

Midiendo la Performance de SPDY

Pablo Maximiliano Lulic

17 de diciembre de 2013

Resumen

Actualmente, la Web se sustenta con el estándar HTTP. Este protocolo tuvo su última versión en el año 1999. Las 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 ambientes 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 [5]. 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 [4], 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% [19] (ver figura 1).

El crecimiento se produce con velocidad [33] y hay otras cuestiones relacionadas al tiempo de carga de una página, no solo el ancho de banda

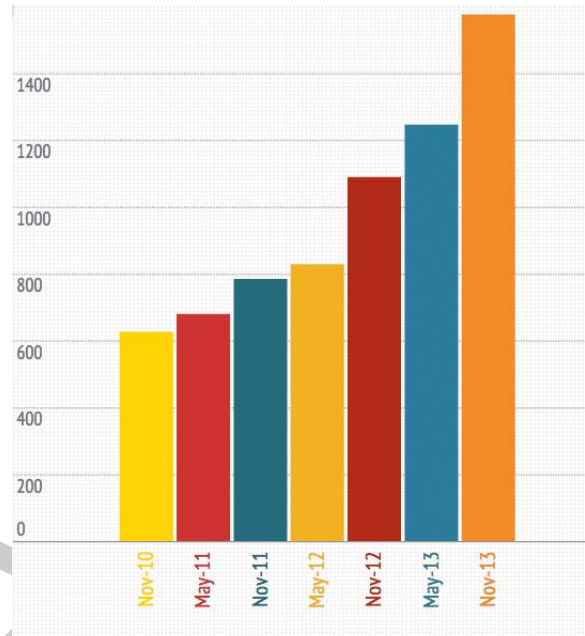


Figura 1: Crecimiento del tamaño de los sitios promedio, extraído de [19]

[15], sino por ejemplo, el RTT¹, como se puede ver en la figura 2

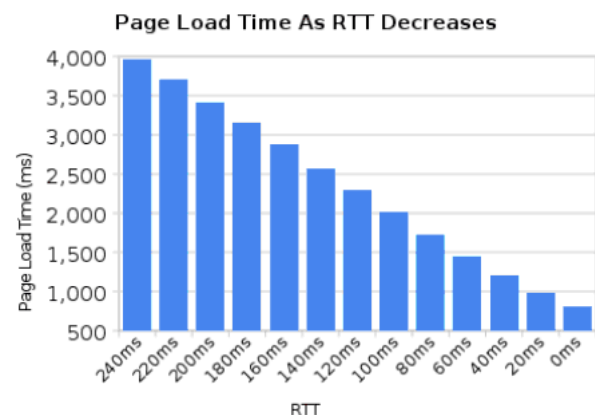


Figura 2: Tiempo de carga de una página mientras se varía el RTT, extraído de [15]

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

3. SPDY

SPDY[34] es un protocolo de la capa de aplicación [30] que funciona sobre SSL [9], permite la transmisión de Streams² sobre una conexión normal de TCP, que es el Protocolo de Control de Transmisión de la capa de Transporte [30]). A continuación se comentarán las características del Protocolo (extraídas de [34]):

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 [3].

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 [13], disponibles para su descarga en [20]. Se diseñó la topología de Red de la Figura 3.

1. SERVIDOR

Se descargaron los sitios³ 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 [14]

²Flujo de Datos.

