HOCHSCHULE DER MEDIEN

Ruben Deyhle
CSM 1. Semester
HdM Stuttgart
Jakob Schröter
Web Performance Optimizations
Entwicklung von Rich-Media-Systemen
13. August 2012

HTTP 2.0

SPDY vs. HTTP Speed+Mobility

Abstract

HTTP 1.1 ist dreizehn Jahre alt. Als Nachfolger sind bislang zwei Vorschläge im Gespräch: SPDY von Google sowie Speed+Mobility von Microsoft. Diese Arbeit erläutert und vergleicht die beiden Vorschläge. Sie entstand als Studienleistung in der Veranstaltung „Web Performance Optimizations“ an der Hochschule der Medien Stuttgart im Sommersemester 2012.

I. Einführung

Zunächst wird in dieser Arbeit die gegenwärtige Technologie in Form von HTTP 1.1 betrachtet (Kapitel II), und erläutert, warum ein neuer Standard überhaupt notwendig ist. Anschließend werden die beiden nennenswerten Vorschläge vorgestellt (SPDY in Kapitel III, Speed+Mobility in Kapitel IV). Schließlich wird auf Gemeinsamkeiten und Unterschiede eingegangen (Kapitel V), bis dann ein kurzer Ausblick eine Einschätzung gibt, wohin der Weg geht (Kapitel VI).

II. Notwendigkeit eines neuen Standards

Der HTTP-Standard, wie er heute für das World Wide Web verwendet wird, ist seit 1999 auf Version 1.1. Während er nach wie vor gut funktioniert, sollte man bedenken, dass sich das Internet in den letzten 13 Jahren gewandelt hat.

In den 90er-Jahren, als HTTP entworfen wurde, sah das World Wide Web noch anders aus als heute. Das typische Zugangsgerät für den Endnutzer war ein Desktop-PC, Websites bestanden hauptsächlich aus einfachem HTML und waren wenig komplex. Meist wurden alle Ressourcen einer Website von einem einzigen Server bedient; um eine einzelne Website anzuzeigen, genügten wenige HTTP-Requests (oft sogar nur ein einzelner).

Heute dagegen hat sich einiges geändert: es gibt eine Vielzahl an sehr unterschiedlichen, auch mobilen Geräten, die Websites anzeigen, die teilweise kaum mehr von einer nativen Applikation unterscheidbar sind. Websites bestehen häufig aus einer Vielzahl an unterschiedlichen Medien, wie Bildern und Videos, und zahlreichen dynamischen Elementen. Die Inhalte kommen meist von mehreren Servern oder Content Distribution Networks, die die Last verteilen sollen. Die Zahl der notwendigen HTTP-Requests, die für den Aufruf einer Website notwendig ist, ist gestiegen: teilweise werden für einen einzigen Aufruf mehrere Dutzend HTTP-Requests abgesetzt.

HTTP funktioniert bislang stets synchron: der Browser wartet nach jeder Anfrage die Antwort des Servers ab, bevor er die nächste Anfrage schickt. Bei heute üblichen, komplexen Websites und Webanwendungen kann dies in sehr langen Ladezeiten und schlechter Performance resultieren. Daher haben sich in der Praxis mittlerweile diverse Optimierungen durchgesetzt, um auch mit HTTP 1.1 eine akzeptable Performance zu erreichen: dazu wer-

den beispielsweise mehrere Verbindungen parallel aufgebaut, versucht, Requests zu sparen (z.B. durch CSS-Sprites) oder die Anfragen auf mehrere Server verteilt.

Gerade die Vorgehensweise, mehrere TCP-Verbindungen zu verwenden, hat aber auch einige Nachteile: für jede Verbindung ist der für TCP obligatorische 3-Wege-Handshake vonnöten, der für einen eigentlich unnötigen Overhead sorgt. Zudem sorgt der TCP Slow Start dafür, dass bei kurzen Übertragungen von nur kleinen Datenmengen gar keine hohen Geschwindigkeiten erreicht werden können.

