Midiendo la Performance de SPDY

Pablo Maximiliano Lulic

16 de diciembre 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 [6] 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 [5], 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 %. Estudios más recientes, indican que el crecimiento de un sitio promedio es del %151 [18] (ver figura 1).

El crecimiento se produce con velocidad [29] y hay otras cuestiones relacionadas al tiempo de carga de una página, no solo el ancho de banda

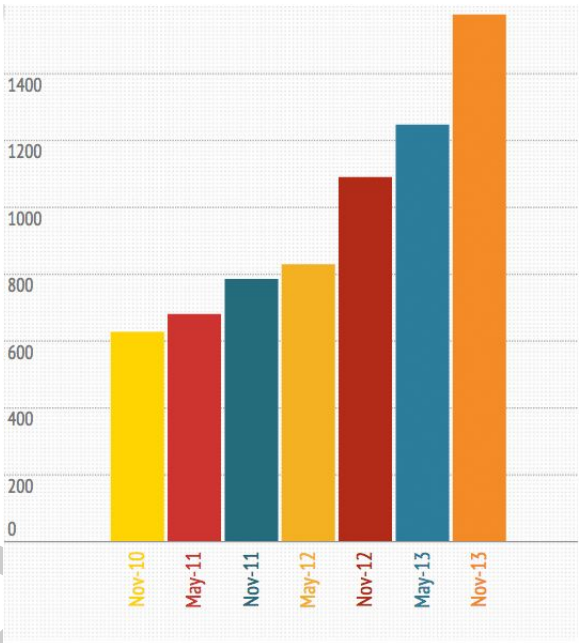

Figura 1: Crecimiento del tamaño de los sitios promedio, extraído de [18]

[14], sino por ejemplo, el RTT[1], como se puede ver en la figura 2

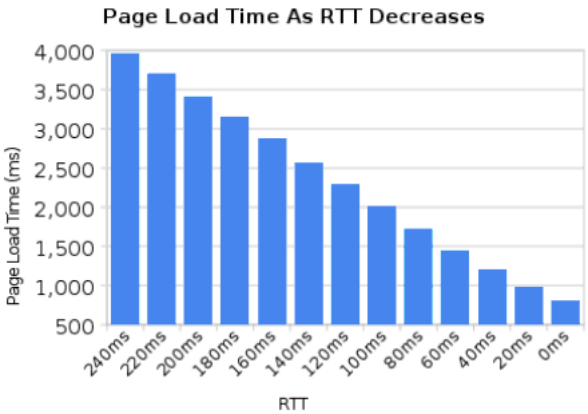

Figura 2: Tiempo de carga de una página mientras se varía el RTT, extraído de [14]

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

3. SPDY

SPDY[30] es un protocolo de la capa de aplicación [26] que funciona sobre SSL [10], 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 [26]). A continuación se comentarán las características del Protocolo (extraídas de [30]):

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 [4].

   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 virtualizaron 3 máquinas utilizando VirtualBox [12], disponibles para su descarga en [1]. Se diseño la topología de Red de la Figura 3.

1. SERVIDOR

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

   a) www.amazon.com
   b) www.bing.com
   c) login.yahoo.com
   d) www.world-flags.com [13]

   Cada sitio se brindó utilizando Apache 2.2 [19]. En HTTP plano, en HTTPS utilizando mod_ssl [2] con un certificado SSL autofirmado con Ubuntu [25, Sección 4], y para brindar SPDY, se instaló mod_spdy [8]. con el propósito de poder realizar un análisis de los paquetes que viajan luego de finalizado el experimento se realizó la siguiente modificación en la configuración del SSL, que es el archivo *"/etc/apache2/mods-enabled/ssl.conf"* [27].

   ```
   # SSL Cipher Suite:
   # List the ciphers that the client is
   permitted to negotiate.
   # See the mod_ssl documentation for
   a complete list.
   # enable only secure ciphers:
   #SSLCipherSuite
   HIGH:MEDIUM:!ADH:!MD5
   SSLCipherSuite DES-CBC3-SHA
   ```

2. PROXY

   Utilizando la herramienta Dummynet [20] que viene instalada en la distribución de FreeBSD, se utilizaron diferentes comandos [21] para filtrar el tráfico en la red[4] y simular diferentes entornos. Se habilitó la Dummynet modificando el archivo *"/etc/rc.conf"* con las siguientes líneas

   ```
   firewall_enable="YES"
   firewall_type="OPEN"
   gateway_enable="YES"
   ```

