HTTPS gegenüber SPDY verglichen. Jede Testreihe bestand aus jeweils drei Durchläufen. Jeder Durchlauf wurde zwei Mal durchgeführt um die Ladezeit der Webseite im ungecachten und im gecachten Zustand zu vergleichen. Der Durchschnitt der drei Durchläufe wurde als Messwert in die folgenden Diagramme eingetragen.

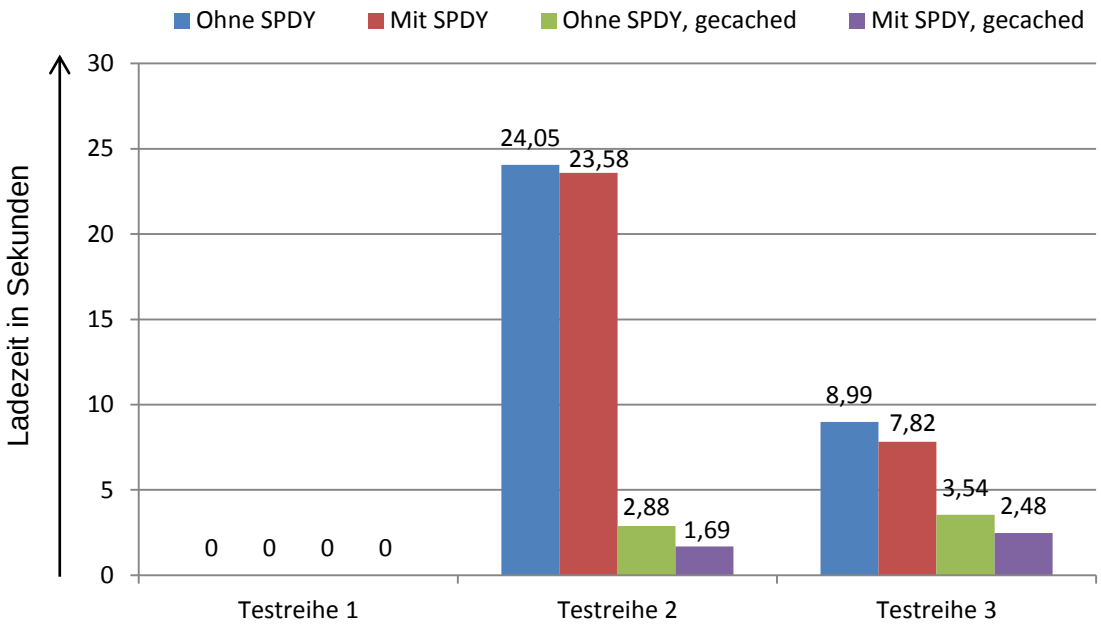

Abbildung 7: Ergebnisse der Testreihen 1 bis 3

Testreihe 1 wurde in beiden Fällen (mit und ohne SPDY) mit einem Timeout abgebrochen.