Als weiteren Kritikpunkt kann bei HTTP außerdem angebracht werden, dass SSL-Verschlüsselung nur optional ist. Dadurch ist ein großer Teil der Kommunikation im Web nach wie vor unverschlüsselt, was leicht ein Sicherheitsrisiko darstellen kann.

In Anbetracht dieser Probleme mit dem aktuellen HTTP-1.1-Standard regte die Internet Engineering Task Force (IETF) im Januar 2012 die Entwicklung von HTTP 2.0 an. [1] Drei wichtige Designziele wurden dabei formuliert: [2]

Zunächst soll HTTP 2.0 – naheliegend – effizienter und schneller als HTTP 1.1 sein. Weiter soll idealerweise stets nur eine einzelne TCP-Verbindung ausreichend sein. Und zu guter Letzt sollte HTTP 2.0 – wie auch schon HTTP 1.1 – vollständig abwärtskompatibel sein. [1]

Bislang gibt es zwei nennenswerte Vorschläge für HTTP 2.0: Googles HTTP-Aufsatz „SPDY“ und Microsofts Gegenentwurf „HTTP Speed+Mobility“.

III. SPDY

Eines der Hauptfeatures von SPDY ist Multiplexing. Dadurch benötigt SPDY stets nur eine einzelne TCP-Verbindung, wie gefordert. Der bei mehreren Verbindungen anfallende TCP-Overhead entfällt. SPDY sendet über seine TCP-Verbindung direkt mehrere Requests ab, wodurch beliebig viele Daten parallel übertragen werden können. Eine Priorisierung, dass beispielsweise der Hauptinhalt der Website bevorzugt wird, ist möglich. [2, 4]

Doch über Multiplexing hinaus wird durch SPDY der Funktionsumfang von HTTP auch um neue Features erweitert: Server Push und Server Hint. [3]

Server Push ermöglicht es einem Webserver, der mit SPDY läuft, Daten auch ohne explizite Anfrage des Clients an diesen auszuliefern. Dies ist für Echtzeitanwendungen hilfreich, die bislang per JavaScript in regelmäßigen Abständen nach Änderungen beim Server nachfragen müssen. Ein SPDY-Webserver kann z.B. neue Nachrichten bei einem Web-Chat-Client direkt ausliefern, ohne dass der Client explizit nach neuen Nachrichten nachfragt.

Außerdem können so beispielsweise auf die Anfrage nach einer HTML-Datei für die Anzeige notwendige oder erwünschte zusätzliche Dateien (wie CSS-Stylesheets oder JavaScript-Code) gleich mitgeschickt werden.

Dies ist natürlich eine unnötige Datenübertragung, wenn der Client die entsprechenden Daten bereits im Cache hält und eigentlich nicht erneut erhalten muss. In dieser Situation hilft Server Hint:

Prinzipiell funktioniert Server Hint ähnlich wie Server Push. Allerdings informiert der Server den Client zunächst nur über die zusätzlichen oder neuen Daten, die er gerne mitschicken

würde. Dann kann der Client prüfen, ob er die Daten bereits im Cache hält. Ist dies nicht der Fall, kann er die Daten über eine normale Anfrage anfordern. [3]

SPDY verwendet die im Web verbreitete gzip-Kompression für alle übertragenen Inhalte. Eine manuelle Aktivierung von Komprimierung ist also nicht mehr notwendig, die zu übertragende Datenmenge wird automatisch reduziert. [2]

Außerdem ist SSL-Verschlüsselung bei SPDY standardmäßig aktiv, wodurch eine sichere Verbindung gewährleistet ist. [2]

Aktuelle Implementierungen von SPDY führen die Umschaltung von herkömmlichem HTTP mithilfe der Next Protocol Negotiation von TLS durch, was (im Gegensatz zu Lösungen wie dem HTTP-1.1-Upgrade-Header) ohne zusätzlichen Round-Trip direkt erfolgen kann.