   Se configuró el siguiente flag:

   ```
   sysctl net.inet.ip.forwarding=1
   ```

   Y por último, para que la Dummynet inicie junto con el Kernel del BSD se agregó la siguiente línea en el archivo *"/bot/loader.conf"*

   ```
   dummynet_load="YES"
   ```

3. CLIENTE

   Se instaló NodeJS [9] y se descargó el software *chrome-har-capturer* [15] de GitHub [22]. Se utilizó el navegador *Chromium* Versión 30 que, a travéz de su API de depuración remota [16], permite al *chrome-har-capturer* interactuar con dicho navegador para poder navegar un sitio particular y obtener un archivo *.har*[5], con los resultados de la interacción del navegador con el sitio. También, fue utilizada la herramienta *TSHark*[11] para poder capturar los paquetes

[2] ???

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

[4] Ancho de Banda y Retardo.

[5] Archivo con notación JSON [7] que contiene la traza del Navegador Web con el sitio

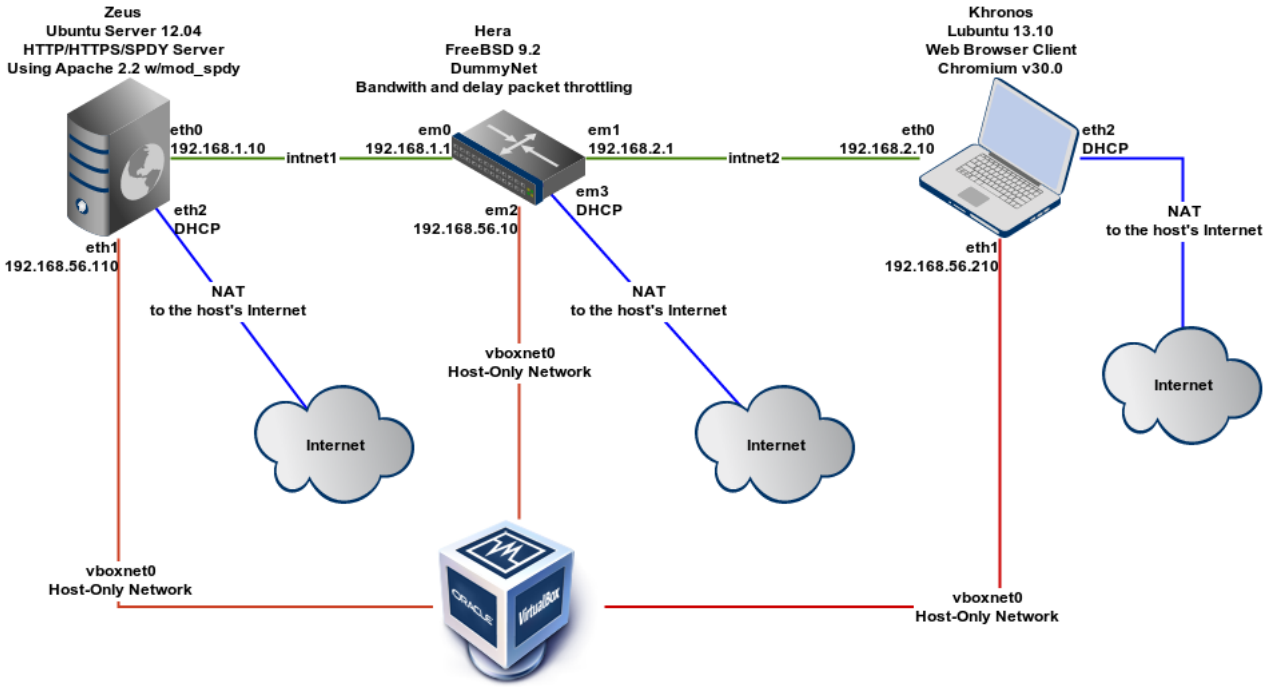

Figura 3: Diagrama de Red del Entorno Virtual del Experimento 1

que viajaron en la red durante el experimento.

4.2. Metodología

Con la idea de comparar los métodos HTTP, HTTPS y SPDY en diferentes ambientes simulados, 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	

En el Proxy, se creó una *tubería*[6] de la siguiente manera

```
ipfw add 1000 pipe 1 ip from any to any;
```

Y los valores se iban configurando automáticamente en mediante el siguiente comando:

```
ipfw pipe 1 config bw 1000Kbp/s delay
100ms;
```

[6]objeto intermediario donde se simulan diferentes entornos

Se combinaron todos los Anchos de Banda con todos los Retrasos para cada uno de los métodos por página[7].