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

Cada sitio se brindó utilizando Apache 2.2 [21]. En HTTP plano, en HTTPS utilizando mod_ssl [1] con un certificado SSL auto-firmado con Ubuntu [27, Sección 4], y para brindar SPDY, se instaló mod_spdy [7]. 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"` [31].

```
# 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 [22] que viene instalada en la distribución de FreeBSD, se utilizaron diferentes comandos [23] para filtrar el tráfico en la red⁴ 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 [8] y se descargó el software *chrome-har-capturer* [16] de GitHub [24]. Se utilizó el navegador *Chromium* Versión 30 que, a través de su API de depuración remota [17], permite al *chrome-har-capturer* interactuar con dicho navegador

⁴Ancho de Banda y Retardo.

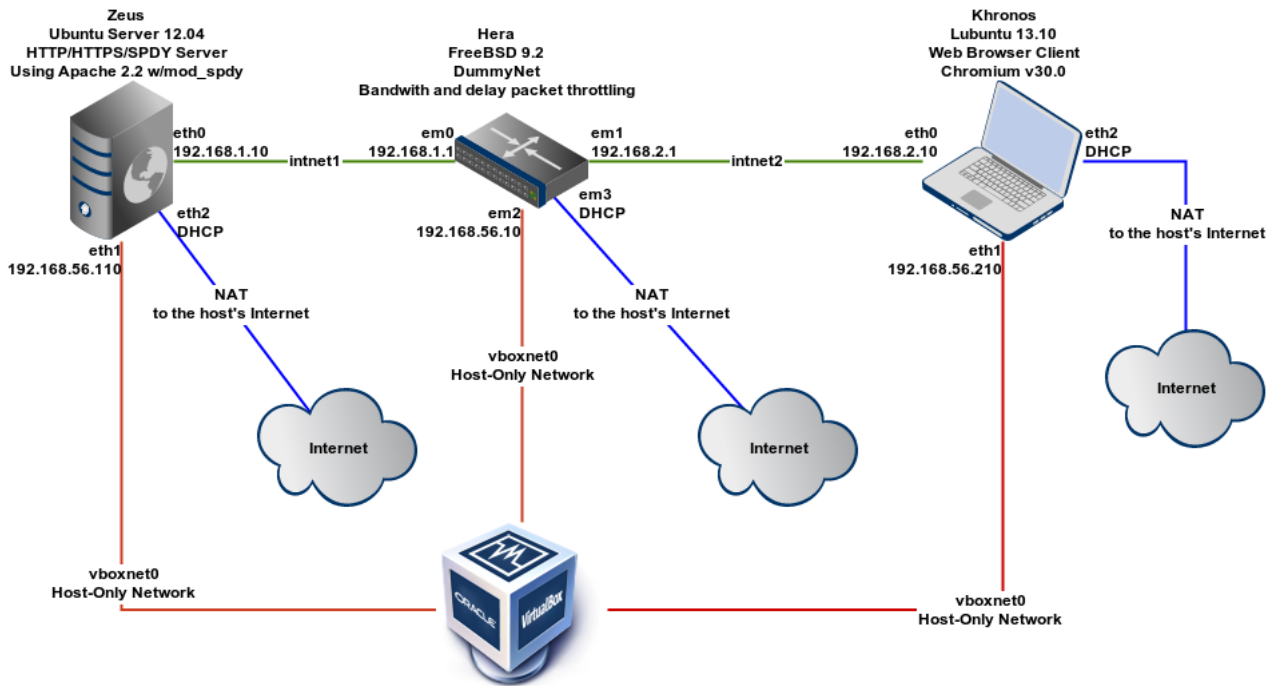


Figura 3: Diagrama de Red del Entorno Virtual del Experimento 1

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 *TS-hark*[12] para poder capturar los paquetes 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;
```

⁵Archivo con notación JSON [6] que contiene la traza del Navegador Web con el sitio

⁶objeto intermediario donde se simulan diferentes entornos

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

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

Se combinaron todos los Anchos de Banda con todos los Retrasos para cada uno de los métodos por página⁷.

En el Servidor se iba activando *mod_spdy* según era requerido, utilizando:

```
a2enmod spdy
```

O desactivando:

```
a2dismod spdy
```

En el cliente cuando se testeaba SPDY se desactivaba también la utilización desde el Browser con el flag⁸:

```
-use-spdy=off
```

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

⁸<http://peter.sh/experiments/chromium-command-line-switches/>

El algoritmo⁹ 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 para poder promediar los resultados. La información obtenida (archivos .har y .pcap) se procesó para obtener los tiempos basados en el .har [29], On-Load¹⁰ y OnContentLoaded¹¹ y el basado en la captura de *Tshark*, ToW¹². Estos resultados se almacenaron en una base de datos *SQLite* [11] para su posterior análisis.

4.3. Resultados

En cuanto a las medidas seleccionadas para el experimento, se observó que, el valor del *onContentLoaded* presenta una diferencia notable de las otras 2 medidas (*onLoad* y *ToW*) cuando el sitio tiene mucho contenido (por ejemplo en los sitios de amazon y worldflags) (Ver Figura 4). En cambio cuando el sitio tiene escaso contenido (el caso de yahoo y bing), las medidas se encuentran mas cercanas en sus valores (Ver Figura 5). De todas maneras, el tiempo *onLoad*, es el que se encuentra más cerca del tiempo que percibe el usuario cuando el navegador que está utilizando termina de renderizar el sitio en cuestión.