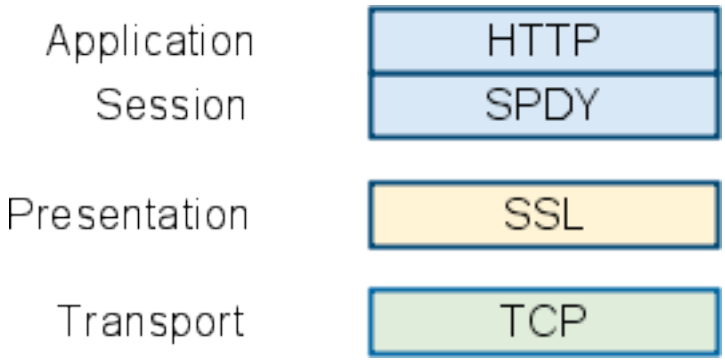

Abbildung 1: SPDY-Layer. Quelle: SPDY-Whitepaper [4]

Die Layer-Architektur von SPDY ist in Abbildung 1 dargestellt. Der Kernpunkt ist der Session-Layer, der auf SSL aufsetzt und parallele Datenströme über eine einzelne TCP-Verbindung ermöglicht. [4]

SPDY wurde von Google bereits Ende 2009 vorgestellt [5], also lange vor dem IETF-Aufruf zu HTTP 2.0. Als Aufsatz für HTTP 1.1 designt ist SPDY durchaus bereits lauffähig und wird auch eingesetzt.

Googles eigener Browser Chrome (bzw. das Chromium-Projekt) unterstützt SPDY seit Version 11 (April 2011); in Mozilla Firefox ist SPDY seit Version 13 (Juni 2012) aktiviert, ebenfalls seit Version 11 (März 2012) ist die Unterstützung bereits vorhanden, das gleiche gilt für SeaMonkey, was auf Firefox basiert. [6]

Der Amazon-Silk-Browser auf Amazons Kindle-Fire-Tablet, im November 2011 vorgestellt, optimiert Websites durch Auslagerung auf Amazons EC2-Server (Elastic Compute Cloud). Er verwendet für die Kommunikation mit dem Client SPDY. [7]

Der norwegische Browser Opera hat bereits experimentelle Unterstützung von SPDY [8] und wird vermutlich in einer kommenden Version SPDY genauso unterstützen wie Firefox und Chrome das derzeit bereits tun.

Doch natürlich wird (mit Ausnahme vom Silk-Browser, der nicht direkt mit den entsprechenden Webservern kommuniziert) auch auf Serverseite SPDY-Unterstützung benötigt.

Dafür stehen bereits einige Implementierungslösungen bereit, beispielsweise ein Modul für den verbreiteten Apache-Webserver. [9] Insbesondere einige Google-Services bieten SPDY bereits an, wenn der Nutzer mit einem SPDY-fähigen Browser unterwegs ist. Dies sind fast alle HTTPS-fähigen Dienste, unter anderem die Suche, GMail, Google Maps, YouTube; außerdem Google-Anzeigen.[1]

[1] Dies kann leicht mit der Google-Chrome-Erweiterung „SPDY indicator“ nachvollzogen werden. Die genannten Dienste verwendeten demzufolge, jeweils über https aufgerufen, am 6.8.2012 alle SPDY in Google Chrome 22.

Auch einige andere Webservices verwenden inzwischen SPDY: prominentestes Beispiel war im März 2012 Twitter; [10] außerdem kündigte Facebook bereits an, SPDY zu implementieren. [11]

SPDY liegt mittlerweile in Version 3.0 vor. Diese unterstützt die HTTP-2.0-Anforderungen der IETF und wurde als Vorschlag für HTTP 2.0 eingereicht. [12]

IV. HTTP Speed+Mobility

Im März 2012 kündigte Microsoft an, mit „HTTP Speed+Mobility“ einen eigenen Vorschlag für HTTP 2.0 abzuliefern. [13] Bislang sind allerdings nur wenige Informationen darüber bekannt, zumal eine Referenzimplementierung derzeit noch fehlt (Stand Juli 2012).