En el Servidor se iba activando o desactivando *mod_spdy* según era requerido.

El algoritmo[8] fué el siguiente:

Algorithm 4.1: EXPERIMENTO1()

```
for ancho de banda ∈ anchos de bandas
 do
  for retardo ∈ retardos
   do
    configurar valores en el proxy
    for metodo ∈ (http, https, spdy)
     do
      if metodo == spdy
       then activar spdy en el servidor
       else desactivar spdy en el servidor
      for sitio ∈ sitios
       do
        iniciar chrome
        iniciar captura con tshark
        ejecutar chrome − har − capturer
        cerrar chrome
        cerrar tshark
```

Se repitió el experimento 7 veces y se prome-

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

[8]Disponible en [24, exp1.sh]

diaron los resultados.

4.3. Resultados

...

5. Experimento 2

5.1. Preparación

Se configuró un Cliente con las mismas configuraciones que las del Experimento 1 (ver sección 4). Se seleccionaron los siguientes sitios, basados en el artículo [17], con el agregado de 2 sitios más con diferente contenido para la prueba:

1. www.facebook.com
2. www.google.com
3. www.youtube.com
4. www.blogger.com
5. www.twitter.com
6. www.wordpress.com
7. www.imgur.com
8. www.youm7.com
9. consigueregalos.blogspot.com
10. oprojetopedal.wordpress.com

5.2. Metodología

Se modificó el algoritmo utilizado en el experimento anterior quedando éste de la siguiente manera:

Algorithm 5.1: EXPERIMENTO2()

for $metodo \in (http, https, spdy)$ **do**
  for $sitio \in sitios$ **do**
    $iniciar\ chrome$
    $iniciar\ captura\ con\ tshark$
    $ejecutar\ chrome - har - capturer$
    $cerrar\ chrome$
    $cerrar\ tshark$

Luego, se dejó el experimento corriendo durante 7 días, ejecutándose el mismo en diferentes horas:

1. 00:00
2. 04:00
3. 08:00
4. 12:00
5. 16:00
6. 20:00

5.3. Resultados

...

6. Conclusiones y Trabajos Relacionados

...

A futuro una de las pruebas a realizar sería, utilizando la suite de prueba del Experimento 1, ejecutar los tests pero con otros servidores que soporten SPDY, tales como *Jetty Web Server*[9], *Python implementation of a SPDY server*[10], *Ruby SPDY*[11] o *node.js SPDY*[12].

Agregar Selenium.

Referencias

[1] `http://www.marcelofernandez.info/files/`.

[2] Apache module mod_ssl. `http://httpd.apache.org/docs/2.2/mod/mod_ssl.html`.

[3] Freebsd. `http://www.freebsd.org/es/`.

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

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

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

[7] Introducing json. `http://www.json.org/`.

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

[9] nodejs. `http://nodejs.org/`.

[9] http://wiki.eclipse.org/Jetty/Feature/SPDY
[10] http://github.com/mnot/nbhttp/tree/spdy
[11] https://github.com/igrigorik/spdy
[12] https://github.com/indutny/node-spdy

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

[11] Tshark. http://www.wireshark.org/docs/man-pages/tshark.html.

[12] Virtual box. https://www.virtualbox.org/.

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

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

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

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

[17] Yehia Elkhatib, Gareth Tyson, and Michael Welzl. The effect of network and infrastructural variables on spdy's performance. 29 de Julio 2013.

[18] Tammy Everts. The average web page has grown 151 % in just three years. http://www.webperformancetoday.com/2013/11/26/web-page-growth-151-percent/. 26 de Noviembre 2013.

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

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

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

[22] GitHub. Github build software better, together. https://github.com/.

[23] Lubuntu. lubuntu, lightweight, fast, easier. http://www.lubuntu.net/.

[24] Pablo Maximiliano Lulic. spdy-tests. https://github.com/maxisoad/spdy-tests.

[25] mdsteele@google.com. Getting started with mod_spdy. https://code.google.com/p/mod-spdy/wiki/GettingStarted.

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

[27] Chris Strom. Ssl that can be sniffed by wireshark. http://japhr.blogspot.com.ar/2011/05/ssl-that-can-be-sniffed-by-wireshark.html.

[28] Ubuntu. Ubuntu server - for scale-out computing. http://www.ubuntu.com/server.

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

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