Hay un caso particular en el que la utilización de SPDY mejora notablemente los tiempos de carga de los sitios, y es en el caso de

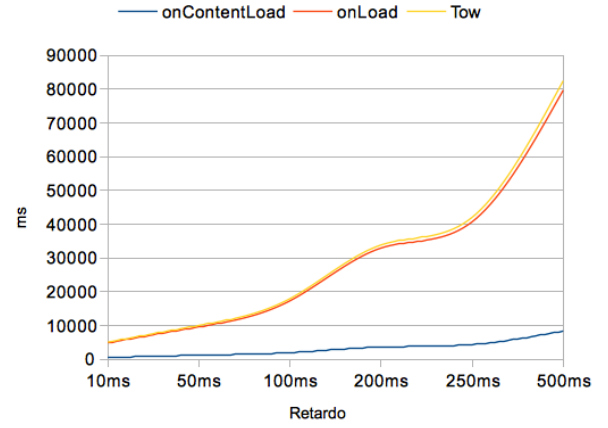


Figura 4: www.worldflags.com - HTTPS - 1024Kbps

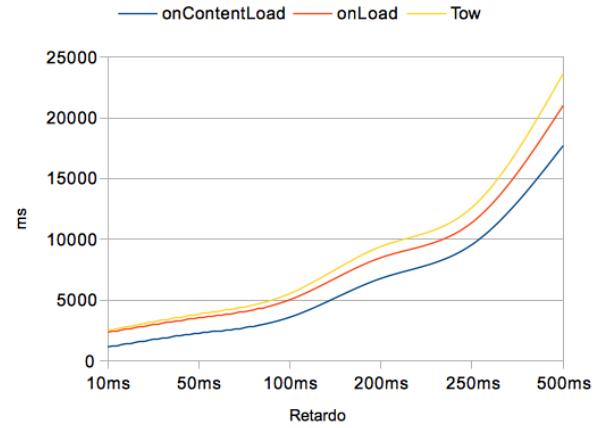


Figura 5: www.bing.com - HTTPS - 1024Kbps

www.worldflags.com, como se puede ver en la Figura 6 y 7.

Este sitio está tomado de [28] y se puede ver online en [14]. Es el caso emblemático para el protocolo, ya que reúne las características para que el funcionamiento de SPDY sea el óptimo. Es un sitio sencillo con 196 recursos (imágenes) todas almacenadas en el mismo lugar, esto aprovecha al máximo la utilización de los flujos de datos del protocolo en contraposición con la cantidad de conexiones que se deben realizar (utilizando HTTP o textschttps) con el servidor para traer tantos recursos.

Se observó el mismo comportamiento que plantea *Mike Belshe* en [15]. Esto se puede ver en la Figura 8 en donde al ir aumentando el Ancho de Banda pero con el mismo retardo de 500ms, el tiempo de carga del sitio va formando una constante.

El rendimiento de los protocolos cuando la ve-

⁹Disponible en [26, exp1.sh]

¹⁰Tiempo en el que el sitio se carga.

¹¹Tiempo en el que el contenido del sitio se carga.

¹²Tiempo en el cable, medido desde el primer paquete que se envía al servidor hasta el último.