Als Kernziel wird von Microsoft angeführt, auch mobile Endgeräte und Apps zu unterstützen, wo Dinge wie niedriger Energieverbrauch wichtig sind – ein Punkt, der bei SPDY kritisiert wird. Um die Abwärtskompatibilität zu gewährleisten, setzt HTTP Speed+Mobility auf eine strenge Layer-Architektur, bei der alle Zusatz-Features optional sind. [13]

Basis für HTTP Speed+Mobility sollen dabei WebSockets bilden. WebSocket ist ein standardisiertes, eigenständiges Protokoll für Echtzeit-Webanwendungen. Es bietet eine bidirektionale, feste Verbindung zwischen Client und Server, über die beide Seiten jederzeit Daten übertragen können. Auch Server-Push ist folglich möglich. Da es nicht auf HTTP aufsetzt, tritt kein HTTP-Overhead auf.

Die Wahl von WebSockets als Basis ergibt in Hinblick auf die angestrebte Abwärtskompatibilität Sinn. HTTP Speed+Mobility versucht so weit wie möglich, existierende Standards zu verwenden, wodurch versucht wird, z.B. Probleme mit Proxy-Servern zu verhindern. Die HTTP-Semantik wird komplett beibehalten, Speed+Mobility verwendet – wie auch WebSockets – den HTTP 1.1 Upgrade-Header. Dieses Feld im HTTP-Header ermöglicht es einem Client, eine Verbindung auf Wunsch auf die gewünschte Technologie zu „upgraden“. Ist der Server nicht dazu in der Lage, wird weiterhin normales HTTP verwendet. Dieses Vorgehen hat den Vorteil, dass die Umstellung auf den neuen HTTP-Standard einen weichen Übergang darstellt. Die Abwärtskompatibilität wäre stets gewährleistet. Allerdings ist ein zusätzlicher Round-Trip für die Aushandlung notwendig. [13, 14]

HTTP Speed+Mobility legt Wert auf eine strenge Layer-Architektur. So soll das Protokoll keine Aufgaben von ISO/OSI-Layer 4, der Transportschicht, übernehmen: die Überlastkontrolle (Congestion Control) soll komplett TCP überlassen bleiben. [13]

HTTP Speed+Mobility legt, schon beim Namen, nicht nur Wert auf Geschwindigkeit, sondern auch auf Mobilität. SPDY-Features wie Push, Verschlüsselung und Kompression werden nicht für alle Clients als passend angesehen: um weniger Last auf mobilen Clients zu erzeugen, und Akku und Bandbreite zu sparen, sollen diese Features nur als optionale Komponenten realisiert werden. Der Client soll angeben können, welche Funktionen er verwenden möchte. [13]

Zur konkreten Funktionsweise von HTTP Speed+Mobility ist, im Gegensatz zu SPDY, mangels Referenzimplementierung noch verhältnismäßig wenig bekannt.

Prinzipiell lässt sich HTTP Speed+Mobility in vier Layer gliedern. Der grundlegende Verbindungslayer wird als WebSocket realisiert (wie erläutert mit der Upgrade-Semantik von HTTP 1.1). Darauf setzt ein Session-Layer auf, der als WebSocket-Extension realisiert ist. Schließlich folgt ein Multiplexing-Layer, der wie SPDYs Multiplexing die parallele Datenübertragung ermöglicht. Allerdings ist auch dieser Layer eine WebSocket-Extension, um die Abwärtskompatibilität zu gewährleisten. Und schließlich definiert HTTP Speed+Mobility einen HTTP-Layer, der SPDY sehr ähnlich sein soll („borrowed from SPDY“). [13, 14]

HTTP Speed+Mobility ist lange nicht so ausgereift wie SPDY, und derzeit nicht einsatzfähig. Erst im März 2012 wurde die Technologie angekündigt, zusammen mit einem ersten

