Midiendo la Performance de SPDY

Pablo Maximiliano Lulic

24 de noviembre de 2013

Resumen

Actualmente, la Web se sustenta con el estándar HTTP. Este protocolo tuvo su última versión en el año 1999. Lás páginas en ese entonces eran muy diferentes a las actuales, tanto en contenido como en recursos. Google desarrolló un protocolo en el año 2009 llamado SPDY, cuyo propósito es mejorar la performance en la recuperación de los recursos de la web. A pesar de su gran aceptación y de sentar las bases del próximo HTTP 2.0, hay ciertas cuestiones que todavía quedan por revisar. En este paper, se propone evaluar la performance de SPDY desde dos enfoques diferentes, uno en embientes específicos y otro en la web.

1. Introducción

El protocolo HTTP tuvo su primera versión en mayo de 1996, culminando en 1999 con el estándar actual que es el HTTP 1.1 [3] Es un protocolo sin estado, el servidor no mantiene información acerca de las diferentes peticiones que le llegan, lo que conlleva a que se necesite realizar una conexión nueva por cada recurso que se necesite de un sitio web.

2. La Web en la Actualidad

En comparación con lo que era la web en la época en la que se implementó el protocolo HTTP, hubo un crecimiento amplio en el tamaño y en la cantidad de recursos de un sitio web. Para Noviembre de 2013 [2], el tamaño promedio de un sitio era de 1614kb, en contraste con Noviembre de 2010 que el tamaño promedio era de 702kb, el aumento fue casi del 50 %. El crecimiento se produce con velocidad [15] y hay otras cuestiones relacionadas al tiempo de carga de una página, no solo el ancho de banda [8], sino por ejemplo, el RTT[1].

[1]Tiempo que tarda un paquete de datos en ir desde el emisor al receptor y volver al emisor.

3. SPDY

SPDY[16] es un protocolo de la capa de aplicación [14] que funciona sobre SSL [6], permite la transmisión de Streams[2] sobre una conexión normal de TCP, que es el Protocolo de Control de Transmisión de la capa de Transporte [14]). A continuación se comentarán las características del Protocolo (extraídas de [16]):

1. Streams Multiplexados.
2. Priorización de Peticiones.

   El Cliente puede tantos recursos como quiera del Servidor y asignarle prioridad a cada uno de ellos.
3. Compresión de los Headers HTTP [1].

   Comprime los headers de petición y respuesta HTTP.
4. Push

   Permite al Servidor enviarle recursos al Cliente sin que este se lo pida.
5. Hint

   Permite al Servidor "sugerirle" al Cliente que pida algún recurso específico.

4. Experimento 1

4.1. Preparación

Se utilizaron 3 computadoras de escritorio con la siguiente topología:

$$\underset{DebianGNU/Linux6{,}0}{\textbf{CLIENTE}} \longleftrightarrow \underset{PicoBSD[5]}{\textbf{PROXY}} \longleftrightarrow \underset{Lubuntu13{,}04}{\textbf{SERVIDOR}}$$

1. SERVIDOR

   Se descargaron los sitios[3] y se configuraron los siguientes hosts virtuales en el Servidor:

[2]???

[3]Utilizando la opción "Guardar como..." de Google Chrome, que obteniene todos los recursos externos y los almacena en una carpeta.

a) www.amazon.com

b) www.bing.com

c) login.yahoo.com

d) www.world-flags.com [7]

Cada sitio se brindó utilizando Apache 2.2 [11] tanto en HTTP como en HTTPS, para brindar SPDY, se instaló mod_spdy [4].

2. PROXY

   Utilizando la herramienta Dummynet [12] que viene instalada con la distribución de PicoBSD, se utilizaron diferentes comandos [13] para filtrar el tráfico en la red[4] y simular diferentes entornos.

3. CLIENTE

   Se utilizó Google Chrome 27 a travéz de chrome-har-capturer [9]. Dicha herramienta interactúa con Google Chrome por su API de depuración remota [10].

4.2. Metodología

Con la idea de comparar los métodos HTTP, HTTPS y SPDY en diferentes ambientes, se definieron los siguientes valores:

Ancho de Banda (BW)	Retraso (RTT)
100 Kbps	10 ms
256 Kbps	50 ms
512 Kbps	100 ms
1024 Kbps	200 ms
2048 Kbps	250 ms
5120 Kbps	500 ms
10240 Kbps	

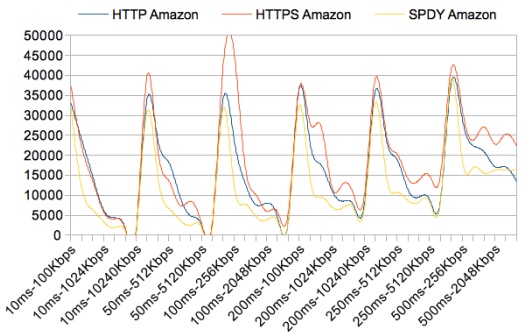
Se combinaron todos los Anchos de Banda con todos los Retrasos para cada uno de los métodos por página[5], se repitió el experimento 5 veces y se promediaron los resultados.

4.3. Resultados

SEGUIR ACA

[4] Ancho de Banda y Retardo.

[5] Por ejemplo: 100Kbps de Ancho de Banda con 100ms de Retraso accediendo al host virtual de Amazon por HTTPS (https://www.amazon.com).



5. Experimento 2

5.1. Preparación

5.2. Metodología

5.3. Resultados

6. Conclusiones y Trabajos Relacionados

Referencias

[1] Header field definitions. `http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html`. part of Hypertext Transfer Protocol – HTTP/1.1 - RFC 2616.

[2] Http archive. `http://httparchive.org`.

[3] Hypertext transfer protocol – http/1.1. `http://www.ietf.org/rfc/rfc2616.txt`.

[4] mod-spdy - apache spdy module. `http://code.google.com/p/mod-spdy/`.

[5] Picobsd. `http://people.freebsd.org/~picobsd/old/picobsd.html`.

[6] The secure sockets layer (ssl) protocol version 3.0. `http://tools.ietf.org/html/rfc6101`.

[7] World flags mod_spdy demo. `https://www.modspdy.com/world-flags/`.

[8] Mike Belshe. More bandwidth doesn't matter (much). Agosto 4 - 2010.

[9] Andrea Cardaci. Capture har files from a remote chrome instance. `https://github.com/cyrus-and/chrome-har-capturer`.

[10] Google Developers. Remote debugging protocol. https://developers.google.com/chrome-developer-tools/docs/debugger-protocol.

[11] The Apache Software Foundation. Apache. http://www.apache.org.

[12] FreeBSD. Dummynet. http://www.freebsd.org/cgi/man.cgi?query=dummynet.

[13] FreeBSD. Ipfw. http://www.freebsd.org/cgi/man.cgi?query=ipfw&sektion=8&apropos=0&manpath=FreeBSD+9.2-RELEASE.

[14] W. Richard Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley Professional, 1994. Section 1.2 Layering.

[15] website optimization. Average web page size triples since 2008. http://www.websiteoptimization.com/speed/tweak/average-web-page/.

[16] website optimization. Spdy: An experimental protocol for a faster web. http://www.chromium.org/spdy/spdy-whitepaper.