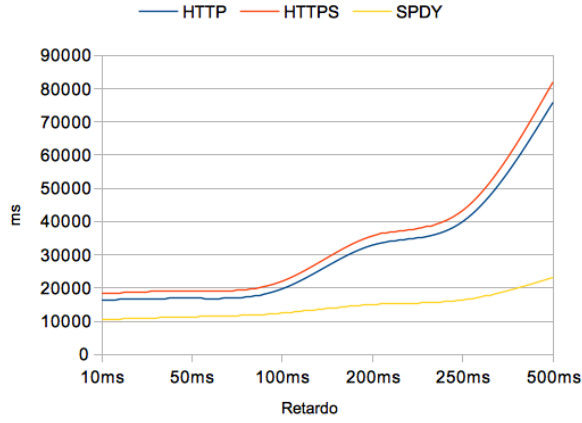


Figura 6: www.worldflags.com - onLoad - 256Kbps

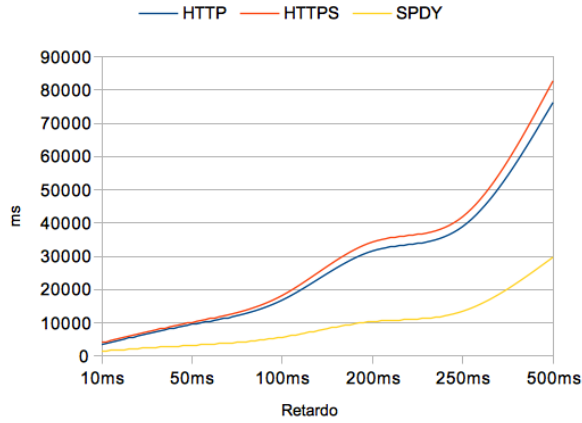


Figura 7: www.worldflags.com - Tow - 1024Kbps

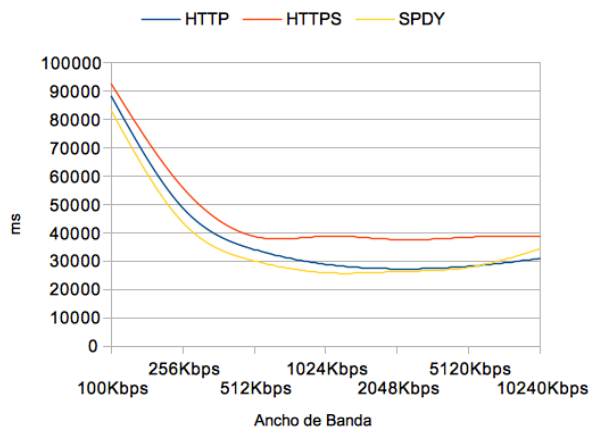


Figura 8: login.yahoo.com - onLoad - 500ms

locidad es alta es similar, en cambio cuando son sometidos a velocidades bajas, SPDY mejora su performance frente a los demás, tal como se ve en la Figura 9

5. Experimento 2

5.1. Preparación

Se configuró un Cliente con las mismas características que las del Experimento 1 (ver sección 4). Se seleccionaron los siguientes sitios, basados en el artículo [18], con el agregado de 2 sitios más con contenido diferente a la página principal de 2 de los sitios testeados (blogspot-blogger y wordpress):

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. consigueregamos.blogspot.com
10. oprojektopedal.wordpress.com

5.2. Metodología

Se quitó todo lo relacionado a ancho de banda y retardo del algoritmo utilizado en el experimento anterior quedando éste de la siguiente manera:

Algorithm 5.1: EXPERIMENTO2()

```

for metodo ∈ (http, https, spdy)
do
  for sitio ∈ 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

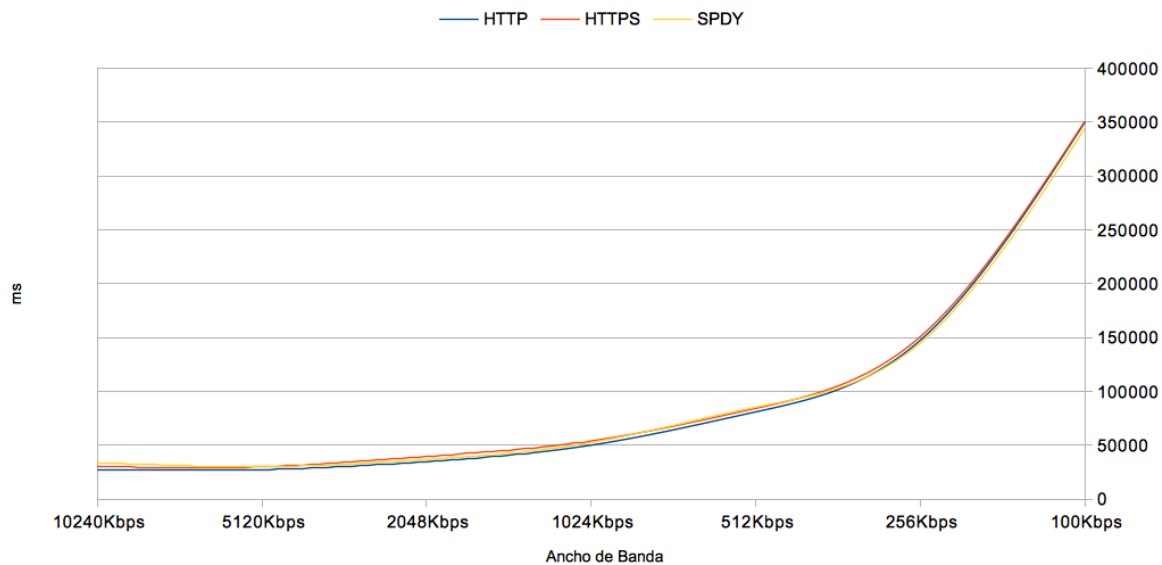


Figura 9: www.amazon - onLoad - 250ms

4. 12:00
5. 16:00
6. 20:00

Finalizada la semana, se recolectaron los datos de la misma manera que en el Experimento 1 (ver 4.2).

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*¹³, *Python implementation of a SPDY server*¹⁴, *Ruby SPDY*¹⁵ o *node.js SPDY*¹⁶. Para poder medir la performance de *spdy* en diferentes implementaciones del protocolo.

Otra herramienta interesante para realizar mediciones es Selenium [10], con esta herramienta se pueden automatizar diferentes interacciones con los sitios utilizando un navegador web.

¹³<http://wiki.eclipse.org/Jetty/Feature/SPDY>

¹⁴<http://github.com/mnot/nbhttp/tree/spdy>

¹⁵<https://github.com/igrigorik/spdy>

¹⁶<https://github.com/indutny/node-spdy>

Si bien estos tests concluyen que SPDY mejora la performance de la carga de un sitio, las características del protocolo se desempeñan al máximo cuando el usuario se encuentra navegando el sitio. Con esta herramienta se podría medir como se comporta SPDY en una utilización cuasi real.

Referencias

- [1] Apache module mod_ssl. http://httpd.apache.org/docs/2.2/mod/mod_ssl.html.
- [2] FreeBSD. <http://www.freebsd.org/es/>.
- [3] Header field definitions. <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>. part of Hypertext Transfer Protocol – HTTP/1.1 - RFC 2616.
- [4] Http archive. <http://httparchive.org>.
- [5] Hypertext transfer protocol – http/1.1. <http://www.ietf.org/rfc/rfc2616.txt>.
- [6] Introducing json. <http://www.json.org/>.
- [7] mod-spdy - apache spdy module. <http://code.google.com/p/mod-spdy/>.
- [8] nodejs. <http://nodejs.org/>.
- [9] The secure sockets layer (ssl) protocol version 3.0. <http://tools.ietf.org/html/rfc6101>.

- [10] selenium - browser automation framework. <https://code.google.com/p/selenium/>.
- [11] Sqlite. <http://www.sqlite.org/>.
- [12] Tshark. <http://www.wireshark.org/docs/man-pages/tshark.html>.
- [13] Virtual box. <https://www.virtualbox.org/>.
- [14] World flags mod_spdy demo. <https://www.modspdy.com/world-flags/>.
- [15] Mike Belshe. More bandwidth doesn't matter (much). 4 de Agosto 2010.
- [16] Andrea Cardaci. Capture har files from a remote chrome instance. <https://github.com/cyrus-and/chrome-har-capturer>.
- [17] Google Developers. Remote debugging protocol. <https://developers.google.com/chrome-developer-tools/docs/debugger-protocol>.
- [18] Yehia Elkhatib, Gareth Tyson, and Michael Welzl. The effect of network and infrastructural variables on spdy's performance. 29 de Julio 2013.
- [19] 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.
- [20] Marcelo Fernandez. <http://www.marcelofernandez.info/files/>.
- [21] The Apache Software Foundation. Apache. <http://www.apache.org>.
- [22] FreeBSD. Dummynet. <http://www.freebsd.org/cgi/man.cgi?query=dummynet>.
- [23] FreeBSD. Ipfw. <http://www.freebsd.org/cgi/man.cgi?query=ipfw&sektion=8&apropos=0&manpath=FreeBSD+9.2-RELEASE>.
- [24] GitHub. Github build software better, together. <https://github.com/>.
- [25] Lubuntu. lubuntu, lightweight, fast, easier. <http://www.lubuntu.net/>.
- [26] Pablo Maximiliano Lulic. spdy-tests. <https://github.com/maxisoald/spdy-tests>.
- [27] mdsteele@google.com. Getting started with mod_spdy. <https://code.google.com/p/mod-spdy/wiki/GettingStarted>.
- [28] mod_spdy. mod_spdy - news and updates for the apache spdy module. <https://www.modspdy.com/>.
- [29] Jan Odvarko odvarko@gmail.com. Har 1.2 spec. <http://www.softwareishard.com/blog/har-12-spec/>.
- [30] W. Richard Stevens. *TCP/IP Illustrated, Volume 1*. Addison-Wesley Professional, 1994. Section 1.2 Layering.
- [31] Chris Strom. Ssl that can be sniffed by wireshark. <http://japhr.blogspot.com.ar/2011/05/ssl-that-can-be-sniffed-by-wireshark.html>.
- [32] Ubuntu. Ubuntu server - for scale-out computing. <http://www.ubuntu.com/server>.
- [33] website optimization. Average web page size triples since 2008. <http://www.websiteoptimization.com/speed/tweak/average-web-page/>.
- [34] website optimization. Spdy: An experimental protocol for a faster web. <http://www.chromium.org/spdy/spdy-whitepaper>.