Draft. Am 11.05.2012 wurde ein erster, unvollständiger Prototyp vorgestellt. Derzeit liegt Draft 2 vom 15.06.2012 vor, bei dem ebenfalls noch einige Details unklar bleiben. [14]

V. Vergleich

Zunächst soll auf die Gemeinsamkeiten von HTTP Speed+Mobility und SPDY eingegangen werden. Beide Vorschläge erfüllen die von der IETF vorgegebenen Anforderungen für HTTP 2.0: beide bieten Multiplexing, wodurch nur eine einzige TCP-Verbindung benötigt wird, und beide sind abwärtskompatibel. Zwar schreibt Microsoft, sein Vorschlag sei „besser“ abwärtskompatibel als SPDY, allerdings traten beim Einsatz von SPDY bislang auch keine Probleme mit nicht unterstützenden Clients oder Servern auf.

Die Unterschiede zwischen SPDY und HTTP Speed+Mobility sind hauptsächlich in der Semantik zu finden. Während SPDY als absolutes Ziel mehr Geschwindigkeit festgelegt hat, legt HTTP Speed+Mobility auch auf die Mobilität wert, und möchte daher die Komponenten optional halten. Dagegen verwendet SPDY beispielsweise grundsätzlich Kompression für alle Daten. Weiteres Alleinstellungsmerkmal von SPDY ist die eingebaute Möglichkeit für Server-Push, was HTTP Speed+Mobility ablehnt, da es Microsoft zufolge nicht mit der HTTP-Semantik vereinbar ist (offenbar wird aber möglicherweise ein optionaler Server-Push unterstützt). [13, 15]

Alleinstellungsmerkmal von HTTP Speed+Mobility dagegen ist die grundsätzliche Verwendung von WebSockets als Basis, anstatt alle Funktionen direkt in das Protokoll einzubauen, wie es bei SPDY der Fall ist.

Nicht zuletzt ein möglicherweise ausschlaggebender Unterschied zwischen den beiden Vorschlägen ist aber, dass SPDY erprobt und einsatzbereit ist, während sich HTTP Speed+Mobility noch in einem sehr frühen Entwicklungsstadium befindet.

Einen kurzen Überblick über die Gemeinsamkeiten und Unterschiede kann die folgende Tabelle bieten:

	SPDY	HTTP Speed+Mobility
Vorgeschlagen von	Google	Microsoft
Kernziel	Geschwindigkeit	Geschwindigkeit, Opt-in-Verhalten für mobile Geräte
Unterstützung (Client)	Chrome, Firefox, Opera angekündigt	bislang keine
Unterstützung (Server)	Diverse Implementierungen einsatzbereit	bislang keine
Kompression	obligatorisch	kann deaktiviert werden
Multiplexing	ja	ja
Verschlüsselung	nicht erforderlich, bislang aber stets aktiv	optional
Upgrade ohne Latenz	nicht erforderlich, bislang aber stets TLS NPN	Nein, nur HTTP-1.1-Upgrade
Server Push	ja	von Client anforderbar

[15]

VI. Ausblick

Die Internet Engineering Task Force (IETF) möchte im Juli 2013 HTTP 2.0 vorstellen.

Es wird vermutet, dass das Protokoll SPDY als Basis haben wird; Googles Vorschlag ist in der Praxis erprobt und unterstützt. Allerdings dürften auch einige von Microsoft eingebrachte Punkte Beachtung finden. Beispielsweise fehlt bislang eine SPDY-Implementierung, die keine TLS-Verschlüsselung voraussetzt – dies dürfte daran liegen, dass die ersten beiden SPDY-Versionen dies verpflichtend vorschrieben und erst die SPDY-3.0-Spezifikation die Verschlüsselung optional machte.