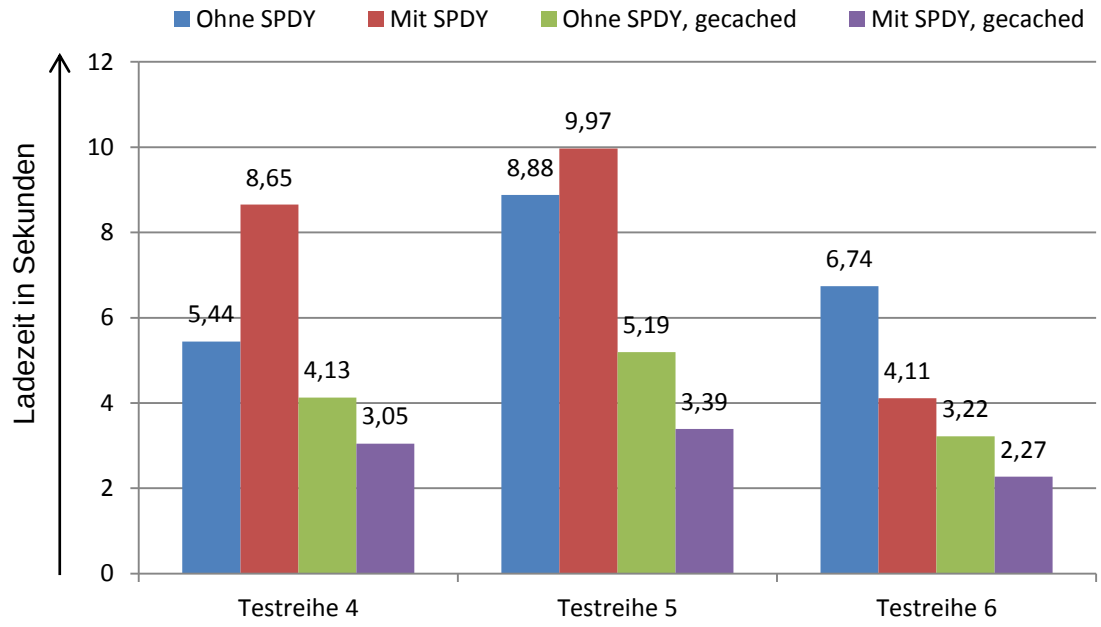

Abbildung 8: Ergebnisse der Testreihen 4 bis 6

In Testreihe 4 und 5 gibt es Ausreiser die zeigen, dass eine Webseite, die mit SPDY geladen wird auch langsamer sein kann als mit regulärem HTTPS. Um sicherzustellen, dass dies kein temporäres Problem (z.B. Server-Load) ist wurde Testreihe 4 und 5 zu einem späterem Zeitpunkt wiederholt mit einem sehr ähnlichem Ergebnis. Es kann dennoch sein, dass der Client, der von Webpagetest zur Verfügung gestellt wird, falsch konfiguriert ist. Auch dies sollte in einer zusätzlichen Ausarbeitung mit einem Testnetzwerk oder weiteren Testseiten behandelt werden.

		Ø ms Ladezeit		**Speedup**
Testreihe	**Cache Status**	**HTTP**	**SPDY**	
1	Ungecached	n/A	n/A	n/A
	Gecached	n/A	n/A	n/A
2	Ungecached	24049	23578	1,96%
	Gecached	2879	1688	41,37%
3	Ungecached	8995	7822	13,04%
	Gecached	3537	2482	29,83%
4	Ungecached	5438	8648	- 59,03%
	Gecached	4130	3048	26,20%
5	Ungecached	8875	9970	- 12,34%
	Gecached	5191	3390	34,69%
6	Ungecached	6739	4108	39,04%
	Gecached	3219	2274	29,36%

Tabelle 4: Eigene Messung: Reduzierte Ladezeit mit SPDY

Die Tabelle zeigt eine durchschnittliche Reduktion der Ladezeit (inkl. der evtl. Fehlmessungen) mit SPDY von 14,61%. Ohne die negativen Werte beträgt der Geschwindigkeitszuwachs durchschnittlich 27,19%. Dies entspricht weniger als den von Google angekündigten Werten [SWIPA], kann aber auch an der vergleichsweise „unsauberen“ Testweise des Online-Tools liegen. Zur genaueren Ermittlung des Geschwindigkeitsvorteils durch SPDY sollten weitere Tests „in der Wildnis“ durchgeführt werden, so auch der Vorschlag und die Bitte von Google:

> „*Our initial results are promising, but we don't know how well they represent the real world. [...] To help with these challenges, we encourage you to get involved.* “

5 Fazit

Trotz der Abweichungen der eigenen Messergebnisse zu Googles Laborergebnissen bringt SPDY doch einen merklichen Geschwindigkeitsvorteil bei minimalem Installationsaufwand. Auch die Änderungen, die bei bereits optimierten Webseiten durchzuführen sind halten sich in Grenzen. Nachteilig sind die aktuell noch geringe Verbreitung bei den Desktopbrowsern sowie die mangelnde Implementierung in gängige Webserver. Weiterhin wird SPDY für viele Webseiten Betreiber durch die technisch bedingte Notwendigkeit einer SSL-Verschlüsselung aus Kostengründen eher unattraktiv sein.