Außerdem ist mittlerweile ein dritter Vorschlag zu HTTP 2.0 aufgetaucht, der allerdings absichtlich keine komplette Spezifikation darstellt, sondern einige zusätzliche Punkte einbringen soll: „Network-Friendly HTTP Upgrade“. [16] Der Fokus liegt auf der Optimierung innerhalb der Netzwerkinfrastruktur durch Reduzierung des Header-Overheads. So sollen Header, die innerhalb einer Verbindung gleich bleiben, gruppiert werden können; außerdem soll binäre Encodierung verwendet werden. [15]

Es wird für gut möglich gehalten, dass kontroverse Punkte bei HTTP 2.0 zunächst ausgelassen werden und erst inkrementelle Verbesserungen standardisiert werden. [17]

Literaturverzeichnis

(1) Mit SPDY auf dem Weg zu HTTP/2.0? Jens Ihrenfeld, Golem.de. 24.1.2012. http://www.golem.de/1201/89265.html

(2) SPDY Protocol - Draft 3. The Chromium Projects. http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3 (zuletzt abgerufen am 7.8.2012)

(3) Server Push and Server Hints. The Chromium Projects. http://www.chromium.org/spdy/link-headers-and-server-hint (zuletzt abgerufen am 7.8.2012)

(4) SPDY Whitepaper. An experimental protocol for a faster web. The Chromium Projects. http://dev.chromium.org/spdy/spdy-whitepaper (zuletzt abgerufen am 7.8.2012)

(5) A 2x Faster Web. The Chromium Blog. 11.11.2009. http://blog.chromium.org/2009/11/2x-faster-web.html

(6) Mozilla Bug 528288 Implement SPDY protocol. https://bugzilla.mozilla.org/show_bug.cgi?id=528288 (zuletzt abgerufen am 7.8.2012)

(7) The secret to Amazon Silk browser´s speediness? WebKit and SPDY. 9to5Google. 28.9.2011. http://9to5google.com/2011/09/29/the-secret-to-amazon-silk-browsers-speediness-webkit-and-spdy/

(8) A labs release to get your SPDY sense tingling. Chris Mills, Dev.Opera. 6.7.2012. http://dev.opera.com/articles/view/opera-spdy-build/

(9) SPDY Home Page. http://www.chromium.org/spdy/ (zuletzt abgerufen am 7.8.2012)

(10) Tweet von Raffi Krikorian, Twitter. 7.3.2012.
https://twitter.com/raffi/status/177616491204714497/photo/1

(11) HTTP2 Expression of Interest. IETF Mailingliste. Doug Beaver, Facebook. 15.7.2012.
http://lists.w3.org/Archives/Public/ietf-http-wg/2012JulSep/0251.html

(12) Google reicht SPDY bei der IETF ein. Jens Ihrenfeld, Golem.de. 24.2.2012.
http://www.golem.de/news/http-2-0-google-reicht-spdy-bei-der-ietf-ein-1202-90005.html

(13) Speed and Mobility: An Approach for HTTP 2.0 to Make Mobile Apps and the Web Faster. Jean Paoli, Microsoft Interoperability Blog. 15.3.2012.
http://blogs.msdn.com/b/interoperability/archive/2012/03/25/speed-and-mobility-an-approach-for-http-2-0-to-make-mobile-apps-and-the-web-faster.aspx

(14) HTTP Speed+Mobility Draft 2. 15.6.2012.
http://tools.ietf.org/html/draft-montenegro-httpbis-speed-mobility-02

(15) Review of HTTP 2.0. Aaron Croyle, Hurricane Labs. 25.7.2012.
http://www.hurricanelabs.com/review-of-http-2-0/

(16) Proposal for a Network-Friendly HTTP-Upgrade.
https://tools.ietf.org/html/draft-tarreau-httpbis-network-friendly-00 (zuletzt abgerufen am 7.8.2012)

(17) Engineers rebuild HTTP as a faster Web foundation. Stephen Shankland, CNET. 30.3.2012.
http://news.cnet.com/8301-30685_3-57406904-264/engineers-rebuild-http-as-a-faster-web-foundation/