Google schreibt in seinem Whitepaper [SWIPA], dass die benötigten Hardwareressourcen beim Einsatz von SPDY kaum ansteigen. Da SPDY aber nun Ressourcen parallel ausliefern kann müssen diese auch von der darunterliegenden Hardware parallel und performant abgerufen und verarbeitet werden können. Beim weit verbreiteten *Shared Hosting*, bei dem bis zu 500 Webseiten parallel auf einem physikalischem Server liegen, könnte sich der Einsatz von SPDY wieder negativ auf die Performance des Gesamtsystems auswirken. Dies sollte in einer weiteren Ausarbeitung untersucht werden.

Meiner Meinung nach steht der Verbreitung von SPDY (bei ausreichender Implementierung in Browser und Webserver) nichts im Wege, bis auf die noch viel zu teuren SSL-Zertifikate. Nicht jeder Webmaster kann oder möchte 30 bis 250€ (Wildcard-Zertifikat) im Jahr dafür ausgeben, dass seine nicht kommerzielle Webseite verschlüsselt übertragen wird und von SPDY profitieren kann. Möchte Google die Verbreitung von SPDY weiter voran treiben sollten sie auch dieses Problem adressieren und SSL-Zertifikate zum Produktionspreis von ein paar Euro anbieten.

6 Literaturverzeichnis

[TPOWP] „The Psychology of Web Performance“ vom 30.05.2008 http://www.websiteoptimization.com/speed/tweak/psychology-web-performance/

[CSPOZ] „Client-side Performance Optimizations“, Januar 2010 von Jakob Schröter http://www.slideshare.net/jakob.schroeter/clientside-performance-optimizations

[WEOPC] „Website Optimization: Speed, Search Engine & Conversion Rate Secrets“, Juli 2008 von Andrew B. King. ISBN 0596515081

[SWIPA] SPDY Whitepaper: http://www.chromium.org/spdy/spdy-whitepaper (Abgerufen am 21.04.2012)

[CHROM] Projekt Webseite von Chromium: http://www.chromium.org/spdy (Abgerufen am 21.04.2012)

[SPDYP] SPDY Protocol Specifiations http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00 (Abgerufen am 07.05.2012)

[MODSP] Quellcode zu mod_spdy und Installationsanweisungen https://developers.google.com/speed/spdy/mod_spdy/ (Abgerufen am 23.04.2012)

[BLOGS] Google Developer Blogpost zu SPDY http://googledevelopers.blogspot.de/ 2012/04/add-spdy-support-to-your-apache-server.html (Abgerufen am 23.04.2012)

[NETCR] Statistiken zum Einsatz von Webservern unter 676.919.707 Webseite: http://news.netcraft.com/archives/category/web-server-survey/ (Abgerufen am 24.04.2012)

[HTTPP] Implementationsstatus von HTTP-Pipelining in verschiedenen Browsern http://en.wikipedia.org/wiki/HTTP_pipelining#Implementation_status (Abgerufen am 07.05.2012)

[HEISE] Artikel „Microsoft bringt eigenen Vorschlag für HTTP 2“ http://www.heise.de/netze/meldung/Microsoft-bringt-eigenen-Vorschlag-fuer-HTTP-2-1479694.html (Abgerufen am 24.04.2012)

[HTTP2] Microsoft Blogpost zu „Speed and Mobility: An Approach for HTTP 2.0 to Make Mobile Apps and the Web Faster“: http://blogs.msdn.com/b/interoperability/ archive/2012/03/26/speed-and-mobility-an-approach-for-http-2-0-to-make-mobile-apps-and-the-web-faster.aspx (Abgerufen am 24.04.2012)

[SPDBP] SPDY Best Practices auf der Chromium Projektwebseite http://dev.chromium.org/spdy/spdy-best-practices (Abgerufen am 25.04.2012)

7 Appendix A

Die Testkonfiguration auf http://www.webpagetest.org/ lautete wie folgt:

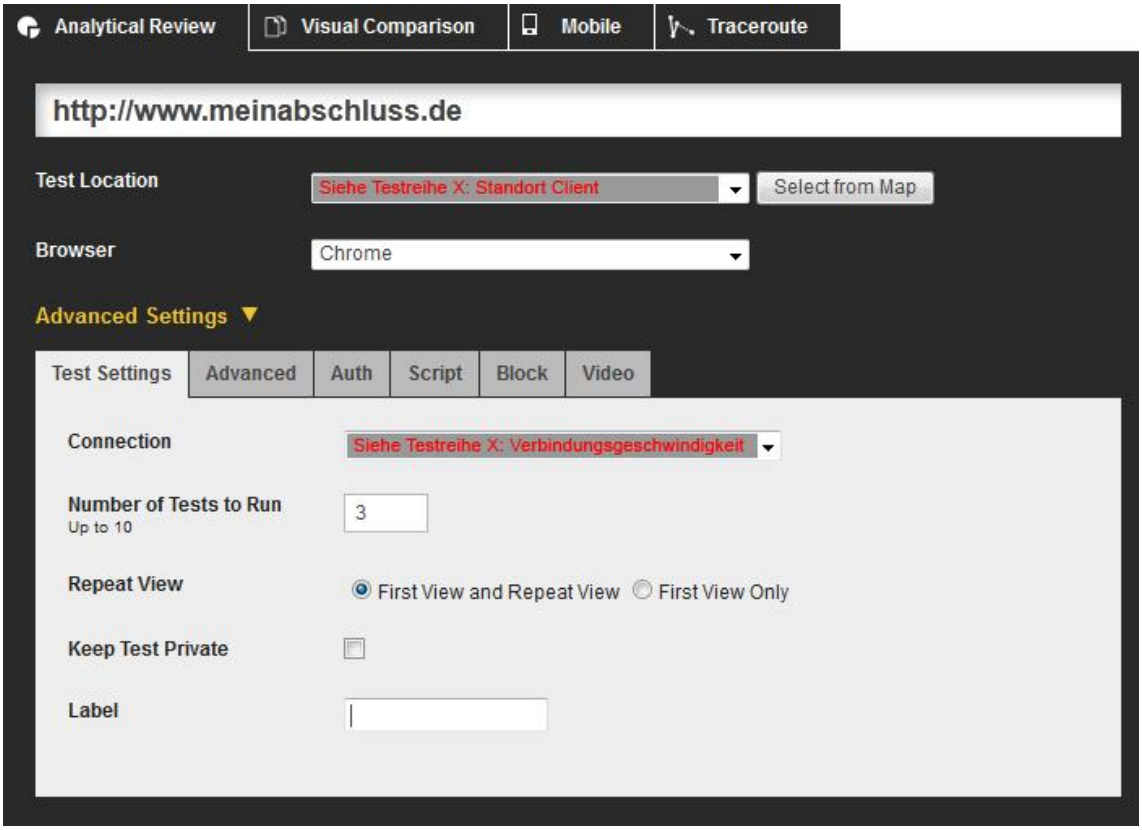

Abbildung 9: Testkonfiguration auf webpagetest.org: Basiseinstellungen

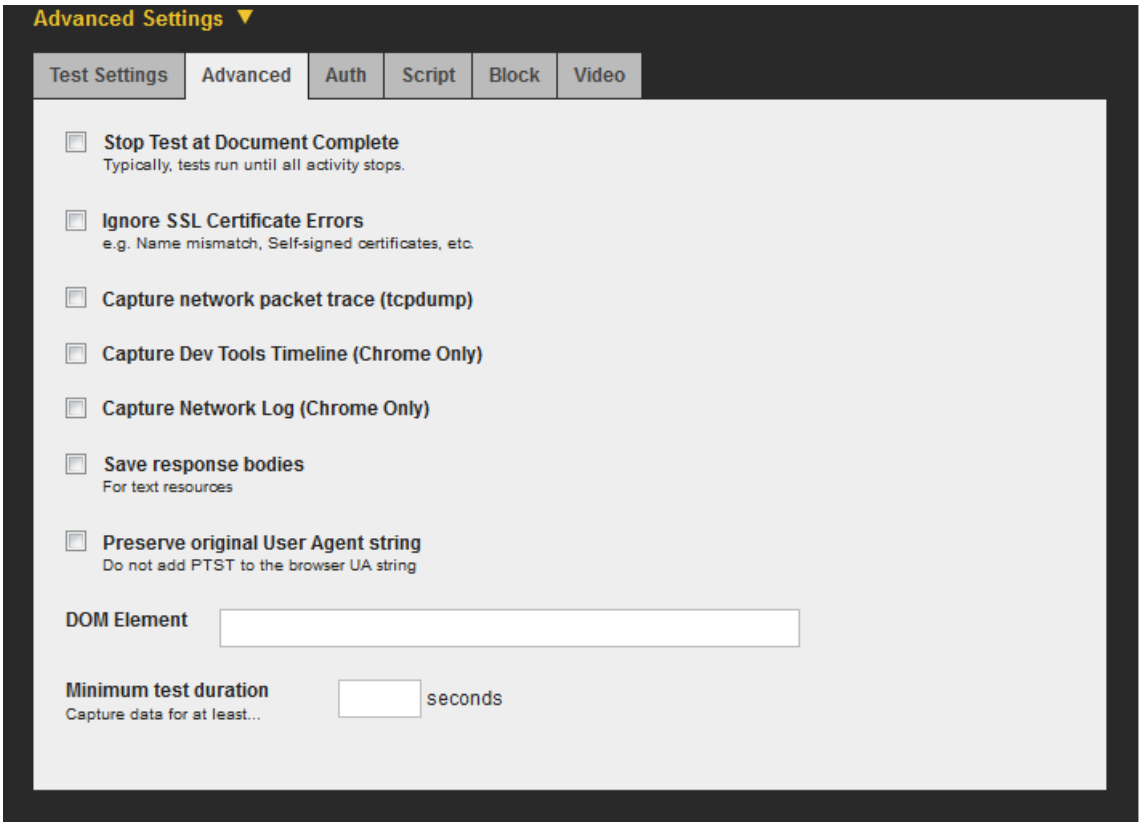

Abbildung 10: Testkonfiguration auf webpagetest.org: Erweiterte Einstellungen 1/5

- Erweiterte Einstellungen 2/5: Tab „Auth“: Kein Auth
- Erweiterte Einstellungen 3/5: Tab „Script“: Kein Skript
- Erweiterte Einstellungen 4/5: Tab „Block“: Kein Block Request

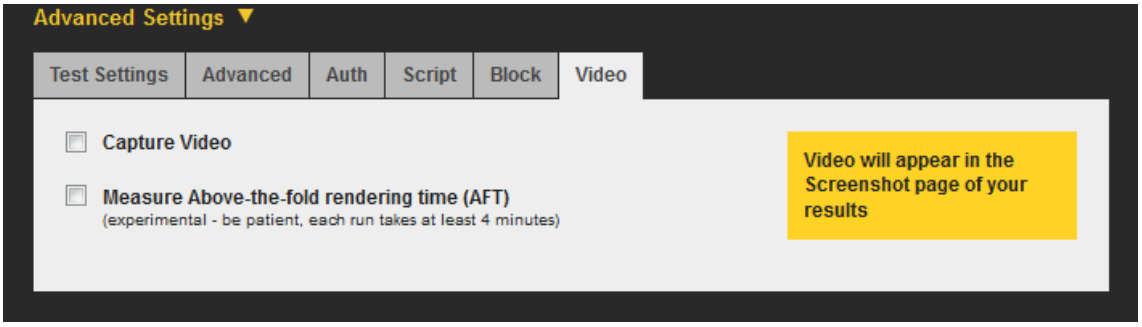

Abbildung 11: Testkonfiguration auf webpagetest.org: Erweiterte Einstellungen 5/5